Symantec C++ Programming for Macintosh®

2nd Edition

Neil Rhodes & Julie McKeehan
Symantec C++ Programming for the Macintosh Second Edition
Symantec C++ Programming for the Macintosh
2nd Edition

Neil Rhodes
Julie McKeehan

Revised by Tom Chavez

SAMS PUBLISHING
Neil Rhodes is an expert programmer and popular lecturer at MacWorld, MADA, and Apple Worldwide Developer's conferences. He is a founder of Calliope Enterprises, which creates innovative tools for Macintosh development and provides consulting services to developers.

Julie McKeehan is a Macintosh software developer and technical writer, as well as principal of Calliope Enterprises. She brings her years of experience directing product development and writing software manuals to this book.

Tom Chavez is a product manager with Taligent, Inc., the joint venture of IBM, Apple, and Hewlett Packard. He was formerly with Apple Computer for over four years, most recently as product manager for MacApp. Tom lives in California and enjoys traveling to and scuba diving in exotic locations around the world.
Credits

Publisher
Rick Ranucci

Managing Editor
Kelly D. Dobbs

Developmental Editor
Sunthar Visuvalingam

Production Editor
Howard Jones

Copy Editor
Jo Anna Arnott

Book Designer
Kevin Spear

Cover Designer
Jay Corpus

Imprint Manager
Scott Cook

Product Marketing Manager
Debra Kempker

Production Analysts
Dennis Clay Hager
Mary Beth Wakefield

Indexer
Jeanne Clark

Production Team
Gary Adair
Brad Chinn
Kim Cofer
Meshell Dinn
Mark Enochs
Stephanie Gregory
Angela P. Judy
Jenny Kucera
Stephanie McComb
Shelly Palma
Beth Rago
Bobbi Satterfield
Marc Shecter
Kris Simmons
Greg Simsic
Carol Stamile
Robert Wolf
Acknowledgments

This book is the product of many people's labors besides our own and they deserve recognition for their contributions. Our thanks to our editors at Brady for all their work on this book under a tight deadline. Gratitude as well to Carole McClendon, our agent at Waterside Productions for her many efforts on our behalf.

We owe a special debt of gratitude to Dave Wilson for his willingness to contribute programming time to this book. Dave is not only a fine teacher but a good friend as well. As the original impulse behind Neil's getting into the programmer training business, he can't be thanked enough.

Thanks to all of our students, but most particularly to Neil's students at Apple Developer University. No teacher could ask for a better, brighter bunch to learn from.

In the more general realm, we would also like to thank Apple Computer for producing the PowerBooks on which this entire book was written. We can hardly wait to see what they will cook up next. Thanks also to Joel West for getting us into the Macintosh software business and for being the impetus behind our striking out on our own.

Last, but not least, our heartfelt gratitude goes to Loring and Barbie Fiske-Phillips. It is only because of their generosity and unceasing labors on our behalf that this book was finished on time. No one could ask for better friends.

Neil Rhodes

Julie McKeehan

April 1994
Dedication

To my mother, Mary, who taught me to always take a book wherever I go. Books have provided a lifetime of reward, as has she.

Neil Rhodes

To my parents, Gloria and Richard McKeehan, who taught me to finish what I start, even if it's harder than I thought. And, to my best friend, Barbie, who mothered my children, fed my dogs, and did a thousand things without hesitation to ensure I would have a life to come back to.

Julie McKeehan
### Contents at a Glance

<table>
<thead>
<tr>
<th>Introduction</th>
<th>xxvii</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Part 1 The Symantec C++ Environment</strong></td>
<td>1</td>
</tr>
<tr>
<td>1 Creating a Project</td>
<td>3</td>
</tr>
<tr>
<td>2 A Multiple-File Project</td>
<td>21</td>
</tr>
<tr>
<td>3 Interpreting Error Messages</td>
<td>49</td>
</tr>
<tr>
<td>4 Debugging a Program</td>
<td>59</td>
</tr>
<tr>
<td>5 Using Resource Utilities</td>
<td>83</td>
</tr>
<tr>
<td>6 Building an Application</td>
<td>95</td>
</tr>
<tr>
<td>7 Using Precompiled Headers</td>
<td>105</td>
</tr>
<tr>
<td><strong>Part 2 The Elements of C++</strong></td>
<td>119</td>
</tr>
<tr>
<td>8 C++: An Enhanced C</td>
<td>121</td>
</tr>
<tr>
<td>9 An Introduction to Object-Oriented Design</td>
<td>167</td>
</tr>
<tr>
<td>10 Getting Started with Objects</td>
<td>185</td>
</tr>
<tr>
<td>11 A Closer Look at Objects</td>
<td>213</td>
</tr>
<tr>
<td>12 Writing Effective C++ Code</td>
<td>235</td>
</tr>
<tr>
<td>13 The THINK Class Browser</td>
<td>249</td>
</tr>
<tr>
<td>14 The C++ Options Panels</td>
<td>261</td>
</tr>
<tr>
<td><strong>Part 3 The THINK ClassLibrary and Other Frameworks</strong></td>
<td>309</td>
</tr>
<tr>
<td>15 Using a Framework</td>
<td>311</td>
</tr>
<tr>
<td>16 RandomRectangle: Building a Rectangle Display Program</td>
<td>323</td>
</tr>
<tr>
<td>17 TextTyper: Building a Text-Editing Program</td>
<td>339</td>
</tr>
<tr>
<td>18 Exception Handling and Templates: PICTPeeker</td>
<td>357</td>
</tr>
<tr>
<td>19 Tick-Tock: Building a Clock Application</td>
<td>383</td>
</tr>
<tr>
<td>20 C++ Calc: Building a Calculator</td>
<td>405</td>
</tr>
<tr>
<td>21 Using Visual Architect: AddressBook</td>
<td>433</td>
</tr>
<tr>
<td>A Compatibility and Porting</td>
<td>511</td>
</tr>
<tr>
<td>B SourceServer and ToolServer</td>
<td>519</td>
</tr>
<tr>
<td>C TCL Reference</td>
<td>541</td>
</tr>
<tr>
<td>D Complete Source Listings</td>
<td>571</td>
</tr>
<tr>
<td>E Annotated Bibliography</td>
<td>717</td>
</tr>
<tr>
<td>F Disk Contents</td>
<td>723</td>
</tr>
<tr>
<td>Index</td>
<td>733</td>
</tr>
</tbody>
</table>
Contents

Part 1  The Symantec C++ Environment .............................................. 1
  1  Creating a Project ................................................................. 3
  The Symantec C++ Folders ......................................................... 4
  Installing Symantec C++ .......................................................... 7
  Creating a New Project ............................................................. 7
  Modifying a Source File ........................................................... 10
  Running the Program ............................................................... 11
  Errors in a Project ........................................................................ 13
    Compile Errors ........................................................................ 13
    Link Errors ............................................................................... 14
  Avoiding Dialogs ......................................................................... 16
    Avoiding the Project Update Dialog ...................................... 16
    Avoiding the Save Changes Dialog ........................................ 17
  Project-Building Shortcuts ......................................................... 18
  Summary .................................................................................... 19

  2  A Multiple-File Project ............................................................ 21
  Adding Files to a Project ........................................................... 22
  Adding a Single Source File to the Project ............................. 25
  Adding Libraries or Multiple Files to the Project ................... 26
  The Add Files Dialog ............................................................... 27
  Source Files in the Project Window ........................................... 29
  Adding Include Files ................................................................. 31
  Compiling a Single File ............................................................ 32
  Checking for Syntax Errors ........................................................ 33
  Removing Objects ....................................................................... 35
  Searching through Multiple Files ............................................. 36
    One-at-a-Time Searches .......................................................... 38
    Batch Searches ........................................................................ 39
  Segmenting Your Program ......................................................... 39
  Using Both C and C++ ................................................................. 41
  File Extensions .......................................................................... 43
  Summary ................................................................................... 47

  3  Interpreting Error Messages .................................................... 49
  An Error-Ridden Sample Program ............................................. 50
  The Compile Errors of an Error-Ridden Program ....................... 52
  Link Errors ............................................................................... 53
  Choosing the Number of Errors To Report ............................... 55
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Getting Rid of Warnings</td>
<td>56</td>
</tr>
<tr>
<td>Establishing a Philosophy about Warnings</td>
<td>56</td>
</tr>
<tr>
<td>The Three Rules of Warning Elimination</td>
<td>57</td>
</tr>
<tr>
<td>Summary</td>
<td>58</td>
</tr>
<tr>
<td>4 Debugging a Program</td>
<td>59</td>
</tr>
<tr>
<td>Turning On the Debugger</td>
<td>60</td>
</tr>
<tr>
<td>Starting Programs with the Debugger Activated</td>
<td>61</td>
</tr>
<tr>
<td>Command Options</td>
<td>62</td>
</tr>
<tr>
<td>The Step Command</td>
<td>63</td>
</tr>
<tr>
<td>The Trace Command</td>
<td>65</td>
</tr>
<tr>
<td>The Out Command</td>
<td>67</td>
</tr>
<tr>
<td>The In Command</td>
<td>68</td>
</tr>
<tr>
<td>The Go Command</td>
<td>69</td>
</tr>
<tr>
<td>The Stop Command</td>
<td>69</td>
</tr>
<tr>
<td>Putting a Command in Automatic Mode</td>
<td>69</td>
</tr>
<tr>
<td>Showing the Call Chain</td>
<td>70</td>
</tr>
<tr>
<td>Examining Functions in the Source Window</td>
<td>71</td>
</tr>
<tr>
<td>Using Breakpoints</td>
<td>72</td>
</tr>
<tr>
<td>Examining and Setting the Values of Variables</td>
<td>73</td>
</tr>
<tr>
<td>Using Debugger Statements in Programs</td>
<td>76</td>
</tr>
<tr>
<td>Calling Functions from the THINK Debugger</td>
<td>77</td>
</tr>
<tr>
<td>Low-Level Debuggers</td>
<td>80</td>
</tr>
<tr>
<td>Using Two Screens</td>
<td>82</td>
</tr>
<tr>
<td>Summary</td>
<td>82</td>
</tr>
<tr>
<td>5 Using Resource Utilities</td>
<td>83</td>
</tr>
<tr>
<td>Writing a Program That Uses Resources</td>
<td>84</td>
</tr>
<tr>
<td>Creating the Project</td>
<td>84</td>
</tr>
<tr>
<td>What the Program Does</td>
<td>87</td>
</tr>
<tr>
<td>Including Resources in the Program</td>
<td>87</td>
</tr>
<tr>
<td>Creating Resources with ResEdit</td>
<td>87</td>
</tr>
<tr>
<td>Adding Resources Directly</td>
<td>89</td>
</tr>
<tr>
<td>Creating Resources with THINK Rez</td>
<td>90</td>
</tr>
<tr>
<td>Comparing THINK Rez with ResEdit</td>
<td>92</td>
</tr>
<tr>
<td>THINK Rez Advantages</td>
<td>92</td>
</tr>
<tr>
<td>ResEdit Advantages</td>
<td>93</td>
</tr>
<tr>
<td>Summary</td>
<td>93</td>
</tr>
<tr>
<td>Exercise</td>
<td>93</td>
</tr>
</tbody>
</table>
6 Building an Application ................................................................. 95
  Setting Build Options ................................................................. 96
    Creator .................................................................................. 97
    Program Type ......................................................................... 97
    Partition Size ..................................................................... 98
    The SIZE Flags .................................................................. 98
    Far CODE and Far DATA ..................................................... 100
    Separate STRS .................................................................. 101
  Building the Application ............................................................ 101
    Smart Link ........................................................................ 102
  Summary ................................................................................... 103

7 Using Precompiled Headers .......................................................... 105
  Using Precompiled Headers ....................................................... 106
  Standard Precompiled Header Files .......................................... 107
  Effects of Compiling with
    Precompiled Headers .......................................................... 107
  Using Precompiled Headers in
    Individual Files .................................................................. 110
  Creating Precompiled Headers ............................................... 111
  Changing Header Files ........................................................... 111
  Creating a Precompiled Header File ........................................... 113
    Customizing MacHeaders++ or TCL .................................... 116
  Summary ................................................................................... 117

Part 2 The Elements of C++ ............................................................. 119

8 C++: An Enhanced C ................................................................. 121
  Overview ................................................................................ 122
  Standard Input/Output ............................................................ 122
    Eliminating Errors in I/O Operations .................................... 122
    Streams Enable User Extension ......................................... 125
    More Examples of Streams ............................................... 125
  The New Form of Comments .................................................. 126
  Declarations Mixed with Executable Statements ....................... 128
  Typedefing and Structs ............................................................ 131
  Default Arguments ................................................................. 132
  Error Checking ...................................................................... 133
    Warnings ........................................................................ 133
    Compile-Time Type Checking .......................................... 134
    Type-Safe Linkage .............................................................. 134
  Overloaded Functions ............................................................. 135
  Memory Allocation/Deallocation ............................................ 137
  Eliminating #define ............................................................... 139
Const Variables .............................................................. 139
Inline Functions .............................................................. 141
Small Functions and Inlining ........................................... 147
Functions that Cannot Be Inlined ................................. 148
Inlining Avoids #define Problems ............................... 149
Precedence Problems ...................................................... 151
Conclusions about Const Variables and Inline Functions .......................................................... 151
References ............................................................................. 152
How References Work ..................................................... 153
References and the Const Keyword ............................... 154
Some Rules of Thumb for Reference Use ...................... 155
Other Uses of References .................................................. 158
Summary ............................................................................. 159
Exercises ............................................................................. 161
9 An Introduction to Object-Oriented Design ...................... 167
Object-Oriented Programming ........................................ 168
Benefits of Object-Oriented Programming ..................... 168
The Genesis of Object-Oriented Programming ............. 169
Modular Programming ...................................................... 169
Data Types ........................................................................... 170
Abstract Data Types .......................................................... 171
Objects ................................................................................. 173
Object-Oriented Design ..................................................... 179
Basic Rules of Object-Oriented Programming .............. 179
The Object-Oriented Design Process ......................... 182
Summary ............................................................................. 183
10 Getting Started with Objects .......................................... 185
Overview ............................................................................. 186
A Procedural Program ......................................................... 186
An Object-Oriented Program ............................................. 188
Comparing Procedural with
Object-Oriented Programs .............................................. 189
The Descendant of Structs: The C++ Class .................... 190
Member Functions and Data Members ......................... 191
The this Keyword .............................................................. 193
C++ Naming Conventions ................................................. 194
Adding Caching ................................................................. 196
Caching in a Procedural Program ................................. 196
Caching in an Object-Oriented Program .................... 197
Access Control ................................................................. 198
Inline Functions .............................................................. 203
TextTyper's Main Program ..................................................354
The Resources ........................................................................354
Summary ................................................................................355
18 Exception Handling and Templates: PICTPeeker ...................357
The PICTPeeker Program ......................................................357
Compiling and Running the Program ...................................358
Inspecting the Program .........................................................358
   The CAboutBoxApplication Class ..................................361
   The CPictureApp Class ..................................................363
   The TDeleteOnFailure Class ..........................................366
   Templates ........................................................................366
   The THandleFailure Class ..............................................367
   The TDisposHandleOnFailure Class ..................................368
   The TCL_AUTO_DESTRUCT_OBJECT Class .......................369
   The THandleLocker Class ...............................................369
   The TWithFileOpen Class ...............................................370
   The CPictureDoc Class ...................................................371
   The CCutCopyPastePicture Class .....................................374
   The CPictureClipboardTask Class .....................................376
   Constructor ....................................................................377
   Doing the Command ......................................................377
   Redoing the Command ...................................................378
   Undoing the Command ...................................................378
   Disposing of Data ..........................................................379
PICTPeeker's Main Program ..................................................380
The Resources ........................................................................380
Summary ................................................................................381
19 Tick-Tock: Building a Clock Application ...............................383
The Tick-Tock Program .........................................................383
Compiling the Program .........................................................384
Running the Program ............................................................384
Inspecting the Program ........................................................385
   The CClockApp Class .....................................................385
   The CQuitOnCloseWindow Class ....................................387
   The CSaveRestoreSizeWindow Class ................................388
   The CDawdleBureaucratChore Class .................................392
   The CClockPane Class ...................................................393
Tick-Tock's Main Program .....................................................402
The Resources ........................................................................402
Summary ................................................................................403
Exercises ................................................................................403
### C++ Calc: Building a Calculator

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>The C++ Calc Program</td>
<td>406</td>
</tr>
<tr>
<td>Compiling the Program</td>
<td>406</td>
</tr>
<tr>
<td>Running the Program</td>
<td>406</td>
</tr>
<tr>
<td>Inspecting the Program</td>
<td>408</td>
</tr>
<tr>
<td>The TCalculator Class</td>
<td>409</td>
</tr>
<tr>
<td>The CCalculatorApp Class</td>
<td>412</td>
</tr>
<tr>
<td>The CCalculatorPane Class</td>
<td>413</td>
</tr>
<tr>
<td>Member Function Descriptions</td>
<td>414</td>
</tr>
<tr>
<td>The CalculatorPane Constructor</td>
<td>415</td>
</tr>
<tr>
<td>The CreateCommandNumber Function</td>
<td>416</td>
</tr>
<tr>
<td>The UpdateMenus Function</td>
<td>417</td>
</tr>
<tr>
<td>The DoKeyDown Function</td>
<td>421</td>
</tr>
<tr>
<td>The Draw Function</td>
<td>423</td>
</tr>
<tr>
<td>The UpdateDisplay Function</td>
<td>424</td>
</tr>
<tr>
<td>The EnterCurrentNumber Function</td>
<td>424</td>
</tr>
<tr>
<td>The CFloatText Class</td>
<td>424</td>
</tr>
<tr>
<td>Member Function Descriptions</td>
<td>425</td>
</tr>
<tr>
<td>The CFloatText Constructor</td>
<td>425</td>
</tr>
<tr>
<td>The SetMaxDisplayWidth Function</td>
<td>426</td>
</tr>
<tr>
<td>The GetValue and SetValue Member Functions</td>
<td>426</td>
</tr>
<tr>
<td>The GetTextString Function</td>
<td>427</td>
</tr>
<tr>
<td>The CButtonWithFontAndSize Class</td>
<td>428</td>
</tr>
<tr>
<td>The C++ Calc Main Program</td>
<td>429</td>
</tr>
<tr>
<td>The Resources</td>
<td>430</td>
</tr>
<tr>
<td>Summary</td>
<td>430</td>
</tr>
<tr>
<td>Exercises</td>
<td>431</td>
</tr>
</tbody>
</table>

### Using Visual Architect: AddressBook

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>The DatabaseTester Program</td>
<td>434</td>
</tr>
<tr>
<td>Compiling DatabaseTester</td>
<td>434</td>
</tr>
<tr>
<td>Running DatabaseTester</td>
<td>434</td>
</tr>
<tr>
<td>Inspecting DatabaseTester</td>
<td>435</td>
</tr>
<tr>
<td>The MCollectible Class</td>
<td>437</td>
</tr>
<tr>
<td>The TNameAddressPhone Class</td>
<td>438</td>
</tr>
<tr>
<td>The TEntry Class</td>
<td>441</td>
</tr>
<tr>
<td>The TNode Class</td>
<td>441</td>
</tr>
<tr>
<td>The TDatabase Class</td>
<td>445</td>
</tr>
<tr>
<td>The Tlterator Class</td>
<td>448</td>
</tr>
<tr>
<td>The DatabaseTester Main Program</td>
<td>452</td>
</tr>
<tr>
<td>The AddressBook Program</td>
<td>453</td>
</tr>
<tr>
<td>Compiling AddressBook</td>
<td>454</td>
</tr>
</tbody>
</table>

---

Symantec C++ for the Macintosh, 2nd Edition
CPanorama ................................................................. 559
CPicture .................................................................. 560
CPane .................................................................... 560
CStream .................................................................. 561
CTask ...................................................................... 561
CView ...................................................................... 562
CWindow .................................................................. 563
Global Variables ....................................................... 565
  gDecorator .......................................................... 565
  gBartender ......................................................... 565
  gClipboard ........................................................ 565
  gSignature ........................................................ 565
  gSystem ............................................................. 565
Global Routines ........................................................ 566
  ConcatPStrings .................................................. 566
  CopyPString ...................................................... 566
  Failure .................................................................. 566
  FailNIL ................................................................ 566
  FailOSErr ........................................................... 566
  FailMemError ..................................................... 566
  LongToQDRect .................................................... 567
  NewHandleCanFail ............................................. 567
  PositionDialog .................................................... 567
  ResizeHandleCanFail ......................................... 567
  SetAllocation ...................................................... 567
Macros .................................................................... 568
  TCLForgetObject ............................................... 568
  Exception Handling ............................................. 568
  TCL_AUTO_DESTRUCT_OBJECT ............................ 568
  Constructor/Destructor Macros ............................. 569
  Creating Objects by Name ..................................... 569
D Complete Source Listings ........................................ 571
  Chapter 2, MultipleFiles ....................................... 571
  Chapter 3, Errors ............................................... 574
  Chapter 4, BinaryTree ......................................... 576
  Chapter 5, Alphabet ............................................ 578
  Chapter 7, Project One ......................................... 581
  Chapter 7, Project Two ......................................... 581
  Chapter 16, RandomRectangle ............................... 582
  Chapter 17, TextTyper .......................................... 592
  Chapter 18, PicturePeeker ...................................... 602
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>TickTock</td>
<td>621</td>
</tr>
<tr>
<td>20</td>
<td>C++ Calc</td>
<td>644</td>
</tr>
<tr>
<td>21</td>
<td>Database Tester</td>
<td>672</td>
</tr>
<tr>
<td></td>
<td>AddressBook</td>
<td>683</td>
</tr>
<tr>
<td>B</td>
<td>PowersOfTwo</td>
<td>715</td>
</tr>
<tr>
<td>E</td>
<td>Annotated Bibliography</td>
<td>717</td>
</tr>
<tr>
<td>C</td>
<td>Reference Material</td>
<td>717</td>
</tr>
<tr>
<td></td>
<td>C++ Reference Material</td>
<td>718</td>
</tr>
<tr>
<td></td>
<td>Object-Oriented Programming</td>
<td>719</td>
</tr>
<tr>
<td></td>
<td>Object-Oriented Design</td>
<td>719</td>
</tr>
<tr>
<td></td>
<td>Programming</td>
<td>720</td>
</tr>
<tr>
<td></td>
<td>Macintosh</td>
<td>722</td>
</tr>
<tr>
<td>F</td>
<td>Disk Contents</td>
<td>723</td>
</tr>
<tr>
<td>2</td>
<td>MultipleFiles</td>
<td>723</td>
</tr>
<tr>
<td>3</td>
<td>Errors</td>
<td>724</td>
</tr>
<tr>
<td>4</td>
<td>Binary Tree</td>
<td>724</td>
</tr>
<tr>
<td>5</td>
<td>Alphabet</td>
<td>724</td>
</tr>
<tr>
<td>7</td>
<td>Precompiled Headers</td>
<td>725</td>
</tr>
<tr>
<td>16</td>
<td>RandomRectangle</td>
<td>725</td>
</tr>
<tr>
<td>17</td>
<td>TextTyper</td>
<td>726</td>
</tr>
<tr>
<td>18</td>
<td>PICTPeeker</td>
<td>726</td>
</tr>
<tr>
<td>19</td>
<td>Tick-Tock</td>
<td>727</td>
</tr>
<tr>
<td>20</td>
<td>C++ Calc</td>
<td>728</td>
</tr>
<tr>
<td>21</td>
<td>Database Tester</td>
<td>729</td>
</tr>
<tr>
<td></td>
<td>AddressBook</td>
<td>729</td>
</tr>
<tr>
<td>B</td>
<td>PowersOfTwo</td>
<td>731</td>
</tr>
<tr>
<td></td>
<td>Application</td>
<td>731</td>
</tr>
</tbody>
</table>
Preface to the Second Edition

This edition of Symantec C++ Programming for the Macintosh has been thoroughly revised to incorporate the enhancements of the latest version of Symantec C++ (7.0). Section Three of this book has changed the most. All of the sample applications in this section have been updated to reflect changes from TCL 1.13 to TCL 2.0. In addition, the application in Chapter 21 now uses Visual Architect, Symantec C++'s 7.0 graphical program generator.

We would like to thank Tom Chavez, who did a great deal of the work on this revision. Our further thanks to Jim Trudeau, who, after finding several errors in the first printing, kindly reported back to us.
Introduction

"C++ is the programming language of the future."

This is, no doubt, a remark you have heard repeated so often your ears are numb. In the past, if you were a Macintosh programmer and wanted to ride this C++ wave into the future you quickly encountered a tsunami—MPW. Macintosh Programmer's Workshop (MPW) from Apple may have supported C++, but it also came with a learning curve the shape of Mt. Everest and compile times the speed of cold molasses.

Faced with this situation, many programmers decided to live without C++. Symantec, the creators of the most popular development environment for the Macintosh—Think C—have thoughtfully provided you with a better alternative, Symantec C++ for Macintosh. This programming environment is a sleek, brilliantly fast vehicle for C++. It’s just the right mode for just the right language. Why are we so sure that C++ is the right language, and that you should learn it? The following are some pretty compelling reasons:

▼ You want a job as a programmer.
▼ You already have a programming job, but want a raise.
▼ You don’t want to be replaced by a C++ programmer.
▼ You want to develop state-of-the-art software.

Of course, knowing C++ well enough to write good programs is not enough to bring about any of the above (except maybe getting a job; there’s a high demand for C++ programmers) but it is guaranteed to help. One reason why this should be as obvious to you as it is to most commercial Macintosh software developers is Pink, the new object-oriented operating system from Taligent (the joint Apple and IBM venture).
Pink, if it turns out as expected, will metamorphose the PC and Macintosh world. Imagine being one of the application developers for this new system. The catch? You will need to write your applications in C++, and Symantec C++ for Macintosh is the right place to start.

**Symantec C++**

Should you learn C++? Only you can decide that. But, if you do, here are some reasons why Symantec C++ is the right compiler to use.

▼ **The Project Manager.** Symantec C++ for Macintosh is not just a new C++ compiler in an old framework. The Project Manager (the development environment) is decoupled from the compiler. This multilingual Project Manager allows multiple translators within the same project; part of your project can be written in C, and part in C++.

▼ **Source Code Control.** The environment also supports source code control. Using Apple's SourceServer (shipping with Symantec C++), you can have the same capabilities MPW users possess for managing project source code. One-person projects were always a snap in Think C; but now you can run a large project with a multi-programmer team just as easily.

▼ **A Resource Compiler.** Symantec C++ for Macintosh also comes with Think Rez, a compiler for Apple's Rez language. Rez is a language that's particularly useful for describing resources in a textual format. You can therefore create your resources visually in ResEdit, textually with Rez, or use a combination of the two.

▼ **On-Line Reference System.** Also from Symantec, Think Reference is an on-line reference program that you can access from the project manager with a few clicks of the mouse. Do you have a routine for which you need the parameters? When you double-click, Think Reference provides a detailed listing of that routine and its parameters. This makes coding in unfamiliar territory far easier.

▼ **Visual Architect.** With this tool, you can graphically create most of the visual components of your application. You draw out your views, design your windows, and create your menus. Then Visual Architect adds to your project source code files based on these interface elements.
This is by no means a complete list of features. It should be just enough to convince you that Symantec C++ for Macintosh is the right way to learn C++ on the Macintosh.

**What This Book Has To Offer**

Should you read this book? If you want to write C++ programs using Symantec C++ for Macintosh, the answer is “Yes.” After you read it, you should be able to do the following:

- Use Symantec C++ for Macintosh to develop applications. This is a practical book about C++, not a theoretical one. The goal is to turn you into a C++ programmer, not a C++ theoretician.
- Design your programs as a collection of objects to make them easier to write, modify, and maintain.
- Use C++ as a more effective C, and as a complete object-oriented language.
- Begin to write C++ code like a native (efficient and idiomatic C++).
- Use application frameworks (primarily the Think Class Library). Application frameworks are more wonderful than freshly baked bread. They shorten your development time and imbue your application with more features than it would normally have.

**Who Is This Book For?**

You may be a committed Think C programmer who wants to learn C++ because you are convinced it’s the language of the future. If you are, you can take comfort from Symantec’s approach with C++: it’s quick and it’s intuitive. You also get an easy transition path for your existing projects because parts of your project can be in C++ while other parts can remain in C.

You may be a MPW C++ programmer who is tired of plodding compiles and long linking in MPW. If you are, you will be pleasantly surprised. You can make a change to your code, and then recompile it and run it in a matter of seconds; contrast this with the minutes of MPW. Further,
having slogged through the long learning curve of MPW, you will be glad to know that Symantec C++ for Macintosh has a tightly integrated, intuitive interface and only takes hours to learn, rather than the days or months of MPW.

You may be a programmer interested in learning C++ because you have heard that object-oriented programming is a better way to write and maintain code. It is, and Symantec C++ for Macintosh is a great environment in which to make the transition to object-oriented programming. You may have been waiting for the right C++ for the Macintosh. One that didn't require extraordinary amounts of time or money. With Symantec C++ for Macintosh, you can start writing and debugging C++ code within an hour or two.

If any of these descriptions applies to you, then this book is worth your time.

What Should You Already Know?

You need to know how to program in C. It helps to have at least some knowledge of how to program on the Macintosh (the more the better). You don't need to know C++ and you don't need to know how to use Think C. If you do know either of these, all the better.

What Does This Book Cover?

In the first part you walk through the Think environment. You create some very small C programs that you compile with both the C and C++ compilers. The second part introduces you to C++, the language. You also put the C++ you are learning into immediate use by writing and testing some programs in it. In the third part, you learn to use the Think Class Library (TCL) to write several complete programs. You also learn a little about other application frameworks. Finally, there are some appendixes that discuss code compatibility, porting from MPW, and some of the advanced features of the Think environment.

First you learn the environment, then the language, then you use both with the Think Class Library to write a bunch of different programs.
How Should You Read This Book?

Everyone should read this introduction, so that you know where every topic is discussed. There is more flexibility with the rest of the book. The following are some suggestions to help you focus your reading:

▼ **If you already know how to use the Think environment.** You don’t need to read all of Part 1. Instead, read the introduction to find out where the new features of this Think C/C++ release are covered. You can then skip to that material.

▼ **If you already know C++.** You don’t need to read all of Part 2. Start with Appendix A to learn how the Symantec C++ compiler differs from other C++ compilers. Chapters 13 and 14 from Part 2 contain the rest of the information you need to know about using Think C++.

▼ **If you already know how to use the Think Class Library.** You may not need to read all of Part 3. Chapter 15 offers an introduction to frameworks: what they are and how to use them. There is also detailed discussion of the TCL. Chapters 16 through 21 are worth reading if you want to see some C++ sample programs that use the TCL. If other application frameworks besides the TCL interest you, then read Chapter 15.

▼ **If you have never used C++ or the Symantec environment before.** The book was written with you in mind. Read it straight through.

What this Book Does Not Cover

This is not a comprehensive guide to either C++ or the Think environment. C++ is too rich a language to cover completely in a book of this size. The Think environment, likewise, is only introduced here; if you want to learn more about the environment, the program documentation is a good place to start.
Symantec C++ for Macintosh has a long and varied history. The environment sprang from THINK C, whereas the compiler started life as Zortech C++. The figure on the next page shows what that lineage looks like.

Back in the early years of the Macintosh, there was a small company with an idea of pairing a C compiler with an exciting development environment. The company was THINK Technologies, and the idea became THINK C, the most popular development environment for the Macintosh. Unlike other environments, THINK C had a Macintosh look and feel—it threw out the command line for a graphical interface. THINK quickly captured the market for third-party C compilers and was acquired by Symantec after a few years.

Zortech developed a successful C++ compiler for DOS that was one of the first true C++ compilers—it did not use C as an intermediate language, but generated object code directly.
Although Zortech ported this compiler to MPW, it never caught on very well. When Symantec acquired Zortech, they had the raw ingredients for what finally turned into Symantec C++ for Macintosh: THINK C and Zortech C++.

Only time can tell whether Symantec C++ for Macintosh will rival THINK C in popularity. But the ingredients are all there: Symantec C++ is fast, functions graphically, and contains a C++ compiler that points the way to the future as well as its C compiler that maintains the present.

This section shows you how to use Symantec C++ within the THINK environment. You learn everything from creating that first source file to building a final stand-alone application. You will do it faster, and have more fun, than you ever could have in MPW.

If you already are familiar with THINK C and want to read about the new features of the THINK Project Manager, read Chapters 2, 3, 4, and 5. Chapter 2 discusses mixing C and C++ in one project. Chapter 3 deals with errors and warnings. Chapter 4 discusses the THINK Debugger, which has some new features. Chapter 5 introduces the THINK Class Inspector. Chapter 6 discusses THINK Rez.
There are few traditions as sacrosanct within the computer programming world as that of starting a C or C++ tutorial with the Hello World program. So in deference to tradition, this chapter shows you how to build Hello World using Symantec C++. The information presented here should give you enough of an introduction to the Symantec C++/THINK environment to be able to navigate through a simple project. In this chapter, you do the following:

- Set up project folders (as well as briefly see what is included with Symantec C++).
- Use the THINK Project Manager and the Symantec C++ compiler by creating a new project (Hello World) and adding source files to it.
- Compile and link the Hello World project.
- Learn some of the option settings and what certain error messages mean.
The Symantec C++ Folders

There are a number of folders and files that come with Symantec C++. Figure 1.1 shows you what the topmost folder looks like after installing the software.

Figure 1.1. The main folder.

Table 1.1 provides brief descriptions of the contents of each folder.

Table 1.1. Contents of the Topmost Folder after Installing Symantec C++

<table>
<thead>
<tr>
<th>Folder</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demos</td>
<td>Sample C++ programs</td>
</tr>
<tr>
<td>Online Documentation</td>
<td>The THINK Reference program with documentation for standard libraries and C++ iostreams</td>
</tr>
<tr>
<td>Release Notes</td>
<td>Latest ReadMe files and notes from Symantec C++</td>
</tr>
<tr>
<td>Scripting</td>
<td>Sample scripts to control the THINK Project Manager using AppleScript and UserLand Frontier</td>
</tr>
<tr>
<td>Symantec C++ for Macintosh</td>
<td>The Project Manager, compilers, header files, and libraries; the contents of this folder are shown in figure 1.2</td>
</tr>
<tr>
<td>TCL Demos</td>
<td>Sample THINK Class Library (TCL) programs</td>
</tr>
<tr>
<td>Utilities</td>
<td>Useful programs, including ResEdit</td>
</tr>
</tbody>
</table>
When you open the Symantec C++ for Macintosh folder, you see that it contains the folders shown in figure 1.2.

![Figure 1.2. The contents of the Symantec C++ for Macintosh folder.](image)

This is obviously the folder of interest. Its contents are detailed in table 1.2.

**Table 1.2. Contents of the Symantec C++ for Macintosh Folder**

<table>
<thead>
<tr>
<th>Folder</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Project Models)</td>
<td>Templates for new projects you will create with the THINK Project Manager</td>
</tr>
<tr>
<td>Aliases</td>
<td>Aliases to the header files and libraries you want in a standard search path</td>
</tr>
<tr>
<td>(AppleScripts)</td>
<td>Sample AppleScript scripts that control the THINK Project Manager</td>
</tr>
<tr>
<td>Inspector Libraries</td>
<td>Libraries (with source code) for use with THINK Inspector</td>
</tr>
<tr>
<td>Mac #includes</td>
<td>Macintosh-specific headers for C, C++, and Rez</td>
</tr>
<tr>
<td>Folder</td>
<td>Contents</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Mac Libraries</td>
<td>Macintosh-specific libraries for C and C++</td>
</tr>
<tr>
<td>oops Libraries</td>
<td>Libraries (with source code) for C++ objects</td>
</tr>
<tr>
<td>Projects</td>
<td>Aliases to the projects you use the most often; these projects are added to the Switch to Project menu in the THINK Project Manager</td>
</tr>
<tr>
<td>Standard Libraries</td>
<td>Standard (not Macintosh-specific) libraries (including source code) and headers for C and C++</td>
</tr>
<tr>
<td>THINK Class Library</td>
<td>Headers and source files for the THINK Class Library (TCL)</td>
</tr>
<tr>
<td>THINK Debugger</td>
<td>The debugging application you use with Symantec C++; it launches automatically when you debug</td>
</tr>
<tr>
<td>THINK Inspector</td>
<td>The object inspector you use with the THINK Debugger to debug object based applications</td>
</tr>
<tr>
<td>THINK Project Manager</td>
<td>The application you use to access the Symantec C++ compiler; it is where you create, edit, compile, and build your applications</td>
</tr>
<tr>
<td>(Tools)</td>
<td>Contains SourceServer (see Appendix A for more information); you put aliases to THINK Reference in this folder so that the THINK Project Manager can launch it as necessary</td>
</tr>
<tr>
<td>Translators</td>
<td>The THINK C and Symantec C++ compilers as well as THINK Rez and other translators</td>
</tr>
<tr>
<td>Visual Architect</td>
<td>The tool for building applications from their user interface parts</td>
</tr>
<tr>
<td>Visual Architect Macro Files</td>
<td>Templates and files used by the Visual Architect to automatically generate TCL applications</td>
</tr>
</tbody>
</table>
Installing Symantec C++

To set up Symantec C++, follow the installation instructions that come with the product. There are some other things worth arranging as well:

1. Locate THINK Reference in the Online Documentation folder and create an alias to it.
2. Put the alias in the Tools folder.
3. Delete the word alias from the alias.
4. If you own the THINK Reference 2.0 product, make aliases for the documentation files in the THINK Reference folder and put them in the Online Documentation folder. This will let you look up all the information at the same time. You can also put aliases to Toolserver and third-party editors, such as BBedit, that support THINK project manager Apple Events and can be used as substitutes for the THINK editor.

Creating a New Project

The THINK Project Manager manages all aspects of the project from beginning to end. Everything—from creating source files to rolling the object files into an application—is done from within the Project Manager. The obvious first step, then, is learning to use the Project Manager.

Begin by launching the THINK Project Manager. Upon launching, the Project Manager displays the dialog box shown in figure 1.3.

![Figure 1.3. The THINK Project Manager initial dialog box.](image)
You create a new project by clicking the New button. The THINK Project Manager then displays a dialog box (see fig. 1.4) from which you select the type of project you want to create. Because this will be a C++ program, select the C++ Project. Make sure that the Create folder box is checked.

![New Project dialog box](image)

**Figure 1.4.** The New Project dialog box.

**TIP**

The THINK Project Manager makes creating new projects very easy when you use template projects. When you make a new project, you choose a template and the project manager makes copies of all of the files in the chosen project model folder, renaming the project file to match the name you enter. If you want all of your new projects to have certain characters, in other words, a certain copyright statement, modify the files in the appropriate project model folder and all of your new projects will inherit this change.

The next dialog box, shown in figure 1.5, asks you to name the project and pick the location for the project directory. Name the new project Hello World.

![Creating a new project](image)

**Figure 1.5.** Creating a new project.
IMPORTANT

Make sure that your project directories are not in the Symantec C++ folder or any of its subfolders. If you put project folders in there, the Project Manager has to search through all of your project folders every time it looks for standard include files. It also must look through all the standard header files to find your project header files.

Figure 1.6 shows a fairly standard arrangement that you might use for your Hello World project folder and other project folders as well.

![Figure 1.6. Location of your Hello World project folder.](image)

If you open up the folder that the THINK Project Manager just created for you, you will find two files in it, Hello World.pi (the project file) and main.cp (an initial source code file). See figure 1.7.
NOTE

The THINK Project Manager creates the project file from the project name you enter. It automatically appends .tt to the end of the project name to distinguish project files from source files.

Figure 1.7. The Hello World project folder.

After you have created the new project, the THINK Project Manager shows you the Project window and its accompanying menus (see fig. 1.8). The Project window also displays a list of all the files in the project; because you chose to create a C++ project, the THINK Project Manager automatically created a main.cp file and included two library files.

Figure 1.8. The Project window.

Modifying a Source File

Open the main.cp source file by selecting it in the project window and pressing Return, or simply double-click on it. You should see the following:

```
#include <stdio.h>
```

main()
{
    return 0;
}

Modify it so that it looks like the following:

```c
#include <stdio.h>

main()
{
    printf("Hello, world!\n");
    return 0;
}
```

After you are finished, save the file.

Running the Program

You run all programs in the THINK Project Manager the same way: Select Run from the Project menu (see fig. 1.9).

![Figure 1.9. Using Run to run your program.](image)

In this case, your program’s source files have not been compiled and linked. In such cases, the THINK Project Manager displays the dialog box shown in figure 1.10, asking whether you want to bring the project up to date before running it.
Figure 1.10. The dialog box shown when a program needs compiling.

Selecting Update (the default) tells the Project Manager to compile and link the program.

While compiling, the THINK Project Manager displays a progress window that it updates as it deals with each file in turn (see figs. 1.11, 1.12, and 1.13).

Figure 1.11. The Progress window showing progress on CPlus.

Figure 1.12. The Progress window showing progress on main.cp.

Figure 1.13. The Progress window showing progress on ANSI++.
After successfully compiling each file, the Project Manager runs the program (if it does not, read on about errors). The ANSI++ library sends standard output to a console window (see fig. 1.14).

![Figure 1.14](image.png)

**Figure 1.14.** Output from running the Hello.world program.

You can quit from the Hello.world program by either pressing return or selecting Quit from the File menu.

## Errors in a Project

You can get a number of errors when building your program. The compiler may report an error in your source file, or the linker may report an error when linking together all of the compiled code.

### Compile Errors

As the Symantec C++ compiler generates errors, it puts them in a Compile Errors window. To see this, create an error in main.cp by removing the semicolon from the `printf` statement:

```c
#include <stdio.h>

void main()
{
    printf("Hello, world!\n") /* semicolon gone */
    return 0;
}
```

Chapter 1: Creating a Project
Try to run the program, making sure to click Update when the Project Manager asks you about bringing the project up to date. The Compile Errors window appears, showing a list of errors (see fig. 1.15). The first line for each error tells where the error occurred, and the second line describes the error.

![Compile Errors Window](image)

**Figure 1.15.** The Compile Errors window.

To go to the offending line in the source file, select the error and press Return or double-click on it. This takes you to the line where the compiler thinks the error occurred (see fig. 1.16).

![Source File with Error](image)

**Figure 1.16.** Going to the line with the error.

As commonly happens with compilers, the selected line is the one immediately following the line with the actual error. Fix the error by adding the deleted semicolon and save the file.

**Link Errors**

The linker links together calls to functions with the code for those functions. If, therefore, you call a function whose code is not present in the project, the linker reports an error. To see this in action, open the ANSI++ Segment (by clicking on the arrow) and select ANSI++ in the Hello World.II Project window. Choose Remove from the Source menu (see fig. 1.17).
Figure 1.17. Using Remove to remove a file from the project.

Having assured yourself of failure, try to run the program (click Update to bring the project up to date). The linker displays an alert (see fig. 1.18) that tells you the link failed. You can click on the link alert to make it go away. The THINK Project Manager displays a Link Errors window that describes exactly why the link failed.

Figure 1.18. A link failure.

The error messages, `undefined: atexit`, `undefined: printf`, in the window indicate that the code for the `atexit` call and the `printf` call could not be found. Add ANSI++ back to your project using Add File or Add Window from the Source menu. Now, run your program again to get everything in working order.
If you get the `undefined: main` link error, this means that the file containing your `main` function is not in your project.

### Avoiding Dialogs

Using the THINK Project Manager preferences dialog, you can turn off confirmation dialogs you find unnecessary (or annoying). Select THINK Project Manager from the Options item in the Edit menu (see fig. 1.19).

![Options panel](image)

**Figure 1.19.** Setting options for the THINK Project Manager.

### Avoiding the Project Update Dialog

To turn the Project Update dialog off, go to the Preferences panel in the THINK Project Manager Options dialog. Uncheck Confirm project updates, as shown in figure 1.20. The THINK Project Manager now automatically brings your project up to date before running it.
Avoiding the Save Changes Dialog

Before your program is actually run, but after it is built, the THINK Project Manager asks whether you want to save changes to any modified source files (see fig. 1.21). The reasoning behind such an offer should be obvious: If your program crashes, forcing a reboot, you lose any changes to unsaved files.

If you want to forgo the pleasure of seeing this dialog every time you run your program, you can have the Project Manager automatically save your files. You do this in the Editor panel of the THINK Project Manager Options dialog. Use the panel pop-up to select the Editor. After you uncheck the Confirm saves check box, the Project Manager no longer alerts you before saves (see fig. 1.22).
Project-Building Shortcuts

The following are some shortcuts you can use in a project to help make life easier:

▼ When the Project window is frontmost, you can type the name of a file to select it (this works the same as the Finder).

▼ To open a file that is selected in the Project window, double-click on it or press Return or Enter.

▼ Option-clicking on the title bar of a source file brings up a pop-up menu of its include files. For example, Option-clicking anywhere on the Main.cp title bar shows you its two include files: size_t.h and stdio.h (see fig. 1.23). To open an include file, select it from this pop-up menu.

Figure 1.22. The Editor panel of the THINK Project Manager Options dialog.

Figure 1.23. Option-clicking on the title of a source file.
Option-clicking anywhere on the title bar of the Project window brings up a pop-up menu of the include files used by all the source files (see fig. 1.24). You can open any include file by selecting it.

![Figure 1.24. Option-clicking on the title of the Project window.](image)

Another way to open an include file is to select its name in a source file, and then choose Open Selection from the File menu (see fig. 1.25).

![Figure 1.25. Using Open Selection to open a header file.](image)

**Summary**

You should now feel fairly comfortable with the THINK Project Manager and its accompanying windows for viewing source files and listing errors. You also should know how to select certain options to lessen the number of interrupts during the running of your project. Next, you learn to use the THINK Project Manager to work with a larger project—one with multiple files and segmentation.
In a large, multiple-file project, there are a number of tasks you have to perform. The most obvious task—writing the source files—is only the beginning. Before you are through, you also need to know how to do the following:

- Add the new source files to an existing project.
- Add libraries to the project, including knowing which ones to use and where to find them.
- Use `#include` file syntax to take advantage of the THINK Project Manager's file-searching rules.
- Compile just one file.
- Check for syntax errors without having to do a costly compile.
- Slim down your projects when you need to send them to someone else or free up space on your drive.
- Search through files to find routines that you and time have forgotten.
Segment your project.

Use both the THINK C and Symantec C++ compilers with the same project.

As the size of your projects grow, you may eventually encounter each of these situations. This chapter explains when you might encounter each problem and how to handle it in a multiple-file project. This chapter does not, however, contain an exhaustive treatment of the Symantec C++ project environment or multiple-file issues. For more information, you should refer to the program documentation.

You cannot progress further, however, until you have created some source files and put them in a project.

Adding Files to a Project

Either open the MultipleFiles.pi project on the source file disk or create a new MultipleFiles project (choose Empty Project instead of C++ project in the new project dialog). The seven files that make up the project are: Main.cp, Reverse.h, Reverse.cp, Factorial.h, Factorial.cp, Average.h, and Average.cp. The code is provided below for your reference.

Here is the code for Main.cp:

```c
#include <stdio.h>
#include "Factorial.h"
#include "Reverse.h"
#include "Average.h"

void main()
{
    int i;
    char s[] = "Neil Rhodes";
    char palindrome[] = "Madam, I'm Adam"
    double values[] = {3.1, 4.1, 5.9, 15.6, 16.3, 15.7};

    for (i = 0; i < 14; i++)
        printf("Factorial(%i) = %ld\n", i, Factorial(i));

    printf("Reverse("%s") = ", s);
    Reverse(s);
    printf("\n\n", s);
```
printf("Reverse("!lss\") = ", palindrome);
Reverse(palindrome);
printf("\"%s\"\n", palindrome);

printf("Average() = %lf\n",
    Average(values, sizeof(values) / sizeof(values[0])));
}

The code for Reverse.h is as follows:
#pragma once
void Reverse(char *s);

The code for Reverse.cp is as follows:
#include "Reverse.h"
#include <string.h>

static void Swap(char *c1, char *c2)
{
    char tmp;
    tmp = *c1;
    *c1 = *c2;
    *c2 = tmp;
}

void Reverse(char *s)
{
    int length = strlen(s);
    int midPoint = length / 2;
    int i;
    for (i = 0; i < midPoint; i++)
        Swap(&s[i], &s[length - i - 1]);
}

Listed below is the code for Factorial.h:
#pragma once

unsigned long Factorial(unsigned long n);
The code for Factorial.cp is as follows:

```c
#include "Factorial.h"

unsigned long Factorial(unsigned long n)
{
    unsigned long result = 1;
    unsigned long i;
    for (i = 2; i <= n; i++)
        result *= i;
    return result;
}
```

Here is the code for Average.h:

```c
#pragma once

long double Average(double *values, int numValues);
```

Here is the code for Average.cp:

```c
#include "Average.h"
#include "SANE.h"

long double Average(double *values, int numValues)
{
    long double total = 0.0;
    int i;
    for (i = 0; i < numValues; i++)
        total += values[i];
    if (numValues > 0)
        return total / numValues;
    else
        return 0.0;
}
NOTE

In most development environments, C source files are identified with the suffix .c. There is less agreement within the community on the proper suffix for C++ source files. UNIX platforms commonly use .C because the UNIX file system differentiates between upper- and lowercase. DOS C++ compilers typically adopt .cpp (for C plus plus). On Macintosh, you commonly use .cp (for CPlus, the name of the MPW C++ compiler).

With Symantec C++, you can use any suffix you desire, although .cp and .cpp are most common. Later in this chapter modifying your file suffixes to something other than .cp or .cpp is discussed.

Adding a Single Source File to the Project

There are two ways to add files to the project:

▼ Use Add Window from the Source menu.

▼ Use Add Files from the Source menu.

The first method, Add Window, is somewhat misleading. The menu only displays Add Window when it cannot be used. For example, when the Project window is frontmost, the Add Window item is dimmed (see fig. 2.1).

Figure 2.1. The Source menu when the Project window is frontmost.
On the other hand, if your frontmost window is a file that can be added to the project (like the C++ file Main.cp), the Add Window menu item changes its name. The new name reflects the name of this frontmost window (see fig. 2.2).

Click on the Main.cp window to make it frontmost and then choose Add ‘Main.cp’ from the Source menu to add it to the project. The Project window is updated to reflect the newly added file, as shown in figure 2.3.

![Figure 2.2. The Source menu when Main.cp is frontmost.](image)

![Figure 2.3. The Project window after adding Main.cp.](image)

**Adding Libraries or Multiple Files to the Project**

You can add the source files to the project one-by-one but it is faster to add them together with the Add Files dialog. To do so, select Add Files from the Source menu (see fig. 2.4).
After Add Files is selected, it brings up a dialog. In this dialog, you can add one or more source or library files to the project (see fig. 2.5).

The buttons in the Add Files dialog perform the following actions:

- The Open/Add button moves the selected file from the top list to the bottom list (if a file in the top list is selected) or opens the selected folder (if a folder in the top list is selected).

- The Add All button moves all the files from the top list to the bottom list.
- The Remove button removes the selected file from the bottom list.
- The Done button quits the dialog and adds the files in the bottom list to the project.

Move into the Average folder and select Average.cp. Click Add (or double-click on Average.cp). The dialog should now look like the one in figure 2.6.

![Figure 2.6](image)

Figure 2.6. The Add Files dialog with Average.cp in the bottom list.

Now move into the Factorial and Reverse folders and add Factorial.cp and Reverse.cp respectively. When you are finished, the Add Files dialog should look like figure 2.7.

![Figure 2.7](image)

Figure 2.7. The Add Files dialog with the three new files.

Click on Done to add the three files and go back to the project.
Source Files in the Project Window

Your Multiple Files Project window should now have the Main.cp, Average.cp, Factorial.cp, and Reverse.cp files in it (see fig. 2.8).

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segment 2</td>
<td>0</td>
</tr>
<tr>
<td>Average.cp</td>
<td>0</td>
</tr>
<tr>
<td>Factorial.cp</td>
<td>0</td>
</tr>
<tr>
<td>Main.cp</td>
<td>0</td>
</tr>
<tr>
<td>Reverse.cp</td>
<td>0</td>
</tr>
<tr>
<td>Totals</td>
<td>582</td>
</tr>
</tbody>
</table>

**Figure 2.8.** The Project window after adding the new files.

If you try to run the project at this point, without adding the necessary libraries, you get the link error shown in figure 2.9.

**Figure 2.9.** Link error due to missing libraries.

You need to add the necessary libraries so that the program runs. The printf and strlen routines used in Main.cp and Reverse.cp are defined by ANSI C. They are found in the ANSI++ library. In Chapter 1, this library was included automatically as part of the C++ program template.

Add the ANSI++ library using the Add Files dialog, as shown in figure 2.10. You'll find this library file in the Standard Libraries folder.
Reviewing the source code reveals that it never directly calls `ULMULT`. Instead, these routines are called by the compiler to do unsigned long multiplication and long division. Because the 68000 does not have instructions to implement these operations, you need to get the subroutines that support them from a library. Which library are they in?

Gaining knowledge about library contents is not an overnight process. If you are not a library expert, here are some of the solutions to which you might resort:

- Read through the product documentation on each library and its contents (the speed-reader's solution).
- Use trial and error by trying each library in turn (the solution for programmers who disdain documentation).
- Ask the resident library expert or just memorize that the CPlusLib library contains the necessary routines (the pragmatic solution).

Now that you know what libraries fix the problem, add them to the project. Add CPlusLib to your project (also found in the Standard Libraries folder).

Try running the program now; you should get the output shown in figure 2.11.
### Figure 2.11. Output of the MultipleFiles program.

<table>
<thead>
<tr>
<th>Factorial(n)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factorial(0)</td>
<td>1</td>
</tr>
<tr>
<td>Factorial(1)</td>
<td>1</td>
</tr>
<tr>
<td>Factorial(2)</td>
<td>2</td>
</tr>
<tr>
<td>Factorial(3)</td>
<td>6</td>
</tr>
<tr>
<td>Factorial(4)</td>
<td>24</td>
</tr>
<tr>
<td>Factorial(5)</td>
<td>120</td>
</tr>
<tr>
<td>Factorial(6)</td>
<td>720</td>
</tr>
<tr>
<td>Factorial(7)</td>
<td>5040</td>
</tr>
<tr>
<td>Factorial(8)</td>
<td>40320</td>
</tr>
<tr>
<td>Factorial(9)</td>
<td>362880</td>
</tr>
<tr>
<td>Factorial(10)</td>
<td>3991680</td>
</tr>
<tr>
<td>Factorial(11)</td>
<td>3628800</td>
</tr>
<tr>
<td>Factorial(12)</td>
<td>479001600</td>
</tr>
<tr>
<td>Factorial(13)</td>
<td>39916800</td>
</tr>
</tbody>
</table>

Reverse("Hello Rhodes") = "sedohR lieH"
Reverse("Adam, I'm Adam") = "moda l'm, adAm"
Average() = 10.116667

---

## Adding Include Files

If you look again at the include statements in the source files, you see two variations in syntax. Symantec C++ (like most C and C++ compilers) differentiates between `#include` statements that use `<>` and those that use `"`.

If you use the `#include <File.h>` syntax in a source file, the compiler looks for `File.h` in the folder containing the THINK tree. That is, the compiler starts searching in the THINK Project Manager and looks through all of the subfolders there.

If you use the `#include "File.h"` syntax, the computer looks first in the folder that contains the file that has the `#include` preprocessor directive. If it doesn't find `File.h` there, the compiler searches for `File.h` in the project tree. It starts looking in the folder containing the current project and then looks through all subfolders. If it does not find `File.h` there, it looks through the THINK tree.

Because of this search order, follow the two rules below for faster compiles.

**Rule**

Use `#include <_>` for standard include files (like `stdio.h` or `quickdraw.h`). Otherwise, the compiler unnecessarily looks through your project tree for files it finds in the THINK tree.
The THINK Project Manager also searches when it looks for source files. This is a refinement that enables you to keep your project headers and your project sources in whatever folder structure you wish. (The file location must remain within the project tree, however.) Figure 2.12 shows a sample rearrangement of the files of the MultipleFiles project. Even if you use such a subfolder arrangement, the project still compiles and runs.

![Figure 2.12. Moving files into subfolders.](image)

### Compiling a Single File

When you have a large project, you may not want to build the whole program; instead, you may want to compile just one file. For instance, if you have not written all your source code, doing a build may produce link errors. The THINK project manager enables you to compile a single file by using the Compile command from the Source menu.

To see how this operates, open Average.cp and make it frontmost. Select Compile (see fig. 2.13) and watch what happens.

The compiler presents you with a progress dialog, as shown in figure 2.14.
Figure 2.13. Using Compile to compile one file.

This is the best way to compile single files in a project. It is not, however, the only shortcut you can take; there is an even briefer route.

Checking for Syntax Errors

As is often the case on a project, you compile only to find syntax errors. You may not be finished writing the code for a file, but may want to snag any bugs introduced so far. If this is the case, then you want to use the Check Syntax command. This choice enables you to run through the compilation process without generating any object code. It is quicker than a compile because there is no code to generate, but it still provides the same errors and warnings.

To try this command, open Average.cp and remove a semicolon. Now that you have doomed yourself to failure, choose Check Syntax from the Source menu (see fig. 2.15).
Figure 2.15. Running Check Syntax on a file.

This syntax check generates an error in the Compile Errors window, as shown in figure 2.16. Before going further, put the semicolon back where it belongs.

Figure 2.16. The results of Check Syntax.
Removing Objects

If you have a big project that you want to send to someone else or you are out of disk space, you have a problem on your hands. The solution is to remove the object files with the Remove Objects command.

You should do this whenever you need to reduce the size of your project file. To give you an idea of the change in size, look at the MultipleFiles project before and after removing objects. Before removing the objects, the file size was 75K. This goes down to a slim 14K after the objects are eliminated.

To give this a try, select Remove Objects from the Project menu, as shown in figure 2.17.

You get a confirmation dialog (see fig. 2.18) that indicates the next build has to start from scratch.

![Figure 2.17. Removing all objects from the project.](image)

![Figure 2.18. The Remove Objects confirmation dialog.](image)
Searching through Multiple Files

Another aspect of a large multiple-file project is that only those lucky few with eidetic memory can remember where every routine is located. The THINK Project Manager provides sophisticated search facilities for the rest of the population. As well as searching through the current window, the Project Manager searches through multiple files.

To see how the search facility works, select Find from the Search menu (see fig. 2.19).

This brings up a dialog that has a place to enter the search string, as well as various search options (see fig. 2.20).

To conduct a multiple-file search, check the Multi-File Search check box. This, in turn, brings up a dialog in which you select the files you want to search (see fig. 2.21).

![Figure 2.19. Using Find from the Search menu.](image)

![Figure 2.20. The Find dialog.](image)
Figure 2.21. The Multi-File Search dialog.

Click on a file in the list to checkmark it. All checkmarked files are searched. You also can use the buttons on the left as shortcuts:

- The Check All button checks all files.
- The Check Sources button checks only source files.
- The Check Includes button checks only include files.
- The Check None button unchecks all files.

For your purposes, you want to click Check All (or press command-A) to checkmark all the files (see fig. 2.22). Click OK to return to the Find dialog.

Figure 2.22. After pressing Check All.

When you click on the Multi-File Search check box, it brings up the Multi-File window. After someone plays with the option awhile, the same question is always asked: How do you turn the check box off if the Multi-File Search dialog always comes up? An easy answer is that you option-click on it. Alternatively you can click on the check box to turn it off, click on Check None in the Multi-File Search dialog and click OK. A shortcut for turning it on (and selecting all files) is to type command-A in the Find dialog.
You perhaps noticed the two check boxes under the Multi-File Search check box. When you select the Exclude <...> Files option, the search skips any files in the THINK tree and only searches through files in your project. The other option, Batch Search, has to do with the two ways the THINK Project Manager has of presenting the results of searches. Both of these ways are covered in the following sections.

**One-at-a-Time Searches**

If Batch Search in the Find dialog is not checked, the THINK Project Manager presents the matching lines one at a time. To try this, click on the Find button in the dialog (making sure you are searching for “include”). The Project Manager opens the first file in which it finds the string “include” (see fig. 2.23).

![Figure 2.23. The first occurrence of the string “include.”](image)

To go to the next occurrence of “include” in this file, use Find Again from the Search menu (see fig. 2.24). When there are no more occurrences in a file, Find Again beeps. To find the next file containing the string, use Find In Next File from the same menu (see fig. 2.24).

![Figure 2.24. The Search menu while a multiple-file search is in progress.](image)
**Batch Searches**

The other alternative is for the THINK Project Manager to output a list of matching lines in a window. To do this sort of search, check the Batch Search check box in the Find dialog. Figure 2.25 shows you the window that results from a batch search of "include."

![Search Results](image)

**Figure 2.25.** The Search Results window.

To go from a matching line shown in the window to the file containing that line, select the line in the window and press return (or just double-click on it). The THINK Project Manager opens the file at that line. You also can sequence through each of the lines by using the Go To Next Match and Go To Previous Match choices in the Search menu.

**Segmenting Your Program**

On the Macintosh, you break your code into code segments. These segments are loaded on demand and then unloaded with the Toolbox call UnloadSeg. The size of each of these segments needs to be less than 32K. The THINK Project Manager enforces this restriction at link time.

When you start a new THINK project, all of your code is in one segment. You create a new segment by doing the following:

1. Select the file you want to place in a new segment and hold the mouse button down on it.

2. Drag the file below the wavy line that follows the last filename.

Try that now with the Average.cp file. Create a new segment by dragging the file below the line under Reverse.cp (see fig. 2.26).

A new partition, named Segment 3, appears with Average.cp in it (see fig. 2.27).

A dotted line separates one segment from another. Notice also that within each segment the files are alphabetized. If you want to move a file from one segment to another, just drag it to the other segment. To create another new segment, drag a file below the last segment.
**Figure 2.26.** Moving a file from one segment to a new segment.

**Figure 2.27.** After creating a new segment.

A real problem arises in your project if you let a segment get too big. This impels the linker to notify you with an error message like the one in figure 2.28. To fix the problem, move some of the files from that segment to a different segment. To reduce the segment size correctly, you need to concentrate on the files with the largest size. The code size of each file is provided in the Project window (see fig. 2.27 above) to help you decide on the rearrangement.

**Figure 2.28.** Linker error if a code segment exceeds 32K.
Using Both C and C++

In Symantec C++, the THINK Project Manager enables you to mix both C and C++ in your project. There are some situations that might be ideal for this setup:

▪ You have a large C code base. Although C++ is mostly a superset of ANSI C, there are certain C language constructs that C++ does not accept (for instance, C++ requires function prototypes). While it is easy to write new C code that can be compiled with both C and C++, it can be tedious to update old C code. If, therefore, you have a large C code base that does not compile with C++, it may be easier to have a multiple-compiler project.

▪ You want to use in-line assembler. Symantec C++ does not support in-line assembler. Because THINK C does, any function that needs in-line assembler needs to be done in C.

If you are going to have a multiple-language project, there are two critical issues that affect the structure of your project:

▪ You need to make sure that the right compiler is used. File extension configuration is necessary because the THINK Project Manager chooses the compiler based on the file extension (or suffix). If you use the extension .c for your C files and .cp for your C++ files, the right compiler will be used.

▪ Calling between C and C++ requires some C++ code modification. You need to make some changes to your C++ code to be able to call a C routine or to enable C to call your C++ routine. For each call from C++, or vice versa, the routine declaration needs to be modified to signify to the C++ compiler that a cross-language call will occur.

For the C++ compiler, add the following line before the declaration:

    extern "C" {

And, add the following line after the declaration:

    }

Here is an example of the change:

    extern "C" {
    void MyCRoutine(const char *s);
    }
This special C++ syntax is usually within an #ifdef statement (the cplusplus symbol is automatically defined by Symantec C++). This way the header file containing the declaration can be shared between C and C++.

```c
#ifdef __cplusplus
extern "C" {
#endif

void MyCRoutine(const char *s);

#ifdef __cplusplus
}
#endif
```

Create a new duo-language project (ideal for midnight Powerbook Duo compiles). Name the project GoodnightMoon. Listed below are the sources for the program. Pay special attention to code that calls C from C++ and vice versa:

Here is the code for Main.cp:

```c
#include "CRoutine.h"

don't

void main()
{
    CRoutine("Goodnight, moon");
}
```

Listed below is the code for CRoutine.h:

```c
#pragma once

#ifdef __cplusplus
extern "C" {
#endif

void CRoutine(const char *s);

#ifdef __cplusplus
}
#endif
```

The code for C++Routine.h is as follows:

```c
#pragma once
```
Here is the code for CRoutine.c:

```c
#include <stdio.h>
#include "CRoutine.h"
#include "C++Routine.h"

void CRoutine(const char *s)
{
    printf("CRoutine: got %s\n", s);
    CPlusPlusRoutine(s);
}
```

The code for C++Routine.cp is as follows:

```c
#include <stdio.h>
#include "C++Routine.h"

void CPlusPlusRoutine(const char *s)
{
    printf("CPlusPlusRoutine: got %s\n", s);
}
```

### File Extensions

Symantec C++ for Macintosh also contains some other translators besides C and C++. A translator is responsible for taking a file from the project and translating it in a way appropriate for the final application.

The following is a list of the translators included with Symantec C++:
Symantec C++  This compiles C++ source code to 68000 object code.
THINK C  This compiles C source code to 68000 object code.
THINK Rez  This compiles Rez source code to resources.
Resource Copier  This copies a resource file to the resource fork of the program.
<<none>>  This does nothing—quite usefully. It is for a project file that might be used for documentation but still must reside in the project. With this file extension, the file can stay in the project without affecting the final program.
.o Converter  This converts an MPW .o object file to THINK Project Manager 68000 object code.

The THINK Project Manager uses these suffixes (or extensions) to determine which compiler to use. If you do not use the default extensions, as shown in table 2.1, you may need to change the settings.

<table>
<thead>
<tr>
<th>Extension</th>
<th>Translator</th>
</tr>
</thead>
<tbody>
<tr>
<td>.cp and .cpp</td>
<td>Symantec C++</td>
</tr>
<tr>
<td>.c and .asm</td>
<td>THINK C</td>
</tr>
<tr>
<td>.r</td>
<td>THINK Rez</td>
</tr>
<tr>
<td>.rsr</td>
<td>Resource Copier</td>
</tr>
<tr>
<td>.o</td>
<td>.o Converter</td>
</tr>
<tr>
<td>.note</td>
<td>&lt;&lt;none&gt;&gt;</td>
</tr>
</tbody>
</table>

To change extension settings or create your own extension types, select the THINK Project Manager from the Options menu item. From the Extensions panel of that dialog, you can see the default mappings and change them (see fig. 2.29).

Practice changing the extensions and the translator. Start by changing the translator for .c files from THINK C to Symantec C++. Here is how you do it:

1. Select .c from the extensions list.
2. Select Symantec C++ from the Translator menu.
3. Click Replace (see fig. 2.30).
Now, create a new extension to use with documentation files. (This is done by adding a "<<none>>" translator for .doc extensions.) Do the following:

1. In the Extensions panel, type .doc in the File Extension text box.
2. Select "<<none>>" from the translator pop-up and click Add (see fig. 2.31).

Figure 2.29. The Extensions panel of the THINK Project Manager Options dialog.

Figure 2.30. Making Symantec C++ the translator for .c files.
Figure 2.31. Making "<none>" the translator for .doc files.

You should now see a new extension for "<none>", as shown in figure 2.32.

Figure 2.32. "<none>" with a new file extension.
Summary

At this point, you should have a fairly good understanding of all of the following:

▼ Adding multiple files to a project.
▼ Figuring out which libraries are needed for a project.
▼ Writing #include file syntax in-line with THINK Project Manager searching rules.
▼ Compiling just one file or checking for syntax errors without doing a full build.
▼ Removing objects from projects to slim down their sizes.
▼ Searching through project files to find a forgotten routine.
▼ Segmenting your projects properly using the rules described.
▼ Creating a multiple-language project that uses both THINK C and Symantec C++ compilers.

If you are not completely comfortable with these skills, you can either review the related section of the chapter again or look at the Symantec C++ documentation. Now that you are a THINK Project Manager manipulator, it is time to deal with the pestilence that attacks any project in progress—bugs. The next chapter shows you how to deal with errors in your project.
Interpreting Error Messages

Like other compilers, the Symantec C++ compiler generates error messages whenever your code is not syntactically correct. Unlike some compilers, Symantec C++ also provides warning messages whenever your code is slightly sloppy. Because the compiler goes to the trouble of generating these warning messages, you should do something with them. While it is true that warnings need not be eliminated for a successful compile, they do point out possible problems in your code. This chapter, therefore, focuses on teaching you the following:

- Symantec C++'s method of presenting compile and link error messages.
- Understanding the error messages you get.
- Determining the number of error messages that the compiler reports.
Before you can learn to fix errors and warnings, however, you need a sample program that produces a large number of them.

**An Error-Ridden Sample Program**

Following is an example program that generates a number of errors and warnings. You can either create a new project named Errors.pi with the following source files, or open the project file on the source code disk. Your project should consist of five source files: Main.cp, Reverse.h, Average.h, Reverse.cp, and Average.cp. For reference, the source listings of these files with each error commented are listed below.

The code for Main.cp is as follows:

```c
#include <stdio.h>
#include "Reverse.h"
#include "Average.h"

void main()
{
    int    i;
    char   s[] = "Neil Rhodes";
    char   palindrome[] = "Madam, I'm Adam";
    double values[] = {3.1, 4.1, 5.9, 15.6, 16.3, 15.7};

    printf("Reverse("%s") = ", s);
    Reverse(s);
    printf("%s\n", s);

    printf("Reverse("%s") = ", palindrome);
    Reverse(palindrome);
    printf("%s\n", palindrome);

    printf("Average() = %lf\n", Average(values, sizeof(values) / sizeof(values[0])));
}
```
Here is the code for Reverse.h:

```c
#pragma once

void Reverse(char *s);
```

Here is the code for Average.h:

```c
#pragma once

long double Average(double *values, int numValues);
```

Listed below is the code for Reverse.cp:

```c
#include "Reverse.h"
#include <string.h>

static void Swap(char *c1, char *c2)
{
    char tmp;

    tmp = c1;   /* should be *c1 */
    c1 = c2;    /* should be *c1 and *c2 */
    c2 = tmp;  /* should be *c2 */
}

void Reverse(char *s)
{
    int length = strlen(s);
    int midPoint = length div 2;   /* should be //, not div */
    int i;

    for (i = 0; i < midPoint; i++)
        Swap(&s[i], &s[length - i - 1]); /*should be s, not ss*/
}
```

The code for Average.cp is as follows:

```c
#include "Average.h"
#include <SANE.h>

long double Average(double *values, int numValues)
{
    long double total = 0.0;
}
```
```c
int i;

for (i == 0; i < numValues; i++) /* should be =, not == */
    total += values[i];
if (numValues > 0)
    return total / numValues;
else
    return; /* should be return 0.0 */
```

After the project files are added, your Project window should look like the one in figure 3.1.

![Errors.png](image)

**Figure 3.1.** The Errors.png Project window.

---

**The Compile Errors of an Error-Ridden Program**

Try to build the program (by using Run or Bring Up To Date in the Project menu). As the files compile, the Compile Errors window should begin listing the following error messages:

_ File "Average.cp"; Line 9
Warning: value of expression is not used

_ File "Average.cp"; Line 14
Error: no return value for function 'Average'

_ File "Reverse.cp"; Line 8
Error: cannot implicitly convert from: char *
  to : char
As you can see, the compiler tries to compile each file even after it finds the first error—it examines the entire file. Even if it finds errors in one file, the compiler still tries to compile the rest. After the compiler completes its task, you can turn to the Compile Errors window and look at the results of its hard work. You can double-click on each error (see fig. 3.2) or select it and press Return. This takes you to the file and line on which the error occurred.

![Figure 3.2. The Compile Errors window.](image)

You can use the up and down arrow keys to move up and down through the list of errors. You also can use Go To Previous Error and Go To Next Error menu items (in the Search menu) to take you backward and forward through the listings.

If you have a number of errors in a file, you might want to adopt the strategy of starting with the last error and working your way backward. Because fixing an error can involve adding or deleting lines, the earlier error line numbers stay correct if you work in reverse order.

## Link Errors

The linker connects calls to functions with their code. It also connects the uses of globals with their definitions. If, therefore, you call a function that does not have code in the project, the
linker reports an error. It also displays an alert (for example, see fig. 3.3) that tells you that the link failed. Just click on the link alert to make it disappear and go to the Link Errors window to learn the cause of failure.

![link failed alert](image)

**Figure 3.3.** A link failure.

In this case, the error message `undefined: printf` in the window indicates that the code for the `printf` call is missing. Because this code is located in the ANSI++ library, you can fix this error by adding ANSI++ to your project. The message `undefined: gMyGlobal` indicates that the project code is using `gMyGlobal` even though it has not been defined.

You can get link errors from any of the following:

- Missing code for a function.
- Missing the definition of a global variable.
- A segment is too large.

Examine these three cases:

- Link error from missing function code. A function can be missing code if you fail to add a library or a source file. Link errors also can happen if you have a name or argument type mismatch between the call of the function and its definition.

- Link error from a missing global variable definition. When you create a global variable, you should declare it in an include file with an `extern`:

  ```
  extern int gMyGlobal;
  ```

  In one source file, you should then define it this way:

  ```
  int gMyGlobal;
  ```
The problem occurs if you declare the global in the include file, but fail to define it in a source file. In this case, the linker notifies you that the global is undefined.

- Link error from a large code segment. On the Macintosh, you must break your code into code segments. These segments are loaded on demand and then unloaded with the Toolbox call `unloadseg`. The size of each segment needs to be less than 32K. The THINK Project Manager enforces this restriction at link time by giving you an error if your segment is too large.

## Choosing the Number of Errors To Report

Symantec C++ gives you control over how many errors are reported per file. To configure this setting, bring up the Symantec C++ options dialog by selecting Symantec C++ from the Options menu in the Edit menu (see fig. 3.4).

Then, use the pop-up menu to select the Debugging panel, as shown in figure 3.5.

![Options Menu](image)

Figure 3.4. Bringing up the Symantec C++ options dialog.

In the error-reporting section of the panel, you get three separate choices. You can report only the first error in a file, the first few errors, or all of them. The wisest choice usually is to select the Report the first few errors option. Choosing the Stop at First Error option only enables you to fix one error per file per build—usually a rather time-consuming method. On the other hand, choosing the Report all errors in a file option can inundate you with an avalanche of errors. This is especially true because compilers commonly get confused by a single error and generate many spurious ones as a result.
Getting Rid of Warnings

Symantec C++ generates numerous warning messages as well as error messages. The intention of a compiler warning is different from an error message, however. A warning points to code that the compiler believes may have an error. As a result of this flexibility in interpretation, some programmers routinely ignore warnings. They figure that, if the warning were really important, it would have been an error message. Other more cautious programmers examine the code that generates warnings. They either find an actual error and fix it or, more likely, find that the warning is spurious and ignore it. Then there is the final class of programmers who examine the code that causes warnings and fix it all. If there is an error they fix it; if the warning is spurious, they modify the code so that it no longer produces the warning.

Establishing a Philosophy about Warnings

You should strive to be in this last class, with code that compiles with no warnings. There are three important benefits to this approach:

▼ Satisfaction in a job well done. It is inherently more satisfying to have the compiler accept your code without argument. It is no different than building a house. Although it may be sufficient to pass code, you have no source of pride when the building inspector says “Well, it will pass code, but I really don’t like the way you hung the drywall, or the way you strung the Romex, or the way you vented this fan.”
The increased capability to notice new warnings. If your project compiles with 43 warnings, are you really going to notice when the 44th warning appears? Warnings are mostly chaff with a sprinkle of wheat thrown in. By removing spurious warnings, you easily can notice the appearance of new warnings that could be serious.

Others have an easier time maintaining your code. Programmers are more secure working on code without warnings. Someone new to the code especially needs to be able to notice new warnings that they might introduce—something much harder to do if it is the 44th warning.

The Three Rules of Warning Elimination

Now that you are filled with renewed vigor to produce warningless code, it is time to learn the right method for doing so. Follow the rules listed below:

1. Determine whether the warning is an actual error or spurious.
2. Fix the actual errors.
3. Rewrite the code that produces spurious warnings so that it no longer produces warnings.

See how you should use these rules. Start with the first one. For example, when Symantec C++ thinks a semicolon is added incorrectly, it generates this warning:

Warning: possible extraneous ';

The following generates the warning for an if statement:

```c
if (i == 3);
i++;
```

This generates one for a while loop:

```c
while ((*s++) != '\0');
```

In the previous examples, the if statement is certainly a real error—the semicolon closes the if, so i always is incremented. The while loop, however, is probably not an error—it increments s until it reaches the end of the string.

If the warning signifies an actual error, follow the second rule and fix it. For the if statement, the fix is to remove the extra semicolon:

```c
if (i == 3)
i++
```
If the warning does not signify an actual error, follow the third rule and modify the code so that it does not generate a warning. For the `while` loop, an adequate fix is as follows:

```c
while ((*s)++ != '\0')
    ; /* do nothing */
```

An even better fix is available, however:

```c
while ((*s)++ != '\0')
{
    // do nothing
}
```

Either modification not only keeps the compiler from producing a warning, but more clearly communicates the intention of the `while` loop to the code reader.

If you simply cannot stand warnings, Symantec C++ does enable you to turn them off. Go to the Debugging panel of the Symantec C++ Options dialog (see fig. 3.5) to do so.

If you uncheck the Generate warning messages check box, you will not receive any warnings. If you uncheck the Generate optimizer warnings check box, you will not receive the warnings based on information from the optimizer. (For details on the exact messages, see Chapter 14.)

While turning warnings off can never be condoned, it can be argued that programmers who never look at warnings in the first place do not need to see them at all.

**Summary**

You now should be ready to handle the error and warning messages you receive when building your programs. You also can determine how many error messages you want to see for each file—a feature sorely missed by earlier users of THINK C. You also should have a clear understanding of the need to clean up your code so that it does not produce spurious warning messages. And, though no one wanted to tell you, you now know how to turn the warnings off.
ymantec C++ provides a debugging resource to help you find code errors in your applications. The THINK Debugger (a source-level debugger) is well worth the small amount of time it takes to learn. This chapter describes how to use the Debugger to do the following:

- Walk through your code statement by statement.
- Move through or display the entire call chain of a function.
- Move from the Debugger into the Source window to edit code.
- Set breakpoints in the Debugger and your code.
- Examine variable values and call functions from within the Debugger.
- Configure your code to use a low-level debugger, such as MacsBug, when using the THINK Debugger is not feasible.

Once you master each of these skills, you should be able to use the THINK Debugger effectively.
Turning On the Debugger

If you are ready to debug your source files, select Use Debugger from the Project menu (see fig. 4.1).

A check mark appears next to Use Debugger indicating that it is active, as shown in figure 4.2.

The Project window also slightly alters because it reserves room on the left side of each source file for a diamond symbol (see fig. 4.3).
Figure 4.3. MAIN.CP, REVERSE.CP, FACTORIAL.CP, and AVERAGE.CP can be debugged.

To enable debugging of a file, click to the left of the file name in the Project window. This toggles the diamond symbol. If the diamond is present, the file can be debugged; if it is not present, the file cannot be debugged. To turn debugging on or off for all source files simultaneously, Option-click to the left of any file name.

NOTE

You normally want all compiler optimization off while debugging because optimization can rearrange when and how the code gets executed. One example of the influence of optimization is that multiple source statements can execute as one statement in the Debugger. To turn off optimization, select the Code Optimization panel from the Symantec C++ Options dialog box and uncheck Use Global Optimizer.

Starting Programs with the Debugger Activated

After the Debugger is turned on, select Run. The THINK Project Manager immediately launches the THINK Debugger, and the Debugger window appears. The first line of your program is displayed in the Debugger window (see fig. 4.4). Because the Debugger is an application that the THINK Project Manager launches, it has its own menu bar.
Command Options

If you look at the window in which the source code is displayed, you see the Go, Step, In, Out, Trace, and Stop buttons. These are the commands that enable you to control the execution of your program. Along with these commands, the window also displays information about the running program. Figure 4.5 shows the Source window and its parts.

The current statement arrow points to the statement that executes next. You use the Status panel buttons to control program execution. The readouts show what is happening in your program. Select the Go button to run your program; select the Stop button to stop execution. In figure 4.5, the Stop button is selected.
You also can select commands from the Debug menu; each has a command key equivalent (see fig. 4.6).

![Debug menu](image)

**Figure 4.6.** The Debug menu.

### The Step Command

If you select Step, the current statement is executed, and the Debugger stops. All function calls in the statement are stepped over (that is, ignored). If you are at the end of a function, Step returns you to the calling function. Figures 4.7 and 4.8 show before and after stepping over a statement without a function call.

![Debug session](image)

**Figure 4.7.** A statement without a function call.
The stepping action steps over statements that have function calls. Figures 4.9 and 4.10 show before and after stepping over a function call statement.

**Figure 4.8.** After stepping over the statement.

**Figure 4.9.** Before stepping over a function call.
The Trace Command

If you select Trace, the Debugger executes one statement and then stops. If the current statement contains a function call, however, the Debugger moves to the called function code and stops at the first statement there. Figures 4.11 and 4.12 show how tracing works on this kind of statement.
If you request a Trace command on a function for which no source code is available, such as a library, the current statement marker disappears. Figures 4.13 and 4.14 show the before and after status for tracing a statement containing a function call that has no available source code. In figure 4.14, you can see that the current file is displayed correctly even though the marker is absent.

Figure 4.13. Before tracing into a function call with no source code.
The Out Command

If you select Out, the Debugger continues until it steps out of the current function. For the most part, you use this command when you have finished debugging the current function. Figures 4.15 and 4.16 show before and after using the Out command while in a function.

Figure 4.14. After tracing into a function call with no source code.

Figure 4.15. Before stepping out of a function.
The In Command

If you select the In command, the Debugger continues stepping through your program until it reaches the next statement that has a function call. After entering the called function, it stops at the first statement. Figures 4.17 and 4.18 show what happens before and after using the In command to step into a function.
The Go Command

If you select the Go command, the Debugger continues executing until one of the following occurs:

▼ The program finishes running.

▼ You use Stop to halt the program.

▼ It reaches a breakpoint.

▼ A Debugger or DebugStr call is executed.

▼ The program causes a system error (such as a bus, address, or divide by zero error).

The Stop Command

If you select the Stop command, the Debugger stops at the next call to GetNextEvent or WaitNextEvent.

Putting a Command in Automatic Mode

All the commands, except Stop, can operate in automatic mode. When a command is in this mode, the Debugger automatically reexecutes it every time it finishes. For example, by running the Trace command in automatic mode, you can view your program as it executes line by line. To run a command in automatic mode, Option-click on its button. To cancel automatic mode, press Command-Shift-period or double-click the Stop button.
Showing the Call Chain

The Source window can show you more than just the currently executing function. It also can display the chain of functions that leads to the current call, all the way back to the main program. Figure 4.19 shows the Call Chain pop-up menu. To display this menu, click on the current function name at the bottom of the source window.

Figure 4.19. The Call Chain pop-up menu.

If you want to move to one of the call chain functions, select the function name in the Call Chain pop-up menu. The function is displayed in the Source window. The highlighted line shows where the current function returns (see fig. 4.20).

Figure 4.20. After selecting main from the Call Chain pop-up menu.
Examining Functions in the Source Window

If you want to look at a function in the Debugger Source window that currently is not displayed there, you need to open the function in the THINK Project Manager. To do that, follow these steps:

1. Go to the THINK Project Manager.
2. Open the source file that contains the function.
3. Select a line in the function.
4. Select Debug from the Source menu (see fig. 4.21).

You are transported to the THINK Debugger. The Source window shows the function with the line you selected highlighted (see fig. 4.22).

What about the other way around? Suppose that you are looking at code in the THINK Debugger and want to see it in the project source file. Simply select Edit from the THINK Debugger Source menu, as shown in figure 4.23.
Using Breakpoints

You also can set breakpoints in your code in the THINK Debugger. To set a breakpoint at a particular statement, click on the statement marker. The marker changes from a hollow to a filled diamond (see fig. 4.24).

With the breakpoint in place, you can use the Go command. Execution continues until the breakpoint is reached; then, the program stops (see fig. 4.25).
You do not have to make the breakpoint permanent, however. To set a temporary breakpoint, Option-click on the statement marker. The THINK Debugger continues until it reaches that statement. It sets the breakpoint, executes until it gets to the statement, and then removes the breakpoint.

Examining and Setting the Values of Variables

When you want to look at variable values, you use the Data window in the Debugger. Figure 4.26 shows the Data window.
To examine a variable value, just type an expression into the entry field of the Data window and press Enter (see fig. 4.27). You can enter a variable or a more complicated expression if you prefer.

```cpp
void Reverse(char *s) {
    int length = strlen(s);
    int midPoint = length / 2;
    int i;
    for (i = 0; i < midPoint; i++)
        Swap(s[i], s[length - i - 1]);
}
```

There is a shortcut for slow typists: Select an expression in the Source window, and then choose Copy To Data from the Debugger Edit window (see fig. 4.28).
In either case, the expression is shown in the expression list on the left. The value of the expression is shown in the value list on the right (see fig. 4.29).

![Figure 4.29. Evaluating an expression.](image)

If you want to change the value of a variable, select its current value in the value list. Type a new value and press Enter, or click the check mark button when you have finished (see fig. 4.30).

![Figure 4.30. Changing the value of a variable.](image)

Double-clicking in the right value column also enables you to do either of the following:

- Look at the contents of a structure or an array.
- Dereference a pointer.

If you want to do a double dereference, (in other words, dereference the pointer twice) just Shift-double-click. Figure 4.31 shows an example of an array with its contents displayed.

Each time you enter the Debugger, the expressions in the Data window are reevaluated. If you no longer want a particular expression reevaluated, select it and choose Locked from the Data menu. To clear an expression, select it and choose Clear from the Edit menu. To clear all the expressions at the same time, from the Data menu, select Clear All Expressions.
Using Debugger Statements in Programs

The Toolbox calls Debugger or DebugStr can be called from within your code. If called, they cause your program to break into the THINK Debugger. If your program is not running with the THINK Debugger, the calls break into your low-level debugger (MacsBug, TMON, or TMON Pro). If you are not running a low-level debugger, the calls snarl up everything and cause a system error (ID 10).

Following is an example function that calls both Debugger and DebugStr:

```c
void Foo()
{
    int i;
    i = 5;
    Debugger();
    i = 10;
    DebugStr("\pAfter i = 10");
}
```

Figure 4.32 shows what happens when the function’s Debugger call is reached.
Figure 4.32. The effect of a Debugger call.

Similarly, figure 4.33 shows what happens when the DebugStr call is reached.

Figure 4.33. The effect of a DebugStr call.

Calling Functions from the THINK Debugger

You can do more than just evaluate variables in the Data window—you can call functions. You must exercise caution when using this feature, however. If the function has a side effect, like changing a global variable value, then the Data window does not display the global variable correctly. Worse yet, if the function exits the program (with exit or ExitToShell), havoc ensues.

There also are restrictions on the kinds of functions you can call. This, coupled with the danger of side effects, is the reason calling functions from the debugger is not allowed by default. To specifically enable such calls, use the Debugging panel in the THINK Project Manager Options dialog box and check Allow dataview function calls (see fig. 4.34).
Close the current project and open the Binary Tree project from the disk. Now, check Allow dataview function calls in the THINK Project Manager Options dialog box.

A particularly valuable use of calling functions from the data window is to call data validation routines that check a data structure. If, for example, you have a binary tree, you can write a routine that checks to make sure that the tree sorts correctly, as follows:

```c
Boolean CheckBinaryTree(BinaryTree *aTree)
{
    if (aTree == NULL)
        return true;
    if (aTree->left != NULL) {
        if (aTree->left->value > aTree->value)
            return false;
        if (!CheckBinaryTree(aTree->left))
            return false;
    }
    if (aTree->right != NULL) {
        if (aTree->value > aTree->right->value)
            return false;
        if (!CheckBinaryTree(aTree->right))
            return false;
    }
    return TRUE;
}
```
You then can call this function as you step through your program, keeping an eye on the result of the CheckBinaryTree call the whole time. If the result is ever 0, you know that the binary tree is invalid and that the code just executed is the culprit.

Following is a main program that you can use to try this feature out on a binary tree problem:

```c
void main()
{
    /* construct binary tree by hand. 
     * Should really write a function to do it */
    BinaryTree *root = NULL;
    BinaryTree *node;

    node = NewNode(5);
    root = node;

    node = NewNode(3);
    root->left = node;

    node = NewNode(4);
    root->left->left = node; /* should be root->left->right */

    node = NewNode(8);
    root->right = node;
}
```

Now that you have the code to debug, it is time to call a function. Run the program. Click Step a few times to get past the creation of the root data. Next, type a call in the entry field of the Data window (see fig. 4.35).

![Figure 4.35. Calling a function from the Data window.](image-url)
Expressions are evaluated each time you enter the Debugger. In other words, by single-stepping through the function, at every statement the binary tree is checked. Figure 4.36 shows the Debugger before the line that makes the tree invalid; figure 4.37 shows the Debugger after that line has executed.

![ Debugger before invalidating the binary tree. ](image1)

![ Debugger after invalidating the binary tree. ](image2)

**Low-Level Debuggers**

You normally use the THINK Debugger for debugging. There are cases, however, when this is not feasible. You may have a bug that does not appear when running with the Debugger. The bug might appear when you run the built application, only to disappear when run from the Project Manager. Or you could be away from work and not have the source code with you.

The Symantec C++ compiler can help you when you cannot use the THINK Debugger. As a source-level debugger, the THINK Debugger gets function names from symbolic information created by the compiler. A low-level debugger (like MacsBug, TMON, or TMON Pro) does not
have access to additional symbolic information. Fortunately, there is a way for the compiler to
give low-level debuggers displayable function names. It can place MacsBug names at the end of
the function. There is nothing complicated about this. Function names are just placed after the
executable portion of the code.

These names can then be used by the low-level debugger to display function names. MacsBug,
for instance, uses the names to display a stack crawl, as follows:

Calling chain using A6 links
A6 Frame     Caller
top level 0059820c 'CODE 0001 0FCC'+0078
005FABA8 005A0348 main+0066
005FAB2C 005A0424 Reverse(char *)+003A

Without the names, MacsBug is reduced to displaying the following:

Calling chain using A6 links
A6 Frame     Caller
top level 0059820c 'CODE 0001 0FCC'+0078
005FABA8 005A0338 'CODE 0002 0FCC'+755C
005FAB28 005A0402 'CODE 0002 0FCC'+7626

To turn on MacsBug names, use the Debugging panel of the Symantec C++ Options dialog box
(see fig. 4.38). Check the Generate MacsBug Names check box.

![Debugging panel of the Symantec C++ Options dialog box](image)

**Figure 4.38.** The Debugging panel of the Symantec C++ Options dialog box.
Using Two Screens

If you have a multiple-monitor Macintosh, you can place the THINK Debugger windows on a second screen. This way, they don’t obscure the windows of the project files. Although you could do this manually, be lazy and let the THINK Debugger place your windows for you. To use automatic placement, check Use Second Screen in the Debugging panel of the THINK Project Manager Options dialog box (see fig. 4.39).

![Debugging panel](image)

**Figure 4.39.** The Debugging panel of the Project Manager Options dialog.

Summary

You should now know how to use the THINK Debugger to do the following:

- Step or trace through your code statement by statement.
- Display the entire call chain of a function and move within it.
- View routines in the Debugger or in the project source files.
- Set temporary or permanent breakpoints.
- Use the Data window to examine variable values and call functions.
- Append MacsBug names to your code to make low-level debugging easier.

With the THINK Debugger, you make a formidable exterminator.
An innovative aspect of the Macintosh interface is that it can easily be tailored to the native language of the user. A Quebec resident can expect to buy a Macintosh application and see not just French, but French Canadian displayed in the dialog boxes. Macintosh keeps this task easy for programmers by storing all the information required for localization in resources. You can modify the information in the resources (such as strings, dialog boxes, menus, and icons) without touching the executable code.

Programmers new to Macintosh often are startled by this strict separation of code and data. It rarely takes them long to move from consternation to approval, however, because separation is a better way to design applications for an international world.
Development environments differ in how they get these resources into the program. Symantec C++ supports two different methods. One approach is to create resources using ResEdit, Apple’s graphical resource editor. The other approach is to use THINK Rez, a tool that converts the resource information in a text file into resources. This chapter shows you how to use both methods.

Writing a Program That Uses Resources

Before you can create resources, you need a program that uses them. The following sections look at one program that has resources to help create words.

Creating the Project

Make a new project called Alphabet or open the one on the source code disk. This project contains the Main.cp and ResourceDefinitions.h source files.

Following is the code for Main.cp:

```c
#include <stdio.h>
#include <Types.h>
#include <OSUtils.h>
#include <string.h>
#include <ToolUtils.h>
#include "ResourceDefinitions.h"

#define kMaxInputLine 512

/* prompt for character input */
char gPrompt[256];

/* if they type an invalid character */
char gBadCharacter[256];

/* output if they type a valid character. This is used as the
 first argument to printf */
char gCharIsForWordTemplate[256];

Boolean FindIndexOfCharacter(char c, int *ip)
{
    const kCaseSensitive = true;
    const kDiacriticalSensitive = true;
    Str255 entry;
```
int index = 0;
Str255 charAsString;

charAsString[0] = 1;
charAsString[1] = c;

do {
    GetIndString(entry, kCharsResourceID, ++index);
} while (StrLength(entry) != 0 &&
    !EqualString(charAsString, entry,
        !kCaseSensitive, !kDiacriticalSensitive));

*ip = index;
return StrLength(entry) != 0;

/* returns false if no more input */
Boolean HandleOneInteraction()
{
    int c;
    char inputLine[kMaxInputLine];

    printf("\n%s", gPrompt);
    if (fgets(inputLine, sizeof(inputLine), stdin) == NULL)
        return false;
    else {
        int i;

        if (!FindIndexOfCharacter(*inputLine, &i))
            printf("%s\n", gBadCharacter);
        else {
            Str255 word;

            GetIndString(word, kWordsResourceID, i);
            printf(gCharIsForWordTemplate, *inputLine, p2cstr(word));
            putchar('\n');
        }
        return true;
    }
}

void InitializeStrings()
{
    Str255 s;
Following is the code for ResourceDefinitions.h:

```c
#define kWordsResourceID 1024
#define kCharsResourceID 1025
#define kMiscStringsID 1026
#define kPromptItem 1
#define kBadCharacterItem 2
#define kTemplateItem 3
```

After you have added all the appropriate source files and libraries, you should have a project that looks like the one shown in figure 5.1.

![Figure 5.1. The Alphabet Project window.](image-url)
What the Program Does

The program queries the user for a character. It then returns a word that begins with that character. The program is not ready to run yet, but the following output should give you an idea of the interaction (the user input is in italic):

Character? a
a is for Alexander

Character? d
d is for Daddy

Character? M
M is for Mommy

Character? u
u is for Unicornfish

Character? j
j is for Jagular

Character? ;
Invalid character

This program uses string resources in three different ways: as a list of valid characters, to find a corresponding string, and as utility strings for prompts and messages.

Including Resources in the Program

Before the program can run, the string resources need to be in the right place. This place is in the file named Alphabet.rs.rsrc. You can get the resources into a rs.rsrc file in one of two ways. The first method is to use ResEdit.

Creating Resources with ResEdit

You can create a new rs.rsrc file with ResEdit, Apple's graphical resource editor.
To create the resources for the Alphabet program, you need to do the following:

1. Launch ResEdit.
2. Create a new file, Alphabet.n.rsrc, and put it in the project folder.
3. Create an 'STR#' resource with ID 1024 containing a list of words that start with the letters A through Z.
4. Create an 'STR#' resource with ID 1025 containing the letters A to Z.
5. Create an 'STR#' resource with ID 1026 containing the following strings:
   "Character?"
   "Invalid character"
   "%c is for %s"

Figure 5.2 shows ResEdit in the act of creating these resources.
Adding Resources Directly

A better use of ResEdit is to explicitly add new .rsrc files to your project using the Add Files dialog box. The next time you bring the project up-to-date, the THINK Project Manager copies the resources to the Alphabet.n.rsrc file.

To take this approach, do the following:

1. Start out with a clean slate by deleting the current Alphabet.n.rsrc.
2. Use ResEdit to create a resource file named AlphabetStrings.rsrc (or any other name that ends in .rsrc) containing the same 'STR#' resources you used in the previous example.
3. Add the file you created in step 2 to your project (see fig. 5.3).
4. When you bring the project up-to-date, the resources in AlphabetStrings.rsrc are copied to Alphabet.n.rsrc.

There are two specific advantages to this approach:

▼ You can organize your resources in different files.
▼ If you open a .rsrc file from the project list, it opens the file in ResEdit. It even launches ResEdit for you.
Creating Resources with THINK Rez

ResEdit is a graphic tool for adding resources to a program. The second approach to creating resources with the THINK Project Manager involves the use of the text-based tool, THINK Rez. This tool compiles a textual resource description into a file of resources. (MPW has a similar tool, the Rez tool.)

THINK Rez stores your text file, full of resource information, in the project. THINK Rez then converts its contents into resources. To see how this is done, look at a text file first.

The following is an Alphabet.r text file that contains THINK Rez descriptions of the string resources:

```
#include "Types.r"
#include "ResourceDefinitions.h"

resource 'STR#' (kCharsResourceID, "Characters", purgeable)
{
    {"a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z"}
};

resource 'STR#' (kWordsResourceID, "Words", purgeable)
{
    {}
```

};

resource 'STR#' (kMiscStringsID, "Misc", purgeable)
{
    {
        "Character? ",
        "Invalid character",
        "%c is for %s"
    }
};

Alphabet.r uses the same ResourceDefinitions.h file included by Main.cp. By setting up the information in this manner, both files can share the definitions of the resource IDs.

Before you can use Alphabet.r, however, you need to add it to the project. Remember to remove AlphabetStrings.rsrc at the same time. Your project window should look like the one shown in figure 5.4 when you are finished making the changes.

![Figure 5.4. Alphabet.r in the project.](image-url)
NOTE

The THINK Project Manager uses the .r extension to indicate that the THINK Rez translator is used. Your Rez files should always end with the .r extension.

The next time you update this project (or run it), you will see the THINK Rez compiler progress dialog box (see fig. 5.5).

![Progress dialog box](image)

**Figure 5.5.** The THINK Rez compiler progress dialog box.

Comparing THINK Rez with ResEdit

You may be asking yourself whether you should take the THINK Rez or ResEdit road to creating resources. Each of the approaches has its benefits.

THINK Rez Advantages

One advantage of using THINK Rez files is that they print out easily. A text format is better for printing, particularly for use in a project notebook, magazine article, or book.

Another advantage is that you can include files that define resource identifiers. You are guaranteed to have identical IDs for the resource ID and the one that is used when the code is accessed. This is why it's easy to avoid inconsistencies when you change IDs.

You also can use the preprocessor to conditionally include or exclude particular text. THINK Rez is great if you want to maintain one .r file that generates different resources depending on the state of preprocessor variables. For example, the following code selects different text strings depending on the language being used:

```
#define qEnglish
#undef qPigLatin
```
resource 'STR#' (1028)
{
  {
    #if defined(qEnglish)
      "Boat",
      "Car"
    #elif defined(qPigLatin)
      "OatBay",
      "ArCay"
    #endif
    
  }
};

And finally, you can define your own resource types. For more information on how to do this, see the THINK Rez documentation.

ResEdit Advantages

ResEdit is not without its advantages as well. It is a better design tool. As a graphical environment, ResEdit enables you to see the effects of your actions immediately. If you change a dialog box, you can look at it without a new build.

You also can copy resources from other files. You might, for example, want to copy a cursor from the system file to use in your application.

Summary

You should now be familiar with the process of writing a program that uses resources. ResEdit can be extremely helpful, especially when you are explicitly adding .rsrc files to your project. THINK Rez, the textual tool, is another approach to creating resources with the THINK project manager. Both THINK Rez and ResEdit have their advantages.

Exercise

Make the alphabet program Script Manager compatible. Make sure that it handles scripts that have more than one byte per character. This is sometimes difficult, but makes the program fully localizable for languages like Kanji that use multi-byte scripts.
Building an Application

During a development cycle, you usually run your program from within the Symantec C++ environment. Eventually, however, you have a program that is ready for testing or, better yet, for that golden day when you produce a master release. Building the stand-alone version of your application does require setting some particular options. This chapter discusses these options and other aspects of creating the final application. You learn how to set all of the following:

- The creator type
- The application partition size
- Certain important SIZE flags
- The Far DATA and Far CODE options
- The Smart Link option for your builds
Except for the Smart Link option, all of these settings are located in the same area of the Project menu. These options are examined in the following sections.

**Setting Build Options**

To access the options that affect your stand-alone application, from the Project menu, you select the Set Project Type option (see fig. 6.1). The Set Project Type dialog box is displayed (see fig. 6.2).

![Fig 6.1](image)

**Figure 6.1.** You select Set Project Type from the Project menu to display the Set Project Type dialog box.

![Fig 6.2](image)

**Figure 6.2.** The Set Project Type dialog box.

This dialog box contains four different types of information that you must specify about your program:
<table>
<thead>
<tr>
<th>Program type</th>
<th>The buttons in the upper left area of the dialog box specify what kind of program it is. You can create a desk accessory, device driver, code resource, or application.</th>
</tr>
</thead>
<tbody>
<tr>
<td>File type</td>
<td>Normally APPL, the file type and creator settings are used to identify and differentiate your program from every other program.</td>
</tr>
<tr>
<td>Memory</td>
<td>The settings in the lower left area of the dialog box set how much memory the program needs to run and how it takes advantage of special System 7 and MultiFinder features.</td>
</tr>
<tr>
<td>Size settings</td>
<td>The options in the lower right area of the dialog box include special settings reserved for bulky applications.</td>
</tr>
</tbody>
</table>

Each choice in the dialog box, starting with the most important, is discussed in the following sections.

**Creator**

By far the most important thing to do in the Set Project Type dialog box is to set the four-character creator in the Creator edit box. This also is the step that takes the most planning.

**NOTE**

Every application should have a unique creator. The Finder uses this value to display the correct application icon and to launch the right application when a document is opened. To guarantee uniqueness for your programs, you must register your proposed creator with Apple's Developer Support Center. To obtain a registration form, write to Developer Support Center, Attn: C/F Type Administrator, 20525 Mariani Ave., M/S 303-2T, Cupertino, CA 95014. You also can use AppleLink (DEVSUPPORT) or Internet (DEVSUPPORT@applelink.apple.com).

You can use any characters you want in the four-character sequence. Your creator cannot consist entirely of four lowercase letters, however, because they are reserved for Apple programs.

**Program Type**

By selecting the appropriate radio button, you can make your program one of four types: application, desk accessory, device driver, or code resource.
Application is the most common choice. Historically, desk accessories (such as the Scrapbook) were the only items that appeared in the Apple menu and they could be accessed only from there. The distinction between applications and desk accessories has blurred with the advent of System 7 because you can put applications in the Apple menu and run desk accessories from the desktop.

Device drivers typically interact with hardware. Some examples include drivers for NuBus cards and the LaserWriter driver for PostScript printers. Code resources are blocks of code used in various situations. For example, start-up documents (INITs) and window definition procedures (WDEFs) are included in the code resource category.

**Partition Size**

You also must set the partition size of your program in the Partition (K) edit box. A typical application has a default value between 384 and 2,500K. The default value of 384K is suitable for small to medium applications, but you may need a larger value. You want to avoid setting the default value too low because your application can easily run out of memory and otherwise annoy the user. The number you set is used for all three partition sizes the user sees in the Finder (see fig. 6.3).

![Figure 6.3. Partition sizes in the Finder.](image)

**The SIZE Flags**

You also must set the SIZE flags. At build time, they are stored in the SIZE resource with ID-1 in the application. They provide information to the operating system about the application's
capabilities. Although you can set the bits by hand (by typing in the hex value for the flags), it is much easier to use the SIZE Flags pop-up menu (see fig. 6.4).

![Figure 6.4](image.jpg)

**Figure 6.4.** Using the SIZE Flags pop-up menu.

When you select a flag, the dialog box places a checkmark in the pop-up menu and calculates the appropriate value for the combined flags (see fig. 6.5).

![Figure 6.5](image.jpg)

**Figure 6.5.** Setting SIZE flags using the SIZE Flags pop-up menu.
These flags fall into three categories of information about your application's MultiFinder and System 7 support:

- How well your application follows MultiFinder and System 7 guidelines. (The MultiFinder-Aware, Background Null Events, and Suspend & Resume Events flags.)

- If it runs in the background and can receive application information from the Finder. (The Background Only, GetFrontClicks, and Accept ChildDiedEvents flags.)

- Whether your application is 32-bit clean, can receive high-level events (like AppleEvents) both locally and remotely, can create stationery, and can use Script Manager in-line text entry. (The 32-Bit Compatible, HighLevelEvent-Aware, Accept Remote High-LevelEvents, Stationery-Aware, and Use TextEdit Services flags.)

**NOTE**

If you are using the THINK Class Library, you should set the MultiFinder-Aware, Suspend & Resume Events, and High-LevelEvent-Aware flags.

For a complete description of each flag, see the Symantec C++ product documentation.

**Far CODE and Far DATA**

Calls to functions in different segments of the project are made using the jump table. Each function called by another one in a different segment must have a jump table entry. Functions for which you take addresses must each have a jump table entry as well (because these functions could be called from anywhere). Finally, virtual member functions take a jump table entry.

With so many functions requiring entries, the jump table can fill up quickly. If it does, check the Far CODE option to make the jump table larger. If Far CODE is unchecked, the references to the jump table are made using 16-bit (positive) values. Because each jump table entry is eight bytes, this limits the number of jump table entries to 4,096.

If Far CODE is checked, references to the jump table are made using 32-bit values. This increases the number of jump table entries dramatically. But remember that there is a price to pay for such ample leg room. Your application becomes somewhat larger because it takes more code to use 32-bit values. Programs with the Far CODE setting checked have an average six percent increase in their code size.
You encounter the same issues with global variables. These variables (variables defined outside any function, static variables within a function, and static data members) are accessed by a method similar to jump table entries.

If the Far DATA option is off, globals are accessed by 16-bit (negative) values. This limits the size of global space to 32K. If the Far DATA option is on, globals are accessed using 32-bit values, which makes the size of global space essentially unlimited. The price you pay is that your code becomes slightly bigger.

**NOTE**

If you combine code that uses the Far CODE or Far DATA options with libraries that do not use these settings, you need to consult the Symantec C++ documentation for instructions.

**Separate STRS**

Normally, string constants (like “Hello” or “\pWorld”) are stored in global space. If the Separate STRS option is on, string constants are stored in a special STRS component instead.

This option exists for compatibility with earlier versions of THINK C—don’t use it for new code. This option is an artifact from the time when you could only have 32K of globals. There were some programs with large numbers of string constants that exceeded the 32K limit. The STRS components were the kludge that resulted. Times are better now, and you can use Far DATA if you need to exceed 32K of globals.

**Building the Application**

Now that every setting is configured properly, you can build your application by selecting Build Application from the Project menu. If your project is not up-to-date, you receive the Bring the project up to date message (unless you have made updates automatic).

After your source files have compiled and your libraries have loaded, you must decide where to place the application and what to name it (see fig. 6.6).
After you click Save, the THINK Project Manager begins linking your application (see fig. 6.7).

After the application is linked, the Project Manager copies the resources (from Project.rsrc) into your application.

Quit the THINK Project Manager and give your application a whirl.

**Smart Link**

The Smart Link option makes your code smaller by removing functions that are never referenced. For example, it is common to have unreferenced functions when using libraries because you rarely use all the functions they contain.

Smart Link has its price; it takes time to decide what functions to strip. A very rough rule of thumb is that linking using Smart Link takes 50 percent longer, but makes your application 50 percent smaller.
NOTE

Regardless of the time it adds to the build, you always should turn on Smart Link for your final application build.

Summary

This chapter described all the aspects you need to consider when it is time to create your final application. Except for Smart Link, all these options are found in the Set Project Type dialog box. You also learned not to wait to the last minute to send your file creator type to Apple. It would be tragic if all that stood between you and a shrink-wrapped package were four characters.
ymantec C++ is a quick compiler inside an efficient environment, but the way to get your compiler really flying is to use precompiled headers. These files provide a way to store the information from a number of header files in a compiled format. The Symantec C++ compiler then can read this compiled format in the twinkling of an eye and, thereby, forgo reading the individual header files.

To use precompiled headers in your source files, replace your standard include files with one #include of the precompiled file. This chapter shows you how to do this. You also learn how to use the standard precompiled headers provided with Symantec C++ and how to make your own versions. You also are shown some of the pitfalls to avoid when making modifications to header files or to the files that use them.
Using Precompiled Headers

There are a couple of rules for using precompiled headers, but they are easy to remember.

**RULE**
The `#include` statement for the precompiled header must precede everything but comments in your source file.

You can have other `#includes` in your file, but they must follow the precompiled header. This also is true of the declarations. Your files generally look like this:

```c
#include <myPrecompiledHeaderfile++>
#include <Quickdraw.h>

void Foo()
{
    Rect r = {10, 20, 50, 100};
    FrameRect(&r);
}
```

Or, they can look like this:

```c
// Here is the main program

#include <myPrecompiledHeaderfile++>
#include "Foo.h"

void main()
{
    printf("%d %d\n", kFoo, kBar);
}
```

**RULE**
Only one precompiled header file is allowed per source file.

Different source files can include different precompiled headers, however. But be careful: If you put the precompiled header file in the project prefix, you cannot add any other precompiled header files to individual source files.
Standard Precompiled Header Files

Symantec C++ comes with two already built precompiled headers: MacHeaders++ and TCLHeaders. MacHeaders++ contains about 40 different standard Macintosh header files. TCLHeaders is for use with the THINK Class Library. It contains commonly used Macintosh header files and most of the TCL header files. If you are using the THINK Class Library, you probably want to include TCL Headers.

By default, Symantec C++ automatically includes MacHeaders++ in every source file. It does this through the Prefix panel, which you access by choosing Prefix from the Edit menu (see fig. 7.1). By using the Prefix panel, you can add information to the beginning of each source file; normally, this is an include of a common header file. The default setting in the panel is to include MacHeaders++.

![Prefix panel of the Symantec C++ Options dialog box.](image)

Figure 7.1. The Prefix panel of the Symantec C++ Options dialog box.

Effects of Compiling with Precompiled Headers

Following is a small sample file that demonstrates the advantages of automatic inclusion of MacHeaders++:

```c
void Foo()
{
```
First, the Prefix panel ensures that MacHeaders++ is included when this source file is compiled. Because MacHeaders++ includes QuickDraw.h, Foo.h does not have to include it explicitly in the declaration of FrameRect. When this file is compiled, the Progress window shows a five-line compile (see fig. 7.2).

![Progress window with MacHeaders++ in the Prefix panel.](image)

Figure 7.2. Progress window with MacHeaders++ in the Prefix panel.

If you include a file already in MacHeaders++, it does not affect your compile. For example, it does not hurt to include QuickDraw.h. Because it already has been included (via MacHeaders++), its contents are skipped the second time. Following is the file with Quickdraw.h explicitly added:

```c
#include <Quickdraw.h>

void Foo()
{
    Rect r = {10, 20, 50, 100};
    FrameRect(&r);
}
```

When you compile this example, the Progress window shows a seven-line compile—one additional line for the `#include` and a blank line (see fig. 7.3). Had the compiler also compiled QuickDraw.h, many more lines would have been added.

If you remove the `#include` of MacHeaders++ from the Prefix window, it is another story altogether. The compile gives an error if you forget to include QuickDraw.h:

```c
void Foo()
{
    Rect r = {10, 20, 50, 100};
    FrameRect(&r);
}
```
Figure 7.3. Progress window with MacHeaders++ in the Prefix panel and include of QuickDraw.h.

Both Rect and FrameRect are now undefined without MacHeaders++ (see fig. 7.4).

Figure 7.4. Results with neither MacHeaders++ nor Quickdraw.h.

You also can include QuickDraw.h without using MacHeaders++. The following file compiles correctly:

```c
#include <Quickdraw.h>

void Foo()
{
    Rect r = {10, 20, 50, 100};
    FrameRect(&r);
}
```

Compare the number of lines now compiled with the earlier examples (see fig. 7.5).
Progress window without MacHeaders++ but with include of Quickdraw.h.

Now imagine the more common situation in which you include more than one standard Macintosh header file. The difference between compile times for files using MacHeaders++ and those that rely on standard #includes can be staggering—like driving a Lotus at moped speeds.

Using Precompiled Headers in Individual Files

As long as you remove MacHeaders++ from the Prefix panel, you can include it or some other precompiled header file in any individual source file:

```c++
#include <MacHeaders++>

void Foo()
{
    Rect r = {10, 20, 50, 100};
    FrameRect(&r);
}
```

If you were to compile this, the Progress window would again show a seven-line compile.

You also might have a situation in which you want to use MacHeaders++ on one source file and a special precompiled header file on another source file.
Creating Precompiled Headers

There comes a time when you want a custom precompiled header file that contains your own set of header files. For instance, MacHeaders++ does not include PrintTraps.h. If you want to print, you either can include PrintTraps.h in a source file or, faster yet, create a custom version of MacHeaders++ that includes the file.

If you find that a number of standard project header files are lengthening your compile times, they are good candidates for inclusion.

The most important rule for precompiled header files also is pretty obvious.

RULE
Do not precompile header files that change often.

If your precompiled header files are not fairly stable, they will require precompiling often. Situations like this negate the very benefits of precompiled headers.

Changing Header Files

There is a substantial problem with the way the THINK Project Manager deals with precompiled header files. The situation is made worse by how well it handles standard header files.

The problem is that changes to header files are not reflected automatically in the precompiled header file that includes them. The THINK Project Manager marks neither the precompiled header file nor the source file as out of date.

Walk through some source and header files to see how this problem affects the code. If you change a header file that is included by a source file, the THINK Project Manager nicely marks the source file as being in need of compilation. For instance, look at the following example files.

This is the code for Foo.h:
#include "Bar.h"
#define kFoo 5

This is the code for Bar.h:
#define kBar 10
Main.cp is as follows:

```c
#include <stdio.h>
#include "Foo.h"

void main()
{
    printf("%d %d\n", kFoo, kBar);
}
```

If you modify Foo.h, the THINK Project Manager marks Main.cp as needing recompilation because it includes Foo.h. Similarly, if you modify Bar.h, the THINK Project Manager marks Main.cp as needing recompilation because Main.cp includes Foo.h, which includes Bar.h.

Precompiled headers are not handled the same way. For example, following is the same program using a precompiled header file, FooBar++, that includes Foo.h and Bar.h. Main.cp is similarly modified to include FooBar++:

```c
#include "FooBar++"
#include <stdio.h>

void main()
{
    printf("%d %d\n", kFoo, kBar);
}
```

Now, modify Foo.h and Bar.h. Following is the code for Foo.h:

```c
#include "Bar.h"
#define kFoo 500  //changed the value
```

Here is the code for Bar.h:

```c
#define kBar 1000  //changed the value
```

In this case, the THINK Project Manager does not mark FooBar++ as needing a new precompilation. Therefore, FooBar++ still contains the old Foo.h and Bar.h. If you do not precompile FooBar++ over again, the program erroneously prints out:

```
5 10
```

The only bright spot is that the THINK Project Manager marks files correctly if you do your own precompiling. In the previous example, if you rebuild FooBar++, the THINK Project Manager realizes that Main.cp needs to be recompiled.
This situation gives rise to another rule to remember when working with precompiled header files.

**RULE**

It is up to you to precompile a file whenever one of its included header files changes. If you don’t, mysterious things happen.

# Creating a Precompiled Header File

Creating a precompiled header file is a fairly straightforward process. Follow these steps:

1. Create a `.cp` file that `#include`s all the headers you want in the precompiled header file. For example, create a new file named `FooBar.cp` (or anything else, as long as it ends in `.cp`). Then, add the following `#include`s:

   ```
   #include "Foo.h"
   #include "Bar.h"
   ```

2. Precompile the `.cp` file by choosing Precompile from the Source menu (see fig. 7.6).

![Precompiling headers](image)

**Figure 7.6.** Precompiling headers.
WARNING

A common mistake programmers make is attempting to precompile an .h file instead of a .cp file. The Precompile command only works on .cp files and is enabled only when a .cp file is frontmost. Try as you might, you can never precompile an .h file.

While the .cp file precompiles, the THINK Project Manager shows a Progress window (see fig. 7.7).

![Progress window while precompiling.](image)

**Figure 7.7.** Progress window while precompiling.

After the precompile is complete, you see a dialog box requesting the name and location of the new precompiled header file (see fig. 7.8).

![Saving the precompiled header.](image)

**Figure 7.8.** Saving the precompiled header.

3. Name the precompiled header file, making sure to include the ++ ending. A precompiled header file specific to a project should be stored in the project tree. If the header file is used by many projects, store it in the THINK tree.
WARNING

The ++ on the end of the precompiled header file is a convention indicating that the header file is a C++ compilation. Although both THINK C and Symantec C++ support precompiled headers, the format they use is different. Do not use a header file created by one compiler with the other compiler.

4. Place the .cp file in a location that enables easy access. If any of the header files change, then you can open the .cp file and choose Precompile again. When you are prompted to save the precompiled header file, replace the old version with the new one.

NOTE

You can add the .cp file to your project if you want. This can be both good and bad. You have easy access to the file if you have to modify it. This method can be confusing, however. You may be tempted to think the project uses the file or, worse yet, takes it into account when updating the project files.

If you store a project on a floppy, or transmit it electronically, you may find the precompiled header file rather bulky. In this case, remove the precompiled header and provide instructions along with the project on how to rebuild it.

WARNING

The precompiled headers are built with the compiler settings of the current project. If you use the precompiled header in a later project with different compiler settings, the header could conflict with the project. In particular, if you change the 8-byte doubles option, you see the different configuration for precompiled header error message. To fix this, just rebuild the precompiled header with the later project’s compiler options.
Customizing MacHeaders++ or TCL

Both MacHeaders++ and TCLHeaders have corresponding source files. For MacHeaders++, the source file is `Mac #includes.cpp`, and it is located in the Mac #includes folder in the THINK tree. Its contents start as follows:

```cpp
/*
 * Mac #includes.cpp - source to MacHeaders++
 *
 * To add a header, change its "#if 0" to "#if 1".
 *
 * To remove a header, change its "#if 1" to "#if 0".
 *
*/

#pragma once

#include <ActionAtomIntf.h>

#include <ADSP.h>

#include <AEObjects.h>
```

You can change which files are included in `Mac #includes.cpp` by changing the `#if 0` to `#if 1` for any header you would like to include. You probably should give the resulting precompiled header a different name than MacHeaders++. If you don’t, your MacHeaders++ will be different from the standard, and you or someone else may find that this creates problems when swapping projects.

The sources for TCLHeaders are found in `TCL #includes.cpp` in the THINK Class Library folder. This file includes commonly used Macintosh headers and most of the TCL header files.
Summary

You now should know all you need to know about precompiled headers. Armed with your newfound knowledge of precompiled header files and the rest of the project environment, you should be ready to tackle C++ in force.
Part 2
The Elements of C++

C++ has derived its wide range of capabilities from its rich heritage. On one hand, it is a direct descendant of C, a language that provides both the structures of a high-level language and the operations that enable low-level machine access. Adding to this base, C++ also has benefited from the object-oriented concepts found in Simula and Smalltalk.

It is to this language that you now turn. First you learn how to use C++ simply as a better C. Next, you get an overview of C++ object-oriented programming and design. After getting a feel for the important elements of this type of programming, you are introduced to the most important C++ construct, the class. You learn how the class construct enables you to combine data and the routines that operate on that data. You then turn to the powerful C++ features of polymorphism and multiple inheritance. Having been rooted in these elements of the language, you learn some rules of thumb to help you write good C++ programs. Finally, the discussion returns to the Symantec C++ environment, where you learn about the THINK Class Browser and the various settings of the C++ compiler.

What this section does not do is teach you every important aspect of C++. Having completed this material, however, you can start writing C++ programs. This section also prepares you for more advanced material on C++ (refer to the bibliography in Appendix E for further reading). You have probably heard that C++ is not an easy language to learn or use well. Although there is a certain grain of truth to these statements, it is not a language you want to pass over. Writing good C++ programs takes practice, but the same is true of most worthwhile endeavors.
many of the changes made to C were done to make it a better language. With C, it is fairly easy to fling random syntactic changes in here and there and still have code that compiles; however, that has no more hope of working than a pig has of singing.

C++, a more disciplined language, allows far less nonsense. A C++ compiler catches many more errors at compile time: these errors are not allowed to become run-time errors. The catch, of course, is that for the compiler to find the errors you have to spend much more time preparing your code for the compiler. It also takes you longer to get a clean compile. Rest assured, this additional effort offers you more than just the smug assurance of being more disciplined than your fellow C programmers. You know that, if your C++ program compiles, it may work and might even do what you wanted it to do.

Why should you read this chapter and implement the changes discussed? Because you can become a better programmer than if you just use C. Furthermore, you can make these changes without committing yourself to becoming a full-fledged C++ programmer, which would require you to become an object-oriented programmer. You learn about object-oriented programming in later chapters. In this chapter, you just learn a little grammar—you don’t change your whole programming style.
Overview

What are the changes made to C++ that make it a better C? The following list highlights some of the improvements included in C++:

- The stream library replaces the stdio routines for standard I/O. This gives you fewer chances for error as well as providing extensibility.
- There is a new and easier comment format /* comments instead of /** */.
- Declarations can be mixed with executable statements.
- Structs no longer need to be declared with the typedef statement.
- Functions can have default arguments.
- C++ provides more rigorous error checking, including special warnings and compile-time type checking.
- Functions can be overloaded. This feature enables two different functions to have the same name as long as their parameter lists are different.
- Memory allocation and deallocation functions are handled by the built-in operators new and delete instead of by the C library routines malloc and free.
- Const variables and inline functions replace the use of #define.
- References can be used to pass argument parameters by address instead of by value.

The following sections take a look at each of these improvements in turn.

Standard Input/Output

The standard C I/O statements changed in C++ for two reasons:

- To help eliminate errors.
- To add user extensibility.

Eliminating Errors in I/O Operations

The problem with standard C I/O operations is that, when you call standard I/O routines, you must specify the type of variable you will be outputting or inputting. The following call encodes the types of the items to be printed with %d and %f:
Enforcing the encoding of variable types when the compiler already knows about them not only begs the programmer to type them in wrong, but is harder to read as well. With C++, the same code can be written this way:

```c
cout << i << x;
```

Here, the compiler, knowing the types of i and x, sees to it that they are output appropriately. This first reason for changing C should be easy enough to appreciate.

The stream library that provides this new method for handling I/O works like this: A stream is created that refers to standard output or input, a file, or a string in memory. The syntax for outputting or inputting is the same, regardless of the source or destination. The syntax is as follows:

```
stream << expression1 << expression2 ... << expressionN
```

The following C++ elements may be new to you:

- `<<` This is an example of an **overloaded operator**. When used with streams, `<<` is called the **insertion operator**. Similarly, `>>` is called the **extraction operator**.
- `cout` This is a stream that is connected to standard output. Similarly, `cin` is a stream connected to standard input.

The following is an example of the syntax in use:

```
cout << "3 * 5 = " << 3 * 5 << '\n';
```

This line outputs the following:

```
3 * 5 = 15
```

Overloading functions and operators are discussed in more depth later in this chapter.

**NOTE**

In C, operators are overloaded as well: You just cannot modify them yourself. For example, when applied to integers, `*` means integer multiplication, and when `*` is applied to floating-point numbers, it means floating-point multiplication—two different operations. In C++, you have access to this overloading capability through your own routines. The `<<` operator is usually the left bit-shift operator, but the stream library overloads `<<` to mean stream output.
Although you can still use the printf/scanf family (stdio), you really should use streams. Unlike streams, the stdio library uses different routines for outputting to standard output, files, and strings (printf, fprintf, and sprintf, respectively). Similarly, input from standard input, files, and strings uses three different routines (scanf, fscanf, and sscanf). Obviously, trading six routines for two simplifies the code and reduces the chances for errors. The bounty of the stream library's benefits should become even clearer by looking at some comparative code examples.

Following is a small integer-producing program written using stdio.h:

```c
#include <stdio.h>

void main()
{
    int i = 3;
    printf("i = %d\n", i);
}
```

Here is the same program using stream.h:

```c
#include <stream.h>

void main()
{
    int i = 3;
    cout << "i = ", << i << '\n';
}
```

Both of these routines output the following:

```
i = 3
```

Using streams, each expression is output in order, and the way it is printed depends on the type of the expression. Look at the following line from the example:

```
cout << "i = ", << i << '\n';
```

The first expression, "i = ", is a string; the second, i, is an integer; and the third, '\n', is a character. Each one prints as expected.

Using stdio routines, if you have an integer, you use %d; if you have a floating-point number, you use %f. This is the source of many errors in programs. For instance, there is nothing in C to prevent you from writing the following:

```c
printf("i = %f\n", i);
```
At run time, the compiler dutifully tries to interpret i as a floating-point number and outputs the following:

\[ i = 0.000000 \]

This is an error you would not enjoy finding in a complex program.

Streams Enable User Extension

As well as having unified syntax, the stream library also enables user extensibility. With stdio, suppose that you have the following structure:

```c
struct MyStruct {
   int i;
   int j;
} s;
```

To print out s, you need to print out each field within it. This is what the stdio code looks like:

```c
printf("%d %d", s.i, s.j);
```

Whereas, after having written a routine that can output a MyStruct to a stream, all you have to write is the following:

```c
cout << s;
```

More Examples of Streams

Now that you have examined why standard I/O has been changed in C++, look at some more code examples.

The Hello World program, which you originally learned in Chapter 1, is a good place to start. The new C++ way of coding this program using streams is as follows:

```c
#include <stream.h>

void main()
{
   cout << "Hello, world!\n";
}
```

The original C version looks like this:

```c
#include <stdio.h>
```
void main()
{
    printf("Hello, world!\n");
}

Following is an example of a program that takes two integers as input:

#include <stream.h>

void main()
{
    int i;
    int j;
    cin >> i >> j;
}

Using stdout, the same program looks like this:

#include <stdio.h>

void main()
{
    int i;
    int j;
    scanf("%d %d", &i, &j);
}

You should agree that the first example, although unfamiliar, appears more natural than the second.

As you have seen, C++ I/O using streams reduces errors and adds extensibility. The age-old principle of Occam's razor demands that you choose the simpler solution.

**The New Form of Comments**

C++ gives you a new syntax for comments: The double slash, //, which can be used instead of /* */. The // characters begin a comment that terminates at the end of the line. Following is an example that uses the new comment style:
Returns the maximum of two integers

```c
int Max(int a, int b)
{
    if (b > a) // could be >= instead
        return b;
    else // if a tie, we return a
        return a;
}
```

If you already have been using THINK C, then you have had // comments available for some time and may have experienced their many benefits. For the rest of you who have had to do without // comments until now, it is easy enough to explain the greatest single advantage of this new form of comment: It is auto-terminating.

**NOTE**

Although // comments are not part of ANSI C, the THINK C compiler allows them anyway. Check in the Language Settings panel of the THINK C Options dialog box to make sure that Language Extensions are on. MPW C also accepts // comments.

A recurring C programming error is to forget to close a comment. This is not always easy to find. Even the best C programmer has had something like the following disaster strike:

```c
/* Returns the maximum of two integers */
int Max(int a, int b)
{
    if (b > a) /* could be >= instead
        return b;
    else /* if a tie, we return a */
        return a;
}
```

Now this is the stuff of which dreams are made—bad ones. This code no longer computes the maximum value correctly because the second comment (could be >= instead) is not terminated. However, no error is generated because the closing */ of the third comment closes the second comment. So the code compiles and runs, but acts as though you wrote the following:

```c
int Max(int a, int b)
{
```
if (b > a)
    return a;
}

This is what is left after removing the comments; the code does not do what it is supposed to do. In conclusion, // comments are preferable to the older C-style comments, if not for ease of use, then for a good night’s sleep.

## Declarations Mixed with Executable Statements

C requires that all variable definitions within a function occur at the beginning of a block. C++ relaxes these rules. Variable definitions can occur anywhere.

### NOTE

C and C++ make a distinction between declarations and definitions. **Declarations** state that a variable exists; **definitions** also reserve space for the variable.

The following example shows a C++ function that includes some variable definitions within the block:

```c
float Foo()
{
    int i;
    float total;
    float f;

    f = sin(1.0) / 5.0;
    total = 0;
    for (i = 0; i < 10; i++) {
        float g;

        g = f * i;
        total = g * g;
    }
}
```
float Foo()
{
    float total;
    float f;

    f = sin(1.0) / 5.0;
    total = 0;

    int i;
    for (i = 0; i < 10; i++) {
        float g;

        g = f * i;
        total = g * g;
    }
    return total;
}

You now have executable statements intermingled with variable definitions. C++ enables you to define this variable even closer to where it is used; you can move the definition of i into the for statement:

float Foo()
{
    float total;
    float f;

    f = sin(1.0) / 5.0;
    total = 0;

    for (int i = 0; i < 10; i++) {
        float g;

        g = f * i;
        total = g * g;
    }
    return total;
}
Defining the loop variable within the `for` statement is a common C++ idiom. Though useful, this can be quirky. If you are not careful, you may make mistakes using it until you master one particular idiosyncrasy. The following rule explains the problem.

**RULE**

The variable scope extends from the point where the variable definition occurs to the end of the block in which the variable is defined.

Put another way, the scope of a variable defined within a `for` statement extends from that `for` statement to the end of the block in which the `for` statement occurs, not to the end of the `for` loop. It is an error to write the following code because `i` already is defined when you reach the second `for` loop:

```cpp
for (int i = 0; i < 10; i++) {
    Foo();
}
for (int i = 0; i < 10; i++) {
    // error, i already defined
    Bar();
}
```

Figure 8.1 shows a diagram that should clarify the relationships.

---

**Figure 8.1.** The scope of variables.
NOTE

As you might imagine, forgetting the variable's scope is a common source of errors. There are two reasons why variables in a for loop are scoped as they are. First, many C programmers occasionally use the value of a for loop variable after the for loop to determine whether the loop finished iterating. Second, it is a mistake. As Ellis and Stroustrup say "it would probably have been better to introduce a special rule to limit the scope of a name introduced in the initializing statement of a for statement to the for statement, but now much code exists that depends on the general rule."

If you remember this rule of variable scope you should be fine. This problem is far outweighed by C++'s more flexible approach to variable declaration.

Typedefing and Structs

In C, it is quite common to use typedef when declaring structures in the following fashion:

typedef struct MyStruct {
    ...
} MyStruct;

In doing so, you can use the structure named MyStruct either with or without the struct keyword:

struct MyStruct s;

You also can write:

MyStruct s;

C requires the typedef in the declaration of MyStruct because there is a different name space for structures and types. The previous example is the way around this: The typedef causes the structure name also to be defined as a type name.

In C++, there is only one name space, and declaring a structure automatically declares a type with that name. This removes the dubious possibility that you have in C, in which you can cause struct Foo and Foo to be two totally different types. You can, therefore, write the following:

struct MyStruct {
    ...
};
You then can use:

```c
struct MyStruct s;
```

Or, you can use:

```c
MyStruct s;
```

C++ still accepts the `typedef` to define a type name equivalent to a corresponding struct name. If you forget and still use the `typedef`, you are no worse off.

### Default Arguments

C++ enables the use of default arguments, so that the caller of a function need not pass all parameters. Any unspecified parameters use a default value specified in the function declaration. Here is an example of a function that has default arguments:

```c
extern void *Alloc(int numBytes, Boolean useMFMem = false);
```

Following are the rules for default arguments:

- You can call these functions with a varied number of arguments. Using the previous declaration, the `Alloc` function can be called with two arguments. This first is

```c
ptr = Alloc(1024, false);
```

Or, it can be called with:

```c
ptr = Alloc(1024, true);
```

The function also can be called with just one argument:

```c
ptr = Alloc(1024);
```

In the last case, the second parameter, `useMFMem`, defaults to false.

- Only trailing arguments can have defaults, so the following is incorrect:

```c
extern void *Alloc(Boolean useMFMem = false, int numBytes);
// illegal
```

- Functions may have multiple default arguments, but they all must be together at the end:

```c
extern void *NewAlloc(int numBytes, Boolean useMFMem =
```
false, Boolean clearMem = false);

The following code is incorrect:

extern void *NewAlloc(int numBytes = 5, Boolean useMFMem,

    Boolean clearMem = false); // illegal

You can use defaults only in the function declaration (usually an .h file) and not in the definition.

Error Checking

In general, C++ compilers provide stricter error checking than C compilers. As well as finding more errors, they also usually issue more warnings. Anyone who has used C long enough should be glad to have a little more discipline in his or her life. The following sections discuss some of the important ways in which the Symantec C++ compiler provides you with this additional error information.

Warnings

Historically, THINK C had no concept of warnings. There was no facility for the compiler to notify the programmer of a possible problem while still continuing to compile. The THINK Project Manager’s new Compile Errors window provides a place for Symantec C++ to output warnings as well as errors.

One such warning is used before set warning. If the global optimizer is turned on, Symantec C++ warns you if you are trying to use variables you have not set. Try to compile the following function:

```c
void Warning(int parameter)
{
    int localVariable;

    parameter = localVariable;
}
```

The compiler generates the following warning:

File "Warnings.c"; Line 8
Warning: in function Warning(int) - variable 'localVariable' used before set

These are not the sorts of warnings you should ignore. They point out logic errors in your code that you should fix.
Note, however, that these warnings are available only when the global optimizer is on. The optimizer’s job, among other things, is to compute data-flow information about the lifetime of each variable and its initialization times. Because the optimizer is already gathering this information, the warnings come for free (well almost free—warnings can cause your compilation to be slightly slower).

**Compile-Time Type Checking**

The Symantec C++ compiler also performs more rigorous type checking than the C compiler. You can view this as good or bad, depending on how lazy you are feeling.

In THINK C, using function prototypes and declaring all functions before you use them is optional. This is good because it is less work and appeals to the lazy side of you. Unless you are a very careful programmer, however, you are likely to end up with many errors.

C++ is much stricter. It requires you to dot your i’s and cross your t’s; you must use prototypes and declare your functions before use. Symantec C++ then checks function calls against their declarations to make sure that you are calling routines with the correct number and types of parameters.

**Type-Safe Linkage**

The C++ compiler also gives you an error if the types and numbers of parameters used for the function call don’t match those of the function. Consider the code in table 8.1.

**Table 8.1. Code Example for Type-Safe Linkage**

<table>
<thead>
<tr>
<th>File a.c</th>
<th>File d.c</th>
</tr>
</thead>
<tbody>
<tr>
<td>void Bar()</td>
<td>void Foo(float f)</td>
</tr>
<tr>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td>extern void Foo(int);</td>
<td>//_</td>
</tr>
<tr>
<td>Foo(1);</td>
<td>}</td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

In C, as in C++, both a.c and d.c compile and link correctly. Unfortunately, in C, at run time, the routine Bar calls Foo, passing the integer 1. The routine Foo tries to read this as a floating-point number. Not a pretty sight, is it? At best, the code works incorrectly; at worst, it crashes.
C++ catches this error for you during linking. The compiler finds that the call to Foo does not match the definition of Foo. It reports this as a link error:

`undefined: Foo(int)`

This signifies that there is a call to a routine Foo that takes one `int` parameter, but that no such routine can be found.

Interestingly enough, this error is reported less often in Symantec C++ than in other C++ implementations. Other implementations require makefiles, which specify the dependencies of various files. With makefiles, it is easy to get the dependencies wrong (as any MPW user can tell you). Symantec C++'s Project Manager does this work for you. It keeps track of when files need to be recompiled. There are, however, two cases in which these types of problems can occur in Symantec C++:

- You declare a function `extern` in one source file and provide the function definition in another source file (as was done in the previous example to show this error). You really should declare your functions in a header (an `.h` file) file. This file is included by all source files that call the function, as well as by the file containing the function definition.

- You are using precompiled headers. Unfortunately, the Symantec C++ Project Manager does not rebuild precompiled header files automatically. So, as you can imagine, it is possible to get into a situation where you are using an old version of a precompiled header.

As you have found, Symantec C++ notifies you if your call and function definition don't match in number or type of arguments. The worst that happens with these two cases is that the compiler whines at you to fix the code—much less drastic than causing a run-time error.

### Overloaded Functions

In C, all functions in the same scope must have unique names. This can make life difficult for a programmer on a large project battling to give appropriate, short, easy-to-type names that are unique. It is even more difficult when a programmer combines two libraries, each of which has used the same name for a function. One common way programmers avoid duplication is to encode the type of arguments as part of the function name. This is the approach taken by Apple programmers for one of their functions. The following standard C function returns the square root of a floating-point number:

```c
double sqrt(double);
```
Because the Apple programmers wanted a square root function that operates on Fract numbers (a type that holds fixed-point numbers in the range of -2 to 2), they came up with the following:

\[
\text{Fract FracSqrt(Fract);} \\
\]

You can see how a unique name limitation can quickly lead to a proliferation of slightly differently named functions:

\[
\text{Fract FracSqrt(Fract);} \\
\text{int IntSqrt(int);} \\
\text{float FloatSqrt(float);} \\
\text{long LongSqrt(long);} \\
\]

Ideally, you would like to name them all \text{sqrt}. C++ grants this wish. Functions can still have the same name even though the types of the arguments are different. This is called \textit{overloading} a function. Here are some examples:

\[
\text{extern double sqrt(double);} \\
\text{extern Fract sqrt(Fract);} \\
\text{extern int sqrt(int);} \\
\]

double \textit{f};
\[
\ldots \\
\textit{f} = \text{sqrt(\textit{f})}; \quad \text{// calls sqrt(double)} \\
\]

\[
\text{Fract \textit{l};} \\
\ldots \\
\textit{l} = \text{sqrt(\textit{l})}; \quad \text{// calls sqrt(Fract)} \\
\]

\[
\text{int \textit{i};} \\
\ldots \\
\textit{i} = \text{sqrt(\textit{i})}; \quad \text{// calls sqrt(int)} \\
\]

It is important to realize that the overloading is based on the type and number of arguments and not on the return type. Look at the following:

\[
\text{int sqrt(int);} \\
\text{float sqrt(int);} \quad \text{// illegal} \\
\]

Although the code appears to be a declaration of two overloaded functions, it actually is an error; the compiler flags it as such.

C++ treats \texttt{char}, \texttt{short}, \texttt{int}, \texttt{long}, \texttt{float}, and \texttt{double} as distinct types when used as parameters. This enables functions to be overloaded based on these distinctions (and on whether the type is unsigned or not).
The following code declares eight overloaded functions, each of which is named `PrintValue`. C++, using its own internal wisdom, determines which function to call based on the type of the argument to `PrintValue`.

```c
void PrintValue(char c);
void PrintValue(short s);
void PrintValue(int i);
void PrintValue(long l);
void PrintValue(unsigned char c);
void PrintValue(unsigned short s);
void PrintValue(unsigned int i);
void PrintValue(unsigned long l);
```

Overloaded functions are a real boon to programmers on large projects. Not only are programmers liberated from distinct naming, but code errors are reduced. The C++ compiler figures out which function to call based on the arguments you give; you can relax and cheerfully overload functions to your heart's content.

### Memory Allocation/Deallocation

In C, you allocate and deallocate memory by using the standard library. The library is not built into the language. C++, on the other hand, builds memory management into the language with the `new` and `delete` operators. Table 8.2 shows an example of `new` and `delete` in comparison with C's use of `malloc` and `free`:

<table>
<thead>
<tr>
<th>C</th>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>int *intPtr;</td>
<td>int *intPtr;</td>
</tr>
<tr>
<td>intPtr = (int *) malloc(sizeof(int));</td>
<td>intPtr = new int;</td>
</tr>
<tr>
<td>free(intPtr);</td>
<td>delete intPtr;</td>
</tr>
</tbody>
</table>

From the table you should be able to see that using `new` has two advantages:

- ▼ You don’t need to typecast the return result; it is already the right type.
- ▼ You don’t need to calculate the size with `sizeof`—`new` automatically calculates the correct size.
This is an example of using malloc incorrectly:

```c
int *intPointer;

intPointer= (int *) malloc(sizeof(char)) // error
*intPointer= 0;
```

The code accidentally allocates only enough space for a char. The assignment statement may very well write past the end of the allocated memory, causing hard-to-diagnose run-time errors. Using new, there is no possibility for error because the compiler, rather than the programmer, computes the size needed:

```c
int *intPointer;

intPointer = new int;
*intPointer = 0;
```

So, although you might be exceptionally fond of malloc and free, you should still switch to new and delete if for no other reason than to avoid these kinds of mistakes.

Following are some rules for using new and delete:

- You can specify any type in the new statement:
  ```c
  intPtr = new int;
  
  floatPtr = new float;
  
  myStructPtr = new MyStruct;
  ```

- If you want to allocate an array, use the new [] syntax:
  ```c
  intPtr = new int[10];
  ```

  This code allocates a pointer to 10 integers. If you allocate an array with new [], you delete it with a slightly different syntax using delete [], as follows:

  ```c
  delete [] intPtr;
  ```

- Any time you allocate memory with new [], you should deallocate it with delete[].
  The [] in the delete statement specifies that the pointer has been allocated with the [] syntax.

- The delete operator specifically enables deleting a NULL pointer. You can avoid checks like the following:
if (intptr != NULL)
    delete intptr;

Use the following replacement code:
    delete intptr;

Although you may continue to use malloc and free, you should switch to using new and delete. If you use both, be extremely careful to deallocate memory with the correct routine. If you allocate with malloc, deallocate with free. If you allocate with new, deallocate with delete.

Eliminating \#define

The availability of a macro preprocessor (\#define) is part of what makes C more usable than other languages like Pascal. So why, you ask, get rid of it? The answer lies in the two main uses of \#define:

- Defining named constants, such as:

  \#define kTableSize 512

- Enabling functions to be evaluated inline, rather than requiring a function call:

  \#define Round(myFloat) ((int) ((myFloat) + 0.5)

C++ offers replacements for both. It provides const variables to define named constants:

const int kArraySize = 512;

C++ also replaces C’s pseudo inline functions with real ones:

inline int Round(float myFloat)
{
    return (int) (myFloat + 0.5);
}

Const Variables

Those of you familiar with THINK C (and ANSI C) already know something about const variables. The THINK C compiler does not take full advantage of them, however. It doesn’t allow their use where constant expressions are needed, such as when you are defining an array. Const variable use is relegated mostly to function prototypes, rather than as a general replacement for \#define.
Following is the syntax for `const` variables:

```c
const type name = value;
```

For comparative purposes look at a program written in C and C++. Here is the C version:

```c
#define kArraySize 512

void main()
{
    int myArray[kArraySize];

    myArray[0] = 1;
}
```

Although this program compiles using both C and C++, the preferred C++ way to write it is slightly different:

```c
const int kArraySize = 512;

void main()
{
    int myArray[kArraySize];

    myArray[0] = 1;
}
```

The `const` keyword signifies that the value of `kArraySize` cannot change throughout the program.

Why is this use of `const` better than `#define`? In C++, it is the compiler, and not the preprocessor, that replaces the uses of `kArraySize` with 512. This is favorable for the following four reasons:

- The compiler prevents redeclaration of the variable. For instance, many C compilers allow the following:

  ```c
  #define kArraySize 512
  ...
  
  #define kArraySize 256
  ```

  Unlike other programmers, THINK C users are fortunate in this regard because THINK C does not allow a redeclaration of a `#define`. 

- [ ]

- [ ]

- [ ]

- [ ]
The variable is not substituted literally; it stays a symbol. This is enormously helpful when debugging because the compiler retains a meaningful name that shows up in error messages. If you have ever tried to decipher a program that points to a value like 512, you know that kArraySize is a much more useful name to see.

You are not limited to built-in types with const; you are free to use your own. For example, here is how to declare a constant Rect:

```c
const Rect kEmptyRect = {0, 0, 0, 0};
```

Const uses the same syntax as other variables. Although diversity is nice, consistency makes for easier maintenance and fewer errors.

Clearly, const variables are preferable to #define. You cannot inadvertently redeclare them, they are easier to use in debugging, and you can have types that are not built in and that have a consistent syntax.

### Inline Functions

C programmers also use #define to reduce the overhead of frequent function calls. C++ addresses this problem by creating a new type of function, the inline function. By declaring a function inline, you request that the compiler place the code of the function directly in line rather than making a function call. Here is a typical C example of #define that accomplishes this purpose:

```c
#define Round(myFloat) ((int) ((myFloat) + 0.5))
```

The same goal is far easier to achieve using an inline function in C++:

```c
inline int Round(float f)
{
    return (int) (myFloat + 0.5);
}
```

Inlining a function can have dramatic effects on the speed of your program. (Which, of course, is why #define was used for this purpose in the first place). To see how inlining can speed up the process, first look at a sample program without inlining:

```c
#include <iostream.h>
#include <Events.h>

static int Value(int i)
{
    if (i < 10)
        return 2;

```
else if (i < 1000)
    return 3;
else
    return 4;
}

void main()
{
    long startTime = TickCount();
    int total = 0;
    int i;

    for (i = 0; i < 1000000; i++)
        total += Value(i);

    long endTime = TickCount();
    cout << "total = " << total << 'n' << "time = " << endTime - startTime;
}

When run, this program outputs the following:

total = 3998990
time = 85

This is what the object code looks like:

Value(int):

<table>
<thead>
<tr>
<th>Address</th>
<th>Operation</th>
<th>Offset</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>VALUE</td>
<td>0000</td>
<td>A6,$0000</td>
</tr>
<tr>
<td>00000004</td>
<td>CMPIL</td>
<td>000A</td>
<td>#$0000000A,$0008(A6)</td>
</tr>
<tr>
<td>00000008</td>
<td>LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000000C</td>
<td>BGE.S</td>
<td>0006</td>
<td>+$0006</td>
</tr>
<tr>
<td>0000000E</td>
<td>MOVEQ</td>
<td>$02,D0</td>
<td></td>
</tr>
<tr>
<td>00000010</td>
<td>BRA.S</td>
<td>+$0012</td>
<td></td>
</tr>
<tr>
<td>00000012</td>
<td>CMPIL</td>
<td>#$003E8,$0008(A6)</td>
<td></td>
</tr>
<tr>
<td>00000014</td>
<td>MOVEQ</td>
<td>$03,D0</td>
<td></td>
</tr>
<tr>
<td>00000016</td>
<td>BRA.S</td>
<td>+$0004</td>
<td></td>
</tr>
<tr>
<td>00000018</td>
<td>MOVEQ</td>
<td>$04,D0</td>
<td></td>
</tr>
<tr>
<td>00000020</td>
<td>UNLK</td>
<td>A6</td>
<td></td>
</tr>
<tr>
<td>00000022</td>
<td>MOVEA.L</td>
<td>(A7)+,A0</td>
<td></td>
</tr>
<tr>
<td>00000024</td>
<td>ADDQ.W</td>
<td>#$4,A7</td>
<td></td>
</tr>
<tr>
<td>00000026</td>
<td>JMP</td>
<td>(A0)</td>
<td></td>
</tr>
<tr>
<td>00000028</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
main:
00000000: 4E56 FFF8       LINK    A6,$FF8
00000004: 48E7 1E30      MOVEM.L D3-D6/A2/A3,-(A7)
00000008: 594F           SUBQ.W #$4,A7
0000000A: A975           _TickCount
0000000C: 201F           MOVE.L (A7)+,D0
0000000E: 2600           MOVE.L D0,D3
00000010: 7000           MOVEQ #$00,D0
00000012: 2800           MOVE.L D0,D4
00000014: 7000           MOVEQ #$00,D0
00000016: 2A00           MOVE.L D0,D5
00000018: 0CB5 000F 4240  CMP.L #$00F4240,D5
0000001E: 6C0C           BGE.S  *#$00E ; 0000002C
00000020: 2F05           MOVE.L D5,-(A7)
00000022: 4EBA 0000      JSR    Value(int)
00000026: D880           ADD.L D0,D4
00000028: 5285           ADDQ.L #$1,D5
0000002A: 60EC           BRA.S *-$0012 00000018
0000002C: 594F           SUBQ.W #$4,A7
0000002E: A975           _TickCount
00000030: 201F           MOVE.L (A7)+,D0
00000032: 2C00           MOVE.L D0,D6
00000034: 486D 0000      PEA    $0000(A5)
00000038: 720A           MOVEQ #$0A,D1
0000003A: 1F01           MOVE.B D1,-(A7)
0000003C: 486D 0000      PEA    $0000(A5)
00000040: 486D 0000      PEA    cout
00000044: 4EBA 0000      JSR    ostream::operator <<(const char *)
00000048: 2440           MOVEA.L D0,A2
0000004A: 2D44 FFF8      MOVE.L D4,$FF8(A6)
0000004E: 2F00           MOVE.L D0,-(A7)
00000050: 486E FFF8      PEA    $FF8(A6)
00000054: 7004           MOVEQ #$04,D0
00000056: 2F00           MOVE.L D0,-(A7)
00000058: 7001           MOVEQ #$01,D0
0000005A: 2F00           MOVE.L D0,-(A7)
0000005C: 4EBA 0000      JSR    _2Comp::insert(ostream&,const void *,int,int)
00000060: 2F00           MOVE.L D0,-(A7)
00000062: 4EBA 0000      JSR    ostream::operator <<(char)
00000066: 2F00           MOVE.L D0,-(A7)
00000068: 4EBA 0000      JSR    ostream::operator <<(const char *)

Chapter 8: C++: An Enhanced C
This program makes a time-consuming function call for each iteration of the loop. Inlining was created to fix just this problem. Watch what happens in the same program when Value is inlined:

```cpp
#include <iostream.h>
#include <Events.h>

static inline int Value(int i)
{
    if (i < 10)
        return 2;
    else if (i < 1000)
        return 3;
    else
        return 4;
}

void main()
{
    long startTime = TickCount();
    int total = 0;
    int i;

    for (i = 0; i < 1000000; i++)
        total += Value(i);

    long endTime = TickCount();
}
cout << "total = " << total << 'n' << "time = " <<
   endTime - startTime;
}

This time, when you run the program, it outputs the following:

total = 398990
time = 50

In this example, the object code looks like the following:

main:
00000000: 4E56 FFF8          LINK   A6,$FFFF8
00000004: 48E7 1F30          MOVEM.L D3-D7/A2/A3,-(A7)
00000008: 594F             SUBQ.W #$4,A7
0000000A: A975          _TickCount
0000000C: 201F          MOVE.L (A7)+,D0
0000000E: 2600          MOVE.L D0,D3
00000010: 7000          MOVEQ #$00,D0
00000012: 2800          MOVE.L D0,D4
00000014: 7000          MOVEQ #$00,D0
00000016: 2A00          MOVE.L D0,D5
00000018: 0CB5 000F 4240    CMP.I.L #$000F4240,D5
0000001E: 6C22          BGE.S  *+$0024                  ; 00000042
00000020: 2005          MOVE.L D5,D0
00000022: 2E00          MOVE.L D0,D7
00000024: 720A          MOVEQ #$0A,D1
00000026: B081          CMP.L D1,D0
00000028: 6C04          BGE.S  *+$0006                  ; 0000002E
0000002A: 7202          MOVEQ #$02,D1
0000002C: 600E          BRA.S  *+$0010                  ; 0000003C
0000002E: 0CB0 0000 03E8    CMP.I.L #$000003E8,D0
00000034: 6C04          BGE.S  *+$0006                  ; 0000003A
00000036: 7203          MOVEQ #$03,D1
00000038: 6002          BRA.S  *+$0004                  ; 0000003C
0000003A: 7204          MOVEQ #$04,D1
0000003C: D881          ADD.L D1,D4
0000003E: 5285          ADDQ.L #$1,D5
00000040: 60D6          BRA.S  *-$0028                  ; 00000018
00000042: 594F          SUBQ.W #$4,A7
00000044: A975          _TickCount
00000046: 201F          MOVE.L (A7)+,D0
00000048: 2C00          MOVE.L D0,D6
By adding this one word (inline) to your program, you cut the running time by over 40 percent. Admittedly, this is an unusual example. It does point out, however, that inlining functions can have dramatic performance effects on a program.
Before you get too attached to inline functions, you need to bear in mind that they are not a panacea. In most cases, the inline keyword should be added to your program only after it is running correctly. You should only use inline functions on portions of your program that consume the most time.

A well-known computing rule of thumb, the 80-20 rule, states that 80 percent of your time is spent on 20 percent of your code. Or, if you are an optimist you can use the 90-10 version of this rule. In any event, this means that you should focus your speed-up efforts on the 20 percent of your code that consumes 80 percent of your performance time (the hot spots). Trying to optimize all of your code is best left to individuals who never have to ship a product. In summary, before attempting to speed up your program, determine where your program spends its time and focus your efforts there.

**NOTE**

To find where your program spends your time, use a profiler. Symantec C++ provides profiling tools. See Chapters 9 and 12 for more information on the subject of efficient code.

Now that you have learned *how* to inline, the following section examines *when* to inline.

**Small Functions and Inlining**

The first rule of inlining is, “The smaller the function, the better a candidate it is for inlining.” Unfortunately, inlining a function can actually slow things down instead of speeding them up. Use the following rule to determine whether inlining will help.

**RULE**

If the code emitted for an inline function is smaller than the code necessary to call it, then inlining the function is definitely better.

Making a function inline can increase the size of the code that calls it. This is especially true in extreme cases, such as when you inline a large function that is called many times in your program. Then, you end up with increased segment loading and unloading due to the larger segments. So, instead of getting speed, you actually have injected your program with a dose of molasses.
If you are unclear whether inlined functions are beneficial to your code, you always can measure the speed of your program after making the supposed optimization to make sure that your alleged speed-up is not in reality a slow-down.

**Functions that Cannot Be Inlined**

Some functions cannot be inlined. Different compilers have different rules about which functions can and cannot be inlined. You should realize that, like the register declaration, the inl inle declaration is a suggestion (or request) to the compiler. The compiler may not always inline a function, but usually does so if you request it.

Following are some examples of functions that are not inlined by Symantec C++:

- **Recursive functions** (actually, the first iteration is inlined; the remainder are not):

  ```cpp
  inline unsigned int Factorial(unsigned int i)
  {
    if (i <= 1)
      return 1;
    else
      return i * Factorial(i - 1);
  }
  ```

- **Functions with loops**:

  ```cpp
  inline unsigned int Length(char *s)
  {
    unsigned int total;
    while (*s++ != '\0')
      total++;
    return total;
  }
  ```

- **Functions with switch statements**:

  ```cpp
  inline unsigned int FunctionWithSwitch(int i)
  {
    switch (i) {
    case 1:
      return 1;
    case 2:
      return 2;
    ```
Some C++ compilers warn you if they will not honor your inline request. Unfortunately, Symantec C++ does not provide a warning. You have to disassemble the object code to tell for sure.

**Inlining Avoids #define Problems**

There are a number of problems with `#define` that you can avoid by using inline functions. Some of them can be quite nasty:

▼ Arguments to a macro may be evaluated an incorrect number of times. Contrast this with inline functions whose arguments are evaluated exactly once. Multiple argument evaluations can be frightening to contemplate. For instance, look at this seemingly innocent macro definition:

```c
#define Square(x) ((x)*(x))
```

After the call `Square(i++)`, `i` is incremented twice because the call expands to the following:

```c
(i++)*(i++)
```

This is definitely not what you want `Square(i++)` to do. With inlining, on the other hand, you can write the following code:

```c
inline int Square(int x)
{
    return x*x;
}
```

The call to `Square(i++)` correctly increments `i` exactly once.

▼ Name conflicts occur with any temporary variables. For example, a common way to swap two variables is to use a temporary variable (however, it is possible to do so without one). Here is a macro that swaps two integers:

```c
#define Swap(ptr1, ptr2) \
{ \
```

```c
```
```c
int tmp; \
    tmp = *(ptr1); \
    *(ptr1) = *(ptr2); \
    *(ptr2) = tmp; \
}
```

Under normal circumstances this macro works fine:

```c
int x, y; 
Swap(&x, &y);
```

But, look at what happens when you call this same Swap routine with a variable named tmp:

```c
Swap(&tmp, &y);
```

The routine becomes the following:

```c
{
    int tmp;
    tmp = *(&tmp);
    *(&tmp) = *(&y);
    *(&y) = tmp;
}
```

It swaps the value of y with the new local (uninitialized) variable tmp. Using an inline function, on the other hand, avoids this problem:

```c
inline void Swap(int *ptr1, int *ptr2)
{
    int tmp;
    
    tmp = *ptr1;
    *ptr1 = *ptr2
    *ptr2 = tmp;
}
```

This function works neatly for both examples because the compiler renames variables as necessary within inline functions to avoid conflict. Let the compiler do the work for you.
There is an algorithm for swapping two variables without a temporary. For the uninitiated, it consists of three exclusive OR operations. For example, to swap two variables, \( x \) and \( y \), use the following:

\[
\begin{align*}
  x &= x \oplus y; \\
  y &= x \oplus y; \\
  x &= x \oplus y;
\end{align*}
\]

**Precedence Problems**

Suppose that you have the following \#define:

\[
\text{#define Square}(x) \quad x \times x
\]

Then the expression 1.0/Square(2.0) evaluates to 1.0 rather than the 0.25 you may expect. The reason for this error can be seen by examining the expression after the preprocessor is done:

\[
1.0 \div 2.0 \times 2.0
\]

Because / and * are of equal precedence and are left-associative, the expression is evaluated as follows:

\[
(1.0 \div 2.0) \times 2.0
\]

The expression becomes 0.5 \times 2.0, which equals 1.0.

The correct way to write Square using a \#define macro is to use parentheses:

\[
\text{#define Square}(x) \quad ((x) \times (x))
\]

However, the best way is, again, to use an inline function:

```c
inline float Square(float x)
{
  return x \times x;
}
```

**Conclusions about Const Variables and Inline Functions**

In summary, C++ provides two mechanisms, const variables and inline functions, that reduce or completely eliminate the need for \#define macros. const variables are superior because you can use them with your own data as well as for built-in types. Furthermore, const variables
cannot be redeclared and they are more helpful in debugging. Likewise, you should use inline functions over `#define` because they provide more control over your code. Their arguments cannot be evaluated in a multiple fashion, and any naming conflicts are handled by the compiler. Lastly, inline functions can give you better program performance if they are used correctly.

**References**

If you took a compiler construction class, you probably remember learning about the three different ways parameters are passed. Table 8.3 shows the different parameter-passing methods.

<table>
<thead>
<tr>
<th>Parameter-Passing Method</th>
<th>What Is Passed</th>
</tr>
</thead>
<tbody>
<tr>
<td>call by value</td>
<td>The value of the variable</td>
</tr>
<tr>
<td>call by reference (call by address)</td>
<td>The address of the variable</td>
</tr>
<tr>
<td>call by name</td>
<td>The function acts as if the name of the variable had been replaced textually in the function</td>
</tr>
</tbody>
</table>

You may remember quizzes in school with complicated functions passing different parameters in which you had to figure out what would be printed. In C, all parameters are passed by value (with the following exception).

**NOTE**

Well, actually not all parameters are passed by value in C. Arrays are passed by address because the value of an array name is the address of its first element. Structures are passed by value in modern C compilers and have been for quite some time. An historical tidbit is that behind the structure value/address problem is the first edition of Kernighan and Ritchie, in which they stated that structures are passed by address. Soon afterward, some changes were made to C (including enums and passing structures by value). These changes were implemented in UNIX C compilers, but not in many PC C compilers, which were based on the Kernighan and Ritchie version of the language. Thankfully, ANSI standardization has removed the Tower of Babel. The ANSI standard is that structures are passed by value.
If, for example, you need a procedure called Increment, which increments its argument, you write it as follows:

```c
void Increment(int *p)
{
    (*p)++;
}
```

You call the procedure with the following:

```c
Increment(&myinteger);
```

C++ offers more flexibility; you can pass parameters one of two ways:

- By value, just like C.
- By address, using references.

Using the second method, the Increment procedure looks like this in C++:

```c
void Increment(int &p)
{
    p++;  // Using reference
}
```

You can call this procedure with the following:

```c
Increment(myInteger);
```

**NOTE**

Many FORTRAN compilers use call by reference, even when passing constants. One trick used by FORTRAN programmers is to write a routine `SetToPi` that assigns 3.14159 to its parameter. They sneakily call `SetToPi(3)` early in their program, which changes the value of the constant 3 to 3.14159. Having done this, they then can use 3 anywhere they need the value of \( \pi \) in their program.

**How References Work**

Because it is specific to C++, this book focuses on the second method of passing parameters. A fuller definition of reference is stated in the following rule.
A reference in C++ is a variable that refers to (or is another name for) some other variable.

In the previous example, the parameter `p` in Increment refers to `myInteger`. In both the explicit pointer case (the C example) and the reference case (the C++ example), the actual value passed to the Increment procedure is the address of `myInteger`. The differences between the two are

- How the parameter is passed. With references, no `&` is required (in fact, the `&` is forbidden).
- How the parameter is used in the procedure. With references, you don’t use `*` to dereference the parameter.

The underlying code that is generated is the same in either case, however. The compiler silently adds an `&` before reference parameters in the call to the routine and silently adds an `*` to dereference each use of a reference within the routine.

**References and the Const Keyword**

The `const` keyword works with references in the same way it works with explicit pointer parameters. The combination of `const` and references is especially useful with large structures; you get the simpler syntax of passing by value, while having the efficiency of passing by address.

Table 8.4 shows the possible combinations of references and `const`. From these examples, you should get a better idea of how `const` interacts with references and the differences between references and pointers. You also should note that with references you don’t need to explicitly check for null values.

**Table 8.4. Reference and Const Parameters**

<table>
<thead>
<tr>
<th>Const</th>
<th>Reference Parameter</th>
<th>Pointer Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td><code>void Routine(int &amp;i)</code></td>
<td><code>void Routine(int *ip)</code></td>
</tr>
<tr>
<td></td>
<td><code>{</code></td>
<td><code>{</code></td>
</tr>
<tr>
<td></td>
<td><code>i = 5;</code></td>
<td><code>if (ip != NULL)</code></td>
</tr>
</tbody>
</table>

**Symantec C++ Programming for the Macintosh, 2nd Edition**
<table>
<thead>
<tr>
<th>Const</th>
<th>Reference Parameter</th>
<th>Pointer Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>*ip = 5;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td>Call with:</td>
<td>Routine(myVar); The routine can alter the value</td>
<td>Call with:</td>
</tr>
<tr>
<td></td>
<td>of the reference parameter. The parameter can</td>
<td>Routine(&amp;myVar); or Routine(NULL); The routine can alter the value</td>
</tr>
<tr>
<td></td>
<td>not be NULL (0).</td>
<td>to which the parameter points. The parameter can be NULL (0).</td>
</tr>
<tr>
<td>Yes</td>
<td>void Routine(const int &amp;i)</td>
<td>void Routine(const int *ip)</td>
</tr>
<tr>
<td></td>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>// *i = 5; not allowed</td>
<td>if (ip != NULL) {</td>
</tr>
<tr>
<td></td>
<td>gTotal += *i; // allowed</td>
<td>// *ip = 5; not</td>
</tr>
<tr>
<td></td>
<td>}</td>
<td>allowed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gTotal += *ip;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td>Call with:</td>
<td>Routine(myVar); The routine cannot alter the</td>
<td>Call with:</td>
</tr>
<tr>
<td></td>
<td>value of the reference parameter. The parameter</td>
<td>Routine(&amp;myVar); or Routine(NULL); The routine cannot alter the value</td>
</tr>
<tr>
<td></td>
<td>cannot not be NULL (0).</td>
<td>it points to. The parameter can be</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NULL (0).</td>
</tr>
</tbody>
</table>

This table shows how the const & syntax works. The combination looks like a call by value because the caller does not need to put an explicit & in the code. However, this syntax gives you the efficiency of a call by address because only the address is passed.

**Some Rules of Thumb for Reference Use**

To a certain extent you need to develop your own style of reference use. Until then, the rules in this section offer a nice approach.
**RULE**

If the value of the variable can change within a routine, use a non-const pointer:

```cpp
void AdjustTheRectangle(Rect *r, int amount)
{
    r->top += amount;
    r->bottom += amount;
}
```

**RULE**

If the parameter could be NULL, use a const pointer:

```cpp
// prints point as (v, h)
void PrintThePointIfItExists(const Point *pointPtr)
{
    if (pointPtr != NULL) {
        cout << "(" << pointPtr->v << ", " << pointPtr->h << ");
    }
}
```

**RULE**

For a small variable (one whose size is less than or equal to the size of a pointer, 4 bytes), use neither pointers nor references:

```cpp
// sizeof(Point) == 4
// prints point as (v, h)
void PrintThePoint(Point p)
{
    cout << "(" << p.v << ", " << p.h << ");
}
```
In all other cases, use a const reference:

```cpp
// sizeof(Rect) == 8
// prints rectangle as (top, left)/(bottom, right)
void PrintTheRectangle(const Rect &r)
{
    cout << "(" << r.top << ", " << r.left << ")/(
    r.bottom << ", " << r.right << ")";
}
```

Adopting the style advocated in these rules has one big advantage when you are reading the code: When looking at the call to a function, only parameters preceded by an & can change, and if a parameter is preceded by an &, it probably will change.

To further illustrate the value of these rules, look at the code examples in table 8.5. This table shows calls to two routines: SetRectangle (which modifies the passed-in rectangle) and FrameRectangle (which does not modify the passed-in rectangle). It shows them in normal C, Pascal, and the new C++ style.

**Table 8.5. Three Ways of Calling Toolbox Routines**

<table>
<thead>
<tr>
<th>Normal C</th>
<th>Pascal</th>
<th>New C++ Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>FrameRectangle(&amp;r);</td>
<td>FrameRectangle(r);</td>
<td>FrameRectangle(r);</td>
</tr>
<tr>
<td>SetRectangle(&amp;r, 1, 2, 3, 4);</td>
<td>SetRectangle(r, 1, 2, 3, 4);</td>
<td>SetRectangle(&amp;r, 1, 2, 3, 4);</td>
</tr>
</tbody>
</table>

Using normal C, the calls to SetRectangle and FrameRectangle look the same; both use &. The problem is, that without looking at the declaration (FrameRectangle is declared to take a const Rect *), you cannot tell whether each call modifies r. In the Pascal case, the problem is the same. Without looking at the declaration (SetRectangle is declared to take a var Rect parameter), you cannot tell whether each call modifies r. In the new C++ style, it is clear that SetRectangle modifies r because the & is present and that FrameRectangle does not because the & is absent.
Other Uses of References

Supporting call-by-reference parameter passing is the most common use of references, but not the only one. Other circumstances call for this type of parameter passing as well. Any declaration, for example, can use the & to define a reference. Following is a simple example:

```c++
void main()
{
    int i = 0;
    int &r = i; // initializes r to refer to i
    int j = 3;

    cout << i << r << j;
    r = j; // modifies i
    cout << i << r << j;
    i = 2;
    cout << i << r << j;
}
```

Here is the output:

```
0 0 3
3 3 3
2 2 3
```

Be careful with this kind of use. You cannot write code like the following:

```c++
int &r; // illegal
```

This code declares a reference variable but fails to initialize the reference. One way to understand references is to think of aliases in the Finder. When you create an alias, you have to specify what the alias represents. You cannot just create an alias for nothing and later determine what the alias represents. References are similar: When you define a reference, you must initialize it with a variable to which it refers.

You also can use references with the return type of routines. Usually, you do this so that you can use function calls on the left side of assignment statements:

```c++
int gVar1;
int gVar2;

int &GetCorrectGlobal()
{
    if (IsDaylightSavingsTime())
```


```c
return gVar1;
else
    return gVar2;
}

void main()
{
    GetCorrectGlobal() = 3;
}
```

This main program sets either gVar1 or gVar2 to 3, depending on whether IsDaylightSavingsTime returns true. For functions that return a reference, the reference is initialized with the variable returned from the function.

In summary, there are cases in which you may prefer C++'s use of references for parameter passing over the old style of passing by value. Besides being slicker looking, they give you the following advantages:

- They are simpler to use for parameter passing (you don’t need to use & and * for referencing and dereferencing).
- When combined with the `const` keyword, references provide simpler syntax and greater efficiency.
- References make reading code easier in many cases, especially when you adopt some simple rules for their use.

## Summary

Is C++ a better C? C++ is easier to use and handles errors better—two of C's weakest areas. These reasons alone should convince you that C++ is a better C.

Error handling is a big problem in C. Although your code may compile and link, it may not do what you want it to do. Here is how C++ improves this situation:

- C++ has much more rigorous error checking. This may be the best gift of all. Yes, you are required to do much more code preparation, such as with prototyping. The results, however, are clean compiles that work the way you want them to.
- C++ has fewer I/O errors. You don’t get I/O typing errors of which the compiler never informs you, such as typing %d when you meant %f. When you use the stream library, the C++ compiler figures out what kind of I/O is necessary based on the arguments you pass.
Twilight-zone `#define` macros are eliminated. C++'s inline function arguments cannot be multiply evaluated, so you don't end up with innocent looking `#define` macros like the following that produce nightmare results:

```
#define Square(x) ((x) * (x))
```

A new comment structure reduces the potential for error. `//` comments don't steal portions of your code like `/* */` comments can. Furthermore, you cannot forget to close `//` comments because they auto-terminate.

The second major area of improvement in C++ is ease of use. Many of these improvements obviously go hand in hand with better error handling. Ideally, one would want a language that is both easier to use and handles errors better—a great combination. Following is a review of the changes that make C++ easier to use:

- C++ uses simpler I/O code. With the stream library, you no longer have to write code that looks like this:

  ```
  printf("%d %d", s.i, s.j);
  ```

  On the contrary, your code is short and sweet:

  ```
  cout << s;
  ```

- Debugging is easier. Debugging is much easier with `const` variables because you can examine the values of `const` variables, something you cannot do with `#defined` constants. Similarly, you don't have to remember the special syntax of `#define`.

- C++ code is more readable. Code that mixes execution and declaration also reads much more naturally because you can group definitions of variables with the logically related code that uses them.

- Functions can be overloaded. You no longer have to create different names for identically acting functions just because they take different arguments. This frees your creativity for icon and About box design.

- Structs are easier to use. You no longer need to `typedef` them; the compiler does it for you.

- Code with argument parameters passed by address is easier to read. When you use references, it is easy to tell which variables are modified, something much more difficult with C.
Function arguments can have default values. This makes them much more flexible. You can choose whether to include arguments in function calls; the compiler fills in the blanks with your preset defaults.

Memory management is easier. You can delete null pointers using delete (but not so with free).

These are just the highlights of this chapter. You can start using these new C++ constructs armed with the sure and certain knowledge you are writing better C code in C++.

Exercises

1. Convert the following from ANSI C to idiomatic C++:

```cpp
#include <string.h>
#include <stdio.h>
#include <ctype.h>
#define kArraySize 200
#define kMaxStringSize 100

typedef struct MyStruct {
    char string[kMaxStringSize];
    int count;
} MyStruct;

MyStruct gStringArray[kArraySize];

/* InitializeArray: initializes the string array to empty entries */
void InitializeArray()
{
    int I;

    for (i = 0; i < kArraySize; i++)
    {
        gStringArray[i].string[0] = '\0';
        gStringArray[i].count = 0;
    }
}

/* FindEntry: returns matching entry, or NULL if no matching entry exists */
MyStruct *FindEntry(char *key)
```
{  
  int I;

  for (i = 0; i < kArraySize; I++)
    if (strcmp(gStringArray[i].string, key) == 0)
      return &gStringArray[i];
  return NULL;
}

/* PrintEntries: prints all nonempty entries in the array */
void PrintEntries()
{
  int I;

  for (i = 0; i < kArraySize; I++)
    if (gStringArray[i].string[0] != '\0')
      printf("%s: %d\n", gStringArray[i].string, gStringArray[i].count);
}

/* FindEmptyEntry: returns empty entry in array or NULL if no empty entries */
MyStruct *FindEmptyEntry()
{
  int I;

  for (i = 0; i < kArraySize; I++)
    if (gStringArray[i].string[0] == '\0')
      return &gStringArray[i];
  return NULL;
}

/* AddString: Adds the key to the array if not present, otherwise increments count */
void AddString(char *key)
{
  MyStruct *s;

  s = FindEntry(key);
  if (s == NULL) {
    s = FindEmptyEntry();
    if (s == NULL)
      return;
  }
strcpy(s->string, key);
}
s->count++;
}

/* GetWord: Gets the next word from standard input */
int GetWord(char *word)
{
    int c;

    /* skip white space */
    for (c = getchar(); c != EOF && isspace(c); c = getchar())
        ; /* do nothing */

    if (c == EOF)
        return 0;

    while (!isspace(c)) {
        *word++ = c;
        c = getchar();
    }
    *word++ = '\0';
    ungetc(c, stdin); /* put back the last character */
    return 1;
}

/* prints summary of input words, along with count of each word */
void main()
{
    char incomingWord[256];

    InitializeArray();
    while (GetWord(incomingWord))
        AddString(incomingWord);
    PrintEntries();
}

2. What does the following program print if the compiler uses call-by-value, call-by-reference, or call-by-name parameter passing?

void Foo(int I)
{

i = 3;
}

void Bar(int I)
{
    printf("%d\n", i * I);
}

main()
{
    int j = 4;
    Bar(j);
    Foo(j);
    printf("%d\n", j);
}

3. *Inside Macintosh*, the official Macintosh programming documentation, uses Pascal in its sample code. C programmers may find it difficult to convert from Pascal to C because of the changes they need to make in calls to the Toolbox. Here are the rules that the Pascal compiler uses when passing parameters:

- When passing a parameter whose value changes (VAR parameters), pass the address (call by reference).
- When passing a parameter whose value does not change and whose size is less than or equal to 4 bytes, pass the value (call by value).
- When passing a parameter whose value does not change, and whose size is greater than 4 bytes, pass the address (call by reference).

Look at these rules in action for two Toolbox calls, FrameRect and AddPt. Here are their declarations in Pascal:

Procedure FrameRect(r: Rect);
Procedure AddPt(src: Point; VAR dst: Point);

These routines can be called from Pascal with the following code:

FrameRect(myRect);
AddPt(srcPoint, dstPoint);

The corresponding declarations in C are as follows:
void FrameRect(const Rect *r);
void AddPt(Point src, Point *dst);

They can be called with the following code:

FrameRect(&myRect);
AddPt(srcPoint, &dstPoint);

Calls from both C and Pascal end up passing the same values, although the calls look different. Give alternate declarations for FrameRect and AddPt so that calls to them can be made from C++ the same way they are called from Pascal (with no &s) and so that the actual values that are passed remain the same.

4. memset is a C/C++ library function that sets memory to a given character repetitively. It is called with a pointer, a size, and a character. It is most often used to set memory to 0. Write a routine called MemorySet that, by default, sets the memory to 0 but accepts an argument specifying some other character. Be sure to test your routine.

5. Write a routine, Length, that computes the length of a string. Write routines, Length and Height, that compute the horizontal and vertical measurements of a rectangle. Be sure to test your routines.
As you stand poised on the brink of a new language, it is time to remind yourself what kind of brink it is. Moving from C to C++ (or any other object-oriented language) is not just about learning the syntax of the new features. It also means moving from a procedural to an object-oriented model of code design. There is plenty of hoopla in the industry over oopla (object-oriented programming and language application) which, no doubt, accounts for one of the reasons you are reading this book. This chapter comes before you are introduced to the nuts and bolts of C++ because you first need to shift your mind-set out of the procedural world.

This chapter provides an introduction to object-oriented programming (OOP) in particular and to the object-oriented model of design in general. Along the way, you receive a liberal dose of motivational discussion regarding the benefits of this model and coding style. You also get a glimpse of the practical impulses within computer programming that have given rise to the model shift.
Object-Oriented Programming

The flavor of object-oriented programming is quite different from procedural programming, and countless books have been written on this subject, many of which are worth reading. Before getting into some programming examples, it is instructive to review the motivations behind using this coding style.

Benefits of Object-Oriented Programming

Real world programmers facing recurring problems were the impulse behind the shift to object-oriented programming. As you might expect, the benefits OOP offers bear directly on its origins. The major benefits of OOP are the following:

- Object-oriented programming is more natural. Humans are categorizers. They naturally place objects into categories in their every day lives. Take, for instance, the Library of Congress—certainly a grand testament to the organizational impulse within us. It is no less appropriate to apply this principle to programming. Code becomes easier to design and reason about if programs are organized into categories of easily understandable objects. Procedural languages that use operations are somewhat less natural and, as such, are harder to use.

- Object-oriented programs are more maintainable. Object-oriented programming separates your program into objects. You can change the internal implementation or characteristics of an object without changing its external behavior or appearance. You also can add additional behaviors to some objects without affecting all objects of that type. This makes bugs easier to fix and features easier to add because the effects of changes are contained to smaller areas and thus easier to find.

- Code in object-oriented programs is more reusable. Object-oriented programming is ideally suited for real-world code reuse. The ivory tower approach to programming insists that it is feasible to pull code from a library and use it without modification. In real life, however, the problem is never exactly what existing code solves. Instead, minor (or sometimes major) changes are necessary before the code can perform the new task. OOP insists that there is no such thing as perfect library code that solves every specialized problem. Instead, it enables you to do programming by differences. You inherit some code and modify (override) the routines that you want to customize.

- Revisions in object-oriented code are quicker because less breaks when adding new features. Because code is easier to reuse and fixes affect fewer lines of code, you finish faster. This, of course, takes some time—you have to gain some experience using OOP. It is no different than typing: Hunt-and-peck typists are no match for a good touch.
typist. When hunt-and-peck typists first learn touch typing, they initially are slower than they used to be. Speed then goes back up to the starting speed and eventually exceeds it. The pattern is similar with OOP. Eventually your designs and your skills advance enough so that your code becomes easier to maintain and reuse.

Object-oriented programming is more fun. Separating your program into objects makes it easier to modify one part without breaking another part. Life is more fun because, as your program becomes more malleable, you can experiment with new features that you would not have dared adding before. You are not prevented from making a change because you fear repercussions in the rest of the code. Furthermore, because code is more reusable, you do not spend your time reinventing the wheel in program after program. You instead focus on the unique challenges a particular program presents, something that is more fun.

The Genesis of Object-Oriented Programming

OOP is a natural progression that started with modules. In the beginning, there was a morass, and the morass was made of global variables and routines that haphazardly modified the globals (see fig. 9.1). The hodge-podge of arrows in this figure and the three following ones represent program calls from other parts of the program.

Figure 9.1. Routines haphazardly modifying a data structure.

Modular Programming

Modular programming specifies that global variables and the routines that operate on them should be broken into modules. Rather than having all parts of your program access the binary tree data structure, for example, you have a few specific routines do all the accessing (see fig. 9.2). The other routines call the few.
With modular programming, your program is better structured and more maintainable. This is handled in C with .c and .h files. You implement modules by putting defining data and routines in a .c file and exporting the specific routines in an .h file. Because the data is in a .c file, only the routines in that file can access the data. For example, consider the following example of a binary tree in C. The BinaryTree.h code is as follows:

```c
void Insert(int key);
void Delete(int key);
int Find(int key);
```

Here is the BinaryTree.c code:

```c
struct Node {
    struct Node *leftChild;
    struct Node *rightChild;
    int key;
};
static struct Node *root = NULL;

void Insert(...)
{
    ...
}
...
```

There is a major problem with modules, however. Because modules still operate on global variables, you can have only one data structure. In the example, only one binary tree is allowed in your program. What if you need two, or five, or five hundred?

### Data Types

You can extend the module by packaging the global variables from the module into a structure. This structure can then be passed to and from the module routines. A new BinaryTree.h follows:
struct Node {
    struct Node *leftChild;
    struct Node *rightChild;
    int key;
};
struct BinaryTree {
    struct Node *root;
};

void InitializeBinaryTree(struct BinaryTree *tree);
void Insert(struct BinaryTree *tree, int key);
void Delete(struct BinaryTree *tree, int key);
int Find(const struct BinaryTree *tree, int key);

Here is BinaryTree.c:

void InitializeBinaryTree(struct BinaryTree *tree) {
    tree->root = NULL;
}

void Insert(...) {
    ...
}

This change in program structure enables multiple binary trees to form where there was only one. Unfortunately, it also enables the clients of the binary tree to modify the fields of the BinaryTree structure directly. Figure 9.3 shows the benefit of multiple binary trees, but also shows the weakness of enabling direct access to the data.

You may want to forbid such access because you know what can happen. A programmer is tempted to go to the tree and do wrong. He or she may not want to call the specified routine for fear it is too slow and, instead, set the data directly. Because the compiler cannot prevent direct operations on the structure, the data—and not your interface—is used. Later on, you are unable to change the implementation of the program for fear of breaking all that code.

**Abstract Data Types**

Because of this, the abstract data type (ADT) was developed. It is a data structure in which all access is through procedures; no access to the data structure is allowed. This can be implemented in C by hiding the BinaryTree structure behind a pointer. The client is allowed to know only that BinaryTree is a pointer to something. Direct access to the tree is cut off forever (see fig. 9.4).
Figure 9.3. Storing a binary tree in a structure.

Figure 9.4. ADT hides the internals of the binary tree.

It is up to the BinaryTree module to allocate and deallocate a BinaryTree. BinaryTree.h is as follows:

```c
struct BinaryTree;    /* we're not saying exactly what's in it just yet */

struct BinaryTree *AllocateBinaryTree();
void DeallocateBinaryTree(struct BinaryTree *tree);
void Insert(struct BinaryTree *tree, int key);
```

172
Symantec C++ Programming for the Macintosh, 2nd Edition
void Delete(struct BinaryTree *tree, int key);
int Find(const struct BinaryTree *tree, int key);

Here is BinaryTree.c:

struct Node {
    struct Node *leftChild;
    struct Node *rightChild;
    int key;
};
struct BinaryTree {
    struct Node *root;
};

struct BinaryTree *AllocateBinaryTree()
{
    struct BinaryTree *tree = (struct BinaryTree *) malloc(sizeof(BinaryTree));
    tree->root = NULL;
    return tree;
}

void DeallocateBinaryTree(struct BinaryTree *tree)
{
    free(tree);
}

void Insert(...)
{
    ...
}

Objects

Abstract data types can serve as the basis for a well-structured application. If you want to stop here, you can use OOP as an excellent way to create abstract data types. OOP enables you to do a great deal more, however. By adding the concepts of inheritance and polymorphism, OOP gives you a new kind of type: the class. Variables of this type are called objects or instances of a class.

The concept of inheritance enables you to capture commonality among classes. For instance, there are various kinds of binary trees: red-black, AVL, and splay trees. Each kind arose in an
attempt to overcome a shortcoming with standard binary trees worst-case performance. Standard binary trees can get very unbalanced. In the worst case, a tree with \( n \) nodes has a depth of \( n \). Searches, therefore, take linear time.

Each of these binary tree types implements \texttt{Insert} differently. Most of them do a \texttt{Find} in the same way, however. It is here that you greatly benefit from inheritance. Rather than writing each tree independently with \texttt{Find} code duplicated, inheritance enables you to specify that each tree is derived from the same, more general type. This way, each type of tree gets to inherit all the standard binary tree behavior. Moreover, a tree is free to override any general behavior for which it does not care (each overrides \texttt{Insert} differently). Extra functionality is not a problem, nor is augmenting existing behavior (for instance, splay trees must do additional work readjusting the tree during a \texttt{Find}).

**NOTE**

A red-black tree balances the tree by restructuring it when new nodes are inserted. A tree of \( n \) nodes is roughly \( \log n \) nodes deep, and searches never take linear time. (It is called a red-black tree because each node is marked with a red or a black color. The tree is then structured so that a parent and child node do not have the same color.)

A splay (or self-adjusting) tree also tries to balance the tree. It does not attack the tree at every insertion; instead, it guarantees a balanced tree over the long run. Each time a \texttt{Find} is done, an incremental balancing is performed as well. Over time, the tree slowly becomes balanced. Although one particular search may take linear time, the average length of a search is logarithmic. The philosophy is that of doing a little house cleaning each week rather than a much larger spring cleaning each year.

The second concept, polymorphism, is related to inheritance. By analogy, imagine that your code request transforms itself into the very one required (by the object to which it is talking). Polymorphic code can make a request of a particular object, and that object’s class ensures that the correct routine is called.

Suppose, for example, that you have red-black and splay binary tree objects and that you request an \texttt{Insert} for each one. A different \texttt{Insert} is called depending on whether the request is made of the red-black or the splay tree object.

Figure 9.5 shows the concept. Each binary tree object has the same external interface (one of three entry points along the box). Internally, however, the particular function depends on the particular object.
You can simulate polymorphism and inheritance in C by adding function pointers to each ADT. The following example shows binary, red-black, and splay trees; however, the actual binary tree code for each is not shown. The example focuses on showing the relationship between the different tree types. Following is BinaryTree.h:

```c
#pragma once

struct BinaryTree {
    void (*Insert)(struct BinaryTree *tree, int key);
    void (*Find)(struct BinaryTree *tree, int key);
    void (*Deallocate)(struct BinaryTree *tree);
    void *data; /* unknown data */
};

struct BinaryTree *AllocateBinaryTree();
```

Here is BinaryTree.c:

```c
#include "BinaryTree.h"
#include <stdio.h>
```

Figure 9.5. BinaryTree, RedBlackTree, and SplayTree objects.
#include <stdlib.h>

void BinaryTree_Deallocate(struct BinaryTree *tree)
{
    /* _ */
    free(tree);
}

void BinaryTree_Insert(struct BinaryTree *Tree, int key)
{
    printf("BinaryTree_Insert(%ld)\n", key);
    /* _ */
}

void BinaryTree_Find(struct BinaryTree *Tree, int key)
{
    printf("BinaryTree_Find(%ld)\n", key);
    /* _ */
}

struct BinaryTree *AllocateBinaryTree()
{
    struct BinaryTree *tree = (struct BinaryTree *)
        malloc(sizeof(BinaryTree));

    tree->Insert = BinaryTree_Insert;
    tree->Find = BinaryTree_Find;
    tree->Deallocate = BinaryTree_Deallocate;
    tree->data = NULL; /* no data for now */
    return tree;
}

The code for RedBlackTree.h is as follows:

#include "BinaryTree.h"

struct BinaryTree *AllocateRedBlackTree();

Here is RedBlackTree.cp:

#include "RedBlackTree.h"
#include <stdio.h>

void RedBlackTree_Insert(struct BinaryTree *Tree, int key)
printf("RedBlackTree_Insert(\ld)\n", key);
    /* _ */
}

struct BinaryTree *AllocateRedBlackTree()
{
    struct BinaryTree *tree = AllocateBinaryTree();

    tree->Insert = RedBlackTree_Insert;
    return tree;
}

SplayTree.h is as follows:
#include "BinaryTree.h"

struct BinaryTree *AllocateSplayTree();

Here is SplayTree.c:
#include "SplayTree.h"
#include <stdio.h>

void SplayTree_Insert(struct BinaryTree *Tree, int key)
{
    printf("SplayTree_Insert(\ld)\n", key);
    /* _ */
}

void SplayTree_Find(struct BinaryTree *Tree, int key)
{
    printf("SplayTree_Find(\ld)\n", key);
    /* _ */
}

struct BinaryTree *AllocateSplayTree()
{
    struct BinaryTree *tree = AllocateBinaryTree();

    tree->Insert = SplayTree_Insert;
    tree->Find = SplayTree_Find;
    return tree;
Finally, Main.c is as follows:

```c
#include <stdio.h>
#include "BinaryTree.h"
#include "RedBlackTree.h"
#include "SplayTree.h"

void InsertInTree(BinaryTree *tree, int value)
{
    tree->Insert(tree, value);
}

void main()
{
    BinaryTree *normalBinaryTree;
    BinaryTree *redBlackTree;
    BinaryTree *splayTree;

    normalBinaryTree = AllocateBinaryTree();
    normalBinaryTree->Insert(normalBinaryTree, 5);
    normalBinaryTree->Find(normalBinaryTree, 4);

    redBlackTree = AllocateRedBlackTree();
    redBlackTree->Insert(redBlackTree, 5);
    redBlackTree->Find(redBlackTree, 4);

    splayTree = AllocateSplayTree();
    splayTree->Insert(splayTree, 5);
    splayTree->Find(splayTree, 4);

    InsertInTree(normalBinaryTree, 6);
    InsertInTree(redBlackTree, 6);
    InsertInTree(splayTree, 6);
}
```

Look carefully at this example, especially at the main program. Notice the polymorphic call to Insert. It invokes a different actual function based on which object it is called.

Because you can do object-oriented programming in C, you may ask yourself whether you need C++. The same question could be asked about control structures. Because you can simulate while
loops, for loops, and so forth in assembly, do you need a high-level language? The answer is that it always is easier to program if the language directly supports a feature rather than simulates it. Thus, an object-oriented language automatically sets up function pointers and provides easy ways to create objects.

Another important aspect of OOP is its capability to focus on what and not on how something is done. For instance, when you call a function for an object, you just specify what you want done. The object takes care of implementing your request as it sees fit. Two objects of different classes may even respond to your function call in different routines (although if two objects are in the same class, they end up in the same routine).

**Object-Oriented Design**

The short description of object-oriented design (OOD) is that it is a model for designing your program as a collection of classes. On a practical level, the long description occupies many shelves in the programming section of a bookstore.

**Basic Rules of Object-Oriented Programming**

OOD is not a subject that can be covered in these few pages. Instead, some general rules of thumb are presented.

**RULE**

Learn from a master designer.

Like many forms of design, the more object-oriented code you create, the better you become. Part of the way you become good, however, is by observing a master designer. Although taking classes and reading books on computer programming and object-oriented design can help you on a theoretical level, there is nothing like seeing an artist in action. One of the fastest ways to become proficient in OOD is to become an apprentice to an expert.

**RULE**

Every class you create that is derived from another class must pass the “is a” test.
Suppose that you have a base class *Shape* with a derived class *Rectangle*. To pass the "is a" test, the statement "Rectangle is a Shape" should be true. On the other hand, because an ellipse is not a circle, you should not derive *Ellipse* from *Circle*. It is fine to derive *Circle* from *Ellipse*, however.

By following this rule, you always have derived classes that can take the place of their base classes. Because the derived class is a base class, it can handle the same requests and carry out the same actions. It just may handle the requests differently.

If the base class guarantees that some set of conditions is true after a call to one of its functions, it had also better be true for derived classes. At a minimum, derived classes should guarantee that those conditions are true, although they may have additional conditions as well.

Likewise, if the base class requires some conditions to be true before calling a particular function, any derived class should do the same.

<table>
<thead>
<tr>
<th>RULE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provide data abstractions.</td>
</tr>
</tbody>
</table>

Don't just wrap a bunch of variables in a class with methods to get and set their values. Instead, consider the functionality your clients need and provide it.

If, for example, you are dealing with a dog class, do not just have methods that provide all the data members: for example, name, age, breed, and fang length. Instead, provide routines that your clients really want. These could range from some of the lower level ones (like those just mentioned) to higher level routines like *IsMean* (which uses the age, fang length, and breed data members) or *IsPuppy* (which uses the age data member).

<table>
<thead>
<tr>
<th>RULE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keep your methods small.</td>
</tr>
</tbody>
</table>

Member functions (methods) should be small and written to do one thing. Methods that try to do too much are very difficult to override. You should not, for example, have code to do both a find and a replace in one routine because it makes it very difficult to override just the find part of the code. You end up copying the code and changing the find part, but leaving the replace part.
It is better to have one routine to do the finding and another to do the replacing. If necessary, you can add a `FindAndReplace` routine that calls `Find` and then calls `Replace`. Derived classes can then override `Find`, leaving `Replace` alone.

**RULE**

Model your objects on the real world.

If your objects are modeled on those in the real world, they react more naturally to changes. This style of design also provides an easy predictability for people not familiar with your design. How many times has a marketing person requested “one little change?” How many times has the response been “but that will cause changes to two-thirds of the program!” The requested change may be little in the real world but quite large in the program. If the program were modeled after the real world, a small change in the real world would be small in the program.

You also should use the real world as your model domain. For chemists, the real world may be molecules; for Macintosh programmers, the real world may be windows and menus.

**RULE**

Use abstract classes.

Abstract classes are those that are not useful in and of themselves. Their reason for being is to be derived. In turn, the derived classes provide the missing functionality that makes the abstract class useful. Logically, it also follows that your concrete classes should not need to be derived to be useful.

The abstract class spells out what its objects are able to do, even if derived classes must provide that behavior. Figure 9.6, for example, shows a class hierarchy for some graphical objects.

Clearly, each leaf class (a class with no derived subclasses) implements `Draw` differently. Furthermore, `TShape` is an abstract class; it is not useful by itself. If you had a `TShape`, how would it draw? In fact, the `Draw` function of `TShape` must be overridden by derived classes to provide any actual behavior. Notice that `TInscribedInRectangleShape` is another abstract class. The others are concrete.
Even though TShape does not implement Draw or GetBoundingRectangle, it does declare the routines. It does this to reflect that all TShapes can be drawn and that all TShapes can return a bounding box. This declaration is useful to both readers of the program and the compiler. Any TShape object can be drawn. But if you, the programmer, try to call the Draw function for something that is not a TShape, you receive a compile-time error. The compiler prevents you from calling a nonexistent function on an object.

**The Object-Oriented Design Process**

Object-oriented design can be broken into three steps that overlap and are not strictly sequential. These steps also are influenced by the design process itself. You design a class, do something else, and come back and tweak the class design based on something you learned in the implementation. Good designers do a great deal of tweaking, and the newer you are at this, the more you should tweak.

The three steps are as follows:

1. Identify objects. In, for example, a personal financial program (like Quicken), your objects could be CheckingAccount, SavingsAccount, CreditCardAccount, Check, and so forth.
2. Decide on behaviors for the objects. `CheckingAccount` and `SavingsAccount`, for example, must each accept deposits and support withdrawals. `CheckingAccount` must be able to write a check. `CreditCardAccount` must be able to credit a payment and debit for purchases. All the accounts need to reconcile themselves and provide the current balance. `Checks` must record payee, amount, data, and a memo. They also must be able to sort themselves and mark whether they are cleared.

3. Create a hierarchy from your objects. Hierarchies usually are created to enable code reuse. As you find common functionality among objects, you should move it into some common base class. It is out of such movements that abstract classes are born.

You can, for example, remove common functionality from `CheckingAccount` and `SavingsAccount` and create the abstract class `BankAccount`. You can then move the deposit, withdrawal, reconcile, and current balance behaviors up from `CheckingAccount` and `SavingsAccount` into `BankAccount`. You also might want to consider creating another abstract class, `Account`, that is the base class of `BankAccount` and `CreditCardAccount`. Then you could move the reconcile and current balance functions up to `Account` from its derived classes.

Becoming familiar with good design and practicing it with examples is the best way to become proficient at OOD. Look at the design of as many object-oriented systems as you can. Read books that describe the design of systems. Design your own simple systems. With time, your design abilities will evolve.

**Summary**

What you gained from this chapter is an idea of how object-oriented programming and design can benefit you and why they are worth the time to learn. You also learned some rules of thumb to help you in your initial attempts at this type of programming and design. C++, as an object-oriented language, is flexible and rich and fits well in the object-oriented model. By now, you should be convinced that this type of language is fun, so all that is left to do is learn it.
10

Getting Started with Objects

Now that you know something about the basics of object-oriented design, it is time to see how you can use what you have learned. So far you know that the fundamental structure of object-oriented programming is the object. What you also should know is that objects are not conjured out of thin air. Objects evolve from a familiar feature of C, that is, structs. Unlike their ancestors, however, C++ classes really are the Cadillacs of user-definable types. In this chapter, you look at how to use this important C++ feature.

You create a C program using structs and then do the same in C++. Along the way, you see how object orientation makes the program easier to write and control. After all, you are not learning object-oriented programming just to be a more polite team member. Where relevant, you also learn any important new Symantec C++ syntax.
Overview

The following is a list of what this chapter covers:

- You see a procedural program that uses structs. This program is a C program that takes a set of x,y coordinates. It then uses structs to determine and print out the length of a line.

- You create a C++ program that performs the same task as the procedural program. While writing this kind of program using objects, you also learn about C++ classes. Classes are containers for data and functions.

- You focus on how object-oriented syntax differs from procedural syntax. You also master some other object-oriented code conventions.

- You add a caching feature to both the procedural and object-oriented programs.

- You learn about access control. In C++, access control makes it much easier to deal with potential problems in programs. Access control, a serious power tool for the programmer, is great because it gives you the capability to determine who has access to data. Using C++ access control syntax, you can determine whether data access is public (anyone can access the data), protected (only the member functions of the class or derived classes can access the data), or private (only the member functions of the class can access the data).

- You delve into inline functions. You learn how to use them to create efficient programs.

- You add constructors to your C++ program. A constructor is C++’s way of ensuring guaranteed default values, a luxury you have enjoyed with C global variables for years.

You are going to learn object-oriented programming in the easiest way, by looking at the part of C that it evolved from: structs. To do this, you examine a sample program written procedurally using traditional C structures, and then the same program written using object-oriented structures.

A Procedural Program

This example deals with lines that have start and end points. Here are the struct declarations:

```c
struct PointStruct {
    int x;
    int y;
}
```
struct LineStruct {
    PointStruct from;
    PointStruct to;
};

These declarations are followed by the routines you use with these structures:

void SetPoint(PointStruct *p, int x, int y)
{
    p->x = x;
    p->y = y;
}

void SetLine(LineStruct *l, PointStruct from, PointStruct to)
{
    l->from = from;
    l->to = to;
}

float LineLength(const LineStruct *l)
{
    int height = l->to.x - l->from.x;
    int width = l->to.y - l->from.y;

    return sqrt(height*height + width*width);
}

The main program is as follows:

void main()
{
    LineStruct l;
    PointStruct fromPoint;
    PointStruct toPoint;

    SetPoint(&fromPoint, 1, 3);
    SetPoint(&toPoint, 4, 7);

    SetLine(&l, fromPoint, toPoint);

    cout << LineLength(&l);
}
This program prints out the following:

5.0

An Object-Oriented Program

Look at the code for the object-oriented version. (The main discussion of the new syntax and differences between the versions is in the next section of this chapter.)

The class types (corresponding to the procedural struct types) for the object-oriented case are as follows:

```cpp
class TPoint {
public:
    void Set(int x, int y);
    int fX;
    int fY;
};

class TLine {
public:
    void Set(TPoint from, TPoint to);
    float Length();
    TPoint fFrom;
    TPoint fTo;
};
```

The member functions for the TPoint and TLine classes, which likewise correspond to the routines SetPoint, SetLine, and LineLength in the procedural version, are as follows:

```cpp
void TPoint::Set(int x, int y)
{
    this->fX = x;
    this->fY = y;
}

void TLine::Set(TPoint from, TPoint to)
{
```
```cpp
fFrom = from;
fTo = to;
}

float TLine::Length()
{
    int height = fTo.fX - fFrom.fX;
    int width = fTo.fY - fFrom.fY;

    return sqrt(height*height + width*width);
}

The following is the main program:

main()
{
    TLine l;
    TPoint fromPoint;
    TPoint toPoint;

    fromPoint.Set(1, 3);
    toPoint.Set(4, 7);

    l.Set(fromPoint, toPoint);

    cout << l.Length();
}

This object-oriented version prints out the following:

5.0

Comparing Procedural with Object-Oriented Programs

Look at the syntax changes in the procedural and object-oriented programs as well as the new C++ language constructs.
The Descendant of Structs: The C++ Class

The standard syntax for classes is as follows:

class ClassName {
public:
data declarations
function declarations
};

Table 10.1 shows the procedural and object-oriented versions of the point structure. In particular, turn a sharp eye to the coupling of the functions and the data on which they operate.

Table 10.1. Procedural and Object-Oriented Points

<table>
<thead>
<tr>
<th>Procedural Version</th>
<th>Object-Oriented Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct PointStruct {</td>
<td>class TPoint {</td>
</tr>
<tr>
<td>int x;</td>
<td>public:</td>
</tr>
<tr>
<td>int y;</td>
<td>void Set(int x, int y);</td>
</tr>
<tr>
<td>};</td>
<td>int fX;</td>
</tr>
<tr>
<td></td>
<td>int fY;</td>
</tr>
<tr>
<td></td>
<td>};</td>
</tr>
<tr>
<td>void SetPoint(PointStruct *p, int x, int y)</td>
<td>void TPoint::Set(int x, int y)</td>
</tr>
<tr>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td>p-&gt;x = x;</td>
<td>this-&gt;fX = x;</td>
</tr>
<tr>
<td>p-&gt;y = y;</td>
<td>this-&gt;fY = y;</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

In the procedural example, the only connection between PointStruct and SetPoint is that the procedure takes a pointer to the data as a parameter. In contrast, the object-oriented version has the procedures and the data on which they operate tightly intertwined. The class is the C++ language construct that binds the procedure and data.
Class declarations, like struct declarations, must end with a semicolon (;). Forgetting the semicolon is among the most common mistakes made by C++ programmers of all levels.

Member Functions and Data Members
How do you make the data and functions bind to the class? By giving the class some member functions and data members. In the example, the member function Set is bound to the TPoint class. (There are two other member functions: Set and Length, both in TLine.)

Table 10.2 shows the syntax for accessing data members and calling member functions.

Table 10.2. Accessing Data Members and Member Functions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example</th>
<th>Member Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Members</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Syntax</td>
<td>variable.memberName</td>
<td>variable.memberFunction(parameters)</td>
</tr>
<tr>
<td>Examples</td>
<td>fromPoint.fX</td>
<td>l.Set(fromPoint, toPoint);</td>
</tr>
<tr>
<td></td>
<td>aLine.fFrom</td>
<td>fromPoint.Set(1, 3);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>toPoint.Set(4, 7);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>l.Length();</td>
</tr>
</tbody>
</table>

Keep in mind the following important points about class syntax:

▼ When you declare variables of type struct, they have no special name; you just call them struct variables.

▼ On the other hand, the variables you declare of type class do have specific names; you call them objects.

▼ Member functions are like normal functions, except that you can invoke them only on objects. Furthermore, member functions have access to the data within the object upon which they are invoked.

▼ The data and the member functions that operate on them are declared together within the class.
Given these points, look at how procedural and object-oriented main programs differ from each other. Table 10.3 contrasts the procedural with the object-oriented version of the main program. The syntax in the object-oriented case for calling the Set member function is similar to that used for accessing a data member. Think of it as if the object has a member function embedded in it. You access that member function just as you would access data. Unlike the procedural case, you do not explicitly pass 1. Instead, 1 is passed implicitly to the member function when you call l.Set.

### Table 10.3. Procedural and Object-Oriented Main Programs

<table>
<thead>
<tr>
<th>Procedural</th>
<th>Object Oriented</th>
</tr>
</thead>
<tbody>
<tr>
<td>void main()</td>
<td>main()</td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>LineStruct l;</td>
<td>TLine l;</td>
</tr>
<tr>
<td>PointStruct fromPoint;</td>
<td>TPoint fromPoint;</td>
</tr>
<tr>
<td>PointStruct toPoint;</td>
<td>TPoint toPoint;</td>
</tr>
<tr>
<td>SetPoint(&amp;fromPoint, 1, 3);</td>
<td>fromPoint.Set(1, 3);</td>
</tr>
<tr>
<td>SetPoint(&amp;toPoint, 4, 7);</td>
<td>toPoint.Set(4, 7);</td>
</tr>
<tr>
<td>SetLine(&amp;l, fromPoint, toPoint);</td>
<td>l.Set(fromPoint, toPoint);</td>
</tr>
<tr>
<td>cout &lt;&lt; LineLength(&amp;l);</td>
<td>cout &lt;&lt; l.Length();</td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

Now that you see how the main program differs in the procedural and object-oriented cases, look at tables 10.4 and 10.5 to see the differences in writing functions as member functions. Compare the object-oriented member functions with the functions in the procedural case.

### Table 10.4. Function Definitions versus Member Function Definitions

<table>
<thead>
<tr>
<th>Version</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedural</td>
<td>void SetPoint(PointStruct *p, int x, int y)</td>
</tr>
<tr>
<td></td>
<td>void SetLine(LineStruct *l, PointStruct from, PointStruct to)</td>
</tr>
<tr>
<td>Version</td>
<td>Definition</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>Object Oriented</td>
<td>float LineLength(const LineStruct *l)</td>
</tr>
<tr>
<td></td>
<td>void TPoint::Set(int x, int y)</td>
</tr>
<tr>
<td></td>
<td>void TLine::Set(TPoint from, TPoint to)</td>
</tr>
<tr>
<td></td>
<td>float TLine::Length()</td>
</tr>
</tbody>
</table>

### Table 10.5. Writing a Function with and without Objects

<table>
<thead>
<tr>
<th>Version</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedural</td>
<td>void SetLine(LineStruct *l, PointStruct from, PointStruct to)</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>l-&gt;from = from;</td>
</tr>
<tr>
<td></td>
<td>l-&gt;to = to;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td>Object Oriented</td>
<td>void TLine::Set(TPoint from, TPoint to)</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>fFrom = from;</td>
</tr>
<tr>
<td></td>
<td>fTo = to;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

One main difference is the :: operator (called the *scope resolution operator*). On the left side of the :: is the class name; on the right side is the name of the member function.

You access the data members differently as well. Compare both versions and notice in table 10.5 that member functions automatically have access to their data members. TLine::Set uses the fTo and fFrom data members without specifying a structure or object before the data members.

### The this Keyword

Another important difference in the object-oriented version is the use of the this keyword. Look at table 10.6 and compare TPoint::Set with SetPoint.
Table 10.6. Setting a Procedural and an Object-Oriented Point Value

<table>
<thead>
<tr>
<th>Version</th>
<th>Point Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedural</td>
<td>void SetPoint(PointStruct *p, int x, int y)</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>p-&gt;x = x;</td>
</tr>
<tr>
<td></td>
<td>p-&gt;y = y;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td>Object Oriented</td>
<td>void TPoint::Set(int x, int y)</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>this-&gt;fX = x;</td>
</tr>
<tr>
<td></td>
<td>this-&gt;fY = y;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

The `this` keyword is only defined in a member function; it is a pointer to the object on which the member function is called. If you execute the following code, it calls `TLine::Set`:

```c++
TLine::Set(fromPoint, toPoint);
```

Within `TLine::Set`, this is a pointer to the object `TLine`. Therefore, `this->fFrom` refers to the data member `from` within `TLine`.

Note that using `this->` is optional when accessing data members in a member function. The only reason to do so is to distinguish between the following:

- Accessing members (either data or functions) of this object.
- Accessing other data (local or global) or functions (global).

### C++ Naming Conventions

There is a naming convention, however, that enables you to distinguish between these two without using `this->`. The `f` and the `T` in the following code are examples of this naming convention:

```c++
void TLine::Set(TPoint from, TPoint to)
{
```
fFrom = from;
fTo = to;
}

Here is the convention in a nutshell:

▼ Label all data members with an initial `f`, global variables with an initial `g`, and local variables with another letter of your choice.

▼ Start class names with `T`.

▼ Use the scope resolution operator to call global functions with this syntax:

```cpp
::GlobalFunction()
```

Then, if you see a `::` before a function call, it is a global function.

▼ If you see an object before a function call—for example, `1.Set(_)`—you know it is a member function. You see later that nothing before a function call signifies calling a member function on the object `this`.

**NOTE**

This naming convention, which specifies the use of `T` and `f`, is common among Macintosh C++ programmers. The larger C++ community does not share this idiosyncrasy. As is true of most conventions, it is the result of historical stickiness. In Pascal, classes were called `types` and data members were called `fields`. The initial letters just stuck around to identify types and fields long after their usage corresponded to anything meaningful. Using this convention does, however, make it easy to determine which types are classes and which variables are data members.

At this point, you might be muttering to yourself that object-oriented programming is just an alternative syntax to procedural programming—there are no real advantages. On the contrary, you just haven’t gotten to the fun part yet. Now that you have learned enough about the syntactical changes and new terminology, it is time to tantalize you with the advantages.

In this new syntax, all the functions you can apply to the data are embedded in the class declaration; you see them simultaneously. It is true that you can do something similar in procedural programming by declaring all functions that apply to a structure together. The problem is that there are no language police to force you to do it that way. Enforcement provides much more programming power, as you see later in this chapter.
Adding Caching

Currently, both the procedural and object-oriented versions of the program recalculate the length of lines on demand. When making many line length requests, with the same from and to points, you are obviously doing many unnecessary calculations. You can add caching of line lengths to both versions of the program to deal with this inefficiency in the code.

Caching in a Procedural Program

Proceed with the procedural program. Here, as elsewhere in this book, new and changed elements are displayed in italic. To make this program cache correctly, follow these steps:

1. Add two extra fields to LineStruct, one to hold the cached line length and another to tell you whether the cached length is valid:

   ```
   struct LineStruct {
      PointStruct from;
      PointStruct to;
      float cachedLength;
      Boolean cacheIsValid;
   };
   ```

2. Initialize the cacheIsValid field to false any time you set the from or to points:

   ```
   void SetLine(LineStruct *l, PointStruct from, PointStruct to) {
      l->from = from;
      l->to = to;
      l->cacheIsValid = false;
   }
   ```

3. Finally, modify LineLength to set the cached value, if necessary, and return the cached value:

   ```
   float LineLength(const LineStruct *l) {
      if (!l->cacheIsValid) {
         int height = l->to.x - l->from.x;
         int width = l->to.y - l->from.y;

         l->cachedLength = sqrt(height*height + width*width);
         l->cacheIsValid = true;
      }
      return l->cachedLength;
   }
   ```
Caching in an Object-Oriented Program

The object-oriented program requires many of the same changes you just made to the procedural code. Follow these steps:

1. Add two data members to the class declaration:

   ```cpp
class TLine {
   public:
       void Set(TPoint from, TPoint to);
       float Length();
       TPoint fFrom;
       TPoint fTo;
       float fCachedLength;
       Boolean fCacheIsValid;
   };
```

2. Initialize `fCacheIsValid` to false whenever setting the from and to points:

   ```cpp
   void TLine::Set(TPoint from, TPoint to)
   {
       fFrom = from;
       fTo = to;
       fCacheIsValid = false;
   }
   ```

3. Finally, use the cached value and set it as necessary in `TLine: : Length`:

   ```cpp
   float TLine::Length()
   {
       if (!fCacheIsValid) {
           int height = fTo.x - fFrom.x;
           int width = fTo.y - fFrom.y;

           fCachedLength = sqrt(height*height + width*width);
           fCacheIsValid = true;
       }
   }
   ```
Happily in both cases, the main program does not require changing; it just runs faster if you repetitively calculate line lengths.

**Access Control**

Now that the code runs more efficiently, investigate a potential problem in both programs—a problem, it turns out, that is solvable only in the object-oriented version. The problem occurs if you do not use the `SetLine` function consistently:

```cpp
void main()
{
    LineStruct l;
    PointStruct fromPoint;
    PointStruct toPoint;

    SetPoint(&fromPoint, 1, 3);
    SetPoint(&toPoint, 4, 7);

    SetLine(&l, fromPoint, toPoint);

    cout << LineLength(&l);

    l.fromPoint.x = -33;
    cout << LineLength(&l);
}
```

Much to one’s chagrin, the above code prints the wrong value in the cached case. Only in the inefficient non-cached version does it correctly output the following:

```
5.0
40.8999
```

In the cached version it wrongly outputs the following:

```
5.0
5.0
```

The same snag occurs in the object-oriented version if you use a similar main program that directly modifies the `fX` field of `fFromPoint` rather than using the `TLine::Set` routine.
This problem happens because the main program has bad manners. It should use the SetLine
(TLine::Set in the object-oriented version) routine, but it doesn't.

This obviously raises the question about what you should do with the programmer who is using
your points and lines, but neglecting to use the interface you have so carefully provided.
Documentation and conventions are one approach. Document the routines that you want used
and chastise programmers who do not use them. If you cannot admonish strongly enough,
however, there is a better approach. Make the compiler enforce your convention. Simply have
the compiler refuse to compile code that does not use the interface, thereby heading off the
problem before it occurs. You can perform this nifty trick by structuring your program so that
it is impossible to modify data other than through your thoughtfully provided interface.

In C++, members of a class can have any one of three levels of access control: public, protected,
and private. With the public keyword, any function can access this member. With the
protected keyword, only functions of this class or of derived classes can access this member.
With the private keyword, only member functions of this class can access this member.

There are even defaults already in place. Members are private (now you know why the public
keyword was at the beginning of the TPoint and TLine classes).

Having learned about access control, it is time to fix the code so that this problem cannot occur
any more. You do this by modifying the TLine class declaration to make the data members
private:

```cpp
class TLine {
public:
    void Set(TPoint from, TPoint to);
    float Length();

private:
    TPoint fFrom;
    TPoint fTo;
    float fCachedLength;
    Boolean fCacheIsValid;
};
```

With this change in access control, the only functions in the program that can access (read or
write) fFrom, fTo, fCachedLength, or fCacheIsValid are TLine::Set and TLine::Length. In other
words, the only operations that the program can do with TLine objects are to call the Set member
function or call the Length member function.
Conventionally, you put your public members first, and then follow with your protected and private members. This way, programmers reading your class declaration first see what they can do with the class (the public part) without having to figure out your implementation (the protected and private parts). You do not have to use the order presented earlier. In fact, you can switch back and forth between public, protected, and private if you want:

```cpp
class TFoo {
private:
    int fPrivate1;
public:
    void Public1();
    int Public2();
protected:
    void Protected();
    int fProtected1;
private:
    void Private();
    float fPrivate2;
};
```

With this change in access control, the only functions in the program that can access (read or write) fFrom, fTo, fCachedLength, or fCacheIsValid are TLine::Set and TLine::Length. In other words, the only operations that the program can do with TLine objects are to call the Set member function or call the Length member function.

There always is somebody who wants to know what happens when you try to compile a main program that directly accesses fFrom. Take a look at the following code:

```cpp
void main()
{
    TLine l;
    TPoint fromPoint;
    TPoint toPoint;

    fromPoint.Set(1, 3);
    toPoint.Set(4, 7);
}
```
```cpp
l.Set(fromPoint, toPoint);

cout << l.Length();

l.fFrom.x = -33;
cout << l.Length();
}

The Symantec C++ compiler politely returns the following error message:

File Main.cp; Line 38
Error: member 'TLine::fFrom' of class 'TLine' is not accessible

Herein lies another advantage of object-oriented programming: Access control can make your C++ programs much more maintainable than their C equivalents. If you make all your data members private (which you should), you then have a very limited number of routines that directly access those data members, that is, the member functions of that class. Your code sleuthing should improve dramatically. The reasons for this are as follows:

▼ Tracking down errors is easier; for example, locating the value of a data member that is being corrupted requires sifting through less code.

▼ You can reason more easily about the effects of modifications to your data members.

For consistency’s sake, you also should change the program so that the data members of the TPoint class data are private. To do so, just change the class declaration to make them private:

```cpp
class TPoint {
public:
    void Set(int x, int y);

private:
    int fX;
    int fY;
};
```

The code does need to provide some way to access the x and y coordinates of the point because TLine::Length needs the values to calculate the length. To do this, you add member functions to TPoint, which returns those values:

```cpp
class TPoint {
public:
    void Set(int x, int y);
Member functions (such as GetX or GetY) that return the values of data members are called *accessors* or *getters*. Member functions that exist to modify the values of data members (like `Set`) are called *mutators* or *setters*.

In this case, the code for the accessors, `TPoint::GetX` and `TPoint::GetY`, is simple:

```cpp
int TPoint::GetX()
{
    return fX;
}

int TPoint::GetY()
{
    return fY;
}
```

`TLine::Length` also needs to be changed to use these new accessor methods because it no longer has access to the data members directly:

```cpp
float TLine::Length()
{
    if (fCacheIsValid) {
        int height = fTo.GetX() - fFrom.GetX();
        int width = fTo.GetY() - fFrom.GetY();

        fCachedLength = sqrt(height*height + width*width);
        fCacheIsValid = true;
    }
    return fCachedLength;
}
```

Now that you have controlled access to your `TLine` and `TPoint` objects, things are shaping up rather nicely. It is time to deal with the standard objection to access control that someone invariably raises.
Inline Functions

Progress always brings forth opponents raising objections. Access control opponents ask, "But what about efficiency?" Happily, the efficiency concern is easily addressable. By using inline functions, you get the best of both worlds: the safety of access control, with the efficiencies of direct member access.

Inline functions should appear in a header file, however, so you need to move the accessor methods to the header file that contains the TPoint class declaration and add the inline keyword before each function. Here is the correct Point.h header file:

```cpp
// Point.h

#ifndef POINT_H
#define POINT_H

class TPoint {
public:
   void Set(int x, int y);
   intGetX();
   intGetY();

private:
   int fX;
   int fY;
};

inline int TPoint::GetX()
{
   return fX;
}

inline int TPoint::GetY()
{
   return fY;
}
#endif
```

Chapter 10: Getting Started with Objects
WARNING

Make member functions inline only if you know they cannot change. Otherwise, modifying an inline member function requires recompiling all files that use the TLine class. This case is no different from all other optimizations, however. First, measure your code to see where it is spending time and then speed up those parts. Make sure that you have a fully functional program first in any case.

Adding Constructors

There is yet another good thing about C++. As you know, there are times when you want to ensure guaranteed default values. C++ responds to this desire with constructors.

Simple Constructors

Constructors can make classes even easier to use. For instance, there is nothing to prevent you from writing code like the following:

```cpp
void main()
{
    TLine l;

    cout << l.Length();
}
```

If you try to print the length of an uninitialized TLine, however, you get the following garbage output:

```
3282323.2382348234
```

TLines should have a guaranteed default value, and in C++, you ensure that they and other objects have one by providing a constructor. A constructor is a routine that the compiler executes at the time it creates the object. You normally use a constructor when you want reasonable default values. To see how it works, start with a constructor for TPoint that initializes fX and fY to default values. Following is the modified class declaration:

```cpp
class TPoint {
    // Point.h
}
```
public:
   TPoint();
   void Set(int x, int y);
   int GetX();
   int GetY();

private:
   int fX;
   int fY;
};

A constructor has the same name as its class and does not return anything. Here is the definition of the constructor:

```cpp
tppoint:TPoint()
{
   fX = fY = 0;
}
```

**NOTE**

If constructors do not return anything, why don't you declare them `void`? Because you might then think you could change the return type from `void` to something else, like `int`. Having no return type at all also distinguishes them more readily from normal member functions. You will see later that destructors also have no return type.

The C++ compiler calls the constructor automatically whenever it creates a `TPoint`. In the program, how many times are `TPoints` created? The answer is four. Two of these are found in the main program, where you define two `TPoints`:

```cpp
TPoint fromPoint;
TPoint toPoint;
```

Do not forget that `TLines` have `TPoints` as data members. Each `TLine` has two `TPoints`. Therefore, the following definition creates another two `TPoints`:

```cpp
TLine l;
```

You end up with four `TPoints` in your program, each of which is automatically initialized with the `TPoint` constructor to `(0,0)`.
If TLine has its fFrom and fTo data members initialized automatically to (0,0), do you still need a TLine constructor? Yes, because you have to initialize another data member, fCacheIsValid. If you do not initialize it to false, the following code might still print out garbage values:

```cpp
TLine l;
cout << l.Length();
```

This is because the fCacheIsValid data member might be true; therefore, the TLine constructor must initialize it to false.

**NOTE**

Trivia question: If the fCacheIsValid data member is a random value, what are the odds that it is true? The answer is 255/256. Booleans are unsigned chars, and the only value interpreted as false is zero.

When you are ready to make all these modifications, first add the constructor to the TLine class declaration:

```cpp
class TLine {
public:
    TLine();
    void Set(TPoint from, TPoint to);
    float Length();

private:
    TPoint fFrom;
    TPoint fTo;
    float fCachedLength;
    Boolean fCacheIsValid;
};
```

Next, simply add the definition of the TLine constructor:

```cpp
TLine::TLine()
{
    fCacheIsValid = false;
}
```
Use the same main function:

```cpp
void main()
{
    TLine l;

    cout << l.Length();
}
```

It generates the following output:

```
0.0
```

### Constructors with Parameters

Like all functions, you can overload a constructor based on the type and number of parameters. This leniency gives you the ability to make multiple constructors that initialize an object in different ways.

For instance, the main program can be much simpler if it takes advantage of constructors with parameters. First, you need a constructor to `TPoint` that takes initial x and y values. After you have it, the constructor can create a point initialized to (1,3) with the following:

```cpp
TPoint fromPoint(1, 3);
```

This is simpler than the bulkier code you were using:

```cpp
TPoint fromPoint;

fromPoint.Set(1, 3);
```

In fact, you can use this constructor directly in the call to `TLine::Set`. Here is the bulkier code:

```cpp
TPoint fromPoint;
TPoint toPoint;

fromPoint.Set(1, 3);
toPoint.Set(4, 7);
l.Set(fromPoint, toPoint);
```

Instead of this code, you can use the following:

```cpp
l.Set(TPoint(1, 3), TPoint(4, 7));
```
This reduction in code is a big bonus. So after all that fanfare, here is the definition of such a constructor:

```cpp
TPoint::TPoint(int x, int y)
{
    fX = x;
    fY = y;
}
```

For completeness, TLine also needs a constructor that takes two TPoints as parameters. The declaration is as follows:

```cpp
class TLine {
public:
    TLine();
    TLine(TPoint from, TPoint to);
    void Set(TPoint from, TPoint to);
    float Length();
private:
    TPoint fFrom;
    TPoint fTo;
    float fCachedLength;
    Boolean fCacheIsValid;
};
```

Here is the definition of the constructor:

```cpp
TLine::TLine(TPoint from, TPoint to)
{
    fFrom = from;
    fTo = to;
    fCacheIsValid = false;
}
```

You should remember that each of your constructors may repeat certain fragments of code. In this case, both TLine constructors repeat fCacheIsValid = false. How can you encapsulate the commonality? A constructor in a class cannot call another constructor from that class. So, you cannot do it that way. Both constructors can call a common member function, however, to do common initialization. In the TLine case, you can define a TLine::CommonInitialization member function that initializes fCacheIsValid:

208
Symantec C++ Programming for the Macintosh, 2nd Edition
class TLine {
    ...
private:
    void CommonInitialize();
    ...
};
...
TLine::TLine()
{
    CommonInitialize();
}
TLine::TLine(TPoint from, TPoint to)
{
    fFrom = from;
    fTo = to;
    CommonInitialize();
}
void TLine::CommonInitialize()
{
    fCacheIsValid = false;
}

You don’t duplicate the same code in more than one constructor with this approach. Although
this may not look important for this class, it could be for others. You can have copious quantities
of code to duplicate in some classes.

Now that you have new constructors for both TLine and TPoint, look at how much simpler you
can make the main program:

void main()
{
    TLine l(TPoint(1, 3), TPoint(4, 7));
    cout << l.Length();
}

Chapter 10: Getting Started with Objects
Summary

By now you know that object-oriented programming is not just more courteous than procedural programming. Examine the advantages. In C++, objects make your code better in many ways. Remember, procedural programs only have procedures that go out and pillage the structures. The result is that chaos often reigns. In the object-oriented world, on the other hand, the new class type enlivens the data, and objects can have polite, orderly interactions with other objects. When you add other features, such as access control, inlining functions, and constructors, it is easy to see that object-oriented code is more maintainable, easier to write, and easier to read.

The advantages do not end here, however. There is more to learn about objects and about C++. When you are ready to fully immerse yourself in object-oriented programming, dive into the next chapter.

Exercises

1. Why is TLine::CommonInitialize declared as private? Can you think of other cases where private member functions make sense?
2. How can you rewrite the final version of the main program so that it has only one statement, rather than two?
3. Convert the following procedural program to an equivalent (but better) object-oriented program:

```cpp
#include <string.h>
#include <stream.h>
struct Customer {
    char name[128];
    char address[128];
};
const int kMaxEntries = 64;
Customer gCustomers[kMaxEntries];
int gNumberCustomerEntries = 0;
void InsertInDatabase(Customer &cust) {
    if (++gNumberCustomerEntries > kMaxEntries) {
        cerr << "too many customers\n";
    }
    // Insert data here
}
```
gCustomers[gNumberOfCustomerEntries] = cust;
}

Customer *FindInDatabase(const char *name)
{
    for (int i = 0; i < gNumberOfCustomerEntries; i++)
        if (strcmp(gCustomers[i].name, name) == 0)
            return &gCustomers[i];
    return NULL;
}

void main()
{
    Customer customer1 = {"Nicholas Rhodes", "1328 Clock Avenue"};
    Customer customer2 = {"Carolyn LaGaly", "4416 Longridge Avenue"};
    Customer customer3 = {"Nan Corby", "243 Park Avenue"};

    InsertInDatabase(customer1);
    InsertInDatabase(customer2);
    InsertInDatabase(customer3);

    Customer *cust;
    if ((cust = FindInDatabase("Nicholas Rhodes")) != NULL) {
        cout << "Name: " << cust->name << 'n' << "Address: " << cust->address << 'n';
    } else {
        cout << "not found
";
    }
}
A Closer Look at Objects

Objects have behaviors. You send messages to them, and they do something. That, however, is not all. Objects also act like parents. They give behaviors to their children. These children are new C++ classes that inherit all of their parents' class behaviors as well as having their own unique behaviors. They are called derived classes. Unlike real children, however, they always follow their parents' rules.

This chapter begins with derived classes, moves on to show the power of polymorphism, and lastly, explains dynamic memory allocation and the tricks and intricacies involved in working with objects that allocate dynamic memory.
Overview

Here is an overview of this chapter:

- You learn about derived classes, the C++ method of extending existing classes.
- You modify the caching line program from the previous chapter so it can take advantage of derived classes. You also learn the syntax that derived classes use and what behaviors you can expect from them.
- You then use the TLine and TCachedLine code to see how polymorphism works.
- You have a quick discussion of multiple inheritance. You see how to write objects that inherit from more than one base class and how they are used in C++.
- You are introduced to the special routines you must write for objects with dynamic memory. You learn why you need them and how to write them.

Derived Classes

Look again at some code from Chapter 10. There you started with a TLine class that was modified to support caching. Although this worked, it is not the C++ way of handling the problem. You can use a more elegant approach by creating a caching TCachedLine class that is derived from the TLine base class. This section discusses how to do that. To begin, review the noncaching TLine class declaration from the last chapter:

```cpp
class TLine {
public:
    TLine();
    TLine(TPoint from, TPoint to);
    virtual void Set(TPoint from, TPoint to);
    virtual TPoint GetFrom();
    virtual TPoint GetTo();
    virtual float Length();

private:
    TPoint fFrom;
    TPoint fTo;
};
```
Following are the class's member function definitions:

```cpp
TLine::TLine()
{
}
```

```cpp
TLine::TLine(TPoint from, TPoint to)
{
    fFrom = from;
    fTo = to;
}
```

```cpp
void TLine::Set(TPoint from, TPoint to)
{
    fFrom = from;
    fTo = to;
}
```

```cpp
TPoint TLine::GetFrom()
{
    return fFrom;
}
```

```cpp
TPoint TLine::GetTo()
{
    return fTo;
}
```

```cpp
float TLine::Length()
{
    int height = fTo.GetX() - fFrom.GetX();
    int width = fTo.GetY() - fFrom.GetY();

    return sqrt(height*height + width*width);
}
```

### Declaring a New Derived Class

To create a new version that uses derived classes, first declare a new class, TCachedLine:

```cpp
class TCachedLine: public TLine {
public:
    TCachedLine();
}
```
TCachedLine(TPoint from, TPoint to);
virtual void Set(TPoint from, TPoint to);
virtual float Length();
virtual void InvalidateCache();

private:
    Boolean fCacheIsValid;
    float fCachedLength;
};

Look at the first line. It specifies that TCachedLine is derived from TLine. Here is the syntax you always should use when creating these derived classes:

class TDerivedClass : public TBaseClass {

NOTE

Always use the public keyword before the base class. The other alternatives are protected and private. Private is used only in very uncommon circumstances. Protected, on the other hand, is so rare that we have never figured out a circumstance when you should use it.

Overriding Member Functions

Look again at TCachedLine. Notice that TCachedLine redeclares both Set and Length even though they already are declared in TLine. By doing so, TCachedLine now overrides the original member functions provided in TLine. These new versions either totally replace or augment the original version.

TCachedLine also declares two private data members used for caching. Look at the constructors first:

TCachedLine::TCachedLine()
{
    InvalidateCache();
}
TCachedLine::TCachedLine(TPoint from, TPoint to) : TLine(from, to)
{
    InvalidateCache();
}

The first constructor invalidates the cache. Nicely enough, C++ automatically calls the constructor for the base class, TLine, as well as for the inherited data members (fFrom and fTo). This means you do not have to hover about and ensure that they get initialized.

You need to handle the second constructor differently, however, by calling the TLine constructor with arguments. You do this with an initializer list. Consisting of the base class name and any necessary arguments, an initializer list is separated from a constructor with a colon (:). You also use it to initialize data members.

Look now at the overriding features of TCachedLine and the functions Set and Length. Note that the override of Set augments the behavior of TLine::Set. It continues to do what TLine::Set does, but sets fCacheIsValid to false as well. The code looks like this:

```cpp
void TCachedLine::Set(TPoint fFrom, TPoint fTo)
{
    TLine::Set(fFrom, fTo);
    fCacheIsValid = false;
}
```

The first line of this function calls the old version of Set, and the next line augments the old behavior. The code for Length, on the other hand, is only slightly more complex. If the cache is not valid, it needs to call the original version of Length to find the value for the cache. After that is done, it returns the cached length. The following is what TCachedLine::Length looks like:

```cpp
float TCachedLine::Length()
{
    if (!fCacheIsValid) {
        fCachedLength = TLine::Length();
        fCacheIsValid = true;
    }
    return fCachedLength;
}
```

That's all there is to it. Now, you have code that caches and properly uses derived classes. There are advantages to this approach beyond just being an object-oriented approach.
Advantages of Derived Classes

What are the benefits of writing two separate classes (the derived class and the base class) instead of rewriting the original class? There are two important ones:

- You can now choose whether to use TLine or TCachedLine objects on a case-by-case basis (the tradeoff being space versus time).
- Even if you were to use only TCachedLine, the implementation is easier. The TLine class contains code relating to lines and calculating lengths, whereas TCachedLine contains caching code. This separation makes the code easier to understand and maintain.

Using the Caching Line Class

Now that you know how to write derived class code, look at how to use it:

```cpp
void Foo(Point fromPoint, Point toPoint)
{
    TLine l(fromPoint, toPoint);
    TCachedLine cl(fromPoint, toPoint);

    l.Length();       // calls TLine::Length()
    cl.Length();      // calls TCachedLine::Length()
    l.GetFrom();      // calls TLine::GetFrom()
    cl.GetFrom();     // calls TLine::GetFrom()
}
```

Here, you can see some of the power of an object-oriented language like C++. The program calls the correct version of the member function depending upon which kind of object it needs. Calling the Length member function, therefore, for a TLine calls TLine::Length, but calling the Length member function for a TCachedLine calls TCachedLine::Length because TCachedLine overrides Length. Because TCachedLine does not override GetFrom, both calls to GetFrom call TLine::GetFrom.

Note that the way TCachedLine works when it overrides Length is similar to function overloading, a feature covered in Chapter 8.

Polymorphism

Polymorphism is defined as a variable’s capability to refer to one of a variety of objects. These objects belong to different classes that are members of the same superclass.
Overridden Member Functions versus Function Overloading

As in function overloading, a call to an overridden function calls the right version of a function based on the type of its arguments. There is a crucial difference, however. With overloaded functions, the compiler determines at compile time what function to call. With overridden member functions, it is not until run time that the determination is made. This tiny variation makes all the difference in the world.

Look at an example of delayed determination. An array of lines is initialized with the following code:

```cpp
TLine *myLines[3];
TLine line0(TPoint(-5, 3), TPoint(50, 7));
TCachedLine line1(TPoint(6, 6), TPoint(10, 7));
TLine line2(TPoint(2, 3), TPoint(5, 9));

myLines[0] = &line0;
myLines[1] = &line1;
myLines[2] = &line2;
```

Figure 11.1 shows what the array (and lines) looks like after this code executes.

![Figure 11.1. Array and lines in memory.](image)
Look at the code that calls the \texttt{Length} member function for each line in the array:

\begin{verbatim}
void PrintLines(TLine *myLines)
{
    for (int i = 0; i < 3; i++)
        myLines[i]->Length();
}
\end{verbatim}

When you compile \texttt{PrintLines}, all you know about the array is that it holds pointers to \texttt{TLine} objects or to some derived class of \texttt{TLine}. Here is the important part: Any derived class may override \texttt{Length}. The compiler determines to what class an object belongs, and then calls the correspondingly correct \texttt{Length} function. This run-time determination is called \textit{method lookup} or \textit{polymorphism}.

\section*{Understanding the Power of Polymorphism}

It is common practice to use object-oriented languages in conjunction with frameworks (see Chapter 15). Frameworks are normally written as a collection of classes. Unlike a standard library that has functions you call, frameworks themselves call your routines. For frameworks to do this, you must first create derived classes, and then override particular member functions. Then the framework, using polymorphism, calls your overridden member function when calling that original member function.

To see how this process works in practice, look at the following code, which is a snippet of an application framework:

\begin{verbatim}
extern TView *gTheViewToDraw;
class TView {
public:
  ...  
  virtual void Draw();
  ...
};
...
void TView::Draw()
{
    // do nothing
}
...  
gTheViewToDraw->Draw();
\end{verbatim}
TView is a class that draws a portion of a window. The default Draw function does nothing; it is only when you override that it actually draws something. The global gTheViewToDraw holds a TView* with a Draw member function. You can have the framework call your Draw routine by writing a derived class:

```cpp
class TMyView: public TView {
public:
    ... 
    virtual void Draw();
    ...
};

void TMyView::Draw()
{
    code to do drawing
}
```

Next, you create a new TMyView and assign it to gViewToDraw:

```cpp
gViewToDraw = new TMyView;
```

From this you can see how the framework calls your routines.

**How Polymorphism Works**

Thankfully, you do not have to understand how the compiler implements polymorphism to use it. There are a number of reasons, however, that you might want to plum the depths of polymorphism:

- You enjoy the intellectual stimulation of seeing how a language feature is implemented (the philosophical reason).
- You are using a low-level debugger (like Macsbug) and need to trace through the compiled code (the practical reason).
- You want to better understand the time and space tradeoffs associated with using polymorphism (the efficiency reason).

If any of these reasons apply to you, read on; otherwise, you can skip this section.

How are these virtual functions implemented and run-time determinations made? Although not required by the language, most implementations of C++, including Symantec C++, use the same method. This method of dispatching virtual functions uses a *v-table*, a table of virtual pointers, for method lookup. Each object (with one virtual function or more) contains a hidden
pointer to a table of function pointers. The table, in turn, contains a pointer for each virtual function. When you call a virtual function, therefore, it is transformed into an indirect function call through the table.

As an example, figure 11.2 shows a diagram of two TLine objects along with a TCachedLine object. Note that the two TLine objects share a copy of the TLine v-table (which is what you may expect because they both execute the same member functions). The pointer of the TCachedLine object points to a different v-table.

The TLine::GetFrom and TLine::GetTo member functions are shared by both the TLine objects and the TCachedLine object. The following code uses the objects in figure 11.2:

```cpp
void PrintLines(TLine *myLines)
{
    for (int i = 0; i < 3; i++)
        myLines[i]->Length();
}
```

![Figure 11.2. V-tables in memory.](image)

---

**Symantec C++ Programming for the Macintosh, 2nd Edition**

222
After compilation, it ends up looking roughly like this:

```c
void PrintLines(TLine *myLines)
{
    for (int i = 0; i < 3; i++)
        (*myLines[i]->vptr)[3])(myLines[i]);
}
```

Two things of special interest happen in this code:

1. The compiler passes the `this` pointer as the parameter.
2. Every member function receives the `this` pointer as an extra (hidden) parameter.

The `this` pointer points to the object that is executing the member function. It enables access to members of an object from within a member function. Look at the following code:

```c
void TCachedLine::Set(TPoint fFrom, TPoint fTo)
{
    TLine::Set(fFrom, fTo);
    fCacheIsValid = false;
}
```

When compiled, it becomes something like the following:

```c
void TCachedLine::Set(TCachedLine *this, TPoint fFrom, TPoint fTo)
{
    TLine::Set(this, fFrom, fTo);
    this->fCacheIsValid = false;
}
```

**NOTE**

A virtual member function uses polymorphism. Nonvirtual member functions, on the other hand, work like normal functions—the determination of which function to call is made at compile time, rather than at run time.
Multiple Inheritance

You are not restricted to a single base class. C++ enables you to supply a derived class with more than one base class (called multiple inheritance). Here is the syntax that enables you to set up the class:

```cpp
class TDerivedClass: public TBaseClass1, public TBaseClass2, ...
```

With multiple inheritance, you inherit the data members and member functions from all your base classes as well. This is not a restriction, however, because you can override any of those member functions as well as provide your own.

When you use multiple inheritance, you can end up with a derivation hierarchy that is quite complex. With single inheritance, you have a simple tree structure. With multiple inheritance, on the other hand, you easily can end up with a clutter pile. Because of the problems associated with complexity, there is a debate in the object-oriented community about whether multiple inheritance should be used.

If you do use multiple inheritance, there are ways you can minimize the complexity. One way is to use a disciplined approach to multiple base classes:

- Keep your T classes in a tree hierarchy.
- Define new M classes that modify existing behavior.
- M classes should have names that are adjectives.
- T classes should have names that are nouns.
- A derived class should have at most one T base class, but can have more than one M base class.
- The main inheritance branch is composed of the T classes, while the M classes just add additional, optional functionality.

Examples of M classes are MPrintable, MDisplayable, and MWritable. Some examples of T classes are TPolygon and TShape. Look at figure 11.3 to get an idea of what a well-ordered multiple inheritance structure looks like.
Using Constructors and Destructors in Dynamic Memory Allocation

You need to give special consideration to the objects you use to allocate dynamic memory. To harmonize with the rest of your code, any class that allocates dynamic memory must override a number of standard C++ member functions. To understand why this is so, create a class that allocates dynamic memory and see which member functions have to be overridden.

The TPolygon Class

The TPolygon class is a polygon represented as a pointer to a number of TPoints. Here is the initial class declaration:

```cpp
class TPolygon {
public:
    TPolygon();
    TPolygon(const TPoint *thePoints, int numPoints);
};
```
virtual TPoint GetNthPoint(int pointNumber);
virtual float Length();

private:
    TPoint *fPoints;
    int fNumPoints;
};

There are two constructors. The first is a default constructor with no arguments:

TPolygon::TPolygon()
{
    fPoints = NULL;
    fNumPoints = 0;
}

The second is a constructor that initializes with a number of points:

TPolygon::TPolygon(const TPoint *thePoints, int numPoints)
{
    fNumPoints = numPoints;
    if (fNumPoints == 0)
    {
        fPoints = NULL;
    }
    else {
        fPoints = new TPoint[fNumPoints];
        for (int i = 0; i < fNumPoints; i++)
        {
            fPoints[i] = thePoints[i];
        }
    }
}

Along with these constructors, you need to create an accessor member function that returns a requested point:

TPoint TPolygon::GetNthPoint(int pointNumber)
{
    if (pointNumber >= 0 && pointNumber < fNumPoints)
    {
        return fPoints[pointNumber];
    }
    else {
        TPoint emptyPoint;
        return emptyPoint;
    }
}
Finally, you need some code that returns the length of the perimeter of the polygon by adding together the length of each line.

```cpp
float TPolygon::Length()
{
    float total = 0.0;

    for (int i = 0; i < fNumPoints - 1; i++) {
        TLine lineSegment(fPoints[i], fPoints[i+1]);
        total += lineSegment.Length();
    }
    return total;
}
```

### Problems with Dynamic Memory

Problems with dynamic memory are important and can make for some grueling debugging. This should be reason enough to make it worth your while to allocate and dispose of memory properly. Look at some of the things that can go wrong if you get lazy with your memory handling.

You should immediately seize upon the first of the problems with the `TPolygon` class. Look at this function:

```cpp
void MemoryLeak()
{
    const int kNumPoints = 3;
    TPoint somePoints[kNumPoints];
    ...
    TPolygon aPolygon(somePoints, kNumPoints);

    cout << Length = aPolygon.Length();
}
```

When `aPolygon` is constructed, it allocates a pointer to contain the points. But is this memory ever disposed of? No. This leads to the following rule for C++ objects with dynamic memory allocation.

**RULE**

Write a destructor when using dynamic memory allocation.
Here is an appropriate destructor for the TPolygon class. First, the declaration is as follows:

class TPolygon {
public:
    TPolygon();
    TPolygon(const TPoint *thePoints, int numPoints);
    virtual ~TPolygon();
...
};

The actual destructor code is shown below:

TPolygon::~TPolygon()
{
    delete fPoints; // does nothing if fPoints is NULL
}

The following code shows a second type of problem. Can you ferret out the problem with the TPolygon class?

void Foo()
{
    ...
    TPolygon polyOne(otherPoints, 2);

    {
        TPolygon polyTwo(somePoints, 3);
        polyOne = polyTwo;
    } // what about polyOne now?
}

The problem lies in the default assignment that C++ provides. By default, assigning one object to another assigns each of the data members in turn. Consider the following example:

polyOne = polyTwo;

This code becomes the following:

polyOne.fNumPoints = polyTwo.fNumPoints;
polyOne.fPoints = polyTwo.fPoints;
This assignment leaves you with a cantankerous memory leak when the old `polyOne.fPoints` is lost. `polyOne` and `polyTwo` end up pointing to the same `fPoints`. Figure 11.4 shows a diagram of the objects before and after the assignment. As you can see, this can get nasty.

**Figure 11.4.** Memory leak when assigning objects.

The first problem is that the old points are not disposed of. A second problem is that both objects point to the same points. When `polyTwo` is deleted (at the end of the block it is declared in), its destructor deletes the memory for its `fPoints`. Unfortunately, this leads to `polyOne` pointing to a block of memory that has been deleted, as shown in figure 11.5.

**Figure 11.5.** Polygons after deletion.

**NOTE**

It is not automatically wrong to have two objects sharing a piece of memory; it only becomes problematic when you do it unwittingly. If you intend to share memory, then this capability is an asset. In such cases, you should use a reference count to determine when to actually dispose of the shared memory.
Now, if you try to access the fPoints from polyOne, you get unknown results. Not only do you get random values, but even worse things occur when you call polyOne's destructor (at the end of Foo()). When you leave the scope of polyOne, its destructor deletes fPoints. Now you are trying to deallocate a block of memory that has been deallocated already.

What results from this grim picture? If you are very, very lucky, your code always crashes. More likely, however, you have intermittent crashes.

**NOTE**

Here is one of the more important programmer rules of thumb: Defects causing a crash 100 percent of the time are preferred to defects that cause intermittent crashes. The rarer the crash, the harder the cause is to find. (Users probably have different preferences.)

How do you deal with this memory assignment problem? The following rule for objects should be followed when using dynamic allocation.

**RULE**

Override the default assignment operator.

You should provide your own function for assignment—one that can dispose of the old memory and allocate new memory for the copied data.

Here is the declaration of the assignment operator:

```cpp
class TPolygon {
public:
    TPolygon();
    TPolygon(const TPoint *thePoints, int numPoints);
    virtual ~TPolygon();

    TPolygon &operator=(const TPolygon &poly);
    ...;
};
```

Here is the code for the operator:

```cpp
TPolygon &TPolygon::operator=(const TPolygon &poly) {
```
TPoint *points = NULL;
if (poly.fNumPoints > 0) {
    points = new TPoint[poly.fNumPoints];
    for (int i = 0; i < poly.fNumPoints; i++)
        points[i] = poly.fNumPoints[i];
} else {
    delete fPoints;
    fNumPoints = poly.fNumPoints;
    fPoints = points;
    return *this;
}

Now that you have taken care of the assignment operator, your code is well behaved and you have a scenario similar to that shown in figure 11.6.

![Figure 11.6. Rewritten assignment operator.](image)

**Assignment Operators Details**

You should know some important facts about the assignment operator. The first point to remember is that your assignment operator should take into account the possibility of self-assignment.

This can occur in code like the following:

```cpp
a = a;
```

Below is another example:

```cpp
ptr = &a;
a = *ptr;
```
Because both of these assignments work with normal data types, you should write your assignment operator so that it works correctly with your objects. For instance, you wrote the assignment operator for TPolygon so that it works correctly in the face of self-assignment.

The second point to remember is to return the correct value. You should return a reference to a TPolygon. This enables cascading assignments like the following:

```
  aPoly = bPoly = cPoly
```

To return a TPolygon & reference, use the following code:

```
return *this;
```

You are not done with memory problems yet because there are still more danger zones within the TPolygon class. When initializing one object from another, C++ uses a copy constructor. The default copy constructor initializes the data members from the copy. Here are the cases where you use such a constructor:

- **Explicit initialization of one object from another:**
  ```cpp
  TPolygon onePoly(anotherPoly);
  ```
  This example calls the copy constructor to initialize onePoly from anotherPoly.

- **Nonreference arguments to a function:**
  ```cpp
  void Bar(TPolygon aPolygon)
  {
    ...
  }
  ```

  ```cpp
  void Foo()
  {
    ...
    TPolygon polyOne(otherPoints, 2);
    Bar(polyOne);
  }
  ```
  This example initializes the formal parameter aPolygon with the actual parameter polyOne.
To initialize the return results of functions that return an object:

```cpp
TPolygon GetPoly()
{
    ...
    TPolygon aPoly(somePoints, 3);
    return aPoly;
}
```

This example calls the copy constructor to initialize the return result from `aPoly`.

The default copy constructor's behavior of copying each data member is not sufficient. As with assignment, you need to allocate space for the points. This leads you to the third rule for objects with dynamic allocation.

**RULE**

Override the default copy constructor.

Here is the declaration for the copy constructor:

```cpp
class TPolygon {
public:
    TPolygon();
    TPolygon(const TPoint *thePoints, int numPoints);
    TPolygon(const TPolygon &poly);
    virtual ~TPolygon();

    TPolygon &operator=(const TPolygon &poly);
};
```

Here is the constructor's implementation:

```cpp
TPolygon::TPolygon(const TPolygon &poly)
{
    fNumPoints = poly.fNumPoints;
    if (poly.fPoints == NULL)
        fPoints = NULL;
    else {
        fPoints = new TPoint[fNumPoints];
        for (int i = 0; i < fNumPoints; i++)
            fPoints[i] = poly.fPoints[i];
    }
}
```
Review of Memory Allocation Rules

In summary, when you have a class that has dynamic memory allocation, you need to apply three rules to your code. If you forget these rules, you are almost certain to have memory problems. Here are the rules again. Tattoo them on your forehead, if necessary.

▼ Write a destructor.
▼ Override the default assignment operator.
▼ Override the default copy constructor.

Summary

You learned how to create derived classes and how polymorphism works and why it is so powerful. You also learned why you need to be careful when you create and use objects that allocate dynamic memory.

Having learned this much, you have now gone a long way toward understanding some of the most important differences between an object-oriented language like C++ and a procedural one like C. As you can see, manipulating objects using C++ classes is a flexible, powerful approach to coding. But as any programmer who knows more than one language can tell you, knowing the syntax of a language is not the same as being able to write good code in it. The next important topic of discussion is obvious—how to write good C++, that is, code that is both syntactically correct and that uses an elegant, efficient style. In other words, you need to learn how to use C++ idiomatically. You learn how to do that in Chapter 12.
You might as well jump in quicksand as program in C++ with a C style. While it is true that you can use C++ as a better C, you are not taking advantage of the object-oriented features of C++. After you attempt to write object-oriented code, you may notice that many of your old C habits become more your enemies than your friends. Take, for instance, pointers. Like a kid who has not yet learned to talk, old-time C users point to everything. They have learned, like the toddler, that pointing is very efficient. Compared with what can be accomplished in C++ with references, however, pointers are well worth avoiding. The wizened C user who stubbornly refuses to give up C habits in C++ is no different from stubborn two-year-olds who fuss when it's time for their favorite teddy bear to be washed.
If you incorporate the coding suggestions in this chapter into your work, you can be a more effective C++ programmer. Avoid the temptation to ignore or forget these suggestions, however. You may want to use the warmly familiar C methods that accomplish a task. Unfortunately, using old methods in the new language is the fastest way to sink into unhappiness with both C and C++. As any speaker of multiple languages can tell you, the best way to learn a language is to go to that native country and speak nothing but that language.

Happily enough, most programmers can approach C++ first as users rather than as designers. As such, they are shielded from the more advanced features of the language. This also enables a more natural progression into the intricacies of the language. You can use classes to implement new features without worrying about how the classes are designed. Being cognizant of this reality, we have organized this chapter into two different sections. The first section is geared toward beginning C++ programmers working on a team—the clients of classes. The second section is for programmers with more experience who are taking on the role of class designers.

Both sections are loose collections of C++ tips and hints. They are not comprehensive discussions of C++ idioms or all there is to say about well designed C++. (There are several fine books on C++ design noted in Appendix E—some of which you should not be without.)

**Suggestions for Class Users**

The following tips help you use classes effectively in your programs.

**TIP**

**Use pointers less often than in C.**

C programs use pointers all over the place. They are used to pass large parameters and as function results. It should be no surprise by now that C++ replaces most of these uses. These alluring creatures are among the hardest for C programmers to give up, however, especially when you realize that C++ enables their continued use. You should ignore the temptation and, instead, focus on the advantages of using references.

**TIP**

**Use references to pass large parameters.**
For example, a C program frequently uses a pointer as an efficient way to pass large parameters:

```c
void AFunction(const LargeStructure *s);
```

To accomplish the same result in C++, you should use a reference instead:

```c
void AFunction(const LargeStructure &s);
```

**TIP**

Use references to return function results.

In C, you use pointers instead of values as function results. Using this method, the caller can avoid the overhead of a large value or can change what is pointed to. You might, therefore, find the following example in C:

```c
const LargeStructure *GetIthLargeStructure(int i);
```

You are just as likely to see this:

```c
LargeStructure *GetIthLargeStructure(int i);
```

To provide the same functionality in C++, use references and do the following instead:

```c
const LargeStructure &GetIthLargeStructure(int i);
```

Or, you might do this:

```c
LargeStructure &GetIthLargeStructure(int i);
```

**TIP**

References are limited.

A problem arises, however, for NULL values. In C, it is no big deal to allow NULL as a parameter or return result—it is in the range of possible values for a pointer. This is not so for a reference. Some programmers try to be sneaky and get around this problem by dereferencing NULL and passing a NULL reference:
AFunction(*NULL);    // don't do this

Or, they try checking the address of the return result for NULL:

LargeStructure &returnResult = GetIthLargeStructure(i);
if (&returnResult != NULL) {    // don't do this
    // ...
}

Do not use either of these methods. You have three choices in such situations:

1. Go ahead and use a pointer.
2. Pass an extra Boolean that states whether the function succeeded:

   Boolean success;
   LargeStructure &returnResult = GetIthLargeStructure(i, &success);

   if (success) {
       // ...
   }

3. Return a sentinel value that is different from all valid values:

   in some .h file:
   const kInvalidLargeStructure = {-9999,...}; // sentinel value

   ...

   LargeStructure &returnResult = GetIthLargeStructure(i);

   if (returnResult != kInvalidLargeStructure) {
       // ...
   }

TIP

Select whether to use default parameters or overloaded functions.

C++ provides default parameters so that callers can omit parameters. Similarly, C++ provides overloaded functions. This way you can have multiple functions of the same name with
different parameter lists. In some cases, you have a choice of using one or the other of these C++
features. How do you decide between them?

Default parameters can make life easy for clients who just want default behavior from your
function. You can obtain the same effect using overloaded functions, however. Look at the
following example:

class TFoo {
public:
    enum {kDefaultOne = 5, kDefaultTwo = 7};
    void AMethod(int i, int j = kDefaultOne, int k = kDefaultTwo);
private:
    // ...
};

// in the .cp file:
void TFoo::AMethod(int i, int j, int k)
{
    // ...
}

TFoo::AMethod requires one parameter, i. The remaining two parameters, j and k, are optional.
You can do the same thing with an overloaded function:

class TFoo {
public:
    void AMethod(int i);
    void AMethod(int i, int j);
    void AMethod(int i, int j, int k);
private:
    enum {kDefaultOne = 5, kDefaultTwo = 7};
    // ...
};

// in the .cp file:
void TFoo::AMethod(int i)
{
    AMethod(i, kDefaultOne);
}

void TFoo::AMethod(int i, int j)
{
AMethod(i, j, kDefaultTwo);
}

void TFoo::AMethod(int i, int j, int k)
{
    // ...
}

By overloading AMethod in this manner, you let the client decide whether to call it with one, two, or three parameters.

There are tradeoffs involved in using either approach. Default parameters are less bulky and require less code. You do lose some flexibility, however, because the defaults are passed along with any other parameters at compile time by the caller. Overloaded functions, on the other hand, take more time and require additional function calls. The upside here is that missing parameters are provided at run time rather than compile time. They are not compiled into the calling code; instead, they are supplied by the overloaded functions.

In addition, overloaded functions can be overridden by a derived class to provide a different default value. Conversely, default parameters fix the default value for all time for all derived classes. The decision of which to use—default parameters or overloaded functions—can be made then depending on whether you need flexibility in the default values or run-time compilations. If you do not, use the simpler default parameters.

**Issues for Class Designers**

The following tips are targeted toward those of you who want to design your own classes.

**TIP**

Do not put inline member functions in the class declaration.

Most C++ books and articles present the implementation of a member function within the class declaration. This, of course, makes the member function inline. When you see this sort of construction, you should ask whether profiling reveals the need for this optimization. In most cases the answer is no. Can you imagine the reason for these implementations? Classes are
presented this way to conserve printed space. Authors face very different optimization problems than do programmers. They want to minimize the amount of paper used. Because you do not have the same problem, you do not have the same solution.

Do not clutter your class declaration with inline member functions like the following:

```cpp
class TManyInlines
{
public:
    TManyInlines(int i) {f1 = 3; f2 = 6; f3 = i;}
    int Get1() { return f1; }
    int Get2() { return f2; }
    int Get3() { return f3; }
    void Set1(int i) { f1 = i; }
    void Set2(int i) { f2 = i; }
    void Set3(int i) { f3 = i; }

private:
    int f1;
    int f2;
    int f3;
};
```

If your heart is absolutely set on using inline functions, use the `inline` keyword instead. Then put the implementations of your member functions directly in your header file:

```cpp
class TManyInlines
{
public:
    TManyInlines(int i);
    int Get1();
    int Get2();
    int Get3();
    void Set1(int i);
    void Set2(int i);
    void Set3(int i);

private:
    int f1;
    int f2;
    int f3;
};
```
inline TManyInlines::TManyInlines(int i)
{
    f1 = 3;
    f2 = 6;
    f3 = i;
}

inline int TManyInlines::Get1()
{
    return f1;
}

inline int TManyInlines::Get2()
{
    return f2;
}

inline int TManyInlines::Get3()
{
    return f3;
}

inline void TManyInlines::Set1(int i)
{
    f1 = i;
}

inline void TManyInlines::Set2(int i)
{
    f2 = i;
}

inline void TManyInlines::Set3(int i)
{
    f3 = i;
}

Although this makes your header file slightly larger, it gives you the same effect as putting the code directly in your class declaration. It also has a number of additional benefits:

▼ The class declaration is less complex and easier to read.
Class declaration readers do not have to be aware of the inline functions and their implementation.

It is easier to create a document showing the interface to the class—you just delete the private parts of the class. If the code were directly in the class declaration, you would have to remove it all before creating the document.

Most importantly, changing from inline to non-inline is a simple editing exercise. All you have to do is move the function definition into a .cp file and remove the keyword inline.

**TIP**

Use enumerations for class constants.

Most programmers find themselves in the situation of needing a class constant that they want to be evaluated at compile time. It could be for the private use of the class or for use by clients of the class. This situation poses a puzzler for beginning C++ programmers. They find it difficult to determine the right language construct to use. Is it a const data member? No. A static data member? No. Well, how about a const static data member? Still no.

The right answer is to use an enumeration. This example shows you how to do so for the private use of the class and for its clients:

```cpp
class TDooHickey {
public:
    enum EOperation {eAdd, eSubtract, eMultiply, eDivide};
    enum {kCanPrint = true};
    TDooHickey(EOperation op, Boolean canPrint = !kCanPrint);

private:
    enum {kArraySize = 512};
    int fValues[kArraySize];
};

// in a .cp file:
TDooHickey aDooHickey(TDooHickey::eAdd, TDooHickey::kCanPrint);
TDooHickey anotherDooHickey(TDooHickey::eMultiply);
```
The e prefix signifies a classic enumeration constant where a choice is made from a number of different possibilities. The k prefix signifies a constant that just happens to be implemented with an enumeration.

Note that the class name is required for these constants when outside the class. You must scope the constant with the class name using the scope resolution operator ::. This way, it is possible for a number of classes to define the constant eDivide. The fact that it is nestled safely within the class prevents any conflict between the various eDivides.

Because enumeration constants must have integral values, this only works for constants that can be represented as integers. Booleans and chars work fine, but do not try this for constant TPolygons, for instance.

**TIP**

Use initializer lists instead of assignments.

Initializer lists contain the parameters for the constructors of the base class and the data members of the current class. These lists are present in C++ because there are certain parts of a class that are constructed before the body of the constructor executes. These two class parts are the inherited data members from the base class and the data members of the class.

An initializer list provides the mechanism for passing the parameters to the constructors of those two parts. As an example, take the following class declarations as a given:

```cpp
class TBase {
public:
    TBase(int param1, int param2);

private:
    // ...
};

class TPolygon {
public:
    TPolygon();
    TPolygon(int numPoints, Point *points);
    TPolygon(const TPolygon &);

    TPolygon &operator=(const TPolygon &poly);
};
```
private:
   // ...
};

class TDerived: public TBase {
public:
   TDerived(int x, const TPolygon &aPoly);

private:
   int fDataMember1;
   TPolygon fPolygon;
};

Here is the constructor for TDerived:

TDerived::TDerived(int x, const TPolygon &aPoly) :
   TBase(x, TickCount()),
   fDataMember1(x * x),
   fPolygon(aPoly)
{
}

It uses the initializer list as a way to provide parameters for the TBase constructor and also as a way to initialize its data members, fDataMember1 and fPolygon. You might imagine that initializing data members this way is unnecessary because the constructor could have been written as follows:

TDerived::TDerived(int x, const TPolygon &aPoly) :
   TBase(x, TickCount())
{
   fDataMember1 = x * x;
   fPolygon = aPoly;
}

While it is legal to write it this way, it leads to an inefficiency. In the original correct constructor, initializing fPolygon required a call to one routine—the copy constructor (TPolygon(const TPolygon &)). This new constructor, using assignment, requires these two calls instead:

1. A call to the default constructor (TPolygon()) to initialize the polygon to a known state. This constructor is automatically called before the code within TDerived::TDerived.
2. A call to the assignment operator to assign from the polygon parameter to fPolygon.
Although one extra call may not seem like much, in some classes the default constructor may allocate memory that the assignment operator turns around and deallocates. This is useless work that consumes time.

In addition to the efficiency concerns, there are cases where data members must be initialized using an initialization list. For instance, how do you initialize a const data member? Not in the class declaration. Even if you could do it that way, all instances of your class end up with the same value for that data member. Not within the body of the constructor. It is a const, so you cannot assign to it.

To initialize a const data member, you must use an initializer list:

```cpp
class TSanDiegoWeather {
public:
    TSanDiegoWeather(float temperature, int year);
private:
    const float fTemperature;
    const int fYear;
};
```

```cpp
// in the .cp file:
TSanDiegoWeather::TSanDiegoWeather(float temperature, int year) :
    fTemperature(temperature),
    fYear(year)
{
    // fYear = year; // illegal
    // fTemperature = temperature; // illegal
```

Similarly, reference data members must be initialized with an initializer list.

**TIP**

Make classes work like built-in types.

The classes you define should function just like built-in types. Because you can assign one `int` to another, you should be able to assign one `TPolygon` to another. You can cascade assignments of `ints` (`a = b = c`); the same is expected of `TPolygons`. Your goal is to write your classes so that they have the look and feel of built-in types. This makes users of your classes happy because they have a much easier time writing programs. You become a language god who has extended the language with a newly created species.
To make classes work this way, it is important to write a default constructor, a destructor, a copy constructor, and an assignment operator. In addition, you probably want to write an equality operator (==) and an inequality operator (!=).

If objects of your class can be compared, you also need to write the comparison operators: <, <=, >, and >=. You really must write all four of them. Furthermore, it is up to you to make sure that if a < b evaluates to true, b > a also does. It goes without saying that b <= a and a >= b should evaluate to false. If you break these relationships, users of the class can be quite surprised.

In general, overloading arithmetic operators is another matter. You should overload them only if your class has a numeric nature or if users of your class naturally expect them to work. It is not wise to define the + operator to mean jump, as in theCow + theMoon. It is far better to require clients to write theCow.Jump(theMoon). On the other hand, defining the + operator to mean addition for TPoints is something reasonable to do.

As with gravity, it probably is best if the arithmetic operators follow the laws that users expect:

\[
\begin{align*}
  a + b & \quad \text{is equivalent to} \quad b + a \\
  a * (b + c) & \quad \text{is equivalent to} \quad a * b + a * c \\
  a - b & \quad \text{is equivalent to} \quad a + -(b)
\end{align*}
\]

In your efforts to design class masterpieces, a good design rule is to use the principle of least surprise. Try not to startle your clients; instead, make classes fit seamlessly into their expectations. Design your classes to work exactly as users expect.

**Summary**

By designing your applications and writing code with certain basic C++ rules in mind, you learn to program like a veteran object-oriented coder. The rules listed in this chapter are explored at more length in Part 3 of this book, where we use the THINK Class Library to build applications.
In any large program, it is difficult to remember everything that is going on. This is true even for an object-oriented program; unless you have an eidetic memory, it can be hard to remember classes, their relationships, and the functionality they offer.

Luckily, you do not have to worry about remembering all the relationships with Symantec C++. Symantec liberates you from the limitations of your memory with the THINK Class Browser.

What Is the Browser?

The Browser is a tool that displays your class hierarchy. It shows classes and their subclasses within a graphic framework. Figure 13.1 shows an example of a Browser window displaying a C++ program.
As you can see, the Browser can be a very helpful tool to the object-oriented programmer in certain cases. The next sections discuss how to use the Browser.

## Creating a Browser Window

First, you need to create some simple class declarations:

```cpp
class TShape {
public:
    TShape();
    virtual void Draw();
};

class TQuadrilateral: public TShape {
public:
    TQuadrilateral();
    virtual int GetSideLength(int i);
    virtual int GetPerimeter();
};
```

Figure 13.1. The Class Browser window.
class TCircle: public TShape {
public:
    TCircle();
    virtual void Draw();
};

class TSquare: public TQuadrilateral {
public:
    TSquare();
    virtual void Draw();
};

And you need to create some implementations:

#include "Shape.h"

TShape::TShape()
{
    // nothing to override.
};

void TShape::Draw()
{
    // do nothing
};

TQuadrilateral::TQuadrilateral()
{
};

int TQuadrilateral::GetSideLength(int i)
{
    return 0;
};

int TQuadrilateral::GetPerimeter()
{
    return 0;
};

TCircle::TCircle()
{
};
void TCircle::Draw()
{
};

TSquare::TSquare()
{
};

void TSquare::Draw()
{
};

Bring the project up to date. Now that you have these classes, you can use the Browser. It is located in the Source menu. When you choose the Browser, you see a window that looks like the one shown in figure 13.2.

![Figure 13.2. A simple Browser window.](image)

The graphic representation displays the relationships between the classes. The TShape class is on the left because it has no base class. From TShape, two lines extend to the right to show TShape's derived classes, TCircle and TQuadrilateral. Similarly, a line extends to the right of TQuadrilateral to its derived class, TSquare.

The Browser does not just display the relationships between the classes; it is an interactive tool as well. When you click on one of the classes, a pop-up menu of the member functions is displayed (see fig. 13.3).

![Figure 13.3. A member function pop-up.](image)
If you select one of the items in the pop-up menu, you move directly to the member function source code in the editor. If, for example, you select TShape(void), the Project Manager shows you the source code for the TShape constructor (see fig. 13.4).

![Figure 13.4. After selecting TShape(void) from the pop-up.](image)

If you want to see the class declaration, you can double-click on the class name from within the Browser. Double-clicking on TShape from the Browser, for example, takes you to the TShape class declaration (see fig. 13.5).

![Figure 13.5. After double-clicking TShape in the Browser.](image)

Alternatively, you can use the Enter or Return key after selecting the class from within the Browser to get the same results.
If you want to navigate through C++ classes in the Browser, you can use the arrow keys, as shown in table 13.1.

Table 13.1. Keyboard Shortcuts in the Browser

<table>
<thead>
<tr>
<th>Key</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑ and ↓</td>
<td>Cycles between siblings of current class</td>
</tr>
<tr>
<td>←</td>
<td>Selects the base class of the current class</td>
</tr>
<tr>
<td>→</td>
<td>Selects one of the derived classes of the current class</td>
</tr>
<tr>
<td>Tab</td>
<td>Cycles through all of the classes</td>
</tr>
</tbody>
</table>

Shortcuts from the Editor

When reading code, you often need to go quickly from a function call to its corresponding source code. In the editor, holding down the Option key while you double-click (Option-double-click) on a function name displays the source code for that function. If, for example, you Option-double-click on a call to GetPerimeter (see fig. 13.6), the editor displays the code for TQuadrilateral::GetPerimeter (see fig. 13.7).

Figure 13.6. Option-double-clicking on GetPerimeter.

Figure 13.7. After Option-double-clicking on GetPerimeter.
Using Identically Named Functions

You sometimes may have different functions with the same name—either different classes that have member functions with the same name or overloaded functions. The next sections examine each of these cases.

Member Functions with the Same Name

Option-double-clicking on a name displays the Class Browser window. The Browser then outlines each class that has a function with that name. Figure 13.8 shows an example of Option-double-clicking on Draw.

![Figure 13.8. Option-double-clicking on Draw.](image)

The editor then displays the Class Browser window, shown in figure 13.9. Each of the outlined classes has a Draw member function.

![Figure 13.9. After Option-double-clicking on Draw.](image)

You also should note that a few standard Browser features work slightly differently. Press the Tab key to cycle through the outlined classes, rather than through all classes. Similarly, double-clicking on an outlined class name displays the code for the Draw member function. If, for example, you double-click on TCircle (see fig. 13.10), the Browser displays the source code for TCircle: : Draw, as shown in figure 13.11.
Overloaded Functions

Overloaded functions are the other case in which you can find multiple functions with the same name. If you Option-double-click on a name in this case, the Browser displays the first overloaded function.

In the following code, for example, Option-double-clicking on Bar displays the first Bar, as shown in figure 13.12:

```c++
void Bar(int i)
{
    // ...
}

void Bar()
{
    // ...
}
```

The editor provides a shortcut for getting to class declarations as well as to member functions. The shortcut is to hold down the Command and Option keys while double-clicking (Command-Option-double-click) on the name of a class. The editor then displays the class declaration.
Browsing with Multiple Inheritance

The Browser also works if you are using multiple inheritance. With single inheritance, the class hierarchy forms a tree (actually, a forest because the hierarchy is a collection of rooted trees). With multiple inheritance, the diagram becomes more complicated. Specifically, it forms a directed acyclic graph (DAG). It is directed because the direction of the line between classes matters. It is acyclic because there can be no cycles.

To see how this works, you first need some class declarations:

```cpp
class TFoo {
public:
    void Foo();
};

class MBarable {
public:
    void Bar();
};

class TFooSubclass: public TFoo {
    void FooSubclass();
};

class TBarableFoo: public TFoo, public MBarable {
    void BarableFoo();
};

class MBarableSubclass: public MBarable {
    void BarableSubclass();
};
```

Figure 13.12. After Option-double-clicking on overloaded Bar.
Figure 13.13 shows the Browser for these class declarations.

![Browser with multiple inheritance.](image)

**Figure 13.13.** Browser with multiple inheritance.

## Summary

This chapter discussed the basic features of the Browser. It is a useful tool for quickly navigating through your source code. It can help you easily locate the interrelationships between classes. All in all, the Browser is well worth mastering. To review the Browser commands, use tables 13.2 and 13.3, which show the keyboard and mouse shortcuts.

### Table 13.2. Shortcuts in the Browser

<table>
<thead>
<tr>
<th>Keyboard or Mouse Action</th>
<th>No Classes Outlined</th>
<th>Some Classes Outlined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click</td>
<td>Shows a pop-up menu of member functions</td>
<td>Same</td>
</tr>
<tr>
<td>Double-click</td>
<td>Takes you to the class declaration</td>
<td>Takes you to the code for the member function on which you originally Option-double-clicked</td>
</tr>
<tr>
<td>Enter/Return</td>
<td>Takes you to the class declaration for the selected class</td>
<td>Takes you to the code for the member function on which you originally Option-double-clicked</td>
</tr>
<tr>
<td>↑ and ↓</td>
<td>Cycles between siblings of the current class</td>
<td>Same</td>
</tr>
<tr>
<td>←</td>
<td>Selects the base class of the current class</td>
<td>Same</td>
</tr>
<tr>
<td>Keyboard or Mouse Action</td>
<td>No Classes Outlined</td>
<td>Some Classes Outlined</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>→</td>
<td>Selects one of the derived classes of the current class</td>
<td>Same</td>
</tr>
<tr>
<td>Tab</td>
<td>Cycles through all of the classes</td>
<td>Cycles through all of the outlined classes</td>
</tr>
</tbody>
</table>

**Table 13.3. Double-Clicking Shortcuts in the Editor**

<table>
<thead>
<tr>
<th>Selected Word</th>
<th>Option-Double-Click</th>
<th>Command-Option-Double-Click</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class that has a constructor</td>
<td>Takes you to the code for the constructor</td>
<td>Takes you to the class declaration</td>
</tr>
<tr>
<td>Class that doesn't have a constructor</td>
<td>Takes you to the class declaration</td>
<td>Same</td>
</tr>
<tr>
<td>Member function found in only one class</td>
<td>Takes you to the code for the member function</td>
<td>Same</td>
</tr>
<tr>
<td>Member function found in more than one class</td>
<td>Takes you to the Browser with each class with that member outlined</td>
<td>Same</td>
</tr>
<tr>
<td>Overloaded function</td>
<td>Takes you to the code for the first occurrence of the overloaded function</td>
<td>Same</td>
</tr>
</tbody>
</table>
The C++ Options Panels

Where would we be without options? Symantec, realizing that different C++ users have different needs, gives users a dialog box of options that enables the ultimate in tailoring. In the Edit menu, you find an Options dialog box that governs your C++ language settings. You learn about each option in this chapter.

Option Choices

You get your choice of five different panels in the Symantec C/C++ Options dialog box. To get to the initial Options panel, select Options from the Edit menu, as shown in figure 14.1.
Selecting Options gives you a choice of panels (see fig. 14.2).

Each of these five panels controls a different aspect of the C++ compiler:

- The Language Settings panel controls how the language is compiled.
- The Compiler Settings panel controls how the code is output.
- The Code Optimization panel determines how the output code is optimized.
- The Debugging panel designates whether debugging information is output with the code and how errors and warnings are treated.

**Figure 14.1.** Choosing Options from the Edit menu.

**Figure 14.2.** Symantec C++ Options panel choices.
• The Warning messages panel controls under which conditions the compiler will generate warning messages.

• The Prefix panel shows global variable definitions and include files.

You see in figure 14.2 that there are some other buttons in the dialog box. They perform the following actions:

• The Factory Settings button restores all the options in the panel to their default settings.

• The This Project button changes the settings for the current project.

• The New Projects button changes the settings for future projects.

• The Copy button copies settings from This Project to New Projects and vice versa.

• The ANSI Settings button makes your choices compatible with ANSI C (found only in the Language Settings panel).

**Language Settings Panel**

The first of these C++ Option panels is the Language Settings panel (see fig. 14.3). It governs the compilation of your code.

![Language Settings Panel](image)

*Figure 14.3.* The Language Settings panel.
ANSI Conformance

In almost every way, C++ is a superset of C. Traditional C, however, has a number of features added by ANSI C that are not present in C++. To deal with this situation, you have the ANSI Conformance check box. When you check this box, the C++ compiler recognizes ANSI C features. The Relaxed ANSI Conformance radio button enables all the Symantec C++ Macintosh extensions, such as the Pascal type modifier, extended option, directions, and others. The Strict ANSI Conformance button disables all those extensions. The following items should be taken into account when conforming to ANSI features:

▼ **Trigraphs.** Outside the United States, many computers use characters, such as {, }, |, and ~, for language-specific characters like é. Because the Macintosh is an international computer, it differentiates between these characters. On other computers, however, the character { may appear on-screen as é. To deal with this problem, ANSI C added alternate forms for all but the most common (alphanumeric) characters.

These alternate forms are all three-character sequences beginning with ???. For example, ??' is recognized as ^'. However, many Macintosh programs use multi-character constants that resemble trigraphs. For instance, it is common to use '????' as a file type. With trigraph processing on, this is recognized as '???' because the last three characters are read as ^'. To avoid this problem, prepend a \ to at least one of the ? characters (for example, '\?\?\?\?\?'). Adding the \ works with trigraphs on; it also works when trigraphs are off.

Normally, you only turn this feature on when porting code that contains trigraphs.

▼ **No arithmetic on pointers to functions.** ANSI C does not allow arithmetic on pointers to functions. Thus, the following code compiles only if ANSI Conformance is off:

```c
typedef void (*FuncPtr)(void);

void g(void);

void foo()
{
    FuncPtr f1 = g;
    FuncPtr f2 = g;
    long l = f1 - f2;
}
```
No trailing , in enum lists. ANSI C does not allow enumeration declarations to end in a comma. The following, therefore, does not compile if ANSI Conformance is on:

```c
enum {e1, e2, } ;
```

No binary constants. ANSI C allows decimal, octal, and hexadecimal constants, but not binary constants like the following:

```c
int i = 0b1001;
```

\x must be followed by a hex digit. Symantec C++ interprets a \x that is not followed by a hex number as an x. Because ANSI C requires that \x be followed by a hex digit, the following is illegal:

```c
char *s = "\xNN"
```

If ANSI Conformance is off, the characters in s are 'x', 'N', 'N', and '0'.

## Enums Are Always Ints

In most implementations of C and C++, enumeration variables and ints have the same size. In implementations of Pascal for the Macintosh, however, the size of an enumeration variable is 1, 2, or 4 bytes. This depends on the number of bytes needed to hold the range of values.

Here is the dilemma: If you turn the enums are always ints option on, you have better compatibility with other implementations of C and C++ but worse compatibility with Pascal for the Macintosh. One way to avoid this is to define your enumerations as the size of an integer regardless of the setting of this flag. As an example of how to do that, the following enumeration type, Color, uses 1 byte if this option is off and 4 bytes if this option is on:

```c
enum Color {red, green, blue};
```

To guarantee that it will be 4 bytes long, add an extra enumeration constant at the end. Make sure that it is large enough to force the compiler to use 4 bytes:

```c
enum Color {red, green, blue, purple, bigColor = 100000};
```

You also should know that the Macintosh Toolbox header files use a technique for their enumerations so that they are 1, 2, or 4 bytes regardless of the setting of this option. For example, they define the type GrafVerb as follows:

```c
enum {frame, paint, erase, invert, fill};
typedef unsigned char GrafVerb;
```
This defines the enumeration constants (frame, paint, erase, invert, and fill) separately from
the type GrafVerb. The declaration of GrafVerb forces it to be 1 byte. This technique works fine
in C. In C++, on the other hand, you lose some type checking. GrafVerb is now just an unsigned
char rather than an enumeration. The compiler, therefore, can no longer warn you about code
like the following:

GrafVerb myGrafVerb = red;  // should be an error

It is enough at this point to be aware of this technique and its limitations.

Read Each Header File Once

A common C/C++ convention is to bracket the contents of header files with #ifndef and #endif
preprocessor directives so that you only have to include the contents of header files once:

#ifndef MYFILE_H
#define MYFILE_H
  // actual contents of MyFile.h
#endif

If you #include MyFile.h more than once while compiling a particular file, the compiler ignores
the actual contents of MyFile.h (because the compile-time variable MYFILE_H is defined) after the
first time. You end up with a speedier compile and fewer compile errors due to the redeclaration
of types or variables. You get limited speed-up, however, because the compiler must preprocess
the header file even when it is not compiled.

If you are after real speed, it is quicker to avoid preprocessing the file more than once. Symantec
C++ provides two different ways to do this:

▼ Check the Read each header file once check box. The compiler then reads each header
file once. This option takes advantage of the common header convention by not
rereading header files that start with #ifndef and end with #endif.

▼ Use the once #pragma. If you use that pragma in a header file, the compiler ignores
successive occurrences of #include in the header file. The previous header file, rewritten
to use #pragma once, appears as follows:

#pragma once
  // actual contents of MyFile.h

You may find the pragma useful for your own header files. Be aware, however, that
there are many header files (including the Apple header files) that do not contain the
once #pragma.
Because almost all header files contain `#ifndef` and `#endif` pairs, you should normally check the Read each header file once option. This also is why it is the default.

**Treat Chars As Unsigned**

C and C++ do not determine whether char types are signed or unsigned (whether they range from −128 to 127 or from 0 to 256). Some implementations implement them as signed; others implement them as unsigned. If you want your C/C++ code properly portable, it should work regardless of which implementation you use. Because some code may assume that char types are unsigned, you need the Treat chars as unsigned option. Normally, however, you do not check it.

That is all there is to the Language Settings panel. You will find in practice that you set up your choices only once.

**Compiler Settings Panel**

The second panel you can choose is the Compiler Settings panel (see fig. 14.4).

![Compiler Settings Panel](image)

**Figure 14.4.** The Compiler Settings panel.

This panel contains options that affect how the compiler generates its code. Look at these choices in turn.
Generate 68020 Instructions

Normally, Symantec C++ generates 68000 instructions, thereby enabling your code to run on any Macintosh. If the Generate 68020 instructions option is on, the compiler generates code that also uses 68020 instructions. These instructions are present on the 68020 and higher machines, but not on the earlier machines. The upshot of this is that you get snappier performance on Macintoshes with 68020, 68030, or 68040 processors, but your code cannot run on Macintoshes with the 68000 processor.

Generate 68881 Instructions

For floating-point operations, Symantec C++ normally uses Standard Apple Numerics Environment (SANE), which is present on all Macintoshes. Many Macintoshes contain Motorola floating-point coprocessors that use the 68881 instruction set. SANE uses 68881 instructions if possible, reverting to software floating-point operations on Macintoshes without 68881 instructions. Although SANE uses the hardware to do floating-point calculations, using it as an intermediary to the hardware is not as fast as issuing 68881 instructions directly. Checking the Generate 68881 instructions option increases performance on machines with 68881 instructions (those with 68881 or 68882 coprocessors or those with 68040 processors), but your code cannot run on machines without 68881 instructions.

Use 881 for Transcendentals

For transcendental operations (like sin or cos), you get slightly different results when using the 68881 instructions than when using SANE (only in one or two of the low-order bits). When you check the Use 881 for transcendentals option, the compiler uses 68881 instructions even for transcendental operations. Use this option only if this slight difference does not matter to you. If this option is off, the compiler uses SANE for these operations.

8 Byte Doubles

Symantec C++ stores the type long double in a format that causes floating-point operations to be bullet quick. Long double types are 10 bytes long if SANE is used and 12 bytes long if 68881 instructions are used.

It is surprising, but using the long double type is faster than using shorter floating-point types (4- or 8-byte floating-point types).

If you turn 8 byte doubles off, the type double is the same size as long double and just as fast. If you turn 8 byte doubles on, you save space for double variables at the expense of slowing down calculations with them.
Write your code using the type `long double` directly if you want fast floating-point operations. You should only need to change this option if you are porting code that contains references to `double` types.

**Struct Field Alignment**

The radio buttons in the box control field location and length in structures, unions, and classes. Definitions of the choices follow:

- ▼ Align to 1 byte boundary button enables a field to start at any byte.
- ▼ Align to 2 byte boundary button enables fields of size 2 or more to begin on an even byte boundary. The size of the structure is rounded up to an even size as well.
- ▼ Align to 4 byte boundary button enables fields of size 4 or more to begin on a multiple of a 4-byte boundary. The size of the structure is rounded up to a multiple of 4 as well.

In all cases, arrays follow the alignment of their elements. Table 14.1 shows some example structures and the offsets of each field using the various options.

**Table 14.1. The Effect of Struct Field Alignment**

<table>
<thead>
<tr>
<th>Align to 1-Byte Boundary</th>
<th>Align to 2-Byte Boundary</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct s {</td>
<td>struct s {</td>
</tr>
<tr>
<td>char a; // offset 0</td>
<td>char a; // offset 0</td>
</tr>
<tr>
<td>short b; // offset 1</td>
<td>short b; // offset 2</td>
</tr>
<tr>
<td>char c[2]; // offset 3</td>
<td>char c[2]; // offset 4</td>
</tr>
<tr>
<td>}; // sizeof(s) == 5</td>
<td>}; // sizeof(s) == 6</td>
</tr>
</tbody>
</table>

struct s {              struct s {              |
| char a; // offset 0     | char a; // offset 0     |
| int i; // offset 1      | int i; // offset 2      |
| char c[2]; // offset 5  | char c[2]; // offset 6  |
| }; // sizeof(s) == 7    | }; // sizeof(s) == 8    |
Table 14.1. continued

<table>
<thead>
<tr>
<th>Align to 1-Byte Boundary</th>
<th>Align to 2-Byte Boundary</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct s {</td>
<td>struct s {</td>
</tr>
<tr>
<td>char a;</td>
<td>char a;</td>
</tr>
<tr>
<td>// offset 0</td>
<td>// offset 0</td>
</tr>
<tr>
<td>double b;</td>
<td>double b;</td>
</tr>
<tr>
<td>// offset 1</td>
<td>// offset 2</td>
</tr>
<tr>
<td>char c[2];</td>
<td>char c[2];</td>
</tr>
<tr>
<td>// offset 9</td>
<td>// offset 10</td>
</tr>
<tr>
<td>}; // sizeof(s) == 11</td>
<td>}; // sizeof(s) == 12</td>
</tr>
<tr>
<td>structure s {</td>
<td>structure s {</td>
</tr>
<tr>
<td>char a;</td>
<td>char a;</td>
</tr>
<tr>
<td>// offset 0</td>
<td>// offset 0</td>
</tr>
<tr>
<td>char b[3];</td>
<td>char b[3];</td>
</tr>
<tr>
<td>// offset 1</td>
<td>// offset 1</td>
</tr>
<tr>
<td>char c[2];</td>
<td>char c[2];</td>
</tr>
<tr>
<td>// offset 4</td>
<td>// offset 4</td>
</tr>
<tr>
<td>}; // sizeof(s) == 6</td>
<td>}; // sizeof(s) == 6</td>
</tr>
</tbody>
</table>

Selecting anything but the 2-byte default puts you in a delicate mine field. If you want to remain compatible with the Macintosh Toolbox, you need 2-byte alignments for structures and unions because the Macintosh Toolbox assumes them.

If you are writing code that does not call the Macintosh Toolbox, you can use other alignments. 1-byte alignment utilizes memory the most efficiently, but creates code that causes a fatal address error on Macintoshes with a 68000 processor. This is because the 68000 processor cannot access multi-byte values that are on odd addresses.

Using 4-byte alignment creates code that runs faster on 68020, 68030, and 68040 machines. These machines are able to access multi-byte values on any address, but run quicker with 4-byte addresses. Structures, however, take up more space.

**Place String Literals in Code**

This option defaults to off, causing string literals like “Neil” to be placed in global data space. If you check the Place string literals in code option, literals are placed in the code itself. You check this flag when you are writing stand-alone code resources (which are not able to use globals) and need to use string literals.
Code Optimization Panel

The Code Optimization panel is shown in figure 14.5. You should note that this panel has a master check box, Use Global Optimizer. If you turn this off, you get no more choices (everything else is grayed out).

![Code Optimization Panel](image)

**Figure 14.5.** The Code Optimization panel.

Dead Assignment Elimination

When you check the Dead assignment elimination box, the compiler removes assignments to variables that are not subsequently used. Following is a simple example of code:

```c
{
    int f = 3;
    int g = f;
    f = 5; // dead assignment, f never used afterwards
}
```

A slightly more intricate example follows:

```c
int DeadAssignment(int incoming)
{
    int f = incoming; // not dead assignment
    int g = f+5;
    if (incoming != 3) {
```

---

271

Chapter 14: The C++ Options Panels
\[ f = 3; \quad // \text{dead assignment} \\
g = 3; \quad // \text{not a dead assignment} \]

}\n\text{return } g \times g; \}

In this second example, the first assignment to \( f \) is not dead because \( f \) is used in the next statement. The second assignment to \( f \) is dead, however, because it never is used again. Assignments to \( g \) remain alive throughout because \( g \) is even used in the last statement.

The optimizer compiles \texttt{DeadAssignment} as if you wrote the following:

\begin{verbatim}
int DeadAssignment(int incoming)
{
    int \( f = \) incoming; \quad // \text{not dead assignment}
    int \( g = f+5; \)

    if (incoming != 3) {
        \( g = 3; \quad // \text{not a dead assignment} \)
    }
    \text{return } g \times g; \}
\end{verbatim}

To help you get a better idea of the effects of this optimization, table 14.2 shows a disassembly of the object code for \texttt{DeadAssignment} with dead assignment elimination on and off.

\textbf{Table 14.2. Effects of Dead Assignment Elimination on DeadAssignment}

<table>
<thead>
<tr>
<th>Dead Assignment Elimination Off</th>
<th>Dead Assignment Elimination On</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINK A6,$0000</td>
<td>LINK A6,$FFFFFF</td>
</tr>
<tr>
<td>MOVEM.L D3-D5,-(A7)</td>
<td></td>
</tr>
<tr>
<td>MOVE.L $0008(A6),D3</td>
<td>MOVE.L $0008(A6),$FFFF(A6)</td>
</tr>
<tr>
<td>MOVE.L D3,D4</td>
<td>MOVE.L $FFFF8(A6),D0</td>
</tr>
<tr>
<td>ADDQ.L #$5,D3</td>
<td>ADDQ.L #$5,D0</td>
</tr>
<tr>
<td>MOVE.L D3,D5</td>
<td>MOVE.L D0,$FFFFC(A6)</td>
</tr>
<tr>
<td>CMPI.L #$00000003,$0008(A6)</td>
<td>CMPI.L #$00000003,$0008(A6)</td>
</tr>
<tr>
<td>BEQ.S $+000A \quad 24</td>
<td>BEQ.S $+0008 \quad 24</td>
</tr>
</tbody>
</table>

\textit{Symantec C++ Programming for the Macintosh, 2nd Edition}
## Dead Assignment Elimination

<table>
<thead>
<tr>
<th>Dead Assignment Elimination Off</th>
<th>Dead Assignment Elimination On</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVEQ #$03,D0</td>
<td>MOVEQ #$03,D0</td>
</tr>
<tr>
<td>MOVE.L D0,D4</td>
<td></td>
</tr>
<tr>
<td>MOVEQ #$03,D0</td>
<td></td>
</tr>
<tr>
<td>MOVE.L D0,D5</td>
<td>MOVE.L D0,$FFFC(A6)</td>
</tr>
<tr>
<td>24: MOVE.L D5,D0</td>
<td>24: MOVE.L $FFFC(A6),D0</td>
</tr>
<tr>
<td>MOVE.L D0,D1</td>
<td>MOVE.L $FFFC(A6),D1</td>
</tr>
<tr>
<td>JSR ULMULT</td>
<td>JSR ULMULT</td>
</tr>
<tr>
<td>MOVEM.L (A7)+,D3-D5</td>
<td></td>
</tr>
<tr>
<td>UNLK A6</td>
<td>UNLK A6</td>
</tr>
<tr>
<td>MOVEA.L (A7)+,A0</td>
<td>MOVEA.L (A7)+,A0</td>
</tr>
<tr>
<td>ADDQ.W #$4,A7</td>
<td>ADDQ.W #$4,A7</td>
</tr>
<tr>
<td>JMP (A0)</td>
<td>JMP (A0)</td>
</tr>
</tbody>
</table>

### Dead Variable Elimination

The Dead variable elimination check box depends on two different optimizations that analyze the code in the same way. When you check the Dead variable elimination option, the optimizer calculates the live range of each automatic variable. This code range preserves the variable's value and it starts when the variable is set and ends after the last use. If a variable has an empty live range, the optimizer can eliminate it.

The optimizer also uses this computation of live range to enable variables to share space. Variables can share the same memory location if their live ranges do not overlap. The optimizer uses all of this information to determine which registers to use for which variables.

A simple example of dead variable elimination follows:

```
{  
    int f = 3;  
    g = g * 2;  
    f = 5;  
}
```
The variable \( f \) is never used (its live range is empty), so the optimizer removes it, leaving the following:

\[
\begin{align*}
\{ \\
& \quad g = g \times 2; \\
\}
\end{align*}
\]

Here is a more complicated example:

```c
short DeadVariable(short incoming)
{
  short k = 3;
  short l;

  short sum = incoming + incoming;

  k = 6; // k live

  short otherSum = k + k; // k otherSum live
  k = otherSum; // k, otherSum live
  sum = 7; // k, sum live
  return k * sum; // k, sum live
}
```

The optimizer compiles this example as if it were written as follows:

```c
short DeadVariable(short incoming)
{
  short k = 3;
  // l eliminated
  short sum;

  k = 6; // k live

  short otherSum = k + k; // k otherSum live
  k = otherSum; // k, otherSum live
  sum = 7; // k, sum live
  return k * sum; // k, sum live
}
```

In addition, `sum` and `otherSum` can share the same register.

To see how this optimization works, you can look at the disassembly of the object code for `DeadVariable`. Table 14.3 shows the disassembly with dead variable elimination on and off.
Table 14.3. Effects of Dead Variable Elimination on Dead Variable

<table>
<thead>
<tr>
<th>Dead Variable Elimination Off</th>
<th>Dead Variable Elimination On</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINK A6,#$0000</td>
<td>LINK A6,#$0000</td>
</tr>
<tr>
<td>MOVEM.L D3-D7, -(A7)</td>
<td>MOVEM.L D3-D5, -(A7)</td>
</tr>
<tr>
<td>MOVEQ #$03, D0</td>
<td>MOVEQ #$03, D3</td>
</tr>
<tr>
<td>MOVE.W D0, D4</td>
<td>MOVE.W $0008(A6), D4</td>
</tr>
<tr>
<td>MOVE.W $0008(A6), D3</td>
<td>ASL.W #$1, D4</td>
</tr>
<tr>
<td>ADD.W D3, D3</td>
<td>MOVEQ #$06, D3</td>
</tr>
<tr>
<td>MOVE.W D3, D6</td>
<td>MOVE.W D3, D5</td>
</tr>
<tr>
<td>MOVEQ #$06, D0</td>
<td>ASL.W #$1, D5</td>
</tr>
<tr>
<td>MOVE.W D0, D4</td>
<td>MOVE.Q #$07, D4</td>
</tr>
<tr>
<td>ADD.W D0, D0</td>
<td>MOVE.W D5, D3</td>
</tr>
<tr>
<td>MOVE.W D0, D7</td>
<td>MULS.W D4, D0</td>
</tr>
<tr>
<td>MOVEQ #$07, D1</td>
<td>MOVEM.L (A7)+, D3-D5</td>
</tr>
<tr>
<td>MOVE.W D1, D6</td>
<td>UNLK A6</td>
</tr>
<tr>
<td>MULS.W D1, D0</td>
<td>MOVEA.L (A7)+, A0</td>
</tr>
<tr>
<td>MOVEM.L (A7)+, D3-D7</td>
<td>ADDQ.W #$2, A7</td>
</tr>
<tr>
<td>UNLK A6</td>
<td>JMP (A0)</td>
</tr>
<tr>
<td>MOVEA.L (A7)+, A0</td>
<td></td>
</tr>
<tr>
<td>ADDQ.W #$2, A7</td>
<td></td>
</tr>
<tr>
<td>JMP (A0)</td>
<td></td>
</tr>
</tbody>
</table>

The dead variable elimination optimization works best if you also set Dead assignment elimination. The dead assignment elimination removes assignments; this decreases the live ranges of variables.
Dead variable elimination is one of the most powerful optimizations you have at your disposal because it controls variable placement into registers. Having variables in registers, rather than on the stack, significantly increases the speed of access. This is especially true if you access variables in a loop.

CSE Elimination

When you select CSE (common subexpression) elimination, the optimizer removes the computations of redundant expressions. If you repeat an expression in your source code, the CSE elimination option calculates that expression once, saves it in a temporary, and then uses the precalculated value anywhere the original expression occurs. Following is an example function that repeats the expression $k \times (k+2)$:

```cpp
short Foo(short j, short k)
{
    short i = j + k * (k+2);
    short m = k * (k+2);

    return i * m;
}
```

CSE elimination computes the value of $k \times (k+2)$ once rather than twice. The code compiles as if it were written as follows:

```cpp
short FooConverted(short j, short k)
{
    short tmp = k * (k+2);
    short i = j + tmp;
    short m = tmp;

    return i * m;
}
```

Table 14.4 shows a disassembly of the object code for `Foo` with CSE elimination on and off. Italic type shows the important differences.
Table 14.4. Effects of Common Subexpression Elimination on Foo

<table>
<thead>
<tr>
<th>CSE Elimination Off</th>
<th>CSE Elimination On</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINK A6,#$0000</td>
<td>LINK A6,#$FFF8</td>
</tr>
<tr>
<td>MOVEM.L D3-D5,-(A7)</td>
<td>MOVE.L D3,-(A7)</td>
</tr>
<tr>
<td>MOVE.W $0008(A6),D3</td>
<td>MOVE.W $0008(A6),D3</td>
</tr>
<tr>
<td>ADDQ.W #$2,D3</td>
<td>ADDQ.W #$2,D3</td>
</tr>
<tr>
<td>MULS.W $0008(A6),D3</td>
<td>MULS.W $0008(A6),D3</td>
</tr>
<tr>
<td>MOVE.W $0008(A6),D0; recalculate CSE</td>
<td>MOVE.W $FFFC(A6),D0; restore CSE</td>
</tr>
<tr>
<td>ADD.W $000A(A6),D3</td>
<td>ADD.W $000A(A6),D3</td>
</tr>
<tr>
<td>MOVE.W D3,D4</td>
<td>MOVE.W D3,$FFF8(A6)</td>
</tr>
<tr>
<td>MOVE.W $0008(A6),D0; recalculate CSE</td>
<td>MOVE.W $FFFC(A6),D0; restore CSE</td>
</tr>
<tr>
<td>ADDQ.W #$2,D0</td>
<td>ADDQ.W #$2,D0</td>
</tr>
<tr>
<td>MULS.W $0008(A6),D0</td>
<td>MULS.W $0008(A6),D0</td>
</tr>
<tr>
<td>MOVE.W D0,D5</td>
<td>MOVE.W D0,$FFFA(A6)</td>
</tr>
<tr>
<td>MOVE.W D3,D0</td>
<td>MOVE.W D3,D0</td>
</tr>
<tr>
<td>MULS.W D5,D0</td>
<td>MULS.W $FFFA(A6),D0</td>
</tr>
<tr>
<td>MOVEM.L (A7)+,D3-D5</td>
<td>MOVE.L (A7)+,D3</td>
</tr>
<tr>
<td>UNLK A6</td>
<td>UNLK A6</td>
</tr>
<tr>
<td>MOVEA.L (A7)+,A0</td>
<td>MOVEA.L (A7)+,A0</td>
</tr>
<tr>
<td>ADDQ.W #$4,A7</td>
<td>ADDQ.W #$4,A7</td>
</tr>
<tr>
<td>JMP (A0)</td>
<td>JMP (A0)</td>
</tr>
</tbody>
</table>

The example with $k \times (k+2)$ is one where the common subexpression is very evident. They can be much less evident, however:

```c
struct MyStruct {
    char c;
    short ints[10];
};
```
Here, you get the calculation of the \texttt{ints} field in \texttt{s} done twice. CSE elimination compiles the code as if it were written as follows:

```c
void BarConverted(MyStruct *s) {
    for (int i = 0; i < 9; i++) {
        int *intPtr;
        intPtr = &s->ints[i];
        intPtr[0] = intPtr[1];
    }
}
```

Table 14.5 shows a disassembly of the object code for \texttt{Bar} with CSE elimination on and off. Italic type shows the important differences.

**Table 14.5. Effects of Common Subexpression Elimination on \texttt{Bar}**

<table>
<thead>
<tr>
<th>CSE Elimination Off</th>
<th>CSE Elimination On</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINK A6,$0000</td>
<td>LINK A6,$FFFC</td>
</tr>
<tr>
<td>MOVEQ $00,D0</td>
<td>MOVEQ $00,D0</td>
</tr>
<tr>
<td>MOVE.L D0,D4</td>
<td>\textbf{0E: MOVE.L D0,$FFFC(A6)}</td>
</tr>
<tr>
<td>0C: MOVE.L D4,D3</td>
<td>MOVE.L $FFFC(A6),D3</td>
</tr>
<tr>
<td>MOVEQ $09,D0</td>
<td>MOVEQ $09,D0</td>
</tr>
<tr>
<td>CMP.L D0,D3</td>
<td>CMP.L D0,D3</td>
</tr>
<tr>
<td>BGE.S *+$0026; 0038</td>
<td>BGE.S *+$001A ; 0030</td>
</tr>
</tbody>
</table>

; calculate CSE \&s->ints[0] ; calculate CSE \&s->ints[0]

```c
ADDQ.L \$1,D3
```

```c
MOVEQ \$01,D4
```
<table>
<thead>
<tr>
<th>CSE Elimination Off</th>
<th>CSE Elimination On</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASL.L #$1,D3</td>
<td>ASL.L D4,D3</td>
</tr>
<tr>
<td>ADDQ.W #$2,A2</td>
<td>MOVEA.L D3,A2</td>
</tr>
<tr>
<td>ADD.L A2,D3</td>
<td></td>
</tr>
<tr>
<td>MOVEA.L D3,A0</td>
<td></td>
</tr>
</tbody>
</table>

; calculate CSE &s->ints[1]
MOVE.W (A0),D0      MOVE.W $0004(A2),D0

; calculate CSE &s->ints[0]
MOVE.L D4,D1        MOVE.L D1,A0
ASL.L #$1,D1        ASL.L 04,03
MOVEA.L $0008(A6),A0 MOVEA.L 03,A2
ADDQ.W #$2,A0       ADDQ.W #$4,A7
ADD.L A0,D1         JMP (A0)
MOVEA.L D1,A0       MOVEA.L (A7)+,A0

; save into s->ints[0]          ; save into s->ints[0]
MOVE.W D0,(A0)        MOVE.W D0,$0002(A2)
ADDQ.L #$1,D4         ADDQ.L #$1,$FFFC(A6)
BRA.S *-$002A; 000C    BRA.S *-$0020 ; 000E

UNLK A6               UNLK A6
MOVEA.L (A7)+,A0      MOVEA.L (A7)+,A0
ADDQ.W #$4,A7         ADDQ.W #$4,A7
JMP (A0)              JMP (A0)
CSE elimination can make your code both faster and smaller by eliminating redundant calculation.

**Hoist Very Busy Expressions**

This optimization is a space-saving feature. It takes code that is duplicated in more than one execution path and moves it before the branch. Although it does not save time, it should save code space. The following code, for example, calculates the same expression in both the if and else branches:

```c
int Hoist(int bool, short b, short c)
{
    short a;
    short d;

    if (bool) {
        a = b*b + 2*b*c + c*c;
    }
    else {
        d = b*b + 2*b*c + c*c;
    }
    return a * d;
}
```

With the Hoist very busy expressions option selected, this compiles as if it were written as follows:

```c
int Hoist(int bool, short b, short c)
{
    short a;
    short d;

    short tmp = b*b + 2*b*c + c*c;
    if (bool) {
        a = tmp;
    }
    else {
        d = tmp;
    }
    return a * d;
}
```
Unfortunately, at the time this book went to press, this example incorrectly compiled as if it were as originally written.

This version of the optimization does not save any space and executes just a bit more slowly. Because of this, you might not want to use this optimization unless you are sure it works correctly in your version of the software. Table 14.6 shows a disassembly of the object code for Hoist with the Hoist very busy expressions option on and off.

Table 14.6. Effects of Hoist Very Busy Expressions on Hoist

<table>
<thead>
<tr>
<th>Hoist Very Busy Expressions Off</th>
<th>Hoist Very Busy Expressions On</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINK A6,#$0000</td>
<td>LINK A6,#$FFFFC</td>
</tr>
<tr>
<td>MOVEM.L D3-D6,-(A7)</td>
<td></td>
</tr>
<tr>
<td>MOVE.L $0000C(A6),D0</td>
<td>MOVE.L $0000C(A6),D0</td>
</tr>
<tr>
<td>BEQ.S *+$0020; 002C</td>
<td>BEQ.S *+$0026; 002E</td>
</tr>
</tbody>
</table>

; a = b*b + 2*b*c + c*c

MOVE.W $0008(A6),D3
MULS.W D3,D3
ADD.W D0,D4
ADD.W D4,D3
MOVE.W D3,D5 store in d5
BRA.S *+$001E; 0048
2C: MOVE.W $0008(A6),D3

; a = b*b + 2*b*c + c*c

MOVE.W $0000A(A6),D0
ASL.W #$1,D0
MULS.W $0008(A6),D0
MULS.W $0008(A6),D1
MOVE.W $0000A(A6),D0
ASL.W #$1,D0
MULS.W $0000A(A6),D1
MULS.W $0000A(A6),D2
ADD.W D0,D4
ADD.W D4,D3
ADD.W D1,D0
MOVE.W D0,$FFFC(A6)
BRA.S *+$0024; 0050
2E: MOVE.W $0000A(A6),D0

ASL.W #$1,D0
MULS.W D3,D3
MULS.W $0008(A6),D0
MOVE.W $0000A(A6),D1
### Table 14.6. continued

<table>
<thead>
<tr>
<th>Hoist Very Busy Expressions Off</th>
<th>Hoist Very Busy Expressions On</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULS.W D4, D4</td>
<td>MULS.W $0008(A6), D1</td>
</tr>
<tr>
<td>MOVE.W $000A(A6), D0</td>
<td>MOVE.W $000A(A6), D2</td>
</tr>
<tr>
<td>ASL.W #$1, D0</td>
<td></td>
</tr>
<tr>
<td>MULS.W $0008(A6), D0</td>
<td>MULS.W $000A(A6), D2</td>
</tr>
<tr>
<td>ADD.W D0, D4</td>
<td>ADD.W D2, D1</td>
</tr>
<tr>
<td>ADD.W D4, D3</td>
<td>ADD.W D1, D0</td>
</tr>
<tr>
<td>MOVE.W D3, D6</td>
<td>MOVE.W D0, $FFFE(A6)</td>
</tr>
<tr>
<td>48: MOVE.W D5, D0</td>
<td>50: MOVE.W $FFFC(A6), D0</td>
</tr>
<tr>
<td></td>
<td>EXT.L D0</td>
</tr>
<tr>
<td></td>
<td>MOVE.W $FFFE(A6), D1</td>
</tr>
<tr>
<td></td>
<td>EXT.L D1</td>
</tr>
<tr>
<td>MULS.W D6, D0</td>
<td>JSR ULMULT</td>
</tr>
<tr>
<td>MOVM. L (A7)+, D3-D6</td>
<td></td>
</tr>
<tr>
<td>UNLK A6</td>
<td>UNLK A6</td>
</tr>
<tr>
<td>MOVEA. L (A7)+, A0</td>
<td>MOVEA. L (A7)+, A0</td>
</tr>
<tr>
<td>ADDQ.W #$8, A7</td>
<td>ADDQ.W #$8, A7</td>
</tr>
<tr>
<td>JMP (A0)</td>
<td>JMP (A0)</td>
</tr>
</tbody>
</table>

### Remove Loop Invariants

Loop invariants are expressions that do not get changed during the evaluation of a loop. Rather than evaluating the invariant each time through the loop, the optimizer evaluates the invariant once, before the loop. The optimizer, therefore, moves invariant expressions out of the loop but does not delete them entirely.

Unlike some other choices, this optimization is different depending on whether you are optimizing for space or for time. Look at how each acts in turn.
Optimizing for Space

It is reasonably straightforward to remove loop invariants when optimizing for space. The optimizer just moves the invariant expressions before the loop. The form of the code is as follows:

```plaintext
loop {
    code using an invariant expression
}
```

This is transformed into the following:

```plaintext
temporary = invariant expression;
loop {
    code using temporary
}
```

To make this clearer, look at a specific example:

```plaintext
void Loop(register char *s, short i, short j)
{
    char c;
    while (((c = *s++) != '0'))
        func(i*j + c);
}
```

Here, \(i \times j\) is evaluated every time through the loop, even though it always has the same value. By removing loop invariants, the compiler continues as if the following were the code:

```plaintext
void LoopConverted(register char *s, short i, short j)
{
    char c;
    short temporary = i*j;
    while (((c = *s++) != '0'))
        func(temporary + c);
}
```

Table 14.7 shows a disassembly of the object code for Loop with the Remove loop invariants option on and off.
### Table 14.7. Effects of Remove Loop Invariants on Loop with Space Optimization

<table>
<thead>
<tr>
<th>Remove Loop Invariants Off</th>
<th>Remove Loop Invariants On</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINK A6,#$FFFC</td>
<td>LINK A6,#$FFFE</td>
</tr>
<tr>
<td>; tmp = i*j</td>
<td>; tmp = i*j</td>
</tr>
<tr>
<td>MOVE.W $000A(A6),D0</td>
<td>MOVE.W $000A(A6),D0</td>
</tr>
<tr>
<td>EXT.L D0</td>
<td>EXT.L D0</td>
</tr>
<tr>
<td>MOVE.W $0008(A6),D1</td>
<td>MOVE.W $0008(A6),D1</td>
</tr>
<tr>
<td>EXT.L D1</td>
<td>EXT.L D1</td>
</tr>
<tr>
<td>JSR ULMULT</td>
<td>JSR ULMULT</td>
</tr>
<tr>
<td>MOVE.L D0,$FFFC(A6)</td>
<td>MOVE.L D0,$FFFC(A6)</td>
</tr>
<tr>
<td>BRA.S *+$0014; 002C</td>
<td>BRA.S *+$0014; 002C</td>
</tr>
<tr>
<td>; calculate func(tmp + c);</td>
<td>; calculate func(tmp + c);</td>
</tr>
<tr>
<td>1A: MOVE.B $FFF8(A6),D0</td>
<td>1A: MOVE.B $FFF8(A6),D0</td>
</tr>
<tr>
<td>EXT.W D0</td>
<td>EXT.W D0</td>
</tr>
<tr>
<td>EXT.L D0</td>
<td>EXT.L D0</td>
</tr>
<tr>
<td>ADD.L $FFFC(A6),D0</td>
<td>ADD.L $FFFC(A6),D0</td>
</tr>
<tr>
<td>MOVE.L D0,*(A7)</td>
<td>MOVE.L D0,*(A7)</td>
</tr>
<tr>
<td>JSR func(int)</td>
<td>JSR func(int)</td>
</tr>
<tr>
<td>; start loop</td>
<td>; start loop</td>
</tr>
<tr>
<td>04: MOVEA.L $000C(A6),A0</td>
<td>2C: MOVEA.L $000C(A6),A0</td>
</tr>
<tr>
<td>ADDQ.L #$1,$000C(A6)</td>
<td>ADDQ.L #$1,$000C(A6)</td>
</tr>
<tr>
<td>MOVE.B (A0),D0</td>
<td>MOVE.B (A0),D0</td>
</tr>
<tr>
<td>MOVE.B D0,$FFFC(A6)</td>
<td>MOVE.B D0,$FFFC(A6)</td>
</tr>
<tr>
<td>TST.B D0</td>
<td>TST.B D0</td>
</tr>
<tr>
<td>BEQ.S *+$001A; 002E</td>
<td>BNE.S *-$0022; 001A</td>
</tr>
<tr>
<td>; calculate func(i*j + c)</td>
<td>; calculate func(i*j + c)</td>
</tr>
<tr>
<td>16: MOVE.W $000A(A6),D1</td>
<td>16: MOVE.W $000A(A6),D1</td>
</tr>
<tr>
<td>MULS.W $0008(A6),D1</td>
<td>MULS.W $0008(A6),D1</td>
</tr>
</tbody>
</table>
Surprisingly, there are cases where this optimization can make your code run slower. When the loop body is never executed (the loop condition starts out false) and optimization is off, the compiler does not evaluate the invariant expression. On the other hand, when optimization is on, the invariant is evaluated even when the loop body is never executed. Because of this case, the optimizer acts differently when optimizing for time.

### Optimizing for Time

When you optimize for time, the process is not as straightforward. In this case, the optimizer must evaluate invariant expressions only if the loop body executes one or more times. When the loop body does not execute, the optimizer should not evaluate the invariant expressions. Look at the following code to see how the optimizer satisfies this requirement:

```c
loop {
    code using an invariant expression
}
```

This code is transformed into the following:

```c
if (loop condition) {
    temporary = invariant expression;
    loop {
```
Look at a specific example:

```c
void Loop(register char *s, short i, short j)
{
    char c;

    while ((c = *s++) != '\0')
        func(i*j + c);
}
```

Here, \texttt{i*j} is evaluated every time through the loop, even though it always has the same value. Checking the Remove loop invariants option causes the code to compile as if it looked like the following:

```c
void LoopConverted(register char *s, short i, short j)
{
    char c;

    if ((c = *s++) != '\0') {
        short temporary = i*j;

        do {
            func(temporary + c);
        } while ((c = *s++) != '\0');
    }
}
```

With this transformation, the compiler evaluates \texttt{i*j} only if the first character of \texttt{s} is not \texttt{\0}. If the first character is \texttt{\0}, the loop body and the evaluation of \texttt{i*j} are not executed. Table 14.8 shows a disassembly of the object code for \texttt{Loop} with the Remove loop invariants option on and off. Italic type shows the important differences.

You should note that, even though the optimized code has become larger, the run time is still faster in multiple loop executions. It is just as fast as the original code with a single loop execution.
### Table 14.8. Effects of Remove Loop Invariants on Loop with Time Optimization

<table>
<thead>
<tr>
<th>Remove Loop Invariants Off</th>
<th>Remove Loop Invariants On</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link A6,#$FFFC</td>
<td>Link A6,#$FF8</td>
</tr>
<tr>
<td>04: Movea.l $000C(A6),A0</td>
<td>Movea.l $000C(A6),A0</td>
</tr>
<tr>
<td>Addq.l #$1,$000C(A6)</td>
<td>Addq.l #$1,$000C(A6)</td>
</tr>
<tr>
<td>Move.b (A0),D0</td>
<td>Move.b (A0),D0</td>
</tr>
<tr>
<td>Move.b D0,$FFFC(A6)</td>
<td>Move.b D0,$FF8(A6)</td>
</tr>
<tr>
<td>Tst.b D0</td>
<td>Tst.b D0; if ((c = *s++) != '\0') {}</td>
</tr>
<tr>
<td>Beq.s *+$0018; 002C</td>
<td>Beq.s *+$0030; 0044</td>
</tr>
<tr>
<td>Move.w $000A(A6),D1</td>
<td>Move.w $000A(A6),D0</td>
</tr>
<tr>
<td>Muls.w $0008(A6),D1; i * j</td>
<td>Muls.w $0008(A6),D0; temp = i * j</td>
</tr>
<tr>
<td>Move.b D0,D2</td>
<td>Move.b $FF8(A6),D0</td>
</tr>
<tr>
<td>Ext.w D2</td>
<td>Ext.w D0</td>
</tr>
<tr>
<td>Add.w D2,D1</td>
<td>Add.w $FFFC(A6),D0</td>
</tr>
<tr>
<td>Move.w D1,-(A7)</td>
<td>Move.w D0,-(A7)</td>
</tr>
<tr>
<td>Jsr func(short)</td>
<td>Jsr func(short)</td>
</tr>
<tr>
<td>Movea.l $000C(A6),A0</td>
<td>Movea.l $000C(A6),A0</td>
</tr>
<tr>
<td>Addq.l #$1,$000C(A6)</td>
<td>Addq.l #$1,$000C(A6)</td>
</tr>
<tr>
<td>Move.b (A0),D0</td>
<td>Move.b (A0),D0</td>
</tr>
<tr>
<td>Move.b D0,$FF8(A6)</td>
<td>Move.b D0,$FF8(A6)</td>
</tr>
<tr>
<td>Tst.b D0; } while(...)</td>
<td></td>
</tr>
<tr>
<td>Bra.s *-$0026; 0004</td>
<td>Bne.s *-$0020; 0022</td>
</tr>
<tr>
<td>2c: Unlk A6</td>
<td>44: Unlk A6</td>
</tr>
<tr>
<td>Movea.l (A7)+,A0</td>
<td>Movea.l (A7)+,A0</td>
</tr>
<tr>
<td>Addq.w #$8,A7</td>
<td>Addq.w #$8,A7</td>
</tr>
<tr>
<td>Jmp (A0)</td>
<td>Jmp (A0)</td>
</tr>
</tbody>
</table>
Create Loop Induction Variables

When you select the Create loop induction variables option, the optimizer searches for particular expressions. These are expressions in a loop whose values arithmetically progress as the loop iterates. The optimizer uses addition to evaluate them through each iteration of the loop.

First, look at a simple example and then a more complex example of using loop induction variables. You normally find these arithmetic progressions when an expression references a looping variable and derives from addition or multiplication. A simple example is the following:

```c
for (i = 0; i < 10; i++)
{
    j = i * 3 + 5;
    ...
}
```

This converts to the following:

```c
for (i = 0, j = 5; i < 10; i++, j += 3)
{
    ...
}
```

In each case, j takes on these values: 5, 8, 11, 14, 17, 20, 23, 26, 29, and 32. In the unoptimized case, each iteration through the loop requires two operations to compute j: one addition and one multiplication. In the optimized case, each iteration requires only one addition to compute j.

The optimizer replaces a loop variable with a loop induction variable if all the following are true:

- The loop body does not use the value of the looping variable.
- The loop variable value is not used after the loop.
- Dead variable elimination is on.

For instance, if the loop is as follows:

```c
for (i = 0; i < 10; i++)
{
```

In this case, the loop body is not using the value of `i` and the value of `i` is not used after the loop, so the optimizer will replace it with a loop induction variable.

```c
for (i = 0, j = 0; i < 10; i++, j += 3)
{
    ...
}
```
\[ j = i \times 3 + 5; \]
\[ \text{Func}(j); \]
\]

It is converted to the following:

```c
for (j = 5; j < 35; j += 3)
{
    \text{Func}(j);
}
```

Consider a more complex example of loop optimization. There is a whole class of expressions whose arithmetic progress is not obvious. This optimization is especially helpful in these cases. For instance, array access in a loop can often form an arithmetic progression.

Look at an example with arrays:

```c
void \text{ZeroArray}(\text{long } \star \text{longArray})
{
    \text{for (int } i = 0; i < 10; i++)
    \{ \text{longArray}[i] = 0; \}
}
```

The expression `longArray[i]` forms an arithmetic progression as the loop iterates. Each time through the loop, the calculated address increases by four bytes (the size of a `long`). With loop induction on, the loop can be transformed into the following:

```c
void \text{ZeroArray}(\text{long } \star \text{longArray})
{
    \text{for (int } i = 0, \text{long } \star \text{tmp} = \&\text{longArray}[0]; i < 10; i++, \text{tmp++})
    \{ \text{*tmp} = 0; \}
}
```

In the revised version, rather than incrementing \( i \) by one each time through the loop, the compiler increments \( \text{tmp} \) by four (because \( ++ \) increments pointers by the size of the object to which they point).

Table 14.9 shows a disassembly of the object code for \text{ZeroArray} with Create loop induction variables on and off. Italic type shows the loop.
Table 14.9. Effects of Create Loop Induction Variables on ZeroArray

<table>
<thead>
<tr>
<th>Create Loop Induction Variables Off</th>
<th>Create Loop Induction Variables On</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINK A6,#$0000</td>
<td>LINK A6,#$FFF8</td>
</tr>
<tr>
<td>MOVEM.L D3/D4, -(A7)</td>
<td>MOVEM.L</td>
</tr>
<tr>
<td>MOV EQ #$00,D0</td>
<td>MOV EQ #$00,D0</td>
</tr>
<tr>
<td>MOVE.L D0,D4</td>
<td>MOVE.L</td>
</tr>
<tr>
<td>0C: MOVE.L D4,D3</td>
<td>MOVE.L</td>
</tr>
<tr>
<td>MOVEQ #$0A,D0</td>
<td>MOVE.L D0,$FFF8(A6)</td>
</tr>
<tr>
<td>CMP.L D0,D3</td>
<td>CMP.I #$0000000A,$FFF8(A6)</td>
</tr>
<tr>
<td>BGE.S +$0012; 0024</td>
<td>BGE.S +$002A ; 003C</td>
</tr>
<tr>
<td>MOVEQ #$00,D0</td>
<td>MOVE.L $FFF8(A6),D0</td>
</tr>
<tr>
<td>ASL.L #$2,D3</td>
<td>ASL.L #$2,D0</td>
</tr>
<tr>
<td>ADD.L $0008(A6),D3</td>
<td>ADD.L $0008(A6),D0</td>
</tr>
<tr>
<td>MOVEA.L D3,A0</td>
<td>MOVE.L D0,$FFFC(A6)</td>
</tr>
<tr>
<td>22: MOVEQ #$00,D0</td>
<td>MOVEA.L $FFFC(A6),A0</td>
</tr>
<tr>
<td>MOVE.L D0,(A0)</td>
<td>MOVE.L D0,(A0)</td>
</tr>
<tr>
<td>ADDQ.L #$4,$FFFC(A6)</td>
<td>ADDQ.L #$4,$FFFC(A6)</td>
</tr>
<tr>
<td>ADDQ.L #$1,D4</td>
<td>ADDQ.L #$1,$FFF8(A6)</td>
</tr>
<tr>
<td>CMP.I #$0000000A,$FFF8(A6)</td>
<td>CMP.I</td>
</tr>
<tr>
<td>BRA.S *-$0016; 000C</td>
<td>BLT.S *-$0018; 0022</td>
</tr>
<tr>
<td>24: MOVEM.L (A7)+,D3/D4</td>
<td>MOVEM.L</td>
</tr>
<tr>
<td>UNLK A6</td>
<td>UNLK A6</td>
</tr>
<tr>
<td>MOVEA.L (A7)+,A0</td>
<td>MOVEA.L</td>
</tr>
<tr>
<td>ADDQ.W #$4,A7</td>
<td>ADDQ.W</td>
</tr>
<tr>
<td>JMP (A0)</td>
<td>JMP (A0)</td>
</tr>
</tbody>
</table>
The optimized version of ZeroArray is slightly longer because the optimization is for time, not space. However, the inner loop is only seven instructions long, rather than the nine of the unoptimized case. The total running time of the optimized version, therefore, is still faster.

**Constant Propagation**

Constant propagation substitutes variables that have constant values with the constant values themselves. This replacement transfers these calculations from run time to compile time.

Here is an example:

```cpp
int BytesNeeded()
{
    int numElements;
    int elementSize = sizeof(long); // equals 4

    if (elementSize > 4)
        numElements = 10;
    else
        numElements = 20;
    return numElements * elementSize;
}
```

In BytesNeeded, `elementSize` has a constant value (4 with this compiler). Look at how constant propagation transforms the code:

```cpp
int BytesNeededConverted1()
{
    int numElements;
    int elementSize = sizeof(long); // equals 4

    if (4 > 4)
        numElements = 10;
    else
        numElements = 20;
    return numElements * 4;
}
```

If you also have checked the Loop until can’t optimize option (covered shortly), the optimizer continues using constant propagation to make the following changes to this code. The optimizer calculates that `4 > 4` is false, resulting in the following:

```cpp
int BytesNeededConverted2()
{
```
int numElements;
int elementSize = sizeof(long); // equals 4

numElements = 20;
return numElements * 4;
}

Now numElements has a constant value of 20 after the assignment. Further optimization yields the following:

int BytesNeededConverted3()
{
    int numElements;
    int elementSize = sizeof(long); // equals 4

    numElements = 20;
    return 20 * 4;
}

Finally, you get this:

int BytesNeededConverted4()
{
    return 80;
}

Table 14.10 shows some disassembled code with the Constant propagation option turned on and off for the BytesNeeded function.

<table>
<thead>
<tr>
<th>Constant Propagation Off</th>
<th>Constant Propagation On</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVEM.L D3/D4,-(A7)</td>
<td>LINK A6,$FFFF8</td>
</tr>
<tr>
<td>MOVEQ #$04,D0</td>
<td>MOVEQ #$04,D0</td>
</tr>
<tr>
<td>MOVE.L D0,D4</td>
<td>MOVE.L D0,$FFFC(A6)</td>
</tr>
<tr>
<td>MOVEQ #$04,D1</td>
<td>MOVEQ #$14,D0</td>
</tr>
<tr>
<td>CMP.L D1,D0</td>
<td>MOVE.L D0,$FFFF8(A6)</td>
</tr>
<tr>
<td>BLE.S *+$0008; 0014</td>
<td>MOVEQ #$50,D0</td>
</tr>
<tr>
<td>MOVEQ #$0A,D0</td>
<td>UNLK A6</td>
</tr>
</tbody>
</table>
When you check the Constant propagation option, the optimizer calculates when variables hold constant values. You could have provided this same information more directly by using the const keyword. If you are feeling energetic, you can rewrite `BytesNeeded` to use `const` as follows:

```c
int BytesNeeded()
{
    int numElements;
    const int elementSize = sizeof(long); // equals 4

    if (elementSize > 4)
        numElements = 10;
    else
        numElements = 20;
    return numElements * elementSize;
}
```

**NOTE**

`numElements` is not declared `const` because its value depends on the if statement. You could use code like the following:

```c
const int numElements = (elementSize > 4) ? 10 : 20;
```

This could easily become ridiculous, however.
When you use const in your coding, you liberate the compiler from the optimizer; you, not the optimizer, determine which variables hold constant values. When, for example, `elementSize` is declared `const`, with copy propagation off, the code compiles as if it were written as follows:

```cpp
int BytesNeeded()
{
    int numElements;

    numElements = 10;
    return numElements * 4;
}
```

The resulting disassembled code, though not optimal, is better than it would be without the const keyword:

```
LINK       A6,#$0000
MOVE.L    D3,(A7)
MOVEQ   #$14,D3
MOVE.L    D3,D0
ASL.L    #$2,D0
MOVE.L    $FFFC(A6),D3
UNLK       A6
RTS
```

Normally, you should keep this option on. But do not use it as a substitute for declaring constant variables const.

**Copy Propagation**

The Copy propagation option is supposed to track when one variable is assigned to another. Although they have the same value, the compiler uses the first variable for both. Suppose that you have the following code:

```cpp
int TestCopyPropagation(int incoming)
{
    int i = incoming;
    int j = i;
    int r = r + 1;
    return j;
}
```
When it is compiled, it is written as if it were the following:

```c
int TestCopyPropagation(int incoming)
{
    int i = incoming;
    int j = i;
    int r = r + 1;
    return i;
}
```

Table 14.11 shows the effect of Copy Propagation on and off for the TestCopyPropagation function.

**Table 14.11. Effects of Copy Propagation on TestCopyPropogation**

<table>
<thead>
<tr>
<th>Copy Propagation Off</th>
<th>Copy Propagation On</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINK A6,#$0000</td>
<td>LINK A6,#$FFF4</td>
</tr>
<tr>
<td>MOVEM.L D3-D5,-(A7)</td>
<td></td>
</tr>
<tr>
<td>MOVE.L $0008(A6),D0</td>
<td>MOVE.L $0008(A6),D0</td>
</tr>
<tr>
<td>MOVE.L D0,D3</td>
<td>MOVE.L D0,$FFF4(A6)</td>
</tr>
<tr>
<td>MOVE.L D0,D4</td>
<td>MOVE.L D0,$FFF8(A6)</td>
</tr>
<tr>
<td>MOVE.L D5,D1</td>
<td></td>
</tr>
<tr>
<td>ADDQ.L #$1,D1</td>
<td>ADDQ.L #$1,$FFFC(A6)</td>
</tr>
<tr>
<td>MOVE.L D1,D5</td>
<td>MOVE.L $FFF8(A6),D0</td>
</tr>
<tr>
<td>MOVEM.L (A7)+,D3-D5</td>
<td></td>
</tr>
<tr>
<td>UNLK A6</td>
<td>UNLK A6</td>
</tr>
<tr>
<td>MOVEA.L (A7)+,A0</td>
<td>MOVEA.L (A7)+,A0</td>
</tr>
<tr>
<td>ADDQ.W #$4,A7</td>
<td>ADDQ.W #$4,A7</td>
</tr>
<tr>
<td>JMP (A0)</td>
<td>JMP (A0)</td>
</tr>
</tbody>
</table>
Loop Until Can’t Optimize

When you need the compiler to optimize your code with the unfailing loyalty of a collie dog, you should check the Loop until can’t optimize option. If you do, the optimizer repetitively runs through your code implementing all possible optimizations, including those that depend on preceding ones. The optimizer stops when improvements are no longer possible. If you do not check this option, the compiler runs through the whole suite of optimizations only once. As you might expect, using this option takes longer but produces better code. Look at the following sample:

```c
void ZeroArray(long *longArray) {
    for (int i = 0; i < 10; i++)
        longArray[i] = 0;
}
```

Table 14.12 shows you the effect of using the Loop until can’t optimize option on this function.

**Table 14.12. Effects of Loop Until Can’t Optimize on ZeroArray**

<table>
<thead>
<tr>
<th>Loop Until Can’t Optimize Off</th>
<th>Loop Until Can’t Optimize On</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINK A6,#$0000</td>
<td>LINK A6,#$0000</td>
</tr>
<tr>
<td>MOVEM.L D3/D4,-(A7)</td>
<td>MOVEM.L A2/A3,-(A7)</td>
</tr>
<tr>
<td>MOVEQ #$00,D0</td>
<td>MOVEA.L $0008(A6),A2</td>
</tr>
<tr>
<td>MOVE.L D0,D4</td>
<td>MOVEA.L $0008(A6),A3</td>
</tr>
<tr>
<td>0C: MOVE.L D4,D3</td>
<td>MOVEQ #$28,D0</td>
</tr>
<tr>
<td>MOVEQ #$0A,D0</td>
<td>ADDA.L D0,A3</td>
</tr>
<tr>
<td>CMP.L D0,D3</td>
<td>14: MOVEM.L D0,(A2)</td>
</tr>
<tr>
<td>BGE.S *+$0012; 0024</td>
<td>MOVEQ #$00,D0</td>
</tr>
<tr>
<td>MOVEM.L D3,A0</td>
<td>ADDQ.W #$4,A2</td>
</tr>
<tr>
<td>ASL.L #$2,D3</td>
<td>CMPA.L A3,A2</td>
</tr>
<tr>
<td>ADD.L $0008(A6),D3</td>
<td>BLT.S *-$0008; 0014</td>
</tr>
<tr>
<td>MOVEA.L D3,A0</td>
<td>MOVEM.L (A7)+,A2/A3</td>
</tr>
<tr>
<td>MOVE.L D0,(A0)</td>
<td>UNLK A6</td>
</tr>
<tr>
<td>ADDQ.L #$1,D4</td>
<td>MOVEA.L (A7)+,A0</td>
</tr>
</tbody>
</table>
By turning the flag on, the optimizer devises more improvements, giving you smaller and faster code.

## The Debugging Panel

The C++ Options Debugging panel is shown in figure 14.6.

<table>
<thead>
<tr>
<th>Loop Until Can't Optimize Off</th>
<th>Loop Until Can't Optimize On</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRA.S</td>
<td>ADDQ.W</td>
</tr>
<tr>
<td>*-$0016; 000C</td>
<td>#$4,A7</td>
</tr>
<tr>
<td>24: MOVEM.L (A7)+,D3/D4</td>
<td>JMP (A0)</td>
</tr>
<tr>
<td>UNLK A6</td>
<td></td>
</tr>
<tr>
<td>MOVEA.L (A7)+,A0</td>
<td></td>
</tr>
<tr>
<td>ADDQ.W #$4,A7</td>
<td></td>
</tr>
<tr>
<td>JMP (A0)</td>
<td></td>
</tr>
</tbody>
</table>

There are several different debugging options you can specify in this panel:

- Generating stack frames.
- Generating profiler calls.

---

Figure 14.6. The C++ Options Debugging panel.
Generating MacsBug names.

- Using functions calls for inline functions.
- Having the compiler stop at the first error it finds.
- Having the compiler only report the first few errors it finds.
- Dumping all the errors the compiler finds into a special errors window.

Each of these options is presented in turn.

**Always Generate Stack Frames**

Functions that have variables on the stack have stack space reserved using the MC68000 instruction LINK (the stack space is returned using UNLK). Functions that do not need stack space do not need a LINK/UNLK pair. Selecting the Always generate stack frames option causes the compiler to create a LINK/UNLK pair for all functions, which can make low-level debugging easier.

Table 14.13 shows the disassembly of the following code with and without this option:

```c
int Return1()
{
    return 1;
}
```

**Table 14.13. Effects of Always Generate Stack Frames on Return1**

<table>
<thead>
<tr>
<th>Always Generate Stack Frames Off</th>
<th>Always Generate Stack Frames On</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVEQ #$01,D0</td>
<td>LINK A6,#$0000</td>
</tr>
<tr>
<td>RTS</td>
<td>MOVEQ #$01,D0</td>
</tr>
<tr>
<td></td>
<td>UNLK A6</td>
</tr>
<tr>
<td></td>
<td>RTS</td>
</tr>
</tbody>
</table>
Generate Profiler Calls

If you turn this option on, each function starts with a call to _profile_. The name of the function also is passed as an argument. Suppose that you have the following code:

```c
void Foo()
{
    int i = 3;
}
```

With this option on, it is compiled as if it were written as follows:

```c
void Foo()
{
    _profile_("pFoo");
    int i = 3;
}
```

This enables you to generate timing information for your code that you can then use with the Symantec C++ profile library (see the Symantec C++ Reference Manual for more information on the profile library).

Generate MacsBug Names

There is a Macintosh convention (used by low-level debuggers) of adding the name of the function to the end of its compiled code. Traditionally, these names are called MacsBug names because they are used by the Apple low-level debugger, MacsBug. Other debuggers, like TMON, also use these function names.

Normally, source-level debuggers, like the THINK Debugger, do not need these function names because they have complete information about the relationship between compiled code and the corresponding source code. If you only use the THINK Debugger, you obviously do not need to use this option. If, however, you debug using a low-level debugger, you want it on.

Table 14.14 shows the disassemblies for the following function with and without MacsBug names:

```c
int MyFunction()
{
    return 32 * 7;
}
```
Table 14.14. Effects of Generate MacsBug Names on MyFunction

<table>
<thead>
<tr>
<th>Generate MacsBug Names Off</th>
<th>Generate MacsBug Names On</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINK A6,#$0000</td>
<td>LINK A6,#$0000</td>
</tr>
<tr>
<td>MOVE.L #$000000E0,D0</td>
<td>MOVE.L #$000000E0,D0</td>
</tr>
<tr>
<td>UNLK A6</td>
<td>UNLK A6</td>
</tr>
<tr>
<td>RTS</td>
<td>RTS</td>
</tr>
<tr>
<td>DC.B $80+$0E, 'MyFunction_Fv', $00</td>
<td>DC.W 0</td>
</tr>
<tr>
<td>DC.W 0</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE**

The compiler has mangled the function name from the original MyFunction to MyFunction_Fv. This is the result of C++ name mangling.

**Use Function Calls for Inlines**

As you learned in Chapter 8, in C++ the `inline` keyword is a compiler request that replaces the call to a function with its actual code. When you check the Use function calls for inlines option, the compiler ignores the `inline` keyword; the function call remains a function call. This is enormously useful when you need to debug an inline function—a task that can be very difficult (for example, how do you set a breakpoint in an inline function?). By making an inline function into a normal function, you can use the standard debugging tools.

Table 14.15 shows the disassembly of the following code with this option off and on:

```cpp
inline short Square(short i)
{
    return i * i;
}

short FourthPower(short i)
{
    return Square(i) * Square(i);
}
```
As you would expect, when this option is off, the square function no longer exists in the disassembled code as a separate function. Figures 14.7 and 14.8 show the square function in the THINK Debugger with the option both off and on. Note in figure 14.8 that, when the option is off, you cannot set breakpoints in square. With the option on, you can.

### Table 14.15. Effects of Use Function Calls for Inlines on FourthPower

<table>
<thead>
<tr>
<th>Function Calls for Inlines Off</th>
<th>Function Calls for Inlines On</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LINK</strong> A6,#$0000</td>
<td><strong>LINK</strong> A6,#$0000</td>
</tr>
<tr>
<td><strong>MOVE.L</strong> D3,-(A7)</td>
<td><strong>MOVE.L</strong> A2,-(A7)</td>
</tr>
<tr>
<td><strong>MOVE.W</strong> $0008(A6),D3</td>
<td><strong>MOVE.W</strong> $0008(A6),-(A7)</td>
</tr>
<tr>
<td><strong>MOVE.W</strong> D3,D0</td>
<td><strong>JSR</strong> Square(short)</td>
</tr>
<tr>
<td><strong>MULS.W</strong> D0,D0</td>
<td><strong>MOVEA.L</strong> D0,A2</td>
</tr>
<tr>
<td><strong>MULS.W</strong> D3,D0</td>
<td><strong>MOVE.W</strong> $0008(A6),-(A7)</td>
</tr>
<tr>
<td><strong>MULS.W</strong> D3,D0</td>
<td><strong>JSR</strong> Square(short)</td>
</tr>
<tr>
<td><strong>MOVE.L</strong> (A7)+,D3</td>
<td><strong>MOVE.W</strong> D0,D1</td>
</tr>
<tr>
<td><strong>UNLK</strong> A6</td>
<td><strong>MOVE.L</strong> A2,D0</td>
</tr>
<tr>
<td><strong>MOVEA.L</strong> (A7)+,A0</td>
<td><strong>MULS.W</strong> D1,D0</td>
</tr>
<tr>
<td><strong>ADDQ.W</strong> #$2,A7</td>
<td><strong>MOVEA.L</strong> (A7)+,A2</td>
</tr>
<tr>
<td><strong>JMP</strong> (A0)</td>
<td><strong>UNLK</strong> A6</td>
</tr>
<tr>
<td></td>
<td><strong>MOVEA.L</strong> (A7)+,A0</td>
</tr>
<tr>
<td></td>
<td><strong>ADDQ.W</strong> #$2,A7</td>
</tr>
<tr>
<td></td>
<td><strong>JMP</strong> (A0)</td>
</tr>
</tbody>
</table>

**Square(short):**

| **LINK** A6,#$0000            |
| **MOVE.W** $0008(A6),D0      |
| **MULS.W** D0,D0             |
### Table 14.15. continued

<table>
<thead>
<tr>
<th>Function Calls for Inlines Off</th>
<th>Function Calls for Inlines On</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNLK</td>
<td>A6</td>
</tr>
<tr>
<td>MOVEA.L</td>
<td>(A7)+,A0</td>
</tr>
<tr>
<td>ADDQ.W</td>
<td>#$2,A7</td>
</tr>
<tr>
<td>JMP</td>
<td>(A0)</td>
</tr>
</tbody>
</table>

#### Figure 14.7. The Square function with the Use function calls for inlines option on.

#### Figure 14.8. The Square function with the Use function calls for inlines option off.

### Error Reporting

Rather than reporting just the first error it encounters, Symantec C++ can report multiple errors. You can have errors reported three different ways:

- The Stop at first error button reports only the first error for a given file.
- The Report the first few errors button reports the first six errors in a given file.
The Report all errors in a file button reports all the errors in a file (unless it finds an unrecoverable error).

The default is to report the first few errors. This is the best setting because one error can often cause cascading error messages. Although the compiler attempts to generate only one message for an error, you could have many, many error messages because of one actual error. When the compiler reports only the first few errors, you know that there is a limit on the number of generated messages you have to see—it's also easier on your ego.

The Warning Messages Panel

The C++ Options Warning Messages panel is shown in figure 14.9.

![Warning Messages Panel](image)

**Figure 14.9.** The C++ Options Warning Messages panel.

There are several different warning message options that you can specify in this panel:

- Using = in conditionals.
- Nested comments.
- Unused expressions.
- Empty loops.
- Large automatic variables.
- Old style delete [].
Missing overloads for ++ & -. 
Reference initialization.
Variable used before set.
Return address of auto.
Unrecognized pragma.
Old style function definitions.

Symantec C++ generates warnings for your code if you select any of the options in this panel. You should always keep all of these warning messages on. Compiling without warning messages makes about as much sense as ignoring the traffic signs while driving on an unfamiliar windy road on a dark rainy night. If you compile without warning messages, you are ignoring potential bugs that can haunt you later.

So what do you do if you get warnings? Try to follow these rules:

- When you receive a warning, examine the code that generated it.
- If the warning reflects an error in your code, fix the code.
- If the warning is spurious, rewrite your code so that the warning is no longer generated.

In this last case, you are modifying your code so that you do not end up with files like foo.cp that generates 42 warnings. Although files that generate spurious warnings are not bad per se, you may find it harder to notice when a warning (the 43rd one) appears that is not spurious. Think of these warnings as chaff that must be sifted through to obtain the non-spurious wheat.

Look at examples of each warning message that Symantec C++ generates. Where relevant, you see the code that generates the warning after each error message.

**Using = In Conditionals.** You can generate the possible unintended assignment warning message with code like the following:

```c++
if (a = b)
{
    ...
}
```

You can avoid this message by rewriting your code as follows:

```c++
if (!(a = b) != 0)
{
    ...
}
```
**Nested Comments.** You can create the can't nest comments warning message with the following infamous line:

```c++
/* comment /* nested comment */ */
```

**Unused Expressions.** No doubt you usually intend an assignment here. In either case, any time you write something like the following code you get the value of expression is not used warning message:

```c++
i == 3;
```

**Empty Loops.** If the character immediately following a `for` or `while` statement is a `;` the compiler will produce the possible extraneous `;` warning message. If you want an empty loop, include some white space before the `;`.

**Large Automatic Variables.** The very large automatic warning message appears when you attempt to allocate more than 32K worth of local variables. One small typo (too many zeroes, for example) can prove disastrous:

```c++
void WillOverflowStack()
{
  char bigArray[32000];
}
```

(Note that this probably will cause stack overflow.) At the time this book went to press, the compiler did not report this error.

**Old Style Delete[]** The use `delete[]` rather than `delete[expr]`, `expr ignored` warning message: In the old days of C++, you used to delete arrays with the following syntax:

```c++
p = new SomeType[numArrayElements]
... delete [numArrayElements] p;
```

Nowadays, you should use this:

```c++
p = new SomeType[numArrayElements]
... delete [] p;
```

Although the compiler accepts the older form, it fusses at you each time.

**Missing Overloads for ++ & --.** The using operator++() (or --) instead of missing operator++(int) warning message results in newer versions of C++ that enable separate
overloading of prefix ++ (or --) and postfix ++ (or --) operators. Newer versions have operators that take an argument. This warning reminds you of old code in which both the prefix and postfix ++ (or --) called one overloaded routine. Here is an example:

class TBar {
public:
    TBar();

    operator++();
    ...}
... TBar aBar;

    aBar++;

**Reference Initialization.** The compiler blurts out the non-const reference initialized to temporary warning message when it generates a temporary variable and the reference is initialized to the temporary. Because the reference is not const, you lose any changes made to the reference (only the temporary changed) and you get an error in ANSI C++. Here is an example of code that shows this:

    void SetTo5(float &f)
    {
        f = 5;
    }

    void Bar()
    {
        int i = 3;
        SetTo5(i);
        // i is still 3
    }

**Variable Used Before Set.** The variable '%s' used before set warning message is a simple error, one you no doubt get used to seeing whenever your code looks like the following:

    void Bar()
    {
        int I;
        int j;

        j = i * 3;
    }
**Return Address of Auto.** The returning address of automatic 'auto' warning message indicates that you are returning the address of a local variable:

```c
int* ReturnAuto()
{
    int i;

    return &i;
}
```

**Unrecognized Pragma.** The unrecognized pragma warning message appears when you port code from another compiler to Symantec C++ and forget to remove a compiler specific `#pragma`.

**Old Style Function Definitions.** The function definitions with separate parameter lists are not allowed in C++ error (not a warning) occurs when you use old style definitions for your functions (separate parameter identifier and declaration lists):

```c
int Func(a)
{
    int a;
    ...
}
```

**Cast from Incomplete type.** A warning will be generated when a pointer to an incomplete structure is being cast. This can result in problems when the structure is later defined in a multiple inheritance hierarchy.

---

**Prefix Panel**

The final C++ Option panel is the Prefix panel (see fig. 14.10). In this panel, you set the contents of the code you want prepended to each source file before compilation. This is where you normally define compile-time variables or project-wide include files.

If you look in figure 14.10 at the preprocessor window, you see directives that cause every file to have `<MacHeaders++>` included at the beginning.

Customarily, you use this panel to toggle particular compile-time flags depending on which compile you are doing. You may, for example, define a compile-time flag `qDebug` that is true if you want debugging code included in your program and false if you do not. You then bracket debugging code with the following:
#if qDebug
    DebugStr("\pReached point A1 in the program");
#endif

Figure 14.10. The Prefix panel.

NOTE

If you make any changes to the contents of this panel, the compiler recompiles all your files the next time you build. Because this panel affects what the compiler includes at the beginning of each file, this always must occur.

Summary

You have learned about each of the six panels that affect your C++ compiles. These six panels are the Language Settings, Compiler Settings, Code Optimization, Debugging, Warning Messages, and Prefix panels. You rarely make changes to these panels after you have them set up for a particular project. Nevertheless, it is worth knowing how each option works and what results you are likely to get from each setting.
This is by far the most important practical section in the book. Here, you use the C++ you have learned. Before doing so, however, you learn about present application frameworks and what is planned for the future in Chapter 15. Chapters 16 through 21 contain C++ programs that use the TCL to create a variety of object-oriented applications.

How are you going to learn to use the TCL? TCL is not presented in an abstract fashion, but in the context of coding. This can be both good and bad. It is bad because you are learning only about the parts of the TCL used in the creation of these programs. But it is good because you see the how and when of using a framework.

Along the way you also learn how to use new tools like the Inspector and the Visual Architect.
This chapter provides you with a strong incentive to use frameworks in your programming. You first see the benefits a framework can provide, and then you learn about what is available.

There is more than one application framework in the Macintosh arena. Two of the most important Macintosh application frameworks, MacApp and the THINK Class Library (TCL), were released in 1985 and 1989, respectively; others have arrived since. Some of the most intriguing application frameworks are on the horizon.

**Defining Frameworks**

A *framework* is a collection of objects that collaborate to carry out a task.

If you call to mind a framework, you probably imagine the most common example: an *application framework*. This, of course, is a framework that includes a collection of classes simplifying the job of creating an application. On the Macintosh programming side, application frameworks arrived late, not appearing until 1985.
The tardiness of applications frameworks for the Macintosh meant that pioneer programmers were forced to write their applications from scratch, a rather daunting task. In fact, when some programmers came to Macintosh, they were amazed by how much code they had to write to create a Macintosh application. They muttered about having to call DragWindow to deal with user clicks in the title bar of a window. There was justification to such discontent. If a user clicked on one radio button, an application programmer had to write code to turn off the other radio buttons; a framework should have done that. Born out of this situation were application frameworks such as MacApp and the THINK Class Library.

The late appearance of Macintosh application frameworks is made even more ironic when you think about one of the fundamental tenets of the Macintosh experience: all applications should look and work the same way. When you realize that much of this uniform experience is the result of enormous amounts of duplicated code present in all Macintosh applications, you should be surprised that the applications manage the uniformity that they do.

Capturing this common code in an application framework that sits above the Macintosh operating system is an obvious solution. If the user clicks in the title bar of a window, the framework calls DragWindow. If a user clicks on a radio button, the framework turns off all the others.

All application code must jump through the hoops of Macintosh operating system requirements. By using a framework, you rely on the framework programmer who is a hoop-jumper by nature, and you are freed from such tedious coding.

There is not much of a downside to letting the framework handle such tasks. If you want control, you always can assert it. A framework is object-oriented, so any programmer who wants to do something different in response to the user clicking on the title bar of a window can just override the member function that deals with that activity.

Why Use Frameworks?

There are several general rules to remember when you think about whether to use frameworks:

- Frameworks enable you to finish your program faster. The time spent learning to use a framework is much less than it takes to write the code it handles for you. As a result, you can start working on the unique aspects of your application much sooner.

- Frameworks reduce errors in your programs. Frameworks are tested, not only by their vendor, but by all the programmers who use them. You benefit from that. In addition, frameworks are written by subject experts, programmers who know how to design classes and know their field.
Framework enhancements benefit your program. When a framework is enhanced, your application may be able to take advantage of this simply by recompiling.

Frameworks enable you to concentrate on your area of expertise. If you are a seismologist writing MacEarthquake, you can concentrate on designing and writing your classes (TSanAndreasFault, TRichterScale, and so forth). The framework ensures that your program works on the Macintosh; you ensure that everybody is ready for the Big One.

Using frameworks increases portability. By structuring your code correctly, you easily can port your application. You design your user interface code to be separate from the actual content of your application. Your application ends up being divided into three parts (see fig. 15.1).

Structuring your application this way gives your content classes portability. They can be moved from one framework to another and even from one operating system to another. Your content classes can go anywhere there is a C++ compiler.

![Diagram](image)

**Figure 15.1.** The structure of your application.

### Types of Frameworks

Frameworks come in a variety of useful flavors. Besides the basic vanilla application framework, you also can create any of the following frameworks on the Macintosh:

- Desk accessory frameworks
- Control panel frameworks
- Start-up document (INIT) frameworks
- Printer driver frameworks
- Device driver frameworks
A good framework for each of these items provides all the code required to produce a complete shell, but without any useful content. The control panel framework, for example, could be structured to respond to mouse clicks, menu items, opening, and closing. You, the control panel programmer, need to override a handful of member functions with names like `HandleOpen`, `HandleClose`, `Handle(Mouse)Click`, `HandleKeystroke`, and so forth.

**The Role of Frameworks**

If you approach a framework with the mind-set of a procedural programmer, you run into a brick wall. In Part II of this book, you learned the need to shift from a procedural point of view to an object-oriented point of view when writing C++ code. Likewise, there is a difference between using a procedural library and a framework.

When you use a procedural library, you call the functions as you see fit; you are master of the universe. As an object-oriented programmer using a framework, however, you are not in charge—the framework is. You are the specialist the framework calls in when it needs your (overridden) member functions to do the job right. In an application framework, for example, you override the member function for drawing the contents of a window. The framework, in turn, calls that function whenever the window needs to be redrawn.

**Choices for Application Frameworks**

MacApp is by far the most robust application framework for Macintosh, although you also can create a commercial-quality application with the TCL. The other application frameworks discussed here have more specialized uses.

**The THINK Class Library**

The THINK Class Library (TCL) is a framework from Symantec for creating Macintosh applications. TCL is shipped with Symantec C++.

Class diagrams can help you conceptualize the scope of a framework's classes. With this in mind, look at the diagram of the TCL's class hierarchy shown in figure 15.2.
The goal of the TCL framework writers was to provide an application framework that was both easy to learn and could serve as a basis for commercial-quality applications. Imagine that using the TCL is like driving a Chevrolet. It is not elegant, but it is the car you learned to drive, and it dependably gets you where you need to go.

The applications in Chapters 16 to 21 are all written using the TCL. Appendix C contains a small reference manual for the TCL classes used in those chapters.

**MacApp**

MacApp is a framework from Apple for creating Macintosh applications. Currently, it can be used with either the Apple Macintosh Programmer's Workshop (MPW) C++ or Symantec C++ for MPW.

MacApp is a much larger framework than the TCL. And, as you might expect, the class diagram is correspondingly more complex. Figure 15.3 shows what the MacApp class hierarchy looks like.

The creators of MacApp wanted an application framework that could be used to create bullet-proof, industrial-strength applications. As a result, using MacApp is like driving a tank. It is slow and heavy, but bullet-proof and guaranteed to get you where you want to go (by whatever means necessary).

**QuickApp**

QuickApp, a product from Emergent Behavior, is another Macintosh application framework. It has quite a different goal from the TCL or MacApp, however. It does not attempt to support commercial-quality or commercial size applications. Instead, it focuses on rapid development and prototyping tasks. It is especially well-suited for prototypes or applications that have a limited life span or audience. QuickApp compiles with Apple MPW C++, Symantec C++, and Symantec C++ for MPW.

The C++ limitations present in the TCL and MacApp disappear with QuickApp, which utilizes the full power of C++, including the following:

- constructors
- destructors
- multiple inheritance
- copy constructors
- stack-based objects
Figure 15.2. The TCL Class Hierarchy.
Figure 15.3. The MacApp class hierarchy.
Chapter 15: Using a Framework
Figure 15.4 shows you the class hierarchy for QuickApp. As you can imagine, the small size makes it not only easy to learn, but extremely quick to compile.

Using QuickApp is like touring in a British roadster. It is small, fast, and provides a great driving experience. It may, however, lack some of the amenities that, although not strictly necessary, do make life easier. It also is not the best choice for developing large programs.

Figure 15.4. The QuickApp class hierarchy.

**MicroGA**

MicroGA, another product from Emergent Behavior, is a framework for creating genetic algorithms. MicroGA compiles with Apple MPW C++, Symantec C++, and Symantec C++ for MPW. It can be used with MacApp, MicroGA, or the TCL.

Genetic algorithms are algorithms based on the evolutionary model. Like evolution, a genetic algorithm starts with random solutions to a problem. The solutions compete, and the best solutions survive and reproduce by mating. Slowly, competition improves the quality of the solutions.
MicroGA provides classes that model genes, individual solutions, and populations. Figure 15.5 shows the complete MicroGA class hierarchy.

![MicroGA class hierarchy diagram](image)

**Figure 15.5.** The MicroGA class hierarchy.

### Physics

Emergent Behavior's final framework, Physics, is a tool for modeling physical interactions between objects. For instance, it can be used to model how the moon orbits around the Earth.

Physics compiles with Apple MPW C++, Symantec C++, and Symantec C++ for MPW. It can be used with MacApp, MicroGA, or with the TCL.

Figure 15.6 shows the complete Physics class hierarchy.

![Physics class hierarchy diagram](image)

**Figure 15.6.** The Physics class hierarchy.
Frameworks of the Future

There are other frameworks on the horizon. Of these frameworks, two offer some very interesting times ahead for Macintosh programmers.

OpenDoc Parts Framework
The OpenDoc Parts Framework (ODPF) is being developed by Apple. In some ways it is a successor of both TCL and MacApp. By far its most distinctive feature is that it is a cross-platform compound document framework. Instead of building applications, the ODPF is for building document parts. Initially, it will support Macintosh and Microsoft Windows—a powerful combination.

Taligent
Taligent, an independent company formed from a partnership of Apple and IBM, is taking a more ambitious course. Not content to splice an application framework on top of a procedural operating system, they are writing a wholly object-oriented environment from the ground up. Everything, including the operating system, will be accessed via objects.

The product, code named Pink, will be available on a number of platforms because it was designed from the beginning to be portable.

NOTE
In the late 1980s, Apple was contemplating the future for the Macintosh operating system. As they looked at various features and possible characteristics, they wrote each one down on different colored note cards according to a feasible time frame. Blue note cards were short term, pink note cards were medium term, and red note cards were long term. The blue note cards eventually became the basis of System 7 (thus the engineers stamping out System 7 bugs were called Blue Meanies). The pink note cards became the Pink project, which is now pursued at Taligent. It makes you wonder where the red cards ended up.

Summary
Now that you have an idea of what application frameworks are and why you should use them, it is time to get down to the business of actually using them. The rest of the chapters in Part III use the TCL to create a variety of programs.
RandomRectangle: Building a Rectangle Display Program

The RandomRectangle program will never put any of the sophisticated screen-saver programs to shame, but it does perform a simple task relatively well. It is a rectangle drawing program that does the following:

- Creates a window for the rectangles.
- Draws the rectangles in that window.
- Fills the rectangles with random colors.
- Makes the rectangles random sizes.
- Decides how often to create each new rectangle.
You use the THINK Class Library to write the program in this chapter. By looking at what the program does and by reviewing the source files, you see how the TCL can be used to develop a project. You look at each new class and learn what it does. And finally, you look at the program resources.

**How RandomRectangle Works**

RandomRectangle is an example of one of the simplest programs you can write with the TCL. The dynamic nature of the program continually draws rectangles of random sizes filled with random colors.

When you run RandomRectangle on a machine without Color QuickDraw, it fills the rectangles with random patterns instead of colors. Figure 16.1 shows you that version.

![Figure 16.1. RandomRectangle at work on a machine without Color QuickDraw.](image)

Because it is so small, this program is a good one to familiarize yourself with using the TCL. It also sharpens your use of machine-specific characteristics and teaches you about the system's idle time.

**Setting Up and Compiling RandomRectangle**

The first order of business is to compile and run the program. Before you can do this, however, you probably need to increase the memory partition of the THINK Project Manager. You need to allocate at least 5 megabytes to the partition to compile with the TCL.
NOTE

To increase the partition size of the THINK Project Manager, select the Project Manager and use Get Info from the File menu of the Finder.

You need to copy the RandomRectangle folder from the source code disk. Open the RandomRectangle.pi project, and then choose Run. (If you need help opening or running a project, refer to Chapter 1, which contains a tutorial on the Project Manager.) It takes a few minutes before the program runs because all of the files in the project need to be recompiled.

Running the Program

After you have RandomRectangle.pi running, you should test the program as follows:

- Try moving the window.
- Try resizing the window. Even though there is no grow box, the window is resizable.
- Show the clipboard.
- Notice what does not work (for instance, the About menu item does not do anything).

Later chapters provide functionality that you can use to upgrade this program. Chapter 18 gives you code that makes the About menu item work. There also is code in Chapter 19 that enables background processing even when the window is not active. Currently, the rectangles are drawn only if the rectangle window is active.

Inspecting the Program

This program is built upon three important classes: TRandom, CRandomRectanglePane, and CRectangleApp. Class diagrams help clarify the structure of these and the other classes by providing a mechanism for describing the members of a class (data and functions) as well as showing the inheritance relationship. Figure 16.2 shows the class diagram for this application.

As the diagram shows, TRandom, CRandomRectanglePane, and CRectangleApp are the new derived classes. They are on the right and are connected by lines to their base classes on the left.
Figure 16.2. RandomRectangle's class diagram.

Following are short descriptions of each of these new classes:

**TRandom**
A random number generator. This class is independent of the TCL.

**CRandomRectanglePane**
Draws the random rectangles.

**CRectangleApp**
Creates the window and the CRandomRectanglePane object within that window.

**IMPORTANT**

The THINK Class Library uses a naming convention different from that described in Chapter 10. The TCL class names begin with C and BR_, instead of T, as you might expect. There also is no particular convention for data members in the TCL. In the interests of maintaining convention, Chapters 16-21, which use the TCL, begin class names with C for all the classes derived from TCL classes. Please note, however, if the classes are independent of the TCL, they begin with the more standard T. Any new data members begin with r.
The TRandom Class

Look at the TRandom class first. The class declaration is as follows:

class TRandom {
public:
    TRandom();
    TRandom(unsigned long seed);

    void Advance();
    unsigned long Value();
    unsigned long ValueInRange(unsigned long low, unsigned long high);

private:
    unsigned long fValue;
};

You can construct the TRandom class in one of two ways: with an explicit seed (with the second constructor) or with a default seed (using the first constructor). The Value member function returns the current random value, and ValueInRange returns a random value in the specified range. Advance generates a new random number.

Because this class is stand-alone, you can use it with or without the TCL. As with any good object-oriented class, its declaration does not specify how to generate the random numbers, just how to use them.

Following are the constructors for TRandom:

TRandom::TRandom()
{
    fValue = ::TickCount();
}

TRandom::TRandom(unsigned long seed)
{
    fValue = seed;
}

When you create a TRandom object, you can choose whether to specify a seed for the random number generator. If you do not specify a seed, the initial value is seeded with the number of ticks since machine boot time, which is a reasonably random value. You also can set a particular seed in the constructor if you want. You might find this method useful if you want to recreate a random sequence. The routines that return values are simple. The current random value is stored in fValue, and the Value routines look like this:
unsigned long TRandom::Value()
{
    return fValue;
}

unsigned long TRandom::ValueInRange(unsigned long low, unsigned long high)
{
    return (Value() % (high - low + 1)) + low;
}

ValueInRange returns a number in the range from low to high. In tandem with this, the Advance routine calculates the next random value:

void TRandom::Advance()
{
    fValue = fValue * 12345 + 6789;
}

This is all there is to the TRandom class. It generates random numbers and returns them to the program. Now, look at the second new class, CRandomRectanglePane.

The CRandomRectanglePane Class

In the TCL, you do your doodling or drawing in a pane. Panes are derived from views. These views are representations of visual objects on the screen. Views have other derived classes as well. These include windows, controls, and pictures. Furthermore, the TCL arranges views in an enclosure hierarchy in which each view has one enclosure and can enclose zero or more other views. CRandomRectanglePane exists in such a hierarchy; it is a subview of a window. This new class derives from CPane and does the job of drawing the random rectangles:

class CRandomRectanglePane: public CPane {
public:
    CRandomRectanglePane(CView *anEnclosure,
                          Bureaucrat *aSupervisor,
                          short aWidth, short aHeight,
                          short aHEncl, short aVEncl,
                          SizingOption aHSizing, SizingOption aVSizing);

    virtual void Dawdle(long *maxSleep);

protected:
    virtual void DrawRectangleInRandomColor(const Rect *r);
    virtual void DrawRectangleInRandomPattern(const Rect *r);

328
Symantec C++ Programming for the Macintosh, 2nd Edition
private:
    TRandom fRandom;
};

In this code, the constructor is CRandomRectanglePane, which requires the same parameters as CPane. This is all you need to know for now because each of these functions is described in greater detail later in this chapter.

Take a look at how CRandomRectanglePane works and how it uses its member functions. First, you should note that it overrides its inherited member function, Dawdle, which is called whenever idle time is available. Idle time is available when the main event loop is receiving null events. CRandomRectanglePane only has one data member, fRandom. This is used by the drawing routines to generate random rectangles, colors, and patterns.

Construction

The constructor does nothing beyond calling the constructor of its base class:

CRandomRectanglePane::CRandomRectanglePane(CView *anEnclosure,
                                           CBureaucrat *aSupervisor,
                                           short aWidth, short aHeight,
                                           short aHEncl, short aVEncl,
                                           SizingOption aHSizing, SizingOption aVSizing) :
                                           CPane(anEnclosure, aSupervisor, aWidth, aHeight,
                                                 aHEncl, aVEncl, aHSizing, aVSizing)
{
    TCL_END_CONSTRUCTOR
}

It is important to write a constructor for each class, even when it just calls the base class constructor. If you do this extra work, determining what member function to call to perform initialization is easy.

NOTE

In the TCL, each constructor you write should end with the macro TCL_END_CONSTRUCTOR. Similarly, each destructor you write should start with the macro TCL_START_DESTRUCTOR.

TCL contains an exception library, which requires these macros to correctly deal with errors. The exception library uses these macros in the case of an error to determine which destructors to call for a partially-constructed object.
Obtaining Idle Time

How does the program obtain the idle time it needs to draw the rectangles? It overrides \texttt{Dawdle}. The following code shows the mechanism:

```c
void CRandomRectanglePane::Dawdle(long *maxSleep)
{
  Rect r;
  Point p1;
  Point p2;
  LongRect frameSize;

  GetFrame(&frameSize);

  p1.h = fRandom.ValueInRange(frameSize.left, frameSize.right);
  fRandom.Advance();
  p1.v = fRandom.ValueInRange(frameSize.top, frameSize.bottom);
  fRandom.Advance();
  p2.h = fRandom.ValueInRange(frameSize.left, frameSize.right);
  fRandom.Advance();
  p2.v = fRandom.ValueInRange(frameSize.top, frameSize.bottom);
  fRandom.Advance();

  ::Pt2Rect(p1, p2, &r);

  Prepare(); // prepare coordinate system for drawing

  if (gSystem.hasColorQD)
    DrawRectangleInRandomColor(&r);
  else
    DrawRectangleInRandomPattern(&r);

  *maxSleep = 20; // one rectangle every 1/3 second
}
```

First, the \texttt{GetFrame} member function returns the bounding rectangle (size) of the pane. Unlike the standard QuickDraw 16-bit coordinate system, the TCL panes can use a 32-bit system. \texttt{GetFrame} generates a frame that is then used as the bounds to create two random points. Then, \texttt{Pt2Rect} initializes the random rectangle \texttt{r}, using the two random points as corners.

Using idle time to draw is not standard on the Macintosh. Normally, Macintosh drawing occurs as a response to update events. When an update event occurs, the program calls a pane's Draw member function, which issues QuickDraw calls to draw the contents of the pane.
In RandomRectangle, on the other hand, the pane is not drawing in response to update events, but rather at idle time. Because it is drawing at such an unusual time, the TCL has not prepared its coordinate system for drawing. CRandomRectanglePane must prepare its coordinate system on its own by calling its Prepare member function. This sets up the correct GrafPort, clipping, and origin.

After the pane is prepared, Dawdle examines the global structure gSystem—a TCL structure that describes the environment in which the application is running. The gSystem fields follow:

```c
Boolean hasWNE;
Boolean hasColorQD;
Boolean hasGestalt;
Boolean hasAppleEvents;
Boolean hasAliasMgr;
Boolean hasEditionMgr;
Boolean hasHelpMgr;
Boolean hasScriptMgr;
Boolean hasFPU;
Boolean hasProcessMgr;
short scriptsInstalled;
short systemVersion;
```

This variable is initialized by the TCL as part of the initialization process. Dawdle calls the appropriate DrawRectangle function based on whether this machine has Color QuickDraw.

Finally, Dawdle specifies in the maxSleep parameter how many ticks until the pane receives its next Dawdle. It sets maxSleep to 20 (three rectangles per second draw).

**Drawing in a Random Pattern**

DrawRectangleInRandomPattern draws the specified rectangle in a random pattern, which it gets from the system:

```c
void CRandomRectanglePane::DrawRectangleInRandomPattern(const
    Rect *r)
{
    Pattern aPattern;

    GetIndPattern(&aPattern, sysPatListID, fRandom.ValueInRange(1,31));
    fRandom.Advance();
    FillRect(r, aPattern);
}
```
If you look at Dawdle, you can see that DrawRectangleInRandomPattern is called only if the machine does not have Color QuickDraw. This routine uses patterns as a way to distinguish the rectangles from each other on noncolor machines.

**Drawing in a Random Color**

Next, look at the function that colors the rectangles, DrawRectangleInRandomColor:

```cpp
void CRandomRectanglePane::DrawRectangleInRandomColor(const Rect *r)
{
    RGBColor aColor;
    PixPatHandle myPixPat;

    aColor.red = fRandom.Value();
    fRandom.Advance();
    aColor.blue = fRandom.Value();
    fRandom.Advance();
    aColor.green = fRandom.Value();
    fRandom.Advance();

    myPixPat = NewPixPat();
    MakeRGBPat(myPixPat, &aColor);
    FillCRect(r, myPixPat);
    DisposePixPat(myPixPat);
}
```

Red, green, and blue (an RGB pattern) values are set in aColor. Next, the routine uses this pattern of colors to approximate a given color that works on displays of any bit depth.

**NOTE**

For those of you familiar with the standard method for drawing in color (using the Color QuickDraw call to RGBForeColor), this might appear a strange way to color a rectangle. It should make sense, however, when you realize that, regardless of the color, RGBForeColor draws in either solid black or solid white when in 1-bit mode. To avoid this, the program has DrawRectangleInRandomColor call MakeRGBPat. In 1-bit mode, this creates a black-and-white pattern that simulates the requested color.
The CRectangleApp Class

Now that you have a class that creates a random number and one that draws and colors the rectangles, you need one last class—one that creates a window and the pane for the rectangles. This new class is CRectangleApp and is derived from the TCL class CApplication.

IMPORTANT

Every TCL program needs to create a class derived from CApplication. In this case, it may not be obvious why because CRectangleApp does not override any CApplication member functions. Because most programs need such overriding (certainly any programs that support multiple documents), it is customary to always override CApplication.

The following code shows the declaration:

class CRectangleApp: public CApplication {
public:
    CRectangleApp();
    virtual void StartUpAction(short numPreloads);
};

CRectangleApp has a constructor:

const short kExtraMasters = 1;
const Size kRainyDayFund = 45000;
const Size kCriticalBalance = 40000;
const Size kToolboxBalance = 20000;

CRectangleApp::CRectangleApp() :
    CApplication(kExtraMasters, kRainyDayFund, kCriticalBalance, kToolboxBalance)
{
    TCL_END_CONSTRUCTOR
}

CRectangleApp begins by calling the constructor of its base class, CApplication. A list of its parameters and what they control follows:

kExtraMasters    The number of calls to the Toolbox routine MoreMasters
kRainyDayFund    The amount of memory reserved for low-memory conditions
kCriticalBalance  The amount of memory used for critical operations
kToolboxBalance  The amount of memory used for Toolbox operations

You use this reserved memory to ensure that TCL applications continue to function when
memory space gets tight. If you need more information on memory management, you should
look at the TCL documentation.

The CRectangleApp class has one member function, StartUpAction, to create the window and set
it in motion:

```cpp
void CRectangleApp::StartUpAction(short numPreloads)
{
    CApplication::StartUpAction(numPreloads);

    const short kWINDResourceID = 500;
    const Boolean kWindowFloats = TRUE;
    const Boolean kFitHorizontal = TRUE;
    const Boolean kFitVertical = TRUE;

    CDirector *aDirector = new CDirector(this);

    CWindow *aWindow = new CWindow(kWINDResourceID, kWINDResourceFloats, aDirector);
    aDirector->itsWindow = aWindow;

    CRandomRectanglePane *thePane =
        new CRandomRectanglePane(aWindow, aDirector, 0, 0, 0, 0,
            sizELASTIC, sizELASTIC);
    thePane->FitToEnclosure(kFitHorizontal, kFitVertical);
    aDirector->itsGopher = thePane;
    thePane->BecomeGopher(TRUE); // so it starts dawdling
}
```

The call to the inherited StartUpAction makes sure that any files selected from the Finder would
be opened rather than creating a new document. In RectangleApp, there are no files created, so
this just makes sure that any other startup actions required get done.

**Creating the Director**

All TCL applications consist of managers who control the interactions between elements. As in
the real world, these *bureaucrats* have specific names and functions, and they exist within a chain
of command. The first manager you need to understand is the director, which is an object that handles the interactions between a window and the application. StartUpAction creates just such a director and initializes it with its supervisor, the application.

The first object in the chain of command is the gopher (lowest on the totem pole). If the gopher does not handle the command, it sends it up the chain to its supervisor, the next bureaucrat in line. This processing continues until some bureaucrat handles the command. You need to remember to connect the chain of command; when you create a bureaucrat, you need to specify its supervisor. Be attentive to these command chains because a TCL application creates a very bureaucratic world in which many objects are bureaucrats, including applications, panes, and windows.

Creating the Window

After determining the pecking order in the program, the function creates a window and initializes it. Using kWINDResourceID, it specifies the resource ID for the window. kWInwindowFfoats indicates that the window is not a floating window (that is, not a windoid). And the window’s director is set to aDirector. After initializing the window, it sets the director’s data member, itsWindow, to point to that window.

Creating the RandomRectangle Pane

Finally, the function creates a CRandomRectanglePane and initializes the following:

- Its enclosure
- Its director
- Its horizontal and vertical size
- Its horizontal and vertical location within this enclosure
- Its horizontal and vertical sizing options

Note that, even though the size and location are initialized to 0, they are modified shortly. Look at the sizing options: sizELASTIC specifies that the pane size changes whenever the enclosure changes, and the FitToEnclosure member function sets the pane to the size of the window. The pane, therefore, is always exactly as big as its enclosing window. FitToEnclosure starts it that way; sizELASTIC keeps it that way.

Notice also that the director’s itsGopher data member refers to a particular bureaucrat. StartUpActions sets this data member to the pane, and then calls BecomeGopher to make the pane the active gopher.
Looking at the Final Chain of Command

The chain of command after these objects have been created is shown in figure 16.3.

![Diagram of chain of command]

**Figure 16.3.** The chain of command in RandomRectangle.

The chain of command is used not only for handling commands, but also for dealing with idle time. The TCL calls the `Dawdle` member function for bureaucrats in the chain. If a bureaucrat is not part of the chain of command, its `Dawdle` does not get called. If you circumvent the chain of command, the application stops working correctly. For instance, try showing the clipboard or putting the RandomRectangle application in the background. Notice that the rectangles stop being drawn. This is because the chain of command is based on the active window. If the rectangle window is not active, `CRandomRectanglePane` is not the gopher and, therefore, does not call `Dawdle`.

**NOTE**

In Chapter 19, you learn how to have the `Dawdle` routine called even when the pane is not part of the chain of command. The rectangles then continue to be drawn even if the application is in the background or the window is not active.

The Main Program

All good things must have a beginning. In this case, that is the main function.

```cpp
void main()
{
    CRectangleApp *rectangleApp;
    rectangleApp = new CRectangleApp;
    rectangleApp->Run();
    rectangleApp->Exit();
}
```
This main function is very similar to the one used for all other TCL programs. It creates an instance of the derived class CRectangleApp, calls its Run member function, and then calls its Exit member function. The Run function does not return until the user chooses Quit; at that point, Exit gets a chance to do any necessary cleanup.

**Resources**

The resources for RandomRectangle are in Rez format. Note that quite a few of them come with the TCL. Only the following resources, however, are unique to this application:

```cpp
resource 'BNDL' (128) {
    'C++1',
    0,
    {
        'FREF',
        {
            0, 128
        },
        'ICN#',
        {
            0, 128
        }
    }
}
```

```cpp
data 'C++1' (0, "Owner resource") {
    "$00"
};
```

```cpp
data 'FREF' (128) {
    "$4150 504C 0000 84"
};
```

```cpp
resource 'ICN#' (128) {
    {
        "$0000 0000 0000 0000 003F FE00 0020 0200"
        "$0200 0200 03FF F200 0220 1200 0220 1200"
        "$0220 1200 0220 1200 0FFE 1200 0A22 1200"
        "$0A22 1200 0A22 1200 0BFF F200 0822 0200"
        "$083F FE00 0802 0000 0FFE",
        "$0000 0000 0000 0000 003F FE00 003F FE00"
    }
};
```

*Chapter 10: RandomRectangle: Building a Rectangle Display Program*
The 'BNDL', 'C++1', 'FREF', 'ICN#', and 'ics#' resources are for the application’s icon.

The remaining resources are part of the TCL. The only change you need to make is to the File menu. You need to remove some of the standard menu items that are not needed for RandomRectangle:

```c
resource 'MENU' (2, "File", preload) {
  2,
  textMenuProc,
  0x7FFFFFFFFE,
  enabled,
  "File",
  {
    "Close#4", noIcon, "W", noMark, plain,
    "Quit#1", noIcon, "Q", noMark, plain
  }
};
```

**Summary**

You have learned that the RandomRectangle program uses three important classes: TRandom, CRandomRectanglePane, and CRectangleApp. TRandom (the only TCL-independent class) handles all the randomization for the program. CRandomRectangle manages the drawing of the rectangles including their size, color, and frequency. CRectangleApp creates the window in which the rectangles are drawn and makes sure that they exist in a coherent world.
Because many Macintosh applications have some text-editing capabilities, it is worth knowing how to use the TCL to do text editing. The application that follows enables only simple editing. It does not provide features for font changing, spell checking, or even formatting the text. What it shows you, however, is how to use the TCL with text-based applications.
The TextTyper Program

The text editor, TextTyper, is a far more substantial TCL program than the RandomRectangle application. It does the following:

- Opens multiple documents.
- Creates or edits the text of a document.
- Prints a document.

Figure 17.1 shows the text-entry window and should give you a fairly good idea of TextTyper's functionality.

![Text Editor-typing test](image)

Figure 17.1. The text-entry window of TextTyper.

Compiling the Program

Copy the TextTyper folder from the source code disk. Open the TextTyper.x project, and then choose Run. It takes a little time for all the files in the project to recompile.

Running the Program

After TextTyper is running, take it for a test drive. Feel free to do any of the following:

- Create new files.
- Open existing text files.
- Open a large text file (greater than 32,767 characters).
- Print a text file.

What happened when you tried to open the large text file? You should have gotten the alert shown in figure 17.2. TextTyper has a limit of 32,767 characters. Because TCL text-editing classes...
use TextEdit, they inherit this limitation. You also might have noticed that the About box does not work. The next chapter has the code to fix this problem.

If you printed one of your text files, you should have seen that pages break between lines rather than in the middle of them. This works correctly because TextTyper deals properly with line breaks when printing.

![Alert](image)

**Figure 17.2.** The alert from opening a big file in TextTyper.

## Inspecting the Program

Now that you have manipulated the application, look at the code. Figure 17.3 shows the class diagram.

![Class Diagram](image)

**Figure 17.3.** TextTyper's class diagram.

There are two new classes: `CTextApp` and `CTextDoc`. They do the following:

- `CTextApp` creates the documents and sets parameters for the Open dialog.
- `CTextDoc` creates the window and text-editing pane and reads and writes the text to disk.
The CTextApp Class

Following is the declaration of CTextApp:

class CTextApp: public CApplication {
public:
    CTextApp();
    virtual void CreateDocument();
    virtual void OpenDocument(SFReply *macReply);
    virtual void SetUpFileParameters();
};

The constructor, CTextApp, is simpler than the one in the last chapter. In TextTyper, it is the
document and not the application that creates the windows.

const short  
kExtraMasters = 1;
const Size    kRainyDayFund = 45000;
const Size    kCriticalBalance = 40000;
const Size    kToolboxBalance = 20000;

CTextApp::CTextApp() :
    CApplication(kExtraMasters, kRainyDayFund, 
    kCriticalBalance, kToolboxBalance)
{
    TCL_END_CONSTRUCTOR
}

In the TCL world, when the user opens an old document or creates a new one, the application
object creates a document object. In TextTyper, it is CApplication's member functions,
OpenDocument and CreateDocument, that perform this job. CTextApp overrides these routines. Take
a look at the simplified version of CreateDocument:

void CTextApp::CreateDocument()
{
    CTextDoc *theDocument = new CTextDoc;
    theDocument->NewFile();
    TCLForgetObject(theDocument);
}

The function creates a CTextDoc, initializes it, and then calls the NewFile member function
(which you see shortly). CreateDocument is not quite complete, however, because it does not deal
with the errors that might occur in the NewFile call. This error needs to be dealt with by an
exception handler.
Error handling in the TCL is based on *exceptions*, not on the error codes with which you are familiar. Although this approach may seem novel to you, it is worth adopting for two reasons: First, ANSI C++ directly supports exception handling (assuring its increasing popularity); second, it makes for easier maintenance of code.

### Generating Exceptions

Instead of returning error codes, the TCL uses functions that generate exceptions. Take a look at the functions that do this:

- **FailOSErr**(short error) generates an exception if the error is not noErr.
- **FailNIL**(void *p) is equivalent to FailOSErr(memFullErr) if p is NULL.
- **FailMemError()** is equivalent to FailOSErr(MemError()).
- **FailResError()** is equivalent to FailOSErr(ResError()).
- **FailNILRes**(void* p) generates an exception if p is NULL and provides a more useful error message than FailNIL.

Here is how exception handling works:

1. When an exception is generated, the function in progress is aborted.
2. All of the other functions currently in the call chain abort, all the way back to the main event loop.
3. Next, the TCL puts up an error alert to notify the user about what went wrong.
4. After the user responds to the alert, the main event loop continues to the next event.

Following are some examples:

```c
Handle h;

h = ::NewHandle(20);
::FailNIL(h);

h = ::GetResource('ALRT', 128);
::FailNILRes(h);

::FailOSErr(::HandToHand(&h)); // HandToHand returns an OSErr
```
Exception-Handling Cases

It is not enough to just generate the exceptions. Sometimes you must respond to them, as in the cases where some cleanup is necessary. Table 17.1 shows some cases where exception handlers are necessary.

Table 17.1. Examples of Exceptions That Need To Be Handled

<table>
<thead>
<tr>
<th>Normal Case</th>
<th>Exception Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>The program opens a file, reads its contents,</td>
<td>If the read fails, the file still needs to be</td>
</tr>
<tr>
<td>and closes the file.</td>
<td>closed.</td>
</tr>
<tr>
<td>The program creates an object, does something</td>
<td>If the action fails, the object still needs to be</td>
</tr>
<tr>
<td>with it, and then disposes of it.</td>
<td>disposed of.</td>
</tr>
<tr>
<td>The program locks a handle, does something with</td>
<td>If the something fails, the handle still needs to</td>
</tr>
<tr>
<td>it, and unlocks it.</td>
<td>be unlocked.</td>
</tr>
</tbody>
</table>

The TCL handles exceptions and cleanup using three macros: TRY, CATCH, and ENDTTRY. Here is how you use them:

```
TRY {
   code that could generate an exception
}
CATCH {
   code that cleans up if an exception occurs
}
ENDTRY;
```

To clarify the functioning of these macros, first examine a case that does not fail. If the code in TRY does not fail, the code in CATCH is not executed (sorry, but double negatives are like that). If CATCH is not executed, the code continues after ENDTTRY. On the other hand, examine a failing case: If an exception occurs (TRY fails), the CATCH code executes. After the last line of CATCH finishes, the stack of currently executing functions is aborted all the way back to the next exception handler.
Handling Exceptions in CreateDocument

Now, imagine handling errors while trying to create a document. In the function, CreateDocument, if an error occurs during the initialization of the document or in the call to NewFile, you need to throw the document away. Here is the code you must add to CreateDocument to perform that action:

```c++
void CTextApp::CreateDocument()
{
    CTextDoc *theDocument = new CTextDoc;

    TRY
    {
        theDocument->NewFile();
    }
    CATCH
    {
        TCLForgetObject(theDocument);
    }
    ENDTRY
}
```

Notice that, if NewFile cause an exception, the document is thrown away.

You add exception handling to the OpenDocument member function similarly:

```c++
void CTextApp::OpenDocument(SFReply *macReply)
{
    CTextDoc *theDocument = new CTextDoc;

    TRY
    {
        theDocument->OpenFile(macReply);
    }
    CATCH
    {
        TCLForgetObject(theDocument);
    }
    ENDTRY;
}
```

There are some differences between OpenDocument and CreateDocument:
The presence of the SFReply (obtained from the user in the Open dialog).

The call to OpenFile rather than NewFile.

The TCL shows all the available files in an Open dialog. The user of a text-editing program, however, only wants to see the available text files. TextTyper, therefore, needs to modify the TCL's default behavior to show only 'TEXT' files:

```cpp
void CTextApp::SetUpFileParameters()
{
    CApplication::SetUpFileParameters();

    sfNumTypes = 1;
    sfFileTypes[0] = 'TEXT';
    gSignature = 'C++2';
}
```

The overriding member function augments the existing behavior of CApplication. It sets the data members sfNumTypes and sfFileTypes, which are later used as parameters to the Toolbox call SFGet-File. After this is accomplished, TextTyper only shows text files when the user selects the Open dialog.

---

**NOTE**

This is also a good place to set the signature for the application because the signature is a file-related piece of information. Alternatively, the signature could be set in the TextApp constructor.

---

**The CTextDoc Class**

The CTextDoc class has these duties in TextTyper:

- Create the window.
- Create the panes within the window.
- Read the text from disk.
- Write the text to disk.
Following is the class declaration:

class CTextDoc: public CDocument {
public:
    virtual void NewFile();
    virtual void OpenFile(SFReply *macReply);
    virtual Boolean DoSave();
    virtual Boolean DoSaveAs(SFReply *macSFReply);
protected:
    virtual void BuildWindow();
private:
    CEditText *fEditText;
};

The member functions NewFile, OpenFile, DoSave, and DoSaveAs are all overrides of CDocument member functions. BuildWindow is a utility member function that encapsulates common code from NewFile and OpenFile. The sole data member, fEditText, refers to the CEditText pane in the document's window.

Creating a New File

When the user wants to create a new text document, CTextApp calls the member function NewFile (existing documents open differently). Following is the NewFile code:

```cpp
void CTextDoc::NewFile()
{
    BuildWindow();

    Str255 wTitle;
    short wCount;
    Str255 wNumber;

    itsWindow->GetTitle(wTitle);
    wCount = gDecorator->GetWCount();
    ::NumToString(wCount, wNumber);
    ::ConcatPStrings(wTitle, "\p-"),
    ::ConcatPStrings(wTitle, wNumber);
    itsWindow->SetTitle(wTitle);
    itsWindow->Select();
}
```
NewFile calls BuildWindow to create the necessary windows and panes. The TCL, by default, sets the title of new windows to the oh-so-familiar “Untitled.” TextTyper is a bit more sophisticated than this, however, because it sets the titles to “Untitled-1,” “Untitled-2,” and so on. To do this, NewFile sends a message to the global gDecorator asking how many windows have been created. It takes this number and attaches it to the end of the untitled document it just created. So the user does not have to watch a schizoid window, this title is not displayed until its Select member function is called.

Building the Window

The BuildWindow member function creates the window and the necessary panes for TextTyper:

```cpp
void CTextDoc::BuildWindow()
{
    const short kWINDResourceID = 500;
    const Boolean kWindowFloats = TRUE;
    const Boolean kFitHorizontal = TRUE;
    const Boolean kFitVertical = TRUE;
    const Boolean kHasHorizontalScrollbar = TRUE;
    const Boolean kHasVerticalScrollbar = TRUE;
    const Boolean kHasGrowBox = TRUE;
    const short kEditTextWidth = 72 * 6;
    const Boolean kRedraw = TRUE;

    CWindow *aWindow = new CWindow(kWINDResourceID, kWindowFloats, this);
    itsWindow = aWindow;
    CScrollPane *theScrollPane = new CScrollPane(aWindow, this, 10, 10, 0, 0,
        sizELASTIC, sizELASTIC,
        kHasHorizontalScrollbar, kHasVerticalScrollbar, kHasGrowBox);
    theScrollPane->FitToEnclosure(kFitHorizontal, kFitVertical);

    CEditText *thePane = new CEditText(theScrollPane, this, 1, 1, 0, 0,
        sizELASTIC, sizELASTIC, kEditTextWidth);
    itsGopher = thePane;
    itsMainPane = thePane;
    fEditText = thePane;
    theScrollPane->InstallPanorama(thePane);
    thePane->FitToEnclosure(kFitHorizontal, kFitVertical);
}```
Rect margin;
::SetRect(&margin, 2, 2, -2, -2);
thePane->ChangeSize(&margin, !kRedraw);

gDecorator->PlaceNewWindow(itsWindow);
}

This is a long function, so you can examine it a bit at a time. First, peruse the declarations to get familiar with them. A more detailed discussion follows.

- kWINDResourceID is the ID of the 'WIND' resource.
- kWindowFloats makes the window float (windoid).
- kFitHorizontal makes this pane the same horizontal size as its enclosure.
- kFitVertical makes this pane the same vertical size as its enclosure.
- kHasHorizontalScrollbar signals the panorama to create a horizontal scroll bar.
- kHasVerticalScrollbar signals the panorama to create a vertical scroll bar.
- kHasGrowBox ensures the panorama has a grow box.
- kEditTextWidth sets the allowable width of the edit text.
- kRedraw redraws this pane.

Now see how BuildWindow handles window scrolling. A TCL application uses two special panes to support scrolling: CScrollPane and CPanorama. A CPanorama is a scrollable CPane. Here is the process of creating a window:

1. BuildWindow creates a CWindow.
2. Next, it creates a CScrollPane, specifying horizontal and vertical scroll bars and a grow box. The FitToEnc1Frame function resizes the scroll pane to match the size of the window.
3. Then, BuildWindow creates an editable text with a width of 6 inches.
4. The itsMainPane data member determines which pane prints when the user selects Print.
5. CEditText also is stored in fEditText, a CTextDoc data member. This is used for document saving.
6. The call to InstallPanorama notifies the scroll pane of its panorama (the edit text).
7. Then, the edit text is resized to fit the window.

8. The edit text is next inset by 2 pixels on each side. This way it cannot bump up against the edge of the window.

9. Finally, PlaceNewWindow staggers the window by adjusting its size and location.

NOTE

The itsGopher data member determines the chain of command. Chains of command, along with gophers, are described in detail in Chapter 16.

After BuildWindow executes, TextTyper has a new window ready for text.

Opening an Existing File

If a document already exists, then TextTyper must use a different process. CTextDoc calls its OpenFile member function to open an existing file. It is specified with an SFReply parameter:

```cpp
void CTextDoc::OpenFile(SFReply *macReply)
{
    CDataFile *theDataFile;
    Handle theData;
    Str63 theName;
    OSErr theError;
    const long kMaxFileSize = 32000;

    theDataFile = new CDataFile;

    itsFile = theDataFile;

    theDataFile->SFSpecify(macReply);

    theDataFile->Open(fsRdPerm);

    TRY {
        if (theDataFile->GetLength() > kMaxFileSize)
            ::Failure(paramErr, excExceedTELim);
        theData = theDataFile->ReadAll();
    }
}
```
CDataFile is a TCL class that interacts with a data file on disk. Note that OpenFile creates a CDataFile and stores it in the CDocument data member, itsFile. It waits there until it is time to save the document.

When given an SFReply, SFSpecify notifies the data file of its location. OpenFile then opens the file, reads its data with ReadAll, and closes it.

Note that the code uses the exception-handling macros: TRY, CATCH, and ENDTRY. This exception handler, which encompasses ReadAll, ensures that the file is closed even if there is an error reading the file. The check for file length makes sure that the file is not too long for TextTyper.

After reading the text and closing the file, OpenFile builds the window. Next, it calls the SetTextHandle member function, which provides the text to the CEditText pane (the exception handler is here to dispose of the handle in case of an error).

Finally, OpenFile sets the title of the window appropriately, and then calls Select to show the window to the user.
Handling the Save Option

When the user selects Save, CDocument calls the DoSave member function:

```c
Boolean CTextDoc::DoSave()
{
    Handle theData;

    if (itsFile == NULL)
        return(DoSaveFileAs());
    else {
        CDataFile *theDataFile = (CDataFile *) itsFile;

        theData = fEditText->GetTextHandle();
        theDataFile->Open(fsWrPerm);
        TRY {
            theDataFile->WriteAll(theData);
        } CATCH {
            theDataFile->Close();
        } ENDTRY;
        theDataFile->Close();
        dirty = FALSE;
        return(TRUE);
    }
}
```

If the user is still working with an untitled document (one not yet saved) then DoSave calls DoSaveFileAs. This generates a file name prompt for the user. When the file already exists, DoSave does the following:

- Gets the data from CEditText.
- Opens the file.
- Writes the data.
- Closes the file.

Again, an exception handler is necessary for cleanup.
After the file is saved, the file's dirty data member is reset to false. This flag enables Save and controls whether the Save changes before the closing alert dialog appears when the document is closed.

The return value of DoSave is only set to false if the user cancels the Save (as might happen if he or she cancels the SFPutFile for an untitled document).

Handling the Save As Option

When the user selects Save As, CDocument calls the member function DoSaveAs. It also passes the SFReply to specify where the document should be saved. DoSaveAs is as follows:

```c++
Boolean CTextDoc::DoSaveAs(SFReply *macSFReply)
{
    if (itsFile != NULL)
        TCLForgetObject(itsFile);

    CDataFile *theDataFile = new CDataFile;
    itsFile = theDataFile;
    theDataFile->SFSpecify(macSFReply);
    if (theDataFile->ExistsOnDisk())
        theDataFile->ThrowOut();
    theDataFile->CreateNew(gSignature, 'TEXT');

    itsWindow->SetTitle(macSFReply->fName);

    return(DoSave());
}
```

First, the itsFile data member determines if the file exists. If it already exists, then DoSaveAs disposes of it. DoSaveAs then creates a new data file and saves it in the data member itsFile.

Second, the function deals with the case of an existing file that needs replacement. If it exists, DoSaveAs deletes it then creates the new file using CreateNew. The appropriate signature and file type also are set.

Third, DoSaveAs updates the window's title to reflect the new file name. It then calls DoSave to actually write the contents of the file.
TextTyper’s Main Program

Here is another very similar TCL main program:

```c
void main()
{
    CTextApp *app;
    app = new CTextApp;
    app->Run();
    app->Exit();
}
```

The Resources

Other than its different icons, the only new resource TextTyper has is a modified version of the edit menu:

```c
resource 'MENU' (3, "Edit", preload) {
    3,
    textMenuProc,
    0x7FFFFF80,
    enabled,
    "Edit",
    {
        "Undo#16", noIcon, "Z", noMark, plain,
        ".", noIcon, noKey, noMark, plain,
        "Cut#18", noIcon, "X", noMark, plain,
        "Copy#19", noIcon, "C", noMark, plain,
        "Paste#20", noIcon, "V", noMark, plain,
        "Clear#21", noIcon, noKey, noMark, plain,
        "Select All#23", noIcon, "A", noMark, plain,
        ".", noIcon, noKey, noMark, plain,
        "Show Clipboard#22", noIcon, noKey, noMark, plain
    }
};
```
The Select All menu item has been added to the Edit menu. Notice that all the menu items have a number after them. These numbers are not displayed in the running application. They are command numbers, which are used within the TCL to refer to the menu item. This is a much more convenient method than using the menu ID and item number. Using command numbers enables you to modify a program more easily because you can move a menu item within a menu, or even into another menu, without recompiling your code.

Summary

TextTyper, the application featured in this chapter, has two new classes: CTextApp and CTextDoc. With these, you can use the TCL to create a program that does the following:

- Creates a new text document (CreateDocument and NewFile).
- Opens an existing document (OpenDocument and OpenFile).
- Sets up the windows to look and work correctly (BuildWindow).
- Names and saves a document (DoSave and DoSaveAs).

You also learned about an important new way to deal with errors: exception handling.
he application in this chapter, PICTPeeker, is a PICT-viewing program that enables the user to open, print, cut, copy, and paste. It contains new classes that handle undo and redo as well as enable more menus. It also contains important new exception-handling classes and uses a feature of C++ known as templates.

The PICTPeeker Program

PICTPeeker is yet another kind of program you can build with the TCL. Although still on the simpler side of the application spectrum, it nevertheless opens multiple documents and enables cut, copy, paste, and print operations. PICTPeeker is built with an augmented version of the TCL class that displays pictures. Figure 18.1 shows the application in action.
Compiling and Running the Program

Copy the PICTPeeker folder from the source code disk. Open the PICTPeeker.π project and choose Run. It takes a few minutes for the compiler to recompile all the files in this project.

After it is running, put the program through its paces. Feel free to try any of the following:

- Create some new files.
- Open some existing PICT files.
- Cut, copy, and paste a PICT file.
- Undo and redo what you just did.
- Show the clipboard during these operations to convince yourself that it works correctly.
- Look at the About box.

After you have examined PICTPeeker, it is time to study the code.

Inspecting the Program

First, look at the class diagram in figure 18.2. Spend a few minutes studying it because there are several new classes.
Figure 18.2. The PICTPeeker class diagram.
Each of these classes is covered in detail later in this chapter. For now, look at the following list to get the gist of what each class controls:

- **CAboutBoxApplication** displays the About box.
- **CPictureApp** creates new documents and sets parameters for the Open dialog box.
- **CPictureDoc** creates the window and picture pane and reads and writes the PICT to disk.
- **CCutCopyPastePicture** is a picture object that supports cut, copy, paste, and clear functions.
- **CPictureClipboardTask** handles doing, undoing, and redoing cut, copy, paste, and clear functions.
- **THandleFailure** is an abstract class used for creating exception-handling classes.
- **TDeleteOnFailure** frees an object if an exception occurs during its scope.
- **TDisposHandleOnFailure** disposes of a handle if an exception occurs during its scope.
- **THandleLocker** locks a handle and restores its state when its scope is exited, even if an exception occurs.
- **TWithFileOpen** opens a file and closes it when its scope is exited, even if an exception occurs.

**NOTE**

If you skipped Chapter 16, pay special attention to this note. The THINK Class Library uses a naming convention different from that described in Chapter 10. The TCL class names begin with c, or BR_ instead of T, as you might expect. There is no particular naming convention for data members in the TCL. In the interest of maintaining convention, Chapters 16 to 21 (which use the TCL) begin class names with c for all the classes derived from TCL classes. Any new data members begin with f.

Let's look at the new class that handles the display of the About box first.
The CAboutBoxApplication Class

When the user chooses the About menu item, the CAboutBoxApplication class displays the About box. This new class also serves as the base class for PICTPeeker itself:

```cpp
class CAboutBoxApplication: public CApplication {
public:
    CAboutBoxApplication(short extraMasters, Size aRainyDayFund, Size aCriticalBalance, Size aToolboxBalance);
    virtual void DoCommand(long theCommand);
};
```

The constructor, CAboutBoxApplication, is quite straightforward as well:

```cpp
CAboutBoxApplication::CAboutBoxApplication(short extraMasters, Size aRainyDayFund, Size aCriticalBalance, Size aToolboxBalance) :
    CApplication(extraMasters,aRainyDayFund, aCriticalBalance, aToolboxBalance)
{
    TCL_END_CONSTRUCTOR
}
```

In the TCL, when a menu item is selected—whether by mouse or by a command key equivalent—the member function DoCommand is called for each bureaucrat in the chain of command. CAboutBoxApplication overrides DoCommand. It is on the lookout for the user selecting the About menu item:

```cpp
void CAboutBoxApplication::DoCommand(long theCommand)
{
    const short kAboutBoxResourceID = 1024;
    switch (theCommand) {
    case cmdAbout:
        Str255 applicationName;
        short appRefNum;
        Handle h;
        ::PositionDialog('ALRT', kAboutBoxResourceID);
        ::GetAppParms(applicationName, &appRefNum, &h);
        ::DisposHandle(h);
        ::ParamText(applicationName, NULL, NULL, NULL);
        Alert(kAboutBoxResourceID, NULL);
    };
```
break;
default:
    CApplication::DoCommand(theCommand);
    break;
}
}

DoCommand puts up the alert with ID 1024. This alert is customized to include the application name with the Toolbox call ParamText. The ALERT resource is shown in figure 18.3.

![Alert Resource ID](image)

**Figure 18.3.** The About box alert in ResEdit.

The string ‘0 in the alert is replaced with the application name by the call to ParamText. When you look at this alert in the running program it transforms into the dialog box shown in figure 18.4.

![About Box Alert](image)

**Figure 18.4.** The About box alert as seen in PICTPeeker.

When the incoming command is not cmdAbout, the command is passed on to the default DoCommand.
The CPictureApp Class

The CPictureApp class takes advantage of the behavior of CAboutBoxApplication by deriving from it:

class CPictureApp: public CAboutBoxApplication {
public:
    CPictureApp();
virtual void CreateDocument();
virtual void OpenDocument(SFReply *macReply);
virtual void SetUpFileParameters();
};

The member functions also might look familiar. They are quite similar to the ones used in TextTyper (discussed in Chapter 17). Table 18.1 compares TextTyper’s functions with those in this chapter. The following are CAboutBoxApplication’s member functions:

const short kExtraMasters = 1;
const Size kRainyDayFund = 45000;
const Size kCriticalBalance = 40000;
const Size kToolboxBalance = 20000;

CPictureApp::CPictureApp() :
    CAboutBoxApplication(kExtraMasters, kRainyDayFund,
        kCriticalBalance, kToolboxBalance)
{
    TCL_END_CONSTRUCTOR
}

void CPictureApp::CreateDocument()
{
    CPictureDoc *theDocument = new CPictureDoc;

    TDeleteOnFailure<CPictureDoc> freer(theDocument);

    theDocument->NewFile();
    freer.Success();
}

void CPictureApp::OpenDocument(SFReply *macReply)
{
    CPictureDoc *theDocument = new CPictureDoc;
TDeleteOnFailure<CPictureDoc> freer(theDocument);

theDocument->OpenFile(macReply);
freer.Success();
}

void CPictureApp::SetUpFileParameters()
{
CAppllication::SetUpFileParameters();

sfNumTypes = 1;
sfFileTypes[0] = 'PICT';
gSignature = 'C++3';
}

If you look at table 18.1, you see that CPictureApp and CTextApp are identical except for their names. The SetUpFileParameters function also is similar, except that the file type is not 'TXT' but 'PICT' and that PICTPeeker has a different signature. The substantial differences lie in the member function's handling of exceptions. The PICTPeeker code does not contain the exception handlers (TRY, CATCH, and ENDTRY) in CreateDocument and OpenDocument. Instead, these two routines use an object of the class TDeleteOnFailure. The object makes sure that theDocument is disposed of if exceptions occur between the declaration of the object and the end of the routine.

Table 18.1. Comparing PICTPeeker with TextTyper Member Functions

<table>
<thead>
<tr>
<th>PICTPeeker Functions</th>
<th>TextTyper Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPictureApp::CPictureApp(): CApplication(kExtraMasters, kRainyDayFund, kCriticalBalance, kToolboxBalance);</td>
<td>CTextApp::CTextApp(): CApplication(kExtraMasters, kRainyDayFund, kCriticalBalance, kToolboxBalance);</td>
</tr>
<tr>
<td>void CPictureApp::CreateDocument()</td>
<td>void CTextApp::CreateDocument()</td>
</tr>
<tr>
<td>{ CPictureDoc *theDocument = new CPictureDoc;</td>
<td>{ CTextDoc *theDocument = new CTextDoc;</td>
</tr>
</tbody>
</table>
...
The TDeleteOnFailure Class

TDeleteOnFailure is a new class that handles exceptions:

template<class T> class TDeleteOnFailure: public THandleFailure {
public:
    TDeleteOnFailure(T *object)
    {
        fObject = object;
        TCL_END_CONSTRUCTOR
    }
    ~TDeleteOnFailure()
    {
        TCL_START_DESTRUCTOR
        if (!fSucceeded)
            delete fObject;
    }
private:
    T *fObject;
};

The TDeleteOnFailure class keeps track of an object and deletes it if an exception occurs. This class is a template class. This means that it can be instantiated for any class of object rather than just for a specific class, like TTextDoc or TPictureDoc.

Templates

Templates are one of the newer features of C++. In the early days of C++, if you wanted to create a class that implemented a generic function, such as a heap, you could create a base class THeap. You might pick a first data type, say integers, that the heap would store. But then you would find that you needed a heap to store floating-point numbers. With the strong type checking in C++, you couldn’t just pass in a new parameter type because the types don’t match.

This resulted in some class designs in which more functionality was put in derived classes lower in the hierarchy and there was less shared code. Functions that were different only in the type of the data would be implemented in full in derived classes that knew about the type.

C++ now supports templates, which are a way of passing the type of an object when creating a new object. For example, in the past you might have had a set of list classes TIntegerList, TFloatList, TCharList in which most of the functions were identical (and each of these probably was derived from a generic TList class); now you can create a single TList template class that supports any type of object passed into it.
template <class T> TList
{
public:
    void AddToList (T *item);
    void DeleteFromList (T *item);
}

Rather than having a specific type as the arguments to the AddToList and DeleteFromList
functions, the types are declared as T. When you need a list of integers, you declare it with
TList<integer> anIntegerList;. The type passed in the <>’s is substituted wherever the T occurs
in the definition. So item is declared as an integer argument to AddToList and DeleteFromList
when this class is used as shown here. If an incompatible type of argument was later passed to
to this integer list through one of these functions, a type mismatch error would be signaled.

The THHandleFailure Class

THHandleFailure is an abstract class that serves as a foundation for other beneficial classes:

class THHandleFailure TCL_AUTO_DESTRUCT_OBJECT {
public:
    TCL_DECLARE_CLASS
    THHandleFailure();
    ~THHandleFailure(); // stack-based, don't need virtual

    void Success();
protected:
    Boolean fSucceeded;
};

Following are the constructor, destructor, and Success function:

THHandleFailure::THHandleFailure()
{
    fSucceeded = FALSE;
    TCL_END_CONSTRUCTOR
}

THHandleFailure::~THHandleFailure()
{
    TCL_START_DESTRUCTOR
}

void THHandleFailure::Success()
{

367

Chapter 18: Exception Handling and Templates: PICTPeeker
The THandleFailure class is used with objects that are created on the stack and that must be correctly destructed and removed from the stack if exceptions occur. Normally, these would be cleaned up with the regular stack cleanup at the end of a function's execution. THandleFailure is derived from TCL_AUTO_DESTRUCT_OBJECT, which is described in the following section.

The TDisposHandleOnFailure Class

The TDisposHandleOnFailure class is similar to the TDeleteOnFailure class; however, rather than disposing an object in case of an exception, this class disposes a handle:

class TDisposHandleOnFailure: public THandleFailure {
public:
    TCL_DECLARE_CLASS
    TDisposHandleOnFailure(Handle h);
    ~TDisposHandleOnFailure();
private:
    Handle fHandle;
};

TDisposHandleOnFailure::TDisposHandleOnFailure(Handle h)
{
    fHandle = h;
    TCL_END_CONSTRUCTOR
}

TDisposHandleOnFailure::~TDisposHandleOnFailure()
{
    TCL_START_DESTRUCTOR
    if (!fSucceeded && fHandle != NULL)
        ::DisposeHandle(fHandle);
}

You see how TDisposHandleOnFailure is used later in this chapter in CPictureDoc::OpenFile.
The TCL_AUTO_DESTRUCT_OBJECT Class

TCLAUTO_DESTRUCT_OBJECT is an abstract class. Derived classes inherit the following behavior: If an exception occurs during the lifetime of a class, then its destructor is called. Such derived classes should not be allocated dynamically; instead, they should be declared globally or as automatic (stack-based) variables.

IMPORTANT

ANSI C++ adds exception handling to C++. After C++ compilers implement this, the destructors of stack-based objects automatically are called when exceptions occur. This leads to the use of resource acquisition classes, a method of handling exceptions. These classes have constructors that acquire a resource and destructors that release the resource.

The resource acquisition class idiom is used by classes derived from TCL_AUTO_DESTRUCT_OBJECT.

TCL_AUTO_DESTRUCT_OBJECT has a number of derived classes based on it. The next sections take a look at each of these.

The THandleLocker Class

The THandleLocker class locks a handle in its constructor, and then restores the handle state when its destructor is called:

class THandleLocker TCL_AUTO_DESTRUCT_OBJECT {
public:
    TCL_DECLARE_CLASS
    THandleLocker(Handle h);
    ~THandleLocker();

private:
    Handle fHandle;
    char fState;
};

THandleLocker::THandleLocker(Handle h)
{
    fHandle = h;
    fState = ::HGetState(fHandle);
The destructor calls HSetState to restore the handle’s state, rather than using HUnlock. It performs this way because the handle cannot start out locked.

**The TWithFileOpen Class**

The TWithFileOpen class has a constructor that opens a file and destructor that closes it:

```cpp
class TWithFileOpen TCL_AUTO_DESTRUCT_OBJECT {
public:
   TCL_DECLARE_CLASS
   TWithFileOpen(CFile *theFile, SignedByte permission);
   -TWithFileOpen();

private:
   CFile *fFile;
};

TWithFileOpen::TWithFileOpen(CFile *theFile, SignedByte permission)
{
   theFile->Open(permission);
   fFile = theFile;
}

TWithFileOpen::-TWithFileOpen()
{
   if (fFile != NULL)
      fFile->Close();
}
```

That is all there is to the exception classes. You can now use these sparkling new classes to simplify dealing with exceptions in other parts of the program.
The CPictureDoc Class

The CPictureDoc class is quite similar to the CTextDoc class of Chapter 17. It manipulates a PICT document, however, instead of text:

class CPictureDoc: public CDocument {
public:
    virtual void NewFile();
    virtual void OpenFile(SFReply *macReply);
    virtual Boolean DoSave();
    virtual Boolean DoSaveAs(SFReply *macSFReply);
protected:
    virtual void BuildWindow();
private:
    CPicture *fPicture;
};

void CPictureDoc::NewFile()
{
    BuildWindow();

    Str255 wTitle;
    short wCount;
    Str255 wNumber;

    itsWindow->GetTitle(wTitle);
    wCount = gDecorator->GetWCount();
    ::NumToString(wCount, wNumber);
    ::ConcatPStrings(wTitle, "\p");
    ::ConcatPStrings(wTitle, wNumber);
    itsWindow->SetTitle(wTitle);
}

void CPictureDoc::BuildWindow()
{
    const short kWINDResourceID = 500;
    const Boolean kWindowFloats = TRUE;
    const Boolean kFitHorizontal = TRUE;
    const Boolean kFitVertical = TRUE;
    const Boolean kHasHorizontalScrollbar = TRUE;
    const Boolean kHasVerticalScrollbar = TRUE;
const Boolean kHasGrowBox = TRUE;
const Boolean kRedraw = TRUE;

CWindow *aWindow = new CWindow(kWINDResourceID, !kWindowFloats, this);
itsWindow = aWindow;
CScrollPane *theScrollPane = new CScrollPane(aWindow, this, 10, 10, 0, 0,
sizELASTIC, sizELASTIC,
kHasHorizontalScrollbar, kHasVerticalScrollbar, kHasGrowBox);
theScrollPane->FitToEnclosure(kFitHorizontal, kFitVertical);

CCutCopyPastePicture *thePane =
new CCutCopyPastePicture(theScrollPane, this, 1, 1, 0, 0,
sizELASTIC, sizELASTIC);
itsGopher = thePane;
itsMainPane = thePane;
fPicture = thePane;
theScrollPane->InstallPanorama(thePane);
thePane->FitToEnclosure(kFitHorizontal, kFitVertical);

gDecorator->PlaceNewWindow(itsWindow);
}

Boolean CPictureDoc::DoSaveAs(SFReply *macSFReply)
{
    if (itsFile != NULL)
        delete itsFile;

    CDataFile *theDataFile = new CDataFile;
    itsFile = theDataFile;
    theDataFile->SFSpecify(macSFReply);
    if (theDataFile->ExistsOnDisk())
        theDataFile->ThrowOut();
    theDataFile->CreateNew(gSignature, 'PICT');

    itsWindow->SetTitle(macSFReply->fName);

    return(DoSave());
}

Look at the differences. Here, BuildWindow creates a TCutCopyPastePicture instead of a TEditText. There are more pronounced differences in DoSave and OpenFile:
void CPictureDoc::OpenFile(SFReply *macReply)
{
    CDataFile  *theDataFile;
    Handle theData;
    Str63 theName;
    OSErr theError;

    theDataFile = new CDataFile;

    itsFile = theDataFile;

    theDataFile->SFSpecify(macReply);

    Handle theMacPicture = ::NewHandleCanFail(0);
    ::FailNIL(theMacPicture);
    {
        TWithFileOpen fileOpener(theDataFile, fsRdPerm);
        TDisposHandleOnFailure handleFreer(theMacPicture);

        long thePictureLength = theDataFile->GetLength() - kPICTHeaderLength;
        if (thePictureLength <= 0)
            ::FailOSErr(eofErr);
        theDataFile->SetMark(kPICTHeaderLength, fsFromStart);
        ::ResizeHandleCanFail(theMacPicture, thePictureLength);
        ::FailMemError();
        {
            THandleLocker lock(theMacPicture);
            theDataFile->ReadSome(*theMacPicture, thePictureLength);
        }
        BuildWindow();
        handleFreer.Success();
    }
    fPicture->SetMacPicture((PicHandle) theMacPicture);

    itsFile->GetName(theName);
    itsWindow->SetTitle(theName);

    itsWindow->Select();
}

Boolean CPictureDoc::DoSave()
{

Handle theData;

if (itsFile == NULL)
    return(DoSaveFileAs());
else {
    CDataFile *theDataFile = (CDataFile *) itsFile;
    Handle theMacPicture = (Handle) fPicture->GetMacPicture();
    {
        TWithFileOpen fileOpener(theDataFile, fsWrPerm);
        char zeros[kPICTHeaderLength];

        for (int i = 0; i < kPICTHeaderLength; i++)
            zeros[i] = 0;
        theDataFile->WriteSome(zeros, sizeof(zeros));
        THandleLocker lock(theMacPicture);
        theDataFile->WriteSome(*theMacPicture, ::GetHandleSize(theMacPicture));
    }
    dirty = FALSE;
    return(TRUE);
}

Both DoSave and OpenFile extensively use the exception-handling classes just created. The functions are complicated further by the fact that the PICT files have a 512-byte header that must be skipped or written. The basic structure, however, is the same as the corresponding functions in CTextDoc.

The **CCutCopyPastePicture Class**

The **CCutCopyPastePicture** class is derived from the TCL class **CPicture** and adds support for the cut, copy, paste, and clear menu items:

```cpp
class CCutCopyPastePicture: public CPicture {
    public:
        CCutCopyPastePicture(CView *anEnclosure, CBureaucrat *aSupervisor,
                      short aWidth, short aHeight, short aHEncl, short aVEncl,
                      SizingOption aHSizing, SizingOption aVSizing);

        virtual void UpdateMenus();
        virtual void DoCommand(long cmd);
};
```

374

Symantec C++ Programming for the Macintosh, 2nd Edition
Its constructor is simple:

```cpp
CCutCopyPastePicture::CCutCopyPastePicture(CView *anEnclosure,
   CBureaucrat *aSupervisor,
   short aWidth, short aHeight, short aHEnc1, short aVEnc1,
   SizingOption aHSizing, SizingOption aVSizing) :
   CPicture(anEnclosure, aSupervisor, aWidth, aHeight,
            aHEnc1, aVEnc1, aHSizing, aVSizing)
{
   TCL_END_CONSTRUCTOR
}
```

It also enables the clipboard menu items:

```cpp
void CCutCopyPastePicture::UpdateMenus()
{
   CPicture::UpdateMenus();

   if (GetMacPicture() != NULL) {
      gBartender->EnableCmd(cmdCopy);
      gBartender->EnableCmd(cmdCut);
      gBartender->EnableCmd(cmdClear);
   }

   if (gClipboard->GetData('PICT', NULL))
      gBartender->EnableCmd(cmdPaste);
}
```

`UpdateMenus` is called to enable menu items when the user clicks on the menu bar (or uses a command key equivalent). The TCL dims the menu items before calls to `UpdateMenus`. This function also can selectively enable certain menu items and ignore the rest. The TCL ensures that ignored items remain disabled.

`CBartender` is a class that handles the menu bar. The global variable `gBartender` is the one and only `CBartender` object. Its member function, `GetMacPicture`, enables cut, copy, and clear if there is currently a picture. It also enables paste if the clipboard contains PICT data.

In tandem with `UpdateMenus`, `CCutCopyPastePicture` must actually carry out the menu commands with `DoCommand`:

```cpp
void CCutCopyPastePicture::DoCommand(long cmd)
{
   switch (cmd) {
      case cmdCopy:
```
case cmdCut:
case cmdClear:
case cmdPaste:
    CPictureClipboardTask *clipTask = new CPictureClipboardTask(this, cmd);
    itsLastTask = clipTask;
    if (cmd == cmdCopy)
        itsSupervisor->NotifyClean(clipTask);
    else
        itsSupervisor->Notify(clipTask);
    clipTask->Do();
    break;

default:
    CPicture::DoCommand(cmd);
}

The CPictureClipboardTask class handles doing, undoing, and redoing of cut, copy, paste, and clear operations. DoCommand must create one of these objects and instruct it to do its task.

The TCL handles undo and redo using the CPane data member itsLastTask (in which the member function saves its task).

Notify and NotifyClean alert the supervisor that a task is going to be executed. The document, in turn, responds to Notify by setting its dirty data member to true. NotifyClean, on the other hand, does not change dirty.

**The CPictureClipboardTask Class**

The actual code for CPictureClipboardTask is as follows:

class CPictureClipboardTask: public CTask {
public:

    CPictureClipboardTask(CPicture *thePicture, long command);

virtual ~CPictureClipboardTask();
virtual void Do();
virtual void Undo();
virtual void Redo();

private:

    CPicture *fPicture;
    long    fCommand;
Derived classes of CTask must override Do, Undo, and Redo to carry out, undo, and redo their operations.

In this class, the data member, fOldScrap, holds the old contents of the scrap, and fOldMacPicture holds the old PicHandle in the CPicture.

**Constructor**

The constructor calls the superclass constructor and initializes the data members:

```cpp
CPictureClipboardTask::CPictureClipboardTask(CPicture *thePicture,
   long command) :
   CTask(command - cmdCut + 2)
{
   fOldMacPicture = NULL;
   fOldScrap = NULL;
   fPicture = thePicture;
   fCommand = command;
   TCL_END_CONSTRUCTOR
}
```

The parameter to CTask is an index from the task names 'STR#' resource. This index contains text for the Undo menu. Thus, when the user chooses Cut, the Undo menu shows “Undo Cut” rather than the less descriptive “Undo.”

**Doing the Command**

The following is the function that actually carries out the command:

```cpp
void CPictureClipboardTask::Do()
{
   if (fCommand != cmdClear) {
      Handle h;
      gClipboard->GetData('PICT', &h);
      fOldScrap = h;
   }
   fOldMacPicture = fPicture->GetMacPicture();
   Redo();
}
```
Unless the command is Clear, Do saves the old clipboard. This is needed for Undo. Do then saves the current PicHandle and calls Redo, which contains the code common to both Do and Redo.

**Redoing the Command**

Redo is called by Do the first time a command is executed. It also is called whenever the user selects Redo:

```cpp
void CPictureClipboardTask::Redo()
{
    if (fCommand == cmdCopy || fCommand == cmdCut) {
        gClipboard->EmptyScrap();
        gClipboard->PutData('PICT', (Handle) fOldMacPicture);
    }
    if (fCommand == cmdCut || fCommand == cmdClear) {
        fPicture->SetPosition((LongPt *) &kZeroZeroPt);
        fPicture->SetMacPicture(NULL);
    }
    if (fCommand == cmdPaste) {
        fPicture->SetPosition((LongPt *) &kZeroZeroPt);
        fPicture->SetMacPicture((PicHandle) fOldScrap);
    }
    if (fCommand != cmdCopy)
        fPicture->Refresh();
    undone = FALSE;
}
```

Redo modifies the clipboard and the picture. Last, it sets the CTask data member undone to FALSE, signifying that the command has not been undone. This data member is, in turn, used by the destructor.

**IMPORTANT**

This is an important function, so read through this code four times, once for each different command. Be certain that you understand how it works.

**Undoing the Command**

The member function Undo restores any change to the clipboard and any change to the PicHandle:
void CPictureClipboardTask::Undo()
{
    if (fCommand == cmdCopy || fCommand == cmdCut) {
        gClipboard->EmptyScrap();
        if (fOldScrap != NULL)
            gClipboard->PutData('PICT', fOldScrap);
    }
    if (fCommand != cmdCopy) {
        fPicture->SetPosition((LongPt *)&kZeroZeroPt);
        fPicture->SetMacPicture(fOldMacPicture);
        fPicture->Refresh();
    }
    undone = TRUE;
}

Notice that this time the CTask data member undone is set to true. This signifies that the task has indeed been undone.

Disposing of Data

As a new undoable task is created, any existing undoable task is jettisoned—it is no longer undoable. When a task can no longer be undone or redone, the TCL gets rid of it in the destructor:

CPictureClipboardTask::~CPictureClipboardTask()
{
    TCL_START_DESTRUCTOR
    if (!IsUndone()) {
        switch (fCommand) {
            case cmdCut:
                ::DisposHandle((Handle) fOldMacPicture);
                ::DisposHandle(fOldScrap);
                break;
            case cmdClear:
                ::DisposHandle((Handle) fOldMacPicture);
                break;
            case cmdPaste:
                ::DisposHandle((Handle) fOldMacPicture);
                // old scrap is now the picture
                break;
            case cmdCopy:
                // old picture remains the picture
                break;
        }
    }
}
::DisposHandle(fOldScrap);
break;
}
} else
if (fOldScrap != NULL)
::DisposHandle(fOldScrap);

Notice that the function disposes of data differently, depending on what command is executed and whether the task has been undone.

PICTPeeker's Main Program
Finally, the main program arrives. By now, it should seem like an old friend:

```c++
void main()
{
    CPictureApp *app;

    app = new CPictureApp;
    app->Run();
    app->Exit();
}
```

The Resources
Other than different icons, the only new resources are those for the About box:

```c++
resource 'ALRT' (1024, "kAlertResourceID") {
    {40, 40, 184, 276},
    1024,
    { /* array: 4 elements */
        OK, visible, sound1,
        OK, visible, sound1,
        OK, visible, sound1,
        OK, visible, sound1
    }
};
```

```c++
resource 'DITL' (1024, "kAlertResourceID") {
    {5, 12, 93, 228},
```
Summary

PICTPeeker contains three important new features:

▼ A better way to do exception handling.

▼ A working About box.

▼ The capability to undo and redo cut, copy, paste, and clear operations.

You saw that many of PICTPeeker's functions are similar to those used to create TextTyper in Chapter 17. You also learned about creating classes for dealing with exception handling and how to use those classes.
The application presented in this chapter, Tick-Tock, is a simple desk accessory-like clock. It displays the time in digital or analog fashion. Surprising as it may seem, this small program contains valuable features, such as managing idling states, creating preference files, and performing off-screen drawing. Because time waits for no one, move ahead to the program.

The Tick-Tock Program

As with previous programs in this part, Tick-Tock is built with the TCL. This program displays a working clock (in either digital or analog format) on-screen in either the foreground or background. Figure 19.1 gives you a view of time standing still in a screen dump of Tick-Tock.
Figure 19.1. A view of time standing still in Tick-Tock.

Compiling the Program

Copy the Tick-Tock folder from the source code disk. Open the Tick-Tock.n project and choose Run. All the files in the project need to be recompiled, so this takes a few moments.

Running the Program

After you have the program running, you should test it. Try the following:

- Use the Clock menu to switch back and forth from digital to analog.
- Try turning the second hand on and off.
- Resize the window by dragging the lower right corner.
- Move the Clock window, quit, and then run the program again. The window should reappear in the new location.

After you have played with the program, it is time to move on to the code.
Inspecting the Program

Figure 19.2 shows the class diagram for Tick-Tock. As you can see, there are a number of new classes.

The classes are covered in detail later in this chapter. Listed below are short descriptions to help you get started:

- ▼ CClockApp creates the Clock window.
- ▼ CQuitOnCloseWindow quits the application when the window is closed.
- ▼ CSaveRestoreSizeWindow saves its location and size at closing and restores it when opened.
- ▼ CClockPane draws the clock in either analog or digital mode.
- ▼ CDawdleBureaucratChore calls the clock pane’s Dawdle member function when the clock is not the frontmost window.

The CClockApp Class

The CClockApp class has CAboutBoxApplication as its base class. As you might expect from the last chapter, the code is pretty straightforward. The following is the class definition:

```cpp
CClockApp::CClockApp() :
    CAboutBoxApplication(kExtraMasters, kRainyDayFund,
                          kCriticalBalance, kToolboxBalance)
{
}
```

The constructor is similar to RandomRectangle's constructor in Chapter 16:

```cpp
const short kExtraMasters = 1;
const Size kRainyDayFund = 20480;
const Size kCriticalBalance = 20480;
const Size kToolboxBalance = 20480;

CClockApp::CClockApp() :
    CAboutBoxApplication(kExtraMasters, kRainyDayFund,
                          kCriticalBalance, kToolboxBalance)
{
}
```
And as in Chapter 16, the CClockApp class has one member function to start it up:

```cpp
class short kWINDStarter = 500;

void CClockApp::StartUpAction(short numPreloads)
{
    CAboutBoxApplication::StartUpAction(numPreloads);
}
```
const Boolean kFloatingWindow = TRUE;
const short kMinWindowHeight = 30;
const short kMinWindowWidth = 30;

CDirector *aDirector = new CDirector(this);

CSaveRestoreSizeWindow *aWindow =
    new CSaveRestoreSizeWindow(kWINDStarter, !kFloatingWindow, aDirector);
aDirector->itsWindow = aWindow;
Rect sizeRect = aWindow->sizeRect;
sizeRect.top = kMinWindowHeight;
sizeRect.left = kMinWindowWidth;
aWindow->SetSizeRect(&sizeRect);

aWindow->Select();
CClockPane *thePane =
    new CClockPane(aWindow, aDirector, 0, 0, 0, 0, sizELASTIC, sizELASTIC);
thePane->FitToEnclosure(TRUE, TRUE);
aDirector->itsGopher = thePane;
thePane->BecomeGopher(TRUE);

The only substantial difference between the two programs' StartUpAction functions is the call to SetSizeRect for the window in Tick-Tock. This sets the minimum size to which the user can resize the window. Besides this and a few other minor differences, the startup in this program is handled the same way as in Chapter 16.

The CQuitOnCloseWindow Class

The CQuitOnCloseWindow class provides desk accessory-like behavior to Tick-Tock. If you close the Clock window, the application quits. The class has just a constructor and a destructor:

class CQuitOnCloseWindow: public CWindow {
public:
    CQuitOnCloseWindow(short WINDid, Boolean aFloating,
        CDirector *aSupervisor);
    virtual ~CQuitOnCloseWindow();
};
The class’s constructor method is simple:

```c
CQuitOnCloseWindow::CQuitOnCloseWindow(short WINDid, Boolean aFloating, 
   CDirector *aSupervisor) :
   CWindow(WINDid, aFloating, aSupervisor)
{
   TCL_END_CONSTRUCTOR;
}
```

The destructor is not much more complex:

```c
CQuitOnCloseWindow::~CQuitOnCloseWindow()
{
   TCL_START_DESTRUCTOR;
   gApplication->AssignUrgentChore(new CQuitApplicationChore);
}
```

The destructor creates a chore to quit the application, and then tells the application to do that chore with the AssignUrgentChore function call.

**The CSaveRestoreSizeWindow Class**

The `CSaveRestoreSizeWindow` class is responsible for saving the window position and location at closing and restoring them when opening. It saves this information in a preferences file that is located in the Preferences folder in the System folder. `CSaveRestoreSizeWindow` is derived from `CQuitOnCloseWindow`:

```c
class CSaveRestoreSizeWindow: public CQuitOnCloseWindow {
public:

CSaveRestoreSizeWindow( 
   short     windID, 
   Boolean   aFloating, 
   CDirector *aSupervisor);

virtual ~CSaveRestoreSizeWindow();

protected:
   virtual void StoreInfoInFile(CDataFile *aFile);
   virtual void SetInfoFromFile(CDataFile *aFile);
   virtual void SpecifyFile(CFile *aFile);

private:
```

---

388

*Symantec C++ Programming for the Macintosh, 2nd Edition*
CDataFile *fDataFile;
};

CSaveRestoreSizeWindow includes a constructor and a destructor. Next, it presents three utility functions. Last, CSaveRestoreSizeWindow stores a CDataFile as a member, which it uses to deal with the preferences file.

The class's constructor creates the fDataFile data member and attempts to read information from the specified file:

    CSaveRestoreSizeWindow::CSaveRestoreSizeWindow(
        short windID,
        Boolean aFloating,
        CDirector *aSupervisor) :
        CQuitOnCloseWindow(windID, aFloating, aSupervisor)
    {
        fDataFile = NULL;

        fDataFile = new CDataFile;
        SpecifyFile(fDataFile);
        SetInfoFromFile(fDataFile);
        TCL_END_CONSTRUCTOR;
    }

When the window is deleted, the destructor saves the window's information to the preferences file and then removes its data member:

    CSaveRestoreSizeWindow::~CSaveRestoreSizeWindow()
    {
        TCL_START_DESTRUCTOR;
        StoreInfoInFile(fDataFile);
        TCLForgetObject(fDataFile);
    }

The member function SpecifyFile specifies the location of the preferences file:

    void CSaveRestoreSizeWindow::SpecifyFile(CFile *aFile)
    {
        short vRefNum;
        long dirID;

        FailOSErr(FindFolder(kOnSystemDisk, kPreferencesFolderType,
            kCreateFolder, &vRefNum,&dirID));
SpecifyFile calls FindFolder, a Toolbox routine. This routine returns in its parameters the volume reference number and directory ID of the Preferences folder on the system disk. If the directory does not exist, FindFolder creates it. SpecifyFile then sets aFile to refer to the Tick-Tock Preferences file in the Preferences folder.

StoreInfoInFile stores the window position and location in the preferences file as a rectangle:

```cpp
void CSaveRestoreSizeWindow::StoreInfoInFile(CDataFile *aFile)
{
    if (!aFile->ExistsOnDisk())
        aFile->CreateNew('C++5', 'PREF');

    aFile->Open(fsWrPerm);

    Rect theWindowLocSize = macPort->portRect;

    { TgrafPortsetter setPort(macPort);

        ::LocalToGlobal((Point *) &theWindowLocSize.top);
        ::LocalToGlobal((Point *) &theWindowLocSize.bottom);
    }

    TRY {
        aFile->WriteSome((Ptr) &theWindowLocSize, sizeof(theWindowLocSize));
    }
    CATCH {
        NO_PROPAGATE;       // not writing window location isn't fatal
    }
    ENDTRY;
    aFile->Close();
}
```

If the file does not yet exist, CSaveRestoreSizeWindow creates it. It then gets the local coordinates that form the bounds of the window and converts them to global coordinates. Note that it temporarily must set the port using a TGrafPortSetter because LocalToGlobal converts based on the coordinates of the current window.

The call to WriteSome is within a TRY block. Note that you need to revert to the TRY, CATCH, and ENDTRY macros for handling exceptions in this case. This is because the CATCH portion of the
exception handler is unusual—it calls NO_PROPAGATE and thereby stops the exception from propagating any further. After the exception, the code following ENDTRY is executed, and this action closes the file. If the exception had continued to propagate, it would send an alert to the user. Because this exception does not affect the functioning of the program in any substantial way, an alert is unnecessary for such a minor error.

SetInfoFromFile then reads the location and size of the window from the preferences file, if it exists:

```cpp
void CSaveRestoreSizeWindow::SetInfoFromFile(CDataFile *aFile)
{
    if (aFile->ExistsOnDisk()) {
        Rect portRect;
        aFile->Open(fsRdPerm);

        TRY {
            aFile->ReadSome((Ptr) &portRect, sizeof(portRect));
            Move(portRect.left, portRect.top);
            ChangeSize(portRect.right - portRect.left, portRect.bottom - portRect.top);
        }
        CATCH {
            NO_PROPAGATE;    // not reading window location isn't fatal
        }
        ENDTRY;
        aFile->Close();
    }
}
```

SetInfoFromFile first checks to see whether the file exists. If it finds the preferences file, it performs the following actions:

- Opens the file.
- Reads the rectangle information.
- Sets the location with Move.
- Sets the size with ChangeSize.

An exception handler is again used to prevent propagation of the error. Failing to read the stored information is not important enough to be brought to the user's attention.
The CDawdleBureaucratChore Class

The CDawdleBureaucratChore class calls the clock pane's Dawdle function, even when the clock is not the active window. The TCL's CApplication class keeps a list of idle chores whose Perform member function is called when nothing else is occurring.

NOTE

You may remember that the rectangles in Chapter 16 stopped drawing if another window or application was brought to the front. Although this might have been acceptable for the rectangles, it is not for a clock. Tick-Tock is the kind of application the user is likely to keep in a background window. If it is in the background, the user is sure to want the clock to keep ticking along.

The following is the class declaration:

class CDawdleBureaucratChore: public CChore {
public:
    CDawdleBureaucratChore(CBureaucrat *theBureaucrat);
    virtual void Perform(long *maxSleep);
private:
    CBureaucrat *fBureaucratToDawdle;
};

The class is constructed with a bureaucrat (in this program, the bureaucrat is the clock pane). CDawdleBureaucratChore stores that bureaucrat in a data member, fBureaucratToDawdle. It does so in its constructor:

CDawdleBureaucratChore::CDawdleBureaucratChore(CBureaucrat *theBureaucrat)
{
    fBureaucratToDawdle = theBureaucrat;
}

In turn, fBureaucratToDawdle is used as a data member in Perform:

void CDawdleBureaucratChore::Perform(long *maxSleep)
{
    fBureaucratToDawdle->Dawdle(maxSleep);
}
The CClockPane Class

CClockPane is the class that actually draws the clock:

class CClockPane: public CPane {
public:
    CClockPane(CView *anEnclosure, CBureaucrat *aSupervisor,
                short aWidth, short aHeight,
                short aHEncl, short aVEncl,
                SizingOption aHSizing, SizingOption aVSizing);

    virtual ~CClockPane();

    virtual void Draw(Rect *area);
    virtual void Activate(void);
    virtual void Deactivate(void);

    virtual void Dawdle(long *maxSleep);

    virtual void UpdateMenus();
    virtual void DoCommand(long theCommand);

protected:
    enum {cmdShowSeconds = 1024, cmdDigital=1025, cmdAnalog=1026};

    virtual void DrawClock(const Rect &clockSize);
    virtual void CopyFrom(const Rect &areaToCopy);
    virtual void CreateOffscreen(const Rect &area);
    virtual void UpdateOffscreen(const Rect &area);
    virtual long CalculateSleepTime();
    virtual void DrawAnalogClock(const Rect &clockSize, unsigned long now);
    virtual void DrawDigitalClock(const Rect &clockSize, unsigned long now);
    virtual void GetDigitalString(unsigned long now, Str255 s);
    virtual void GetDigitalFontAndSize(const Str255 s, const Rect &clockSize,
                                       short *font, short *size);
    virtual void GetDigitalStringLocation(const Str255 s, const Rect &clockSize,
                                           short *h, short *v);

private:
    CDawdleBureaucratChore *fIdleChore;
    Boolean fIdleChoreIsInstalled;
    BitMap fBitMap;
As you might have expected, this is a complex class that overrides a number of functions and provides some new member functions. It also stores a BitMap member and GrafPtr, which it uses for off-screen drawing.

The constructor must do a number of things:

```cpp
cClockPane::cClockPane(CView *anEnclosure, CBureaucrat *aSupervisor,
    short aWidth, short aHeight,
    short aHEnc1, short aVEnc1,
    SizingOption aHSizing, SizingOption aVSizing) :
    CPane(anEnclosure, aSupervisor, aWidth, aHeight,
        aHEnc1, aVEnc1, aHSizing, aVSizing)
{
    fIdleChore = NULL;
    fIdleChoreIsInstalled = FALSE;
    fBitMap.baseAddr = NULL;
    fGrafPtr = NULL;
    SetRect(&fBitMap.bounds, 0, 0, 0, 0);

    fIdleChore = new CDawdleBureaucratChore(this);
    fGrafPtr = (GrafPtr) ::NewPtrCanFail(sizeof(GrafPort));
    ::FailNIL(fGrafPtr);
    ::OpenPort(fGrafPtr);

    fDigital = false;
    fShowSeconds = true;
}
```

The constructor creates a CDawdleBureaucratChore, but does not yet install that chore in the application's idle list—a task reserved for when the pane becomes deactivated. It also creates and opens a GrafPort that is used for off-screen drawing.

The destructor cleans up quite a bit for CClockPane:

```cpp
cClockPane::~CClockPane()
{
    if (fIdleChoreIsInstalled)
        gApplication->CancelIdleChore(fIdleChore);
```
TCLForgetObject(fIdleChore);
if (fBitMap.baseAddr != NULL) {
    ::DisposPtr(fBitMap.baseAddr);
}
if (fGrafPtr != NULL) {
    ::ClosePort(fGrafPtr);
    ::DisposPtr((Ptr) fGrafPtr);
}
}

First, it removes the chore from the application’s idle list, if it is currently installed there. It also frees memory occupied by the off-screen BitMap member and GrafPort.

CClockPane then uses its two member functions, Activate and Deactivate, to install and deinstall CBureaucratDawdleChore. The code for these two functions is simple:

void CClockPane::Activate(void)
{
    CPane::Activate();
    if (fIdleChoreIsInstalled) {
        gApplication->CancelIdleChore(fIdleChore);
        fIdleChoreIsInstalled = false;
    }
}

void CClockPane::Deactivate(void)
{
    CPane::Deactivate();
    gApplication->AssignIdleChore(fIdleChore);
    fIdleChoreIsInstalled = true;
}

These routines are called when the pane’s window is activated or deactivated.

The UpdateMenus function handles the updating of the three menu items: Show seconds, Digital, and Analog:

void CClockPane::UpdateMenus()
{
    CPane::UpdateMenus();

    gBartender->EnableCmd(cmdShowSeconds);
    gBartender->EnableCmd(cmdDigital);
    gBartender->EnableCmd(cmdAnalog);
gBartender->CheckMarkCmd(cmdShowSeconds, fShowSeconds);
gBartender->CheckMarkCmd(cmdDigital, fDigital);
gBartender->CheckMarkCmd(cmdAnalog, !fDigital);
}

UpdateMenus checkmarks the appropriate items as well as updates them. It uses CheckMarkCmd, which takes a Boolean specifying whether to checkmark.

DoCommand actually implements the menu commands:

```cpp
void CClockPane::DoCommand(long theCommand)
{
    switch (theCommand) {
    case cmdShowSeconds:
        fShowSeconds = !fShowSeconds;
        Refresh();
        if (fShowSeconds)
            gSleepTime = 1;       // Make Dawdle start being called
        break;

    case cmdDigital:
        if (!fDigital) {
            fDigital = TRUE;
            Refresh();
        }
        break;

    case cmdAnalog:
        if (fDigital) {
            fDigital = FALSE;
            Refresh();
        }
        break;

    default:
        CPPane::DoCommand(theCommand);
        break;
    }
}
```
DoCommand sets the fShowSeconds and fDigital data members appropriately. If the data members change, the pane must be redrawn by calling the Refresh member function. This invalidates the whole pane.

The Dawdle member function performs a substantial portion of the drawing work:

```cpp
void CClockPane::Dawdle(long *maxSleep)
{
    LongRect frame;
    Rect qdFrame;

    GetFrame(&frame);
    ::LongToQDRect(&frame, &qdFrame);
    UpdateOffscreen(qdFrame);

    Prepare();
    CopyFromOffscreen(qdFrame);

    *maxSleep = CalculateSleepTime();
}
```

First, Dawdle calls UpdateOffscreen to update the off-screen BitMap member with the current clock image. It then calls CopyFromOffscreen, which copies that image back onto the screen. CalculateSleepTime then determines when Dawdle must be called next to show a new time.

Draw is needed in addition to Dawdle because the clock’s image may need refreshing due to an update event and because Dawdle may be called as rarely as once per minute:

```cpp
void CClockPane::Draw(Rect *area)
{
    UpdateOffscreen(*area);
    CopyFromOffscreen(*area);
}
```

CClockPane uses CalculateSleepTime to determine when Dawdle is going to need to be called next:

```cpp
long CClockPane::CalculateSleepTime()
{
    const short kTicksPerSecond = 60;

    if (fShowSeconds)
        return kTicksPerSecond;
    else {
        unsigned long now;
```
DateTimeRec    nowDateTime;
const short    kSecondsPerMinute = 60;

GetDateTime(&now);
Secs2Date(now, &nowDateTime);
return kTicksPerSecond * (kSecondsPerMinute - nowDateTime.second);
}

If the second hand is shown, CalculateSleepTime returns the number of ticks in a second. Otherwise, the function figures out how long it takes for the minute hand to change and returns that number of ticks.

UpdateOffscreen calls CreateOffscreen, which creates an off-screen bit map and calls DrawClock to draw into it:

```cpp
void CClockPane::UpdateOffscreen(const Rect &area)
{
    LongRect    frame;
    Rect        qdFrame;

    GetFrame(&frame);
    ::LongToQDRect(&frame, &qdFrame);
    CreateOffscreen(qdFrame);    // we would use area, if we weren't dawdling

    TGrafPortSetter setPort(fGrafPtr);
    DrawClock(qdFrame);
}
```

CreateOffscreen also allocates an off-screen bit map if the size of the current bit map is not sufficient and sets the necessary clipping and visible regions of the off-screen GrafPort:

```cpp
void CClockPane::CreateOffscreen(const Rect &area)
{
    if (area.left != fBitMap.bounds.left ||
        area.right != fBitMap.bounds.right ||
        area.top != fBitMap.bounds.top ||
        area.bottom != fBitMap.bounds.bottom) {
        fBitMap.bounds = area;
        fBitMap.rowBytes =
            ::NumberOfBytesRoundedUpToAMultipleOf2(fBitMap.bounds.right -
                                               fBitMap.bounds.left);
        if (fBitMap.baseAddr != NULL)
```
::DisposPtr(fBitMap.baseAddr);
fBitMap.baseAddr =
::NewPtrCanFail(fBitMap.rowBytes * (fBitMap.bounds.bottom -
fBitMap.bounds.top));
::FailNIL(fBitMap.baseAddr);
::RectRgn(fGrafPtr->visRgn, &area);
::RectRgn(fGrafPtr->clipRgn, &area);
fGrafPtr->portRect = area;

TGrifPortSetter setPort(fGrafPtr);
::SetPortBits(&fBitMap);
}

CopyFromOffscreen copies the off-screen bit map to the current GrafPort using the Toolbox call
CopyBits:
void CClockPane::CopyFromOffscreen(const Rect &areaToCopy)
{
   ::CopyBits(&fBitMap, &thePort->portBits,
            &areaToCopy, &areaToCopy, srcCopy, 0L);
}

DrawClock draws the clock by calling the appropriate routine to draw either a digital or an analog
version:
void CClockPane::DrawClock(const Rect &clockSize)
{
   unsigned long now;
   ::GetDateTime(&now);
   ::EraseRect(&clockSize);

   if (fDigital)
      DrawDigitalClock(clockSize, now);
   else
      DrawAnalogClock(clockSize, now);
}

As you might expect, DrawAnalogClock draws the analog version of the clock:
void CClockPane::DrawAnalogClock(const Rect &clockSize, unsigned long now)
{  
  DateTimeRec nowDateTime;
  const short kSecondHandThickness = 1;
  const short kMinuteHandThickness = 2;
  const short kHourHandThickness = 3;

  ::Secs2Date(now, &nowDateTime);

  if (fShowSeconds)
    DrawLine(clockSize, ((float) nowDateTime.second) / 60.0, kSecondHandThickness);
  DrawLine(clockSize, nowDateTime.minute / 60.0, kMinuteHandThickness);

  Rect smallFrame = clockSize; // hour hand will be inscribed here
  const kHourHandSmallerConstant = 2;  // half as big as other hands
  ::InsetRect(&smallFrame,
      (clockSize.right - clockSize.left) / (2 * kHourHandSmallerConstant),
      (clockSize.bottom - clockSize.top) / (2 * kHourHandSmallerConstant));
  DrawLine(smallFrame, nowDateTime.hour / 12.0, kHourHandThickness);
}

DrawAnalogClock has a DrawLine member function that draws a line at a given position around the clock, inscribed in a given rectangle, with a specified line width:

// fractionOfCircle: 0 means straight up, .25 means to the right

void DrawLine(const Rect &frame, long double fractionOfCircle, short size)
{
  long double angle = 2 * 3.14159 * (1-(fractionOfCircle - 0.25));
  short radiusX = (frame.right - frame.left) / 2;
  short radiusY = (frame.bottom - frame.top) / 2;

  short xCoord = ::Round(::cos(angle) * radiusX);
  short yCoord = ::Round(::sin(angle) * radiusY);

  ::PenSize(size, size);
  ::MoveTo(frame.left + radiusX, frame.top + radiusY);
  ::Line(xCoord, -yCoord);  // Macintosh uses Y decreasing
}

DrawLine figures the x and y location of the line (the hour, minute, or second hand line depending on where DrawLine is called), using fractionOfCircle to determine the angle. It then uses the Toolbox calls PenSize, MoveTo, and Line to actually do the drawing.
DrawDigitalClock, in drawing the digital version, obviously has an easier time of it than in drawing the analog version:

```cpp
void CClockPane::DrawDigitalClock(const Rect &clockSize, unsigned long now)
{
  Str255 nowStr;
  short h;
  short v;
  short font;
  short size;

  GetDigitalString(now, nowStr);
  GetDigitalFontAndSize(nowStr, clockSize, &font, &size);
  ::TextFont(font);
  ::TextSize(size);
  GetDigitalStringLocation(nowStr, clockSize, &h, &v);
  ::MoveTo(h, v);
  ::DrawString(nowStr);
}
```

DrawDigitalClock uses the utility member functions GetDigitalString, GetDigitalFontAndSize, and GetDigitalStringLocation to determine what to draw, how big it is, and where to draw it:

```cpp
void CClockPane::GetDigitalString(unsigned long now, Str255 s)
{
  ::IUTimeString(now, fShowSeconds, s);
}
```

```cpp
void CClockPane::GetDigitalFontAndSize(const Str255 s, const Rect &clockSize, short *font, short *size)
{
  *font = geneva;
  *size = 9;
}
```

```cpp
void CClockPane::GetDigitalStringLocation(const Str255 s, const Rect &clockSize, short *h, short *v)
{
  FontInfo fInfo;

  ::GetFontInfo(&fInfo);
  short excessHeight = clockSize.bottom - clockSize.top - fInfo.descent - fInfo.ascent;
```
if (excessHeight < 0)
    *v = clockSize.bottom;
else
    *v = clockSize.bottom - fInfo.descent - excessHeight / 2;

short excessWidth = clockSize.right - clockSize.left - ::StringWidth(s);
if (excessWidth < 0)
    *h = clockSize.left;
else
    *h = clockSize.left + excessWidth / 2;
}

Tick-Tock is a suave, international program, so the GetDigitalString function uses IUTimeString, an international utilities call. This way it can present the time correctly in all the countries to which it travels.

**Tick-Tock’s Main Program**

After the preceding extended tour of the CClockPane class, it is probably refreshing to finally get to the TCL main program:

```cpp
void main()
{
    CClockApp    *clockApp;

    clockApp = new CClockApp;
    clockApp->Run();
    clockApp->Exit();
}
```

**The Resources**

Other than different icons, the only new resources are those for the Clock menu itself:

```cpp
resource 'MENU' (4, "Clock", preload) {
4,
textMenuProc,
0x7FFFFFFF,
enabled,
"Clock",
{
"Show seconds"#1024", noIcon, noKey, noMark, plain,
```
Summary

The application in this chapter is a utility clock that keeps time in digital or analog fashion, with or without a second hand. Even when it is no longer the foreground window, Tick-Tock still keeps ticking.

More important than the application are the new programming skills you have acquired as a result of your further use of the TCL. You should now be able to do the following:

▼ Offer specialized menus in an application.
▼ Create and update preference files.
▼ Use the system’s idle time in a more sophisticated way.
▼ Take advantage of desk accessory-like behavior in the TCL.
▼ Handle an application that continues to run in either the foreground or background.

These new skills should really add up when you look at the next programming challenge—creating a desktop calculator.

Exercises

1. If the user saves the clock on the second screen of a two-screen system and then tries to open it on a one-screen system, the clock cannot be seen. Add code to make sure that the restored window position is on-screen and to move the clock back on-screen if the window position is off-screen.

2. Add a Font and Size menu to change the presentation of the digital clock.
3. Write a window definition procedure (WDEF) that draws a round window (warning: this is difficult).

4. Have the Copy menu item copy the current time in a text format that is suitable for pasting into a word-processing document.

5. Draw tick marks around the clock.

6. Draw hour markings around the clock.

7. The hour hand currently moves only at the top of the hour. So, at 1:59, the hour hand is on the 1, rather than almost on the 2. Make the hour hand move slowly, so that at 1:30 it is halfway between 1 and 2.

8. Use GWorlds, the preferred mechanism for off-screen drawing, for the off-screen bit map drawing.
The C++ Calc application is a simple desktop calculator with a twist. Unlike the standard Calculator desk accessory, it calculates using the Reverse Polish Notation (RPN) system you find on a Hewlett-Packard calculator.

After you review the classes that go into the creation of C++ Calc, you will know how to do the following:

- Create content classes
- Use and customize control classes

C++ Calc is a more complete application than the others in this part. The ease with which you can create a finished application in the TCL should convince you that the multiple benefits of frameworks are worth their few subtractions.
The C++ Calc Program

Figure 20.1 shows C++ Calc in action. As you can see, it displays a functional desktop calculator.

![C++ Calc calculator](image)

Figure 20.1. The C++ Calc calculator.

Compiling the Program

Copy the C++Calc folder from the source code disk. Open the C++Calc.[GP] project, and then choose Run. All the files in the project need to be recompiled, and this takes a few minutes.

NOTE

Standard calculators use infix notation, which means that the operator is between the operands. In an infix system, the sum of 3 and 4 is represented as $3 + 4$. In a calculator using the Reverse Polish Notation (RPN) system, the calculations use postfix notation, which means that the operator is after the two operands. In postfix, the sum of 3 and 4 is represented as $3 4 +$. One of the advantages of RPN is that it makes parentheses unnecessary. The infix expression $(2 + 3) \times (5 + 6)$ can be written in RPN as $2 3 + 5 6 + \times$.

Every calculator needs some way of knowing when you are finished entering a number. In infix, after you finish a number entry you must click the equal sign (=). In RPN, you must click the Enter key (Ent) to separate two numbers.

Running the Program

After you have C++ Calc running, work a few examples to check it out. You can balance your checkbook or just try the following examples:
1. Calculate $5 \times (3 + 4)$. The result should be 35. Click the following calculator buttons:

```
5
Ent
3
Ent
4
+
*
```

2. Calculate the same thing using the keyboard (notice that the calculator keys flash each time you type a character). Use the following keystrokes:

```
5
return
3
return
4
+
*
```

Select Copy from the Edit menu, and then show the Clipboard. The value from the calculator should be displayed in the Clipboard.

Type 77.

Next, clear the current number by selecting Clear, pressing the C key, or clicking the C calculator button. Then choose Paste from the Edit menu. The Clipboard contents should be pasted into the calculator.

3. Try to calculate the old-time favorite $1 + 0$. Here are the keystrokes:

```
1
Ent
0
/
```

4. Calculate $-1 + 0$. Here are the keystrokes:

```
1
±
Ent
0
/
```

5. Move the calculator window, quit, and then run the program again. The window should appear in the new location.

Now that you know the ins and outs of C++ Calc, it is time to look at the code.
Inspecting the Program

Figure 20.2 shows the class diagram for the application. As you can see, there are several new classes.

Figure 20.2. The C++ Calc class diagram.
The following are brief descriptions of the classes you have not seen before:

- **TCalculator** contains the calculator engine.
- **CCalculatorApp** creates the calculator window.
- **CCalculatorPane** stores a TCalculator object. It also contains the buttons and the numeric display. It handles keystrokes, menu items, and all communication with the TCalculator object.
- **CFloatText** creates a text field that holds a floating-point number.
- **CButtonWithFontAndSize** makes a button that displays in a given font and size.

**The TCalculator Class**

The content class of the application emulates a Hewlett-Packard calculator using Reverse Polish Notation (RPN):

```cpp
class TCalculator {
public:
    TCalculator();
    void Enter(long double num);
    void Add();
    void Subtract();
    void Multiply();
    void Divide();
    long double GetX();
    long double GetY();
    long double GetZ();

private:
    long double PopX();
    void SetX(long double num);
    void SetY(long double num);
    void SetZ(long double num);

    long double fX;
    long double fY;
    long double fZ;
};
```

TCalculator contains a three-value stack. First, Enter pushes a new value onto the stack. Then, the operator member functions (Add, Subtract, Multiply, and Divide) do the following:
Perform an operation on the top two values in the stack.

Pop those values from the stack.

Push the result onto the stack.

None of these routines is more than a few lines long and should not present any problems. Start by looking at the constructor that initializes the values to 0:

```cpp
TCalculator::TCalculator()
{
    fX = 0.0;
    fY = 0.0;
    fZ = 0.0;
}
```

The Enter routine puts the value in the x register, pushing the remaining values:

```cpp
void TCalculator::Enter(long double num)
{
    fZ = fY;
    fY = fX;
    fX = num;
}
```

The operators all pop two values on the stack, and then return the result of the operation to the top of the stack:

```cpp
void TCalculator::Add()
{
    long double v = PopX();
    Enter(v + PopX());
}

void TCalculator::Subtract()
{
    long double v = PopX();
    Enter(PopX() - v);
}

void TCalculator::Multiply()
{
    long double v = PopX();
    Enter(v * PopX());
}
void TCalculator::Divide()
{
    long double v = PopX();
    Enter(PopX() / v);
}

The setters and getters are simple:

long double TCalculator::GetX()
{
    return fX;
}
long double TCalculator::GetY()
{
    return fY;
}
long double TCalculator::GetZ()
{
    return fZ;
}

void TCalculator::SetX(long double num)
{
    fX = num;
}

void TCalculator::SetY(long double num)
{
    fY = num;
}

void TCalculator::SetZ(long double num)
{
    fZ = num;
}

The Pop routine is equally simple. It returns the x value and moves the remaining values up:

long double TCalculator::PopX()
{
    long double v = GetX();
}
fX = fY;
fY = fZ;
fZ = 0.0;
return v;
}

Notice that, even though none of the routines is very difficult, this class still contains a working calculator engine.

NOTE

If you are unclear about how the TCalculator operations function, test the class members with a simple program using cout/cin (or scanf/printf).

The CCalculatorApp Class

The CCalculatorApp class has CAboutBoxApplication as its base class. Here is the class definition:

```cpp
class CCalculatorApp: public CAboutBoxApplication
{
public:
    CCalculatorApp();
    void StartUpAction(short numPreloads);
};
```

The constructor and StartUpAction function bear remarkable similarity to those from CClockApp's in Chapter 19:

```cpp
const short kExtraMasters = 1;
const Size kRainyDayFund = 20480;
const Size kCriticalBalance = 20480;
const Size kToolboxBalance = 20480;
const short kWINDStarter = 500;

CCalculatorApp::CCalculatorApp():
    CAboutBoxApplication(kExtraMasters, kRainyDayFund,
```
As with Tick-Tock, C++ Calc needs to use CSaveRestoreSizeWindow to take advantage of desk accessory-like behavior. Note that CSaveRestoreSizeWindow is derived from CQuitOnCloseWindow and, as such, gets desk accessory behavior; that is, closing the calculator window quits the application.

The CCalculatorPane Class

The CCalculatorPane class displays a calculator:

class CCalculatorPane: public CPane {
{
public:

    CCalculatorPane(CView *anEnclosure,
                    CBureaucrat *aSupervisor,
                    short aWidth, short aHeight,


Chapter 20: C++ Calc: Building a Calculator
short aHEncl, short aVEncl,
SizingOption aHSizing, SizingOption aVSizing);

virtual void UpdateMenus();
virtual void DoCommand(long theCommand);
virtual void DoKeyDown(char theChar, Byte keyCode, EventRecord *macEvent);
virtual void Draw(Rect *r);

protected:
enum {cmdFirstCommand = 1025,
cmdAdd = cmdFirstCommand, cmdMultiply, cmdDivide, cmdSubtract, cmdEnter,
cmdPeriod, cmdE, cmdClearEntry, cmdChangeSign,
cmd0, cmd1, cmd2, cmd3,
cmd4, cmd5, cmd6, cmd7, cmd8, cmd9, cmdLastCommand = cmd9 };

virtual void UpdateDisplay();
virtual void EnterCurrentNumber();
virtual void CreateCommandButton(short rowNumber, short colNumber, long comand);

private:
TCalculator fCalculator;
CFloatText *fCalcDisplay;
unsigned char fCurrentNumber[9];
CBUTTON *fButtons[cmdLastCommand - cmdFirstCommand + 1];
};

**Member Function Descriptions**

Listed below is a short description of each member function:

- **Calculator**Pane constructs the calculator pane and creates all the calculator buttons and the floating-point display.
- **Update**Menus enables the Clear, Copy, and Paste menu items.
- **Do**Command implements the Clear, Copy, and Paste menu items. It also handles the commands cmdAdd through cmd9.
- **Do**KeyDown handles all the calculator keystrokes.
- **Draw** draws a gray background for the calculator.
- **Update**Display updates the numeric display to show the current value of the x register.
EnterCurrentNumber takes any current user-entered number and enters it into the calculator.

CreateCommandButton creates one of the calculator buttons with each call to this function.

fCalculator is the calculating engine, fCalcDisplay handles the numeric display of the calculator, and fCurrentNumber is a Pascal string that holds what the user has typed in. If the value is empty, then the numeric display shows the x register; otherwise, the numeric display shows this string. fButtons is an array of button objects, indexed by command number.

The CalculatorPane Constructor

The constructor is as follows:

```cpp
CCalculatorPane::CCalculatorPane(CView *anEnclosure, CBureaucrat *aSupervisor,
                                 short aWidth, short aHeight,
                                 short aHEnc1, short aVEnc1,
                                 SizingOption aHSizing, SizingOption aVSizing) :
CPane(anEnclosure, aSupervisor, aWidth, aHeight,
      aHEnc1, aVEnc1, aHSizing, aVSizing)
{
    const short kFloatTextWidth = 86;
    const short kFloatTextHeight = 20;
    const short kFloatTextLeftMargin = 10;
    const short kFloatTextTopMargin = 9;

    wantsClicks = true;

    fCalcDisplay = new CFloatText(this, aSupervisor,
                                 kFloatTextWidth, kFloatTextHeight,
                                 kFloatTextLeftMargin, kFloatTextTopMargin,
                                 sizFIXEDSTICKY, sizFIXEDSTICKY, -1);
    fCalcDisplay->Specify(kNotEditable, kNotSelectable, kNotStylable);
    fCalcDisplay->SetFontName("\pMonaco");
    fCalcDisplay->SetFontSize(9);
    fCalcDisplay->SetMaxDisplayWidth(sizeof(fCurrentNumber)-1);
    
    CPaneBorder *border = new CPaneBorder(kBorderFrame);
    border->SetPenSize(1, 1);
    fCalcDisplay->SetBorder(border);
}
```
CreateCommandButton(5, 2, cmd0);
CreateCommandButton(5, 3, cmdPeriod);
CreateCommandButton(5, 4, cmdEnter);

CreateCommandButton(4, 1, cmd1);
CreateCommandButton(4, 2, cmd2);
CreateCommandButton(4, 3, cmd3);
CreateCommandButton(4, 4, cmdChangeSign);

CreateCommandButton(3, 1, cmd4);
CreateCommandButton(3, 2, cmd5);
CreateCommandButton(3, 3, cmd6);
CreateCommandButton(3, 4, cmdAdd);

CreateCommandButton(2, 1, cmd7);
CreateCommandButton(2, 2, cmd8);
CreateCommandButton(2, 3, cmd9);
CreateCommandButton(2, 4, cmdSubtract);

CreateCommandButton(1, 1, cmdClearEntry);
CreateCommandButton(1, 2, cmdE);
CreateCommandButton(1, 3, cmdDivide);
CreateCommandButton(1, 4, cmdMultiply);

Length(fCurrentNumber) = 0;
TCL_END_CONSTRUCTOR;

This function calls its base constructor, and then sets its inherited data member, wantsClicks, to true. wantsClicks ensures that both this pane and subpanes accept mouse clicks.

CFloatText is created and constructed next. This is important because a calculator that calculates but doesn't show you the result is only useful to the folks at Langley. The CPaneBorder frames the text as a visual nicety. CreateCommandButton is called once for each button, with the row, column, and command number as parameters. Finally, CCalculatorPane initializes fCurrentNumber to an empty string.

The CreateCommandNumber Function

The CreateCommandNumber function creates a button at the given row and column location. It also gives each button a specific command number:
void CCalculatorPane::CreateCommandButton(short rowNumber, short colNumber, long command)
{
    CButtonWithFontAndSize *aButton;
    Str255 buttonName;
    const short kButtonWidth = 18;
    const short kButtonHeight = 16;
    const short kColumnWidth = 23;
    const short kColumnHeight = 22;
    const short kLeftMargin = 9;
    const short kTopMargin = 29;
    const short kButtonTitleStringID = 1025;

    GetIndString(buttonName, kButtonTitleStringID, command - cmdFirstCommand + 1);
    aButton = new CButtonWithFontAndSize(kButtonWidth, kButtonHeight, kLeftMargin + (colNumber-1)* kColumnHeight, kTopMargin + (rowNumber-1) * kColumnHeight, buttonName, TRUE, 0, this, this, "Geneva", 9);
    aButton->SetClickCmd(command);
    fButtons[command - cmdFirstCommand] = aButton;
}

The function gets the button title from an 'STR#' resource with the Toolbox call GetIndString. It then creates a CButtonWithFontAndSize button at the correct location with the font Geneva 9. CreateCommandButton sets the button’s click command to the command number. This click command is what the button reports to its supervisor when it is clicked. Finally, CreateCommandButton saves the button in the fButtons array.

**The UpdateMenus Function**

UpdateMenus enables the Edit menu items:

void CCalculatorPane::UpdateMenus()
{
    CPane::UpdateMenus();

    if (Length(fCurrentNumber) > 0)
        gBartender->EnableCmd(cmdClear);
    gBartender->EnableCmd(cmdCopy);
    if (gClipboard->GetData('TEXT', NULL) > 0)
        gBartender->EnableCmd(cmdPaste);
}
Note that Clear is enabled only if the user has typed in a number (for example, if fCurrentNumber is not empty). Copy is always enabled; Paste is enabled if there is text in the Clipboard.

DoCommand is the processing station for everything. When the user clicks on a button, the button sends the DoCommand message to its supervisor, the CCalculatorPane. Because it is such a large switch statement, the code is shown in pieces to make it easier to follow:

```cpp
void CCalculatorPane::DoCommand(long theCommand)
{
    switch (theCommand) {
    case cmdCopy:
        gClipboard->EmptyScrap();
        gClipboard->PutData('TEXT', fCalcDisplay->GetTextHandle());
        break;

    case cmdPaste:
        Handle h;
        Str255 s;
        gClipboard->GetData('TEXT', &h);
        if (GetHandleSize(h) > sizeof(fCurrentNumber)-1)
            SetHandleSize(h, sizeof(fCurrentNumber)-1);
        fCalcDisplay->SetTextHandle(h);
        ::DisposeHandle(h);
        fCalcDisplay->GetTextString(s);
        CopyPString(s, fCurrentNumber);
        break;

    case cmdClear:
        fButtons[cmdClearEntry - cmdFirstCommand]->_SimulateClick();
        break;
    }
}
```

Note that cmdCopy puts the current text into the Clipboard and cannot be undone (unlike PICTPeeker in Chapter 18).

cmdPaste gets the Clipboard text. It also resizes the text if it is too large to fit in the fCurrentNumber string.

```cpp
    case cmdPaste:
        Handle h;
        Str255 s;

        gClipboard->GetData('TEXT', &h);
        if (GetHandleSize(h) > sizeof(fCurrentNumber)-1)
            SetHandleSize(h, sizeof(fCurrentNumber)-1);
        fCalcDisplay->SetTextHandle(h);
        ::DisposeHandle(h);
        fCalcDisplay->GetTextString(s);
        CopyPString(s, fCurrentNumber);
        break;
```

Note that cmdPaste sets the display and the fCurrentNumber to the contents of that handle. The Paste command cannot be undone.

The Clear menu item does the same thing as clicking the clear entry (CE) button on C++ Calc. Therefore, the menu item is redirected to be a click on the button with _SimulateClick:

```cpp
    case cmdClear:
        fButtons[cmdClearEntry - cmdFirstCommand]->_SimulateClick();
        break;
```
The clear entry item clears any user-entered number by restoring the display to the x register and clearing the fCurrentNumber string:

```cpp
case cmdClearEntry:
    fCalcDisplay->SetValue(fCalculator.GetX());
    Length(fCurrentNumber) = 0;
    break;
```

Each of the operations uses EnterCurrentNumber to enter any number the user has typed. Each operation then tells the calculator to carry out the required computation:

```cpp
case cmdAdd:
    this->EnterCurrentNumber();
    fCalculator.Add();
    UpdateDisplay();
    break;

case cmdSubtract:
    this->EnterCurrentNumber();
    fCalculator.Subtract();
    UpdateDisplay();
    break;

case cmdMultiply:
    this->EnterCurrentNumber();
    fCalculator.Multiply();
    UpdateDisplay();
    break;

case cmdDivide:
    this->EnterCurrentNumber();
    fCalculator.Divide();
    UpdateDisplay();
    break;
```

UpdateDisplay ensures that the display shows the new value of the x register.

Enter takes whatever value is displayed and enters it:

```cpp
case cmdEnter:
    fCalculator.Enter(fCalcDisplay->GetValue());
    Length(fCurrentNumber) = 0;
    UpdateDisplay();
    break;
```
cmdChangeSign changes the sign of the number in the display. It does this by getting the text version, and then adding or removing a leading minus sign:

case cmdChangeSign:
    Str255 minusSign = "\p-";
    Str255 current;

    fCalcDisplay->GetTextString(current);

    if (Length(current) > 0 && current[1] == '·') {
        ::BlockMove(&current[2], &current[1], Length(current)-1);
        Length(current) = Length(current)-1;
    } else {
        ::ConcatPStrings(minusSign, current);
        ::CopyPString(minusSign, current);
    }
    fCalcDisplay->SetTextString(current);
    ::CopyPString(current, fCurrentNumber);
    break;

Each number is converted to a string (with NumToString) and appended to the fCalcDisplay (if it fits):

case cmd0:
    case cmd1:
    case cmd2:
    case cmd3:
    case cmd4:
    case cmd5:
    case cmd6:
    case cmd7:
    case cmd8:
    case cmd9:
    Str31 numberAsString;

    NumToString(theCommand - cmd0, numberAsString);
    if (Length(fCurrentNumber) + Length(numberAsString) < sizeof(fCurrentNumber)) {
        ConcatPStrings(fCurrentNumber, numberAsString);
        fCalcDisplay->SetTextString(fCurrentNumber);
    }
    break;
Scientific notation is handled the same way as appending a period to a number. In each case, if the appendage does not make the string too long, it is added to the fCurrentNumber string. The whole string is then displayed:

```cpp
case cmdPeriod:
    Str31 periodString = "\p.";
    if (Length(fCurrentNumber) + Length(periodString) < sizeof(fCurrentNumber)) {
        ConcatPStrings(fCurrentNumber, periodString);
        fCalcDisplay->SetTextString(fCurrentNumber);
    }
    break;

case cmdE:
    Str31 eString = "\pE";
    if (Length(fCurrentNumber) + Length(eString) < sizeof(fCurrentNumber)) {
        ConcatPStrings(fCurrentNumber, eString);
        fCalcDisplay->SetTextString(fCurrentNumber);
    }
    break;
```

The default case just calls the inherited version:

```cpp
default:
    CPane::DoCommand(theCommand);
}
```

### The DoKeyDown Function

DoKeyDown handles user commands in the form of keystrokes:

```cpp
void CCalculatorPane::DoKeyDown(char theChar, Byte keyCode, EventRecord *macEvent) {
    const kReturnChar = 13;
    const kEnterChar = 3;
    const kBackspaceChar = 8;
    long commandNumber;

    switch (theChar){
    case kBackspaceChar:
        if (Length(fCurrentNumber) == 1)
DoCommand(cmdClearEntry);
else if (Length(fCurrentNumber) > 1) {
    Length(fCurrentNumber) -;
    fCalcDisplay->SetTextString(fCurrentNumber);
}
return;
case '0':
case '1':
case '2':
case '3':
case '4':
case '5':
case '6':
case '7':
case '8':
case '9':
    commandNumber = cmd0 + theChar - '0';
    break;
case 'e':
case 'E':
    commandNumber = cmdE;
    break;
case 'c':
case 'C':
    commandNumber = cmdClearEntry;
    break;
case '.':
    commandNumber = cmdPeriod;
    break;
case '+':
    commandNumber = cmdAdd;
    break;
case '-':
    commandNumber = cmdChangeSign;
    break;
case '-':
    commandNumber = cmdSubtract;
    break;
case '*':
    commandNumber = cmdMultiply;
    break;
case '/':
    commandNumber = cmdDivide;
    break;

case kReturnChar:
    case kEnterChar:
    commandNumber = cmdEnter;
    break;

default:
    inherited::DoKeyDown(theChar, keyCode, macEvent);
    return;

fButtons[commandNumber - cmdFirstCommand]->SimulateClick();

The Backspace key removes the last character from the fCurrentDisplay string. If fCurrentDisplay only has one character in it, then the Backspace key, in effect, clears the entry. DoCommand is called to clear the character.

All the other keystrokes are mapped to buttons. The switch statement calculates the correct command number, and then DoKeyDown calls SimulateClick. This, in turn, makes the appropriate button act as if it were clicked. The ' and - keys are used as keyboard equivalents for ± because ± is difficult to type.

**The Draw Function**

The Draw function draws a light gray background for the buttons and display:

```cpp
void CCalculatorPane::Draw(Rect *r)
{
    if (!lgSystem.hasColorQD)
        ::FillRect(r, &qd.ltGray);
    else {
        RGBColor aColor;
        PixPatHandle myPixPat;

        aColor.red = aColor.green = aColor.blue = 0xaaaa; // light gray
        myPixPat = ::NewPixPat();
        ::MakeRGBPat(myPixPat, &aColor);
        ::FillCRect(r, myPixPat);
        ::DisposPixPat(myPixPat);
    }
}
```
To learn how to use MakeRGBPat to draw in color, see Chapter 16. Look at the discussion of DrawRectangleInRandomColor.

**The UpdateDisplay Function**

UpdateDisplay makes the display show the current x register:

```cpp
void CCalculatorPane::UpdateDisplay()
{
    fCalcDisplay->SetValue(fCalculator.GetX());
}
```

**The EnterCurrentNumber Function**

If the user has typed a number, then EnterCurrentNumber enters it into the display:

```cpp
void CCalculatorPane::EnterCurrentNumber()
{
    if (Length(fCurrentNumber) > 0) {
        fCalculator.Enter(fCalcDisplay->GetValue());
        Length(fCurrentNumber) = 0;
        UpdateDisplay();
    }
}
```

**The CFloatText Class**

The CFloatText class displays a floating-point number in a text field:

```cpp
class CFloatText: public CEditText {
public:
    CFloatText(
        CView *anEnclosure,
        CBureaucrat *aSupervisor,
        short aWidth,
        short aHeight,
        short aHEncl,
        short aVEncl,
        SizingOption aHSizing,
        SizingOption aVSizing,
        short aLineWidth);
```

---

424
Symantec C++ Programming for the Macintosh, 2nd Edition
virtual long double GetValue();
virtual void SetValue(long double value);
virtual void GetTextString(Str255 s);
virtual void SetMaxDisplayWidth(short numChars);

protected:
    virtual void ValueToText(long double value, Str255 s);

private:
    short fMaxDisplayChars;
};

Note that the class stores the number as a text string rather than as a floating-point number. Floating-point values are needed only when the user enters a number from the keyboard. The rest of the time only the text string is necessary for handling the on-screen display. Because of this, it is quicker to store numbers in text format and convert them to floating-point values as necessary.

### Member Function Descriptions

Brief descriptions of the member functions are as follows:

- **CFloatText** constructs the CFloatText object.
- **GetValue** returns the floating-point value.
- **SetValue** sets the floating-point value.
- **GetTextString** returns the text as a string.
- **SetMaxDisplayWidth** sets the maximum number of characters in the floating-point number display.
- **ValueToText** converts a floating-point value to a string representation.
- **fMaxDisplayChars**, the only data member, stores the maximum number of characters needed to display the number.

### The CFloatText Constructor

The following is the constructor:

```c++
CFloatText::CFloatText(
    CView *anEnclosure,
```
CBureaucrat *aSupervisor,
short aWidth,
short aHeight,
short aHEncl,
short aVEncl,
SizingOption aHSizing,
SizingOption aVSizing,
short aLineWidth)
CEditText(anEnclosure, aSupervisor, aWidth, aHeight, aHEncl, aVEncl,
aHSizing, aVSizing, aLineWidth)
{
  SetValue(0.0);
  SetMaxDisplayWidth(0);
  TCL_END_CONSTRUCTOR;
}

A newly created CFloatText has a value of 0 and is displayed in, at most, eight characters.

The SetMaxDisplayWidth Function

The SetMaxDisplayWidth function is a simple setter:

```c++
void CFloatText::SetMaxDisplayWidth(short numChars)
{
  fMaxDisplayChars = numChars;
}
```

The GetValue and SetValue Member Functions

The GetValue function converts the text to a floating-point value using the SANE call str2num:

```c++
long double CFloatText::GetValue()
{
  Str255 s;

  GetTextString(s);
  if (Length(s) == 0)
    return 0.0;
  else
    return ::str2num(s);
}
```
If the text is empty, GetValue returns 0.

The SetValue function does the opposite. It converts the incoming floating-point number to text and saves it:

```cpp
void CFloatText::SetValue(long double value)
{
    Str255 newString;
    Str255 existingString;

    ValueToText(value, newString);
    GetTextString(existingString);
    if (!::EqualString(newString, existingString, TRUE, TRUE))
        SetTextString(newString);
}
```

Notice the efficiency nicety: SetValue first checks to see whether the new string differs from the current string (using the Toolbox call EqualString) before saving it. This check prevents an unnecessary screen redraw for identical values.

The ValueToText routine is responsible for the actual conversion from floating-point to text numbers:

```cpp
void CFloatText::ValueToText(long double value, Str255 s)
{
    ::sprintf((char *) s, "%."Lg", fMaxDisplayChars, value);
    ::c2pstr((char *) s);
}
```

ValueToText uses the C standard library call, sprintf, to print the floating-point number value into s. The %g format determines whether to print the floating-point number in standard floating-point notation (523.67, for example) or scientific notation (5.2367e2, for example) depending on the size of the number. The .* signifies that the field width is passed as a parameter. The field width itself determines how many characters the number has when it is printed.

Because the TCL and the Toolbox work with only Pascal strings, the c2pstr call converts from a C string (with trailing zero byte) to a Pascal string (with leading length byte).

**The GetTextString Function**

Finally, GetTextString returns the text as a string:

```cpp
void CFloatText::GetTextString(Str255 s)
{
```
Handle h = GetTextHandle();
*s = Min(::GetHandleSize(h), 255);
::BlockMove(*h, s+1, *s);
}

CFloatText’s base class, CEditText, has no member function to return the text as a string—it only
returns the text as a handle. This routine gets that handle, sets the length byte of the string (up
to 255), and then uses the Toolbox call BlockMove to copy the characters from the handle to the
string.

The CButtonWithFontAndSize Class

Customarily, button titles use Chicago 12 as the default font. This class enables you to vary that
behavior; you can specify the font and size of the button titles:

class CButtonWithFontAndSize: public CButton {
public:
    CButtonWithFontAndSize(short aWidth, short aHeight,
        short aHEncl, short aVEncl,
        StringPtr title, Boolean fVisible, short procID,
        CView *anEnclosure, CBureaucrat *aSupervisor,
        StringPtr fontName, short fontSize);

    virtual void Draw(Rect *area);

private:
    short fFont;
    short fFontSize;
};

The constructor converts the passed-in font name to a font number. Next, it saves both the font
number and size:

CButtonWithFontAndSize::CButtonWithFontAndSize(short aWidth, short aHeight,
    short aHEncl, short aVEncl,
    StringPtr title, Boolean fVisible, short procID,
    CView *anEnclosure, CBureaucrat *aSupervisor, StringPtr fontName, short fontSize) :
    CButton(aWidth, aHeight, aHEncl, aVEncl, title, fVisible,
        useWFont , procID, anEnclosure, aSupervisor)
{
short fontNumber;

::GetFNum(fontName, &fontNumber);
ffont = fontNumber;
ffontSize = fontSize;
   TCL_END_CONSTRUCTOR;
}

Notice that, if you change the default behavior, you also must tell the Control Manager to get font information from the GrafPort. CNewButtonWithFontAndSize does this by adjusting the procID in useWFont, the Control Manager constant. This constant specifies that the current font and size should come from the GrafPort (rather than using Chicago 12).

The Draw routine sets the font and size before calling CButton::Draw:

void CButtonWithFontAndSize::Draw(Rect *area)
{
   ::TextFont(ffont);
   ::TextSize(ffontSize);

   CButton::Draw(area);
}

**The C++ Calc Main Program**

The following is C++ Calc's main program:

void main()
{
   CCalculatorApp *CalculatorApp;

   CalculatorApp = new CCalculatorApp;
   CalculatorApp->Run();
   CalculatorApp->Exit();
}
The Resources

There is only one new resource (other than those needed for the icons). The 'STR#' resource contains the calculator button titles. The associated command numbers of the constants determines the order of the strings in the resource:

```tcl
resource 'STR#' (1025, "button names", purgeable, preload) {
{
   "+", 
   "*", 
   "/", 
   ".", 
   "Ent", 
   ".", 
   "E", 
   "CE", 
   "±", 
   "0", 
   "1", 
   "2", 
   "3", 
   "4", 
   "5", 
   "6", 
   "7", 
   "8", 
   "9"
}
};
```

Summary

In this chapter you saw how easy it is to create a fully functional calculator using the TCL and C++. You should now be able to do the following:

- Construct content classes that are separate from the interface portion of the application. The TCalculator class gives you a working calculator engine before you have to go about designing the interface.

- Customize control classes so that you can set particular parameters and override default behaviors you do not want.
When you calculate the time saved creating this application with C++ and the TCL, you should have plenty left over for the next challenge, which is creating an address book database.

**Exercises**

1. Save the values in the calculator when it closes and restore them when it reopens. This may require creating an abstract class from `CSaveRestoreSizeWindow`. This new class should have member functions that read and write the data in the preferences file, but do not open or close it. `CSaveRestoreSizeWindow` can then override those two member functions to read and write the window location and size. After this is done, you can create a new derived class from `CSaveRestoreSizeWindow` that overrides the two member functions to read and write the x, y, and z registers.


3. Make Enter, +, -, *, and / undoable.

4. Add a display of the x, y, and z registers to C++ Calc.
Using Visual Architect: AddressBook

Separating your user interface from the core of your application can make a great deal of sense. If the central features are independent of the user interface, they can be ported from one application framework to another or even from one platform to another.

Another advantage of this separation is that it is easier to maintain the software. When an application has all its classes designed around the user interface, every change to the interface requires that you rework all the code. You may even need to completely redesign your classes. On the other hand, if you design your content classes (sometimes called the content model) based on the problem domain, you should not have to touch the classes when redesigning the interface. Only a change in the problem domain requires changing your content classes.
With this in mind, the application in this chapter has been bifurcated. The first part deals with creating the new C++ classes that make up the database engine and testing them. The effort results in the DatabaseTester program, which is an engine test bed that tests the addresses and the database of addresses. With the engine running smoothly, the second phase adds a full-featured TCL user interface (built with Visual Architect) to the underlying address and database classes. This results in the AddressBook application.

The DatabaseTester Program

What makes the DatabaseTester interesting is the C++ classes that represent the database and the C++ objects that fill it. The main program is quite simple. It just creates address book entries and adds them to a database. Before exiting, it prints out the contents of the database. Figure 21.1 shows this output.

```
press <return> to exit

(714) 555-1212
Rhodes, Fen
123 Park Avenue
Long Beach, CA 02139
(310) 798-5792
Rhodes, Alexander
1328 Clock Avenue
Redlands, CA 92374
(909) 798-5792
McKeehan, Julie
1328 Clock Avenue
Redlands, CA 92374
(909) 798-5792
La Gay, Carolyn
123 Longridge Avenue
Santa Monica, CA 92374
(818) 555-1212

Name: Rhodes, Helen Address: 1328 Clock Avenue
Name: McKeehan, Julie Address: 1328 Clock Avenue
Name: Rhodes, Alexander Address: 1328 Clock Avenue
Name: Rhodes, Nicholas Address: 1328 Clock Avenue
Name: Rhodes, Fen Address: 123 Park Avenue
Name: Rhodes, Mary Address: 1234 Main St.
Name: La Gay, Carolyn Address: 123 Longridge Avenue
```

Figure 21.1. Output from the DatabaseTester program.

Compiling DatabaseTester

Copy the DatabaseTester folder from the source code disk. Open the DatabaseTester.pi project and select Run.

Running DatabaseTester

After you have DatabaseTester running, test it by comparing the output with the main program. Following are the relevant portions of the main function for comparison purposes:
void main()
{
    TDatabase theDatabase;
    theDatabase.Insert(new TEntry("Rhodes, Neil", "1328 Clock Avenue", "Redlands", "CA", "92374", 
        "(909) 798-5792"));
    theDatabase.Insert(new TEntry("McKeehan, Julie", "1328 Clock Avenue", "Redlands", "CA", 
    "92374", "(909) 798-5792"));
    theDatabase.Insert(new TEntry("Rhodes, Alexander", "1328 Clock Avenue", "Redlands", "CA", 
        "92374", "(909) 798-5792"));
    theDatabase.Insert(new TEntry("Rhodes, Nicholas", "1328 Clock Avenue", "Redlands", "CA", 
    "92374", "(909) 798-5792"));
    theDatabase.Insert(new TEntry("Rhodes, Mary", "1234 Main St.", "Fullerton", "CA", "92635", 
        "(714) 555-1212"));
    theDatabase.Insert(new TEntry("Rhodes, Fen", "123 Park Avenue", "Long Beach", "CA", "02138", 
        "(310) 798-5792"));
    theDatabase.Insert(new TEntry("La Galy, Carolyn", "123 Longridge Avenue", "Santa Monica", "CA", 
    "92374", "(818) 555-1212"));
    PrintDatabase(theDatabase);
    FindAndPrint(theDatabase, "Rhodes, Neil");
    FindAndPrint(theDatabase, "McKeehan, Julie");
    FindAndPrint(theDatabase, "Rhodes, Alexander");
    FindAndPrint(theDatabase, "Rhodes, Nicholas");
    FindAndPrint(theDatabase, "Rhodes, Fen");
    FindAndPrint(theDatabase, "Rhodes, Mary");
    FindAndPrint(theDatabase, "La Galy, Carolyn");
}

Inspecting DatabaseTester

Now that you have perused the output, look at the application's class diagram, shown in figure 21.2. As you can see, there are very few classes.

The following are short descriptions of the classes; the details of each class are described in the following sections:

- MCollectible is an abstract class that represents objects that can be stored in the database.
- **TNameAddressPhone** is a class that holds a name, address, and phone number.
- **TEntry** is derived from **MCollectible** and **TNameAddressPhone**. One **TEntry** can compare itself with another **TEntry**.
- **TNode** is used in the implementation of **TDatabase**. It represents a node in a binary tree.
- **TDatabase** stores pointers to **MCollectibles**. It also provides a mechanism for finding a particular **MCollectible**.
- **TIterator** is a class that can cycle through the elements of a **TDatabase**.

![Class Diagram](image_url)

**Figure 21.2.** The class diagram of DatabaseTester.
The MCollectible Class

Multiple inheritance comes into play in the MCollectible class because its derived classes can be inserted into TDatabase.

IMPORTANT

Multiple inheritance is the capability to inherit the characteristics of more than one class. Recall from the discussion of multiple inheritance in Chapter 11, that a disciplined approach is needed. This approach involves using T (noun) classes to form trees combined with M (adjective) classes. Ideally, you create an M class, which can be combined with a T class at any point in the class tree. This approach provides selective functionality; each class can pick and choose from which M classes it derives and, therefore, what functionality it inherits.

A less favorable method of making a member function common to all classes is adding the member function to some T class high in the tree. This results in class bloat because more and more common functionality is added to these root classes.

The abstract class, MCollectible, defines a function that compares the values of two objects. Note that derived classes override this Compare member function.

Here is the class declaration:

class MCollectible {
public:
    enum {kItem1EqualItem2 = 0, kItem1LessThanItem2 = -1, kItem1GreaterThanItem2 = 1};
    MCollectible();
    virtual ~MCollectible();
    virtual int Compare(const MCollectible &item2) const;
};

The Compare function is declared const. You can call Compare on a const MCollectible object because it does not change the data members of the object.

The constructor and destructor do nothing:

MCollectible::MCollectible()
{
The `Compare` member function performs a three-way comparison against another item. It determines whether an object is less than, greater than, or equal to its argument and returns that information:

```cpp
int MCollectible::Compare(const MCollectible &item2) const
{
    if (this < &item2)
        return kItem1LessThanItem2;
    else if (this > &item2)
        return kItem1GreaterThanItem2;
    else
        return kItem1EqualItem2;
}
```

This version of `Compare` compares two locations in memory. `item1` is less than `item2` if its memory location is less than `item2`'s memory location.

Later on, you see that the database class calls the `Compare` function to find where to insert an `MCollectible` in the binary tree.

**The `TNameAddressPhone` Class**

`TNameAddressPhone` holds name, address, and phone number information:

```cpp
class TNameAddressPhone {
public:
    TNameAddressPhone(const char *name = NULL, const char *address = NULL,
                      const char *fCity = NULL, const char *fState = NULL,
                      const char *fZip = NULL, const char *fTelephone = NULL);
    virtual ~TNameAddressPhone();

    const char *GetName() const;
    const char *GetAddress() const;
    const char *GetCity() const;
    const char *GetState() const;
    const char *GetZip() const;
    const char *GetTelephone() const;
};
```
private:
    char *fName;
    char *fAddress;
    char *fCity;
    char *fState;
    char *fZip;
    char *fTelephone;
};

TNameAddressPhone provides a constructor, a destructor, and a number of getters (GetName, GetAddress, GetCity, GetState, GetZip, and GetTelephone).

The constructor code is as follows:

TNameAddressPhone::TNameAddressPhone(const char *name, const char *address,
    const char *city, const char *state, const char *zip, const char *telephone)
{
    fName = fAddress = fCity = fState = fZip = fTelephone = NULL;
    fName = CopyOfString(name);
    fAddress = CopyOfString(address);
    fCity = CopyOfString(city);
    fState = CopyOfString(state);
    fZip = CopyOfString(zip);
    fTelephone = CopyOfString(telephone);
}

Note that the constructor initializes each of the data members using a utility routine,
CopyOfString:

static char *CopyOfString(const char *s)
{
    if (s == NULL)
        s = " ";
    char *copy;

    copy = new char[::strlen(s)+1];
    ::strcpy(copy, s);
    return copy;
}

All those newly created chunks of memory for each data member also must be destroyed
eventually. This, of course, is the destructor's job:
TNameEntryPhone::~TNameEntryPhone()
{
    delete [] fName;
    delete [] fAddress;
    delete [] fCity;
    delete [] fState;
    delete [] fZip;
    delete [] fTelephone;
}

Notice that the destructor uses delete [] rather than just delete. This is necessary to match the allocation, which used new[].

Following are all the getters (accessors) previously mentioned—all perfect one-liners:

const char *TNameAddressPhone::GetName() const
{
    return fName;
}

const char *TNameAddressPhone::GetAddress() const
{
    return fAddress;
}

const char *TNameAddressPhone::GetCity() const
{
    return fCity;
}

const char *TNameAddressPhone::GetState() const
{
    return fState;
}

const char *TNameAddressPhone::GetZip() const
{
    return fZip;
}

const char *TNameAddressPhone::GetTelephone() const
{
    return fTelephone;
}
Each of these functions returns a const char *, which is a pointer to unmodifiable characters. Because the functions do not modify any data members of TNameAddressPhone, they are declared const.

The TEntry Class

TNameAddressPhone objects cannot be inserted into the database because they are not derived from the MCollectible class. The way to rectify the situation is to use multiple inheritance in the design of the next class. Therefore, you create a new class that derives from both TNameAddressPhone (so that it can hold names, addresses, and phone numbers) and MCollectible (so that it can be inserted in a TDatabase). This new class is TEntry:

class TEntry: public MCollectible, public TNameAddressPhone {
public:
    TEntry(const char *name = NULL, const char *address = NULL,
           const char *fCity = NULL, const char *fState = NULL,
           const char *fZip = NULL, const char *fTelephone = NULL);

    virtual int Compare(const MCollectible &item2) const;
};

TEntry's Compare member function overrides MCollectible::Compare and compares fName data members:

int TEntry::Compare(const MCollectible &item2) const
{
    int compareResult = ::strcmp(GetName(), ((TEntry & ) item2).GetName());

    if (compareResult < 0)
        return kItem1LessThanItem2;
    else if (compareResult > 0)
        return kItem1GreaterThanItem2;
    else
        return kItem1EqualItem2;
}

Note that only the fName data member is compared at this point. This is because the name is used as the key to the rest of the information. When inserting a TEntry, TDatabase calls the Compare routine to find the proper location. Compare also is used to find a TEntry.

The TNode Class

The TNode class represents one node in a binary tree:
class TNode {
public:
    TNode(MCollectible *value, TNode *up = NULL);
    ~TNode();

    void Insert(MCollectible *value);
    MCollectible *Find(const MCollectible &key) const;

private:
    const TNode *LeftMost() const;
    const TNode *RightMost() const;

    MCollectible *GetValue() const;

    MCollectible *fValue;
    TNode *fLeft;
    TNode *fRight;
    TNode *fUp;
    friend class TIterator;
};

The nodes are hooked together in a binary tree, as shown in figure 21.3.

Figure 21.3. TNodes in a tree.
Besides its constructor and destructor, TNode has a few other functions. One function, Insert, handles the insertion of an MCollectible into the tree starting at this node. Another function, Find, attempts to locate a matching MCollectible in the tree rooted at this node. The other two member functions, LeftMost and RightMost, return the left- and rightmost subnode.

Pay special attention to the friend designation of TIterator in TNode:

friend class TIterator;

As a friend, TIterator has access to the private members of TNode. Because TIterator is tied tightly to TNode, it needs more access than a normal class does.

The following is the actual code for the TNode functions, starting with the LeftMost and RightMost routines. Notice that LeftMost chases the fLeft data member and that RightMost chases fRight.

const TNode *TNode::LeftMost() const {
    const TNode *n = this;
    while (n->fLeft != NULL)
        n = n->fLeft;
    return n;
}

const TNode *TNode::RightMost() const {
    const TNode *n = this;
    while (n->fRight != NULL)
        n = n->fRight;
    return n;
}

GetValue is a simple getter function that returns the current value in fValue:

MCollectible *TNode::GetValue() const {
    return fValue;
}
Find finds a matching MCollectible in this node or a subnode:

```cpp
MCollectible *TNode::Find(const MCollectible&key) const
{
    const TNode *node = this;

    while (node != NULL) {
        int compareResult = node->GetValue()->Compare(key);

        if (compareResult == MCollectible::kItem1EqualItem2)
            return node->GetValue();
        else if (compareResult <= MCollectible::kItem1LessThanItem2)
            node = node->fRight;
        else
            node = node->fLeft;
    }
    return NULL;
}
```

Note that Find takes advantage of the fact that the nodes are sorted. For a given node, all nodes in the left subtree are less than it and all nodes in the right subtree are greater than it (see fig. 21.4).

![Diagram of node structure](image)

**Figure 21.4.** The sorting of nodes.
Notice the use of the class qualifier (MCollectible::) before the enumeration constants kItem1EqualItem2 and kItem1LessThanItem2. This is done because you must specify the originating class when using an enumeration constant that is defined in another class. Each new class defines a new scope.

Next, take a look at the Insert routine in TNode:

```cpp
void TNode::Insert(MCollectible *value)
{
    TNode **node = &this;  // after loop: holds address within previous node
    // which points to current node
    TNode *up = NULL;       // after loop: pointer to previous node

    do {
        int compareResult = (*node)->GetValue()->Compare(*value);

        up = *node;    // points at the current node

        if (compareResult == MCollectible::kItem1EqualItem2)
        {
            delete value;
            return;
        }
        else if (compareResult <= MCollectible::kItem1LessThanItem2)
        {
            node = &(*node)->fRight;    // advance to next node
        }
        else
        {
            node = &(*node)->fLeft;     // advance to next node
        }
    } while (*node != NULL);

    *node = new TNode(value, up);
}
```

After the loop is finished, the node variable is a pointer to either the fLeft or fRight data member, depending on where the new node needs to be placed. If an identical entry is already in the node, Insert deletes the value it was going to insert and then returns.

**The TDatabase Class**

TDatabase is the class that holds the database of MCollectible objects. These objects can be inserted or searched.
Each TDatabase tree looks different, depending on the order in which MCollectible objects were inserted. For example, when using the names in DatabaseTester's main program and the same order of entry, you get the TNode tree shown in figure 21.5.

![Figure 21.5. The tree of names from the main program.](image)

The order in which you insert nodes can greatly affect the resulting tree. Can you imagine a more balanced tree holding these particular objects? (If so, see exercise 2 at the end of this chapter.)

The class declaration for TDatabase is as follows:

```cpp
class TDatabase {
public:
    TDatabase();
    virtual ~TDatabase();
};
```
void Insert(MCollectible* dataElement);
MCollectible *Find(const MCollectible& key) const;

private:
    TNode *fRoot;
    friend class TIterator;
};

The constructor and destructor are simple:

TDatabase::TDatabase()
{
    fRoot = NULL;
}

TDatabase::~TDatabase()
{
    delete fRoot;
}

The Find routine just delegates the chore to the root node, if there is one:

MCollectible *TDatabase::Find(const MCollectible& key) const
{
    if (fRoot == NULL)
        return NULL;
    else
        return fRoot->Find(key);
}

Similarly, the Insert routine delegates the insertion to the root node:

void TDatabase::Insert(MCollectible* dataElement)
{
    if (fRoot == NULL)
        fRoot = new TNode(dataElement);
If there is no root node, it creates a new node and makes the new node the root node.

**The TIterator Class**

The TIterator class iterates over objects within a TDatabase. Because it is common, data structures like TDatabase should provide iteration capabilities to access each element of a data structure. This class also guarantees that the objects are visited in sorted order (sorting is imposed by the Compare function for the objects in the database).

There is an advantage to placing the iteration in a separate class rather than providing iteration member functions directly in TDatabase. With this method, you can have multiple iterators iterating over a TDatabase simultaneously, each at its own pace.

The declaration for TIterator is as follows:

```cpp
class TIterator {
public:
    TIterator(const TDatabase &database);
    void MoveForward();
    void MoveBackward();
    int NoMore();
    int AtEnd();
    int AtBeginning();
    void MoveToBeginning();
    void MoveToEnd();
    MCollectible *GetCurrent();
protected:
    TNode *GetRoot();
private:
    const TDatabase &fDatabase;
    const TNode *fCurrent;
};
```

TIterator is initialized with the database that it iterates. For example, you use a TIterator to iterate in sorted order using the following code:
TIteator iter(aDatabase);
for (iter.MoveToBeginning(); !iter.NoMore(); iter.MoveForward()) {
    MCollectible *currentObject = iter.GetCurrent();
    ...
}

Similarly, you can iterate in reverse order:

TIteator iter(aDatabase);
for (iter.MoveToEnd(); !iter.NoMore(); iter.MoveBackward()) {
    MCollectible *currentObject = iter.GetCurrent();
    ...
}

The fDatabase data member contains the database over which the iterator iterates. The fCurrent data member points to the node where the iterator is currently processing.

The GetRoot and GetCurrent member functions are simple:

TNode *TIterator::GetRoot()
{
    return fDatabase.fRoot;
}

MCollectible *TIterator::GetCurrent()
{
    if (fCurrent == NULL)
        return NULL;
    else
        return fCurrent->GetValue();
}

The NoMore function returns true at the end of the iteration and it returns false if there are remaining elements to iterate over. The function uses the fact that fCurrent is NULL when reaching the end of iteration:

int TIterator::NoMore()
{
    return fCurrent == NULL;
}
The first element in sorted order is the leftmost node of the root. AtBeginning returns true if the iterator is currently at the first (leftmost) node.

Similarly, the last element in sorted order is the rightmost node of the root. AtEnd returns true if the iterator is currently at the last (rightmost) node:

```cpp
tint TIterator::AtBeginning()
{
    return fCurrent != NULL && GetRoot()->LeftMost() == fCurrent;
}

tint TIterator::AtEnd()
{
    return fCurrent != NULL && GetRoot()->RightMost() == fCurrent;
}
```

MoveToBeginning moves the iterator to the first node in the tree:

```cpp
void TIterator::MoveToBeginning()
{
    if (GetRoot() == NULL)
        fCurrent = NULL;
    else
        fCurrent = GetRoot()->LeftMost();
}
```

MoveToEnd moves the iterator to the last node in the tree:

```cpp
void TIterator::MoveToEnd()
{
    if (GetRoot() == NULL)
        fCurrent = NULL;
    else
        fCurrent = GetRoot()->RightMost();
}
```

MoveForward just moves to the next node in the tree:

```cpp
void TIterator::MoveForward()
{
if (fCurrent == NULL)
    MoveToBeginning();
// if we have a right-hand side, the next node is the left-most of our RHS
else if (fCurrent->fRight != NULL)
    fCurrent = fCurrent->fRight->LeftMost();
else if (fCurrent->fUp == NULL)  // if we don't have a RHS, and don't have an up,
    // we're done
    fCurrent = NULL;
else {
    // the next is our first ancestor who we are on the left-hand-side of
    const TNode *last;
    do {
        last = fCurrent;
        fCurrent = last->fUp;
    } while (fCurrent != NULL && fCurrent->fRight == last);
}

MoveBackward, as you might expect, moves to the previous node in the tree:

void TIterator::MoveBackward()
{
    if (fCurrent == NULL)
        MoveToEnd();
    // if we have a left-hand side, the next node is the right-most of our LHS
    else if (fCurrent->fLeft != NULL)
        fCurrent = fCurrent->fLeft->RightMost();
    else if (fCurrent->fUp == NULL)  // if we don't have a LHS, and don't have an up,
        // we're done
        fCurrent = NULL;
    else {
        // the next is our first ancestor who we are on the right-hand-side of
        const TNode *last;
        do {
            last = fCurrent;
            fCurrent = last->fUp;
        } while (fCurrent != NULL && fCurrent->fLeft == last);
    }
}
If you find yourself getting lost in these last two functions, you may want to draw a small binary tree and convince yourself that the algorithms find the desired node correctly.

The DatabaseTester Main Program

This main program has more in it than most programs. First, it inserts some entries into a database. Next, it iterates over the database in reverse order, printing out each entry. Last, it finds entries within the database and prints them out:

```cpp
void main()
{
    TDatabase theDatabase;

    theDatabase.Insert(new TEntry("Rhodes, Neil", "1328 Clock Avenue", "Redlands", "CA", "92374", 
        "(909) 798-5792");
    theDatabase.Insert(new TEntry("McKeehan, Julie", "1328 Clock Avenue", "Redlands", "CA", 
        "92374", "(909) 798-5792");
    theDatabase.Insert(new TEntry("Rhodes, Alexander", "1328 Clock Avenue", "Redlands", "CA", "92374", 
        "(909) 798-5792");
    theDatabase.Insert(new TEntry("Rhodes, Nicholas", "1328 Clock Avenue", "Redlands", "CA", "92374", 
        "(909) 798-5792");
    theDatabase.Insert(new TEntry("Rhodes, Mary", "1234 Main St.", "Fullerton", "CA", "92635", 
        "(714) 555-1212");
    theDatabase.Insert(new TEntry("Rhodes, Fen", "123 Park Avenue", "Long Beach", "CA", 
        "02138", "(310) 798-5792");
    theDatabase.Insert(new TEntry("La Galy, Carolyn", "123 Longridge Avenue", "Santa Monica", "CA", "92374", 
        "(818) 555-1212");
    PrintDatabase(theDatabase);

    FindAndPrint(theDatabase, "Rhodes, Neil");
    FindAndPrint(theDatabase, "McKeehan, Julie");
    FindAndPrint(theDatabase, "Rhodes, Alexander");
    FindAndPrint(theDatabase, "Rhodes, Nicholas");
    FindAndPrint(theDatabase, "Rhodes, Fen");
    FindAndPrint(theDatabase, "Rhodes, Mary");
    FindAndPrint(theDatabase, "La Galy, Carolyn");
}
```

The following are the two functions that the main program uses, `FindAndPrint` and `PrintDatabase`:

```cpp
void FindAndPrint(const TDatabase &theDatabase, char *name)
{
    TEntry *entry = (TEntry *) theDatabase.Find(TEntry(name));
    if (entry != NULL)
```
cout << "Name: " << entry->GetName() << " Address: " << entry->GetAddress() << '
';
}

void PrintDatabase(const TDatabase &theDatabase)
{
    printf("Database: %n:
"),
    TIterator iter(theDatabase);
    for (iter.MoveToEnd(); !iter.NoMore(); iter.MoveBackward()) {
        TEntry *entry = (TEntry *) iter.GetCurrent();
        if (entry)
            cout << entry->GetName() << '
' <<
                entry->GetAddress() << '
' <<
                entry->GetCity() << ", " << entry->GetState() << " " << entry->GetZip() << '
' <<
                entry->GetTelephone() << '
';
    }
}

With these new C++ classes and main program you can see how easy it is to test the database
and entry classes without first having to create a full-blown user interface. At this point, if you
want a limited form of user interaction, you can modify the main program to read input from
 cin. After you assure yourself that the engine is purring as smoothly as a kitten, it is time to turn
your attention to the database interface.

The AddressBook Program

AddressBook is larger than any of the other TCL programs presented so far. With it, you can do
any of the following:

▼ Enter names, addresses, and telephone numbers
▼ Save the names and restore them
▼ Browse through the entries one at a time
▼ Flip to the beginning or end of the database

Figure 21.6 shows what AddressBook's main entry screen looks like. You also can see the main
address database positioned behind it.
Compiling AddressBook
Copy the AddressBook 2 folder from the source code disk. Open the AddressBookFull.pi project, and then choose Run. All the files in the project need a few minutes to recompile.

Running AddressBook
After the program is up and running, test it out by trying the following:
- Use the New Entry menu item to create some new names and addresses.
- Save your new address book in a file.
- Cycle through the address book forward and backward. Flip to the front and back of the book.
- Quit the application, and then reopen your saved file.

The Classes of AddressBook
The following are short descriptions of the new classes:
- CAddressBookApp creates the documents and sets up parameters for the Open dialog.
- x_CAddressBookApp is the class generated by the Visual Architect as a superclass for CAddressBookApp.
- CAddressBookDoc holds the addresses, saves them to disk, and reads them from disk. It also creates the windows and panes to display them.
x_CAddressBookDoc is the class generated by the Visual Architect as a superclass for CAddressBookDoc.

CAddressBookPane keeps track of which address is currently on display. It also handles the different database view requests and handles the New Entry menu command.

x_CAddressBookPane is the class generated by the Visual Architect as a superclass for CAddressBookPane.

CNewEntryDialog ensures that the New Entry dialog has the correct font and styles.

x_CNewEntryDialog is the class generated by the Visual Architect as a superclass for CNewEntryDialog.

Using Visual Architect

In all the preceding TCL projects, you have ignored the Visual Architect file that was created as part of the project and built the user-interface procedurally. For AddressBook, you will use Visual Architect to build the main window and entry dialog.

The AddressBook_7t_ project already has a complete Visual Architect resource file, so create a new project folder in the THINK Project Manager that is a Visual Architect Project.

Launching Visual Architect

One of the application segments created from the Visual Architect project template is the Visual Architect segment (see fig. 21.7). It contains two files, Visual Architect.rsrc and READ ME.cp. You can remove the readme file. To launch Visual Architect, double-click on this file.

Visual Architect launches and opens the files (see fig. 21.8).

The main file window (the one that shows the name of the file that you are editing) lists all the views in the file. A view is a collection of user-interface objects, such as buttons, editable text, icons, and so forth.

If you double-click on the main view entry, the view is displayed in a window for editing (see fig. 21.9).

This view has two items in it: the pictures of the circles and a static text item, “hello, world!”.

You will not be using either of these in the main window, so select All from the Edit menu and choose Clear to clear the items. A gray outline should remain. This shows the size that is set in the View Info dialog box.
From the View menu, choose View Info. You need to edit the information in the View Info dialog box to make it match figure 21.10. To do this, follow these steps:
Figure 21.10. The Main document View Info dialog box.

1. Type AddressBook into the name field. This sets the name of the class that will be derived from CWindow.

2. Type 3 in the Title field. New windows will be created based on this title.

3. In the procID field, set the window type to 4 or just choose the second small window icon in the row. This is a window with no size box, zoom box, or scroll bars.

4. Set the size to 338 for the width and 82 for the height.

5. Select OK to save the new view information.

Adding Items to a View

You now have a view that is completely usable in a TCL application but which doesn’t do anything. This view doesn’t have any user-interface elements inside of it, and there is no source code supporting it.

Visual Architect has a tool palette in the Tools menu. It is removable, so drag a copy of the palette off the menu bar and place it in a convenient location. The tools on the palette are shown in figure 21.11.

Figure 21.11. The Tools palette.
Each of the tools (other than the arrow tool) creates a specific kind of interface element, and each element is based on a class from the TCL. A static text item, for example, is based on CStaticText.

In AddressBook's main window, both the text labels and the text items themselves are static text items. There are 12 items altogether: the six labels and the six display items.

Click on the static text tool in the tools palette and click in the AddressBook window for the name label. Type Name: and press Enter. Don't press Return because static text items can be more than one line long so Return is a valid character.

Click on the static text tool and create the Name field. Enter the text Name goes here. This text will be removed later; if there is no text, Visual Architect resizes the item down to a very small object that is hard to work with. Your window should look something like figure 21.12.

![Figure 21.12. The main window.](image)

Now do the same for the remaining fields (Address, City, State, Zip, and Telephone), first creating the label, and then the field itself. When you are finished, your window should look something like figure 21.13.

![Figure 21.13. The main window.](image)

When you ran the program, the labels and fields didn't use the Chicago font, but rather used Geneva. From the Edit menu, Select All. Then from the Pane menu, choose Font and Geneva. Set the size of the items to 9 point by selecting Size from the Pane menu. To make the labels bold, select them; then from the Pane menu, choose Style and Bold. Now your window should resemble the one shown in figure 21.14.
If the positions of your labels and fields don’t satisfy you, the following list shows the four coordinates for each of the labels and fields. The order is left, top, width, and height.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Label Position</th>
<th>Field Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>5, 3, 34, 12</td>
<td>70, 3, 250, 12</td>
</tr>
<tr>
<td>Address</td>
<td>5, 23, 55, 12</td>
<td>70, 23, 250, 12</td>
</tr>
<tr>
<td>City</td>
<td>5, 43, 40, 12</td>
<td>70, 43, 80, 12</td>
</tr>
<tr>
<td>State</td>
<td>155, 43, 35, 12</td>
<td>195, 43, 20, 12</td>
</tr>
<tr>
<td>Zip</td>
<td>220, 43, 24, 12</td>
<td>245, 43, 75, 12</td>
</tr>
<tr>
<td>Telephone</td>
<td>5, 63, 60, 12</td>
<td>70, 63, 200, 12</td>
</tr>
</tbody>
</table>

To change any of the characteristics of a view element, double-click on it. For example, you can double-click on the Name field (not the label). The resulting dialog box is shown in figure 21.15.

To set the data for and get the data from the fields, these static text items must have identifiers. By default, Visual Architect assigns an identifier to any element. You can see that the Name field has the default name Stat9. The code that is already written expects that the identifiers for the fields will be Name, Address, City, State, Zip, and Telephone. You can either set these as you go, resizing them, or you can use the Pane menu Identifier command (see fig. 21.16), which displays a dialog box that you use for setting only the identifier (see fig. 21.17).
One last thing you need to do is remove the placeholder text that you created in the fields (Name goes here and so forth). Again, you can do this as you go along by expanding the dialog as shown in figure 21.18. To do this, click the arrow to the left of the CEditText label. Remove the text from the hText field.

Figure 21.16. The Identifier command.

Figure 21.17. The Identifier dialog.

Figure 21.18. The Expanded Element Info dialog box.
After you have done this for the six display fields, your dialog box will look like the one in figure 21.19 when you select all.

![Dialog box](image)

**Figure 21.19.** The final main window.

Now that the first part of the visual interface of AddressBook is done, you need to define the application and document classes; you will also define a panorama-based class to control the behavior of the main window. After that, you will create the menu bar, new menu items, and assign commands for the menu items.

### Creating Classes

Choose Classes from the Edit menu. The two classes come from the default Visual Architect project template (see fig. 21.20). `CApp` is the application class (a subclass of `CApplication`) and `CMain` is the document class (a subclass of `CDocument`). Click on `CApp` and type `CAddressBookApp` as the new class name. For `CMain`, type `CAddressBookDoc`. Press Return to create a new class and type `CAddressBookPane`. To set the base class for `CAddressBookPane`, choose `CPanorama` from the Base Class pop-up menu (see fig. 21.21).

![Class selection](image)

**Figure 21.20.** Creating Classes.

Now that you have a class (`CAddressBookPane`) for the main window, click in the main AddressBook window, but not on any of the items in the window. You may need to open it from the main file window. From the Pane menu, choose `CAddressBookPane` from the Class hierarchical menu.
Creating New Commands

Unless your application has no custom behavior, you will be creating new commands. When you create commands in Visual Architect, you make it easy to assign behavior to user interface elements, such as menus and buttons, that generate commands and do action.

From the Edit menu, choose Commands. Here you see all the commands that are supplied and supported by the TCL. At this point you will want to add commands for the four entries that you will be adding to the Entries menu in the next section.

Press Return to enter a new command. Type cmdGoForward. In the Actions part of the dialog box, choose CAddressBookPane from the In Class pop-up menu (see fig. 21.22). From the Do pop-up menu, choose Call.

Repeat this process, creating cmdGoBackward, cmdGoToBeginning, and cmdGoToEnd. Create one last command, cmdNewEntry, but don't choose a class or other action for this command. You will add it later.
Editing the Menus and Menu Bar

Now that you have all the interface items, the last step is to create a new menu and menu items that will call the four new commands.

From the Edit menu, choose Menu Bar. Press Return to add a new menu to the menu bar. Type Entries and click on the Edit Menu Items button to create the items for this menu (see fig. 21.23).

Type Go Forward for the menu item name. Enter F in the Cmd-key field and, from the Command pop-up menu, choose cmdGoForward. Create an entry for Go Backward, using the command key B, and the command cmdGoBackward. To create a separate line in the menu, create an entry whose text is just a hyphen (-).
Add two more items: Go to Beginning (the command is `cmdGoToBeginning`) and Go to End (the command is `cmdGoToEnd`). Insert another separator line and add the last menu item, Add Entry (the command key is E; the command is `cmdNewEntry`).

Choose OK to close the Menu Items dialog box. Choose OK in the Menu Bar dialog box to close it.

**Setting the Application Information**

From the Edit menu, choose Application to set the copyright information, signature for the application, and document types that the application can open. Fill in the dialog box as shown in figure 21.24.

![Figure 21.24. Setting the application information.](image)

**Generating Code**

For this part of the AddressBook application, you have all the visual interface elements that you need: a window with static text labels and fields, a new menu, and menu items that will generate commands.

Visual Architect can generate .h and .cp files for all the new classes (application, document, and pane) and insert the command handlers in the right places. The files are generated from template files that come with Visual Architect. For more information on customizing these files to add new functionality, see the THINK reference manuals.

Save your file and choose Generate All from the THINK Project Manager menu (see figure 21.25). Visual Architect will generate a set of files and store them a Source folder in the same folder as the Visual Architect.rs file. If you are running System 7, Visual Architect will add these new files to the current project in the THINK Project Manager (see figs. 21.26 and 21.27).
Figure 21.25. Choosing Generate All.

Figure 21.26. Generating new files.

Figure 21.27. Updating the project.

If you go back to the project in the THINK Project Manager, you should see the new files at the bottom of the project window in the Application class. You can remove the PLACEHOLDER.cpp file from this project now.

In the Source folder on your hard disk, the following files will have been created:

- AddressBookItems.h
- AppCommands.h
- CAddressBookApp.cp
- CAddressBookApp.h
- CAddressBookDoc.cp
- CAddressBookDoc.h
- CAddressBookPane.cp
- CAddressBookPane.h
- CSaver_class.cpp
- ItsContentsClass.h
- main.cp
Some of these files are read only, meaning that you don’t modify them; Visual Architect generates them anew each time you modify the visual interface of your application. The most obvious regenerated files are those with the x_ prefix. Before you add code to a file, check the header and make sure that it doesn’t warn against modifying the file.

You can proceed by building the project now. You will create an application with a main window (the database display window), an Entries menu with all its items dimmed, and the capability to Save As a document on your hard drive and open it up again. The application also includes an About box.

**Modifying the Generated Source Code**

The program code generated by Visual Architect does not have any custom behavior, therefore it does not work just yet: no records can be created, you cannot navigate from record to record, and so forth. In this section, you add new code and other files, such as the database files, to the project. At the end of this section, the program will not be complete (you will not be able to add new records), but it will have application functionality and it will be able to work with documents.

**Adding Database Functionality**

In the first part of this chapter you learned about the Database class. Add the files Database.cp and Entry.cp to the project. Remember to copy them and their header files to the source folder before adding them to the project.

**The AddressBook Main Program**

Along with the read-only files generated by the Visual Architect, you will not modify main.cp.
Here is that main function:

```cpp
void main()
{
    CAddressBookApp *application = new CAddressBookApp;
    application->ICAddressBookApp();
    application->Run();
    application->Exit();
}
```

### The `CAddressBookApp` Class

The `CAddressBookApp` class is defined by Visual Architect:

```cpp
class CAddressBookApp : public x_CAddressBookApp
{
public:

    TCL_DECLARE_CLASS

    void ICAddressBookApp(void);

    virtual void ForceClassReferences(void);
    virtual void DoCommand(long theCommand);

    // Remove this function if you do not have
    // multiple document types

    virtual OSType GetDocTypeFromDialog(CDialogDirector *dialog_box);
};
```

You may notice that the `CAddressBookApp` class inherits from `x_CAddressBookApp` rather than `CApplication` as you have seen before. Visual Architect puts default behavior in the `x_` classes and lets you customize the behavior in the derived classes.

The class declaration for the `x_CAddressBookApp` class follows:

```cpp
class x_CAddressBookApp : public CApplication
{
public:
```
TCL_DECLARE_CLASS

    void        Ix_CAddressBookApp(short extraMasters, Size aRainyDayFund,
            Size aCriticalBalance, Size aToolboxBalance);
    virtual void SetUpFileParameters(void);
    virtual void SetUpMenus(void);
    virtual void ForceClassReferences(void);
    virtual void DoCommand(long theCommand);
    virtual void CreateDocument(void);
    virtual void OpenDocument(SFReply *macSFReply);
    
    virtual Boolean    FileAlreadyOpen(CFile *aFile);
};

Any custom application functionality will be added to the CAddressBookApp class; remember, don’t modify the x_ classes or their files. Visual Architect may regenerate any of the x_ files every time you change some aspect of the visual interface.

You don’t need to modify any of the generated code! Visual Architect has created a completely functioning application class. Here are the functions that were created. Some of the instruction comments have been deleted.

Rather than using constructors alone, Visual Architect creates initialization functions for classes. The following is for ICAddressBookApp. There is no application specific data, so you don’t need to add any new code after the comment:

    void        CAddressBookApp::ICAddressBookApp()
    {
        Ix_CAddressBookApp(4, 24000L, 20480L, 2048L);

        // Initialize your own application data here.
    }

This calls the superclass initialization function, Ix_CAddressBookApp, which just calls IApplication:

    void        x_CAddressBookApp::Ix_CAddressBookApp(short extraMasters, Size aRainyDayFund,
            Size aCriticalBalance, Size aToolboxBalance)
You can begin to see how you can customize the behavior of your application in the derived class, CAddressBookApp.

If you have any custom classes for which you have no direct references, the linker could strip out the code. ForceClassReferences gives you a place to put a reference to a class and guarantee that it will not be stripped:

```cpp
void CAddressBookApp::ForceClassReferences(void)
{
    x_CAddressBookApp::ForceClassReferences();

    // Insert your own class references here
    // by calling TCL_FORCE_REFERENCE for each class
    // See x_CAddressBookApp.cp
}
```

Again, the superclass function is called:

```cpp
void x_CAddressBookApp::ForceClassReferences(void)
{
    CApplication::ForceClassReferences();

    /* From References.c */
    ReferenceStdClasses(); /* See template file Ref */

    TCL_FORCE_REFERENCE(CAddressBookPane);
}
```

The call to ReferenceStdClasses makes a reference to all the classes listed in the file References.c. The classes there are required for object input and output. Notice that Visual Architect automatically has included a reference to the CAddressBookPane class.

DoCommand will be modified later if you add any application class commands:

```cpp
void CAddressBookApp::DoCommand(long theCommand)
{
    switch (theCommand)
    {
```
// Insert your command cases here

default:
    x_CAddressBookApp::DoCommand(theCommand);
    break;
}

The superclass DoCommand handles a simple About box. Other commands are passed on for handling elsewhere.

void x_CAddressBookApp::DoCommand(long theCommand)
{
    switch (theCommand)
    {
    case cmdAbout:

        // Simple About alert. Subclasses will probably
        // trap this command to do something sexier.

        PositionDialog('ALRT', ALRTabout);
        InitCursor();
        Alert(ALRTabout, NULL);
        break;

    default:
        CApplication::DoCommand(theCommand);
        break;
    }
}

The call to GetDocTypeFromDialog can be removed because there is just one file type for the AddressBook but there is not much cost to keeping it here.

OSType CAddressBookApp::GetDocTypeFromDialog(CDialogDirector *dialog box)
{
    return 0;
}
There are a few more functions defined in the x_CAddressBookApp class that you may need to override in your derived class.

The call to SetUpFileParameters gets the list of document types from the application dialog box in Visual Architect:

```cpp
void x_CAddressBookApp::SetUpFileParameters()
{
    CApplication::SetUpFileParameters();

    // File types as defined in CAddressBookApp.h

    sfNumTypes = kNumFileTypes;
    sfFileTypes[0] = kFileType1;
    sfFileTypes[1] = kFileType2;
    sfFileTypes[2] = kFileType3;
    sfFileTypes[3] = kFileType4;
    gSignature = 'C++4';
}
```

SetUpMenus simply calls CApplication::SetUpMenus. If you wanted to take any actions before setting up the menus (such as creating floating windows) you could do this by adding a function, CAddressBookApp::SetUpMenus, which performs the creation, and then calls x_CAddressBookApp::SetUpMenus:

```cpp
void x_CAddressBookApp::SetUpMenus()
{
    CApplication::SetUpMenus();
}
```

CreateDocument creates a new CAddressBookDoc object in memory, and then tries to initialize the document and create a new file. If this fails, the document is disposed in the call to ForgetObject:

```cpp
void x_CAddressBookApp::CreateDocument()
{
    CDocument *theDocument = NULL;

    theDocument = new CAddressBookDoc;
    TRY
    {
        ((CAddressBookDoc*) theDocument)->ICAddressBookDoc();
        theDocument->NewFile();
    }
```
CATCH
{
   ForgetObject(theDocument);
}
ENDTRY

OpenDocument is like CreateDocument but, instead of creating a new file, it attempts to open the file with the OpenFile function. If this fails, the document is disposed:

```cpp
void x_CAddressBookApp::OpenDocument(SFReply *macSFReply)
{
    CDocument *theDocument = NULL;

    theDocument = new CAddressBookDoc;
    TRY
    {
        ((CAddressBookDoc*) theDocument)->ICAddressBookDoc();
        theDocument->OpenFile(macSFReply);
    }
    CATCH
    {
        ForgetObject(theDocument);
    }
    ENDTRY
}
```

FileAlreadyOpen and its EqualFile function make sure that the same file is not opened more than once:

```cpp
static Boolean EqualFile(CDirector *obj, long param)
{
    FSSpec theFileSpec;
    CFile *theFile;
    FSSpec *theParam = (FSSpec*) param;

    if (member(obj, CDocument))
    {
        theFile = ((CDocument*)obj)->itsFile;
        if (theFile == NULL)
            return FALSE;
        theFile->GetFSSpec(&theFileSpec);
    }
```
return theFileSpec.vRefNum == theParam->vRefNum  
   && theFileSpec.parID == theParam->parID  
   && !IUEqualString(theFileSpec.name, theParam->name);  
}
return FALSE;
}

Boolean x_CAddressBookApp::FileAlreadyOpen(
    CFile    *aFile)
{
    FSSpec    fileSpec;
    CDirector *theDirector = NULL;

    aFile->GetFSSpec(&fileSpec);
    if (itsDirectors)    /* Search for matching file */
        theDirector = (CDirector*) itsDirectors->FindItem(EqualFile, (long) &fileSpec);
    if (theDirector)    /* If find one, bring to front */
        theDirector->GetWindow()->Select();
    return theDirector != NULL;  /* Tell caller */
}

The CAddressBookDoc Class

The CAddressBookDoc class does less than previous document classes you have seen. Its main task
is to keep the document in sync with the window.

The class declaration of CAddressBookDoc follows:

#include Entry.h

class CAddressBookDoc : public x_CAddressBookDoc
{
    public:

    TCL_DECLARE_CLASS

    void ICAccessAddressBookDoc(void);
virtual void MakeNewContents(void);
virtual void ContentsToWindow(void);

virtual TIterator *CreateDatabaseIterator();

private:
    TDatabase *fDatabase;
};

const long kReasonNewEntry = 1001;

And the x_CAddressBookDoc class is declared as follows:
#include "ItsContentsClass.h"

#define x_CAddressBookDoc_super csaver<ITSCONTENTSCLASS>
class CFile;

class x_CAddressBookDoc : public x_CAddressBookDoc_super
{
    public:

    TCL_DECLARE_CLASS

    // Pointers to panes in window
    CStaticText *fAddressBook_Stat3;
    CStaticText *fAddressBook_Stat4;
    CStaticText *fAddressBook_Stat5;
    CStaticText *fAddressBook_Stat6;
    CStaticText *fAddressBook_Stat7;
    CStaticText *fAddressBook_Stat8;
    CStaticText *fAddressBook_Name;
    CStaticText *fAddressBook_Address;
    CStaticText *fAddressBook_City;
    CStaticText *fAddressBook_State;
    CStaticText *fAddressBook_Zip;
    CStaticText *fAddressBook_Telephone;

    void Ix_CAddressBookDoc(void);
protected:

    virtual void MakeNewWindow(void);
    virtual void FailOpen(CFile *aFile);
    virtual void PositionWindow(void);

};

Notice that the class x_CAddressBookDoc is derived from the x_CAddressBookDoc_super class, which is defined as a template class based on the CSaver template class. The class declaration for CSaver is as follows:

template<class T> class CSaver : public CDocument {

public:

    // Instance Variable
    T *itsContents; // Contents of document in memory

    // Member Functions

    // Construction/Destruction
    CSaver() { itsContents = NULL; }
    CSaver(Boolean printable); ~CSaver();

    // New/Open
    virtual void NewFile();
    virtual void OpenFile(SFReply *macSFReply);

    // Save/Save As/Revert
    virtual Boolean DoSave();
    virtual Boolean DoSaveAs(SFReply *macSFReply);
    virtual void DoRevert();

protected:

    // I/O
    virtual void MakeNewWindow(void) = 0;
    virtual void ReadDocument();
    virtual void WriteDocument();
virtual void MakeNewContents(void) = 0;
virtual void ReadContents(CFileStream *aStream);
virtual void WriteContents(CFileStream *aStream);

virtual void ContentsToWindow();
virtual void WindowToContents();

// Utility
virtual void ExistingFile();
virtual OSType NewFileType();
virtual void PositionWindow();
virtual void MakeWindowName(Str255 name);
};

The CSaver class is used for defining document types that will be written and read to and from storage devices, such as hard drives. The itsContents data member will hold the data that is read and written from disk. At this point, the document is not readable or writable until you make the new entry dialog box and add the related code.

Visual Architect automatically creates pointers for all the elements in the window (see the set of CStaticText pointers).

Let's go back to CAddressBookDoc. The function WindowToContents was removed because this window doesn't allow editing; it is only used if changes to the document can be made from the document window. To iterate through the database, the function CreateDatabaseIterator was added to this class. The fDatabase variable holds a pointer to the database.

For this initial working version of AddressBook, in ICAddressBookDoc you will create the database and fill it with some records. You can view these before you add the new entry dialog box.

#include CAddressBookPane.h
void CAddressBookDoc::ICAddressBookDoc()
{
    Ix_CAddressBookDoc();

    fDatabase = new TDatabase;

    fDatabase->Insert(new TEntry("Rhodes, Neil", "1328 Clock Avenue", "Redlands", "CA", "92374", "(909) 798-5792"));
    fDatabase->Insert(new TEntry("McKeehan, Julie", "1328 Clock Avenue", "Redlands", "CA", "92374", "(909) 798-5792"));
    fDatabase->Insert(new TEntry("Rhodes, Alexander", "1328 Clock Avenue", "Redlands", "CA", "92374", "(909) 798-5792"));
}
fDatabase->Insert(new TEntry("Rhodes, Nicholas", "1328 Clock Avenue", "Redlands", "CA", "92374", "(909) 798-5792");
fDatabase->Insert(new TEntry("Rhodes, Mary", "1234 Main St.", "Fullerton", "CA", "92635", "(714) 555-1212");
fDatabase->Insert(new TEntry("Rhodes, Fen", "123 Park Avenue", "Long Beach", "CA", "02138", "(310) 798-5792");
fDatabase->Insert(new TEntry("La Galy, Carolyn", "123 Longridge Avenue", "Santa Monica", "CA", "92374", "(818) 555-1212");
}

The superclass initializer simply calls IDocument:
void x_CAddressBookDoc::Ix_CAddressBookDoc()
{
    IDocument(gApplication, TRUE);

    // Initialize data members below.
}

There are no changes now for the MakeNewContents function. Later on, the database will be created here.
void CAddressBookDoc::MakeNewContents()
{
    // Initialize document contents and itsWindow here
}

First, call the superclass to set up the window from the database. x_CAddressBookDoc::ContentsToWindow. Point the gopher to the main pane and call the pane’s InitialSetup function (described in the next section).
void CAddressBookDoc::ContentsToWindow(void)
{
    x_CAddressBookDoc::ContentsToWindow();
    itsGopher = itsMainPane;
    ((CAddressBookPane *) itsMainPane)->InitialSetup(this);
}

To iterate through the database, CreateDatabaseIterator returns an iterator for the entries in the database:
TIterator *CAddressBookDoc::CreateDatabaseIterator()
{
TIterator *iter;

iter = new TIterator(*fDatabase);
return iter;
}

There is no override for MakeNewWindow, so it is defined in the x_CAddressBookDoc class as follows:

```cpp
void x_CAddressBookDoc::MakeNewWindow(void)
{
    itsWindow = TCLGetNamedWindow("pAddressBook", this);

    itsMainPane = (CPane*) TCLGetItemPointer(itsWindow, 0);

    // Initialize pointers to the subpanes in the window

    fAddressBook_Stat3 = (CStaticText*) itsWindow->FindViewByID(kAddressBook_Stat3ID);
    ASSERT(member(fAddressBook_Stat3, CStaticText));

    fAddressBook_Stat4 = (CStaticText*) itsWindow->FindViewByID(kAddressBook_Stat4ID);
    ASSERT(member(fAddressBook_Stat4, CStaticText));

    fAddressBook_Stat5 = (CStaticText*) itsWindow->FindViewByID(kAddressBook_Stat5ID);
    ASSERT(member(fAddressBook_Stat5, CStaticText));

    fAddressBook_Stat6 = (CStaticText*) itsWindow->FindViewByID(kAddressBook_Stat6ID);
    ASSERT(member(fAddressBook_Stat6, CStaticText));

    fAddressBook_Stat7 = (CStaticText*) itsWindow->FindViewByID(kAddressBook_Stat7ID);
    ASSERT(member(fAddressBook_Stat7, CStaticText));

    fAddressBook_Stat8 = (CStaticText*) itsWindow->FindViewByID(kAddressBook_Stat8ID);
    ASSERT(member(fAddressBook_Stat8, CStaticText));

    fAddressBook_Name = (CStaticText*) itsWindow->FindViewByID(kAddressBook_NameID);
    ASSERT(member(fAddressBook_Name, CStaticText));

    fAddressBook_Address = (CStaticText*) itsWindow->FindViewByID(kAddressBook_AddressID);
    ASSERT(member(fAddressBook_Address, CStaticText));

    fAddressBook_City = (CStaticText*) itsWindow->FindViewByID(kAddressBook_CityID);
    ASSERT(member(fAddressBook_City, CStaticText));
```
fAddressBook_State = (CStaticText*) itsWindow->FindViewByID(kAddressBook_StateID);
ASSERT(member(fAddressBook_State, CStaticText));

fAddressBook_Zip = (CStaticText*) itsWindow->FindViewByID(kAddressBook_ZipID);
ASSERT(member(fAddressBook_Zip, CStaticText));

fAddressBook_Telephone = (CStaticText*) itsWindow->FindViewByID(kAddressBook_TelephoneID);
ASSERT(member(fAddressBook_Telephone, CStaticText));

The many calls to FindViewByID return pointers to the static text items in the window.

With the FileAlreadyOpen call, FailOpen checks whether the file is already open in AddressBook. If so, the error is silenced. An override of this function could handle other cases for failure (for example, the file is open in another application).

```cpp
void x_CAddressBookDoc::FailOpen(
    CFile *aFile)
{
    /* Only the application knows */
    if ( ((CAddressBookApp*)gApplication)->FileAlreadyOpen(aFile) )
        Failure(kSilentErr, 0);
}
```

PositionWindow is overridden to prevent the window from being positioned twice. When a window is created from a view, it is automatically staggered by the decorator class.

```cpp
void x_CAddressBookDoc::PositionWindow()
{
}
```

**The CAddressBookPane Class**

The CAddressBookPane class is responsible for the following:

- Ensuring that each database view menu item works. For example, if Go Forward is selected, CAddressBookPane directs the next name in the database to be shown.

- Handling the New Entry menu item.
Making sure that the correct entry is displayed.

CAddressBookPane does not directly display an entry. Instead, it acts as the enclosing pane for a number of CStaticText objects, each of which displays a component of the entry.

As created by Visual Architect, CAddressBookPane is defined as follows:

```cpp
class CAddressBookPane : public x_CAddressBookPane
{

public:

    TCL_DECLARE_CLASS

    /** Commands **/
    virtual void DoCommand(long theCommand);

    /** Object I/O **/
    virtual void PutTo(CStream &aStream);
    virtual void GetFrom(CStream &aStream);

};
```

None of these three functions (DoCommand, PutTo, or GetFrom) require new code in this class, so they will be removed. Some of the functions implemented in the superclass will be reimplemented, however, so they need to be added to the class definition:

```cpp
#include CAddressBookDoc.h
class CAddressBookPane : public x_CAddressBookPane
{

public:

    TCL_DECLARE_CLASS

    virtual void UpdateMenus();

    virtual void DoCmdGoForward(void);
    virtual void DoCmdGoBackward(void);
    virtual void DoCmdGoToBeginning(void);
    virtual void DoCmdGoToEnd(void);

    virtual void InitialSetup(CAddressBookDoc *doc);

protected:

    virtual void ProviderChanged(CCollaborator *aProvider, long reason, void* info);

};
```
private:
  virtual void UpdateEntryData(void);
  TIterator *fAddressBookIterator;
  CActiveDoc *GetAddressBookDoc();
  CActiveDoc *fAddressBookDoc;
};

The CActiveBookPane class is derived from CActivePane, which is declared as follows:

class CActiveBookPane : public CPAntorama {
public:

  TCL_DECLARE_CLASS

  /** Commands **/
   virtual void DoCommand(long theCommand);
   virtual void UpdateMenus();
   virtual void DoCmdGoBackward(void);
   virtual void DoCmdGoForward(void);
   virtual void DoCmdGoToBeginning(void);
   virtual void DoCmdGoToEnd(void);

  /** Object I/O **/
   virtual void PutTo(CStream &aStream);
   virtual void GetFrom(CStream &aStream);
};

The InitialSetup function sets the value of fAddressBookDoc and sets up a dependency between the pane and the document. This allows the pane to be notified whenever the document data is changed so that it can update the display. To iterate through the database, fAddressBookIterator is set to the result of a call to CreateDatabaseIterator. And DoCmdGoToBeginning makes sure that the first record of the database is displayed.

#include AppCommands.h
#include <CStaticText.h>
#include <CBartender.h>
extern CBartender *gBartender;
void CAddressBookPane::InitialSetup(CAddressBookDoc *doc)
{
    fAddressBookDoc = doc;
    DependUpon(fAddressBookDoc);

    fAddressBookIterator = doc->CreateDatabaseIterator();
    DoCmdGoToBeginning();
}

The calls to DoCmdGoBackward, DoCmdGoForward, DoCmdGoToBeginning, and DoCmdGoToEnd are overridden in the CAddressBookPane class:

void CAddressBookPane::DoCmdGoBackward(void)
{
    fAddressBookIterator->MoveBackward();
    UpdateEntryData();
}

void CAddressBookPane::DoCmdGoForward(void)
{
    fAddressBookIterator->MoveForward();
    UpdateEntryData();
}

void CAddressBookPane::DoCmdGoToBeginning(void)
{
    fAddressBookIterator->MoveToBeginning();
    UpdateEntryData();
}

void CAddressBookPane::DoCmdGoToEnd(void)
{
    fAddressBookIterator->MoveToEnd();
    UpdateEntryData();
}

Each of these simply sends the appropriate message to the database iterator and updates the display.

Take a look at both UpdateMenus functions:

void CAddressBookPane::UpdateMenus()
{
void x_CAddressBookPane::UpdateMenus()
{
    CPanorama::UpdateMenus();
    gBartender->EnableCmd(cmdGoBackward);
    gBartender->EnableCmd(cmdGoForward);
    gBartender->EnableCmd(cmdGoToBeginning);
    gBartender->EnableCmd(cmdGoToEnd);
}

By default, Visual Architect creates code to enable all new menu items in the superclass. It is up to you to override this and disable any menu items when they should not be available. CAddressBookPane::UpdateMenus disables the menu items appropriately (for example, Go Forward isn’t available if the database is displaying the last record).

To update the display with a record from the database, UpdateEntryData calls GetCurrent to get the current database entry, copies the fields to local variables, and then sets the static text items to these variables:

void CAddressBookPane::UpdateEntryData(void)
{
    const TEntry *entry = (TEntry *) fAddressBookIterator->GetCurrent();
    char name[256] = "";
    char address[256] = "";
    char city[256] = "";
    char state[256] = "";
    char zip[256] = "";
    char phone[256] = "";
if (entry) {
    ::strcpy(name, entry->GetName());
    ::strcpy(address, entry->GetAddress());
    ::strcpy(city, entry->GetCity());
    ::strcpy(state, entry->GetState());
    ::strcpy(zip, entry->GetZip());
    ::strcpy(phone, entry->GetTelephone());
}
GetAddressBookDoc()->fAddressBook_Name->SetTextPtr(name, (long) strlen(name));
GetAddressBookDoc()->fAddressBook_Address->SetTextPtr(address, (long)
    strlen(address));
GetAddressBookDoc()->fAddressBook_City->SetTextPtr(city, (long) strlen(city));
GetAddressBookDoc()->fAddressBook_State->SetTextPtr(state, (long) strlen(state));
GetAddressBookDoc()->fAddressBook_Zip->SetTextPtr(zip, (long) strlen(zip));
GetAddressBookDoc()->fAddressBook_Telephone->SetTextPtr(phone, (long)
    strlen(phone));
}
GetAddressBookDoc returns the value of the fAddressBookDoc member. This is used by the
UpdateEntryData function shown earlier:
CAddressBookDoc *CAddressBookPane::GetAddressBookDoc()
{
    return fAddressBookDoc;
}
ProviderChanged is called as a result of the dependency on the document that was created earlier
in InitialSetup. The only significant reason that the document would be changed is for a new
entry (which will not happen yet because you don’t support the new entry command). In that
case, iterate through the database until the new entry is found, and display the new record:
void CAddressBookPane::ProviderChanged(CCollaborator *aProvider, long reason, void *
    *info)
{
    if (reason == kReasonNewEntry) {
        TEntry *theEntry = (TEntry *) info;

        // Display the new entry
        fAddressBookIterator->MoveToBeginning();
        for (; !fAddressBookIterator->NoMore();
            fAddressBookIterator->MoveForward()) {
            if (theEntry == (TEntry *) fAddressBookIterator->GetCurrent())
                break;
    
}
There is one more non-empty function defined in the superclass. Remember that this was generated automatically by Visual Architect! You don’t need to write basic code like this anymore.

DoCommand passes control on to the individual DoCmd_ functions, which were overridden and provided in the derived class CAddressBookPane:

```cpp
void x_CAddressBookPane::DoCommand(long theCommand)
{
    switch (theCommand)
    {
        case cmdGoBackward:
            DoCmdGoBackward();
            break;
        case cmdGoForward:
            DoCmdGoForward();
            break;
        case cmdGoToBeginning:
            DoCmdGoToBeginning();
            break;
        case cmdGoToEnd:
            DoCmdGoToEnd();
            break;
        default:
            CPanorama::DoCommand(theCommand);
    }
}
```

**Running the Incomplete AddressBook Application**

You now have enough code to test the AddressBook. Try to run the program. If you have problems, the complete source code and project are included in the AddressBook 1 folder.

Notice that the AddressBook database already contains records that were added as the application began. The menu commands let you move forward and backward through these records. And the Go to Beginning and Go to End commands work.
The Save As command will create a document on your disk but no records will be saved there. In the next section, you add saving and reading commands, along with the Add Entry dialog box.

If you open a saved file, you will get the same records because the database is always constructed with the same entries.

**Adding the New Entry Dialog Box with Visual Architect**

To make the AddressBook fully functional, create a dialog box for entering new records. Open the Visual Architect.rsrc file from the THINK Project Manager.

From the View menu, choose New View. Type NewEntryDialog. Set the View Kind to Modal Dialog (see fig. 21.28).

![Figure 21.28. Creating the New Entry dialog box.](image)

Your new dialog box should appear. From the View menu, choose View Info. You need to change the information so that it matches that shown in figure 21.29.

![Figure 21.29. The New Entry dialog box information.](image)

Follow these steps:

1. Set the procID to 5 (you can click on the second to last mini window to set this).
2. Set the width to 338 and the height to 149.
Choose OK to close the Dialog Box Information dialog box. Now add the text New Entry to the top of the dialog box using the static text tool. For reference, the tools palette is shown in figure 21.30.

Create the labels for the six fields as you did earlier using the static text tool. For the field contents, instead of using the static text tool, use the editable text tool. Set the font of each to geneva, and the size to 9.

If you want to set the coordinates manually, the following table shows the settings used in the samples. As before, the order of the coordinate values is left, top, width, and height:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>70, 23, 250, 12</td>
</tr>
<tr>
<td>Address</td>
<td>70, 46, 250, 12</td>
</tr>
<tr>
<td>City</td>
<td>70, 69, 80, 12</td>
</tr>
<tr>
<td>State</td>
<td>195, 69, 20, 12</td>
</tr>
<tr>
<td>Zip</td>
<td>245, 69, 75, 12</td>
</tr>
<tr>
<td>Telephone</td>
<td>70, 92, 200, 12</td>
</tr>
</tbody>
</table>

As before, each of the fields must have its identifier set so that the data can be retrieved when the user enters a new record. The code as shown will expect the identifiers to be Name, Address, City, State, Zip, and Telephone.

You also need to set the maximum length for entries in each of these editable text elements. The length is set in the CDialogText part of the Pane Info dialog box.
The last step in building this dialog box is to add the OK and Cancel buttons. Select the button tool from the tools palette. Drag out an OK button. Visual Architect correctly assumes that you are creating an OK button for the dialog box. Press the Enter key to accept the button.

Drag out another button for the Cancel button. Notice that Visual Architect again makes the right assumption. Press the Enter key to accept this Cancel button.

Modify the existing cmdNewEntry command in the Commands dialog box (accessed from the Edit menu). Scroll down and click on cmdNewEntry. Select CAddressBookPane from the In Class popup. Select Open from the Do popup. Select CNewEntryDialog from the View popup. This makes Visual Architect generate code to open the new entry dialog box.

Save your work.

Now, from the TPM menu, choose Generate. You should now have five new files in the Source folder:

CNewEntryDialog.cp
CNewEntryDialog.h
NewEntryDialogItems.h
x_CNewEntryDialog.cp
x_CNewEntryDialog.h

The files CNewEntryDialog.cp and x_CNewEntryDialog.cp should have been added to your project file in the Application segment.
Modifying the Newly Created Code

You can remove some functions from the CNewEntryDialog class:

- ProviderChanged. You don’t need to know when specific items change.
- DoCommand. There are no commands for this class.
- UpdateData. You don’t need to react when specific items change.

Following is the class definition for CNewEntryDialog:

```cpp
class CDirectorOwner;

class CNewEntryDialog : public x_CNewEntryDialog
{
public:

    TCL_DECLARE_CLASS

    // Insert your own public data members here

    void ICNewEntryDialog(CDirectorOwner *aSupervisor);

protected:

    virtual void BeginData(CNewEntryDialogData *initial);
    virtual void EndData(CNewEntryDialogData *final);
};
```

You need to modify only the EndData function to create a new entry object from the dialog box data, and then add that entry to the document:

```cpp
#include CAddressBookDoc.h
#include Entry.h

void CNewEntryDialog::EndData(CNewEntryDialogData *final)
{
    if (dismissCmd != cmdCancel && Length(final->fNewEntryDialog_Name) > 0) {
        TEntry *newEntry;

        newEntry = new TEntry(
            p2cstr(final->fNewEntryDialog_Name),
```
p2cstr(final->fNewEntryDialog_Address),
p2cstr(final->fNewEntryDialog_City),
p2cstr(final->fNewEntryDialog_State),
p2cstr(final->fNewEntryDialog_Zip),
p2cstr(final->fNewEntryDialog_Telephone)
);

((CAddressBookDoc *) itsSupervisor)->AddEntry(newEntry);
}
}

The rest of the code in this section is as it was created automatically by Visual Architect. If you
are working through this chapter and creating the program step-by-step, you don’t need to
modify any of the following functions.

The initialize function ICNewEntryDialog simply calls the superclass initializer, IX_CNewEntryDialog:
void CNewEntryDialog::ICNewEntryDialog(CDirectorOwner *aSupervisor)
{
    // Initialize data members that must be set up before
    // BeginData is called here

    x_CNewEntryDialog::IX_CNewEntryDialog(aSupervisor);

    // Initialize any other data members here
}

BeginData is called when the window is created. CollectPaneValues will copy the initial values
set in Visual Architect. If other values were wanted, this function could be changed:
void CNewEntryDialog::BeginData(CNewEntryDialogData *initial)
{
    CollectPaneValues(*initial);
}

From the header file x_CNewEntryDialog.h, you get not only the definition of the xCNewEntryDialog
superclass but also some other data structures:
typedef struct
typedef struct
{
    Str255    stringvalue;   // CDialogText
    long      longvalue;     // CIntegerText
    short     value;         // All other controls and buttons
    Point     selection;     // CArrayPane
} CNewEntryDialogData;

class CDirectorOwner;
class CPanorama;
class CPane;

class x_CNewEntryDialog : public CDialogDirector
{
public:

    TCL_DECLARE_CLASS

    // Pointers to panes in window
    CStaticText  *fNewEntryDialog.Stat1;
    CStaticText  *fNewEntryDialog.Stat2;
    CStaticText  *fNewEntryDialog.Stat3;
    CStaticText  *fNewEntryDialog.Stat4;
    CStaticText  *fNewEntryDialog.Stat5;
    CStaticText  *fNewEntryDialog.Stat6;
    CButton      *fNewEntryDialog.Butn7;
    CButton      *fNewEntryDialog.Butn8;

    }
The CNewDialogEntryData structure is used for setting and retrieving the complete dialog box data—all the fields at once rather than one at a time.

The CNewEntryDialogUpdate structure is passed as a parameter in UpdateData, which identifies when a single field changes.

Then comes the class declaration for x_CNewEntryDialog. The member functions are described here. Remember that this is not code that you have to change; it is created this way by Visual Architect.
The initialize function `Ix_CNewEntryDialog` calls its superclass initializer. Setting `ignore` to `TRUE` disables calls to `UpdateData` because that wouldn’t make sense while the dialog box is being built:

```cpp
void x_CNewEntryDialog::Ix_CNewEntryDialog(CDirectorOwner *aSupervisor)
{
    IDialogDirector(aSupervisor);

    // There are several circumstances where we don't want
    // ProviderChanged to be called. During initialization,
    // during calls to UpdateData, and so forth. The ignore flag
    // heads these off.

    ignore = TRUE;        /* Don't call UpdateData now             */
    MakeNewWindow();      /* Create the dialog box's window         */
    DoBeginData();        /* Gather initial values               */

    ignore = FALSE;
}
```

`MakeNewWindow` creates the dialog box from the view resource created in Visual Architect. It initializes the many pane pointers to point to their objects:

```cpp
void x_CNewEntryDialog::MakeNewWindow(void)
{
    itsWindow = TCLGetNamedWindow("\pNewEntryDialog", this);

    // Initialize pointers to the subpanes in the window

    fNewEntryDialog_Stat1 = (CStaticText*) FindPane(kNewEntryDialog_Stat1ID);
    ASSERT(member(fNewEntryDialog_Stat1, CStaticText));

    fNewEntryDialog_Stat2 = (CStaticText*) FindPane(kNewEntryDialog_Stat2ID);
    ASSERT(member(fNewEntryDialog_Stat2, CStaticText));

    fNewEntryDialog_Stat3 = (CStaticText*) FindPane(kNewEntryDialog_Stat3ID);
    ASSERT(member(fNewEntryDialog_Stat3, CStaticText));

    fNewEntryDialog_Stat4 = (CStaticText*) FindPane(kNewEntryDialog_Stat4ID);
```
ASSERT(member(fNewEntryDialog_Stat4, CStaticText));

fNewEntryDialog_Stat5 = (CStaticText*) FindPane(kNewEntryDialog_Stat5ID);
ASSERT(member(fNewEntryDialog_Stat5, CStaticText));

fNewEntryDialog_Stat6 = (CStaticText*) FindPane(kNewEntryDialog_Stat6ID);
ASSERT(member(fNewEntryDialog_Stat6, CStaticText));

fNewEntryDialog_Butn7 = (CButton*) FindPane(kNewEntryDialog_Butn7ID);
ASSERT(member(fNewEntryDialog_Butn7, CButton));

fNewEntryDialog_Butn8 = (CButton*) FindPane(kNewEntryDialog_Butn8ID);
ASSERT(member(fNewEntryDialog_Butn8, CButton));

fNewEntryDialog_Name = (CDialogText*) FindPane(kNewEntryDialog_NameID);
ASSERT(member(fNewEntryDialog_Name, CDialogText));

fNewEntryDialog_Address = (CDialogText*) FindPane(kNewEntryDialog_AddressID);
ASSERT(member(fNewEntryDialog_Address, CDialogText));

fNewEntryDialog_City = (CDialogText*) FindPane(kNewEntryDialog_CityID);
ASSERT(member(fNewEntryDialog_City, CDialogText));

fNewEntryDialog_State = (CDialogText*) FindPane(kNewEntryDialog_StateID);
ASSERT(member(fNewEntryDialog_State, CDialogText));

fNewEntryDialog_Zip = (CDialogText*) FindPane(kNewEntryDialog_ZipID);
ASSERT(member(fNewEntryDialog_Zip, CDialogText));

fNewEntryDialog_Telephone = (CDialogText*) FindPane(kNewEntryDialog_TelephoneID);
ASSERT(member(fNewEntryDialog_Telephone, CDialogText));

fNewEntryDialog_Stat15 = (CStaticText*) FindPane(kNewEntryDialog_Stat15ID);
ASSERT(member(fNewEntryDialog_Stat15, CStaticText));

}

To find a single pane (such as one of the entry fields) FindPane calls FindViewByID:

CPane *x_CNewEntryDialog::FindPane(long ID)
{

}
return (CPane*) itsWindow->FindViewByID(ID);
}

DoBeginData copies the data from a single data structure into the dialog box fields:

```cpp
void x_CNewEntryDialog::DoBeginData()
{
    CNewEntryDialogData data = {0};    /* The initial value record */
    BeginData(&data);                   /* Ask subclass for initial values */

    // Initialize the panes based on the values supplied.
    // The ASSERT statements ensure that the generated
    // code is in sync with the view resource.
    fNewEntryDialog_Name->SetTextString(data.fNewEntryDialog_Name);
    fNewEntryDialog_Address->SetTextString(data.fNewEntryDialog_Address);
    fNewEntryDialog_City->SetTextString(data.fNewEntryDialog_City);
    fNewEntryDialog_State->SetTextString(data.fNewEntryDialog_State);
    fNewEntryDialog_Zip->SetTextString(data.fNewEntryDialog_Zip);
    fNewEntryDialog_Telephone->SetTextString(data.fNewEntryDialog_Telephone);

    // Save the initial values in case user cancels
    saveData = data;
}
```

And DoEndData pulls the values of the fields out by calling CollectPaneValues unless the user has canceled the dialog box by pressing the Cancel button:

```cpp
void x_CNewEntryDialog::DoEndData(long theCommand)
{
    CNewEntryDialogData data;    /* The initial value record */
    RgnHandle rgn;              /* Selection region */
```
/* If user canceled the dialog box, */
if (theCommand == cmdCancel) /* return the initial values */
{
    data = saveData;
    EndData(&data);
    return;
}

CollectPaneValues(data); // Get current pane values
EndData(&data); // Tell the derived class

CollectPaneValues gets the values of each of the editable text fields and stores them in the CNewEntryDialogData structure:

void x_CNewEntryDialog::CollectPaneValues(CNewEntryDialogData& data)
{
    RgnHandle rgn; /* Selection region */

    /* Collect values from panes */
    fNameNewEntryDialog_Name->GetTextString(data.fNameNewEntryDialog_Name);

    fNameNewEntryDialog_Address->GetTextString(data.fNameNewEntryDialog_Address);

    fNameNewEntryDialog_City->GetTextString(data.fNameNewEntryDialog_City);

    fNameNewEntryDialog_State->GetTextString(data.fNameNewEntryDialog_State);

    fNameNewEntryDialog_Zip->GetTextString(data.fNameNewEntryDialog_Zip);

    fNameNewEntryDialog_Telephone->GetTextString(data.fNameNewEntryDialog_Telephone);
}

BeginData does nothing and has to be overridden in the subclass if you want to modify the initial data:

void x_CNewEntryDialog::BeginData(CNewEntryDialogData *initial)
{
}

496
Symantec C++ Programming for the Macintosh, 2nd Edition
If the subclass wants to do an action whenever the data in a single field changes, override UpdateData:

```c
void x_CNewEntryDialog::UpdateData(CNewEntryDialogUpdate *update, long itemNo)
{
}
```

EndData gives the subclass the opportunity to do something with the data from the dialog box. The default behavior is to do nothing:

```c
void x_CNewEntryDialog::EndData(CNewEntryDialogData *final)
{
}
```

DoModalDialog is an overridden function that adds postprocessing to the regular DoModalDialog to get the data out of the dialog box:

```c
long x_CNewEntryDialog::DoModalDialog(long defaultCmd)
{
    long result = CDialogDirector::DoModalDialog(defaultCmd);

    DoEndData(result);
    return result;
}
```

This override of Close catches the dialog box closing if it is not modal and calls DoEndData:

```c
Boolean x_CNewEntryDialog::Close(Boolean quitting)
{
    if (itsWindow && !itsWindow->active)
        itsWindow->Select();

    if (EndDialog(cmdOK, TRUE))
    {
        /* For a modal dialog box, exit through */
        if (itsWindow->IsModal()) /* DoModalDialog */
        {
            dismissCmd = cmdClose;
            return TRUE;
        }
    }
```
else
{
    /* Collect final values and call */
    DoEndData(cmdOK); /* EndData */
    /* Do the Close */
    return CDialogDirector::Close(quitting);
}
else
    return FALSE;
}

If ignore is false, ProviderChanged will call UpdateData with the changed data from the dialog box and the ID of the changed pane:

```c++
void x_CNewEntryDialog::ProviderChanged(CCollaborator *aProvider, long reason, void* info)
{
    CNewEntryDialogUpdate data; /* The update value record */
    Str255 str;
    Boolean saveIgnore = ignore;

    if (ignore) /* Don't be a chatterbox */
        return;
    ignore = TRUE;

    TRY
    {
        if (FALSE) {}
        else if (reason == dialogTextChanged && aProvider == fNewEntryDialog_Name)
        {
            ((CDialogText*) aProvider)->GetTextString(data.stringvalue);
            UpdateData(&data, kNewEntryDialog_NameID);
        }
        else if (reason == dialogTextChanged && aProvider == fNewEntryDialog_Address)
        {
            ((CDialogText*) aProvider)->GetTextString(data.stringvalue);
            UpdateData(&data, kNewEntryDialog_AddressID);
        }
        else if (reason == dialogTextChanged && aProvider == fNewEntryDialog_AddressID)
        {
            ((CDialogText*) aProvider)->GetTextString(data.stringvalue);
            UpdateData(&data, kNewEntryDialog_AddressID);
        }
        else if (reason == dialogTextChanged
```
& & aProvider == fNewEntryDialog_City)
{
    ((CDialogText*) aProvider)->GetTextString(data.stringvalue);
    UpdateData(&data, kNewEntryDialog_CityID);
}
else if (reason == dialogTextChanged
    && aProvider == fNewEntryDialog_State)
{
    ((CDialogText*) aProvider)->GetTextString(data.stringvalue);
    UpdateData(&data, kNewEntryDialog_StateID);
}
else if (reason == dialogTextChanged
    && aProvider == fNewEntryDialog_Zip)
{
    ((CDialogText*) aProvider)->GetTextString(data.stringvalue);
    UpdateData(&data, kNewEntryDialog_ZipID);
}
else if (reason == dialogTextChanged
    && aProvider == fNewEntryDialog_Telephone)
{
    ((CDialogText*) aProvider)->GetTextString(data.stringvalue);
    UpdateData(&data, kNewEntryDialog_TelephoneID);
}
else
    CDialogDirector::ProviderChanged(aProvider, reason, info);
}
CATCH
    ignore = saveIgnore;
ENDTRY

ignore = saveIgnore;  /* ProviderChanged() cannot Close()! */

Adding Streaming to the Document

Document saving in TCL 2.0 is performed with streams rather than standard file operations. You have already seen streams earlier in the book used for input and output. Streams allow more flexibility for supporting different input and output sources. File I/O works only with files. If a new storage type, such as an object-oriented file system, is created, the application using standard file I/O needs to be rewritten to read and write to this new system. If the applications are written to use streams for input and output, then only the streaming functions need to be updated to work with this new storage type.
To start, create a class declaration for an auto-destructing object with the member functions PutTo and GetFrom. These are the streaming functions used for writing to and reading from a stream, respectively. Create a private data member to hold the database:

```cpp
class CStreamableDatabase TCL_AUTO_DESTRUCT_OBJECT {
public:

    TCL_DECLARE_CLASS

    virtual void PutTo(CStream& stream);
    virtual void GetFrom(CStream& stream);

    TDatabase &GetDatabase();

private:

    TDatabase fDatabase;
};
```

GetDatabase returns the database:

```cpp
TDatabase &CStreamableDatabase::GetDatabase()
{
    return fDatabase;
}
```

The PutTo function saves the data from the database to the stream. To do this, create an iterator to iterate through the database counting the number of records. That number is then streamed out as total. Next, the contents of each of the records is output to the stream, field by field:

```cpp
void CStreamableDatabase::PutTo(CStream& stream)
{
    long total = 0;
    TIterator iter(fDatabase);

    for (iter.MoveToBeginning(); !iter.NoMore();
        iter.MoveForward())
        total++;
    stream « total;
    for (iter.MoveToBeginning(); !iter.NoMore();
        iter.MoveForward()) {
        TEntry *curEntry = (TEntry *) iter.GetCurrent();

        stream « curEntry->GetName();
        stream « curEntry->GetAddress();
        stream « curEntry->GetCity();
```
stream << curEntry->GetState();
stream << curEntry->GetZip();
stream << curEntry->GetTelephone();
}

GetFrom does the opposite of PutTo. It reads a record count from the stream, and then reads from
the stream that many records and enters each record into the database:

void CStreamableDatabase::GetFrom(CStream& stream)
{
    long total;

    stream >> total;

    for (long i = 0; i < total; i++) {
        char name[256];
        char address[256];
        char city[256];
        char state[256];
        char zip[256];
        char telephone[256];

        stream >> name;
        stream >> address;
        stream >> city;
        stream >> state;
        stream >> zip;
        stream >> telephone;

        TEntry *newEntry = new TEntry(name, address, city,
                                state, zip, telephone);
        fDatabase.Insert(newEntry);
    }
}

In the file ItsContentsClass.h, change the occurrences of CCollaborator to CStreamableDatabase. When the CSaver_class.cpp is compiled, it generates a CSaver class from the template defined in the file CSaver.tem. Here are the function templates.

The constructor calls the superclass constructor and sets itsContents to NULL:

template<class T>
CSaver<T>::CSaver(
    Boolean printable
) : CDocument(printable)
{
    itsContents = NULL; // Document has no contents
    TCL_END_CONSTRUCTOR
}

The destructor disposes of the member itsContents:

template<class T>
CSaver<T>::~CSaver()
{
    TCL_START_DESTRUCTOR
    TCLForgetObject(itsContents);
}

NewFile makes a new window object and fills in the contents of the window. Calls to
PositionWindow, MakeWindowName, and SetTitle put the window in a good location and set the
title of the window:

template<class T>
void CSaver<T>::NewFile()
{
    Str255 title;

    MakeNewWindow();       // Create window
    MakeNewContents();     // And contents
    ContentsToWindow();    // May be something to do

    PositionWindow();      // Must be done before making name
    MakeWindowName(title); // Set window title
    itsWindow->SetTitle(title);

    SetChanged(FALSE);     // New document isn't dirty
    itsWindow->Select();   // Make the front window
}

OpenFile attempts to open a file and then call ReadDocument:

template<class T>
void CSaver<T>::OpenFile(
    SFReply *macSFReply
)
CDataFile *theFile = NULL;
FSSpec spec;

try_
{
    theFile = new(CDataFile); // Try to open this file
    theFile->IDataFile();
    theFile->SFSpecify(macSFReply);
    theFile->GetFSSpec(&spec);
    FailOpen(&spec); // Fail if open in this application
    theFile->Open(fsRdWrPerm); // Fail if open in another app
    // (and lots of other reasons)
    itsFile = theFile; // Promote to document file
}
catch_all_()
{
    TCLForgetObject(theFile);
}
end_try_

ReadDocument(); // Read document from file

To set the window title, MakeWindowTitle concatenates the name of the window (set in Visual
Architect or left as untitled) and the window count:

template<class T>
void CSaver<T>::MakeWindowTitle(
    Str255 newName)
{
    Str31 count;

    if (itsFile != NULL)
        itsFile->GetName(newName); // Return file name
    else
    {
        // Append window count to window
        // title (from resource)
        itsWindow->GetTitle(newName);
        if (Length(newName) == 0)
            GetIndString(newName, STRcommon, strUNTITLED);
        NumToString(gDecorator->GetWCount(), count);
        ConcatPStrings(newName, (unsigned char *)"p ");

ConcatPStrings(newName, count);
} 
} 

To keep the windows orderly, PositionWindow tells the decorator (gDecorator) to PlaceNewWindow:

```cpp
template<class T>
void CSaver<T>::PositionWindow()
{
  gDecorator->PlaceNewWindow(itsWindow);
}
```

To get the data from a document, ReadContents calls GetObject:

```cpp
template<class T>
void CSaver<T>::ReadContents(CFileStream *aStream)
{
  GetObject(*aStream, (T*) itsContents);
}
```

ReadDocument gets a document from a stream and displays it in a new window:

```cpp
template<class T>
void CSaver<T>::ReadDocument()
{
  CFileStream *theStream = NULL;
  Str255 theName;

  try_
  {
    // Create a file stream
    theStream = NewInputFileStream((CDataFile*) itsFile);
    ReadContents(theStream); // Read contents of document
    SetChanged(FALSE); // Not dirty any longer
    delete theStream; // Get rid of stream
  }
  catch_all()
  {
    delete theStream;
  }
```
end_try_

if (itsWindow == NULL) // If window wasn't created by
    MakeNewWindow(); // ReadContents, make it now
ContentsToWindow(); // Make contents displayable

PositionWindow(); // Must be done before making name
MakeWindowName(theName); // Change the window title
itsWindow->SetTitle(theName);
itsWindow->Select(); // Activate the window
}

To save the data in the document, WriteContents calls PutObject for the document:

```cpp
template<class T>
void C_Saver<T>::WriteContents(
    CFileStream *aStream)
{
    PutObject(*aStream, itsContents);
}
```

Because saving a file is something that the user should never be prevented from accomplishing safely, the WriteDocument function uses an exception handler to make sure that the document is saved:

```cpp
template<class T>
void C_Saver<T>::WriteDocument()
{
    CFileStream *theStream = NULL;

    // We never want Save to fail, but
    // it can. Prevent a memory leak
    try_
    {
        // Allow critical operation memory
        // to be used for Save
        gApplication->SetCriticalOperation(TRUE);

        WindowToContents(); // Make sure contents are current
        // Create a stream and write the
        // contents to it
        theStream = NewOutputFileStream((CDataFile*) itsFile);
        WriteContents(theStream);
    }
```
theStream->Close();
SetChanged(FALSE); // Not dirty any longer
delete theStream;    // Clean up after Save complete
gApplication->SetCriticalOperation(FALSE);
}
catch_all()
{
    delete theStream;  // Same cleanup if Save failed
    gApplication->SetCriticalOperation(FALSE);
}
end_try_
}

DoSave responds to the Save menu item and treats the request as a Save As if the file name hasn’t been selected yet; otherwise, the document is written out:

\[
\text{template<class } T \text{>}
\]

Boolean CSaver\(<T>: :\text{DoSave}() \{
    // If no file associated with this
    if (itsFile == NULL) // document, treat like Save As
        return DoSaveFileAs();
    else
    {
        WriteDocument(); // Write document to file
        return TRUE;     // TRUE or failed
    }
\}

Because there is only one type of file for the AddressBook application, NewFileTypes returns just the first type in the list of file types. If there were more file types, this function would be overridden:

\[
\text{template<class } T \text{>}
\]

OSType CSaver\(<T>: :\text{NewFileType}() \{
    return gApplication->sfFileTypes[0];
\}

DoSaveAs responds to the Save As menu item in the file menu. It deletes the old file from memory, creates a new file in memory, and then calls DoSave with the new document:
template<class T>
Boolean CSaver<T>::DoSaveAs(
   SFReply *macSFReply)
{
   OSErr theError;
   CDataFile *theFile = NULL;
   FSSpec spec;

   try_  // Most of the following operations
      // can fail for various reasons
   {
      if (itsFile != NULL)  // Delete the current file
         TCLForgetObject(itsFile);

      theFile = new(CDataFile);  // Create a new file
      theFile->IDataFile();
      theFile->SFSpecify(macSFReply);
      theFile->GetFSSpec(&spec);
      FailOpen(&spec);  // Thanks to Ken Paulson!
      itsFile = theFile;
      if (theFile->ExistsOnDisk())  // If file is on disk
         ExistingFile();  // deal with it
      itsFile->CreateNew(gSignature, NewFileType());

      theFile->Open(fsRdWrPerm);  // Open the file

         // Change window name to match
         // the new file name
      itsWindow->SetTitle(macSFReply->fName);
   }
   catch_all_()
   {
      TCLForgetObject(theFile);
   }
   end_try_

   itsFile = theFile;  // Set instance variable

      // Let DoSave() write the document to disk. Note
      // that DoSave() may have called DoSaveAs(), but
      // itsFile is now non-NULL, so the recursion stops.
To revert a document, DoRevert either goes back to the last saved version or creates a new document if there was no previous file:

```cpp
template<class T>
void CSaver<T>::DoRevert()
{
    TCLForgetObject(itsWindow);
#ifdef CONTENTS_DISPOSE
    TCLForgetObject(itsContents);
#else
    delete itsContents;
#endif
    itsGopher = this; // TCL bug - not reset when window
                     // disposed

    if (itsFile != NULL)
        ReadDocument(); // Reread the file
    else
        NewFile(); // Back to square one
}
```

ContentsToWindow defines a protocol for making the window reflect the document’s contents. Subclasses have to implement this behavior:

```cpp
template<class T>
void CSaver<T>::ContentsToWindow()
{
}
```

To keep the document in sync with the Window, WindowToContents defines a protocol for updating a document’s contents:

```cpp
template<class T>
void CSaver<T>::WindowToContents()
{
}
```

When the user chooses to create a new file with the same name as an existing file, ExistingFile calls ThrowOut on the old file before the new file data is saved. This function can be overridden to change this behavior:
template<class T>
void CSaver<T>::ExistingFile()
{
    itsFile->ThrowOut();
}

Modifying the Document To Support Streaming

The previous version of CAddressBookDoc created some database records when the document was constructed. Now the document is going to support reading and writing the AddressBook data from a file.

The database data member comes from the CSaver template class, so it has been removed from the CAddressBookDoc class definition:

```cpp
class CAddressBookDoc : public x_CAddressBookDoc
{
    public:

    TCL_DECLARE_CLASS

    void ICAddressBookDoc(void);

    virtual void MakeNewContents(void);
    virtual void ContentsToWindow(void);

    virtual Titerator *CreateDatabaseIterator();
    virtual void AddEntry(TEntry *entry);
};
```

The new constructor follows. The database will be created in MakeNewContents:

```cpp
void CAddressBookDoc::ICAddressBookDoc()
{
    Ix_CAddressBookDoc();
}
```

```cpp
void CAddressBookDoc::MakeNewContents()
```
{  
    itsContents = new CStreamableDatabase;
    // Initialize document contents and itsWindow here
}

The Resources

Unlike previous chapters, this project doesn't need all the resources that have been used before. The MENU, DLOG, DITL, WIND, FREF, BNDL, and other resources are all maintained from the Visual Architect.

Summary

This chapter contains an application that was developed in two parts.

First, you learned the advantages of bifurcating an application. Then, you learned how to generate content classes that take advantage of their independence from the TCL by using multiple inheritance. The second part of the development focused on using Visual Architect to design the user interface, including the use of modal dialogs. After the interface was finished, it was melded to the working database with TDatabase to form a usable address book application.

Exercises

1. Add a Find menu item that displays a dialog box into which the user can type a name. If an entry matches the name, the address book pane should then display it.

2. As it is written, the binary tree has poor worst-case performance. For example, if entries are inserted in alphabetical order (when reading from an existing file, for instance), the binary tree is completely lopsided, with each node having only one child.

   Rewrite the TDatabase class to use a balanced binary tree, such as a Red-Black, an AVL, or a Splay tree.

3. AddressBook loses memory each time you create a document and then close it. The CAddressBookPane allocates a TIterator (by calling TAddressBookDoc::CreateDatabaseIterator), but never frees it. Fix this memory leak.
This appendix provides a broad overview of issues related to implementing compatibility and portability features in code. MPW C++, THINK C, and standard C++ code are covered.
Converting from MPW C++

If you have been using Apple's MPW C++, there are some differences that you may want to know about. These are discussed in the following sections.

No #pragma Segment
Symantec C++ does not recognize the following pragma:

#pragma segment segmentName

Segmentation in Symantec C++ is done on a file-by-file basis from the Project window.

No SingleObject
Apple C++ provides an extension to C++, SingleObject. This is a special class that cannot use multiple inheritance. Furthermore, any derived classes of this class also are prohibited from using multiple inheritance. Its advantage is that virtual function calls are slightly faster with SingleObject because there is no overhead for multiple inheritance.

I/O Differences with SANE
The way in which Symantec C++ outputs illegal floating-point values is different, as shown in table A.1.

Table A.1. Differences in How Nonnumbers Are Output

<table>
<thead>
<tr>
<th></th>
<th>Symantec C++</th>
<th>MPW C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infinity</td>
<td>INFINITY</td>
<td>INF</td>
</tr>
<tr>
<td>Negative Infinity</td>
<td>-INFINITY</td>
<td>-INF</td>
</tr>
<tr>
<td>Nonnumber</td>
<td>NAN or -NAN</td>
<td>NAN(numeral) or -NAN(numeral), where numeral is the NAN code</td>
</tr>
</tbody>
</table>
Converting from THINK C

THINK C and Symantec C++ handle enumeration sizes and Pascal string literals differently.

Enumeration Sizes

THINK C and Symantec C++ calculate the size of enumeration types differently. For example, given the following enumerations:

```c
typedef enum {a=0, k=255} Enum1;
type def enum {b=-128, l=127} Enum2;
type def enum {c=0, m=256} Enum3;
type def enum {d=-129, n=127} Enum4;
type def enum {e=-32768, o=32767} Enum5;
type def enum {f=0, p=32768} Enum6;
type def enum {g=0, q=32769} Enum7;
type def enum {x=0, r=65535} Enum8;
type def enum {i=0, s=65536} Enum9;
type def enum {j=-32769, t=32767} Enum10;
```

THINK C calculates the sizes as follows:
```
sizeof(Enum1)=2
sizeof(Enum2)=1
sizeof(Enum3)=2
sizeof(Enum4)=2
sizeof(Enum5)=2
sizeof(Enum6)=4
sizeof(Enum7)=4
sizeof(Enum8)=4
sizeof(Enum9)=4
sizeof(Enum10)=4
```

Symantec C++ and MPW C++ calculate the sizes as follows:
```
sizeof(Enum1)=1
sizeof(Enum2)=1
sizeof(Enum3)=2
sizeof(Enum4)=2
```
Size of Pascal String Literals

Pascal string literals (which have the form "\pchars") are laid out differently in memory. In Symantec C++, MPW C, and MPW C++, the following string occupies six bytes:

"\pNeil"

Following are the contents of six bytes:

4 'N' 'e' 'i' 'l' 0

In THINK C, on the other hand, the string is only five bytes long because it is missing the 0:

4 'N' 'e' 'i' 'l'

Differences from Standard C++

Symantec C++'s variations from standard C++ can be grouped into three categories: modifications to standard C++, extensions, and additional classes.

Modifications

In standard C++, the size of an enumeration is the same as the size of an int. In Symantec C++, the size of an enumeration is as big as necessary to fit the enumeration constants. Symantec C++ calculates sizes in this way for compatibility with Macintosh Pascal compilers.

You can turn off this modification with the Enums Are Always Ints check box in the Language Settings panel of the Symantec C++ Options dialog box.

Extensions

Extensions are additions to Standard C++. They provide additional Macintosh functionality.
SANE Support
To support the Standard Apple Numerics Environment (SANE), Symantec C++ has added some new types to C++:

- **extended** is an 80-bit floating-point number. It is equivalent to `long double`.
- **comp** is a 64-bit integer.

You should use the type `long double` instead of `extended` so that your code is portable. If you need a 64-bit integer, your only choice is to use `comp` because there is no synonymous type.

Direct Function Calls
Direct calls are used to call traps. A direct call provides 68000 machine instructions that are executed instead of the standard jump to subroutine (JSR) call.

As an example, the Toolbox call `FrameRect` is declared as follows:

```pascal
pascal void FrameRect(const Rect *r) = 0xABA1;
```

The call to `FrameRect` looks like this:

```pascal
FrameRect(&aRectangle);
```

Because `FrameRect` is a direct function call, the example is compiled as follows:

```assembly
PEA address of aRectangle; // push the one parameter
FrameRect                // the two bytes A8 and A1
```

If it were *not* a direct function call, the resulting assembly would be the following:

```assembly
PEA address of aRectangle; // push the one parameter
JSR FrameRect              // call the subroutine FrameRect
```

The Pascal Modifier
The `pascal` type modifier specifies that a function uses Pascal calling conventions. Most Toolbox routines use Pascal calling conventions. These routines, therefore, are declared with the `pascal` modifier. Here is an example:

```pascal
pascal void FillRect(const Rect *r);
```
With Pascal calling conventions, parameters are pushed left to right, return results are passed on the stack (in space reserved by the caller), and the function pops the parameters off of the stack rather than the caller.

**Pascal String Literals**

In addition to standard string literals (like “Hello”), Symantec C++ supports Pascal string literals, which begin with \p. A Pascal string literal is one byte longer than a standard string literal because a length byte is placed at the front.

For a standard string literal, “Hello” is an array with the following six bytes:

```
'H' 'e' 'l' 'l' 'o' 0
```

The corresponding Pascal string literal, “\pHello,” is an array with seven bytes:

```
5 'H' 'e' 'l' 'l' 'o' 0
```

The type of a standard string literal is an array of char; the type of a Pascal string literal is an array of unsigned char.

**Additional Classes**

The additional classes discussed in this section are abstract classes that modify how their derived classes behave.

**HandleObject Class**

Objects of type HandleObject are allocated as relocatable blocks on the heap with NewHandle. To preserve source code compatibility, the compiler automatically inserts an extra dereference.

Suppose that you have the following code:

```
aHandleObject->fDataMember
```

It is compiled as if it were written as follows:

```
(*aHandleObject)->fDataMember
```

Objects of this type must be allocated dynamically, so the following definition is illegal:

```
TMyHandleObject aHandleObject;
```
There are other restrictions placed on `HandleObjects`:

- No multiple inheritance.
- No operator overloading.
- No pointers to member functions.
- No pointers to data members.
- They can be allocated only dynamically. They must be declared as pointers and have to be allocated with `new`.

**PascalObject Class**

The `PascalObject` class provides objects that behave like those in Object Pascal.

Like `HandleObjects`, `PascalObjects` are relocatable blocks on the heap. In addition, calls to member functions are handled quite differently. Normally, calls to member functions are implemented using a table of function pointers stored in the object. For `PascalObjects`, calls to member functions are implemented by a call to a dispatch routine. To determine the actual function to call, this routine performs a lookup based on the class ID stored in the object and the desired member function.
SourceServer and ToolServer are two utilities that can make working on large or multi-person projects easier. SourceServer is a source code control system that comes with Symantec C++. ToolServer, which must be ordered from the Apple Programmers and Developers Association (APDA), is a stripped-down MPW that can execute tools and scripts.

This appendix introduces you to SourceServer, giving you enough information so that you can use it in your projects. You need to refer to the documentation if you want to know about advanced features. This appendix also provides a very brief introduction to Apple’s ToolServer.
SourceServer

Source code control is an important feature of a programming environment. Symantec has added source code control to the THINK Project Manager with Apple's Projector system, using the SourceServer application.

By way of introducing SourceServer, let's first examine the two most likely scenarios in which source code control can be useful:

▼ **A multiple-person program.** There are many development conflicts that can arise when working on a multiple-person program. One of most prevalent problems occurs when two people attempt to modify the same file at the same time. SourceServer deals with this by having Projector maintain control of the project files. Projector ensures that only one person can have write access to a file at a given time, while still enabling other people to have read-only access. Under this system, if you want to modify a file, you check it out with write access. When you are finished modifying it, you check it in and retain a copy with read-only access. During the period you have it checked out, you are guaranteed that no one else can modify it.

▼ **A program with two tracks of development.** Imagine that you have the typical development effort of a program that has been released as 1.0 version and is undergoing a major enhancement (to become version 2.0). The enhancement engenders a long period of development, and version 1.0 needs some minor bug fix releases (versions 1.01, 1.1, and so on) before 2.0 is ready. Commonly, to deal with this, you start with two copies of 1.0; one becomes version 1.01 and the other becomes 2.0. The two sets of sources diverge, and the fixes in version 1.01 quite likely are not reflected in version 2.0.

When using Projector, all the files in the program are held in a database and the files for version 1.0 are named by a revision name (perhaps VERSION 1.0). Any file that needs to be modified for both 2.0 and 1.01 is branched. The branched version contains the 1.01 modification, and the unbranched version contains the 2.0 version. Thus, you have a project database that reflects a real-world situation—both versions were spawned from version 1.0. Figure B.1 shows a small example of this.
In the past, many THINK C programmers found Projector so useful that they used MPW to get access to it. They developed their code using THINK C, and then exported it to MPW for Projector. As you might expect, this system was an inconvenient solution. Apple fortuitously broke Projector out of MPW and turned it into an application (SourceServer) that responds to Projector commands via Apple Events. THINK Project Manager is a client that sends Apple Events to SourceServer in response to selections from the SourceServer menu.

Now that you know a little of its history and situations in which you might want to use it, you can learn how to handle source code control with SourceServer.

**Creating a Project**

First, you have to create some source files so that SourceServer has something to control. Perform the following steps:

1. Create a new C++ IOStreams project named PowersOfTwo.+ within the THINK Project Manager.

2. Create Main.cp, Power.h, and Power.cp files. Main.cp should include the following code:

   ```
   #include <iostream.h>
   #include "Power.h"
   ```
void main()
{
    cout << "2 to the power 8 = " << Power(2, 8) << 'n';
}

Use the following code for Power.h:

    int Power(int number, int powerToRaise);

The following code constitutes Power.cp:

    #include "Power.h"
    int Power(int number, int powerToRaise)
    {
        int returnValue = 1;

        for (int i = 0; i < powerToRaise; i++)
        {
            returnValue *= number;
        }
        return returnValue;
    }

3. Test the program and make sure that it prints the following:

    2 to the power 8 = 256

4. When you have finished, your project should look like the one shown in figure B.2.

Figure B.2. The PowersOfTwo Project window.
**Launching SourceServer**

All SourceServer commands are accessed from the SourceServer menu item in the Source menu. From the SourceServer menu, select Launch SourceServer (see fig. B.3).

![SourceServer Menu](image)

**Figure B.3.** Launching SourceServer.

After you launch SourceServer, verify that it is running by checking the Application menu (see fig. B.4). You should see the SourceServer icon and name in the menu.

![Application Menu](image)

**Figure B.4.** Verifying that SourceServer is running.

**NOTE**

THINK Project Manager uses Apple Events to communicate with SourceServer and ToolServer. As a result, you must be running System 7 or later to use them.

Now, all the items in the SourceServer menu should be available. (Before selecting Launch SourceServer, they were dimmed.) Your next step is to create a Projector database for your files.
WARNING

THINK Project Manager uses the term project to refer to the files used to build a program. Projector also uses this word to refer to a collection of files, including all the archived revisions of the files. A Projector project (also called a Projector database) is one large file containing all the revisions of all the files. To avoid confusion, throughout this discussion the term Projector database is used for Projector, and the term project is used to mean a THINK Project Manager project.

Creating a Projector Database

Select New ProjectorDB from the SourceServer menu. Create your Projector database in the same folder as the PowersOfTwo.n project and name it PowersOfTwo (see fig. B.5).

A dialog box appears in which you can provide a comment for this Projector database. Figure B.6 shows this dialog box, along with the comment you should type.

Figure B.5. Creating a new Projector database.

Figure B.6. The New ProjectorDB Comment dialog box.
This creates a folder named PowersOfTwo with a file in it named ProjectorDB (for Projector database). Figure B.7 shows the folder layout.

![Folder Layout](image)

**Figure B.7.** The folder layout after creating the Projector database.

### The SourceServer Check In Dialog Box

Now you are ready to add your files to the Projector project. Select Check In from the SourceServer menu. Figure B.8 shows the dialog box that is displayed.

![Check In Dialog](image)

**Figure B.8.** The Check In dialog box.

The Project pop-up menu contains all the mounted projects (see fig. B.9).

![Project Pop-up](image)

**Figure B.9.** The Project pop-up menu.
In the Select Files section of the dialog, you determine which files you want to check in (see fig. B.10).

![Select files dialog box]

**Figure B.10.** The Select files section of the Check In dialog box.

The options in this dialog box are as follows:

- The All button selects all the files in the list.
- The Modified button selects all the files that have been modified since the last check-in.
- The None button deselects all the files in the list.
- The Sources button selects all the files that are listed in the project window (normally .cp files).
- The Headers button selects all the files that are referenced from the source files (normally .h files).
- The Show <...> Files check box, if checked, lists the files that belong to the THINK project tree (normally Macintosh or standard C++ header or TCL files).

To the right of the select files section is the area in which you enter the user name, task description, and project comment (see fig. B.11).

In the User field, enter the name of the user who is checking in. In the Task field, enter a short description of the task associated with this modification. You can enter a longer description of this modification in the Comment field.
Figure B.11. The user and task information portion of the dialog box.

Below the Comment field are the choices that affect how the files are checked in (see fig. B.12).

Figure B.12. The file parameter section of the dialog.

The three radio buttons are mutually exclusive. If you select Keep read-only, after checking in, the files are checked out read-only to you. If you select Keep modifiable, after checking in, the files are checked out modifiable. If you select Delete my copy, after checking in, your copies of the files are deleted.

The three check boxes on the right control aspects of file check-in. If you select Touch mod date, SourceServer checks in the files with the current date, rather than using the modification dates of the files. If you select Verify, SourceServer double-checks the check-in by checking out each checked-in file to a temporary file and comparing the temporary with the checked-in file. SourceServer aborts the check-in if the comparison reveals a difference. This is useful if you are worried about data loss. If you select Branch, the checked files are made into a new branch.

After you have set all the parameters, you have three choices that govern your next action (see fig. B13).
Figure B.13. The action choices in the dialog box.

If you select the Done button, the dialog box is closed. If you select Check In, the files are checked in with the parameters you have set. If you select Info, you get more information about the selected files.

Now that you know what each choice entails, check in some source files.

**Adding Files to the Projector Database**

To check in the files you created in the PowersOfTwo project, do the following:

1. Click the All button to select all three files in the PowersOfTwo folder (or Shift-click each file to select them all).
2. Type a description in the Task field (see fig. B.14).
3. Click the Check In button.

![Projector screenshot](image)

Figure B.14. Checking in three files.

Projector copies the files to its database and leaves you with a copy of each file. Because the Keep read-only radio button is selected, each of the three files is now read-only.
After you leave the Check In window, open Main.cp and look at the lower left corner. It should now look like the file shown in figure B.15. Main.cp should have a read-only icon in the bottom left corner. This icon signifies that the file cannot be modified.

![Read-only Icon](image1)

**Figure B.15.** A read-only file.

### Checking Out a File

Now, modify Main.cp so that it calculates two numbers the user chooses, rather than 2 to the power 8. To modify the file, you need to check out a modifiable copy. To do that, make Main.cp the front-most window and select Check Out ‘Main.cp’ from the SourceServer menu (see fig. B.16).

![SourceServer Menu](image2)

**Figure B.16.** Checking out Main.cp.

After the file is selected, you see the Check Out dialog box, shown in figure B.17. Fill in the Task and Comment fields and click OK.
Now, if you look at Main.cp in the THINK Project Manger, it should have a writable icon (see fig. B.18).

To modify Main.cp so that it can accept numbers from users and calculate their power, change it to the following:

```cpp
#include <iostream.h>
#include "Power.h"

void main()
{
    int number;
    int power;

    cin >> number >> power;

    cout << number << " to the power " << power << " = " << Power(number, power) << '\n';
}
```

Save the file using Save from the File menu.
Checking In a File

Although you could use Check In to check in the changes to Main.cp, there is an easier way. Make sure that Main.cp is the front-most window in the THINK Project Manager and select Check In ‘Main.cp’ from the SourceServer menu (see fig. B.19).

![Figure B.19. Checking in the Main.cp file.]

The Check In dialog box is displayed. With it, you can specify a task and command in the Task and Comment fields (see fig. B.20). Because you already specified a task and comment when checking out the file, you don’t need to specify one when checking in.

![Figure B.20. The Check In dialog box.]

Getting Information about Revisions

Now that you have two versions of Main.cp, you need to learn how to access the information the Projector database keeps on each file. Select Check Out and double-click on Main.cp. The Check Out window shows a list of all revisions of Main.cp (see fig. B.21).
Figure B.21. The Check Out window showing the revisions of Main.cp.

Select version 2 and click the Info button (this is the version you just checked in). The window shows the task, comment, user, and date of that version (see fig. B.22).

Figure B.22. The Check Out window showing information about revision 2.

Click on version 1 to see the information about that version (see fig. B.23). Note that the Task field displays Initial revisions, the task description you typed when you initially checked in Main.cp.
Checking Out Old Versions of Files

Now that you know how to get information about each version, it's time to actually check the file out. Select Check Out and double-click on Main.cp. The Check Out window again shows a list of all revisions of Main.cp.

Select revision 1. Because you are not going to be modifying revision 1 of Main.cp, select the Read-only radio button, and then click on the Check Out button (see fig. B.24).

Figure B.23. The Check Out window showing information about revision 1.

Figure B.24. Checking out revision 1 of Main.cp.
Open Main.cp. You should find the original version of the file (see fig. B.25).

![Main.cp](image)

**Figure B.25.** Revision 1 of Main.cp.

### Checking Out on a Branch Revision

Sometimes, you need to modify a revision of a file that is not the latest. To do that, follow these steps:

1. Go to the Check Out dialog box.
2. Select the revision you want to check out.
3. Select Modifiable and check Branch (see fig. B.26).

![Checkout](image)

**Figure B.26.** Checking out a branch.

At check-in, your file is a new revision, branched off the revision you checked out. In this case, you checked out based on revision 1 and checked back in the branched revision 1a1 (see fig. B.27).
Sharing a Projector Database

If you are going to have multiple users sharing a Projector database, you need to set it up accordingly. The database needs to be stored on a volume that can be accessed by all of the users. Each user runs a copy of SourceServer on his or her machine with the shared database mounted. SourceServer arbitrates access to the database and ensures that only one user has a particular revision of a file checked out modifiable at any given time. SourceServer also lets you know who has checked out the file you want to use.

Modifying Files without the Projector Database

SourceServer keeps a copy of each checked-in version of a file and retrieves any version you request. To ensure that two people are not modifying the same file simultaneously, SourceServer requires you to check the file out modifiable if you want to make changes. This is all well and good, but what do you do if you want to change a file but don’t have access to the Projector database? For example, the database may be on a shared server at work, and you may be working out of the office without access to the server. THINK Project Manager deals with this by enabling you to create modified read-only versions of files.

To simulate a lack of access to the Projector database and the need to get files, do the following:

1. Quit SourceServer (using Quit SourceServer from the SourceServer menu).
2. Open Power.cp (which should be version 1).
3. Select Modify Read-Only from the File menu of the THINK Project Manager, as shown in figure B.28.

Figure B.27. The results of creating a branched file.
Modify Read-Only marks a source file as a modified read-only copy. (Figure B.29 shows what the modified read-only icon looks like.) When you check out files this way, you can modify them. Checking them back in may be a problem, however. When the Projector database is available, you can check the file in, as long as it has not been checked out modifiable in the meantime. If the file has been checked out, it is your responsibility to merge your changes with the changes of the person who checked out the modifiable file.

By using Modify Read-Only as a shortcut, you are subverting Projector’s capability to maintain file integrity. It should go without saying that you should use this shortcut only when absolutely necessary; normally, this is when the Projector database is unavailable.

**NOTE**

Projector adds a ‘CKID’ resource to each file when it is checked out. The ‘CKID’ resource contains information about what revision the file is and whether the file is read-only, modified read-only, or modifiable. Modify Read-Only sneaks around Projector by changing this resource from read-only to modified read-only.
Sending Commands to SourceServer

SourceServer responds to a variety of different commands. (You can find a list of all the commands in the SourceServer documentation.) To send a command to SourceServer, do the following:

1. Type the command in a window (you may want to use an empty window).
2. Select the command in the window.
3. Press Ctrl-Enter to send the command to SourceServer or choose Send Command from the SourceServer menu.

Figure B.30 shows a sample command that can be sent to SourceServer. Figure B.31 shows the response that SourceServer gives to this command.

Figure B.30. Sending a command to SourceServer.

Following are some of the more useful commands that you might want to send SourceServer:

- ProjectInfo displays information about a file or files in a Projector database.
- DeleteRevisions deletes revisions of a file or the file itself from the Projector database.
- TransferCKID moves the ‘CKID’ resource from one file to another.

For information on how to use these and other SourceServer commands, refer to the SourceServer documentation.
Automatically Mounting Your Projector Database

You can customize the THINK Project Manager to automatically mount a Projector database for you (as well as launch SourceServer) when you mount a project. You create this time-saving device by moving the Projector database file (named ProjectorDB) into the same folder as the project.

Try this trick by moving the ProjectorDB file located in PowersOfTwo folder to the same folder as PowersOfTwo.π. You also can delete the original folder that contained the ProjectorDB file. Figure B.32 shows what your PowersOfTwo folder should look like when you are through.

To test automatic mounting, quit SourceServer and the THINK Project Manager. Double-click on the PowersOfTwo.π Project to relaunch the THINK Project Manager. If everything went as
expected, the THINK Project Manager should have launched SourceServer and mounted the projector database. Choose Check Out and verify that your files are listed.

This ProjectorDB file also can be an alias. If your Projector database is located on a remote machine, you can mount it automatically by creating an alias to the ProjectorDB file on the remote machine. Put the alias (which should be called ProjectorDB) in the same folder as your project. THINK Project Manager takes care of the rest.

**ToolServer**

ToolServer is a stripped-down MPW that can execute tools and scripts. ToolServer comes in its own folder, and the scripts and tools are located in subfolders. If you have ToolServer and want the THINK Project Manager to launch it automatically, create an alias to ToolServer and place the alias (which should be named ToolServer) in the Tools folder of Symantec C++. Figure B.33 shows you the setup you should have.

![ToolServer alias in the Symantec C++ Tools folder.](image)

**Figure B.33.** ToolServer alias in the Symantec C++ Tools folder.

THINK Project Manager enables you to send commands to ToolServer. You send the commands much the same way you send SourceServer commands:

1. Type a ToolServer command into a window (usually an empty untitled window).
2. Select the command.
3. Press Option-Enter (see fig. B.34).

The results of the ToolServer command appear in the same window (see fig. B.35).

For a description of ToolServer commands, refer to the ToolServer documentation.
Figure B.34. Sending a command to ToolServer.

Figure B.35. Output from a ToolServer command.

NOTE

ToolServer is available from the Apple Programmers and Developers Association (APDA). Call APDA at (800) 282-2732 or (716) 871-6555 for more information.
This appendix provides a condensed reference guide for the THINK Class Library. It covers the TCL classes, member functions, data members, global variables, and global functions that are used by the programs in Section 3. It does not, however, cover any functionality not used in those chapters. For a full reference to the TCL, refer to the Symantec C++ documentation.

Class Diagram

Figure C.1 shows an abridged class diagram of the TCL classes used in this book. Not all intermediate classes are shown.

TCL Classes

The classes are listed alphabetically. Within each class, data members are presented first, with member functions following alphabetically. Note that only class members used in the sample programs are shown. Within each member, the syntax is shown before the description.
**CAbstractText**

This is an abstract class used for displaying text. Derived classes actually implement text storage and display. This class handles the creation of tasks to type, cut, copy, and paste.

**GetTextHandle**

```cpp
virtual Handle GetTextHandle();
```

This returns the text stored in this class as a handle. The handle should not be disposed by the caller.

**SetFontName**

```cpp
virtual void SetFontName(Str255 aFontName);
```

This sets the font used to display the text to aFontName.

**SetFontSize**

```cpp
virtual void SetFontSize(short aSize);
```

This sets the size of the font used to display the text to aSize.
**SetTextHandle**

```cpp
virtual void SetTextHandle(Handle textHand);
```

This sets the text to `textHand`. The caller is responsible for disposing the handle.

**SetTextPtr**

```cpp
virtual void SetTextPtr(Ptr textPtr, long numChars);
```

This sets the text to `numChars` characters starting at `textPtr`.

**SetTextString**

```cpp
virtual void SetTextString(Str255 textStr);
```

This sets the text to `textStr`.

**SetFontStyle**

```cpp
virtual void SetFontStyle(short aStyle);
```

This sets the style of the font used to display the text to `aSize`.

**Specify**

```cpp
virtual void Specify(Boolean fEditable, Boolean fSelectable, Boolean fStylable);
```

This sets whether the text is editable (can be changed), selectable (can be selected for copying), or stylable (can use multiple fonts, styles, or sizes).

**CApplication**

Every TCL program must create a `CApplication`. It is the final bureaucrat in the chain of command and handles application-wide events (like the Quit command). You may only create one instance of this class.

**sfFileTypes**

```cpp
SFTypeList sfFileTypes;
```

This contains the file types that are shown when the user chooses Open.
sfNumTypes
short sfNumTypes;

This contains the number of file types in sfFileTypes.

AssignIdleChore
virtual void AssignIdleChore(CChore *theChore);

This adds theChore to the list of idle chores. The Perform function of idle chores is called when there is nothing better to do (no events are available).

CancelIdleChore
virtual void CancelIdleChore(CChore *theChore);

This removes theChore from the list of idle chores.

CApplication
CApplication(short extraMasters, Size aRainyDayFund,
    Size aCriticalBalance, Size aToolboxBalance);

This is the constructor. extraMasters controls the number of calls to the Toolbox call MoreMasters. aRainyDayFund is the amount of memory to reserve for critical memory operations. Of that amount, aCriticalBalance controls the amount to use for critical operations (like quitting or saving), and aToolboxBalance controls the amount to use for Toolbox calls (like CopyBits).

CreateDocument
virtual void CreateDocument();

This is called when a new document is created (either by the user choosing New from the File menu, or when the application is first launched). You must override this routine to create a document, initialize it, and call its NewFile member function.
Exit
virtual void Exit();

You call this routine in your main program after TApplication::Run() returns. This function needs overriding if there is any cleanup to do when the application finishes.

IApplication
void IApplication(short extraMasters, Size aRainyDayFund,
Size aCriticalBalance, Size aToolboxBalance);

This initializes the application object. See CApplication for a description of the parameters.

OpenDocument
virtual void OpenDocument(SFReply *macSFReply);

This is called when an old document is created (either by the user choosing Open from the File menu, or when the user opens a document from the finder). You must override this routine to create a document, initialize it, and call its OpenFile member function.

Run
virtual void Run();

You call this routine in your main program after initializing the application object. It runs the main event loop, and doesn’t return until the user quits.

Quit
virtual Boolean Quit();

The TCL calls this routine when the user quits. This function calls quit for the directors of all open windows. Any director can return false to signify that the user has canceled the quit.

SetupFileParameters
virtual void SetUpFileParameters();
This routine is called to set up parameters for the Toolbox SFGetFile. You should override this routine to set the data members sfNumTypes and sfFileTypes, as well as the global variable gSignature.

**StartupAction**

virtual void StartupAction(short numPreloads);

This routine is called as the application is beginning. numPreloads is the number of files the user selected from the Finder. This routine can be overridden to perform any necessary startup functions (the default version opens an untitled document if numPreloads is zero).

**CBartender**

This class controls the menu bars and menus. When an event occurs that could change the state of the menu items (for example, mouse down events may cause menu items to enable or disable), CBartender marks the menus as dirty. When a mouse down on the menu bar (or a command-key) occurs, CBartender disables and de-checkmarks all the menu items and sends UpdateMenus to the chain of command.

**CheckMarkCmd**

virtual void CheckMarkCmd(long cmdNo, Boolean checked);

This adds or removes a checkmark (depending on checked) to the menu item with command cmdNo.

**DisableCmd**

virtual void DisableCmd(long cmdNo);

This disables the menu item with command cmdNo.

**EnableCmd**

virtual void EnableCmd(long cmdNo);

This enables the menu item with command cmdNo.
CBureaucrat

A bureaucrat is an object that can handle events. Examples of bureaucrats include applications, panes, and documents. Bureaucrats can handle events like keystrokes and menu items.

Bureaucrats are arranged in a chain of command. Each bureaucrat points to a supervisor. When an event arrives, the event is sent to the first bureaucrat in the chain of command (gGopher). A bureaucrat can handle the event, or can pass it on to its supervisor by calling its inherited member function. To make a bureaucrat the gopher, call its BecomeGopher function.

The chain of command is composed of bureaucrats associated with the frontmost window. Any bureaucrat not part of the frontmost window will not be part of the chain of command.

BecomeGopher

virtual Boolean BecomeGopher(Boolean fBecoming);

If fBecoming is true, this makes this bureaucrat the gopher and returns true. If the previous gopher refuses to relinquish control, it returns false.

If fBecoming is false, this bureaucrat is no longer the gopher.

Dawdle

virtual void Dawdle(long *maxSleep);

Dawdle is called for bureaucrats in the chain of command during idle time. Override this function if you need to do idle-time processing. Set maxSleep to the number of ticks until your Dawdle is called again.

DoCommand

virtual void DoCommand(long theCommand);

This handles the command specified by theCommand. The command is commonly a menu choice, or the result of the user pressing a control. Override this routine to take action when a command occurs. If the command is not one you handle, call the inherited routine.

DoKeyDown

virtual void DoKeyDown(char theChar, Byte keyCode,
                      EventRecord *macEvent);
This is called when a key is pressed. theChar contains the character, while keyCode contains the raw key code. macEvent contains the actual event that you can use to check for modifier keys. Override it to handle keystrokes and call the inherited version if it’s a keystroke you don’t handle.

**Notify**

```cpp
virtual void Notify(CTask *theTask);
```

You call this routine to notify a bureaucrat that a task has been completed.

**NotifyClean**

```cpp
virtual void NotifyClean(CTask *theTask);
```

You call this routine to notify a bureaucrat that a task has been completed that didn’t modify anything (for example, a Copy command).

**SetChanged**

```cpp
virtual void SetChanged(Boolean isChanged);
```

Call this routine when something changes within the bureaucrat.

**UpdateMenus**

```cpp
virtual void UpdateMenus();
```

You override this routine to enable and checkmark menu items that your bureaucrat handles. Make sure to call the inherited UpdateMenus.

**CButton**

This implements a standard button. You normally don’t need to derive from this class.

```cpp
CButton( short aWidth, short aHeight,
    short aHEnc1, short aVEncl, StringPtr title,
    Boolean fVisible, short procID,
    CView *anEnclosure, CBureaucrat *aSupervisor);
```
This is the constructor. See CPane for descriptions of aWidth, aHeight, aHEnc1, aVEnc1, anEnclosure, and aSupervisor. The text of the button comes from title. If isVisible is false, the button is not shown. procId specifies the CDEF used to draw the button.

**SetClickCmd**

```cpp
virtual void SetClickCmd(long aClickCmd);
```

This sets the command number that this button will send to DoCommand when it is clicked on.

**SimulateClick**

```cpp
virtual void SimulateClick();
```

This makes the button behave as though the user clicked it. You might call this in response to a keyboard shortcut.

**CChore**

This class represents an action that can be performed at idle time or immediately. Objects of this type are stored in the application in a list of chores.

```cpp
CChore();
```

This is the constructor.

**Perform**

```cpp
virtual void Perform(long *maxSleep);
```

Override this routine to perform whatever action your chore wants. If this is installed as an idle chore, maxSleep specifies how many ticks to wait before Perform is called again.

**CClipboard**

This handles the Macintosh clipboard (the desk scrap). Only one object of this class exists—gClipboard. It also displays the clipboard in a window. It only handles TEXT and PICT data.
**EmptyScrap**

```cpp
virtual void EmptyScrap();
```

This empties out the scrap. Call this before calling PutData to write to the scrap.

**GetData**

```cpp
virtual Boolean GetData(ResType theType, Handle *theData);
```

This finds data in the clipboard of type `theType` and returns it as a handle in `theData`. It returns false if no data of the specified type is present. If you want to know whether data is there, but don't want it, you may pass 0 as the value of `theData` (for instance, to enable Paste in `UpdateMenus`). You are responsible for disposing the returned handle.

**PutData**

```cpp
virtual void PutData(ResType theType, Handle theData);
```

This puts the data in `theData` into the clipboard as type `theType`.

**CCollaborator**

This is an abstract class representing objects that depend on one another. It provides a way for one object (a provider) to notify other objects (dependents) that it has changed.

**BroadcastChange**

```cpp
virtual void BroadcastChange(long reason, void *info);
```

This notifies its dependents that it has changed. `reason` describes the change and `info` provides any additional information necessary to dependents.

**DependUpon**

```cpp
virtual void DependUpon(CCollaborator *aProvider);
```

This adds this object as a dependent of `aProvider`. 
**CDataFile**

This class represents the data fork of a file. Use this class to read raw bytes; otherwise, subclass it.

```c
CDataFile()
```

This is the constructor.

**GetLength**

```c
virtual long GetLength();
```

This returns the number of bytes in the file. The file must already be open.

**ReadAll**

```c
virtual Handle ReadAll();
```

This reads all the data from the file into a new handle. You must dispose of the handle that is returned.

**ReadSome**

```c
virtual void ReadSome(Ptr info, long howMuch);
```

This reads `howMuch` bytes from the file into `info`.

**SetMark**

```c
virtual void SetMark(long howFar, short fromWhere);
```

This sets the file position to `howFar` bytes from `fromWhere`. `fromWhere` is one of `fsFromStart`, `fsFromLEOF`, and `fsFromMark`.

**WriteAll**

```c
virtual void WriteAll(Handle contents);
```

This writes all the data in `contents` to the file.
WriteSome
virtual void WriteSome(Ptr info, long howMuch);

This writes howMuch bytes into the file from info.

CDecorator
Use this class to arrange windows on-screen. The only instance of this class is gDecorator.

GetWCount
virtual short GetWCount();

This returns the number of windows that have been staggered or placed.

PlaceNewWindow
virtual void PlaceNewWindow(CWindow *theWindow);

This places theWindow in a good location on-screen. It also staggers and resizes theWindow.

CDialogDirector
This serves as a director for dialogs. Create one of these to control your dialogs.

IDialogDirector
void IDialogDirector(CDirectorOwner *aSupervisor);

This is the initialization function. aSupervisor is normally the application.

CDialogText
This handles text in a dialog and knows not to handle Return, Enter, or Escape. You can set a maximum number of characters.
**CDirector**
This class handles communication between a window and the application. Every window must have an associated director; this is usually the document.

**Dirty**
Boolean dirty;

This is true if the director has been changed since it was created or last saved, false otherwise. It is set to true automatically when a task is executed.

**itsGopher**
CBureaucrat *itsGopher;

When the director’s window is made active, this is the bureaucrat that should be made the gopher. This is normally some pane within the window.

**itsWindow**
CWindow *itsWindow;

This is the window controlled by the director.

**CDirector**
CDirector(CDirectorOwner *aSupervisor);

This is the constructor.

**CDocument**
This object corresponds to a document on disk. Its contents are displayed in one or more windows. You’ll normally override this class and store your data in data members.

**itsFile**
CFile *itsFile;

This is the file the document is saved in.
itsMainPane
CPane *itsMainPane;

This is the main pane of the document. It is the pane that is printed when the document is printed.

CDocument
CDocument(Boolean printable = TRUE);

This is the constructor. printable specifies whether this document can be printed.

DoSave
virtual Boolean DoSave();

This saves the document in itsFile. You should override it to write your data. If itsFile is 0, you need to call DoSaveFileAs.

DoSaveAs
virtual Boolean DoSaveAs(SFReply *macSFReply);

This saves the file as specified in macSFReply. You should override this routine to do the following:

- Delete the file if it already exists
- Create the file on disk
- Create a CFile to refer to it
- Save the CFile in the data member itsFile
- Call DoSave to actually write the file

DoSaveFileAs
virtual Boolean DoSaveFileAs();

This calls the ToolBox routine SFPutFile. If the user cancels the dialog, it returns false. Otherwise, it calls DoSaveAs and returns true.
IDocument

void IDocument(CApplication *aSupervisor, Boolean printable);

This is the initialization function. aSupervisor is the application. printable specifies whether this document can be printed.

NewFile

virtual void NewFile(void);

This is called when the user creates a new file. You should override this routine to create a window and assign it to itsWindow. You also need to create panes to display in the window and assign one to itsMainPane.

OpenFile

virtual void OpenFile(SFReply *macSFReply);

This is called when the user opens an existing file. You should override this routine to do the following:

- Open the file and read its data
- Create a window and assign it to itsWindow
- Create panes to display in the window and assign one to itsMainPane

CEditText

This class represents a pane with editable text. It uses TextEdit for its editing and for a display, and, therefore, shares TextEdit’s 32K limits: no more than 32K of text and the sum of the line heights cannot exceed 32K.

CEditText

CEditText(CView *anEnclosure, CBureaucrat *aSupervisor,
    short aWidth = 0, short aHeight = 0, short aHEncl = 0, short aVEncl = 0,
    SizingOption aHSizing = sizELASTIC, SizingOption aVSizing= sizELASTIC, short aLineWidth = -1, Boolean aScrollHoriz = FALSE);
This is the constructor. See CPane for a description of anEnclosure, aSupervisor, aWidth, aHeight, aHEncl, aVEncl, aHSizing, and aVSizing.

aLineWidth controls how wide the lines of text should be. If it is negative, the lines are the same width as the pane itself. aScrollHoriz specifies whether the text scrolls horizontally.

**CFile**

This is an abstract class that represents a file on disk.

**Close**

```cpp
virtual void Close();
```

This closes the file.

**CreateNew**

```cpp
virtual void CreateNew(OSType creator, OSType fType);
```

This creates the file with the specified creator and file type.

**ExistsOnDisk**

```cpp
virtual Boolean ExistsOnDisk();
```

This returns true if the file exists, false otherwise.

**GetName**

```cpp
virtual void GetName(Str63 theName);
```

This returns the name of the file.

**Open**

```cpp
virtual void Open(SignedByte permission);
```

This opens the file with the specified permission. Permissions are the same as those of the Toolbox call FSOpen.
**SFSpecify**

virtual void SFSpecify(SFReply *macSFReply);

This initializes the name and location of the file from macSFReply.

**SpecifyHFS**

virtual void SpecifyHFS(Str63 aName, short aVolNum, long aDirID);

This initializes the name and location of the file from aName, aVolNum, and aDirID.

**ThrowOut**

virtual void ThrowOut();

This deletes the file.

**CPane**

A pane is a drawable area with its own coordinate system. Drawing within windows is done in panes. Panes exist in a hierarchy; each pane has an enclosure and can enclose zero or more other panes.

**itsLastTask**

CTask *itsLastTask;

This holds the last task performed in this pane.

**ChangeSize**

virtual void ChangeSize(Rect *delta, Boolean redraw);

You call this to change the size of the pane. The values in delta specify by how much the pane should change on each side, with positive values specifying down and to the right. If redraw is true, the pane will be invalidated.
CPane

CPane(CView *anEnclosure, CBureaucrat *aSupervisor,
short aWidth = 0, short aHeight= 0, short aHEncl= 0, short aVEncl= 0,
SizingOption aHSizing = sizFIXEDSTICKY, SizingOption aVSizing =
sizFIXEDSTICKY);

This is the constructor. anEnclosure is the enclosing view, either of a pane, or a window. aSupervisor is the next bureaucrat in the chain of command, normally the director for the window. The size of the pane is specified in aWidth and aHeight. The location of the pane within its enclosure is specified in aHEncl and aVEncl.

The remaining parameters, aHSizing and aVSizing, specify what happens to this pane when its enclosure changes size. The possible values of aHSizing are: sizFIXEDLeft (location from the left is fixed), sizFIXEDRIGHT (location from the right of the enclosure is fixed), sizFIXEDSTICKY (the pane scrolls with its enclosure), and sizELASTIC (the pane resizes by the same amount as its enclosure). The values for aVSizing are: sizFIXEDTOP, sizFIXEDBOTTOM, sizFIXEDSTICKY, an sizELASTIC.

Draw

virtual void Draw(Rect *area);

This draws a pane. As an optimization, only the portion of the pane intersecting area need be drawn. Your panes will override Draw.

FitToEnclFrame

virtual void FitToEnclFrame(Boolean horizFit, Boolean vertFit);

This fits the pane horizontally and/or vertically (depending on horizFit and vertFit) to the frame of its enclosure. Note that the interior and frame of a view need not be the same.

FitToEnclosure

virtual void FitToEnclosure(Boolean horizFit, Boolean vertFit);

You call this to fit the pane horizontally and/or vertically (depending on horizFit and vertFit) to the interior of its enclosure. Note that the interior and frame of a view need not be the same.
virtual void Refresh();

This invalidates the whole pane, forcing an update event.

virtual void SetBorder(CPaneBorder *aBorder);

This adds an adorning CPaneBorder.

This class draws a border around a pane.

CPaneBorder

CPaneBorder(long borderFlags);

This is the constructor. It controls the appearance of the border. borderFlags can be either: kBorderNone, kBorderLeft, kBorderTop, kBorderRight, kBorderBottom, kBorderOval, kBorderRoundRect, or kBorderFrame.

virtual void SetPenSize(short penWidth, short penHeight);

This sets the height and width of the pen used for the border.

This is a pane that can be scrolled (by placing it in a CScrollView).

virtual void SetPosition(LongPt *aPosition);

This sets the scroll position to aPosition.
CPicture
This is a pane that displays a picture. The picture may or may not come from a resource. It cannot be edited.

CPicture
CPicture(CView *anEnclosure, CBureaucrat *aSupervisor,
short aWidth = 0, short aHeight = 0, short aHEncl = 0, short aVEncl = 0,
SizingOption aHSizing = sizELASTIC, SizingOption aVSizing = sizELASTIC);
This is the constructor. See CPane for information on these parameters.

GetMacPicture
virtual PicHandle GetMacPicture();
This returns the picture handle associated with this object.

SetMacPicture
virtual void SetMacPicture(PicHandle aMacPicture);
This sets aMacPicture as the picture. It will not dispose of or release the old picture.

CScrollPane
This pane does scrolling for an enclosed CPanorama.

CScrollPane
CScrollPane(CView *anEnclosure, CBureaucrat *aSupervisor,
short aWidth, short aHeight, short aHEncl, short aVEncl,
SizingOption aHSizing, SizingOption aVSizing,
Boolean hasHoriz, Boolean hasVert, Boolean hasSizeBox);
InstallPanorama

virtual void InstallPanorama(CPanorama *aPanorama);

Call this routine to notify the scroll pane of its panorama.

CStream

This is an abstract class that represents a byte stream. You read to and write from the stream with the extraction (>>) and insertion (<<) operators.

>>

A variety of extraction and insertion operators are defined for streams, including common types such as int, float, and so on.

In order to support such operators for your own class, add a PutTo(CStream& stream) member function that writes out information to stream. Add a GetFrom(CStream& stream) that reads information from stream into your object. In addition, create a file with the following to cause the necessary templates to be used:

```
#pragma template_access public
#pragma template PutObject(CStream&, YourClass*)
#pragma template GetObject(CStream&, YourClass**)&
#include "CStream.tem"
```

CTask

This is an abstract class that represents actions that can be carried out, undone, and re-carried out. All undoable actions are implemented with tasks.

 undone

Boolean undone;

This is true if the last thing done to the task was an Undo, false otherwise.

CTask

CTask(short aNameIndex);
This is the constructor. aNameIndex is an index into the 'STR#' 130 resource. This resource contains the text for the Undo/Redo menu.

**Do**

virtual void Do();

This carries out the initial action of the task and is only called once. Override this routine to do whatever needs to be done.

**IsUndone**

virtual Boolean IsUndone();

This returns true if the last thing done to the task was an Undo.

**Redo**

virtual void Redo();

This redoes the action of the task. Override it to carry out a redo action.

**Undo**

virtual void Undo();

This undoes the action of the task. Override it to carry out an undo.

**CView**

This is an abstract class representing something that can be seen on-screen. Panes and windows are both views.

**wantsClicks**

Boolean wantsClicks;

This is true if the view, or any subview, can accept mouse clicks.
**Activate**

```cpp
virtual void Activate();
```

This is called for all views within a window when that window is activated.

**Deactivate**

```cpp
virtual void Deactivate();
```

This is called for all views within a window when that window is deactivated.

**GetFrame**

```cpp
virtual void GetFrame(LongRect *theFrame);
```

This returns the frame (enclosing coordinates) of the view.

**Prepare**

```cpp
virtual void Prepare();
```

This prepares the view for drawing. This involves setting the port, clipping, and origin. Call this routine if you are drawing at an unusual time (like within your Dawdle routine).

**CWindow**

This is a class that displays a standard Macintosh window.

**sizeRect**

```cpp
Rect sizeRect;
```

The top left coordinates specify the minimum size of the window. The bottom right coordinates specify the maximum size.

**ChangeSize**

```cpp
virtual void ChangeSize(short width, short height);
```

Use this to change the window size to width wide, and height high.
**CWindow**

*CWindow*(short *WINDid*, Boolean *aFloating*, *CDirector* *aSupervisor*);

This is the constructor. *WINDid* is the resource ID of the 'WIND' resource, which contains information about the window. If the window should be a floating window, *aFloating* is true. See *CPane* for information about *aSupervisor*.

**GetTitle**

virtual void GetTitle(Str255 *theTitle);

This returns the title of the window in *theTitle*.

**Move**

virtual void Move(short *hGlobal*, short *vGlobal*);

This moves the window to the position *hGlobal*, *vGlobal* (in global coordinates).

**Select**

virtual void Select();

This brings the window to the front and makes it active.

**SetSizeRect**

virtual void SetSizeRect(Rect *aSizeRect);

This sets the new value of the data member *sizeRect* equal to the parameter *aSizeRect*.

**SetTitle**

virtual void setTitle(Str255 *theTitle);

This sets the title of the window to *theTitle*. 
Global Variables

**gDecorator**

CDecorator *gDecorator;

This contains the only instance of CDecorator. It is created when the TCL application initializes. For more information, see CDecorator.

**gBartender**

CBartender *gBartender;

This contains the only instance of CBartender. It is created when the TCL application initializes. For more information, see CBartender.

**gClipboard**

CClipboard *gClipboard;

This contains the only instance of CClipboard. It is created when the TCL application initializes. For more information, see CClipboard.

**gSignature**

OSType gSignature;

This contains the application signature. Initialize this variable in the initialization method of your application or in your override of TApplication::SetupFileParameters.

**gSystem**

tSystem gSystem;

This contains a number of fields describing the current system. One field is the Boolean hasColorQD.
Global Routines

ConcatPStrings
void ConcatPStrings(Str255 first, ConstStr255Param second);

This appends the second string to the first string.

CopyPString
void CopyPString(ConstStr255Param srcString, Str255 destString);

This copies srcString into destString.

Failure
pascal void Failure(short error, long message);

This causes an exception that aborts the routine in progress, and all other routines in the call chain. It calls exception handlers along the way. Normally, it aborts back to the main event loop, which puts up an alert based on the given error and message. It then continues, getting the next event.

FailNIL
pascal void FailNIL(void *p);

This causes a failure if p is 0.

FailOSErr
pascal void FailOSErr(short error);

This causes a failure if error is not noErr.

FailMemError
pascal void FailMemError();

This causes a failure if MemError() is not noErr.
**LongToQDRect**

pascal void LongToQDRect(LongRect *srcRect, Rect *destRect);

This converts from a 32-bit coordinate rectangle to a 16-bit coordinate rectangle by truncating.

**NewHandleCanFail**

void *NewHandleCanFail(long size);

This allocates a handle that is size bytes big. If there isn’t enough memory to allocate the handle without breaching the critical reserve, it returns 0.

**PositionDialog**

void PositionDialog(ResType theType, short theID);

This centers the dialog or alert specified with theType and theID according to Human Interface guidelines. The guideline is to be centered horizontally, and centered in the top third of the screen vertically. You call PositionDialog before displaying an alert or dialog.

**ResizeHandleCanFail**

void ResizeHandleCanFail(Handle theHandle, long newSize);

This resizes theHandle to be of size newSize. If there isn’t enough memory to resize without breaching the critical memory reserve, the routine doesn’t succeed. Make sure to call either MemError or FailMemError after calling this routine.

**SetAllocation**

Boolean SetAllocation(Boolean canFail);

If canFail is true, memory allocations are not allowed to use the critical reserve. If canFail is false, memory allocations are allowed to use the critical reserve. SetAllocation returns the old setting. Use this routine to allocate memory without breaching the critical reserve. Make sure to set it back to its original value.
Macros

**TCLForgetObject**

Call with:

TCLForgetObject(anObject);

This deletes anObject if it is not 0. It then sets anObject to 0.

**Exception Handling**

Use CATCH, TRY, and ENDTTRY in the following way:

TRY {
    code which could cause an exception
}
CATCH {
    code to clean up if an exception occurs
}
ENDTRY

TRY installs an exception handler for the duration of the block after it. If an exception occurs during that block, control is transferred to the CATCH block. The CATCH block should do whatever cleanup is necessary. Control passes to the next installed exception handler at the end of the CATCH block.

If no exception occurs, the CATCH block will not be executed.

If you don’t want exceptions to continue propagating, but want to continue in the function after the CATCH block is executed, place the NO_PROPAGATE line within the CATCH block.

**TCL_AUTO_DESTRUCT_OBJECT**

Derive with:

class MyClass TCL_AUTO_DESTRUCT_OBJECT {

Then, destructors for MyClass objects that are allocated on the stack will automatically be called if an exception occurs.
Constructor/Destructor Macros

Add `TCL_END_CONSTRUCTOR` to the end of each of your constructors. Add `TCL_START_DESTRUCTOR` to the beginning of each of your destructors. These macros ensure that if an error occurs while in the midst of constructing a complex object (which could require many constructors to be executed), the correct destructors will be called. They also make sure that the destructors of stack-based objects are correctly called in case of an exception.

Creating Objects by Name

The TCL provides the facility to create objects using the name of the class (`new_by_name`). This creation of objects is used for views you create in Visual Architect; the TCL reads a resource containing the names of view classes to create. In order for this facility to work, you must use some macros.

**TCL_DECLARE_CLASS**

Add this macro to your class declaration. It declares some methods in your class that are necessary for creating objects by name.

**TCL_DEFINE_CLASS**

There are a number of different TCL_DEFINE_CLASS macros. They differ in the number of base classes for a class, and whether they provide only run-time typing, or whether they also provide the new_by_name capability:

```c
// provide run-time typing
TCL_DEFINE_CLASS_M0(MyClass)
TCL_DEFINE_CLASS_M1(MyClass, BaseClassOfMyClass)
TCL_DEFINE_CLASS_M2(MyClass, FirstBaseClass, SecondBaseClass)
```

```c
// also provides new_by_name
TCL_DEFINE_CLASS_D0(MyClass)
TCL_DEFINE_CLASS_D1(MyClass, BaseClassOfMyClass)
TCL_DEFINE_CLASS_D2(MyClass, FirstBaseClass, SecondBaseClass)
```
This macro forces a reference to a class to ensure that it is correctly initialized and not linked away.

This macro should not actually be executed, but should be part of the code in your application.
This appendix lists the source code used for each project by chapter.

Chapter 2, MultipleFiles

Average.cp

```c
#include "Average.h"
#include <SANE.h>

long double Average(double *values, int numValues)
{
    long double total = 0.0;
    int i;
```
for (i = 0; i < numValues; i++)
    total += values[i];
if (numValues > 0)
    return total / numValues;
else
    return 0.0;
}

Main.cp
#include <stdio.h>
#include "Factorial.h"
#include "Reverse.h"
#include "Average.h"

main()
{
    int i;
    char s[] = "Neil Rhodes";
    char palindrome[] = "Madam, I'm Adam";
    double values[] = {3.1, 4.1, 5.9, 15.6, 16.3, 15.7};

    for (i = 0; i < 14; I++)
        printf("Factorial(%i) %ld\n", i, Factorial(i));

    printf("Reverse("%s") = ", s);
    Reverse(s);
    printf("%s\n", s);

    printf("Reverse("%s") = ", palindrome);
    Reverse(palindrome);
    printf("%s\n", palindrome);

    printf("Average() = %lf\n", Average(values, sizeof(values) / sizeof(values[0])));

    return 0;
}
**Factorial.cp**

#include "Factorial.h"

unsigned long Factorial(unsigned long n)
{
    unsigned long result = 1;
    unsigned long i;

    for (i = 2; i <= n; i++)
        result *= i;
    return result;
}

**Reverse.cp**

#include "Reverse.h"
#include <string.h>

static void Swap(char *c1, char *c2)
{
    char tmp;

    tmp = *c1;
    *c1 = *c2;
    *c2 = tmp;
}

void Reverse(char *s)
{
    int length = strlen(s);
    int midPoint = length / 2;
    int i;

    for (i = 0; i < midPoint; i++)
        Swap(&s[i], &s[length - i - 1]);
}
Average.h
#pragma once

long double Average(double *values, int numValues);

Factorial.h
#pragma once

unsigned long Factorial(unsigned long n);

Reverse.h
#pragma once

void Reverse(char *s);

Chapter 3, Errors

Average.cp
#include "Average.h"
#include <SANE.h>

long double Average(double *values, int numValues)
{
    long double total = 0.0;
    int i;

    for (i == 0; i < numValues; i++)
        total += values[i];
    if (numValues > 0)
        return total / numValues;
    else
        return;
}
Main.cp

#include <stdio.h>
#include "Factorial.h"
#include "Reverse.h"
#include "Average.h"

main()
{
    int i;
    char s[] = "Neil Rhodes."
    char palindrome[] = "Madam, I'm Adam."
    double values[] = {3.1, 4.1, 5.9, 15.6, 16.3, 15.7};

    printf("Reverse("s") = ", s);
    Reverse(s);
    printf("\n", s);

    printf("Reverse("palindrome") = ", palindrome);
    Reverse(palindrome);
    printf("\n", palindrome);

    printf("Average() = %lf\n", Average(values, sizeof(values) / sizeof(values[0])));

    return 0;
}

Reverse.cp

#include "Reverse.h"
#include <string.h>

static void Swap(char *c1, char *c2)
{
    char tmp;

    tmp = c1;
    c1 = c2;
    c2 = tmp;
}
void Reverse(char *s)
{
    int length = strlen(s);
    int midPoint = length div 2;
    int i;

    for (i = 0; i < midPoint; i++)
        Swap(s[i], &s[length - i - 1]);
}

Average.h
#pragma once

long double Average(double *values, int numValues);

Reverse.h
#pragma once

void Reverse(char *s);

Chapter 4, BinaryTree

Main.cp
struct BinaryTree {
    int value;
    BinaryTree *left;
    BinaryTree *right;

    BinaryTree(int v) {value = v; left = right = NULL;}
};

Boolean CheckBinaryTree(BinaryTree *aTree)
{
    if (aTree == NULL)
        return true;
if (aTree->left != NULL) {
    if (aTree->left->value > aTree->value)
        return false;
    if (!CheckBinaryTree(aTree->left))
        return false;
}
if (aTree->right != NULL) {
    if (aTree->value > aTree->right->value)
        return false;
    if (!CheckBinaryTree(aTree->right))
        return false;
}
return TRUE;

BinaryTree *NewNode(int value)
{
    return new BinaryTree(value);
}

main()
{
    /* construct binary tree by hand. */
    /* Should really write a function to do it */
    BinaryTree *root = NULL;
    BinaryTree *node;

    node = NewNode(5);
    root = node;

    node = NewNode(3);
    root->left = node;

    node = NewNode(4);
    root->left->left = node; /* should be root->left->right */

    node = NewNode(0);
    root->right = node;

    return 0;
}
Chapter 5, Alphabet

**Main.cp**

```c
#include <stdio.h>
#include <Types.h>
#include <OSUtils.h>
#include <string.h>
#include <ToolUtils.h>
#include "ResourceDefinitions.h"

#define kMaxInputLine 512

/* prompt for character input */
char gPrompt[256];

/* if they type an invalid character */
char gBadCharacter[256];

/* output if they type a valid character. This is used as the first argument to printf */
char gCharIsForWordTemplate[256];

Boolean FindIndexOfCharacter(char c, int *ip)
{
    const kCaseSensitive = true;
    const kDiacriticalSensitive = true;
    Str255 entry;
    int index = 0;
    Str255 charAsString;

    charAsString[0] = t;
    charAsString[1] = c;

    do {
        GetIndString(entry, kCharsResourceID, ++index);
    } while (StrLength(entry) != 0 && 
    !EqualString(charAsString, entry, 
    !kCaseSensitive, !kDiacriticalSensitive));

    *ip = index;
    return StrLength(entry) != 0;
}
```

Symantec C++ Programming for the Macintosh, 2nd Edition
/* returns false if no more input */

Boolean HandleOneInteraction()
{
    int c;
    char inputLine[kMaxInputLine];

    printf("\n%s", gPrompt);
    if (fgets(inputLine, sizeof(inputLine), stdin) == NULL)
        return false;
    else {
        int i;

        if (!FindIndexOfCharacter(*inputLine, &i))
            printf("%s\n", gBadCharacter);
        else {
            Str255 word;

            GetIndString(word, kWordsResourceId, i);
            printf(gCharIsForWordTemplate, *inputLine, p2cstr(word));
            putchar('\n');
        }
        return true;
    }
}

void InitializeStrings()
{
    Str255 s;

    GetIndString(s, kMiscStringsID, kPromptItem);
    strcpy(gPrompt, p2cstr(s));

    GetIndString(s, kMiscStringsID, kBadCharacterItem);
    strcpy(gBadCharacter, p2cstr(s));

    GetIndString(s, kMiscStringsID, kTemplateItem);
    strcpy(gCharIsForWordTemplate, p2cstr(s));
}

main()
{
InitializeStrings();

while (HandleOneInteraction())
{
    /* do nothing */
}

return 0;

---

**ResourceDefinitions.h**

```c
#define kWordsResourceId 1024
#define kCharsResourceId 1025

#define kMiscStringsID 1026
#define kPromptItem 1
#define kBadCharacterItem 2
#define kTemplateItem 3
```

**Alphabet.r**

```c
#include "Types.r"
#include "ResourceDefinitions.h"

resource 'STR#'(kCharsResourceId, "Characters", purgeable)
{
    {
        "a", "b", "c", "d", "e", "f", "g",
        "h", "i", "j", "k", "l", "m", "n",
        "o", "p", "q", "r", "s", "t", "u",
        "v", "w", "x", "y", "z"
    }
};

resource 'STR#'(kWordsResourceId, "Words", purgeable)
{
    {
        "Alexander", "Barnacle", "Cornelius",
        "Daddy", "Elephant", "Fox",
        "Gazelle", "Horse", "Iguana",
```
Chapter 7, Project One

Foo.cp

```c
#include <QuickDraw.h>

void Foo()
{
    Rect r = {10, 20, 50, 100};
    FrameRect(&r);
}
```

Chapter 7, Project Two

Main.cp

```c
#include "FooBar++"
#include <stdio.h>
```
void main()
{
    printf("%d %d\n", kFoo, kBar);

    return 0;
}

FooBar.cp
#include "Foo.h"
#include "Bar.h"

Foo.h
#include "Bar.h"
#define kFoo 10

Bar.h
#define kBar 10

Chapter 16, RandomRectangle

CRandomRectanglePane.cp
// Copyright © 1993-94, Neil Rhodes and Julie McKeenan. All rights reserved.
#include <Global.h>
#include "CRandomRectanglePane.h"

CRandomRectanglePane::CRandomRectanglePane(CView *anEnclosure, 
    CBureaucrat *aSupervisor, 
    short aWidth, short aHeight, 
    short aHEncl, short aVEncl, 
    SizingOption aHSizing, SizingOption aVSizing) : 
    CPane(anEnclosure, aSupervisor, aWidth, aHeight, 
        aHEncl, aVEncl, aHSizing, aVSizing)
void CRandomRectanglePane::DrawRectangleInRandomColor(const Rect *r)
{
    RGBColor aColor;
    PixPatHandle myPixPat;

    aColor.red = fRandom.Value();
    fRandom.Advance();
    aColor.blue = fRandom.Value();
    fRandom.Advance();
    aColor.green = fRandom.Value();
    fRandom.Advance();

    myPixPat = NewPixPat();
    MakeRGBPat(myPixPat, &aColor);
    FillCRect(r, myPixPat);
    DisposePixPat(myPixPat);
}

void CRandomRectanglePane::DrawRectangleInRandomPattern(const Rect *r)
{
    Pattern aPattern;

    GetIndPattern(&aPattern, sysPatListID, fRandom.ValueInRange(1, 31));
    fRandom.Advance();
    FillRect(r, &aPattern);
}

void CRandomRectanglePane::Dawdle(long *maxSleep)
{
    Rect r;
    Point p1;
    Point p2;
    LongRect frameSize;

    GetFrame(&frameSize);

    p1.h = fRandom.ValueInRange(frameSize.left, frameSize.right);
    fRandom.Advance();
    p1.v = fRandom.ValueInRange(frameSize.top, frameSize.bottom);
Random. Advance();
p2.h = Random. ValueInRange(frameSize. left, frameSize. right); Random. Advance();
p2.v = Random. ValueInRange(frameSize. top, frameSize. bottom); Random. Advance();
::Pt2Rect(p1, p2, &r);
Prepare(); // prepare coordinate system for drawing

if (gSystem. hasColorQD)
    DrawRectangleInRandomColor(&r);
else
    DrawRectangleInRandomPattern(&r);

*maxSleep = 20; // one rectangle every 1/3 second
}

CRectangleApp.h

// Copyright © 1993-94, Neil Rhodes and Julie McKeehan. All rights reserved.

/*****
* CRectangleApp.h
* *
*****/

#include "CRectangleApp.h"
#include "CRandomRectanglePane.h"
#include <CDirector.h>
#include <CWindow.h>

const short kExtraMasters = 1;
const Size kRainyDayFund = 45000;
const Size kCriticalBalance = 40000;
const Size kToolboxBalance = 20000;

CRectangleApp::CRectangleApp() :
    CApplication(kExtraMasters, kRainyDayFund, kCriticalBalance, kToolboxBalance)
void CRectangleApp::StartUpAction(short numPreloads)
{
    CApplication::StartUpAction(numPreloads);

    const short kWINDResourceID = 500;
    const Boolean kWindowFloats = TRUE;
    const Boolean kFitHorizontal = TRUE;
    const Boolean kFitVertical = TRUE;

    CDirector *aDirector = new CDirector(this);

    CWindow *aWindow = new CWindow(kWINDResourceID, !kWindowFloats, aDirector);
    aDirector->itsWindow = aWindow;

    CRandomRectanglePane *thePane =
        new CRandomRectanglePane(aWindow, aDirector, 0, 0, 0, 0,
            sizeELASTIC, sizeELASTIC);
    thePane->FitToEnclosure(kFitHorizontal, kFitVertical);
    aDirector->itsGopher = thePane;
    thePane->BecomeGopher(TRUE);  // so it starts dawdling

Rectangle.cp

// Copyright © 1993, Neil Rhodes and Julie McKeehan. All rights reserved.

#include "CRectangleApp.h"

void main()
{
    CRectangleApp *rectangleApp;

    rectangleApp = new CRectangleApp;
    rectangleApp->Run();
    rectangleApp->Exit();
}
// Copyright © 1993, Neil Rhodes and Julie McKeehan. All rights reserved.

#include "TRandom.h"
#include <Events.h>

TRandom::TRandom()
{
    fValue = ::TickCount();
}

TRandom::TRandom(unsigned long seed)
{
    fValue = seed;
}

void TRandom::Advance()
{
    fValue = fValue * 12345 + 6789;
}

unsigned long TRandom::Value()
{
    return fValue;
}

unsigned long TRandom::ValueInRange(unsigned long low, unsigned long high)
{
    return (Value() % (high - low + 1)) + low;
}

CRandomRectanglePane.h

#pragma once

#include <CPane.h>
#include "TRandom.h"

class CRandomRectanglePane: public CPane {
public:
virtual void CRandomRectanglePane(CView *anEnclosure,
   CBureaucrat *aSupervisor,
   short aWidth, short aHeight,
   short aHEncl, short aVEncl,
   SizingOption aHSizing, SizingOption aVSizing);

virtual void Dawdle(long *maxSleep);

protected:
   virtual void DrawRectangleInRandomColor(const Rect *r);
   virtual void DrawRectangleInRandomPattern(const Rect *r);

private:
   TRandom fRandom;
};

**CRectangleApp.h**

/*****
* CRectangleApp.h
*
*   Application class for a typical application.
* 
* *****/

#pragma once

#include <CApplication.h>

class CRectangleApp: public CApplication {
public:
   CRectangleApp();
   virtual void StartUpAction(short numPreloads);
};

**TRandom.h**

#pragma once

class TRandom {
public:
   TRandom();
   TRandom(unsigned long seed);
void Advance();
unsigned long Value();
unsigned long ValueInRange(unsigned long low, unsigned long high);

private:
    unsigned long fValue;
};
{ "$0000 0000 0000 0000 003F FE00 0020 0200"
$s"0020 0200 03FF F200 0220 1200 0220 1200"
$s"0220 1200 0220 1200 0FFE 1200 0A22 1200"
$s"0A22 1200 0A22 1200 0BFF F200 0822 0200"
$s"083F FE00 0002 0000 0FFE",
$s"0000 0000 0000 0000 003F FE00 003F FE00"
$s"003F FE00 03FF FE00 03FF FE00 03FF FE00"
$s"03FF FE00 03FF FE00 0FF FE00 0FF FE00"
$s"0FF FE00 0FF FE00 0FF FE00 0FF FE00"
$s"0FF FE00 0FFE 0000 0FFE"
}

resource 'ics#' (128) {

{s"01FE 0102 1FF2 1112 FFD2 9152 9152 91FE"
$s"9050 9FF0 8040 8040 FFC0",
$s"01FE 01FE 1FFE 1FFE FFFE FFFE FFFE FFFE"
$s"FFF0 FFF0 FFC0 FFC0 FFC0"
}

resource 'vers' (1) {
  0x1,
  0x1,
  release,
  0x0,
  verUS,
  "RandomRectangles 1.1",
  "1.1, from Symantec C++ Programming for Macintosh."
}

resource 'vers' (2) {
  0x1,
  0x1,
  release,
  0x0,
  verUS,
  "RandomRectangles 1.1",
  "1.1 ©1993-94, Rhodes and McKeehan."
}
resource 'MENU' (2, "File", preload) {
    2,
    textMenuProc,
    0x7FFFFFFE,
    enabled,
    "File",
    {
        "Close#4", noIcon, "W", noMark, plain,
        "Quit#1", noIcon, "Q", noMark, plain
    }
};

resource 'MBAR' (1, preload) {
    { 1, 2, 3
    }
};

resource 'SICN' (200, "TCL sicns", purgeable, preload) {
    {
        "$FFFF 8001 8001 9FC1 9041 907D 9045 9045"
        "$9045 9FC5 8405 8405 8405 87FD 8001 FFFF",
        "$07FF 03FE 01FC 00F8 0070 0020"
    }
};

resource 'WIND' (200, "Clipboard", purgeable, preload) {
    {130, 158, 238, 444},
    zoomDocProc,
    invisible,
    goAway,
    0x0,
    "Clipboard"
};

resource 'WIND' (500, "Rectangle window", purgeable, preload) {
    {68, 44, 254, 286},
    zoomDocProc,
    visible,
    goAway,
    0x0,
    "Untitled"
};
resource 'MENU' (3, "Edit", preload) {
    3,
textMenuProc,
0x7FFFFFFF80,
enabled,
"Edit",
{
    "Undo#16", noIcon, "Z", noMark, plain,
    ",", noIcon, noKey, noMark, plain,
    "Cut#18", noIcon, "X", noMark, plain,
    "Copy#19", noIcon, "C", noMark, plain,
    "Paste#20", noIcon, "V", noMark, plain,
    "Clear#21", noIcon, noKey, noMark, plain,
    ",", noIcon, noKey, noMark, plain,
    "Show Clipboard#22", noIcon, noKey, noMark, plain
}
};

resource 'MENU' (1, "Apple", preload) {
    1,
textMenuProc,
0x7FFFFFFFD,
enabled,
apple,
{
    "About RandomRectangles_#256", noIcon, noKey, noMark, plain,
    ",", noIcon, noKey, noMark, plain
}
};

resource 'CNTL' (300, "Scroll Bar", preload) {
    {0, 0, 0, 0},
    0,
visible,
    0,
    0,
scrollBarProc,
    0,
    ""
Chapter 17, TextTyper

CTextApp.cp

// Copyright © 1993, Neil Rhodes and Julie McKeenan. All rights reserved.

#include "CTextApp.h"
#include "CTextDoc.h"
#include <Global.h>

extern OSTYPE gSignature;                     // The application's signature
const short kExtraMasters = 1;
const Size kRainyDayFund = 45000;
const Size kCriticalBalance = 40000;
const Size kToolboxBalance = 20000;

CTextApp::CTextApp() :
    CApplication(kExtraMasters, kRainyDayFund,
                  kCriticalBalance, kToolboxBalance)
{
    TCL_END_CONSTRUCTOR
}

void CTextApp::CreateDocument()
{
    CTextDoc *theDocument = new CTextDoc;

    TRY
    {
        theDocument->NewFile();
    }
    CATCH
    {
        TCLForgetObject(theDocument);
    }
    ENDTRY
}

void CTextApp::OpenDocument(SFReply *macReply)
{

CTextDoc *theDocument = new CTextDoc;

TRY
{
    theDocument->OpenFile(macReply);
}
CATCH
{
    TCForgetObject(theDocument);
}
ENDTRY;

void CTextApp::SetUpFileParameters()
{
    CAapplication::SetUpFileParameters();

    sfNumTypes = 1;
    sfFileTypes[0] = 'TEXT';
    gSignature = 'C++2';
}

/*CTextDoc.cp

// Copyright © 1993, Neil Rhodes and Julie McKeehan. All rights reserved.

#include "CTextDoc.h"
#include <CEDitText.h>
#include <CScrollPane.h>
#include <CDesktop.h>
#include <CWindow.h>
#include <CDataFile.h>
#include <CDecorator.h>
#include <TBUUtilities.h>

extern Cdesktop *gDesktop;
extern OSType gSignature;
extern CDecorator *gDecorator;

void CTextDoc::NewFile()
{
*/
BuildWindow();

Str255 wTitle;
short wCount;
Str255 wNumber;

itsWindow->GetTitle(wTitle);
wCount = gDecorator->GetWCount();
::NumToString(wCount, wNumber);
::ConcatPStrings(wTitle, "\p-" );
::ConcatPStrings(wTitle, wNumber);
itsWindow->SetTitle(wTitle);

itsWindow->Select();
}

void CTextDoc::BuildWindow()
{
    const short kWINDResourceID = 500;
    const Boolean kWindowFloats = TRUE;
    const Boolean kFitHorizontal = TRUE;
    const Boolean kFitVertical = TRUE;
    const Boolean kHasHorizontalScrollbar = TRUE;
    const Boolean kHasVerticalScrollbar = TRUE;
    const Boolean kHasGrowBox = TRUE;
    const short kEditTextWidth = 72 * 6;
    const Boolean kRedraw = TRUE;

    CWindow *aWindow = new CWindow(kWINDResourceID, !kWindowFloats, this);
    itsWindow = aWindow;
    CScrollPane *theScrollPane = new CScrollPane(aWindow, this, 10, 10, 0, 0,
        sizELASTIC, sizELASTIC,
        kHasHorizontalScrollbar, kHasVerticalScrollbar, kHasGrowBox);
    theScrollPane->FitToEnclFrame(kFitHorizontal, kFitVertical);

    CEditText *thePane = new CEditText(theScrollPane, this, 1, 1, 0, 0,
        sizELASTIC, sizELASTIC, kEditTextWidth);
    itsGopher = thePane;
    itsMainPane = thePane;
    fEditText = thePane;
    theScrollPane->InstallPanorama(thePane);
    thePane->FitToEnclosure(kFitHorizontal, kFitVertical);
Rect margin;
::SetRect(&margin, 2, 2, -2, -2);
thePane->ChangeSize(&margin, 1kRedraw);

gDecorator->PlaceNewWindow(itsWindow);
}

void CTextDoc::OpenFile(SFReply *macReply)
{
    CDataFile *theDataFile;
    Handle theData;
    Str63 theName;
    OSErr theError;
    const long kMaxFileSize = 32000;

    theDataFile = new CDataFile;
    itsFile = theDataFile;

    theDataFile->SFSpecify(macReply);

    theDataFile->Open(fsRdPerm);

    TRY {
        if (theDataFile->GetLength() > kMaxFileSize)
            ::Failure(paramErr, excExceedTELimit);
        theData = theDataFile->ReadAll();
    } CATCH {
        theDataFile->Close();
    }
    ENDTRY;

    TRY {
        theDataFile->Close();
        BuildWindow();
        fEditText->SetTextHandle(theData);
    } CATCH {
        ::DisposHandle(theData);
        throw_same_();
    }
ENDTRY;
::DisposHandle(theData);

itsFile->GetName(theName);
itsWindow->SetTitle(theName);
itsWindow->Select();

Boolean CTextDoc::DoSave()
{
Handle theData;

if (itsFile == NULL)
    return(DoSaveFileAs());
else {
    CDataFile *theDataFile = (CDataFile *) itsFile;

    theData = fEditText->GetTextHandle();
    theDataFile->Open(fsWrPerm);
    TRY {
        theDataFile->WriteAll(theData);
    } CATCH {
        theDataFile->Close();
    }
    ENDTry;
    theDataFile->Close();
dirty = FALSE;
return(TRUE);
}

Boolean CTextDoc::DoSaveAs(SFReply *macSFReply)
{
    if (itsFile != NULL)
        TCL ForgetObject(itsFile);

    CDataFile *theDataFile = new CDataFile;
    itsFile = theDataFile;
    theDataFile->SFSpecify(macSFReply);
    if (theDataFile->ExistsOnDisk())
        theDataFile->ThrowOut();
    theDataFile->CreateNew(gSignature, 'TEXT');
itsWindow->SetTitle(macSFReply->fName);

return(DoSave());
}

Text.cpp
#include "CTextApp.h"

void main()
{
    CTextApp *app;

    app = new CTextApp;
    app->Run();
    app->Exit();
}

CTextApp.h
#pragma once
#include <CApplication.h>

class CTextApp: public CApplication {
public:

    CTextApp();

    virtual void CreateDocument();
    virtual void OpenDocument(SFReply *macReply);
    virtual void SetUpFileParameters();
};

CTextDoc.h
#pragma once

#include <CDocument.h>
#include <CEditText.h>
class CTextDoc: public CDocument {
    public:
        virtual void NewFile();
        virtual void OpenFile(SFReply *macReply);
        virtual Boolean DoSave();
        virtual Boolean DoSaveAs(SFReply *macSFReply);
    protected:
        virtual void BuildWindow();
    private:
        CEditText *fEditText;
};
data 'C++2' (0, "Owner resource") {
    "$00"
};

data 'FREF' (128) {
    "$4150 504C 0000 84"
};

resource 'ics#' (128) {
    { /* array: 2 elements */
        /* [1] */
        "$0000 0000 0000 3FF8 0100 0100 0100 0100"
        "$0100 0100 0100 0100 01"
        /* [2] */
        "$0000 0000 7FFC 7FFC 7FFC 0380 0380 0380"
        "$0380 0380 0380 0380 0380 0380 0380"
    }
};

resource 'ICN#' (128) {
    { /* array: 2 elements */
        /* [1] */
        "$0000 0000 0000 0000 0000 0000 0000 0000"
        "$0FFF FE00 0FFF FE00 000E 0000 000E 0000"
        "$000E 0000 000E 0000 000E 0000 000E 0000"
        "$000E 0000 000E 0000 000E 0000 000E 0000"
        "$000E 0000 000E",
        /* [2] */
        "$0000 0000 0000 0000 0000 0000 1FFF FF00"
        "$1FFF FF00 1FFF FF00 001F 0000"
        "$001F 0000 001F 0000 001F 0000 001F 0000"
        "$001F 0000 001F 0000 001F 0000 001F 0000"
        "$001F 0000 001F"
    }
};

resource 'MENU' (2, "File", preload) {
    2,
    textMenuProc,
    0x7FFFFFFE,
    enabled,
"File",
{
   "New#2", noIcon, "N", noMark, plain,
   "Open#3", noIcon, "O", noMark, plain,
   ".", noIcon, noKey, noMark, plain,
   "Close#4", noIcon, "W", noMark, plain,
   "Save#5", noIcon, "S", noMark, plain,
   "Save As_#6", noIcon, noKey, noMark, plain,
   ".", noIcon, noKey, noMark, plain,
   "Page Setup_#8", noIcon, noKey, noMark, plain,
   "Print_9", noIcon, "P", noMark, plain,
   ".", noIcon, noKey, noMark, plain,
   "Quit#1", noIcon, "Q", noMark, plain
}
};

resource 'MBAR' (1, preload) {
   { 1, 2, 3
}
};

resource 'SICN' (200, "TCL sicns", purgeable, preload) {
   {
      "$FFFF 8001 8001 9FC1 9041 907D 9045 9045"
      "$9045 9FC5 8405 8405 8405 87FD 8001 FFFF",
      "$07FF 03FE 01FC 00F8 0020"
   }
};

resource 'WIND' (200, "Clipboard", purgeable, preload) {
   {130, 158, 238, 444},
   zoomDocProc,
   invisible,
   goAway,
   0x0,
   "Clipboard"
};

resource 'MENU' (3, "Edit", preload) {
   3,
   textMenuProc,
   0x7FFFFFF80,
   enabled,

400
Symantec C++ Programming for the Macintosh, 2nd Edition
"Edit",
{
    "Undo#16", noIcon, "Z", noMark, plain,
    ".", noIcon, noKey, noMark, plain,
    "Cut#18", noIcon, "X", noMark, plain,
    "Copy#19", noIcon, "C", noMark, plain,
    "Paste#20", noIcon, "V", noMark, plain,
    "Clear#21", noIcon, noKey, noMark, plain,
    "Select All#23", noIcon, "A", noMark, plain,
    ".", noIcon, noKey, noMark, plain,
    "Show Clipboard#22", noIcon, noKey, noMark, plain
}
};

resource 'MENU' (1, "Apple", preload) {
    1,
    textMenuProc,
    0xFFFFFFFDFD,
    enabled,
    apple,
    {
        "About TextTyper_#256", noIcon, noKey, noMark, plain,
        ".", noIcon, noKey, noMark, plain
    }
};

resource 'CNTL' (300, "Scroll Bar", preload) {
    {0, 0, 0, 0},
    0,
    visible,
    0,
    0,
    scrollBarProc,
    0,
    ""
};

resource 'MENU' (10, "Font") {
    10,
    textMenuProc,
    allEnabled,
    enabled,
    "Font",
    "Font"}
Chapter 18, PicturePeeker

CAboutBoxApplication.cp

// Copyright © 1993, Neil Rhodes and Julie McKeehan. All rights reserved.

#include "CAboutBoxApplication.h"
#include <SegLoad.h>
CAboutBoxApplication::CAboutBoxApplication(short extraMasters,
    Size aRainyDayFund, Size aCriticalBalance, Size aToolboxBalance) :
    CApplication(extraMasters,aRainyDayFund, aCriticalBalance,
        aToolboxBalance)
{
    TCL_END_CONSTRUCTOR
}

void CAboutBoxApplication::DoCommand(long theCommand)
{
    const short kAboutBoxResourceID = 1024;

    switch (theCommand) {
    case cmdAbout:
        Str255 applicationName;
        short appRefNum;
        Handle h;

        ::PositionDialog('ALRT', kAboutBoxResourceID);
        ::GetAppParms(applicationName, &appRefNum, &h);
        ::DisposHandle(h);
        ::ParamText(applicationName,
            NULL, NULL, NULL);
        Alert(kAboutBoxResourceID, NULL);
        break;
    default:
        CApplication::DoCommand(theCommand);
        break;
    }
}

CCutCopyPastePicture.cp

// Copyright © 1993, Neil Rhodes and Julie McKeenan. All rights reserved.

#include "CCutCopyPastePicture.h"
#include <CBartender.h>
#include <CClipboard.h>
#include "CPictureClipboardTask.h"

extern CBartender *gBartender;
extern CClipboard *gClipboard;
CCutCopyPastePicture::CCutCopyPastePicture(CView *anEnclosure,
   CBureaucrat *aSupervisor,
   short aWidth, short aHeight, short aHEncl, short aVEncl,
   SizingOption aHSizing, SizingOption aVSizing) :
   CPicture(anEnclosure, aSupervisor, aWidth, aHeight,
   aHEncl, aVEncl, aHSizing, aVSizing)
{
   TCL_END_CONSTRUCTOR
}

void CCutCopyPastePicture::UpdateMenus()
{
    CPicture::UpdateMenus();

    if (GetMacPicture() != NULL) {
        gBartender->EnableCmd(cmdCopy);
        gBartender->EnableCmd(cmdCut);
        gBartender->EnableCmd(cmdClear);
    }

    if (gClipboard->GetData('PICT', NULL))
        gBartender->EnableCmd(cmdPaste);
}

void CCutCopyPastePicture::DoCommand(long cmd)
{
    switch (cmd) {
    case cmdCopy:
    case cmdCut:
    case cmdClear:
    case cmdPaste:
        CPictureClipboardTask *clipTask = new CPictureClipboardTask(this, cmd);
        itsLastTask = clipTask;
        if (cmd == cmdCopy)
            itsSupervisor->NotifyClean(clipTask);
        else
            itsSupervisor->Notify(clipTask);
        clipTask->Do();
        break;

    default:
        CPicture::DoCommand(cmd);
    }
}
#include "CPictureApp.h"
#include "CPictureDoc.h"
#include "TDeleteOnFailure.h"
#include <Global.h>

const short kExtraMasters = 1;
const Size kRainyDayFund = 45000;
const Size kCriticalBalance = 40000;
const Size kToolboxBalance = 20000;

extern OSType gSignature; // The application's signature

CPictureApp::CPictureApp()
    : CAboutBoxApplication(kExtraMasters, kRainyDayFund,
                            kCriticalBalance, kToolboxBalance)
{
    TCL_END_CONSTRUCTOR
}

void CPictureApp::CreateDocument()
{
    CPictureDoc *theDocument = new CPictureDoc;
    TDeleteOnFailure<CPictureDoc> freer(theDocument);
    theDocument->NewFile();
    freer.Success();
}

void CPictureApp::OpenDocument(SFReply *macReply)
{
    CPictureDoc *theDocument = new CPictureDoc;
void CPictureApp::SetUpFileParameters()
{
    CApplication::SetUpFileParameters();

    sfNumTypes = 1;
    sfFileTypes[0] = 'PICT';
    gSignature = 'C++3';
}

CPictureClipboardTask.cpp
// Copyright © 1993, Neil Rhodes and Julie McKeegan. All rights reserved.

#include "CPictureClipboardTask.h"
#include <CClipboard.h>

extern CClipboard *gClipboard;
const LongPt kZeroZeroPt = {0, 0};

// cmdTyping is first string, cmdCut is next
CPictureClipboardTask::CPictureClipboardTask(CPicture *thePicture,
    long command)
    : CTask(command - cmdCut + 2)
{
    fOldMacPicture = NULL;
    fOldScrap = NULL;
    fPicture = thePicture;
    fCommand = command;
    TCL_END_CONSTRUCTOR
}

CPictureClipboardTask::~CPictureClipboardTask()
{
    TCL_START_DESTRUCTOR
    if (!IsUndone()) {
switch (fCommand) {
    case cmdCut:
        ::DisposHandle((Handle) fOldMacPicture);
        ::DisposHandle(fOldScrap);
        break;
    case cmdClear:
        ::DisposHandle((Handle) fOldMacPicture);
        break;
    case cmdPaste:
        ::DisposHandle((Handle) fOldMacPicture);
        // old scrap is now the picture
        break;
    case cmdCopy:
        // old picture remains the picture
        ::DisposHandle(fOldScrap);
        break;
    }
} else
    if (fOldScrap != NULL)
        ::DisposHandle(fOldScrap);

void CPictureClipboardTask::Do()
{
    if (fCommand != cmdClear) {
        Handle h;

        gClipboard->GetData('PICT', &h);
        fOldScrap = h;
    }
    fOldMacPicture = fPicture->GetMacPicture();
    Redo();
}

void CPictureClipboardTask::Undo()
{
    if (fCommand == cmdCopy || fCommand == cmdCut) {
        gClipboard->EmptyScrap();
        if (fOldScrap != NULL)
            gClipboard->PutData('PICT', fOldScrap);
    }
    if (fCommand != cmdCopy) {

Appendix D: Complete Source Listings
void CPictureClipboardTask::Redo()
{
    if (fCommand == cmdCopy || fCommand == cmdCut) {
        gClipboard->EmptyScrap();
        gClipboard->PutData('PICT', (Handle) fOldMacPicture);
    }
    if (fCommand == cmdCut || fCommand == cmdClear) {
        fPicture->SetPosition((LongPt *) &kZeroZeroPt);
        fPicture->SetMacPicture(NULL);
    }
    if (fCommand == cmdPaste) {
        fPicture->SetPosition((LongPt *) &kZeroZeroPt);
        fPicture->SetMacPicture((PicHandle) fOldScrap);
    }
    if (fCommand != cmdCopy)
        fPicture->Refresh();
    undone = FALSE;
}

CPictureDoc.cp
// Copyright © 1993, Neil Rhodes and Julie McKeefhan. All rights reserved.

#include "CPictureDoc.h"
#include "CCutCopyPastePicture.h"
#include <CScrollPane.h>
#include <CDesktop.h>
#include <CWindow.h>
#include <CDataFile.h>
#include <CDecorator.h>
#include <TBUtilities.h>
#include "TDisposHandleOnFailure.h"
#include "TWithFileOpen.h"
#include "THandleLocker.h"
extern Cdesktop *gDesktop;
extern OSType gSignature;
extern CDecorator *gDecorator;
const long kPICTHeaderLength = 512;

void CPictureDoc::NewFile()
{
    BuildWindow();
    Str255 wTitle;
    short wCount;
    Str255 wNumber;
    itsWindow->GetTitle(wTitle);
    wCount = gDecorator->GetWCount();
    ::NumToString(wCount, wNumber);
    ::ConcatPStrings(wTitle, \"p\");  
    ::ConcatPStrings(wTitle, wNumber);
    itsWindow->SetTitle(wTitle);
    itsWindow->Select();
}

void CPictureDoc::BuildWindow()
{
    const short kWINDResourceID = 500;
    const Boolean kWindowFloats = TRUE;
    const Boolean kFitHorizontal = TRUE;
    const Boolean kFitVertical = TRUE;
    const Boolean kHasHorizontalScrollbar = TRUE;
    const Boolean kHasVerticalScrollbar = TRUE;
    const Boolean kHasGrowBox = TRUE;
    const Boolean kRedraw = TRUE;
    CWindow *aWindow = new CWindow(kWINDResourceID, !kWindowFloats, this);
    itsWindow = aWindow;
    CScrollPane *theScrollPane = new CScrollPane(aWindow, this, 10, 10, 0, 0,
              sizELASTIC, sizELASTIC,
              kHasHorizontalScrollbar, kHasVerticalScrollbar, kHasGrowBox);
    theScrollPane->FitToEnclFrame(kFitHorizontal, kFitVertical);
CCutCopyPastePicture *thePane =
    new CCutCopyPastePicture(theScrollPane, this, 1, 1, 0, 0,
    sizELASTIC, sizELASTIC);
itsGopher = thePane;
itsMainPane = thePane;
fPicture = thePane;
theScrollPane->InstallPanorama(thePane);
thePane->FitToEnclosure(kFitHorizontal, kFitVertical);

gDecorator->PlaceNewWindow(itsWindow);
}

void CPictureDoc::OpenFile(SFReply *macReply)
{
    CDataFile *theDataFile;
    Handle theData;
    Str63 theName;
    OSErr theError;

    theDataFile = new CDataFile;
    itsFile = theDataFile;
    theDataFile->SFSpecify(macReply);

    Handle theMacPicture = ::NewHandleCanFail(0);
    ::FailNIL(theMacPicture);
{
    TWithFileOpen fileOpener(theDataFile, fsRdPerm);
    TDisposHandleOnFailure handleFreer(theMacPicture);

    long thePictureLength = theDataFile->GetLength() - kPICTHeaderLength;
    if (thePictureLength <= 0)
        ::FailOSErr(eofErr);
    theDataFile->SetMark(kPICTHeaderLength, fsFromStart);
    ::ResizeHandleCanFail(theMacPicture, thePictureLength);
    ::FailMemError();
{
    THandleLocker lock(theMacPicture);
    theDataFile->ReadSome(*theMacPicture, thePictureLength);
}
BuildWindow();
   handleFreer.Success();
}
fPicture->SetMacPicture((PicHandle) theMacPicture);

itsFile->GetName_theName);
itsWindow->SetTitle_theName);

itsWindow->Select();
}

Boolean CPictureDoc::DoSave()
{
   Handle theData;

   if (itsFile == NULL)
      return(DoSaveFileAs());
   else {
      CDataFile *theDataFile = (CDataFile *) itsFile;

      Handle theMacPicture = (Handle) fPicture->GetMacPicture();
      
      TWithFileOpen fileOpener(theDataFile, fsWrPerm);
      char zeros[kPICTHeaderLength];

      for (int i = 0; i < kPICTHeaderLength; i++)
         zeros[i] = 0;
      theDataFile->WriteSome(zeros, sizeof(zeros));
      THandleLocker lock(theMacPicture);
      theDataFile->WriteSome(*theMacPicture, ::GetHandleSize(theMacPicture));

      dirty = FALSE;
      return(TRUE);
   }
}

Boolean CPictureDoc::DoSaveAs(SFReply *macSFReply)
{
   if (itsFile != NULL)
      delete itsFile;

   CDataFile *theDataFile = new CDataFile;
   itsFile = theDataFile;
theDataFile->SFSpecify(macSFReply);
if (theDataFile->ExistsOnDisk())
    theDataFile->ThrowOut();
theDataFile->CreateNew(gSignature, 'PICT');

itsWindow->SetTitle(macSFReply->fName);

return(DoSave());

Picture.cp

// Copyright © 1993, Neil Rhodes and Julie McKeehan. All rights reserved.

#include "CPictureApp.h"

void main()
{
    CPictureApp *app;

    app = new CPictureApp;
    app->Run();
    app->Exit();
}

TDisposHandleOnFailure.cp

// Copyright © 1993, Neil Rhodes and Julie McKeehan. All rights reserved.

#include "TDisposHandleOnFailure.h"

TCL_DEFINE_CLASS_M1(TDisposHandleOnFailure, THandleFailure);
TDisposHandleOnFailure::TDisposHandleOnFailure(Handle h)
{
    fHandle = h;
    TCL_END_CONSTRUCTOR
}
TDisposHandleOnFailure::~TDisposHandleOnFailure()
{
    TCL_START_DESTRUCTOR
    if (!fSucceeded && fHandle != NULL)
        ::DisposHandle(fHandle);
}

THandleFailure.cp
// Copyright © 1993-94, Neil Rhodes and Julie McKeehan. All rights reserved.

#ifdef TCL_PCH
#include <TCLHeaders>
#endif

#include "THandleFailure.h"

TCL_DEFINE_CLASS_M0(THandleFailure);

THandleFailure::THandleFailure()
{
    fSucceeded = FALSE;
    TCL_END_CONSTRUCTOR
}

THandleFailure::~THandleFailure()
{
    TCL_START_DESTRUCTOR
}

void THandleFailure::Success()
{
    fSucceeded = true;
}

THandleLocker.cp
// Copyright © 1993, Neil Rhodes and Julie McKeehan. All rights reserved.

#ifdef TCL_PCH
#include <TCLHeaders>
#endif
#include "THandleLocker.h"

TCL_DEFINE_CLASS_M0(THandleLocker);

THandleLocker::THandleLocker(Handle h)
{
    fHandle = h;
    fState = ::HGetState(fHandle);
    ::HLock(h);
}

THandleLocker::~THandleLocker()
{
    ::HSetState(fHandle, fState);
}

TWithFileOpen.cp
// Copyright © 1993-94, Neil Rhodes and Julie McKeehan. All rights reserved.

#ifdef TCL_PCH
#include <TCLHeaders>
#endif

#include "TWithFileOpen.h"

TCL_DEFINE_CLASS_M0(TWithFileOpen);

TWithFileOpen::TWithFileOpen(CFile *theFile, SignedByte permission)
{
    theFile->Open(permission);
    fFile = theFile;
}

TWithFileOpen::~TWithFileOpen()
{
    if (fFile != NULL)
        fFile->Close();
}
CAboutBoxApplication.h

#pragma once
#include <CApplication.h>

class CAboutBoxApplication: public CApplication {
public:

    CAboutBoxApplication(short extraMasters, Size aRainyDayFund,
                          Size aCriticalBalance, Size aToolboxBalance);

    virtual void DoCommand(long theCommand);
};

CCutCopyPastePicture.h

#pragma once
#include <CPicture.h>

class CCutCopyPastePicture: public CPicture {
public:

    CCutCopyPastePicture(CView *anEnclosure, CBureaucrat *aSupervisor,
                          short aWidth, short aHeight, short aHEncl, short aVEncl,
                          SizingOption aHSizing, SizingOption aVSizing);

    virtual void UpdateMenus();
    virtual void DoCommand(long cmd);
};

CPictureApp.h

/*****
 * CPictureApp.h
 *
 * Application class for a typical application.
 *
 *****/

#pragma once

#include "CAboutBoxApplication.h"
class CPictureApp: public CAboutBoxApplication {
    public:
        CPictureApp();

        virtual void CreateDocument();
        virtual void OpenDocument(SFReply *macReply);
        virtual void SetUpFileParameters();
};

CPictureClipboardTask.h
#pragma once
#include <CPicture.h>
#include <CTask.h>

class CPictureClipboardTask: public CTask {
    public:
        CPictureClipboardTask(CPicture *thePicture, long command);

        virtual -CPictureClipboardTask();
        virtual void Do();
        virtual void Undo();
        virtual void Redo();

    private:
        CPicture *fPicture;
        long fCommand;
        Handle fOldScrap;
        PicHandle fOldMacPicture;
};

CPictureDoc.h
#pragma once

#include <CDocument.h>
#include <CPicture.h>

class CPictureDoc: public CDocument {
    public:

virtual void NewFile();
virtual void OpenFile(SFReply *macReply);
virtual Boolean DoSave();
virtual Boolean DoSaveAs(SFReply *macSFReply);
protected:
  virtual void BuildWindow();
private:
  CPicture *fPicture;
};

TDisposHandleOnFailure.h
#pragma once
#include "TFailureObjects.h"

// disposes the handle passed in the constructor on failure
class TDisposHandleOnFailure: public TCatchFailure {
public:
  TDisposHandleOnFailure(Handle h);
protected:
  virtual void HandleFailure(OSErr err, long message);
private:
  Handle fHandle;
};

DeleteOnCompletion.h
#pragma once
#include "TCL.h"

// disposes the handle passed in the constructor on failure
template<class T> class TDeleteOnCompletion TCL_AUTO_DESTRUCT_OBJECT {
public:
   TCL_DECLARE_CLASS
   TDeleteOnCompletion(T *object);
   ~TDeleteOnCompletion();
private:
   T *fObject;
TDeleteOnCompletion::TDeleteOnCompletion()
{
    fObject = object;
    TCL_END_CONSTRUCTOR
}

TDeleteOnCompletion::~TDeleteOnCompletion()
{
    TCL_START_DESTRUCTOR
    delete fObject;
}

DeleteOnFailure.h

#pragma once
#include "THandleFailure.h"

// disposes the handle passed in the constructor on failure

template<class T> class TDeleteOnFailure: public THandleFailure {
public:
    TDeleteOnFailure(T *object)
    {
        fObject = object;
        TCL_END_CONSTRUCTOR
    }
    ~TDeleteOnFailure()
    {
        TCL_START_DESTRUCTOR
        if (!fSucceeded)
            delete fObject;
    }
private:
    T *fObject;
};
DisposeHandleOnFailure.h

#pragma once

#include "THandleFailure.h"

// disposes the handle passed in the constructor on failure
class TDisposeHandleOnFailure: public THandleFailure {
public:
    TCL_DECLARE_CLASS
    TDisposeHandleOnFailure(Handle h);
    ~TDisposeHandleOnFailure();
private:
    Handle fHandle;
};

HandleFailure.h

#pragma once

#include <TCL.h>

/* calls HandleFailure if a failure occurred during its lifetime.
   Use like:
   {
       THandleFailure aFailureObject;
       // do something which could fail
       aFailureObject.Success();
   } // aFailureObject.HandleFailure() called if Success was not called.
*/
class THandleFailure TCL_AUTO_DESTRUCT_OBJECT {
public:
    TCL_DECLARE_CLASS
    THandleFailure();
    ~THandleFailure(); // stack-based, don't need virtual

    void Success();
protected:
    Boolean fSucceeded;
};
HandleLocker.h

#pragma once

#include "TCL.h"
#include <Memory.h>

class THandleLocker TCL_AUTO_DESTRUCT_OBJECT {
public:
    TCL_DECLARE_CLASS
    THandleLocker(Handle h);
    ~THandleLocker();

private:
    Handle fHandle;
    char fState;
};

TWithFileOpen.h

#pragma once

#include "TCL.h"
#include <CFile.h>
#include <Types.h>

class TWithFileOpen TCL_AUTO_DESTRUCT_OBJECT {
public:
    TCL_DECLARE_CLASS
    TWithFileOpen(CFile *theFile, SignedByte permission);
    ~TWithFileOpen();

private:
    CFile *fFile;
};
Chapter 19, TickTock

CAboutBoxApplication.cp

// Copyright © 1993-94, Neil Rhodes and Julie McKeehan. All rights reserved.

#include "CAboutBoxApplication.h"
#include <SegLoad.h>

CAboutBoxApplication::CAboutBoxApplication(short extraMasters,
  Size aRainyDayFund, Size aCriticalBalance, Size aToolboxBalance) :
  CApplication(extraMasters, aRainyDayFund, aCriticalBalance,
               aToolboxBalance)
{
    TCL_END_CONSTRUCTOR;
}

void CAboutBoxApplication::DoCommand(long theCommand)
{
    const short kAboutBoxResourceID = 1024;

    switch (theCommand) {
    case cmdAbout:
        Str255 applicationName;
        short appRefNum;
        Handle h;

        ::PositionDialog('ALRT', kAboutBoxResourceID);
        ::GetAppParms(applicationName, &appRefNum, &h);
        ::DisposHandle(h);
        ::ParamText(applicationName, NULL, NULL, NULL);
        ::Alert(kAboutBoxResourceID, NULL);
        break;
    default:
        CApplication::DoCommand(theCommand);
    break;
    }
}
CAboutBoxApplication.h

#pragma once
#include <CApplication.h>

class CAboutBoxApplication: public CApplication {
public:
    CAboutBoxApplication(short extraMasters, Size aRainyDayFund,
    Size aCriticalBalance, Size aToolboxBalance);
    virtual void DoCommand(long theCommand);
};

CClockApp.cp

// Copyright © 1993-94, Neil Rhodes and Julie McKeehan. All rights reserved.

/****
 * CClockApp.c
 *
 ****/

#include "CClockApp.h"
#include "CClockPane.h"
#include <CDecorator.h>
#include <CDirector.h>
#include <CScrollPane.h>
#include <Commands.h>
#include "CSaveRestoreSizeWindow.h"

const short kExtraMasters = 1;
const Size kRainyDayFund = 20480;
const Size kCriticalBalance = 20480;
const Size kToolboxBalance = 20480;

class CClockApp: public CApplication {
public:
    CClockApp():
        CAboutBoxApplication(kExtraMasters, kRainyDayFund,
        kCriticalBalance, kToolboxBalance)
void CClockApp::StartUpAction(short numPreloads)
{
    CAboutBoxApplication::StartUpAction(numPreloads);

    const Boolean kFloatingWindow = TRUE;
    const short kMinWindowHeight = 30;
    const short kMinWindowWidth = 30;

    CDirector *aDirector = new CDirector(this);

    CSaveRestoreSizeWindow *aWindow =
        new CSaveRestoreSizeWindow(kWINDStarter, !kFloatingWindow, aDirector);
    aDirector->itsWindow = aWindow;
    Rect sizeRect = aWindow->sizeRect;
    sizeRect.top = kMinWindowHeight;
    sizeRect.left = kMinWindowWidth;
    aWindow->SetSizeRect(&sizeRect);

    aWindow->Select();
    CClockPane *thePane =
        new CClockPane(aWindow, aDirector, 0, 0, 0, 0, sizELASTIC, sizELASTIC);
    thePane->FitToEnclosure(TRUE, TRUE);
    aDirector->itsGopher = thePane;
    thePane->BecomeGopher(TRUE);
}

CClockApp.h

/*****
 * CClockApp.h
 *
 *  Application class for a typical application.
 *  *
 *  *****/

#pragma once

#include "CAboutBoxApplication.h"
class CClockApp: public CAboutBoxApplication
{
public:
    CClockApp();
    virtual void StartUpAction(short numPreloads);
};

CClockPane.cp
// Copyright © 1993-94, Neil Rhodes and Julie McKeegan. All rights reserved.

#include "CClockPane.h"
#include "CDawdleBureaucratChore.h"
#include <Global.h>
#include <Commands.h>
#include <CApplication.h>
#include <CBartender.h>
#include <Packages.h>
#include <SANE.h>
#include "Utilities.h"
#include "TGrafPortSetter.h"

extern CApplication *gApplication;
extern CBartender *gBartender;
extern long gSleepTime;

CClockPane::CClockPane(CView *anEnclosure, CBureaucrat *aSupervisor,
short aWidth, short aHeight,
short aHEncl, short aVEncl,
SizingOption aHSizing, SizingOption aVSizing) :
CPane(anEnclosure, aSupervisor, aWidth, aHeight,
aHEncl, aVEncl, aHSizing, aVSizing)
{
    fIdleChore = NULL;
    fIdleChoreIsInstalled = FALSE;
    fBitMap.baseAddr = NULL;
    fGrafPtr = NULL;
    SetRect(&fBitMap.bounds, 0, 0, 0, 0);

    fIdleChore = new CDawdleBureaucratChore(this);
    fGrafPtr = (GrafPtr) ::NewPtrCanFail(sizeof(GrafPort));
::FailNIL(fGrafPtr);
::OpenPort(fGrafPtr);

fDigital = false;
fShowSeconds = true;
}

CClockPane::~CClockPane()
{
if (fIdleChoreIsInstalled)
gApplication->CancelIdleChore(fIdleChore);
TCLForgetObject(fIdleChore);
if (fBitMap.baseAddr != NULL)
::DisposPtr(fBitMap.baseAddr);
if (fGrafPtr != NULL) {
::ClosePort(fGrafPtr);
::DisposPtr((Ptr) fGrafPtr);
}

void CClockPane::Activate(void)
{
CPane::Activate();
if (fIdleChoreIsInstalled) {
    gApplication->CancelIdleChore(fIdleChore);
    fIdleChoreIsInstalled = false;
}
}

void CClockPane::Deactivate(void)
{
CPane::Deactivate();
gApplication->AssignIdleChore(fIdleChore);
fIdleChoreIsInstalled = true;
}

void CClockPane::UpdateMenus()
{
CPane::UpdateMenus();

gBartender->EnableCmd(cmdShowSeconds);
gBartender->EnableCmd(cmdDigital);
gBartender->EnableCmd(cmdAnalog);
gBartender->CheckMarkCmd(cmdShowSeconds, fShowSeconds);
gBartender->CheckMarkCmd(cmdDigital, fDigital);
gBartender->CheckMarkCmd(cmdAnalog, !fDigital);
}

void CClockPane::DoCommand(long theCommand)
{
    switch (theCommand) {
    case cmdShowSeconds:
        fShowSeconds = !fShowSeconds;
        Refresh();
        if (fShowSeconds)
            gSleepTime = 1;        // Make Dawdle start being called
        break;

    case cmdDigital:
        if (!fDigital) {
            fDigital = TRUE;
            Refresh();
        }
        break;

    case cmdAnalog:
        if (fDigital) {
            fDigital = FALSE;
            Refresh();
        }
        break;

    default:
        CPane::DoCommand(theCommand);
        break;
    }
}

inline short Round(float f)
{
    return (short) (f + 0.5);
}

void CClockPane::UpdateOffscreen(const Rect &area)
{
LongRect   frame;
Rect       qdFrame;

GetFrame(&frame);
::LongToQDRect(&frame, &qdFrame);
CreateOffscreen(qdFrame);       // we would use area, if we weren't dawdling

TGrafPortSetter setPort(fGrafPtr);
DrawClock(qdFrame);
}

void CClockPane::Draw(Rect *area)
{
    UpdateOffscreen(*area);
    CopyFromOffscreen(*area);
}

// fractionOfCircle: 0 means straight up, .25 means to the right
void DrawLine(const Rect &frame, long double fractionOfCircle, short size)
{
    long double angle = 2 * 3.14159 * (1-(fractionOfCircle - 0.25));
    short radiusX = (frame.right - frame.left) / 2;
    short radiusY = (frame.bottom - frame.top) / 2;

    short xCoord = ::Round(::cos(angle) * radiusX);
    short yCoord = ::Round(::sin(angle) * radiusY);

    ::PenSize(size, size);
    ::MoveTo(frame.left + radiusX, frame.top + radiusY);
    ::Line(xCoord, -yCoord);       // Macintosh uses Y decreasing
}

inline short NumberOfBytesRoundedUpToAMultipleOf2(short num)
{
    return (num + 15) / 16 * 2;
}

void CClockPane::CreateOffscreen(const Rect &area)
{
    if (area.left != fBitMap.bounds.left ||
        area.right != fBitMap.bounds.right ||
        area.top != fBitMap.bounds.top ||
        area.bottom != fBitMap.bounds.bottom) {

fBitmap.bounds = area;
fBitmap.rowBytes =
    ::NumberOfBytesRoundedUpToAMultipleOf2(fBitmap.bounds.right - fBitmap.bounds.left);
if (fBitmap.baseAddr != NULL)
    ::DisposPtr(fBitmap.baseAddr);

fBitmap.baseAddr =
    ::NewPtrCanFail(fBitmap.rowBytes * (fBitmap.bounds.bottom - fBitmap.bounds.top));
::FailNIL(fBitmap.baseAddr);
::RectRgn(fGrafPtr->visRgn, &area);
::RectRgn(fGrafPtr->clipRgn, &area);
fGrafPtr->portRect = area;

TGrfPortSetter setPort(fGrafPtr);
::SetPortBits(&fBitmap);
}

void CClockPane::GetDigitalString(unsigned long now, Str255 s)
{
    ::IUTimeString(now, fShowSeconds, s);
}

void CClockPane::GetDigitalFontAndSize(const Str255 s, const Rect &clockSize,
    short *font, short *size)
{
    *font = geneva;
    *size = 9;
}

void CClockPane::GetDigitalStringLocation(const Str255 s, const Rect &clockSize,
    short *h, short *v)
{
    Fontinfo finfo;
    ::GetFontinfo(&finfo);
    short excessHeight = clockSize.bottom - clockSize.top - finfo.descent - finfo.ascent;
    if (excessHeight < 0)
        *v = clockSize.bottom;
    else
        *v = clockSize.bottom - finfo.descent - excessHeight / 2;

    short excessWidth = clockSize.right - clockSize.left - ::StringWidth(s);
    if (excessWidth < 0)
        *h = clockSize.left;

Symantec C++ Programming for the Macintosh, 2nd Edition
else
    *h = clockSize.left + excessWidth / 2;
}

void CClockPane::DrawDigitalClock(const Rect &clockSize, unsigned long now)
{
    Str255 nowStr;
    short h;
    short v;
    short font;
    short size;

    GetDigitalString(now, nowStr);
    GetDigitalFontAndSize(nowStr, clockSize, &font, &size);
    ::TextFont(font);
    ::TextSize(size);
    GetDigitalStringLocation(nowStr, clockSize, &h, &v);
    ::MoveTo(h, v);
    ::DrawString(nowStr);
}

void CClockPane::DrawAnalogClock(const Rect &clockSize, unsigned long now)
{
    DateTimeRec nowDateTime;
    const short kSecondHandThickness = 1;
    const short kMinuteHandThickness = 2;
    const short kHourHandThickness = 3;

    ::Secs2Date(now, &nowDateTime);

    if (fShowSeconds)
        DrawLine(clockSize, ((float) nowDateTime.second) / 60.0, kSecondHandThickness);
    DrawLine(clockSize, nowDateTime.minute / 60.0, kMinuteHandThickness);

    Rect smallFrame = clockSize; // hour hand will be inscribed here
    const kHourHandSmallerConstant = 2; // half as big as other hands
    ::InsetRect(&smallFrame,
        (clockSize.right - clockSize.left) / (2 * kHourHandSmallerConstant),
        (clockSize.bottom - clockSize.top) / (2 * kHourHandSmallerConstant));
    DrawLine(smallFrame, nowDateTime.hour / 12.0, kHourHandThickness);
}
void CClockPane::DrawClock(const Rect &clockSize)
{
    unsigned long now;
    ::GetDateTime(&now);
    ::EraseRect(&clockSize);
    if (fDigital)
        DrawDigitalClock(clockSize, now);
    else
        DrawAnalogClock(clockSize, now);
}

void CClockPane::CopyFromOffscreen(const Rect &areaToCopy)
{
    ::CopyBits(&fBitMap, &thePort->portBits,
                &areaToCopy, &areaToCopy, srcCopy, 0L);
}

long CClockPane::CalculateSleepTime()
{
    const short kTicksPerSecond = 60;
    if (fShowSeconds)
        return kTicksPerSecond;
    else {
        unsigned long now;
        DateTimeRec nowDateTime;
        const short kSecondsPerMinute = 60;
        GetDateTime(&now);
        Secs2Date(now, &nowDateTime);
        return kTicksPerSecond * (kSecondsPerMinute - nowDateTime.second);
    }
}

void CClockPane::Dawdle(long *maxSleep)
{
    LongRect frame;
    Rect qdFrame;
GetFrame(&frame);
::LongToQDRect(&frame, &qdFrame);
UpdateOffscreen(qdFrame);

Prepare();
CopyFromOffscreen(qdFrame);

*maxSleep = CalculateSleepTime();
}

**CClockPane.h**

/****
 ****/

#pragma once

#include <CPane.h>
#include <Quickdraw.h>
#include "CDawdleBureaucratChore.h"

class CClockPane: public CPane {
public:
    CClockPane(CView *anEnclosure, CBureaucrat *aSupervisor,
                short aWidth, short aHeight,
                short aHEncl, short aVEncl,
                SizingOption aHSizing, SizingOption aVSizing);

    virtual ~CClockPane();

    virtual void Draw(Rect *area);
    virtual void Activate(void);
    virtual void Deactivate(void);
    virtual void Dawdle(long *maxSleep);
    virtual void UpdateMenus();
    virtual void DoCommand(long theCommand);

protected:
    enum {cmdShowSeconds = 1024, cmdDigital=1025, cmdAnalog=1026};
virtual void DrawClock(const Rect &clockSize);
virtual void CopyFromOffscreen(const Rect &areaToCopy);
virtual void CreateOffscreen(const Rect &area);
virtual void UpdateOffscreen(const Rect &area);
virtual long CalculateSleepTime();
virtual void DrawAnalogClock(const Rect &clockSize, unsigned long now);
virtual void DrawDigitalClock(const Rect &clockSize, unsigned long now);
virtual void GetDigitalString(unsigned long now, Str255 s);
virtual void GetDigitalFontAndSize(const Str255 s, const Rect &clockSize, short *font, short *size);
virtual void GetDigitalStringLocation(const Str255 s, const Rect &clockSize, short *h, short *v);

private:

CDawdleBureaucratChore *fIdleChore;
Boolean fIdleChoreIsInstalled;
BitMap fBitMap;
GrafPtr fGrafPtr;
Boolean fShowSeconds;
Boolean fDigital;

CDawdleBureaucratChore.cp
// Copyright © 1993-94, Neil Rhodes and Julie McKeenan. All rights reserved.

#include "CDawdleBureaucratChore.h"

CDawdleBureaucratChore::CDawdleBureaucratChore(CBureaucrat *theBureaucrat)
{
    fBureaucratToDawdle = theBureaucrat;
}

void CDawdleBureaucratChore::Perform(long *maxSleep)
{
    fBureaucratToDawdle->Dawdle(maxSleep);
}
CDawdleBureaucratChore.h

#pragma once

#include <CChore.h>
#include <CBureaucrat.h>

class CDawdleBureaucratChore: public CChore {
public:
    CDawdleBureaucratChore(CBureaucrat *theBureaucrat);
    virtual void Perform(long *maxSleep);
private:
    CBureaucrat *fBureaucratToDawdle;
};

Clock.cp

// Copyright © 1993-94, Neil Rhodes and Julie McKeahan. All rights reserved.
******************************************************************************/

#include "CClockApp.h"

void main()
{
    CClockApp *clockApp;

    clockApp = new CClockApp;
    clockApp->Run();
    clockApp->Exit();
}
#include "Types.r"
#include "SysTypes.r"

resource 'MENU' (4, "Clock", preload) {
    4,
    textMenuProc,
    0x7FFFFFFF,
    enabled,
    "Clock",
    {
        "Show seconds", noIcon, noKey, noMark, plain,
        ".", noIcon, noKey, noMark, plain,
        "Digital", noIcon, noKey, noMark, plain,
        "Analog", noIcon, noKey, noMark, plain,
    }
};

resource 'MBAR' (1, preload) {
    { 1, 2, 3, 4
    }
};

resource 'ALRT' (1024, "kAlertResourceID") {
    {40, 40, 184, 276},
    1024,
    { /* array: 4 elements */
        OK, visible, sound1,
        OK, visible, sound1,
        OK, visible, sound1,
        OK, visible, sound1
    }
};

resource 'DITL' (1024, "kAlertResourceID") {
    {5, 12, 93, 228},
    StaticText {
        disabled,
resource 'WIND' (500, "Clock window", purgeable, preload) {
    {68, 44, 168, 144},
    documentProc,
    invisible,
    goAway,
    0x0,
    "Clock"
};

resource 'BNDL' (128) {
    'C++5',
    0,
    {
        'FREF',
        {
            0, 128
        },
        'ICN#',
        {
            0, 128
        }
    }
};
data 'C++5' (0, "Owner resource") {
    "$00"
};
data 'FREF' (128) {
    "$4150 504C 0000 84"
};
resource 'ICN#' (128) {
    { /* array: 2 elements */
        /* [1] */
        "$0000 0000 0000 0000 0003 B000 000C 6000"
        "$0018 3000 2410 1004 A81F F015 7078 3C0E"
        "$0E00 0E3E 70C0 030E 31B1 018C 3301 00CC"
        "$3301 00CC 3E01 007C 3E01 007C 0601 0060"
        "$0601 FC60 0602 0606 0604 0606 0608 0606"
        "$0310 00C0 0320 00C0 0180 0180 00C0 0300"
        "$00F0 0F00 007F FE00 007F FE00 0070 0E00"
        "$0600 0600 0600 0600 00F0 0F00 01F8 1F80",
        /* [2] */
        "$0000 0000 0000 0000 0003 B000 000F E000"
        "$001F F000 241F F004 A81F F015 707F FC0E"
        "$0F00 0F00 241F F004 A81F F015 707F FC0E"
        "$07FF FF00 241F F004 A81F F015 707F FC0E"
        "$7FFC FE3E 70FF FF0E 31FF FF0C 33FF FFCC"
        "$33FF FFCC 3FFF FFFF 3FFF FFFF 07FF FE00"
        "$7FFC FF00 07FF FF0E 07FF FF00 07FF FE00"
        "$33FF FF00 03FF FF00 01FF FF00 00FF FF00"
        "$00FF FF00 007F FE00 007F FE00 0070 0E00"
        "$0600 0600 0600 0600 00F0 0F00 01F8 1F80"
    }
};

resource 'ics#' (128) {
    { /* array: 2 elements */
        /* [1] */
        "$07C0 1830 2108 4104 4104 8102 8102 81F2"
        "$8202 8402 4804 4004 2008 1830 1FF0 701C",
        /* [2] */
        "$07C0 1FF0 3FF8 7FFC 7FFC FFFE FFFE FFFE"
        "$FFE0 FFFE 7FFC 7FFC 3FF8 1FF0 1FF0 701C"
    }
};

resource 'vers' (1) {
    Ox1,
    Ox1,
    release,
    Ox0,
    verUS,
    "PICTPeeker 1.1",
    "1.1, from Symantec C++ Programming for t"
}
"he Macintosh."

resource 'vers' (2) {
    0x1,
    0x1,
    release,
    0x0,
    verUS,
    "PICTPeeker 1.1",
    "1.1 ©1993-94, Rhodes and McKeehan."
};

resource 'MENU' (2, "File", preload) {
    2,
    textMenuProc,
    0x7FFFFFFFFE,
    enabled,
    "File",
    {
        "Close#4", noIcon, "W", noMark, plain,
        ":-", noIcon, noKey, noMark, plain,
        "Quit#1", noIcon, "Q", noMark, plain
    }
};

resource 'MENU' (3, "Edit", preload) {
    3,
    textMenuProc,
    0x7FFFFF80,
    enabled,
    "Edit",
    {
        "Undo#16", noIcon, "Z", noMark, plain,
        ":-", noIcon, noKey, noMark, plain,
        "Cut#18", noIcon, "X", noMark, plain,
        "Copy#19", noIcon, "C", noMark, plain,
        "Paste#20", noIcon, "V", noMark, plain,
        "Clear#21", noIcon, noKey, noMark, plain,
        "Select All#23", noIcon, "A", noMark, plain,
        ":-", noIcon, noKey, noMark, plain,
        "Show Clipboard#22", noIcon, noKey, noMark, plain
    }
}
resource 'MENU' (1, "Apple", preload) {
    1,
    textMenuProc,
    0x7FFFFFFFD,
    enabled,
    apple,
    {
        "About Tick-Tock_#256", noIcon, noKey, noMark, plain,
        ",", noIcon, noKey, noMark, plain
    }
};

resource 'WIND' (200, "Clipboard", purgeable, preload) {
    {130, 158, 238, 444},
    zoomDocProc,
    invisible,
    goAway,
    0x0,
    "Clipboard"
};

resource 'CNTL' (300, "Scroll Bar", preload) {
    {0, 0, 0, 0},
    0,
    visible,
    0,
    0,
    scrollBarProc,
    0,
    ""
};

CQuitApplicationChore.cp

// Copyright © 1993-94, Neil Rhodes and Julie McKeehan. All rights reserved.

#include "CQuitApplicationChore.h"
#include <CApplication.h>
extern CApplication *gApplication;

void CQuitApplicationChore::Perform(long *maxSleep)
{
    gApplication->Quit();
}

CQuitApplicationChore.h
#pragma once
#include <CChore.h>

class CQuitApplicationChore: public CChore {
public:
    virtual void Perform(long *maxSleep);
};

CQuitOnCloseWindow.cp
// Copyright © 1993, Neil Rhodes and Julie McKeehan. All rights reserved.

#include "CQuitOnCloseWindow.h"
#include "CQuitApplicationChore.h"
#include <CApplication.h>

extern CApplication *gApplication;

CQuitOnCloseWindow::CQuitOnCloseWindow(short WINDid, Boolean aFloating, CDirector *aSupervisor) :
    CWindow(WINDid, aFloating, aSupervisor)
{
    TCL_END_CONSTRUCTOR;
}

CQuitOnCloseWindow::~CQuitOnCloseWindow()
{
    TCL_START_DESTRUCTOR;
    gApplication->AssignUrgentChore(new CQuitApplicationChore);
}
CQuitOnCloseWindow.h
CSaveRestoreSizeWindow.cp// Copyright © 1993-94, Neil Rhodes and Julie McKeenan. All rights reserved.

#include "CSaveRestoreSizeWindow.h"
#include <Folders.h>
#include "TGrafPortSetter.h"

CSaveRestoreSizeWindow::CSaveRestoreSizeWindow(
    short windID,
    Boolean aFloating,
    CDirector *aSupervisor) :
    CQuitOnCloseWindow(windID, aFloating, aSupervisor)
{
    fDataFile = NULL;

    fDataFile = new CDataFile;
    SpecifyFile(fDataFile);
    SetInfoFromFile(fDataFile);
    TCL_END_CONSTRUCTOR;
}

CSave RestoreSizeWindow:~-CSaveRestoreSizeWindow()
{
    TCL_START_DESTRUCTOR;
    StoreInfoInFile(fDataFile);
    TCLForgetObject(fDataFile);
}

void CSaveRestoreSizeWindow::SpecifyFile(CFile *aFile)
{
    short vRefNum;
    long dirID;

    FailOSErr(FindFolder(kOnSystemDisk, kPreferencesFolderType,
        kCreateFolder, &vRefNum,&dirID));
    aFile->SpecifyHFS("\pTick-Tock Preferences", vRefNum, dirID);
}

void CSaveRestoreSizeWindow::StoreInfoInFile(CDataFile *aFile)
{
if (!aFile->ExistsOnDisk())
    aFile->CreateNew('C++5', 'PREF');

aFile->Open(fsWrPerm);

Rect theWindowLocSize = macPort->portRect;

TGrarfPortSetter setPort(macPort);

::LocalToGlobal((Point *) &theWindowLocSize.top);
::LocalToGlobal((Point *) &theWindowLocSize.bottom);

TRY {
    aFile->WriteSome((Ptr) &theWindowLocSize, sizeof(theWindowLocSize));
}
CATCH {
    NO_PROPAGATE;  // not writing window location isn't fatal
} ENDTRY;
aFile->Close();

void CSaveRestoreSizeWindow::SetinfoFromFile(CDataFile *aFile)
{
    if (aFile->ExistsOnDisk()) {
        Rect portRect;

        aFile->Open(fsRdPerm);

        TRY {
            aFile->ReadSome((Ptr) &portRect, sizeof(portRect));
            Move(portRect.left, portRect.top);
            ChangeSize(portRect.right - portRect.left, portRect.bottom - portRect.top);
        }
        CATCH {
            NO_PROPAGATE;  // not reading window location isn't fatal
        } ENDTRY;
aFile->Close();
    }
}
CSaveRestoreSizeWindow.h

#pragma once

#include "CQuitOnCloseWindow.h"
#include <CDataFile.h>

class CSaveRestoreSizeWindow: public CQuitOnCloseWindow {
public:

    CSaveRestoreSizeWindow(
        short windID,
        Boolean aFloating,
        CDirector *aSupervisor);

    virtual ~CSaveRestoreSizeWindow();

protected:

    virtual void StoreInfoInFile(CDataFile *aFile);
    virtual void SetInfoFromFile(CDataFile *aFile);
    virtual void SpecifyFile(CFile *aFile);

private:

    CDataFile *fDataFile;
};

TGrafPortSetter.h

#pragma once

#include <QuickDraw.h>
#include <TCL.h>

class TGrafPortSetter TCL_AUTO_DESTRUCT_OBJECT {
public:

    TGrafPortSetter(GrafPtr newGrafPtr);
    virtual ~TGrafPortSetter();

private:

    GrafPtr fOldGrafPtr;
};
inline TGrafPortSetter::TGrafPortSetter(GrafPtr newGrafPtr)
{
    ::GetPort(&fOldGrafPtr);
    ::SetPort(newGrafPtr);
    TCL_END_CONSTRUCTOR;
}

inline TGrafPortSetter::-TGrafPortSetter()
{
    TCL_START_DESTRUCTOR;
    ::SetPort(fOldGrafPtr);
}

 Utilities.cp

// Copyright © 1993, Neil Rhodes and Julie McKeehan. All rights reserved.

#include <TCLUtilities.h>
#include "Utilities.h"

Ptr NewPtrCanFail(Size size)
{
    Ptr p;
    Boolean savedAlloc;

    savedAlloc = SetAllocation(kAllocCanFail);
    p = NewPtr(size);
    SetAllocation(savedAlloc);

    return p;
}

Utilities.h

#pragma once

#include <Memory.h>

Ptr NewPtrCanFail(Size size);
Chapter 20, C++ Calc

#include "CAboutBoxApplication.h"
#include <SegLoad.h>

CAboutBoxApplication::CAboutBoxApplication(short extraMasters,
   Size aRainyDayFund, Size aCriticalBalance, Size aToolboxBalance) :
   CApplication(extraMasters, aRainyDayFund, aCriticalBalance,
   aToolboxBalance)
{
   TCL_END_CONSTRUCTOR;
}

void CAboutBoxApplication::DoCommand(long theCommand)
{
   const short kAboutBoxResourceID = 1024;

   switch (theCommand) {
   case cmdAbout:
      Str255 applicationName;
      short appRefNum;
      Handle h;
      ::PositionDialog('ALRT', kAboutBoxResourceID);
      ::GetAppParms(applicationName, &appRefNum, &h);
      ::DisposHandle(h);
      ::ParamText(applicationName, NULL, NULL, NULL);
      ::Alert(kAboutBoxResourceID, NULL);
      break;
   default:
      CApplication::DoCommand(theCommand);
      break;
   }
}
CAboutBoxApplication.h

#pragma once
#include <CAppliication.h>

class CAboutBoxApplication: public CApplication {
   public:
       CAboutBoxApplication(short extraMasters, Size aRainyDayFund,
                              Size aCriticalBalance, Size aToolboxBalance);

       virtual void DoCommand(long theCommand);
   }

Calculator.cp

// Copyright © 1993, Neil Rhodes and Julie McKeenan. All rights reserved.

#include "Calculator.h"

TCalculator::TCalculator()
{
    fX = 0.0;
    fY = 0.0;
    fZ = 0.0;
}

void TCalculator::Enter(long double num)
{
    fZ = fY;
    fY = fX;
    fX = num;
}

void TCalculator::Add()
{
    long double v = PopX();
    Enter(PopX() + v);
}

void TCalculator::Subtract()
long double v = PopX();
Enter(PopX() - v);

void TCalculator::Multiply()
{
    long double v = PopX();
Enter(PopX() * v);
}

void TCalculator::Divide()
{
    long double v = PopX();
Enter(PopX() / v);
}

long double TCalculator::GetX()
{
    return fX;
}

long double TCalculator::GetY()
{
    return fY;
}

long double TCalculator::GetZ()
{
    return fZ;
}

void TCalculator::SetX(long double num)
{
    fX = num;
}

void TCalculator::SetY(long double num)
{
    fY = num;
}

void TCalculator::SetZ(long double num)
long double TCalculator::PopX()
{
    long double v = GetX();
    
    fX = fY;
    fY = fZ;
    fZ = 0.0;
    return v;
}

Calculator.h

#pragma once

class TCalculator {
public:
    TCalculator();
    void Enter(long double num);
    void Add();
    void Subtract();
    void Multiply();
    void Divide();
    long double GetX();
    long double GetY();
    long double GetZ();

private:
    long double PopX();
    void SetX(long double num);
    void SetY(long double num);
    void SetZ(long double num);

    long double fX;
    long double fY;
    long double fZ;
};
// Copyright © 1993-94, Neil Rhodes and Julie McKeehan. All rights reserved.

#include "Types.r"
#include "SysTypes.r"

resource 'MBAR' (1, preload) {
    { 1, 2, 3
    }
};

resource 'ALRT' (1024, "kAlertResourceID") {
    {40, 40, 184, 276},
    1024,
    { /* array: 4 elements */
        OK, visible, sound1,
        OK, visible, sound1,
        OK, visible, sound1,
        OK, visible, sound1
    } 
};

resource 'DITL' (1024, "kAlertResourceID") {
    {5, 12, 93, 228},
    StaticText {
        disabled,
        "\"0, from Symantec C++ Programming for the Macintosh, ©1993-94 b"
        "y Neil Rhodes and Julie McKeehan, Brady "
        "Books"
    },
    {116, 147, 136, 205},
    Button {
        enabled,
        "OK"
    }
};

resource 'STR#' (1025, "button names", purgeable, preload) {
resource 'WIND' (500, "C++ Calc", purgeable, preload) {
    {68, 44, 216, 150},
    rDocProc,
    invisible,
    goAway,
    0x00,
    "C++ Calc"
};

resource 'BNDL' (128) {
    'C++6',
    0,
    {
        'FREF',
        {
            0, 128
        },
        'ICN#',
        {
            0, 128
        }
    }
}
data 'C++6' (0, "Owner resource") {
    "$00"
};

data 'FREF' (128) {
    "$4150 504C 0000 84"
};

resource 'ics#' (128) {
    { /* array: 2 elements */
        /* [1] */
        "$3FE0 4010 4010 7FF0 4010 5550 4010 5550",
        "$4010 5550 4010 5550 4010 4550 4010 3FE0",
        /* [2] */
        "$3FE0 7FF0 7FF0 7FF0 7FF0 7FF0 7FF0 7FF0",
        "$7FF0 7FF0 7FF0 7FF0 7FF0 7FF0 7FF0 3FE0"
    }
};

resource 'ICN#' (128) {
    { /* array: 2 elements */
        /* [1] */
        "$1FFF FF80 3FFF FFC0 3FFF FFC0 3FFF FFC0",
        "$2000 0040 27FF FE40 2400 0240 2400 0240",
        "$27FF FE40 2000 0040 2739 CE40 2529 4A40",
        "$2739 CE40 2000 0040 2739 CE40 2529 4A40",
        "$2739 CE40 2000 0040 2739 CE40 2529 4A40",
        "$2739 CE40 2000 0040 2039 CE40 2029 4A40",
        "$2039 CE40 2000 0040 1000 0080 0FFF FF",
        /* [2] */
        "$1FFF FF80 3FFF FFC0 3FFF FFC0 3FFF FFC0",
        "$3FFF FFC0 3FFF FFC0 3FFF FFC0 3FFF FFC0",
        "$3FFF FFC0 3FFF FFC0 3FFF FFC0 3FFF FFC0",
        "$3FFF FFC0 3FFF FFC0 3FFF FFC0 3FFF FFC0",
        "$3FFF FFC0 3FFF FFC0 3FFF FFC0 3FFF FFC0",
        "$3FFF FFC0 3FFF FFC0 3FFF FFC0 3FFF FFC0",
        "$3FFF FFC0 3FFF FFC0 3FFF FFC0 3FFF FFC0",
        "$3FFF FFC0 3FFF FFC0 3FFF FFC0 3FFF FFC0",
        "$3FFF FFC0 3FFF FFC0 1FFF FF80 0FFF FF"
    }
}
resource 'vers' (1) {
    0x1,
    0x1,
    release,
    0x0,
    verUS,
    "C++ Calc 1.1",
    "1.1, from Symantec C++ Programming for the Macintosh."
};

resource 'vers' (2) {
    0x1,
    0x1,
    release,
    0x0,
    verUS,
    "C++ Calc 1.0",
    "1.1 ©1993-94, Rhodes and McKeehan."
};

resource 'MENU' (2, "File", preload) {
    2,
    textMenuProc,
    0x7FFFFFFE,
    enabled,
    "File",
    {
        "Close#4", noIcon, "W", noMark, plain,
        "Quit#1", noIcon, "Q", noMark, plain
    }
};

resource 'WIND' (200, "Clipboard", purgeable, preload) {
    {130, 158, 238, 444},
    zoomDocProc,
    invisible,
    goAway,
    0x0,
    "Clipboard"
};
resource 'MENU' (3, "Edit", preload) {
    3,
    textMenuProc,
    0x7FFFFFFF80,
    enabled,
    "Edit",
    {
        "Undo", noIcon, "Z", noMark, plain,
        ".", noIcon, noKey, noMark, plain,
        "Cut", noIcon, "X", noMark, plain,
        "Copy", noIcon, "C", noMark, plain,
        "Paste", noIcon, "V", noMark, plain,
        "Clear", noIcon, noKey, noMark, plain,
        "Select All", noIcon, "A", noMark, plain,
        ",", noIcon, noKey, noMark, plain,
        "Show Clipboard", noIcon, noKey, noMark, plain
    }
};

resource 'MENU' (1, "Apple", preload) {
    1,
    textMenuProc,
    0x7FFFFFFFD,
    enabled,
    apple,
    {
        "About C++ Calc", noIcon, noKey, noMark, plain,
        ",", noIcon, noKey, noMark, plain
    }
};

resource 'CINTL' (300, "Scroll Bar", preload) {
    {0, 0, 0, 0},
    0,
    visible,
    0,
    0,
    scrollbarProc,
    0,
    "=
};
#include "CCalculatorApp.h"

void main()
{
    CCalculatorApp *CalculatorApp;

    CalculatorApp = new CCalculatorApp;
    CalculatorApp->Run();
    CalculatorApp->Exit();
}

CBButtonWithFontAndSize.cp
// Copyright © 1993, Neil Rhodes and Julie McKeethan. All rights reserved.

#include "CBButtonWithFontAndSize.h"

CBButtonWithFontAndSize::CBButtonWithFontAndSize(short aWidth, short aHeight,
short aHEncl, short aVEncl,
StringPtr title, Boolean fVisible, short procID,
CView *anEnclosure, CBureaucrat *aSupervisor, StringPtr fontName, short fontSize)
    : CButton(aWidth, aHeight, aHEncl, aVEncl, title, fVisible,
              useWFont ? procID, anEnclosure, aSupervisor)
{
    short fontNumber;

    ::GetFNum(fontName, &fontNumber);
    fFont = fontNumber;
    fFontSize = fontSize;
TCL_END_CONSTRUCTOR;

void CButtonWithFontAndSize::Draw(Rect *area)
{
    ::TextFont(fFont);
    ::TextSize(fFontSize);

    CButton::Draw(area);
}

CButtonWithFontAndSize.h

#pragma once

#include <CButton.h>

class CButtonWithFontAndSize: public CButton {
public:
    CButtonWithFontAndSize(short aWidth, short aHeight,
                           short aHEncl, short aVEncl,
                           StringPtr title, Boolean fVisible, short procID,
                           CView *anEnclosure, CBureaucrat *aSupervisor,
                           StringPtr fontName, short fontSize);

    virtual void Draw(Rect *area);

private:
    short fFont;
    short fFontSize;
};

CCalculatorApp.cp

// Copyright © 1993-94, Neil Rhodes and Julie McKeehan. All rights reserved.

/*****
 * CCalculatorApp.c
 *
*****/
#include "CCalculatorApp.h"
#include "CCalculatorPane.h"
#include <CDirector.h>
#include "CSaveRestoreSizeWindow.h"

const short kExtraMasters = 1;
const Size kRainyDayFund = 20480;
const Size kCriticalBalance = 20480;
const Size kToolboxBalance = 20480;

const short kWINDStarter = 500;

CCalculatorApp::CCalculatorApp() :
    CAboutBoxApplication(kExtraMasters, kRainyDayFund,
        kCriticalBalance, kToolboxBalance)
{
    TCL_END_CONSTRUCTOR;
}

void CCalculatorApp::StartUpAction(short numPreloads)
{
    CAboutBoxApplication::StartUpAction(numPreloads);

    const Boolean kFloatingWindow = TRUE;
    const short kMinWindowHeight = 30;
    const short kMinWindowWidth = 30;

    CDirector *aDirector = new CDirector(this);

    CSaveRestoreSizeWindow *aWindow =
        new CSaveRestoreSizeWindow(kWINDStarter, !kFloatingWindow, aDirector);
    aDirector->itsWindow = aWindow;

    aWindow->Select();
    CCalculatorPane *thePane =
        new CCalculatorPane(aWindow, aDirector, 0, 0, 0, 0,
            sizELASTIC, sizELASTIC);
    thePane->FitToEnclosure(TRUE, TRUE);
    aDirector->itsGopher = thePane;
    thePane->BecomeGopher(TRUE);
#pragma once

#include "CAboutBoxApplication.h"

class CCalculatorApp: public CAboutBoxApplication
{
public:
    CCalculatorApp();
    void StartUpAction(short numPreloads);
};

void CCalculatorPane::CreateCommandButton(short rowNumber, short colNumber, long command) {
    CButtonWithFontAndSize *aButton;
    Str255 buttonName;
    const short kButtonWidth = 18;
    const short kButtonHeight = 16;
    const short kColumnWidth = 23;
    const short kColumnHeight = 22;
const short kLeftMargin = 9;
const short kTopMargin = 29;
const short kButtonTitleStringID = 1025;

GetIndString(buttonName, kButtonTitleStringID,
    command - cmdFirstCommand + 1);
aButton= new CButtonWithFontAndSize(kButtonWidth, kButtonHeight,
    kLeftMargin + (colNumber-1)* kColumnHeight,
    kTopMargin + (rowNumber-1) * kColumnHeight,
    buttonName, TRUE, 0, this, this, "\pGeneva", 9);
aButton->SetClickCmd(command);
    fButtons[command - cmdFirstCommand] = aButton;
}

CCalculatorPane::CCalculatorPane(CView *anEnclosure, CBureaucrat *aSupervisor,
    short aWidth, short aHeight,
    short aHEncl, short aVEncl,
    SizingOption aHSizing, SizingOption aVSizing) :
    CPane(anEnclosure, aSupervisor, aWidth, aHeight,
    aHEncl, aVEncl, aHSizing, aVSizing)
{
    const short kFloatTextWidth= 86;
    const short kFloatTextHeight = 20;
    const short kFloatTextLeftMargin = 10;
    const short kFloatTextTopMargin = 9;

    wantsClicks = true;

    fCalcDisplay = new CFloatText(this, aSupervisor,
        kFloatTextWidth, kFloatTextHeight,
        kFloatTextLeftMargin, kFloatTextTopMargin,
        sizFIXEDSTICKY, sizFIXEDSTICKY, -1);
    fCalcDisplay->Specify(kNotEditable, kNotSelectable, kNotStylable);
    fCalcDisplay->SetFontName("\pMonaco");
    fCalcDisplay->SetFontSize(9);
    fCalcDisplay->SetMaxDisplayWidth(sizeof(fCurrentNumber)-1);
    CPaneBorder *border = new CPaneBorder(kBorderFrame);
    border->SetPenSize(1, 1);
    fCalcDisplay->SetBorder(border);

    CreateCommandButton(5, 2, cmd0);
void CCalculatorPane::UpdateDisplay()
{
    fCalcDisplay->SetValue(fCalculator.GetX());
}

void CCalculatorPane::UpdateMenus()
{
    CPane::UpdateMenus();

    if (Length(fCurrentNumber) > 0)
    {
        gBartender->EnableCmd(cmdClear);
        gBartender->EnableCmd(cmdCopy);
    }
    if (gClipboard->GetData('TEXT', NULL) > 0)
    {
        gBartender->EnableCmd(cmdPaste);
    }
void CCalculatorPane::Draw(Rect *r)
{
    if (!gSystem.hasMoreColorQD)
        ::FillRect(r, &qd.ltGray);
    else {
        RGBColor aColor;
        PixPatHandle myPixPat;

        aColor.red = aColor.green = aColor.blue = 0xaaaa; // light gray
        myPixPat = ::NewPixPat();
        ::MakeRGBPat(myPixPat, &aColor);
        ::FillCRect(r, myPixPat);
        ::DisposPixPat(myPixPat);
    }
}

void CCalculatorPane::DoCommand(long theCommand)
{
    switch (theCommand) {
    case cmdCopy:
        gClipboard->EmptyScrap();
        gClipboard->PutData('TEXT', fCalcDisplay->GetTextHandle());
        break;

    case cmdPaste:
        Handle h;
        Str255 s;

        gClipboard->GetData('TEXT', &h);
        if (GetHandleSize(h) > sizeof(fCurrentNumber)-1)
        {
            SetHandleSize(h, sizeof(fCurrentNumber)-1);
            fCalcDisplay->SetTextHandle(h);
            ::DisposHandle(h);
            fCalcDisplay->GetTextString(s);
            CopyPString(s, fCurrentNumber);
        }
    case cmdClear:
        fButtons[cmdClearEntry - cmdFirstCommand]->SimulateClick();
        break;

    case cmdClearEntry:
    }
fCalcDisplay->SetValue(fCalculator.GetX());
Length(fCurrentNumber) = 0;
break;

case cmdAdd:
    this->EnterCurrentNumber();
    fCalculator.Add();
    UpdateDisplay();
    break;

case cmdSubtract:
    this->EnterCurrentNumber();
    fCalculator.Subtract();
    UpdateDisplay();
    break;

case cmdMultiply:
    this->EnterCurrentNumber();
    fCalculator.Multiply();
    UpdateDisplay();
    break;

case cmdDivide:
    this->EnterCurrentNumber();
    fCalculator.Divide();
    UpdateDisplay();
    break;

case cmdEnter:
    fCalculator.Enter(fCalcDisplay->GetValue());
    Length(fCurrentNumber) = 0;
    UpdateDisplay();
    break;

case cmdChangeSign:
    Str255 minusSign = "\p-";
    Str255 current;
    
    fCalcDisplay->GetTextString(current);
    
    if (Length(current) > 0 && current[1] == '.') {


```c
::BlockMove(&current[2], &current[1], Length(current)-1);
Length(current) = Length(current)-1;
}
else
{
    ::ConcatPStrings(minusSign, current);
    ::CopyPString(minusSign, current);
}
fCalcDisplay->SetTextString(current);
::CopyPString(current, fCurrentNumber);
break;

case cmd0:
    case cmd1:
    case cmd2:
    case cmd3:
    case cmd4:
    case cmd5:
    case cmd6:
    case cmd7:
    case cmd8:
    case cmd9:
        Str31 numberAsString;

        NumToString(theCommand - cmd0, numberAsString);
        if (Length(fCurrentNumber) + Length(numberAsString) < sizeof(fCurrentNumber)) {
            ConcatPStrings(fCurrentNumber, numberAsString);
            fCalcDisplay->SetTextString(fCurrentNumber);
        }
        break;
    case cmdPeriod:
        Str31 periodString = "\p.";
        if (Length(fCurrentNumber) + Length(periodString) < sizeof(fCurrentNumber)) {
            ConcatPStrings(fCurrentNumber, periodString);
            fCalcDisplay->SetTextString(fCurrentNumber);
        }
        break;
    case cmdE:
        Str31 eString = "\pE";
        if (Length(fCurrentNumber) + Length(eString) < sizeof(fCurrentNumber)) {
```
ConcatPStrings(fCurrentNumber, eString);
    fCalcDisplay->SetTextString(fCurrentNumber);
}
break;
default:
    CPane::DoCommand(theCommand);
}

void CCalculatorPane::EnterCurrentNumber()
{
    if (Length(fCurrentNumber) > 0) {
        fCalculator.Enter(fCalcDisplay->GetValue());
        Length(fCurrentNumber) = 0;
        UpdateDisplay();
    }
}

void CCalculatorPane::DoKeyDown(char theChar, Byte keyCode, EventRecord *macEvent)
{
    const kReturnChar = 13;
    const kEnterChar = 3;
    const kBackspaceChar = 8;
    long commandNumber;

    switch (theChar){
    case kBackspaceChar:
        if (Length(fCurrentNumber) == 1)
            DoCommand(cmdClearEntry);
        else if (Length(fCurrentNumber) > 1) {
            Length(fCurrentNumber)--;
            fCalcDisplay->SetTextString(fCurrentNumber);
        }
        return;
    case '0':
    case '1':
    case '2':
    case '3':
    case '4':
    case '5':
    case '6':
    case '7':
    case '8':
case '9':
    commandNumber = cmd0 + theChar - '0';
    break;
case 'e':
case 'E':
    commandNumber = cmdE;
    break;
case 'c':
case 'C':
    commandNumber = cmdClearEntry;
    break;
case '.':
    commandNumber = cmdPeriod;
    break;
case '+':
    commandNumber = cmdAdd;
    break;
case '-':
    commandNumber = cmdChangeSign;
    break;
case '-':
    commandNumber = cmdSubtract;
    break;
case '*':
    commandNumber = cmdMultiply;
    break;
case '/':
    commandNumber = cmdDivide;
    break;
case kReturnChar:
case kEnterChar:
    commandNumber = cmdEnter;
    break;

default:
    inherited::DoKeyDown(theChar, keyCode, macEvent);
    return;
}
fButtons[commandNumber - cmdFirstCommand]->SimulateClick();
CCalculatorPane.h

#pragma once
#include <CPane.h>
#include "Calculator.h"
#include "CFloatText.h"
#include <CButton.h>

class CCalculatorPane: public CPane {
public:
    CCalculatorPane(CView *anEnclosure,
                    CBureaucrat *aSupervisor,
                    short aWidth, short aHeight,
                    short aHEncl, short aVEncl,
                    SizingOption aHSizing, SizingOption aVSizing);
    virtual void UpdateMenus();
    virtual void DoCommand(long theCommand);
    virtual void DoKeyDown(char theChar, Byte keyCode, EventRecord *macEvent);
    virtual void Draw(Rect *r);

protected:
    enum {cmdFirstCommand = 1025,
          cmdAdd = cmdFirstCommand, cmdMultiply, cmdDivide, cmdSubtract, cmdEnter,
          cmdPeriod, cmdE, cmdClearEntry, cmdChangeSign,
          cmd0, cmd1, cmd2, cmd3,
          cmd4, cmd5, cmd6, cmd7, cmdB, cmd9, cmdLastCommand = cmd9 };
    virtual void UpdateDisplay();
    virtual void EnterCurrentNumber();
    virtual void CreateCommandButton(short rowNumber, short colNumber, long command);

private:
    TCalculator fCalculator;
    CFloatText *fCalcDisplay;
    unsigned char fCurrentNumber[9];
    CButton *fButtons[cmdLastCommand - cmdFirstCommand + 1];
};
CFloatText.cp

#include "CFloatText.h"
#include <SANE.h>
#include <Packages.h>
#include <stdio.h>

CFloatText::CFloatText(
    CView     *anEnclosure,
    CBureaucrat  *aSupervisor,
    short    aWidth,
    short    aHeight,
    short    aHEncl,
    short    aVEncl,
    SizingOption   aHsizing,
    SizingOption   aVsizing,
    short    aLineWidth)
    CEditText(anEnclosure, aSupervisor, aWidth, aHeight, aHEncl, aVEncl,
              aHsizing, aVsizing, aLineWidth)
{
    SetValue(0.0);
    SetMaxDisplayWidth(8);
    TCL_END_CONSTRUCTOR;
}

void CFloatText::SetMaxDisplayWidth(short numChars)
{
    fMaxDisplayChars = numChars;
}

long double CFloatText::GetValue()
{
    Str255 s;
    GetTextString(s);
    if (Length(s) == 0)
        return 0.0;
    else
        return ::str2num(s);
}
void CFloatText::SetValue(long double value)
{
    Str255 newString;
    Str255 existingString;

    ValueToText(value, newString);
    GetTextString(existingString);
    if (!::EqualString(newString, existingString, TRUE, TRUE))
        SetTextString(newString);
}

void CFloatText::ValueToText(long double value, Str255 s)
{
    ::sprintf((char *) s, "%.*Lg", fMaxDisplayChars, value);
    ::c2pstr((char *) s);
}

void CFloatText::GetTextString(Str255 s)
{
    Handle h = GetTextHandle();
    *s = Min(::GetHandleSize(h), 255);
    ::BlockMove(*h, s+1, *s);
}

CFloatText.h
#pragma once
#include <CEditText.h>

class CFloatText: public CEditText {
public:
    CFloatText(
        CView *anEnclosure,
        CBureaucrat *aSupervisor,
        short aWidth,
        short aHeight,
        short aHEncl,
        short aVEncl,
        SizingOption aHSizing,
SizingOption aVSizing,
short aLineWidth);

virtual long double GetValue();
virtual void SetValue(long double value);
virtual void GetTextString(Str255 s);
virtual void SetMaxDisplayWidth(short numChars);

protected:
    virtual void ValueToText(long double value, Str255 s);
private:
    short fMaxDisplayChars;
};
// Copyright © 1993, Neil Rhodes and Julie McKeehan. All rights reserved.

#include "CQuitOnCloseWindow.h"
#include "CQuitApplicationChore.h"
#include <CApplication.h>

extern CApplication *gApplication;

CQuitOnCloseWindow::CQuitOnCloseWindow(short WINDid, Boolean aFloating, 
CDirector *aSupervisor) :
CWindow(WINDid, aFloating, aSupervisor)
{
    TCL_END_CONSTRUCTOR;
}

CQuitOnCloseWindow::~CQuitOnCloseWindow()
{
    TCL_START_DESTRUCTOR;
    gApplication->AssignUrgentChore(new CQuitApplicationChore);
}

CQuitOnCloseWindow.h

#pragma once

#include <CWindow.h>

class CQuitOnCloseWindow: public CWindow {
public:
    CQuitOnCloseWindow(short WINDid, Boolean aFloating, 
CDirector *aSupervisor);
    virtual ~CQuitOnCloseWindow();
};

CSaveRestoreSizeWindow.cp

// Copyright © 1993-94, Neil Rhodes and Julie McKeehan. All rights reserved.

#include "CSaveRestoreSizeWindow.h"
#include <Folders.h>
#include "TGrafPortSetter.h"
CSaveRestoreSizeWindow::CSaveRestoreSizeWindow(
    short    windID,
    Boolean  aFloating,
    Cdirector *aSupervisor) :
CQuitOnCloseWindow(windID, aFloating, aSupervisor)
{
    fDataFile = NULL;

    fDataFile = new CDataFile;
    SpecifyFile(fDataFile);
    SetInfoFromFile(fDataFile);
    TCL_END_CONSTRUCTOR;
}

CSaveRestoreSizeWindow::~CSaveRestoreSizeWindow()
{
    TCL_START_DESTRUCTOR;
    StoreInfoInFile(fDataFile);
    TCLForgetObject(fDataFile);
}

void CSaveRestoreSizeWindow::SpecifyFile(CFile *aFile)
{
    short vRefNum;
    long dirID;

    FailOSErr(FindFolder(kOnSystemDisk, kPreferencesFolderType,
                          kCreateFolder, &vRefNum, &dirID));
    aFile->SpecifyHFS("pCalculator Preferences", vRefNum, dirID);
}

void CSaveRestoreSizeWindow::StoreInfoInFile(CDataFile *aFile)
{
    if (!aFile->ExistsOnDisk())
        aFile->CreateNew('C++6', 'PREF');

    aFile->Open(fsWrPerm);

    Rect theWindowLocSize = macPort->portRect;

    { TGFPortSetter setPort(macPort);
::LocalToGlobal((Point *) &theWindowLocSize.top);
::LocalToGlobal((Point *) &theWindowLocSize.bottom);

TRY {
    aFile->WriteSome((Ptr) &theWindowLocSize, sizeof(theWindowLocSize));
}
CATCH {
    NO_PROPAGATE; // not writing window location isn't fatal
}
ENDTRY;
aFile->Close();

void CSaveRestoreSizeWindow::SetInfoFromFile(CDataFile *aFile)
{
    if (aFile->ExistsOnDisk()) {
        Rect portRect;

        aFile->Open(fsRdPerm);

        TRY {
            aFile->ReadSome((Ptr) &portRect, sizeof(portRect));
            Move(portRect.left, portRect.top);
            ChangeSize(portRect.right - portRect.left, portRect.bottom - portRect.top);
        }
        CATCH {
            NO_PROPAGATE; // not reading window location isn't fatal
        }
        ENDTRY;
        aFile->Close();
    }
}

CSaveRestoreSizeWindow.h
#pragma once
#include "CQuitOnCloseWindow.h"
#include <CDataFile.h>

class CSaveRestoreSizeWindow: public CQuitOnCloseWindow {
}
public:

    CSaveRestoreSizeWindow(
            short windID,
            Boolean aFloating,
            CDirector *aSupervisor);

    virtual ~CSaveRestoreSizeWindow();

protected:

    virtual void StoreinfoInFile(CDataFile *aFile);
    virtual void SetinfoFromFile(CDataFile *aFile);
    virtual void SpecifyFile(CFile *aFile);

private:

    CDataFile *fDataFile;
};

TGrafPortSetter.h

#pragma once
#include <QuickDraw.h>
#include <TCL.h>
class TGrafPortSetter TCL_AUTO_DESTRUCT_OBJECT {
public:
    TGrafPortSetter(GrafPtr newGrafPtr);
    virtual ~TGrafPortSetter();

private:
    GrafPtr fOldGrafPtr;
};

inline TGrafPortSetter::TGrafPortSetter(GrafPtr newGrafPtr) {
    ::GetPort(&fOldGrafPtr);
    ::SetPort(newGrafPtr);
    TCL_END_CONSTRUCTOR;
}

inline TGrafPortSetter::~TGrafPortSetter()
TCL_START_DESTRUCTOR;
::SetPort(fOldGrafPtr);

Chapter 21, Database Tester

Database.cp
// Copyright © 1993, Neil Rhodes and Julie McKeenan. All rights reserved.

#include "Database.h"

MCollectible::MCollectible()
{
}

MCollectible::~MCollectible()
{
}

int MCollectible::Compare(const MCollectible &item2) const
{
    if (this < &item2)
        return kItem1LessThanItem2;
    else if (this > &item2)
        return kItem1GreaterThanItem2;
    else
        return kItem1EqualItem2;
}

TDatabase::TDatabase()
{
    fRoot = NULL;
}

TDatabase::~TDatabase()
{
    delete fRoot;

672
Symantec C++ Programming for the Macintosh, 2nd Edition
void TDatabase::Insert(MCollectible* dataElement)
{
    if (fRoot == NULL)
        fRoot = new TNode(dataElement);
    else
        fRoot->Insert(dataElement);
}

MCollectible *TDatabase::Find(const MCollectible& key) const
{
    if (fRoot == NULL)
        return NULL;
    else
        return fRoot->Find(key);
}

TNode::TNode(MCollectible *value, TNode *up)
{
    fRight = fLeft = NULL;
    fUp = up;
    fValue = value;
}

TNode::~TNode()
{
    delete fRight;
    delete fLeft;
    delete fValue;
}

MCollectible *TNode::GetValue() const
{
    return fValue;
}

MCollectible *TNode::Find(const MCollectible& key) const
{
    const TNode *node = this;
    while (node != NULL) {
        ...
int compareResult = node->GetValue()->Compare(key);

if (compareResult == MCollectible::kItem1EqualItem2)
    return node->GetValue();
else if (compareResult <= MCollectible::kItem1LessThanItem2)
    node = node->fRight;
else
    node = node->fLeft;
}
return NULL;

void TNode::Insert(MCollectible *value)
{
    TNode **node = &this; // after loop: holds address within previous node
    // which points to current node
    TNode *up = NULL; // after loop: pointer to previous node

do {
    int compareResult = (*node)->GetValue()->Compare(*value);

    up = *node; // points at the current node

    if (compareResult == MCollectible::kItem1EqualItem2)
    {
        delete value;
        return;
    }
    else if (compareResult <= MCollectible::kItem1LessThanItem2)
        node = &(*node)->fRight; // advance to next node
    else
        node = &(*node)->fLeft; // advance to next node
} while (*node != NULL);
*node = new TNode(value, up);

const TNode *TNode::LeftMost() const
{
    const TNode *n = this;

    while (n->fLeft != NULL)
        n = n->fLeft;

    return n;
}
return n;
}

const TNode *TNode::RightMost() const
{
    const TNode *n = this;

    while (n->fRight != NULL)
    {
        n = n->fRight;
    }
    return n;
}

TIterator::TIterator(const TDatabase &database):
    fDatabase(database)
{
    fCurrent = NULL;
}

TNode *TIterator::GetRoot()
{
    return fDatabase.fRoot;
}

void TIterator::MoveForward()
{
    if (fCurrent == NULL)
    {
        MoveToBeginning();
        // if we have a right-hand side, the next node is the left-most of our RHS
    }
    else if (fCurrent->fRight != NULL)
    {
        fCurrent = fCurrent->fRight->LeftMost();
    }
    else if (fCurrent->fUp == NULL)  // if we don't have a RHS, and don't have an up,
    {
        fCurrent = NULL;
    }
    else {
        // the next is our first ancestor who we are on the left-hand-side of
        const TNode *last;
        do {
            last = fCurrent;
            fCurrent = last->fUp;
        } while (fCurrent != NULL && fCurrent->fRight == last);
    }
}
void TIterator::MoveBackward()
{
    if (fCurrent == NULL)
        MoveToEnd();
    // if we have a left-hand side, the next node is the right-most of our LHS
    else if (fCurrent->fLeft != NULL)
        fCurrent = fCurrent->fLeft->RightMost();
    else if (fCurrent->fUp == NULL)    // if we don't have a LHS, and don't have an up, we're done
        fCurrent = NULL;
    else {
        // the next is our first ancestor who we are on the right-hand-side of
        const TNode *last;
        do {
            last = fCurrent;
            fCurrent = last->fUp;
        } while (fCurrent != NULL && fCurrent->fLeft == last);
    }
}

int TIterator::NoMore()
{
    return fCurrent == NULL;
}

int TIterator::AtBeginning()
{
    return fCurrent != NULL && GetRoot()->LeftMost() == fCurrent;
}

int TIterator::AtEnd()
{
    return fCurrent != NULL && GetRoot()->RightMost() == fCurrent;
}

MCollectible *TIterator::GetCurrent()
{
    if (fCurrent == NULL)
        return NULL;
    else
return fCurrent->GetValue();
}

void TIterator::MoveToBeginning()
{
    if (GetRoot() == NULL)
        fCurrent = NULL;
    else
        fCurrent = GetRoot()->LeftMost();
}

void TIterator::MoveToEnd()
{
    if (GetRoot() == NULL)
        fCurrent = NULL;
    else
        fCurrent = GetRoot()->RightMost();
}

Database.h

#pragma once

class MCollectible {
public:
    enum {kItem1EqualItem2 = 0, kItem1LessThanItem2 = -1,
        kItem1GreaterThanItem2 = 1};
    MCollectible();
    virtual ~MCollectible();

    virtual int Compare(const MCollectible &item2) const;
};

class TNode {
public:
    TNode(MCollectible *value, TNode *up = NULL);
    ~TNode();

    void Insert(MCollectible *value);
    MCollectible *Find(const MCollectible &key) const;

private:
const TNode *LeftMost() const;
const TNode *RightMost() const;

MCollectible *GetVal() const;
MCollectible *fVal;
TNode *fLeft;
TNode *fRight;
TNode *fUp;
friend class TIterator;
};

class TDatabase {
public:
    TDatabase();
    virtual ~TDatabase();

    void Insert(MCollectible* dataElement);
    MCollectible *Find(const MCollectible& key) const;

private:
    TNode *fRoot;
    friend class TIterator;
};

class TIterator {
public:
    TIterator(const TDatabase &database);

    void MoveForward();
    void MoveBackward();
    int NoMore();
    int AtEnd();
    int AtBeginning();
    void MoveToBeginning();
    void MoveToEnd();
    MCollectible *GetCurrent();

protected:
    TNode *GetRoot();

private:
    const TDatabase &fDatabase;
    const TNode *fCurrent;
};
#include "Entry.h"
#include <string.h>

static char *CopyOfString(const char *s)
{
    if (s == NULL)
        s = "";
    char *copy;

    copy = new char[strlen(s)+1];
    ::strcpy(copy, s);
    return copy;
}

TNameAddressPhone::TNameAddressPhone(const char *name, const char *address,
    const char *city, const char *state, const char *zip, const char *telephone)
{
    fName = fAddress = fCity = fState = fZip = fTelephone = NULL;

    fName = CopyOfString(name);
    fAddress = CopyOfString(address);
    fCity = CopyOfString(city);
    fState = CopyOfString(state);
    fZip = CopyOfString(zip);
    fTelephone = CopyOfString(telephone);
}

TNameAddressPhone::~TNameAddressPhone()
{
    delete [] fName;
    delete [] fAddress;
    delete [] fCity;
    delete [] fState;
    delete [] fZip;
    delete [] fTelephone;
}

const char *TNameAddressPhone::GetNam...
{ 
    return fName;
}

const char *TNameAddressPhone::GetAddress() const
{
    return fAddress;
}

const char *TNameAddressPhone::GetCity() const
{
    return fCity;
}

const char *TNameAddressPhone::GetState() const
{
    return fState;
}

const char *TNameAddressPhone::GetZip() const
{
    return fZip;
}

const char *TNameAddressPhone::GetTelephone() const
{
    return fTelephone;
}

TEntry::TEntry(const char *name, const char *address, const char *city,
               const char *state, const char *zip, const char *telephone) :
    TNameAddressPhone(name, address, city, state, zip, telephone)
{
}

int TEntry::Compare(const MCollectible &item2) const
{
    int compareResult = ::strcmp(GetName(), ((TEntry & item2).GetName()));
    if (compareResult < 0)
        return kItem1LessThanItem2;
else if (compareResult > 0)
    return kItem1GreaterThanItem2;
else
    return kItem1EqualItem2;
}

#pragma once

#include "Database.h"

class TNameAddressPhone {
public:
    TNameAddressPhone(const char *name = NULL, const char *address = NULL,
                        const char *fCity = NULL, const char *fState = NULL,
                        const char *fZip = NULL, const char *fTelephone = NULL);
    virtual ~TNameAddressPhone();

    const char *GetName() const;
    const char *GetAddress() const;
    const char *GetCity() const;
    const char *GetState() const;
    const char *GetZip() const;
    const char *GetTelephone() const;

private:
    char *fName;
    char *fAddress;
    char *fCity;
    char *fState;
    char *fZip;
    char *fTelephone;
};

class TEntry: public MCollectible, public TNameAddressPhone {
public:
    TEntry(const char *name = NULL, const char *address = NULL,
                        const char *fCity = NULL, const char *fState = NULL,
                        const char *fZip = NULL, const char *fTelephone = NULL);

    virtual int Compare(const MCollectible &item2) const;
};

Appendix D: Complete Source Listings
#include "Database.h"
#include "Entry.h"
#include <iostream.h>
#include <stdio.h>

void FindAndPrint(const TDatabase &theDatabase, char *name)
{
    TEntry *entry = (TEntry *) theDatabase.Find(TEntry(name));
    if (entry != NULL)
        cout << "Name: " << entry->GetName() << " Address: " << entry->GetAddress() << '
';
}

void PrintDatabase(const TDatabase &theDatabase)
{
    printf("Database\n:"
    TIterator iter(theDatabase);
    for (iter.MoveToEnd(); !iter.NoMore(); iter.MoveBackward()) {
        TEntry *entry = (TEntry *) iter.GetCurrent();
        if (entry)
            cout << entry->GetName() << '
' <<
                entry->GetAddress() << '
' <<
                entry->GetCity() << ", " << entry->GetState() << " " << entry->GetZip() << '
' <<
                entry->GetTelephone() << '
';
    }
}

void main()
{
    TDatabase theDatabase;

    theDatabase.Insert(new TEntry("Rhodes, Neil", "1328 Clock Avenue", "Redlands", "CA", "92374", "(909) 798-5792");
    theDatabase.Insert(new TEntry("McKeehan, Julie", "1328 Clock Avenue", "Redlands", "CA", "92374", "(909) 798-5792");
    theDatabase.Insert(new TEntry("Rhodes, Alexander", "1328 Clock Avenue", "Redlands", "CA", "92374", "(909) 798-5792");
    theDatabase.Insert(new TEntry("Rhodes, Nicholas", "1328 Clock Avenue", "Redlands", "CA", "92374", "(909) 798-5792");
}
"Redlands", "CA", "92374", "(909) 798-5792");
theDatabase.Insert(new TEntry("Rhodes, Mary", "1234 Main St.",
    "Fullerton", "CA", "92635", "(714) 555-1212");
theDatabase.Insert(new TEntry("Rhodes, Fen", "123 Park Avenue",
    "Long Beach", "CA", "02138", "(310) 798-5792");
theDatabase.Insert(new TEntry("La Galy, Carolyn", "123 Longridge Avenue",
    "Santa Monica", "CA", "92374", "(818) 555-1212");
PrintDatabase(theDatabase);

FindAndPrint(theDatabase, "Rhodes, Neil");
FindAndPrint(theDatabase, "McKeehan, Julie");
FindAndPrint(theDatabase, "Rhodes, Alexander");
FindAndPrint(theDatabase, "Rhodes, Nicholas");
FindAndPrint(theDatabase, "Rhodes, Fen");
FindAndPrint(theDatabase, "Rhodes, Mary");
FindAndPrint(theDatabase, "La Galy, Carolyn");

Chapter 21, AddressBook

CAddressBookApp.cp

CAddressBookApp.c

My Application Class

Copyright © 1994 Neil Rhodes and Julie McKeehan. All rights reserved.

Generated by Visual Architect_ 7:59 AM Sun, Apr 3, 1994

This file is only generated once. You can modify it by filling
in the placeholder methods and adding any new methods you want.

If you change the name of the application, a fresh version of this
file will be generated. If you have made any changes to the file
with the old name, you will have to copy those changes to the new
file by hand.

******************************************************************************/
#include "CAddressBookApp.h"
#include "CStreamableDatabase.h"

#include <CDlg.h>
#include <TCLForceReferences.h>

TCL_DEFINE_CLASS_M1(CAddressBookApp, x_CAddressBookApp);

/*****************************************************************************/
/** C 0 N S T R U C T I O N / D E S T R U C T I O N M E T H O D S *****/
/*****************************************************************************/

ICAddressBookApp

    Initialize an Application.

/*****************************************************************************/

void CAddressBookApp::ICAddressBookApp()
{

    // The values below are:
    //
    // extraMasters - The number of additional master pointer blocks
to be allocated.
    // extraMasters
    // aRainyDayFund - The total amount of reserved memory. When
allocation digs into the rainy day fund, the
user is notified that memory is low. Set this
value to the sum of aCriticalBalance plus
aToolboxBalance plus a fudge for user warning.
// aRainyDayFund
    // aCriticalBalance - The part of the rainy day fund reserved for
// critical operations, like Save or Quit. Set
this value to the memory needed for the largest
// aCriticalBalance
possible Save plus aToolboxBalance. This
memory will only be used if SetCriticalOperation()
// aCriticalBalance
is set TRUE or if RequestMemory()/SetAllocation()
// aCriticalBalance
is set FALSE (kAllocCantFail).
// aCriticalBalance
    // aToolboxBalance - The part of the rainy day fund reserved for
// ToolBox bozos that bomb if a memory request fails.
// This memory is used unless RequestMemory()/
// SetAllocation() is set TRUE (kAllocCanFail).
// aToolboxBalance
Almost all TCL memory allocation is done with
// aToolboxBalance, and yours should be, too. The
default 2K is probably enough.

Ix_CAddressBookApp(4, 24000L, 20480L, 2048L);

    // Initialize your own application data here.

}

(mysql GENERATED 85130)

/*****************************/
ForceClassReferences {OVERRIDE}

    Reference classes that do object I/O or are created only
    by new_by_name.
    85130
/*****************************/

void CAddressBookApp::ForceClassReferences(void)
{
    x_CAddressBookApp::ForceClassReferences();
    TCL_FORCE_REFERENCE(CStreamableDatabase);
    // Insert your own class references here
    // by calling TCL_FORCE_REFERENCE for each class
    // See x_CAddressBookApp.cp
}

/**** C O M M A N D    M E T H O D S ****/
x_CAddressBookApp::DoCommand(theCommand);
break;
}

/********************************************************************************
GetDocTypeFromDialog

Get the document type from the dialog. If you have multiple
document types, you must override this function to extract
the type from the dialog state. (The dialog has been executed
and OK'd by the user.) If you do not have multiple document
types, you can remove this function.
**********************************************************************************/

OSType CAddressBookApp::GetDocTypeFromDialog(CDialogDirector *dialog)
{
    return 0;
}

CAddressBookApp.h
/********************************************************************************
CAddressBookApp.h

Header File For CAddressBookApp Application Class

Copyright © 1994 Neil Rhodes and Julie McKeehan. All rights reserved.

Generated by Visual Architect_ 7:59 AM Sun, Apr 3, 1994

This file is only generated once. You can modify it by redefining
in the placeholder definitions and adding any new methods and
data members you want.

If you change the name of the application, a fresh version of this
file will be generated. If you have made any changes to the file
with the old name, you will have to copy those changes to the new
file by hand.
**********************************************************************************/
#pragma once
#include "x_CAddressBookApp.h"
#include <CDialog.h>
class CAddressBookApp : public x_CAddressBookApp
{
public:

    TCL_DECLARE_CLASS

    void ICAAddressBookApp(void);

    virtual void        ForceClassReferences(void);
    virtual void        DoCommand(long theCommand);

    // Remove this function if you do not have
    // multiple document types

    virtual OSType      GetDocTypeFromDialog(CDialogDirector *dialog);
};

// File Types
//
// Change the #defines below to reflect the file types
// your application actually creates or opens

#define kNumFileTypes 1
#define kFileType1 'ADDR'
#define kFileType2 'TEXT'
#define kFileType3 'TEXT'
#define kFileType4 'TEXT'

CAddressBookDoc.cp

/**************************************************************************
CAddressBookDoc Document Class

Copyright © 1994 Neil Rhodes and Julie McKeehan. All rights reserved.
This file is only generated once. You can modify it by filling in the placeholder functions and adding any new functions you want.

If you change the name of the document class, a fresh version of this file will be generated. If you have made any changes to the file with the old name, you will have to copy those changes to the new file by hand.

******************************************************************************/
#include "CAddressBookDoc.h"
#include "CAddressBookPane.h"
#include "AppCommands.h" // Remove comments if DoCommand overridden
#include <CApplication.h>

TCL_DEFINE_CLASS_M1(CAddressBookDoc, x_CAddressBookDoc);

/**** CONSTRUCTION / DESTRUCTION METHODS ****/

/*****************************************************************************/
ICAddressBookDoc

Initialize the document

*****************************************************************************/

void CAddressBookDoc::ICAddressBookDoc()
{
    Ix_CAddressBookDoc();
}

/*****************************************************************************/
MakeNewContents
Create "blank" document contents. MakeNewContents is called after itsWindow is created and before it is first selected, whether or not the document uses a file.

void CAddressBookDoc::MakeNewContents()
{
    itsContents = new CStreamableDatabase;
    // Initialize document contents and itsWindow here
}

TIterator *CAddressBookDoc::CreateDatabaseIterator()
{
    TIterator *iter;
    iter = new TIterator(itsContents->GetDatabase());
    return iter;
}

void CAddressBookDoc::AddEntry(TEntry *anEntry)
{
    itsContents->GetDatabase().Insert(anEntry);
    BroadcastChange(kReasonNewEntry, anEntry);
    SetChanged(TRUE); // the document is now dirty
}

void CAddressBookDoc::ContentsToWindow(void)
{
    x_CAddressBookDoc::ContentsToWindow();
    itsGopher = itsMainPane;
    ((CAddressBookPane *) itsMainPane)->InitialSetup(this);
}

CAddressBookDoc.h
/******************************************************************************/
CAddressBookDoc.h
/******************************************************************************/

Header File For CAddressBookDoc Document Class

Copyright © 1994 Neil Rhodes and Julie McKeehan. All rights reserved.
This file is only generated once. You can modify it by extending the placeholder definitions and adding any new methods and data members you want.

If you change the name of the document class, a fresh version of this file will be generated. If you have made any changes to the file with the old name, you will have to copy those changes to the new file by hand.

******************************************************************************/

#pragma once
#include "x_CAddressBookDoc.h"
#include "CStreamableDatabase.h"
#include "Entry.h"

class CAddressBookDoc : public x_CAddressBookDoc
{
public:

    TCL_DECLARE_CLASS

    void ICAddressBookDoc(void);

    virtual void MakeNewContents(void);
    virtual void ContentsToWindow(void);

    virtual TIterator *CreateDatabaseIterator();
    virtual void AddEntry(TEntry *entry);
};

const long kReasonNewEntry = 1001;

// If you have multiple document classes, you must change
// the file type below to the appropriate type for this class.
// If not, this #define is not used.

#define CAddressBookDocFType 'ADDR'
CAddressBookPane.cp

/**************************************************************************
 CAddressBookPane Panorama/EditText Class

 Copyright © 1994 Neil Rhodes and Julie McKeenan. All rights reserved.

 Generated by Visual Architect_ 8:16 AM Sun, Apr 3, 1994

 This file is only generated once. You can modify it by filling
 in the placeholder functions and adding any new functions you want.

 If you change the name of the document class, a fresh version of this
 file will be generated. If you have made any changes to the file
 with the old name, you will have to copy those changes to the new
 file by hand.

**************************************************************************/

#include "CAddressBookPane.h"
#include "CAddressBookDoc.h"
#include <CStaticText.h>
#include <CBartender.h>
extern CBartender *gBartender;
#include <Commands.h>
#include "AppCommands.h" // Uncomment to include VA-defined commands

TCL_DEFINE_CLASS_D1(CAddressBookPane, x_CAddressBookPane);

void CAddressBookPane::UpdateMenus()
{
    x_CAddressBookPane::UpdateMenus();

    // now all five menu items are enabled. Disable as appropriate

    if (fAddressBookIterator->AtEnd())
        gBartender->DisableCmd(cmdGoForward);
    if (fAddressBookIterator->AtBeginning())
        gBartender->DisableCmd(cmdGoBackward);
    if (fAddressBookIterator->NoMore()) {

gBartender->DisableCmd(cmdGoToEnd);
gBartender->DisableCmd(cmdGoToBeginning);
}

void CAddressBookPane::InitialSetup(CAddressBookDoc *doc)
{
    fAddressBookDoc = doc;
    DependUpon(fAddressBookDoc);

    fAddressBookIterator = doc->CreateDatabaseIterator();
    DoCmdGoToBeginning();
}

void CAddressBookPane::DoCmdGoToBeginning(void)
{
    fAddressBookIterator->MoveToBeginning();
    UpdateEntryData();
}

void CAddressBookPane::DoCmdGoToEnd(void)
{
    fAddressBookIterator->MoveToEnd();
    UpdateEntryData();
}

void CAddressBookPane::DoCmdGoForward(void)
{
    fAddressBookIterator->MoveForward();
    UpdateEntryData();
}

void CAddressBookPane::DoCmdGoBackward(void)
{
    fAddressBookIterator->MoveBackward();
    UpdateEntryData();
}

void CAddressBookPane::UpdateEntryData(void)
{
    const TEntry *entry = (TEntry *) fAddressBookIterator->GetCurrent();
    char name[256] = "";
    char address[256] = "";
char city[256] = "";
char state[256] = "";
char zip[256] = "";
char phone[256] = "";

if (entry) {
    ::strcpy(name, entry->GetName());
    ::strcpy(address, entry->GetAddress());
    ::strcpy(city, entry->GetCity());
    ::strcpy(state, entry->GetState());
    ::strcpy(zip, entry->GetZip());
    ::strcpy(phone, entry->GetTelephone());
}

GetAddressBookDoc()->fAddressBook_Name->SetTextPtr(name, (long) strlen(name));
GetAddressBookDoc()->fAddressBook_Address->SetTextPtr(address, (long) strlen(address));
GetAddressBookDoc()->fAddressBook_City->SetTextPtr(city, (long) strlen(city));
GetAddressBookDoc()->fAddressBook_State->SetTextPtr(state, (long) strlen(state));
GetAddressBookDoc()->fAddressBook_Zip->SetTextPtr(zip, (long) strlen(zip));
GetAddressBookDoc()->fAddressBook_Telephone->SetTextPtr(phone, (long) strlen(phone));

CAddressBookDoc *CAddressBookPane::GetAddressBookDoc()
{
    return fAddressBookDoc;
}

void CAddressBookPane::ProviderChanged(CCollaborator *aProvider, long reason, void *info)
{
    if (reason == kReasonNewEntry) {
        TEntry *theEntry = (TEntry *) info;

        // Display the new entry
        fAddressBookIterator->MoveToBeginning();
        for (; !fAddressBookIterator->NoMore();
            fAddressBookIterator->MoveForward()) {
            if (theEntry == (TEntry *) fAddressBookIterator->GetCurrent())
                break;
        }
        UpdateEntryData();
    }
}
CAddressBookPane.h

Header for CAddressBookPane Panorama/EditText Class

Copyright © 1994 Neil Rhodes and Julie McKeehan. All rights reserved.

Generated by Visual Architect_ 8:16 AM Sun, Apr 3, 1994

This file is only generated once. You can modify it by filling in the placeholder functions and adding any new functions you want.

If you change the name of the document class, a fresh version of this file will be generated. If you have made any changes to the file with the old name, you will have to copy those changes to the new file by hand.

******************************************************************************/

#pragma once

#include "x_CAddressBookPane.h"
#include "Database.h"
#include "CAddressBookDoc.h"

class CAddressBookPane : public x_CAddressBookPane
{
    public:

    TCL_DECLARE_CLASS

    virtual void UpdateMenus();
    virtual void DoCmdGoForward(void);
    virtual void DoCmdGoBackward(void);
    virtual void DoCmdGoToBeginning(void);
    virtual void DoCmdGoToEnd(void);
    virtual void InitialSetup(CAddressBookDoc *doc);

    protected:
virtual void ProviderChanged(CCollaborator *aProvider, long reason, void* info);

private:
    virtual void UpdateEntryData(void);
    Titerator *fAddressBookIterator;
    CAddressBookDoc *fAddressBookDoc;
    CAddressBookDoc *GetAddressBookDoc();

};

CNewEntryDialog.cp

CNewEntryDialog Dialog Director Class

Copyright © 1994 Neil Rhodes and Julie McKeenan. All rights reserved.

Generated by Visual Architect_ 3:16 PM Sun, Apr 3, 1994

This file is only generated once. You can modify it by filling
in the placeholder functions and adding any new functions you want.

If you change the name of the dialog class, a fresh version of this
file will be generated. If you have made any changes to the file
with the old name, you will have to copy those changes to the new
file by hand.

******************************************************************************/

#include "CNewEntryDialog.h"

#include "NewEntryDialogItems.h"

#include "CAddressBookDoc.h"
#include "Entry.h"
// #include "AppCommands.h"       // Remove comments if DoCommand overridden

#include <CIntegerText.h>
#include <CControl.h>
#include <CPopupMenu.h>
#include <CPopupPane.h>
#include <CArrayPane.h>
#include <CIconButton.h>
#include <CPictureButton.h>
#include <CSwissArmyButton.h>
#include <CApplication.h>
#include <Constants.h>
#include <CDecorator.h>
#include <CDesktop.h>
#include <CDirectorOwner.h>
#include <CFile.h>
#include <TBUtilities.h>
#include <CWindow.h>

extern CApplication *gApplication; /* The application */
extern CDecorator *gDecorator; /* Decorator for arranging windows */
extern CDesktop *gDesktop; /* The visible Desktop */

TCL_DEFINE_CLASS_M1(CNewEntryDialog, x_CNewEntryDialog);

**** CONSTRUCTION/DESTRUCTION METHODS ****/

/********************************************
CNewEntryDialog

Initialize the dialog

*********************************************/

void CNewEntryDialog::ICNewEntryDialog(CDirectorOwner *aSupervisor)
{
    // Initialize data members that must be set up before
    // BeginData is called here

    x_CNewEntryDialog::Ix_CNewEntryDialog(aSupervisor);

    Initialize the dialog

    ********************************************/
// Initialize any other data members here

}  

/******************************************************************************
BeginData

Set up initial pane values
******************************************************************************/

void CNewEntryDialog::BeginData(CNewEntryDialogData *initial)
{
    // Base class calls BeginData once after the window is created
    // to gather the initial values for the dialog panes. Note that
    // BeginData is called *before* IX_CNewEntryDialog returns. The
    // initial struct is cleared to zeros.
    
    // Calling CollectPaneValues copies the initial values you set
    // in Visual Architect from the panes. This lets you
    // use these values as the starting point every time the dialog
    // is run. If you want to use values determined by your program
    // instead, omit this call.
    
    CollectPaneValues(*initial);
    
    // Add code to replace some or all of the initial pane values below.

}

/******************************************************************************
EndData

Receive final values from the dialog
******************************************************************************/

void CNewEntryDialog::EndData(CNewEntryDialogData *final)
{
    // The values of all panes are returned by this function,
which is called just before Close for a modeless dialog, or just before returning from DoModalDialog.

If DoModalDialog returns cmdCancel, EndData is called with the values initially supplied to BeginData, allowing you to back out any intermediate changes made in response to UpdateData. If you do not use UpdateData, you can test the value of dismissCmd to see whether to respond to EndData.

```c
if (dismissCmd != cmdCancel && Length(final->fNewEntryDialog_Name) > 0) {
    TEntry *newEntry;

    newEntry = new TEntry(  
        p2cstr(final->fNewEntryDialog_Name),  
        p2cstr(final->fNewEntryDialog_Address),  
        p2cstr(final->fNewEntryDialog_City),  
        p2cstr(final->fNewEntryDialog_State),  
        p2cstr(final->fNewEntryDialog_Zip),  
        p2cstr(final->fNewEntryDialog_Telephone)
    );

    ((CAddressBookDoc *) itsSupervisor)->AddEntry(newEntry);
}
```

CNewEntryDialog.h

**************************************************************************
CNewEntryDialog.h

CNewEntryDialog Dialog Director Class

Copyright © 1994 Neil Rhodes and Julie McKeehan. All rights reserved.

Generated by Visual Architect_ 3:16 PM Sun, Apr 3, 1994

This file is only generated once. You can modify it by filling in the placeholder functions and adding any new functions you want.

If you change the name of the dialog class, a fresh version of this file will be generated. If you have made any changes to the file
with the old name, you will have to copy those changes to the new file by hand.

******************************************************************************/
#pragma once
#include "x_CNewEntryDialog.h"

class CDirectorOwner;

class CNewEntryDialog : public x_CNewEntryDialog
{
public:

    TCL_DECLARE_CLASS

    // Insert your own public data members here

    void ICNewEntryDialog(CDirectorOwner *aSupervisor);

protected:

    virtual void BeginData(CNewEntryDialogData *initial);
    virtual void EndData(CNewEntryDialogData *final);

};

CStreamableDatabase.cp
/*****************************************************************************/
CStreamableDatabase.cp

Copyright © 1994 Neil Rhodes and Julie Mc Keehan. All rights reserved.

******************************************************************************/
#include "CStreamableDatabase.h"
#include "Entry.h"
#include <CStream.h>
TCL_DEFINE_CLASS_D0(CStreamableDatabase);

TDatabase &CStreamableDatabase::GetDatabase()
{
    return fDatabase;
}

void CStreamableDatabase::PutTo(CStream& stream)
{
    long total = 0;
    Titerator iter(fDatabase);
    for (iter.MoveToBeginning(); !iter.NoMore();
        iter.MoveForward())
    {
        total++;
        stream << total;
        for (iter.MoveToBeginning(); !iter.NoMore();
            iter.MoveForward()) {
            TEntry *curEntry = (TEntry *) iter.GetCurrent();
            stream << curEntry->GetName();
            stream << curEntry->GetAddress();
            stream << curEntry->GetCity();
            stream << curEntry->GetState();
            stream << curEntry->GetZip();
            stream << curEntry->GetTelephone();
        }
    }
}

void CStreamableDatabase::GetFrom(CStream& stream)
{
    long total;
    stream >> total;
    for (long i = 0; i < total; i++) {
        char name[256];
        char address[256];
        char city[256];
        char state[256];
        char zip[256];
        // Code to read and store entries
char telephone[256];

stream >> name;
stream >> address;
stream >> city;
stream >> state;
stream >> zip;
stream >> telephone;

TEntry *newEntry = new TEntry(name, address, city,
                               state, zip, telephone);
fDatabase.Insert(newEntry);

}
CStream_CStreamableDatabase.cpp

#include "CStream.h"
#include "CStreamableDatabase.h"

#pragma template_access public

#pragma template PutObject(CStream&, CStreamableDatabase*)
#pragma template GetObject(CStream&, CStreamableDatabase*&)

#include "CStream.tem"

Database.cp

// Copyright © 1993, Neil Rhodes and Julie McKeenan. All rights reserved.

#include "Database.h"

MCollectible::MCollectible()
{
}

MCollectible::~MCollectible()
{
}

int MCollectible::Compare(const MCollectible &item2) const
{
    if (this < &item2)
        return kItem1LessThanItem2;
    else if (this > &item2)
        return kItem1GreaterThanItem2;
    else
        return kItem1EqualItem2;
}

TDatabase::TDatabase()
{
    fRoot = NULL;
}

TDatabase::~TDatabase()
void TDatabase::Insert(MCollectible* dataElement)
{
    if (fRoot == NULL)
        fRoot = new TNode(dataElement);
    else
        fRoot->Insert(dataElement);
}

MCollectible *TDatabase::Find(const MCollectible& key) const
{
    if (fRoot == NULL)
        return NULL;
    else
        return fRoot->Find(key);
}

TNode::TNode(MCollectible *value, TNode *up)
{
    fRight = fLeft = NULL;
    fUp = up;
    fValue = value;
}

TNode::~TNode()
{
    delete fRight;
    delete fLeft;
    delete fValue;
}

MCollectible *TNode::GetValue() const
{
    return fValue;
}

MCollectible *TNode::Find(const MCollectible& key) const
const TNode* node = this;

while (node != NULL) {
    int compareResult = node->GetValue()->Compare(key);

    if (compareResult == MCollectible::kItem1EqualItem2)
        return node->GetValue();
    else if (compareResult <= MCollectible::kItem1LessThanItem2)
        node = node->fRight;
    else
        node = node->fLeft;
}
return NULL;

void TNode::Insert(MCollectible* value)
{
    TNode **node = &this;  // after loop: holds address within previous node
    // which points to current node
    TNode *up = NULL;       // after loop: pointer to previous node

do {
    int compareResult = (*node)->GetValue()->Compare(*value);

    up = *node;           // points at the current node

    if (compareResult == MCollectible::kItem1EqualItem2)
    {  
delete value;
        return;
    }
    else if (compareResult <= MCollectible::kItem1LessThanItem2)
        node = &(*node)->fRight;   // advance to next node
    else
        node = &(*node)->fLeft;    // advance to next node
} while (*node != NULL);
*node = new TNode(value, up);

const TNode* TNode::LeftMost() const
{

const TNode *n = this;

while (n->fLeft != NULL)
    n = n->fLeft;
return n;
}

const TNode *TNode::RightMost() const
{
    const TNode *n = this;

    while (n->fRight != NULL)
        n = n->fRight;
    return n;
}

TIterator::TIterator(const TDatabase &database):
    fDatabase(database)
{
    fCurrent = NULL;
}

TNode *TIterator::GetRoot()
{
    return fDatabase.fRoot;
}

void TIterator::MoveForward()
{
    if (fCurrent == NULL)
        MoveToBeginning();
    // if we have a right-hand side, the next node is the left-most of our RHS)
    else if (fCurrent->fRight != NULL)
        fCurrent = fCurrent->fRight->LeftMost();
    else if (fCurrent->fUp == NULL) // if we don't have a RHS, and don't have an up, we're done
        fCurrent = NULL;
    else {
        // the next is our first ancestor who we are on the left-hand-side of
        const TNode *last;
        do {
            last = fCurrent;
            fCurrent = last->fUp;
    }
void TIterator::MoveBackward()
{
    if (fCurrent == NULL)
        MoveToEnd();
    // if we have a left-hand side, the next node is the right-most of our LHS
    else if (fCurrent->fLeft != NULL)
        fCurrent = fCurrent->fLeft->RightMost();
    else if (fCurrent->fUp != NULL) // if we don't have a LHS, and don't have an up, we're done
        fCurrent = NULL;
    else {
        // the next is our first ancestor who we are on the right-hand-side of
        const TNode *last;
        do {
            last = fCurrent;
            fCurrent = last->fUp;
        } while (fCurrent != NULL && fCurrent->fLeft == last);
    }
}

int TIterator::NoMore()
{
    return fCurrent == NULL;
}

int TIterator::AtBeginning()
{
    return fCurrent != NULL && GetRoot()->LeftMost() == fCurrent;
}

int TIterator::AtEnd()
{
    return fCurrent != NULL && GetRoot()->RightMost() == fCurrent;
}

MCollectible *TIterator::GetCurrent()
{
    if (fCurrent == NULL)
return NULL;
else
    return fCurrent->GetValue();
}

void TIterator::MoveToBeginning()
{
    if (GetRoot() == NULL)
        fCurrent = NULL;
    else
        fCurrent = GetRoot()->LeftMost();
}

void TIterator::MoveToEnd()
{
    if (GetRoot() == NULL)
        fCurrent = NULL;
    else
        fCurrent = GetRoot()->RightMost();
}

Database.h

#pragma once

class MCollectible {
public:
    enum {kItem1EqualItem2 = 0, kItem1LessThanItem2 = -1,
          kItem1GreaterThanItem2 = 1};
    MCollectible();
    virtual ~MCollectible();

    virtual int Compare(const MCollectible &item2) const;
};

class TNode {
public:
    TNode(MCollectible *value, TNode *up = NULL);
    ~TNode();

    void Insert(MCollectible *value);
MCollectible *Find(const MCollectible &key) const;

private:
    const TNode *LeftMost() const;
    const TNode *RightMost() const;

MCollectible *GetValue() const;

MCollectible *fValue;
TNode *fLeft;
TNode *fRight;
TNode *fUp;
friend class TIterator;
};

class TDatabase {
public:
    TDatabase();
    virtual ~TDatabase();

    void Insert(MCollectible* dataElement);
    MCollectible *Find(const MCollectible& key) const;

private:
    TNode *fRoot;
    friend class TIterator;
};

class TIterator {
public:
    TIterator(const TDatabase &database);

    void MoveForward();
    void MoveBackward();
    int NoMore();
    int AtEnd();
    int AtBeginning();
    void MoveToBeginning();
    void MoveToEnd();
    MCollectible *GetCurrent();

protected:
    TNode *GetRoot();

private:
const TDatabase &fDatabase;
cnst TNode *fCurrent;
};

Entry.cp

// Copyright © 1993, Neil Rhodes and Julie McKeegan. All rights reserved.

#include "Entry.h"
#include <string.h>

static char *CopyOfString(const char *s)
{
    if (s == NULL)
        s = ";
    char *copy = new char[strlen(s)+1];
    ::strcpy(copy, s);
    return copy;
}

TNameAddressPhone::TNameAddressPhone(const char *name, canst char *address,
cnst char *city, canst char *state, canst char *zip, canst char *telephone)
{
    fName = fAddress = fCity = fState = fZip = fTelephone = NULL;

    fName = CopyOfString(name);
    fAddress = CopyOfString(address);
    fCity = CopyOfString(city);
    fState = CopyOfString(state);
    fZip = CopyOfString(zip);
    fTelephone = CopyOfString(telephone);
}

TNameAddressPhone::~TNameAddressPhone()
{
    delete [] fName;
    delete [] fAddress;
    delete [] fCity;
delete [] fState;
delete [] fZip;
delete [] fTelephone;
}

const char *TNameAddressPhone::GetName() const
{
    return fName;
}

const char *TNameAddressPhone::GetAddress() const
{
    return fAddress;
}

const char *TNameAddressPhone::GetCity() const
{
    return fCity;
}

const char *TNameAddressPhone::GetState() const
{
    return fState;
}

const char *TNameAddressPhone::GetZip() const
{
    return fZip;
}

const char *TNameAddressPhone::GetTelephone() const
{
    return fTelephone;
}

TEntry::TEntry(const char *name, const char *address, const char *city,
               const char *state, const char *zip, const char *telephone) :
TNameAddressPhone(name, address, city, state, zip, telephone)
{
}

int TEntry::Compare(const MCollectible &item2) const


```cpp
{
    int compareResult = ::strcmp(GetName(), ((TEntry &)item2).GetName());

    if (compareResult < 0)
        return kItem1LessThanItem2;
    else if (compareResult > 0)
        return kItem1GreaterThanItem2;
    else
        return kItem1EqualItem2;
}

Entry.h

#pragma once

#include "Database.h"

class TNameAddressPhone {
public:
    TNameAddressPhone(const char *name = NULL,
                      const char *address = NULL,
                      const char *fCity = NULL,
                      const char *fState = NULL,
                      const char *fZip = NULL,
                      const char *fTelephone = NULL);
    virtual ~TNameAddressPhone();

    const char *GetName() const;
    const char *GetAddress() const;
    const char *GetCity() const;
    const char *GetState() const;
    const char *GetZip() const;
    const char *GetTelephone() const;

private:
    char *fName;
    char *fAddress;
    char *fCity;
    char *fState;
    char *fZip;
    char *fTelephone;
};
```
class TEntry: public MCollectible, public TNameAddressPhone {
public:
    TEntry(const char *name = NULL, const char *address = NULL,
           const char *fCity = NULL, const char *fState = NULL,
           const char *fZip = NULL, const char *fTelephone = NULL);
    virtual int Compare(const MCollectible &item2) const;
};

ItsContentsClass.h
/****************************************************************************
ItsContentsClass.h

Define class of CSaver itsContents variable

Copyright © 1994 Neil Rhodes and Julie McKeehan. All rights reserved.

Generated by Visual Architect_ 7:47 AM Sun, Apr 3, 1994

This file is only generated once.

Change CCollaborator below to the actual class of
your itsContents object
******************************************************************************/
#pragma once
#include "CStreamableDatabase.h"
#define ITSCONTENTSCLASS CStreamableDatabase
#ifdef GENERATE_TEMPLATE
#pragma template CSaver<CStreamableDatabase>
#endif

#pragma once
#include "CStreamableDatabase.h"
#define ITSCONTENTSCLASS CStreamableDatabase
#ifdef GENERATE_TEMPLATE
#pragma template CSaver<CStreamableDatabase>
#endif
#include "CAddressBookApp.h"

void main()
{
    CAddressBookApp *application = new CAddressBookApp;
    application->IGAddressBookApp();
    application->Run();
    application->Exit();
}
// Use TCLGetItemPointer (ViewUtilities) to convert these
// item numbers to a pointer to the named item.

enum
{
    NewEntryDialog_Begin_,
    kNewEntryDialog_Stat1 = 1,
    kNewEntryDialog_Stat1ID = 4097L,
    kNewEntryDialog_Stat2 = 2,
    kNewEntryDialog_Stat2ID = 4098L,
    kNewEntryDialog_Stat3 = 3,
    kNewEntryDialog_Stat3ID = 4099L,
    kNewEntryDialog_Stat4 = 4,
    kNewEntryDialog_Stat4ID = 4100L,
    kNewEntryDialog_Stat5 = 5,
    kNewEntryDialog_Stat5ID = 4101L,
    kNewEntryDialog_Stat6 = 6,
    kNewEntryDialog_Stat6ID = 4102L,
    kNewEntryDialog_Butn7 = 7,
    kNewEntryDialog_Butn7ID = 4103L,
    kNewEntryDialog_Butn8 = 8,
    kNewEntryDialog_Butn8ID = 4104L,
    kNewEntryDialog_Name = 9,
    kNewEntryDialog_NameID = 4105L,
    kNewEntryDialog_Adddress = 10,
    kNewEntryDialog_AddressID = 4106L,
    kNewEntryDialog_City = 11,
    kNewEntryDialog_CityID = 4107L,
    kNewEntryDialog_State = 12,
    kNewEntryDialog_StateID = 4108L,
    kNewEntryDialog_Zip = 13,
    kNewEntryDialog_ZipID = 4109L,
    kNewEntryDialog_Telephone = 14,
    kNewEntryDialog_TelephoneID = 4110L,
    kNewEntryDialog_Stat15 = 15,
    kNewEntryDialog_Stat15ID = 4111L,
    NewEntryDialog_End_
};
Appendix B, PowersOfTwo

Main.cp

#include <iostream.h>
#include "Power.h"

void main()
{
   cout << "2 to the power 8 = " << Power(2, 8) << 'n';
}

Power.cp

// a comment
#include "Power.h"

int Power(int number, int powerToRaise)
{
   int returnValue = 1;

   for (int i = 0; i < powerToRaise; i++)
      returnValue *= number;
   return returnValue;
}

Power.h

int Power(int number, int powerToRaise);
Annotated Bibliography

This list includes the best programming resources for C, C++, object-oriented development, and the Macintosh environment.

C Reference Material

The C Programming Language, 2nd Edition

Referred to as the white bible, the following is the classic text on C. If you are a C programmer, you should have it. It begins, of course, with the "Hello, world" program.

Harbison and Steele cover the intricacies of the language with emphasis on the hidden areas in C that cause problems with particular compilers. This book certainly can help you learn to write portable C code.


C++ Reference Material

C++ Programming Style
The following book explores a number of C++ programs that have appeared in various books and articles. It gives you a useful analysis of their design, implementation, and efficiency.


Advanced C++ Programming Styles and Idioms
Coplien is not for the fainthearted because each chapter is so dense it could be expanded into a book of its own. It is especially good for its presentation of the weird and wonderful things you can do with C++, such as how to make C++ dynamic like LISP.


The Annotated C++ Reference Manual
If you want to know whether a particular construct is allowed in C++, you can try the compiler or check the ARM. Of course, the one true answer is found in the ARM. Every C++ programmer should have a copy of this book, just as every household should have a dictionary—you don’t need to know the whole book, but it’s nice to have around for when definitive answers are needed.

**Effective C++: 50 Specific Ways to Improve Your Programs and Designs**

*Effective C++* is the best real world book on C++. Although it doesn’t teach you C++ features (it assumes that you already know them), it does teach you how to use them and how they interact with one another. If you are a C++ programmer, you must have this book; you will refer to it often. In addition to being useful, it also is funny and entertaining.


**A C++ Toolkit**

Shapiro presents a useful set of library classes (binary trees, hash tables, and lists) that you can use in your own programs.


**Object-Oriented Programming**

**Object-Oriented Software Construction**

Meyer describes how OOP can be used to construct solid software. No existing language suited his needs, so he designed his own, Eiffel. The book describes why Eiffel is designed the way it is, including its support for assertions, preconditions, and postconditions. Although support is not provided explicitly for these, you still can implement them in C++. Furthermore, the programming-by-contract paradigm he presents is well worth knowing.


**Object-Oriented Design**

**Object-Oriented Design: With Applications**

This is *the* book on object-oriented design. It presents not only instructions on how to design, but a number of full-sized applications in different languages: Smalltalk, CLOS, Object Pascal (on the Macintosh, using MacApp), and C++.

## Programming

### Structure & Interpretation of Computer Programs
Abelson and Sussman is the textbook for the introductory programming class at MIT. Don’t let that stop you, however. The book uses Scheme, a dialect of LISP, and does so much with it that you may finally overcome your LISP fears. It is not a Scheme textbook, however, but rather a tutorial on many different aspects of computer science. This is not something you can read in an afternoon, but it is highly recommended.


### Programming Pearls, and More Programming Pearls
Jon Bentley had a column in “Communications of the ACM,” the monthly publication of the Association for Computing Machinery (ACM). The ACM is the professional organization for computing professionals, so you should consider joining. *Programming Pearls* and *More Programming Pearls* are collections of the columns he wrote there. Each chapter is small, but provides some fresh ideas.


### Writing Efficient Programs
Do not optimize another program until you have read *Writing Efficient Programs*. Among other things, you learn to optimize only after your program is correct. You also are convinced that you need to do measurements to determine the effect of your optimizations.

**Software Tools**

Two books—one in RATFOR (a preprocessed FORTRAN) and one in Pascal—demonstrate how to build interactive tools. If you are familiar with UNIX, you will be right at home with this tool philosophy.


**TeX (Computers & Typesetting Ser.: Vol. B)**

Hats off to Knuth, who has published the source code to the TeX typesetting program. Read this book to see how an expert computer scientist writes (and documents) code.


**Books by Gerald Weinberg**

Any book by Weinberg is well worth reading. These three are but a sampling of his work. *Handbook of Walkthroughs, Inspections & Technical Reviews* describe how to conduct code walkthroughs and why they are important. As the book points out so well, having others pore over your code with a fine-tooth comb is the single most effective way to catch errors.

*Becoming a Technical Leader: An Organic Problem-Solving Approach* shows you how to become a problem solver and how to rally others around you to help generate the solutions. *Understanding the Professional Programmer* is geared more for managers than for those they manage, but is an entertaining look at programmers.


**Macintosh**

*Inside Macintosh*

If you don't have it already, you need to know that you cannot be without the series *Inside Macintosh*. The books are the official programming guides to Macintosh.


*How to Write Macintosh Software: The Debugging Reference for the Macintosh*

Knaster details how the operating system works and how to debug your code in relation to it.

Chapter 2, MultipleFiles

These are the files used to create the MultipleFiles project described in Chapter 2. The following files are located in the Average subfolder:

- Average.cp contains the source code for Average.
- Average.h contains the declaration of Average.

The following files are located in the Factorial subfolder:

- Factorial.cp contains the source code for Factorial.
- Factorial.h contains the declaration of Factorial.

The MultipleFiles folder contains three directories—Average, Factorial, and Reverse—and the following two files:

- Main.cp contains the main program.
- MultipleFiles.pi is the project file.

The Reverse subfolder contains the following files:

- Reverse.cp contains the source code for Reverse.
- Reverse.h contains the declaration of Reverse.
Chapter 3, Errors

These are the files used to create the Errors project described in Chapter 3. Within the Average subfolder are the following files:

- Average.cp contains the source code for Average.
- Average.h contains the declaration of Average.

The following files are located in the Factorial subfolder:

- Factorial.cp contains the source code for Factorial.
- Factorial.h contains the declaration of Factorial.

The Errors folder contains three directories—Average, Factorial, and Reverse—and the following two files:

- Main.cp contains the main program.
- Errors.n is the project file.

Within the Reverse subfolder are the following files:

- Reverse.cp contains the source code for Reverse.
- Reverse.h contains the declaration of Reverse.

Chapter 4, Binary Tree

These are the files used to create the Binary Tree project described in Chapter 4:

- Main.cp contains the main program.
- BinaryTree.n is the project file.

Chapter 5, Alphabet

These are the files used to create the Alphabet project described in Chapter 5:

- Alphabet.r contains all the resources in Rez format. It can be added to the project.
- Alphabet.n is the project file.
- Alphabet.n.rsrc contains all the resources.
AlphabetStrings.rsrc contains all the resources in binary format. It can be added to the project.

Main.cp contains the main program.

ResourceDefinitions.h contains the constants for resource IDs.

Chapter 7, Precompiled Headers

These are the files used to create the two Precompiled headers projects described in Chapter 7. The following files are located in the #1 subfolder:

- chap7,Proj1.π is the project file.
- Foo.cp contains the main program.

Within the #2 subfolder are the following files:

- Bar.h contains the definition of kBar.
- chap7,Proj2.π is the project file.

This is the project file:

- Foo.h contains the definition of kFoo.
- FooBar++ is a precompiled header file.
- FooBar.cp contains the source code to create the precompiled header file.
- Main.cp contains the main program.

Chapter 16, RandomRectangle

These are the project files used to create the RandomRectangle application described in Chapter 16:

- CRandomRectanglePane.cp contains the source code for CRandomRectanglePane.
- CRandomRectanglePane.h contains the declaration of CRandomRectanglePane.
- CRectangleApp.h contains the declaration of CRectangleApp.
- Rectangle++.π is the project file.
• Rectangle.cp contains the main program.
• Rectangle.r contains all the resources.
• TRandom.cp contains the code for TRandom.
• TRandom.h contains the declaration of TRandom.

Chapter 17, TextTyper

These are the project files used to create the TextTyper application described in Chapter 17:

• CTextApp.cp contains the source code for CTextApp.
• CTextApp.h contains the declaration of CTextApp.
• CTextDoc.cp contains the source code for CTextDoc.
• CTextDoc.h contains the declaration of CTextDoc.
• Text++.t is the project file.
• Text.cp contains the main program.
• Text.r contains all the resources.

Chapter 18, PICTPeeker

These are the project files used to create the PICTPeeker application described in Chapter 18:

• CAboutBoxApplication.cp contains the source code for CAboutBoxApplication.
• CAboutBoxApplication.h contains the declaration of CAboutBoxApplication.
• CCutCopyPastePicture.cp contains the source code for CCutCopyPastePicture.
• CCutCopyPastePicture.h contains the declaration of CCutCopyPastePicture.
• CPictureApp.cp contains the source code for CPictureApp.
• CPictureApp.h contains the declaration of CPictureApp.
• CPictureClipboardTask.cp contains the source code for CPictureClipboardTask.
• CPictureClipboardTask.h contains the declaration of CPictureClipboardTask.
• CPictureDoc.cp contains the source code for CPictureDoc.
Chapter 19, Tick-Tock

These are the project files used to create the Tick-Tock application described in Chapter 19:

- CAboutBoxApplication.cp contains the source code for CAboutBoxApplication.
- CAboutBoxApplication.h contains the declaration of CAboutBoxApplication.
- CClockApp.cp contains the source code for CClockApp.
- CClockApp.h contains the declaration of CClockApp.
- CClockPane.cp contains the source code for CClockPane.
- CClockPane.h contains the declaration of CClockPane.
- CDawdleBureaucratChore.cp contains the source code for CDawdleBureaucratChore.
- CDawdleBureaucratChore.h contains the declaration of CDawdleBureaucratChore.
- Clock.cp contains the main program.
Clock.r contains all the resources.

CQuitOnCloseWindow.cp contains the source code for CQuitOnCloseWindow.

CQuitOnCloseWindow.h contains the declaration of CQuitOnCloseWindow.

CSaveRestoreSizeWindow.cp contains the source code for CSaveRestoreSizeWindow.

CSaveRestoreSizeWindow.h contains the declaration of CSaveRestoreSizeWindow.

T GrafPortSetter.h contains the code (inlined) and the declaration of T GrafPortSetter.

TickTock++.7t is the project file.

Utilities.cp contains the code for utility routines.

Utilities.h contains the declaration of utility routines.

Chapter 20, C++ Calc

These are the project files used to create the C++ Calc application described in Chapter 20:

C++ Calc++.7t is the project file.

CAboutBoxApplication.cp contains the source code for CAboutBoxApplication.

CAboutBoxApplication.h contains the declaration of CAboutBoxApplication.

Calculator.cp contains the source code for Calculator.

Calculator.h contains the declaration of Calculator.

Calculator.r contains all the resources.

CalculatorMain.cp contains the main program.

CButtonWithFontAndSize.cp contains the source code for CButtonWithFontAndSize.

CButtonWithFontAndSize.h contains the declaration of CButtonWithFontAndSize.

CCalculatorApp.cp contains the source code for CCalculatorApp.

CCalculatorApp.h contains the declaration of CCalculatorApp.

CCalculatorPane.cp contains the source code for CCalculatorPane.

CCalculatorPane.h contains the declaration of CCalculatorPane.

CFloatText.cp contains the source code for CFloatText.

CFloatText.h contains the declaration of CFloatText.
Chapter 21, Database Tester

These are the project files used to create the Database Tester application described in Chapter 21:

- Database.cp contains the source code for TDatabase.
- Database.h contains the declaration of TDatabase.
- Database TESTer.n is the project file.
- Entry.cp contains the source code for TEntry.
- Entry.h contains the declaration of TEntry.
- Main.cp contains the main program.

Chapter 21, AddressBook

These are the project files used to create the AddressBook application described in Chapter 21:

- AddressBookFull.n is the project file.
- Project Resources.rsrc contains the application resources.
- Visual Architect.rsrc contains all the user interface resources.
- AddressBookItems.h contains constants for the main document window.
- AppCommands.h contains constants for the application commands.
- CAddressBookApp.cp contains the source code for CAddressBookApp.
- CAddressBookApp.h contains the declaration for CAddressBookApp.
- CAddressBookDoc.cp contains the source code for CAddressBookDoc.
- CAddressBookDoc.h contains the declaration for CAddressBookDoc.
- CAddressBookPane.cp contains the source code for CAddressBookPane.
- CAddressBookPane.h contains the declaration for CAddressBookPane.
- CNewEntryDialog.cp contains the source code for CNewEntryDialog.
- CNewEntryDialog.h contains the declaration for CNewEntryDialog.
- CSaver_class.cpp contains the template code to generate a class saver.
- CStreamableDatabase.cp contains the source code for CStreamableDatabase.
- CStreamableDatabase.h contains the declaration for CStreamableDatabase.
- CStream_CStreamableDatabase.cpp contains code to prevent unnecessary loading of database template code.
- Database.cp contains the source code for Database.
- Database.h contains the declaration for Database.
- Entry.cp contains the source code for Entry.
- Entry.h contains the declaration for Entry.
- ItsContentsClass.h contains the declaration for the database storage.
- main.cp contains the main program.
- NewEntryDialogItems.h contains the constants for the NewEntryDialog items.
- References.cp contains the source code to force a reference to classes to prevent optimization out of the project.
- References.h contains the declaration for forcing references.
- x_CAddressBookApp.cp contains the source code for x_CAddressBookApp.
- x_CAddressBookApp.h contains the declaration for x_CAddressBookApp.
- x_CAddressBookDoc.cp contains the source code for x_CAddressBookDoc.
- x_CAddressBookDoc.h contains the declaration for x_CAddressBookDoc.
- x_CAddressBookPane.cp contains the source code for x_CAddressBookPane.
- x_CAddressBookPane.h contains the declaration for x_CAddressBookPane.
- x_CNewEntryDialog.cp contains the source code for x_CNewEntryDialog.
- x_CNewEntryDialog.h contains the declaration for x_CNewEntryDialog.
Appendix B, PowersOfTwo

These are the files used to create the PowersOfTwo project described in Appendix B:

- Main.cp contains the main program.
- Powers.cp contains the source code for Power.
- Powers.h contains the declaration of Power.
- PowersOfTwo.pi is the project file.
- ProjectorDB contains the Projector database of the project.

Application

The Applications folder contains the programs in Chapters 16-21.
### Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot; &quot;</td>
<td>(double quotes) in #include statements, 31</td>
</tr>
<tr>
<td>//</td>
<td>(double slash), comments, 126-127</td>
</tr>
<tr>
<td>::</td>
<td>(scope resolution) operator, 193</td>
</tr>
<tr>
<td>;</td>
<td>(semicolon) in declarations, 191</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>(insertion operator), 123</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>characters in #include statements, 31</td>
</tr>
<tr>
<td>=</td>
<td>(equals sign) in conditionals, 304</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>(extraction operator), 123</td>
</tr>
<tr>
<td>\x</td>
<td>conforming to ANSI, 265</td>
</tr>
<tr>
<td>68020</td>
<td>instructions, generating, 268</td>
</tr>
<tr>
<td>68881</td>
<td>instructions, generating, 268</td>
</tr>
</tbody>
</table>

### A

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>About boxes displaying, 361-362 resources, 380-381</td>
</tr>
<tr>
<td>About command, support for, 361-362</td>
</tr>
<tr>
<td>abstract classes, 181-182</td>
</tr>
<tr>
<td>access control to class members, 198-202</td>
</tr>
<tr>
<td>inline functions, 203-204</td>
</tr>
<tr>
<td>accessors, 202</td>
</tr>
<tr>
<td>Activate member function CClockPane class, 395</td>
</tr>
<tr>
<td>CView class, 563</td>
</tr>
<tr>
<td>Add Files command (Source menu), 26-27</td>
</tr>
<tr>
<td>Add Files dialog box, 26-28</td>
</tr>
<tr>
<td>Add Window command (Source menu), 25-26</td>
</tr>
<tr>
<td>Add Window-name command (Source menu), 26</td>
</tr>
<tr>
<td>AddressBook project, 453-454, 683-714</td>
</tr>
<tr>
<td>adding database functionality, 466</td>
</tr>
<tr>
<td>classes, 454-455</td>
</tr>
<tr>
<td>CAddressBookApp, 467-473</td>
</tr>
<tr>
<td>CAddressBookDoc, 473-479, 509-510</td>
</tr>
<tr>
<td>CAddressBookPane, 479-499</td>
</tr>
<tr>
<td>CNewEntryDialog, 489-499</td>
</tr>
<tr>
<td>CSaver, 501-509</td>
</tr>
<tr>
<td>CStreamableDatabase, 500-501</td>
</tr>
<tr>
<td>x_CAddressBookApp, 467-473</td>
</tr>
<tr>
<td>x_CAddressBookDoc, 474-479</td>
</tr>
<tr>
<td>x_CAddressBookPane, 481-499</td>
</tr>
<tr>
<td>x_CNewEntryDialog, 490-499</td>
</tr>
</tbody>
</table>
keyboard/mouse shortcuts, 258-259
multiple inheritance, 257-258
windows, creating, 250-254
Browser command (Source menu), 252
Build Application command (Project menu), 101-103
BuildWindow member function, CTextDoc class, 347-350
bureaucrats, 334-335
class for, 547-548
buttons class for, 548-549
creating, 416-417
drawing backgrounds, 423-424
Options dialog box, 263
titles resources, 430
fonts/sizes, 428-429
byte streams, class for, 561

c
C language, mixing with C++ in multiple-file projects, 41-43
.c file extension, 25, 44
The C Programming Language, 2nd Edition, 717
C++
comparing Symantec and standard C++, 514-517
comparing with C, 122
allocating/deallocating memory, 137-139
 caching, 196-198
comments, 126-128
const variables and inline functions versus #define macros, 139-152
declaring structures, 131
error checking and warnings, 133-135
I/O (input/output) operations, 122-126
locating variable definitions, 128-131
overloaded functions, 135-137
procedural versus object-oriented programs, 189-195
references, 152-159
converting from Apple’s MPW C++, 512
THINK C, 513-514
installing Symantec C++, 7
mixing with C in multiple-file projects, 41-43
publications, 718-719
C++ compiler, conforming to ANSI, 264-265
C++ Programming Style, 718
A C++ Toolkit, 719
C++Calc project, 406, 644-672
classes, 408-409
CButtonWithFont-AndSize, 428-429
CCalculatorApp, 412-413
CCalculatorPane, 413-424
CFloatText, 424-428
hierarchy diagram, 408
TCalculator, 409-412
compiling, 406
main function, 429
project files for creating, 728-729
resources, 430
running and testing, 406-407
C++Routine.cp source file, 43
C++Routine.h header file, 42-43
CAboutBoxApplication class, 361-362
CAboutBoxApplication.cp source file
C++Calc project, 644
PICTPeeker project, 602
TickTock project, 621
CAboutBoxApplication.h header file
C++Calc project, 645
PICTPeeker project, 615
TickTock project, 622
CAbstractText class, member functions, 542-543
caching, comparing procedural and object-oriented programming, 196-198
CAddressBookApp class, 467-473
CAddressBookApp.cp source file, AddressBook project, 683-686
CAddressBookApp.h header file, AddressBook project, 686-687
CAddressBookDoc class, 473-479, 509-510
CAddressBookDoc.cp source file, AddressBook project, 687-689
CAddressBookDoc.h header file, AddressBook project, 689-690
CAddressBookPane class, 479-499
CAddressBookPane.cp source file, AddressBook project, 691-693
CAddressBookPane.h header file, AddressBook project, 694-695
CalculateSleepTime member function, CClockPane class, 397-398
calculations power of two, 524-539
redundant, eliminating, 276-280

Symantec C++ Programming for the Macintosh
Calculators, using Reverse Polish Notation (RPN)
button title resources, 430
button titles, setting fonts/sizes, 428-429
displaying, 413-424
displaying floating-point numbers in text fields, 424-428
eengines, 409-412
windows, displaying, 412-413
call chains, displaying, 70
calling between C and C++, 41-42
CancelIdleChore member function, CApplication class, 544
can't nest comments
warning message, 305
CApplication class, data members and member functions, 543-546
CApplication member function, CApplication class, 544
CATCH macro, 344
CBartender class, 375-376
member functions, 546
CBureaucrat class, member functions, 547-548
CBButton class, member functions, 548-549
CBButton member function, CBButton class, 548-549
CBButtonWithFontAndSize class, 428-429
CButtonWithFontAndSize class, 428-429
CButtonWithFontAndSize.cpp source file, C++Calc project, 653-654
CButtonWithFontAndSize.h header file, C++Calc project, 654
CCalculatorApp class, 412-413
CCalculatorApp member function, CCalculatorApp class, 412-413
CCalculatorApp.cpp source file, C++Calc project, 654-655
CCalculatorApp.h header file, C++Calc project, 656
CCalculatorPane class, 413-424
CCalculatorPane.cpp source file, C++Calc project, 656-663
CCalculatorPane.h header file, C++Calc project, 664
CChore class, member functions, 549
CChore member function, CChore class, 549
CClipboard class, member functions, 549-550
CClockApp class, 385-387
CClockApp.cpp source file, TickTock project, 622-623
CClockApp.h header file, TickTock project, 623-624
CClockPane class, 393-402
CClockPane.cpp source file, TickTock project, 624-631
CClockPane.h header file, TickTock project, 631-632
CCollaborator class, member functions, 550
CCutCopyPastePicture class, 374-376
CCutCopyPastePicture.cpp source file, PICTPeeker project, 603-604
CCutCopyPastePicture.h header file, PICTPeeker project, 615
CDataFile class, member functions, 551-552
CDataFile member function, CDataFile class, 551
CDawdleBureaucratChore class, 392
CDawdleBureaucratChore.cpp source file, TickTock project, 632
CDawdleBureaucratChore.h header file, TickTock project, 633
CDirector class, member functions, 553
CDirector member function, CDirector class, 553
CFileDialog class, data members and member functions, 553-555
CFileDialog member function, CFileDialog class, 554
CEditText class, member functions, 555-556
CEditText member function, CEditText class, 555-556
CFile class, member functions, 556-557
CFloatText class, 424-428
CFloatText member function, CFloatText class, 425-426
CFloatText.cpp source file, C++Calc project, 665-666
CFloatText.h header file, C++Calc project, 666-667
chain of command, RandomRectangle project, 336
ChangeSize member function
CPane class, 557
CWindow class, 563
char types, signed versus unsigned, 267
Check In dialog box, 525-528, 531
Check Out dialog box, 529-530, 534
Check Out window, 531-533
Check Syntax command (Source menu), 33-34
CheckBinaryTree function, 78
CheckMarkCmd member function, CBartender class, 546
child classes, see derived classes
chores, class for, 549
cin stream, 123
Class command (Visual Architect Pane menu), 461
classes
abstract, 181-182
CAboutBoxApplication, 361-362
CAddressBookApp, 467-473
CAddressBookDoc, 473-479, 509-510
CAddressBookPane, 479-499
CBartender, 375-376
CButtonWithFontAndSize, 428-429
CCalculatorApp, 412-413
CCalculatorPane, 413-424
CClockApp, 385-387
CClockPane, 393-402
CCutCopyPastePicture, 374-376
CDawdleBureaucrat-Chore, 392
CFloatText, 424-428
CNewEntryDialog, 489-499
comparing with C structs, 190-191
CPictureApp, 363-365
CPictureClipboardTask, 376-380
CPictureDoc, 371-374
CQuitOnCloseWindow, 387-388
CRandomRectanglePane, 328-332
creating, 461-462
CRectangleApp, 333-336
CSaver, 501-509
declaring, 475-476
CSaveRestoreSizeWindow, 388-391
CStreamableDatabase, 500-501
CTextApp, 342-346
CTextDoc, 346-353
data members, 191-193
derived, 213-215
advantages, 218
allocating dynamic memory, 225-234
declaring, 215-216
multiple inheritance, 224-225
overriding member functions, 216-217
polymorphism, 218-223
using, 218
field location and length, 269-270
guidelines
for designing, 240-247
for using, 236-240
HandleObject, 516-517
hierarchy diagrams
C++Calc project, 408
DatabaseTester
project, 436
MicroGA
framework, 321
MacApp framework, 318
Physics framework, 321
PICTPeeker project, 359
QuickApp
framework, 320
TextTyper project, 341
TickTock project, 386
MCollectible, 437-438
member functions, 191-193
members, access control, 198-202
objects, 173-179
PascalObject, 517
SingleObject, 512
TCalculator, 409-412
TCL
CAbstractText, 542-543
CApplication, 543-546
CBartender, 546
CBureaucrat, 547-548
CButton, 548-549
CChore, 549
CClipboard, 549-550
CCollaborator, 550
CDataFile, 551-552
CDecorator, 552
CDialogDirector, 552
CDIalogText, 552
CDirector, 553
CDocument, 553-555
CEditText, 555-556
CFile, 556-557
CPane, 557-559
CPaneBorder, 559
CPanorama, 559
CPicture, 560
CScrollPane, 560-561
CStream, 561
CTask, 561-562
CView, 562-563
CWindow, 563-564
hierarchy diagram, 316, 541-542
TCL_AUTO_DESTRUCT_OBJECT, 369
TDatabase, 445-448
TDeleteOnFailure, 366-367
TDisposeHandleOnFailure, 368
TEntry, 441
THandleFailure, 367-368
THandleLocker, 369-370
TIterator, 448-452
TNameAddressPhone, 438-441
TNode, 441-445
TPolygon, 225-227
TRandom, 327-328
TWithFileOpen, 370
x_CAddressBookApp, 467-473
x_CAddressBookDoc, 474-479
x_CAddressBookPane, 481-499
x_CNewEntryDialog, 490-499
Classes command (Visual Architect Edit menu), 461
Clear command enabling, 417-418
processing, 418-421
support for pictures, 374-376
clearing pictures, doing/undoing/redoing, 376-380
Clipboard, class for handling, 549-550
clock (TickTock project), 383-403
Clock menu, TickTock resources, 402-403
Clock.cp source file, TickTock project, 633
Clock.r resource file, TickTock project, 634-638
Close member function CFile class, 556
x_CNewEntryDialog class, 497-498
closing files, 370
windows, exiting
TickTock project by, 387-388
CNewEntryDialog class, 489-499
CNewEntryDialog.cp source file, AddressBook project, 695-698
CNewEntryDialog.h header file, AddressBook project, 698-699
code, see source code
Code Optimization panel, Options dialog box, 262, 271-297
code resources, 98
CollectPaneValues member function, x_CNewEntryDialog class, 496
colors (random), drawing rectangles in, 332
command numbers, 355
commands
About, support for, 361-362
adding to menus in Visual Architect, 463-464
Clear enabling, 417-418
processing, 418-421
support for pictures, 374-376
Copy enabling, 417-418
processing, 418-421
support for pictures, 374-376
Debug menu, 62-63
Go, 69
In, 68-69
in automatic mode, 69
Out, 67-68
Step, 63-65
Stop, 69
Trace, 65-67
Edit menu, Options, see Options command
File menu, Modify Read-Only, 535-536
keystrokes, processing, 421-423
New Entry, handling, 479-499
Paste enabling, 417-418
processing, 418-421
support for pictures, 374-376
Project menu Build Application, 101-103
Remove Objects, 35
Run, 11
Set Project Type, 96-101
Use Debugger, 60-61
Save, handling options for, 352-353
Save As, handling options for, 353
Search menu
Find, 36-39
Find Again, 38
Find In Next File, 38
Go To Next Match, 39
Go To Previous Match, 39
sending to ToolServer, 539-540
to SourceServer, 537-538
Source menu
Add Files, 26-27
Add Window, 25-26
Add Window-name, 26
Browser, 252
Check Syntax, 33-34
Compile, 32-33
Precompile, 113-114
SourceServer Check In, 525-528
SourceServer Check In ‘xxx’, 531
SourceServer Check Out ‘xxx’, 529
SourceServer Launch SourceServer, 523
SourceServer New ProjectorDB, 524-525
SourceServer Quit SourceServer, 535
SourceServer Send Command, 537
THINK Project Manager menu, Generate All, 464-465
View menu
New View, 486
View Info, 486
Visual Architect Edit menu
All Clear, 455

Symantec C++ Programming for the Macintosh
Application, 464
Classes, 461
Commands, 462
Menu Bar, 463-464
Select All, 458
Visual Architect Pane menu
Class, 461
Font, 458
Identifier, 459-460
Size, 458
Style, 458
Visual Architect View menu, View Info, 456
Commands command (Visual Architect Edit menu), 462
comments
comparing C++ with C, 126-128
nested, 305
common subexpressions (CSEs), deleting, 276-280
communication between windows and programs, class for handling, 553
Compare member function MCollectible class, 437-438
TEntry class, 441
Compile command (Source menu), 32-33
compile errors, 13-14, 52-53
Compile Errors window, 52-53
Compiler Settings panel, Options dialog box, 262, 267-270
compilers, conforming C++ to ANSI, 264-265
compiling
AddressBook project, 454
C++Calc project, 406
DatabaseTester project, 434
improving performance with locations for include files, determining, 31-32
PICTPeeker project, 358
programs before running, 11-12
RandomRectangle project, 324-325
single files, 32-33
TextTyper text editor, 340
TickTock project, 384
type checking, 134
type-safe linkage, 134-135
warnings during, 133
ConcatPStrings global function, 566
conditionals, = (equals sign), 304
confirmation dialog boxes, turning off, 16-18
const keyword with references, comparing C++ with C, 154-155
const variables versus #define macros, 139-141, 151-152
constant propagation, 291-294
constants
binary, conforming to ANSI, 265
guidelines for designing classes, 243-244
string, storing, 101
constructors
allocating dynamic memory, 225-234
CAboutBoxApplication, 361
CalculatorPane, 415-416
CButtonWithFontAndSize, 428-429
CCalculatorApp, 412-413
CClockApp, 385
CClockPane, 394
CDawdleBureaucratChore, 392
CFloatText, 425-426
copy, overriding default, 232-233
CPictureClipboardTask, 377
CQuitOnCloseWindow, 388
CRandomRectanglePane, 329
CSaveRestoreSizeWindow, 389
CTextApp, 342
initializer lists, guidelines for designing classes, 244-246
macro, 569
simple, 204-207
TCalculator, 410
templates, 501-502
with parameters, 207-209
ContentsToWindow function
member, CAddressBookDoc class, 477
template, 508
Copy command enabling, 417-418
processing, 418-421
support for pictures, 374-376
copy constructors, overriding default, 232-233
copy propagation, 294-295
CopyFromOffscreen member function, CClockPane class, 399
copying pictures, doing/undoing/redoing, 376-380
CopyPString global function, 566
cout stream, 123
.cp file extension, 25, 44
CPane class, data members and member functions, 557-559
CPane member function, CPane class, 558
CPaneBorder class, member functions, 559
CPaneBorder member function, CPaneBorder class, 559
CPanorama class, member functions, 559
CPicture class, member functions, 560
CPicture member function, CPicture class, 560
CPictureApp class, 363-365
CPictureApp.cp source file, PICTPeeker project, 605-606
CPictureApp.h header file, PICTPeeker project, 615-616
CPictureClipboardTask class, 376-380
CPictureClipboardTask member function, CPictureClipboardTask class, 377
CPictureClipboardTask.cp source file, PICTPeeker project, 606-608
CPictureClipboardTask.h header file, PICTPeeker project, 616
CPictureDoc class, 371-374
CPictureDoc.cp source file, PICTPeeker project, 608-612
CPictureDoc.h header file, PICTPeeker project, 616-617
.cpp file extension, 25, 44
CQuitApplicationChore.cp source file
C++Calc project, 667
TickTock project, 638-639
CQuitApplicationChore.h header file
C++Calc project, 667
TickTock project, 639
CQuitOnCloseWindow class, 387-388
CQuitOnCloseWindow.cp source file
C++Calc project, 668
TickTock project, 639
CQuitOnCloseWindow.h header file
C++Calc project, 668
TickTock project, 640-641
CRandomRectanglePane class, 328-332
CRandomRectanglePane.cp source file, RandomRectangle project, 582-584
CRandomRectanglePane.h header file, RandomRectangle project, 586-587
CreateDocument member function
CApplication class, 544
CTextApp class, 342, 345-346
x_AddressBookApp class, 471-472
CreateNew member function, CFile class, 556
CreateOffscreen member function, CAddressBookDoc class, 477-478
CScrollPane class, member functions,
CScrollPane member function, CClockPane class, 398-399
creating, setting, 97
CRectangleApp class, 333-336
CRectangleApp.cp source file, RandomRectangle project, 584-585
CRectangleApp.h header file, RandomRectangle project, 587
CRoutine.c source file, 43
CRoutine.h header file, 42
CSaver class, 501-509
declaring, 475-476
CSaveRestoreSizeWindow class, 388-391
CSaveRestoreSizeWindow member function, CClockPane class, 413
CSaveRestoreSizeWindow.cp source file, C++Calc project, 668-670
CSaveRestoreSizeWindow.h header file
C++Calc project, 670-671
TickTock project, 642
CScrollView class, member functions, 560-561
CScrollView member function, CScrollView class, 560
CStream class, 561
CStream_CSrollableDatabase.cpp source file, AddressBook project, 702
CStreamableDatabase class, 500-501
CStreamableDatabase.cp source file, AddressBook project, 699-701
CStreamableDatabase.h header file, AddressBook project, 701
CTask class, data members and member functions, 561-562
CTask data member, CPictureClipboardTask class, 377-379
CTask member function, CTask class, 561-562
CTextApp class, 342-346
CTextApp.cp source file, TextTyper project, 592-593
CTextApp.h header file, TextTyper project, 597
CTextDoc class, 346-353
CTextDoc.cp source file, TextTyper project, 593-597
CTextDoc.h header file, TextTyper project, 597-598
Cut command, support for pictures, 374-376
cutting pictures, doing/undoing/redoing, 376-380
CView class, data members and member functions, 562-563
CWindow class, data members and member functions, 563-564
CWindow member function, CWindow class, 564

DAGs (directed acyclic graphs), 257
data forks, 551-552
data members, 191-193
CTask, CPictureClipboardTask class, 377-379
dirty, CDirector class, 553
fBureaucratToDawdle, CDawdleBureaucratChore class, 392
fButtons, CCalculatorPane class, 415
fCalcDisplay, CCalculatorPane class, 415
fCalculator, CCalculatorPane class, 415
fCurrent, TIterator class, 449
fCurrentNumber, CCalculatorPane class, 415
fDatabase, TIterator class, 449
fDataFile, CSaveRestoreSizeWindow class, 389
fEditText, CTextDoc class, 347
fName, TEntry class, 441
fOldMacPicture, CPictureClipboardTask class, 377
fOldScrap, CPictureClipboardTask class, 377
fRandom, CRandomRectanglePane class, 329
itsFile
CDocument class, 553
CTextDoc class, 353
itsGopher
CDirector class, 553
CTextDoc class, 350
itsLastTask, CPane class, 557
itsMainPane, CDocument class, 554
itsWindow, CDirector class, 553
sfFileTypes, CApplication class, 543
sfNumTypes, CApplication class, 544
sizeRect, CWindow class, 563
undone, CTask class, 561
wantsClicks
CCalculatorPane class, 416
CView class, 562
data structures
ADT (abstract data type), 171-173
data types, 170-171
modular programming, 169-170
x_CNewEntryDialog class, 490-492
data types, 170-171
ADT (abstract data type), 171-173
char, signed versus unsigned, 267
classes, 173-179
long double versus floating-point types, 268-269
data validation, 78-80
DatabaseTester project, 672-683
files for creating, 729
database view, handling menu items, 479-499
Database.cp source file
AddressBook project, 702-707
DatabaseTester project, 672-677
Database.h header file
AddressBook project, 707-709
DatabaseTester project, 677-678
databases, Projector accessing revision information, 531-533
automatically mounting, 538-539
checking in files, 528-529, 531
checking out files, 529-530
old versions of files, 533-534
revisions and modifying, 534-535
creating, 524-525
sharing, 535
DatabaseTester project, 434
classes, 435-436
hierarchy diagram, 436
MCollectible, 437-438
TDatabase, 445-448
TEntry, 441
TIterator, 448-452
TNameAddressPhone, 438-441
TNode, 441-445
compiling, 434
main function, 452-453
testing, 434-435
Dawdle member function
CBureaucrat class, 547
CClockPane class, 397
Deactivate member function
CClockPane class, 395
CView class, 563
dead
assignments, deleting, 271-273
variables, deleting, 273-276
deallocating memory, comparing C++ with C, 137-139
Debug menu commands, 62-63
Go, 69
In, 68-69
in automatic mode, 69
Out, 67-68
Step, 63-65
Stop, 69
Trace, 65-67
Debugger, 6
calling from within
programs, 76-77
calling functions from,
77-80
turning on, 60-61
viewing on second
monitor, 82
Debugger function, 76-77
Debugging panel, Options
dialog box, 262, 297-303
Symantec C++, 55-56,
58, 81
THINK
Project
Manager, 82
debugging source files
breakpoints, 72-73
call chains, displaying, 70
continuing through ends
of functions, 67-68
continuing to next
function call, 68-69
data validation, 78-80
enabling, 60-61
low-level debuggers, 80-81
starting programs, 61-62
stepping through, 63-65
tracing into statements
with function calls, 65-67
viewing functions, 71-72
viewing/setting variable
values, 73-76
DebugStr function, 76-77
declarations, 128
declaring
classes
CAddressBookApp, 467
CAddressBookDoc,
473-474, 509
CAddressBookPane,
480-481
CCalculatorApp, 412
CClockApp, 385
CDawdleBureaucrat-
Chore, 392
CNewEntryDialog, 489
CSaver, 475-476
CStreamableDatabase,
500
CTextApp, 342
CTextDoc, 347
derived, 215-216
guidelines for designing
classes, 240-243
MCollectible, 437
TDatabase, 446-447
TIterator, 448
x_CAddressBookApp,
467-468
x_CAddressBookDoc,
474-475
x_CAddressBookPane,
481
x_CNewEntryDialog,
490-492
structures, comparing C++
with C, 131
default
arguments, 132
assignment operator,
overriding, 230-231
copy constructors,
overriding, 232-233
file extensions for
translators, 44
parameters versus
overloaded functions,
238-240
values, guaranteed,
204-209
#define macros versus const
variables and inline
functions, 139-152
definitions, 128
delete operator, 137-139
DeleteOnCompletion.h
header file, PICTPeeker
project, 617-618
DeleteOnFailure.h header
file, PICTPeeker project,
618
deleting
arrays, 305
CSEs (common
subexpressions), 276-280
dead
assignments, 271-273
variables, 273-276
objects from multiple-file
projects, 35
unreferenced functions,
102-103
Demos folder, 4
dependent objects, class for
contact from provider
objects, 550
DependUpon member
function, CCollaborator
class, 550
dereferencing pointers, 75
derived classes, 213-215
advantages, 218
allocating dynamic
memory, 225-234
declaring, 215-216
multiple inheritance,
224-225
overriding member
functions, 216-217
polymorphism, 218-223
using, 218
desk accessories, 98
destructors
allocating dynamic
memory, 225-234
calling automatically at
exceptions, 369
CClockPane, 394-395
CPictureClipboardTask,
379-380
CQuitOnCloseWindow,
388
CSaveRestoreSizeWindow,
389
macro, 569
templates, 502
device drivers, 98
Dialog Box Information
dialog box, 486
dialog boxes
Add Files, 26-28
Check In, 525-528, 531
Check Out, 529-530, 534
class for directing, 552
confirmation, turning off,
16-18
Dialog Box Information,
486
directed acyclic graphs (DAGs), 257
directors, 335
dirty data member, CDirector class, 553
DisableCmd member function, CBartender class, 546
disks, classes for corresponding to documents, 553-555
files, 556-557
displays
drawing backgrounds, 423-424
setting maximum widths, 426
updating, 424
DisposeHandleOnFailure.h header file, PICTPeeker project, 619
Do member function CPictureClipboardTask class, 377-378
CTask class, 562
DoBeginData member function, x_CNewEntryDialog class, 495
DoCmdGoBackward member function, CAddressBookPane class, 482
DoCmdGoForward member function, CAddressBookPane class, 482
DoCmdGoToBeginning member function, CAddressBookPane class, 482
DoCmdGoToEnd member function, CAddressBookPane class, 482
DoCommand member function CAboutBoxApplication class, 361-362
CAddressBookApp class, 469-470
CBureaucrat class, 547
CCalculatorPane class, 418-421
CClockPane class, 396-397
x_CAddressBookApp class, 470
x_CAddressBookPane class, 485
documents creating, 347-348, 363-365, 467-473
exception handling during, 345-346
keeping in sync with windows, 473-479
on disk, class for corresponding to, 553-555
opening, 350-351
saving, 352-353
streamable, 509-510
DoEndData member function, x_CNewEntryDialog class, 495-496
DoKeyDown member function CBureaucrat class, 547-548
CCalculatorPane class, 421-423
DoModalDialog member function, x_CNewEntryDialog class, 497
DoRevert function template, 508
DoSave function member
CDocument class, 554
CPictureDoc class, 372-374
CTextDoc class, 352-353
template, 506
DoSaveAs function member
CDocument class, 554
CTextDoc class, 352
template, 506-508
DoSaveFileAs member function
CDocument class, 554
CTextDoc class, 352
double dereferencing of pointers, 75
double quotes (" ") in #include statements, 31
double slash (//), comments, 126-127
double types (long) versus floating-point types, 268-269
Draw member function
 CCALCULATORPANE class, 423-424
 CCLOCKPANE class, 397
 CPANE class, 558
Draw AnalogClock member function, CCLKPANE class, 399-400
CClockPane class, 399
Draw member function, CCLOCKPANE class, 399
DrawDigitalClock member function, CCLOCKPANE class, 401
drawing
clocks, 393-402
obtaining idle time for, 330-331
rectangles in random colors, 332
in random patterns, 331-332
DrawLine member function, 400
DrawRectangleInRandomColor member function, CRandomRectanglePane class, 332
DrawRectangleInRandomPattern member function, CRandomRectanglePane class, 331-332
dynamic memory, allocating, 225-234

E

Edit menu, modifying, 354-355
Edit menu commands
 Clear
 enabling, 417-418
 processing, 418-421
 support for pictures, 374-376
Copy enabling, 417-418
processing, 418-421
support for pictures, 374-376
Cut, support for pictures, 374-376
Options, 261-263
SyMantec C++, 55
THINK Project Manager, 16-18
Paste enabling, 417-418
processing, 418-421
support for pictures, 374-376
Visual Architect
All Clear, 455
Application, 464
Classes, 461
Commands, 462
Menu Bar, 463-464
Select All, 458
editable text, class for panes with, 555-556
editing
files without access to Projector databases, 535-536
source code, 466-485
source files, 10-11
views, 459
editors, TextTyper, 340-355, 726

Effective C++: 50 Specific Ways to Improve Your Programs and Designs, 719
Emergent Behavior
MicroGA framework, 320-321
Physics framework, 321
QuickApp framework, 315-320
empty loops, 305
EmptyScrap member function, CCLIPBOARD class, 550
EnableCmd member function, CBARTENDER class, 546
EndData member function
CNEWENTRYDIALOG class, 489-490
x_CNewEntryDialog class, 497
ENDTRY macro, 344
EnterNumber member function,
CCALCULATORPANE class, 424
Entry.cp source file
AddressBook project, 709-711
DatabaseTester project, 679-681
Entry.h header file
AddressBook project, 711-712
DatabaseTester project, 681
enumerations
conforming to ANSI, 265
defining, 265-266
guidelines for designing classes, 243-244
types, sizes
comparing C++ and THINK C, 513-514
comparing SyMantec and standard C++, 514
EqualFile function, 472-473
equals sign (=) in conditionals, 304
error messages
Different configuration for precompiled header, 115
function definitions with separate parameter lists are not allowed in C++, 307
member 'xxx::xxxx' of class 'xxx' is not accessible, 201
see also warning messages
errors
checking and warnings, comparing C++ with C, 133-135
extensions
  comparing Symantec and
  standard C++, 514-516
SingleObject, 512
extraction operator (>>), 123

F

Factorial.cp file,
  MultipleFiles project, 24
Factorial.cp source file,
  MultipleFiles project, 573
Factorial.h file, MultipleFiles
  project, 23
Factorial.h header file,
  MultipleFiles project, 574
FailMemError global
  function, 343, 566
FailNIL global function,
  343, 566
FailNILRes function, 343
FailOpen member function,
  x_CAddressBookDoc
  class, 479
FailOSErr global function,
  343, 566
FailResError function, 343
Failure global function, 566
fBureaucratToDawdle data
  member, CDawdleBureaucratChore class, 392
fButtons data member,
  CCalculatorPane class, 415
fCalcDisplay data member,
  CCalculatorPane class, 415
fCalculator data member,
  CCalculatorPane class, 415
fCurrent data member,
  TIterator class, 449
fCurrentNumber data
  member, CCalculatorPane
  class, 415
fDatabase data member,
  TIterator class, 449
fDataFile data member,
  CSaveRestoreSizeWindow
class, 389

fEditText data member,
  CTextDoc class, 347
fields
  location and length in
  structures/unions/
  classes, 269-270
text, displaying floating-
  point numbers, 424-428

File menu commands
  Modify Read-Only,
  535-536
Save, handling options
  for, 352-353
Save As, handling options
  for, 353

FileAlreadyOpen member
  function, x_CAddress-
  BookApp class, 472-473
files
  adding to projects, 22-31
Alphabet.r, 90-91, 580-581
Average.cp, 24, 51-52,
  571-572, 574
Average.h, 24, 51, 574, 576
Bar.h, 582
BinaryTree.c, 175-176
BinaryTree.h, 175
C++Routine.cp, 43
C++Routine.h, 42-43
CAboutBoxApplication.cp,
  602, 621, 644
CAboutBoxApplication.h,
  615, 622, 645
CAddressBookApp.cp,
  683, 684
CAddressBookApp.h,
  686, 687
CAddressBookApp.cp,
  688-689
CAddressBookApp.h,
  689-690
CAddressBookDoc.cp,
  694-695
CAddressBookDoc.h,
  696-697
CAddressBookPane.cp,
  698-700
CAddressBookPane.h,
  701-703
Calculator.cp, 645-647
Calculator.h, 647
Calculator.r, 648-652
CalculatorMain.cp, 653
checking in to Projector databases, 528-529, 531
checking out from Projector databases, 529-530
old versions, 533-534
Clock.cp, 633
Clock.r, 634-638
closing, 370
CNewEntryDialog.cp, 695-698
CNewEntryDialog.h, 698-699
compiling single files, 32-33
CPictureApp.cp, 605-606
CPictureApp.h, 615-616
CPictureClipboardTask.cp, 606-608
CPictureClipboardTask.h, 616
CPictureDoc.cp, 608-612
CPictureDoc.h, 616-617
CQuitApplicationChore.cp, 638-639, 667
CQuitApplicationChore.h, 639, 667
CQuitOnCloseWindow.cp, 639, 668
CQuitOnCloseWindow.h, 640-641, 668
CRandomRectanglePane.cp, 582-584
CRandomRectanglePane.h, 586-587
creating, 347-348
CRectangleApp.cp, 584-585
CRectangleApp.h, 587
CRoutine.c, 43
CRoutine.h, 42
CSaveRestoreSizeWindow.cp, 668-670
CSaveRestoreSizeWindow.h, 642, 670-671
CStream_CStreamableDatabase.cpp, 702
CStreamableDatabase.cp, 699-701
CStreamableDatabase.h, 701
CTextApp.cp, 592-593
CTextApp.h, 597
CTextDoc.cp, 593-597
CTextDoc.h, 597-598
data forks, 551-552
Database.cp, 672-677, 702-707
Database.h, 677-678, 707-709
DeleteOnCompletion.h, 617-618
DeleteOnFailure.h, 618
DisposeHandleOnFailure.h, 619
Entry.cp, 679-681, 709-711
Entry.h, 681, 711-712
extensions
.asm, 44
.c, 25, 44
.cp, 25, 44
.cpp, 25, 44
for translators, 43-46
.note, 44
.o, 44
.4, 44
Factorial.cp, 24, 573
Factorial.h, 23, 574
Foo.cp, 581
Foo.h, 582
FooBar.cp, 582
for creating
AddressBook project, 729-730
Alphabet project, 724-725
BinaryTree project, 724
C++Calc project, 728-729
DatabaseTester project, 729
Errors project, 724
MultipleFiles project, 723
PICTPeeker project, 726-727
PowersOfTwo project, 731
Precompiled headers projects, 725
RandomRectangle project, 725-726
TextTyper text editor, 726
TickTock project, 727-728
HandleFailure.h, 619
HandleLocker.h, 620
header
precompiled, 105-116
speed in processing, 266-267
include
locations for faster compiles, determining, 31-32
opening from source files, 18-19
ItsContentsClass.h, 712
Main.c, 178
Main.cp
AddressBook project, 713
frames (stack), generating, 298
frameworks, 311-312
guidelines for using, 314
MacApp, 315, 318
MicroGA, 320-321
ODPF (OpenDoc Parts Framework), 322
Physics, 321
polymorphism, 220-221
QuickApp, 315-320
reasons for using, 312-313
TCL (THINK Class Library), 314-315
types, 313-314
fRandom data member, CRandomRectanglePane class, 329
free function, comparing C++ with C, 137
function definitions with separate parameter lists are not allowed in C++ error message, 307
functions
Activate
CClockPane class, 395
CView class, 563
Advance,TRandom class, 327-328
arguments, default, 132
AssignIdleChore, CApplication class, 544
AtBeginning, TIterator class, 450
AtEnd, TIterator class, 450
BecomeGopher, CBureaucrat class, 547
BeginData
CNewEntryDialog class, 490
x_CNewEntryDialog class, 496
BroadcastChange, CCollaborator class, 550
BuildWindow, CTextDoc class, 347-350
CAboutBoxApplication class, 363-365
CalculateSleepTime, CClockPane class, 397-398
CalculatorPane, CCalculatorPane class, 415-416
call chains, displaying, 70 calls
direct, comparing Symantec and standard C++, 515
from Debugger, 77-80
profiler, generating, 299
CancelIdleChore, CApplication class, 544
CApplication, CApplication class, 544
CButton, CButton class, 548-549
CCalculatorApp, CCAl-
culatorApp class, 412-413
CChore, CChore class, 549
CDataFile, CDataFile class, 551
CDirector, CDirector class, 553
CDocument, CDocument class, 554
CEditText, CEditText class, 555-556
CFloatText, CFloatText class, 425-426
ChangeSize
CPane class, 557
CWindow class, 563
CheckBinaryTree, 78
CheckMarkCmd, CBartender class, 546
Close
CFile class, 556
x_CNewEntryDialog class, 497-498
CollectPaneValues, x_CNewEntryDialog class, 496
Compare
MCollectible class, 437-438
TEntry class, 441
comparing procedural and object-oriented, 192-193
ConcatPStrings, 566
constructors, 204-209
ContentsToWindow, CAddressBookWindow class, 477
CopyFromOffscreen, CClockPane class, 399
CopyPString, 566
CPane, CPane class, 558
CPaneBorder, CPaneBorder class, 559
CPicture, CPicture class, 560
CPictureClipboardTask, 377
CreateCommandButton, CCAl-
culatorPane class, 416-417
CreateDatabaseIterator, CAddressBookDoc class, 477-478
CreateDocument
CApplication class, 544
CTextApp class, 342, 345-346
x_CAddressBookApp class, 471-472
CreateNew, CFile class, 556
CreateOffscreen, CClockPane class, 398-399
CSaveRestoreSizeWindow, CCalculatorApp class, 413
CScrollPane, CCScrollView class, 560
CTask, CTask class, 561-562
CWindow, CWindow class, 564
Dawdle
CBureaucrat class, 547
CClockPane class, 397
Deactivate
CClockPane class, 395
CView class, 563
Debugger, 76-77
DebugStr, 76-77

Symantec C++ Programming for the Macintosh

748
Depend Upon,
  CCollaborator class, 550
Disable Cmd, CBartender
  class, 546
displaying source code in
  Browser, 254
Do
  CPictureClipboardTask
class, 377-378
CTask class, 562
DoBeginData, x_CNew-
  EntryDialog class, 495
DoCmdGoBackward,
  CAddressBookPane
class, 482
DoCmdGoForward,
  CAddressBookPane
class, 482
DoCmdGoToBeginning,
  CAddressBookPane
class, 482
DoCmdGoToEnd, CAd-
  dressBookPane class, 482
DoCommand
  CAboutBoxApplication
class, 361-362
CAddressBookApp class,
  469-470
CBureaucrat class, 547
CCalculatorPane class,
  418-421
CClockPane class,
  396-397
x_CAddressBookApp
class, 470
x_CAddressBookPane
class, 485
DoEndData, x_CNewEn-
  tryDialog class, 495-496
DoKeyDown
  CBureaucrat class,
  547-548
CCalculatorPane class,
  421-423
DoModalDialog,
  x_CNewEntryDialog
class, 497
DoSave
  CDocument class, 554
CPictureDoc class,
  372-374
CTextDoc class, 352-353
DoSaveAs
  CDocument class, 554
CTextDoc class, 352
DoSaveFileAs
  CDocument class, 554
CTextDoc class, 352
Draw
  CCalculatorPane class,
  423-424
CClockPane class, 397
CPane class, 558
DrawAnalogClock,
  CClockPane class,
  399-400
DrawClock, CClockPane
class, 399
DrawDigitalClock,
  CClockPane class, 401
DrawLine, 400
DrawRectangleInRandom-
  Color, CRandomRect-
  anglePane class, 332
DrawRectangleIn-
  RandomPattern,
  CRandomRectanglePane
class, 331-332
EmptyScrap, CClipboard
class, 550
EnableCmd, CBartender
class, 546
EndData
  CNewEntryDialog class,
  489-490
x_CNewEntryDialog
class, 497
EnterCurrentNumber,
  CCalculatorPane
class, 424
EqualFile, 472-473
ExistsOnDisk, CFile
class, 556
Exit, CApplication
class, 545
FailMemError, 343, 566
FailNIL, 343, 566
FailNILRes, 343
FailOpen, x_CAddress-
  BookDoc class, 479
FailOSErr, 343, 566
FailResError, 343
Failure, 566
FileAlreadyOpen,
  x_CAddressBookApp
class, 472-473
Find
  TDatabase class, 447
TNode class, 443-445
FindAndPrint, 452-453
FindPane, x_CNewEn-
  tryDialog class, 494-495
FitToEnclosure, CPane
class, 558
FitToEnclosure, CPane
class, 558
ForceClassReferences
  CAddressBookApp
class, 469
  x_CAddressBookApp
class, 469
free, comparing C++ with
  C, 137
GetAddressBookDoc,
  CAddressBookPane
class, 484
GetCurrent, TIterator
class, 449
GetData, CClipboard
class, 550
GetDatabase, CStream-
  ableDatabase class, 500
GetDigitalFontAndSize,
  CClockPane class,
  401-402
GetDigitalString, CClock-
  Pane class, 401-402
GetDigitalStringLocation,
  CClockPane class,
  401-402
GetDocTypeFromDialog,
  CAddressBookApp
class, 469
GetFrame
  CRandomRectanglePane
class, 330
CView class, 563
GetFrom,
CStreamableDatabase
class, 501
GetLength, CDataFile
class, 551
GetMacPicture
CBartender class, 375
CPicture class, 560
GetName, CFile class, 556
GetRoot, Tlterator
class, 449
GetTextHandle,
CAbstractText class, 542
GetTextString, CFloatText
class, 427-428
GetTitle, CWindow
class, 564
GetValue, CFloatText
class, 426
GetWCount, CDecorator
class, 552
global, 566-567
guidelines for returning
results, 237
IApplication,
CApplication class, 545
IAddressBookApp,
CAddressBookApp
class, 468
IAAddressBookDoc,
CAddressBookDoc class,
476-477, 509
ICNewEntryDialog,
CNewEntryDialog
class, 490
identically named,
255-256
IDialogDirector,
CDialogDirector
class, 552
IDocument, CDocument
class, 555
InitialSetup,
CAddressBookPane class,
481-482
inline, 203-204
guidelines for designing
classes, 240-243
turning into normal
functions, 300-302
versus #define macros,
141-152
Insert
TDatabase class,
447-448
TNode class, 443, 445
InstallPanorama,
CScrollPane class, 561
IsUndone, CTTask
class, 562
Ix_ CAddressBookApp,
x_CAddressBookApp
class, 468-469
Ix_ CAddressBookDoc,
x_CAddressBookDoc
class, 477
Ix_ CNewEntryDialog,
x_CNewEntryDialog
class, 493
jump table entries, 100
LeftMost, TNode
class, 443
LongToQDRect, 567
MacsBug names, generating,
299-300
main
AddressBook project,
466-467
C++Calc project, 429
DatabaseTester project,
452-453
PICTPeeker project, 380
RandomRectangle
project, 336-337
TextTyper text
editor, 354
TickTock project, 402
MakeNewContents,
CAddressBookDoc class,
477, 509-510
MakeNewWindow
x_CAddressBookDoc
class, 478-479
x_CNewEntryDialog
class, 493-494
malloc, comparing C++
with C, 137
member, 180-181,
191-193
overriding, 216-217
see also member
functions
Move, CWindow class, 564
MoveBackward, Tlterator
class, 451-452
MoveForward, Tlterator
class, 450-451
MoveToBeginning,
Tlterator class, 450
MoveToEnd, Tlterator
class, 450
NewFile
CDocument class, 555
CTextApp class, 345-346
NewHandleCanFail, 567
NoMore, Tlterator
class, 449
Notify, CBureaucrat
class, 548
NotifyClean, CBureaucrat
class, 548
Open, CFile class, 556
OpenDocument
CDocument class, 555
CApplication class, 545
CTextApp class, 345-346
x_CAddressBookApp
class, 472
OpenFile
CApplication class, 545
CTextApp class, 345-346
versus default param-
eters, 238-240
versus overridden mem-
er functions, 219-220
passing parameters by
reference, comparing
C++ with C, 152-159
Perform, CChore
class, 549
PlaceNewWindow,
CDecorator class, 552
pointers to, conforming to
ANSI, 264
PositionDialog, 567
PositionWindow, x_CAddressBookDoc class, 479
Prepare, CView class, 563
PrintDatabase, 452-453
ProviderChanged
   CActiveAddressBookPane
      class, 484-485
   x_CNewEntryDialog
      class, 498-499
PutData, CActiveClipboard
      class, 550
PutTo, CActiveStreamableDatabase class, 500-501
Quit, CActiveApplication
      class, 545
ReadAll, CActiveDataFile
      class, 551
ReadSome, CActiveDataFile
      class, 551
Redo
   CActivePictureClipboardTask
      class, 378
   CActiveTask class, 562
Refresh, CActivePane class, 559
ResizeHandleCanFail, 567
RightMost, TNode
      class, 443
Run, CActiveApplication
      class, 545
Select, CActiveWindow class, 564
SetAllocation, 567
SetBorder, CActivePane
      class, 559
SetChanged, CBureaucrat
      class, 548
SetClickCmd, CActiveButton
      class, 549
SetFontName,
   CActiveAbstractText class, 542
SetFontSize, CActiveAbstractText
      class, 542
SetFontStyle,
   CActiveAbstractText class, 543
SetInfoFromFile,
   CActiveSaveRestoreSizeWindow
      class, 391
SetMacPicture, CActivePicture
      class, 560
SetMark, CActiveDataFile
      class, 551
SetMaxDisplayWidth,
   CFloatText class, 426
SetPenSize, CActivePaneBorder
      class, 559
SetPosition, CActivePanorama
      class, 559
SetSizeRect, CActiveWindow
      class, 564
SetTextHandle
   CActiveAbstractText class, 543
   CActiveTextDoc class, 351
SetTextPtr, CActiveAbstractText
      class, 543
SetTextString,
   CActiveAbstractText class, 543
SetTitle, CActiveWindow
      class, 564
SetUpFileParameters
   CActiveApplication
      class, 545-546
      CActiveTextDoc class, 346
   x_CAddressBookApp
      class, 471
SetUpMenus, x_CAddressBookApp
      class, 471
SetValue, CFIntText
      class, 378-379
   CActivePictureClipboardTask
      class, 378-379
   CActiveTask class, 562
ThrowOut, CActiveFile
      class, 557
Undo
   CActivePictureClipboardTask
      class, 378-379
   CActiveTask class, 562
unreferenced, removing,
   102-103
UpdateData,
   x_CNewEntryDialog
      class, 497
UpdateDisplay,
   CActiveCalculatorPane
      class, 424
UpdateEntryData,
   CActiveAddressBookPane class,
      483-484
UpdateMenus
   CActiveAddressBookPane,
      482-483
   CBureaucrat class, 548
   CActiveCalculatorPane,
      417-418
   CActiveClockPane
      class, 395-396
   CActiveCutCopyPastePicture
      class, 375-376
   x_CAddressBookPane,
      483
templates, 501
constructors, 501-502
ContentsToWindow,
   508
destructors, 502
DoRevert, 508
DoSave, 506
DoSaveAs, 506-508
ExistingFile, 508-509
MakeWindowName,
   503-504
NewFile, 502
NewFileType, 506
OpenFile, 502-503
PositionWindow, 504
ReadContents, 504
ReadDocument,
   504-505
WindowToContents,
   508
WriteContents, 505
syntax changes, 307
GetDigitalStringLocation member function, CClockPane class, 401-402
GetDocTypeFromDialog member function, CAddressBookApp class, 470
GetFrame member function, CRandomRectanglePane class, 330
CView class, 563
GetFrom member function, CStreamableDatabase class, 501
GetLength member function, CDataFile class, 551
GetMacPicture member function
CBartender class, 375
CPicture class, 560
GetName member function, CFile class, 556
GetRoot member function, TIterator class, 449
getters, 202
GetTextHandle member function, CAbstractText class, 542
GetTextString member function, CFloatText class, 427-428
GetTitle member function, CWindow class, 564
GetValue member function, CFloatText class, 426
GetWCount member function, CDecorator class, 552

G

gBartender global variable, 375, 565
gClipboard global variable, 565
gDecorator global variable, 565
Generate All command (THINK Project Manager menu), 464-465

Genetic algorithms, frameworks for creating, 320-321

GetAddressBookDoc member function, CAddressBookPane class, 484
GetCurrent member function, TIterator class, 449
GetData member function, CClipboard class, 550
GetDatabase member function, CStreamableDatabase class, 500
GetDigitalFontAndSize member function, CClockPane class, 401-402
GetDigitalString member function, CClockPane class, 401-402

GetDigitalStringLocation member function, CClockPane class, 401-402
GetDocTypeFromDialog member function, CAddressBookApp class, 470
GetFrame member function, CRandomRectanglePane class, 330
CView class, 563
GetFrom member function, CStreamableDatabase class, 501
GetLength member function, CDataFile class, 551
GetMacPicture member function
CBartender class, 375
CPicture class, 560
GetName member function, CFile class, 556
GetRoot member function, TIterator class, 449
getters, 202
GetTextHandle member function, CAbstractText class, 542
GetTextString member function, CFloatText class, 427-428
GetTitle member function, CWindow class, 564
GetValue member function, CFloatText class, 426
GetWCount member function, CDecorator class, 552

global functions
ConcatPStrings, 566
CopyPString, 566
FailMemError, 343, 566
FailNIL, 343, 566
FailOSErr, 343, 566
Failure, 566
LongToQDRect, 567
NewHandleCanFail, 567
PositionDialog, 567
ResizeHandleCanFail, 567
SetAllocation, 567

G

gBartender global variable, 375, 565
gClipboard global variable, 565
gDecorator global variable, 565
Generate All command (THINK Project Manager menu), 464-465

Genetic algorithms, frameworks for creating, 320-321

GetAddressBookDoc member function, CAddressBookPane class, 484
GetCurrent member function, TIterator class, 449
GetData member function, CClipboard class, 550
GetDatabase member function, CStreamableDatabase class, 500
GetDigitalFontAndSize member function, CClockPane class, 401-402
GetDigitalString member function, CClockPane class, 401-402

GetDigitalStringLocation member function, CClockPane class, 401-402
GetDocTypeFromDialog member function, CAddressBookApp class, 470
GetFrame member function, CRandomRectanglePane class, 330
CView class, 563
GetFrom member function, CStreamableDatabase class, 501
GetLength member function, CDataFile class, 551
GetMacPicture member function
CBartender class, 375
CPicture class, 560
GetName member function, CFile class, 556
GetRoot member function, TIterator class, 449
getters, 202
GetTextHandle member function, CAbstractText class, 542
GetTextString member function, CFloatText class, 427-428
GetTitle member function, CWindow class, 564
GetValue member function, CFloatText class, 426
GetWCount member function, CDecorator class, 552

global structures, gSystem, 331
global variables accessing, 101
ADT (abstract data type), 171-173
data types, 170-171
gBartender, 375, 565
gClipboard, 565
gDecorator, 565
gSignature, 565
gSystem, 565
modular programming, 169-170

Go command (Debug menu), 69
Go To Next Match command (Search menu), 39
Go To Previous Match command (Search menu), 39
GoodnightMoon project, 42-43
gophers, 335

Graphics
cutting/copying/pasting/clearing
doing/undoing/redoing, 376-380
support for, 374-376
displaying, 357-381
panes for displaying, class for, 560
gSignature global variable, 565
gSystem global structure, 331
gSystem global variable, 565

H

Handbook of Walkthroughs, Inspections & Technical Reviews, 721
HandleFailure.h header file, PICTPeeker project, 619
HandleLocker.h header file, PICTPeeker project, 620
HandleObject class, 516-517
handles

disposing of, exception

dlcocking, 369-370

header files

Average.h

Errors project, 51, 576

MultipleFiles project, 24, 574

Bar.h, Project Two

project, 582

BinaryTree.h, 175

C++Routine.h, 42-43

CAboutBoxApplication.h

C++Calc project, 645

PICTPeeker project, 615

TickTock project, 622

CAddressBookApp.h,

AddressBook project, 686-687

CAddressBookDoc.h,

AddressBook project, 689-690

CAddressBookPane.h,

AddressBook project, 694-695

Calculator.h, C++Calc

project, 647

CButtonWithFontAndSize.h,

C++Calc project, 654

CCalculatorApp.h,

C++Calc project, 656

CCalculatorPane.h,

C++Calc project, 664

CClockApp.h, TickTock

project, 623-624

CClockPane.h, TickTock

project, 631-632

CCutCopyPastePicture.h,

PICTPeeker project, 615

CDawdleBureaucratChore.h,

TickTock project, 633

CFloatText.h, C++Calc

project, 666-667

CNewEntryDialog.h,

AddressBook project, 698-699

CPictureApp.h,

PICTPeeker project, 615-616

CPictureClipboardTask.h,

PICTPeeker project, 616

CPictureDoc.h, PICT-

Peeker project, 616-617

CQuitApplicationChore.h

C++Calc project, 667

TickTock project, 639

CQuitOnCloseWindow.h

C++Calc project, 668

TickTock project, 640-641

CRandomRectanglePane.h,

RandomRectangle

project, 586-587

CRectangleApp.h,

RandomRectangle

project, 587

CRoutine.h, 42

CSaveRestoreSizeWindow.h

C++Calc project,

670-671

TickTock project, 642

CStreamableDatabase.h,

AddressBook project, 701

CTextApp.h, TextTyper

project, 597

CTextDoc.h, TextTyper

project, 597-598

Database.h

AddressBook project,

707-709

DatabaseTester project,

677-678

DeleteOnCompletion.h,

PICTPeeker project,

617-618

DeleteOnFailure.h,

PICTPeeker project, 618

DisposeHandleOnFailure.h,

PICTPeeker project, 619

Entry.h

AddressBook project,

711-712

DatabaseTester project,

681

Factorial.h, MultipleFiles

project, 23, 574

Foo.h, Project Two

project, 582

HandleFailure.h,

PICTPeeker project, 619

HandleLocker.h,

PICTPeeker project, 620

ItsContentsClass.h,

AddressBook project, 712

Power.h, PowersOfTwo

project, 715

precompiled, 105

changing, 111-113

comparing with

standard include files,

107-110

creating, 113-115

determining need

for, 111

including in individual

source files, 110

rules for use, 106

standard, 107, 116

RedBlackTree.h, 176

ResourceDefinitions.h,

Alphabet project, 86, 580

Reverse.h

Errors project, 51, 576

MultipleFiles project,

23, 574

speed in processing,

266-267

SplayTree.h, 177

TDisposeHandleOnFailure.h,

PICTPeeker project, 617

TGrafPortSetter.h

C++Calc project,

671-672

TickTock project,

642-643

TRandom.h,

RandomRectangle

project, 587-588

TWithFileOpen.h,

PICTPeeker project, 620

Utilities.h, TickTock

project, 643

Hello World project

creating, 8-10

running program, 11-13

source files, editing, 10-11

SYMANTEC C++ Programming for the Macintosh
How to Write Macintosh
Software: The Debugging Reference for the Macintosh, 722

idle-time chores, class for, 549
IDocument member function, CDocument class, 555
In command (Debug menu), 68-69
include files comparing with precompiled header files, 107-110
locations for faster compiles, determining, 31-32
opening from source files, 18-19
infix notation, 406
inheritance, 173-174
multiple, 224-225
Browser, 257-258
initializer lists, 217
guidelines for designing classes, 244-246
InitialSetup member function,
CAddressBookPane class, 481-482
inline functions, 203-204
guidelines for designing classes, 240-243
turning into normal functions, 300-302
versus #define macros, 141-152
inline keyword, 241-243
Insert member function
TDatabase class, 447-448
TNode class, 443, 445
insertion (<<) operator, 123
Inside Macintosh, 722
Inspector Libraries folder, 5
installing Symantec C++, 7
InstallPanorama member function, CScrollPane class, 561
instances, 173-179
IsUndone member function, CTask class, 562
ItsContentsClass.h header file, AddressBook project, 712
itsFile data member
CDocument class, 553
CTextDoc class, 353
itsGopher data member
CDirector class, 553
CTextDoc class, 350
itsLastTask data member,
CPane class, 557
itsMainPane data member,
CDocument class, 554
itsWindow data member,
CDirector class, 553
Ix_CAddressBookApp member function,
x_CAddressBookApp class, 468-469
Ix_CAddressBookDoc member function,
x_CAddressBookDoc class, 477
Ix_CNewEntryDialog member function,
x_CNewEntryDialog class, 493

jump tables, setting sizes, 100
keyboard shortcuts
Check All (Command-A), 37
in Browser, 254, 258-259
processing, 421-423
Send command to SourceServer (Ctrl-Enter), 537
Send command to ToolServer (Option-Enter), 539
keywords
inline, 203-204, 241-243
private, 199-202
protected, 199-202
public, 199-202
this, comparing procedural and object-oriented programming, 193-194

loops
empty, 305
inlining functions with,
148-149

Language Settings panel,
Options dialog box,
262-267
LeftMost member function,
TNode class, 443
libraries, adding to projects,
29-31
lines with start/end points
C program, 186-188
C++ program, 188-189
link errors, 14-16, 53-55
Link Errors window, 54
linker errors, 40
linking
programs before running,
11-12
type-safe checking,
134-135
with Smart Link, 102-103
literals, see string literals
local variables
allocating too much memory,
305
returning addresses of, 307
locking handles, 369-370
long double type versus floating-point types,
268-269
LongToQDRect global function, 567
loop induction variables,
288-291
loop invariants, removing from loops after first evaluation, 282-287
looping through all possible optimizations, 296-297

Mac #includes folder, 5
Mac Libraries folder, 6
MacApp framework, 315
class hierarchy diagram, 318
MacHeaders++ precompiled header file, 107
comparing with standard include files, 107-110
customizing, 116
Macintosh publications, 722
macros
CATCH, 344
#define, versus const variables and inline functions, 139-152
ENDTRY, 344
exception handling, 568
TCL_AUTO_DESTRUCT_OBJECT, 568
TCL_DECLARE_CLASS, 569
TCL_DEFINE_CLASS, 569
TCL_END_CONSTRUCTOR, 569
TCL_FORCE_REFERENCES, 570
TCL_START_DESTRUCTOR, 569
TCLForgetObject, 568
TRY, 344
MacsBug names, 81
generating, 299-300
Main document View Info dialog box, 456-457
main folder, 4
main function
AddressBook project,
466-467
C++Calc project, 429
DatabaseTester project,
452-453
PICTPeeker project, 380
RandomRectangle project, 336-337
TextTyper text editor, 354
TickToc project, 402
main programs, comparing procedural and object-oriented, 192
Main.c source file, 178
Main.cp source file, 42
AddressBook project, 713
Alphabet project, 84-86, 578-580
BinaryTree project,
576-577
DatabaseTester project,
682-683
Errors project, 50, 575
MultipleFiles project,
22-23, 572
PowersOfTwo project, 715
Project Two project,
581-582
MakeNewContents member function,
CAddressBookDoc class,
477, 509-510
MakeNewWindow member function
x_CAddressBookDoc class,
478-479
x_CNewEntryDialog class,
493-494
MakeWindowName function template, 503-504
malloc function, comparing C++ with C, 137
MCollectible class, 437-438
member functions, 180-181, 191-193
Activate
CClockPane class, 395
CView class, 563
Advance, TRandom class,
327-328
AssignIdleChore,
CApplication class, 544

Symantec C++ Programming for the Macintosh
DoSave
  CDocument class, 554
  CPictureDoc class, 372-374
  CTextDoc class, 352-353
DoSaveAs
  CDocument class, 554
  CTextDoc class, 353
DoSaveFileAs
  CDocument class, 554
  CTextDoc class, 352
Draw
  CCalculatorPane class, 423-424
  CClockPane class, 397
  CPane class, 558
DrawAnalogClock,
  CClockPane class, 399-400
DrawClock, CClockPane class, 399
DrawDigitalClock,
  CClockPane class, 401
DrawLine, 400
DrawRectangleIn-
  RandomColor,
  CRandomRectanglePane class, 332
DrawRectangleIn-
  RandomPattern,
  CRandomRectanglePane class, 331-332
EmptyScrap, CClipboard class, 550
EnableCmd, CBartender class, 546
EndData
  CNewEntryDialog class, 489-490
  x_CNewEntryDialog class, 497
EnterCurrentNumber,
  CCalculatorPane class, 424
ExistsOnDisk, CFile class, 556
Exit, CApplication class, 545
FailOpen, x_CAddress-
  BookDoc class, 479
FileAlreadyOpen,
  x_CAddressBookApp class, 472-473
Find
  TDatabase class, 447
  TNode class, 443-445
FindPane, x_CNewEntry-
  Dialog class, 494-495
FitToEncaseFrame, CPane class, 558
FitToEnclosure, CPane class, 558
ForceClassReferences
  CAddressBookApp class, 469
  x_CAddressBookApp class, 469
GetAddressBookDoc, CAd-
  dressBookPane class, 484
GetCurrent, Tlterator class, 449
GetData, CClipboard class, 550
GetDatabase, CStream-
  ableDatabase class, 500
GetDigitalFontAndSize,
  CClockPane class, 401-402
GetDigitalString,
  CClockPane class, 401-402
GetDigitalStringLocation,
  CClockPane class, 401-402
GetDocTypeFromDialog,
  CAddressBookApp class, 470
GetFrame
  CRandomRectanglePane class, 330
  CView class, 563
GetFrom, CStreamable-
  Database class, 501
GetLength, CDataFile class, 551
GetMacPicture
  CBartender class, 375
  CPicture class, 560
GetName, CFile class, 556
GetRoot, Tlterator class, 449
GetTextHandle,
  CAbstractText class, 542
GetTextString, CFloafText class, 427-428
GetTitle, CWindow class, 564
GetValue, CFloafText class, 426
GetWCount, CDecorator class, 552
IAplication,
  CApplication class, 545
ICAddressBookApp,
  CAddressBookApp class, 468
ICAddressBookDoc,
  CAddressBookDoc class, 476-477, 509
ICNewEntryDialog,
  CNewEntryDialog class, 490
identically named,
  255-256
IDialogDirector,
  CDialogDirector class, 552
IDocument, CDocument class, 555
InitialSetup,
  CAddressBookPane class, 481-482
Insert
  TDatabase class, 447-448
  TNode class, 443, 445
InstallPanorama,
  CScrollPane class, 561
IsUndone, CTTask class, 562
Ix_CAddressBookApp,
  x_CAddressBookApp class, 468-469

Symantec C++ Programming for the Macintosh
Ix_CAddressBookDoc, 
  x_CAddressBookDoc 
class, 477
Ix_CNewEntryDialog, 
  x_CNewEntryDialog 
class, 493
LeftMost, TNode 
class, 443
MakeNewContents, 
  CAddressBookDoc class, 
  477, 509-510
MakeNewWindow 
  x_CAddressBookDoc 
class, 478-479
  x_CNewEntryDialog 
class, 493-494
Move, CWindow 
class, 564
MoveBackward, TIterator 
class, 451-452
MoveForward, TIterator 
class, 450-451
MoveToBeginning, 
  TIterator class, 450
MoveToEnd, TIterator 
class, 450
NewFile 
  CDocument class, 555
  CTextApp class, 347-348
NoMore, TIterator 
class, 449
Notify, CBureaucrat 
class, 548
NotifyClean, CBureaucrat 
class, 548
Open, CFile class, 556
OpenDocument 
  CApplication class, 545
  CTextApp class, 345-346
  x_CAddressBookApp 
class, 472
OpenFile 
  CDocument class, 555
  CPictureDoc class, 
  372-374
  CTextDoc class, 350-351
overriding, 216-217
  versus overloading 
  functions, 219-220
Perform, CChore 
class, 549
PlaceNewWindow, 
  CDecorator class, 552
PositionWindow, x_CAd­
dressBookDoc class, 479
Prepare, CView class, 563
ProviderChanged 
  CAddressBookPane 
class, 484-485
  x_CNewEntryDialog 
class, 498-499
PutData, CClipboard 
class, 550
PutTo, CStreamable-
Database class, 500-501
Quit, CApplication 
class, 545
ReadAll, CDataFile 
class, 551
ReadSome, CDataFile 
class, 551
Redo 
  CPictureClipboardTask 
class, 378
  CTTask class, 562
Refresh, CPane class, 559
RightMost, TNode 
class, 443
Run, CApplication 
class, 545
Select, CWindow 
class, 564
SetBorder, CPane 
class, 559
SetChanged, CBureaucrat 
class, 548
SetClickCmd, CButton 
class, 549
SetFontName, 
  CAbstractText class, 542
SetFontSize, CAbstractText 
class, 542
SetFontStyle, 
  CAbstractText class, 543
SetlnfoFromFile, 
  CSaveRestoreSizeWindow 
class, 391
SetMacPicture, CPicture 
class, 560
SetMark, CDataFile 
class, 551
SetMaxDisplayWidth, 
  CFloatText class, 426
SetPenSize, CPaneBorder 
class, 559
SetPosition, CPanorama 
class, 559
SetSizeRect, CWindow 
class, 564
SetTextHandle 
  CAbstractText class, 543
  CTextDoc class, 346
SetTextPtr, CAbstractText 
class, 543
SetTextString, 
  CAbstractText class, 543
SetTitle, CWindow 
class, 564
SetUpFileParameters 
  CApplication class, 
  545-546
  CTextApp class, 346
  x_CAddressBookApp 
class, 471
SetUpMenus, x_CAd­
dressBookApp class, 471
SetValue, CFloatText 
class, 427
SFSpecify, CFile class, 557
SimulateClick, CButton 
class, 549
Specify, CAbstractText 
class, 543
SpecifyFile, 
  CSaveRestoreSizeWindow 
class, 389-390
SpecifyHFS, CFile 
class, 557
StartUpAction 
  CApplication class, 546
  CCalculatorApp class, 
  412-413
  CClockApp class, 
  386-387
  CRectangleApp class, 
  334-335
StoreInfoInFile, class, 390
CSaveRestoreSizeWindow class, 557
ThrowOut, CFile class, 557
Undo
CPictureClipboardTask class, 378-379
CTask class, 562
UpdateData, x_CNewEntryDialog class, 497
UpdateDisplay, CCaculatorPane class, 424
UpdateEntryData, CAddressBookPane class, 483-484
UpdateMenus
CAddressBookPane, 482-483
CBureaucrat class, 548
CCalculatorPane class, 417-418
CClockPane class, 395-396
CCutCopyPastePicture class, 375-376
x_CAddressBookPane, 483
UpdateOffscreen, CClockPane class, 398
Value, TRandom class, 327-328
ValueInRange, TRandom class, 327-328
ValueToText, CFloatText class, 427
WriteAll, CDataFile class, 551
WriteSome, CDataFile class, 552
member 'xxx::xxxx' of class 'xxx' is not accessible error message, 201
memory allocating
comparing C++ with C, 137-139
dynamic, 225-234
too much for local variables, 305
deallocating, comparing C++ with C, 137-139
partitions, setting sizes, 98
Menu Bar command (Visual Architect Edit menu), 463-464
menu bars, class for controlling, 546
Menu Items dialog box, 463-464
menus class for controlling, 546
Clock, TickTock resources, 402-403
creating in Visual Architect, 463-464
Edit, modifying, 354-355
handling database-view items, 479-499
pop-up, source-file include files, 18-19
messages compile errors, 52-53
error
Different configuration for precompiled header, 115
function definitions with separate parameter lists are not allowed in C++, 307
member 'xxx::xxxx' of class 'xxx' is not accessible, 201
link errors, 53-55
warning, 304
can't nest comments, 305
eliminating, 56-58
non-const reference initialized to temporary, 306
possible extraneous ;, 305
possible unintended assignment, 304
returning address of automatic '66s', 307
unrecognized pragma, 307
use delete[] rather than delete[expr], expr ignored, 305
using operator++(int), 305-306
value of expression is not used, 305
variable '%%' used before set, 306
very large automatic, 305
methods, 180-181
MicroGA framework, 320-321
class hierarchy diagram, 321
modes, automatic for Debug menu commands, 69
modifiers, pascal type, comparing Symantec and standard C++, 515-516
Modify Read-Only command (File menu), 535-536
modular programming, 169-170
More Programming Pearls, 720
mounting Projector databases automatically, 538-539
mouse shortcuts in Browser, 258-259
Move member function, CWindow class, 564
MoveBackward member function, TIterator class, 451-452
MoveForward member function, TIterator class, 450-451
MoveToBeginning member function, TIterator class, 450
MoveToEnd member function, TIterator class, 450
MPW C++, converting from, 512
Multi-File Search dialog box, 36-39
multiple errors, reporting, 302-303
multiple inheritance, 224-225
Browser, 257-258
multiple-file projects, 22-31
compiling single files, 32-33
include files, determining locations for faster compile, 31-32
mixing both C and C++, 41-43
removing objects, 35
searching through files, 36-39
segmenting code, 39-40
translators, file extensions for, 43-46
MultipleFiles project, 22-24, 571-574
files for creating, 723
mutators, 202

temporary variables, inline functions versus #define macros, 149-150
nested comments, 305
New Entry command, handling, 479-499
New Entry dialog box creating, 486-488
fonts and styles, 489-499
new operator, 137-139
New Project dialog box, 8
New ProjectorDB Comment dialog box, 524-525
New View command (View menu), 486
NewFile function member
CDocument class, 555
CTextApp class, 347-348
template, 502
NewFileTypes function template, 506
NewHandleCanFail global function, 567
nodes, binary trees, 441-445
NoMore member function, TIterator class, 449
non-const reference initialized to temporary warning message, 306
>Note file extension, 44
Notify member function, CBureaucrat class, 548
NotifyClean member function, CBureaucrat class, 548
numbers command, 355
entering into displays, 424
floating-point
converting text to, 426-427
converting to text, 427
displaying in text fields, 424-428
phone
AddressBook, 453-510

holding information about, 438-441
power of two calculations, 524-539

.o file extension, 44
Object-Oriented Design: With Applications, 719-720
Object-Oriented Software Construction, 719
objects, 173-179
creating with class names, 569-570
disposing of, exception handling, 366-367
modeling on real world, 181
removing from multiple-file projects, 35
ODPF (OpenDoc Parts Framework), 322
Online Documentation folder, 4
OOD (object-oriented design), 179-183
OOP (object-oriented programming)
advantages, 168-169
allocating dynamic memory, 225-234
C++ naming conventions, 194-195
classes
access control to members, 198-202
derived, 214-218
comparing with procedural programming
 caching, 196-198
program components, 189-195
constructors, 204-209
history of development, 169-179

names
AddressBook, 453-510
holding information about, 438-441
of functions (MacsBug names) generating, 299-300
naming
C++ conventions for, 194-195
inline functions, 203-204
multiple inheritance, 224-225
polymorphism, 218-223
oopla (object-oriented programming and language application), 167
oops Libraries folder, 6
Open dialog box, setting parameters for, 467-473
Open member function, CFile class, 556
OpenDoc Parts Framework (ODPF), 322
OpenDocument member function
CApplication class, 545
CTextApp class, 345-346
x_CAddressBookApp class, 472
OpenFile function
member
CDocument class, 555
CPictureDoc class, 372-374
CTextDoc class, 350-351
template, 502-503
opening
documents, 350-351
files, 370
include files from source files, 18-19
operators
:: (scope resolution), 193
<< (insertion), 123
>> (extraction), 123
assignment, allocating dynamic memory, 230-233
delete, 137-139
copy propagation, 294-295
CSE (common subexpression) elimination, 276-280
dead assignment elimination, 271-273
dead variable elimination, 273-276
Hoist very busy expressions, 280-282
loop induction variables, 288-291
loop invariants, removing from loops after first evaluation, 282-287
looping through all possible optimizations, 296-297
Options command (Edit menu), 261-263
Symantec C++, 55
THINK Project Manager, 16-18, 44
Options dialog box
Code Optimization panel, 271-279
Compiler Settings panel, 267-270
components, 261-263
Debugging panel, 297-303
Language Settings panel, 263-267
Prefix panel, 307-308
Warning messages panel, 303-307
order of precedence, inline functions versus #define macros, 151
Out command (Debug menu), 67-68
overloading
constructors, 207-209
functions, 135-137
identically named, 256
operators, 123
syntax changes, 305-306
overloading
default
assignment operator, 230-231
copy constructors, 232-233
member functions, 216-217
versus overloading functions, 219-220
Pane menu commands
(Visual Architect)
Class, 461
Font, 458
Identifier, 459-460
Size, 458
Style, 458
panes, 328
borders, class for
drawing, 559
class for, 557-559
creating, 335, 348-350, 371-374
displaying pictures, class for, 560
scrollable
class for, 559
scrolling, class for, 560-561
with editable text, class for, 555-556
parameters
constructors with, 207-209
default, versus overloaded functions, 238-240
passing by reference, comparing C++ with C, 152-159
large, guidelines for, 236-237
partitions, setting sizes, 98
Pascal
calling functions, 157
string literals
comparing C++ and THINK C, 514
comparing Symantec and standard C++, 516
PascalObject class, 517
passing by reference, comparing C++ with C, 152-159
Paste command
enabling, 417-418
processing, 418-421
support for pictures, 374-376
pasting pictures, doing/undoing/redoing, 376-380
patterns (random), drawing rectangles in, 331-332
Perform member function, CChore class, 549
phone numbers
AddressBook, 453-510
holding information about, 438-441
Physics framework, 321
class hierarchy diagram, 321
PICTPeeker project, 357-358, 602-620
classes, 358-360
CAboutBoxApplication, 361-362
CBartender, 375-376
CCutCopyPastePicture, 374-376
CPictureApp, 363-365
CPictureClipboardTask, 376-380
CPictureDoc, 371-374
hierarchy diagram, 359
TCL_AUTO_DESTRUCT_OBJECT, 369
TDeleteOnFailure, 366-367
TDISposHandleOnFailure, 368
THandleFailure, 367-368
THandleLocker, 369-370
TWithFileOpen, 370
compiling and running, 358
main function, 380
project files for creating, 726-727
resources, 380-381
Picture.cp source file,
PICTPeeker project, 612
pictures
cutting/copying/pasting/clearing
doing/undoing/redoing, 376-380
support for, 374-376
displaying, 357-381
panes for displaying, class for, 560
Pink object-oriented environment, 322
PlaceNewWindow member function, CDecorator class, 552
pointers
double dereferencing, 75
guidelines for using, 236
to functions, conforming to ANSI, 264
polygons, class for, 225-227
polymorphism, 174-175, 218-223
pop-up menus, source-file include files, 18-19
PositionDialog global function, 567
PositionWindow function
member, x_CAddressBookDoc class, 479
template, 504
possible extraneous warning message, 305
possible unintended assignment warning message, 304
Power.cp source file,
PowersTwo project, 715
Power.h header file,
PowersTwo project, 715
PowersTwo project, 715
checking in files, 528-529, 531
checking out files, 529-530
old versions, 533-534
creating, 524-525
files for, 731
revisions
accessing information about, 531-533
checking out and modifying, 534-535
#pragma
segment, 512
unrecognized, 307
precedence, inline functions versus #define macros, 151
Precompile command
(Source menu), 113-114
precompiled header files, 105
changing, 111-113
comparing with standard include files, 107-110
creating, 113-115
determining need for, 111
including in individual source files, 110
rules for use, 106
standard, 107
customizing, 116
Precompiled headers
projects, files for creating, 725
Prefix panel, Options dialog box, 263, 307-308
Prepare member function, CView class, 563
PrintDatabase function, 452-453
private class members, 199-202
procedural programming, comparing with OOP (object-oriented programming)
caching, 196-198
program components, 189-195
profiler calls, 299
programming, publications for, 720-721
Programming Pearls, 720
programs
building, 101-103
class for handling communication between windows and, 553
debugging
breakpoints, 72-73
continuing through ends of functions, 67-68
continuing to next function call, 68-69
data validation, 78-80
displaying call chains, 70
enabling, 60-61
low-level debuggers, 80-81
starting, 61-62
stepping through, 63-65
tracing into statements with function calls, 65-67
viewing functions, 71-72
viewing/setting variable values, 73-76
lines with start/end points
C structures, 186-188
C++ structures, 188-189
main, comparing procedural and object-oriented, 192
running, 11-13
SourceServer, 520-539
THINK Debugger, 6
calling from within programs, 76-77
calling functions from, 77-80
turning on, 60-61
viewing on second monitor, 82
THINK Inspector, 6
THINK Project Manager, 6
ToolServer, 539-540
types, selecting, 97-98
Visual Architect, 6,
455-466
visual interfaces, creating, 455-466
see also projects
Project menu commands
Build Application, 101-103
Remove Objects, 35
Run, 11
Set Project Type, 96-101
Use Debugger, 60-61
(Project Models) folder, 5
Project One project, 581
Project Two project, 581-582
Project Update dialog box,
turning off, 16-17
Projector databases, 524
automatically mounting, 538-539
checking in files, 528-529, 531
checking out files, 529-530
old versions, 533-534
creating, 524-525
revisions
accessing information about, 531-533
checking out and modifying, 534-535
sharing, 535
projects, 524
adding resources, 89-90
AddressBook, 453-510,
683-714
project files for creating,
729-730
Alphabet, 84-87, 578-581
files for creating,
724-725
automatically mounting
Projector databases,
538-539
BinaryTree, 175-179,
576-577
files for creating, 724
C++Calc, 408-430,
644-672
project files for creating,
728-729
creating, 7-10, 521-522
shortcuts for, 18-19
DatabaseTester, 434-453,
672-683
project files for creating,
729
Errors, 50-52, 574-576
files for creating, 724
GoodnightMoon, 42-43
Hello World
creating, 8-10
editing source files,
10-11
running program, 11-13
multiple-file
adding compiling files, 22-31
single files, 32-33
include files,
locationing for faster compiles, 31-32
mixing both C and C++, 41-43
removing objects, 35
searching through files,
36-39
segmenting code, 39-40
syntax errors, checking for in single files, 33-34
translators, file extensions for, 43-46
MultipleFiles, 22-24, 571-574
files for creating, 723
PICTPeeker, 358-381, 602-620
project files for creating, 726-727
PowersOfTwo, 524-539, 715
files for creating, 731
Precompiled headers, files for creating, 725
Project One, 581
Project Two, 581-582
RandomRectangle, 324-338, 582-591
project files for creating, 725-726
TextTyper text editor, 340-355, 592-602
project files for creating, 726
TickTock, 383-403, 621-643
project files for creating, 727-728
types, setting, 96-101
Projects folder, 6
propagation constant, 291-294
copy, 294-295
protected class members, 199-202
provider objects, class for contacting dependents, 550
ProviderChanged member function
CAddressBookPane class, 484-485
x_CNewEntryDialog class, 498-499
public class members, 199-202
publications
C, 717-718
C++, 718-719
Macintosh, 722
object-oriented design, 719-720
programming, 719
programming, 720-721
PutData member function,
CClipboard class, 550
PutTo member function,
CStreamableDatabase class,
500-501

QuickApp framework,
315-320
class hierarchy diagram, 320
Quit member function,
CApplication class, 545
quotes (" ") (double), in
#include statements, 31

.R file extension, 44, 92
random colors, drawing rectangles in, 332
random patterns, drawing rectangles in, 331-332
RandomRectangle project,
324, 582-591
classes, 325-326
CRandomRectanglePane,
328-332
CRectangleApp, 333-336
TRandom, 327-328
main function, 336-337
project files for creating,
725-726
resources, 337-338
running and testing, 325
setting up and compiling,
324-335
ReadAll member function,
CDataFile class, 551
ReadContents function
template, 504
ReadDocument function
template, 504-505
ReadSome member function,
CDataFile class, 551
rectangle drawing program,
324-338
Rectangle.cp source file,
RandomRectangle project,
585
Rectangle.r resource file,
RandomRectangle project,
588-591
rectangles, drawing
in random colors, 332
in random patterns, 331-332
recursive functions,
inlining, 148-149
red-black binary trees,
173-179
RedBlackTree.cp source file,
176-177
RedBlackTree.h header
file, 176
Redo member function
CPictureClipboardTask
class, 378
CTask class, 562
redoing cutting/copying/
pasting/clearing pictures,
376-380
references
comparing C++ with C,
152-159
guidelines for using,
236-238
initializing to temporary
variables, 306
Refresh member function,
CPane class, 559
Release Notes folder, 4
Remove Objects command (Project menu), 35
removing objects from multiple-file projects, 35
unreferenced functions, 102-103
reporting errors multiple, 302-303
setting number of per file, 55-56
ResEdit graphical resource editor
comparing with THINK Rez, 92-93
creating resources, 87-89
ResizeHandleCanFail global function, 567
resource files
Alphabet.r, Alphabet project, 580-581
Calculator.r, C++Calc project, 648-652
Clock.r, TickTock project, 634-638
Rectangle.r, RandomRectangle project, 588-591
Text.r, TextTyper project, 598-602
ResourceDefinitions.h header file, Alphabet project, 86, 580
resources adding to projects, 89-90
AddressBook project, 510
C++Calc project, 430
code, 98
comparing THINK Rez with ResEdit, 92-93
creating, 87-92
PICTPeeker project, 380-381
RandomRectangle project, 337-338
TextTyper text editor, 354-355
TickTock project, 402-403
restoring window positions/locations at opening, 388-391
returning address of automatic ‘%s’ warning message, 307
Reverse Polish Notation (RPN) system, 406-430
Reverse.cp source file Errors project, 51, 575-576
MultipleFiles project, 23, 573
Reverse.h header file Errors project, 51, 576
MultipleFiles project, 23, 574
RightMost member function, TNode class, 443
routines, see functions RPN (Reverse Polish Notation) system, 406-430
.rsrc file extension, 44
Run command (Project menu), 11
Run member function, CApplication class, 545
scope, jurisdiction of, 130-131
scope resolution (::) operator, 193
screen savers, rectangle drawing program, 324-338
Scripting folder, 4
scripts, executing, 539-540
scrollable panes class for, 559
scrolling, class for, 560-561
Search menu commands
Find, 36-39
Find Again, 38
Find In Next File, 38
Go To Next Match, 39
Go To Previous Match, 39
searching through multiple files, 36-39
segmenting code, 39-40
Select All command (Visual Architect Edit menu), 458
Select member function, CWindow class, 564
semicolons (;) in declarations, 191
Set Project Type command (Project menu), 96-101
SetAllocation global function, 567
SetBorder member function, CPane class, 559
SetChanged member function, CBureaucrat class, 548
SetClickCmd member function, CButton class, 549
SetFontName member function, CAbstractText class, 542
SetFontSize member function, CAbstractText class, 542
SetFontStyle member function, CAbstractText class, 543
S

SANE (Standard Apple Numerics Environment), 268
comparing Symantec and standard C++, 515
Save As command, handling options for, 353
Save Changes dialog box, turning off, 17-18
Save command, handling options for, 352-353
saving handling options for, 352-353
window positions/locations at closing and restoring at opening, 388-391

Symantec C++ Programming for the Macintosh
SetInfoFromFile member function, CSaveRestoreSizeWindow class, 391
SetMacPicture member function, CPicture class, 560
SetMark member function, CDataFile class, 551
SetMaxDisplayWidth member function, CFloatText class, 426
SetPenSize member function, CPaneBorder class, 559
SetPosition member function, CPanorama class, 559
SetRectangle function, calling, 157
SetSizeRect member function, CWindow class, 564
SetTextHandle member function
CAbstractText class, 543
CTextDoc class, 351
setTextPtr member function, CAbstractText class, 543
SetTextString member function, CAbstractText class, 543
SetTitle member function, CWindow class, 564
SetUpFileParameters member function
CApplication class, 545-546
CTextApp class, 346
x_CAddressBookApp class, 471
SetUpMenus member function, x_CAddressBookApp class, 471
SetValue member function, CFloatText class, 427
sfFileTypes data member, CApplication class, 543
sfNumTypes data member, CApplication class, 544
SFSpecify member function, CFile class, 557
-sharing Projector databases, 535
signatures for programs, 346
signed char types versus unsigned char types, 267
SimulateClick member function, CButton class, 549
SingleObject class, 512
Size command (Visual Architect Pane menu), 458
SIZE flags, setting, 98-100
sizeRect data member, CWindow class, 563
Smart Link, 102-103
Software Tools, 721
source code
control, uses for, 520-521
ing, 466-485
functions, displaying in Browser, 254
generating, 464-466
placing string literals in, 270
segmenting, 39-40
source files
Average.cp
Errors project, 51-52, 574
MultipleFiles project, 24, 571-572
BinaryTree.c, 175-176
C++Routine.cp, 43
CAboutBoxApplication.cp
C++Calc project, 644
PICTPeeker project, 602
TickTock project, 621
CAddressBookApp.cp, AddressBook project, 683-686
CAddressBookDoc.cp, AddressBook project, 687-689
CAddressBookPane.cp, AddressBook project, 691-693
Calculator.cp, C++Calc project, 645-647
CalculatorMain.cp, C++Calc project, 653
CBUTTONWithFont-AndSize.cp, C++Calc project, 653-654
CCalculatorApp.cp, C++Calc project, 654-655
CCalculatorPane.cp, C++Calc project, 656-663
CClockApp.cp, TickTock project, 622-623
CClockPane.cp, TickTock project, 624-631
CCutCopyPastePicture.cp, PICTPeeker project, 603-604
CDawdleBureaucrat-Chore.cp, TickTock project, 632
CFloatText.cp, C++Calc project, 665-666
Clock.cp, TickTock project, 633
CNewEntryDialog.cp, AddressBook project, 695-698
CPictureApp.cp, PICTPeeker project, 605-606
CPictureClipboardTask.cp, PICTPeeker project, 606-608
CPictureDoc.cp, PICTPeeker project, 608-612
CQuitApplicationChore.cp
C++Calc project, 667
TickTock project, 638-639
CQuitOnCloseWindow.cp
C++Calc project, 668
TickTock project, 639
CRandomRectanglePane.cp, RandomRectangle project, 582-584
CRectangleApp.c, RandomRectangle project, 584-585
CRoutine.c, 43
CSaveRestoreSizeWindow.cp, C++Calc project, 668-670
CStream_CStreamableDatabase.cpp, AddressBook project, 702
CStreamableDatabase.cp, AddressBook project, 699-701
CTextApp.cp, TextTyper project, 592-593
CTextDoc.cp, TextTyper project, 593-597
Database.cp
  AddressBook project, 702-707
  DatabaseTester project, 672-677
debugging
  breakpoints, 72-73
  continuing through ends of functions, 67-68
  continuing to next function call, 68-69
  data validation, 78-80
  displaying call chains, 70
  enabling, 60-61
  low-level debuggers, 80-81
  starting programs, 61-62
  stepping through, 63-65
  tracing into statements with function calls, 65-67
  viewing functions, 71-72
  viewing/setting variable values, 73-76
documentation
  reading, 10-11
Entry.cp
  AddressBook project, 709-711
  DatabaseTester project, 679-681
Factorial.cp, MultipleFiles project, 24, 573
Foo.cp, Project One project, 581
FooBar.cp, Project Two project, 582
  including precompiled header files, 110
Main.c, 178
Main.cp
  AddressBook project, 713
  Alphabet project, 84-86, 578-580
  BinaryTree project, 576-577
  DatabaseTester project, 682-683
  Errors project, 50, 575
  GoodnightMoon project, 42
  MultipleFiles project, 22-23, 572
  PowersOfTwo project, 715
  Project Two project, 581-582
Picture.cp, PICTPeeker project, 612
Power.cp, PowersOfTwo project, 715
Rectangle.cp, RandomRectangle project, 585
Reverse.cp
  Errors project, 51, 575-576
  MultipleFiles project, 23, 573
SplayTree.c, 177
TDisposeHandleOnFailure.cp, PICTPeeker project, 612
Text.cp, TextTyper project, 597
THandleFailure.cp, PICTPeeker project, 613
THandleLocker.cp, PICTPeeker project, 613-614
TRandom.cp, RandomRectangle project, 586
TWithFileOpen.cp, PICTPeeker project, 614
Utilities.cp, TickTock project, 643
Source menu commands
  Add Files, 26-27
  Add Window, 25-26
  Add Window-name, 26
  Browser, 252
  Check Syntax, 33-34
  Compile, 32-33
  Precompile, 113-114
SourceServer
  Check In, 525-528
  Check In 'xxx', 531
  Check Out 'xxx', 529
  Launch SourceServer, 523
  New ProjectorDB, 524-525
  Quit SourceServer, 535
  Send Command, 537
SourceServer program
  Check In dialog box, 525-528
  checking in files, 528-529, 531
  checking out files, 529-530
    old versions, 533-534
  history, 521
  modifying files without access to Projector databases, 535-536
  Projector databases
    creating, 524-525
    sharing, 535
  projects, creating, 521-522
  revisions
    accessing information about, 531-533
    checking out and modifying, 534-535
sending commands to, 537-538
source code control, uses for, 520-521
starting, 523-524
space, optimizing
CSE (common subexpression) elimination, 276-278
Hoist very busy expressions, 280-282
looping through all possible optimizations, 296-297
removing loop invariants from loops after first event, 283-285
Specify member function, CAbstractText class, 543
SpecifyFile member function, CSaveRestoreSizeWindow class, 389-390
SpecifyHFS member function, CFile class, 557
speed, optimizing
CSE (common subexpression) elimination, 276-280
dead assignment elimination, 271-273
dead variable elimination, 273-276
loop induction variables, 288-291
looping through all possible optimizations, 296-297
removing loop invariants from loops after first event, 283-285
SplayTree.c source file, 177
SplayTree.h header file, 177
stack, generating frames, 298
Standard Apple Numerics Environment (SANE), 268
comparing Symantec and standard C++, 515
Standard Libraries folder, 6
standard precompiled header files, 107
customizing, 116
starting
programs with debugging enabled, 61-62
SourceServer program, 523-524
StartUpAction member function
CApplication class, 546
CCalculatorApp class, 412-413
CClockApp class, 386-387
CRectangleApp class, 334-335
Step command (Debug menu), 63-65
Stop command (Debug menu), 69
StorelnfoInFile member function,
CSaveRestoreSizeWindow class, 390
streamable documents, supporting, 509-510
streams
byte, class for, 561
cin, 123
comparing C++ code with C code, 125-126
cout, 123
making applications streamable, 499-509
syntax, 122-125
user extensibility, 125
string constants, storing, 101
string literals
Pascal
comparing C++ and THINK C, 514
comparing Symantec and standard C++, 516
placing in code, 270
strings, returning text as, 427-428
Symantec C++
comparing with standard C++, 514-517
converting from Apple’s MPW C++, 512
THINK C, 513-514
installing, 7
see also C++
Symantec C++ command (Options submenu), 55
Symantec C++ for Macintosh folder, 4-6
Symantec C++ Options dialog box, 55-56
Debugging panel, 81
syntax errors, checking for, 33-34
T

Taligent Pink object-oriented environment, 322

768
Symantec C++ Programming for the Macintosh
tasks, class for, 561-562
TCalculator class, 409-412
TCL (THINK Class Library),
314-315
classes
CAbstractText, 542-543
CApplication, 543-546
CBartender, 546
CBureaucrat, 547-548
CButton, 548-549
CChore, 549
CClipboard, 549-550
CCollaborator, 550
CDataFile, 551-552
CDecorator, 552
CDialogDirector, 552
CDiaglogText, 552
CDirector, 553
CDocument, 553-555
CEditText, 555-556
CFile, 556-557
CPane, 557-559
CPaneBorder, 559
CPanorama, 559
CPicture, 560
CScrollPane, 560-561
CStream, 561
CTask, 561-562
CView, 562-563
CWindow, 563-564
hierarchy diagram, 316, 541-542
global
functions, 566-567
variables, 565
macros, 568-570
TCL Demos folder, 4
TCL precompiled header
file, 107
customizing, 116
TCL_AUTO_DESTRUCT_OBJECT
class, 369
TCL_AUTO_DESTRUCT_OBJECT
macro, 568
TCL_DECLARE_CLASS
macro, 569
temporary variables
naming, inline functions
versus #define macros,
149-150
references initializing
to, 306
TEntry class, 441
testing
AddressBook project, 454,
485-486
addresses and databases,
434-453
C++Calc project, 406-407
DatabaseTester project,
434-435
RandomRectangle
project, 325
text
tex (Computers 
& Typesetting
Ser.: Vol. B), 721
button titles, setting
fonts/sizes, 428-429
converting
floating-point numbers
to, 427
to floating-point
numbers, 426-427
displaying, class for,
542-543
editable, class for panes
with, 555-556
in dialog boxes, class for
handling, 552
returning as strings,
427-428
text fields, displaying
floating-point numbers,
424-428
Text.cp source file,
TextTyper project, 597
Text.r resource file,
TextTyper project, 598-602
unrecognized pragma
warning message, 307
unsigned char types versus
signed char types, 267
UpdateData member
function,
x_CNewEntryDialog
class, 497
UpdateDisplay member
function, CCalculatorPane
class, 424
UpdateEntryData member
function,
CAddressBookPane class, 483-484
UpdateMenus member
function
CAddressBookPane, 482-483
CBureaucrat class, 548
CCalculatorPane class, 417-418
CClockPane class, 395-396
CCutCopyPastePicture
class, 375-376
x_CAddressBookPane, 483
UpdateOffscreen member
function, CClockPane
class, 398
updating displays, 424
Use Debugger command
(Project menu), 60-61
use delete[] rather than
delete[expr], expr ignored
warning message, 305
used before set warning, 133
user extensibility, 125
using operator++() (or --)
instead of missing
operator++(int) warning
message, 305-306
Utilities folder, 4
Utilities.cp source file,
TickTock project, 643
Utilities.h header file,
TickTock project, 643

V

v-tables (virtual-pointer
tables), 221-223
validating data, 78-80
Value member function,
TRandom class, 327-328
value of expression is not
used warning message, 305
ValueInRange member
function, TRandom class,
327-328
values
constant propagation,
291-294
default, guaranteed,
204-209
variables, viewing/setting
during debugging, 73-76
ValueToText member func-
tion, CFloatText class, 427
variable ‘%s’ used before set
warning message, 306
variables
assignments, deleting
dead, 271-273
const, versus #define
macros, 139-141, 151-152
constant propagation,
291-294
copy propagation, 294-295
dead, deleting, 273-276
global, 565
accessing, 101
ADT (abstract data
type), 171-173
data types, 170-171
gBartender, 375
modular programming,
169-170
local
allocating too much
memory, 305
returning addresses
of, 307
locating definitions, C++
versus C, 128-131
loop induction, 288-291
polymorphism, 218-223
swapping without
temporary variables,
algorithm for, 151
temporary
naming, inline func-
tions versus #define
macros, 149-150
references initializing
to, 306
values, viewing/setting
during debugging, 73-76
very large automatic
warning message, 305
View Info command (View
menu), 486
Visual Architect, 456
View menu commands
New View, 486
View Info, 486
Visual Architect, 456
views, 455
adding items to, 457-461
class for, 562-563
creating, 455-457
database, handling menu
items, 479-499
editing, 459
virtual functions, 221-223
Visual Architect, 6, 455
application information,
setting, 464
classes, creating, 461-462
commands
adding to menus,
463-464
creating, 462-463
menus, creating, 463-464
source code, generating,
464-466
views
adding items to, 457-461
creating, 455-457
Visual Architect Macro Files
folder, 6
W

wantsClicks data member
CCalculatorPane class, 416
CView class, 562
warning messages, 304
can't nest comments, 305
eliminating, 56-58
errors, comparing C++
with C, 133-135
non-const reference
initialized to temporary,
306
possible extraneous \;, 305
possible unintended
assignment, 304
returning address of
automatic '%%', 307
unrecognized pragma, 307
use delete[] rather than
delete[expr], expr
ignored, 305
used before set, 133
using operator++() (or --)
instead of missing
operator++(int), 305-306
value of expression is not
used, 305
variable '%%' used before
set, 306
very large automatic, 305
Warning messages panel,
Options dialog box, 263,
303-307
Weinberg, Gerald, books
authored by, 721

windows
arranging, class for, 552
Browser, creating, 250-254
calculator, creating,
412-413
Check Out, 531-533
class for handling com-
munication between
programs and, 553
closing, exiting TickTock
project by, 387-388
Compile Errors, 13-14,
52-53
creating, 335, 348-350,
371-374
Debugger Data, viewing/
setting variable values,
73-76
Debugger Source, 61-63
displaying call chains, 70
viewing functions, 71-72
displaying, class for,
563-564
keeping documents in
sync with, 473-479
Link Errors, 15, 54
panes, 328
class for, 557-559
class for displaying
pictures in, 560
class for drawing
borders around, 559
class for scrollable, 559
class for scrolling
scrollable panes,
560-561
creating, 335, 348-350,
371-374
with editable text, class
for, 555-556
WindowToContents
function template, 508
WriteAll member function,
CDataFile class, 551
WriteContents function
template, 505
WriteDocument function
template, 505-506
WriteSome member func-
tion, CDataFile class, 552
Writing Efficient Programs,
720

X—Z
\x, conforming to ANSI, 265
x_CAddressBookApp class,
467-473
x_CAddressBookDoc class,
474-479
x_CAddressBookPane class,
481-499
x_CNewEntryDialog class,
The Road to Advanced Software Development Runs Through Monterey

at the Symantec Professional Developers Conference
September 19-21, 1994 at the Hyatt Regency Monterey

Take a career-boosting shortcut through Monterey! Attend this special conference focusing on the programming tools and techniques developers will need for Chicago, NT, Power Macintosh, Power PC, and client/server application development.

You'll pick up free software and you'll spend three full days listening to leading industry experts reveal specific insight and knowledge that will help you meet the challenge of developing within the new operating system paradigms.

Receive up-to-the-minute information on this great new conference from Symantec. Just fill out and return the coupon or call 800-441-8826 Now!

Name
Title
Company
Street
City, State, Zip
Phone                Fax
Email

For more information contact: Symantec Professional Developers Conference
600 Harrison Street, San Francisco, CA 94107
Call 800-441-8826, Fax 415-905-2222
GO AHEAD. PLUG YOURSELF INTO MACMILLAN COMPUTER PUBLISHING.

Introducing the Macmillan Computer Publishing Forum on CompuServe®

Yes, it's true. Now, you can have CompuServe access to the same professional, friendly folks who have made computers easier for years. On the Macmillan Computer Publishing Forum, you'll find additional information on the topics covered by every Macmillan Computer Publishing imprint—including Que, Sams Publishing, New Riders Publishing, Alpha Books, Brady Books, Hayden Books, and Adobe Press. In addition, you'll be able to receive technical support and disk updates for the software produced by Que Software and Paramount Interactive, a division of the Paramount Technology Group. It's a great way to supplement the best information in the business.

WHAT CAN YOU DO ON THE MACMILLAN COMPUTER PUBLISHING FORUM?

Play an important role in the publishing process—and make our books better while you make your work easier:

☐ Leave messages and ask questions about Macmillan Computer Publishing books and software—you're guaranteed a response within 24 hours
☐ Download helpful tips and software to help you get the most out of your computer
☐ Contact authors of your favorite Macmillan Computer Publishing books through electronic mail
☐ Present your own book ideas
☐ Keep up to date on all the latest books available from each of Macmillan Computer Publishing's exciting imprints

JOIN NOW AND GET A FREE COMPUSERVE STARTER KIT!

To receive your free CompuServe Introductory Membership, call toll-free, 1-800-848-8199 and ask for representative #597. The Starter Kit Includes:

☐ Personal ID number and password
☐ $15 credit on the system
☐ Subscription to CompuServe Magazine

HERE’S HOW TO PLUG INTO MACMILLAN COMPUTER PUBLISHING:

Once on the CompuServe System, type any of these phrases to access the Macmillan Computer Publishing Forum:

GO MACMILLAN
GO QUEBOOKS
GO SAMS
GO NEWRIDERS
GO ALPHA

GO BRADY
GO HAYDEN
GO QUESOFT

Once you're on the CompuServe Information Service, be sure to take advantage of all of CompuServe's resources. CompuServe is home to more than 1,700 products and services—plus it has over 1.5 million members worldwide. You'll find valuable online reference materials, travel and investor services, electronic mail, weather updates, leisure-time games and hassle-free shopping (no jam-packed parking lots or crowded stores).

Seek out the hundreds of other forums that populate CompuServe. Covering diverse topics such as pet care, rock music, cooking, and political issues, you're sure to find others with the same concerns as you—and expand your knowledge at the same time.
REPLACEMENT ORDER FORM

In the event that the disk bound to this book is defective, Macmillan Computer Publishing will send you a replacement disk free of charge.

Please fill out the information below and return this page (or a copy of this page) to the address listed below with your original disk. Please print clearly.

BOOK TITLE __________________________ ISBN __________________

NAME __________________________ PHONE __________________

COMPANY __________________________ TITLE __________________

ADDRESS __________________________

CITY __________ STATE __________ ZIP __________

Macmillan Computer Publishing, 201 West 103rd Street, Indianapolis, IN 46290. ATTN: Customer Service Department

Symantec C++ Programming for the Macintosh
LIMITED WARRANTY REGISTRATION CARD

In order to preserve your rights as provided in the limited warranty, this information must be on file with MCP within 30 days of purchase.

Please fill in the information requested:

BOOK TITLE __________________________ ISBN __________________

NAME __________________________ PHONE NUMBER __________________

ADDRESS __________________________

CITY __________________________ STATE __________ ZIP __________

COMPUTER BRAND & MODEL __________ DOS VERSION __________ MEMORY __________ M

Where did you purchase this product?

DEALER NAME __________________________ PHONE NUMBER __________________

ADDRESS __________________________

CITY __________________________ STATE __________ ZIP __________

PURCHASE DATE ___________ PURCHASE PRICE __________

How did you learn about this product? (Check as many as applicable.)

STORE DISPLAY ______ SALESPERSON ______ MAGAZINE ARTICLE ______ ADVERTISEMENT ______

OTHER (Please explain) __________________________

How long have you owned or used this computer?

LESS THAN 30 DAYS ______ LESS THAN 6 MONTHS _______ 6 MONTHS TO A YEAR _______ OVER 1 YEAR ______

What is your primary use for the computer?

BUSINESS ______ PERSONAL ______ EDUCATION ______ OTHER (Please explain) ______

Where is your computer located?

HOME ______ OFFICE ______ SCHOOL ______ OTHER (Please explain) ______

Macmillan Computer Publishing, 201 West 103rd Street, Indianapolis, IN 46290. ATTN: Customer Service Department
Important! Read Before Opening Sealed Disk

END USER LICENSE AGREEMENT

The software in this package is provided to You on the condition that You agree with MACMILLAN COMPUTER PUBLISHING ("MCP") to the terms and conditions set forth below. Read this End User Agreement carefully. You will be bound by the terms of this agreement if you open the sealed disk. If You do not agree to the terms contained in this End User License Agreement, return the entire product, along with your receipt, to Hayden, Macmillan Computer Publishing, 201 West 103rd Street, Indianapolis, IN, 46290 USA, Attn: Refunds, and your purchase price will be refunded.

MCP grants, and You hereby accept, a personal, nonexclusive license to use this software program and associated documentation in this package, or any part of it ("Licensed Product"), subject to the following terms and conditions:

1. License

The license granted to You hereunder authorizes You to use the Licensed Product on any single computer system. A separate license, pursuant to a separate End User License Agreement, is required for any other computer system on which You intend to use the Licensed Product.

2. Term

This End User License Agreement is effective from the date of purchase by You of the Licensed Product and shall remain in force until terminated. You may terminate this End User License at any time by destroying the Licensed Product together with all copies in any form made by You or received by You. Your right to use or copy the Licensed Product will terminate if You fail to comply with any of the terms or conditions of this End User License Agreement. Upon such termination, You shall destroy the copies of the Licensed Product in your possession.

3. Restriction Against Transfer

This End User License Agreement, and the Licensed Product, may not be assigned, sublicensed, or otherwise transferred by You to another party unless the other party agrees to accept the terms and conditions of the End User License Agreement. If You transfer the Licensed Product, You must at the same time either transfer all copies, whether in printed or machine-readable form, to the same party or destroy any copies not transferred.

4. Restrictions Against Copying or Modifying the Licensed Product

The Licensed Product is copyrighted and except for certain limited uses as noted on the copyright page, may not be further copied without the prior written approval of MCP. You may make one copy for backup purposes provided You reproduce and include the complete copyright notice on the backup copy. Any unauthorized copying is in violation of this Agreement and may also constitute a violation of the United States Copyright Law for which You could be liable in a civil or criminal lawsuit. You may not use, transfer, copy, or otherwise reproduce the Licensed Product, or any part of it, except as expressly permitted in this End User License Agreement.

5. Protection and Security

You shall take all reasonable steps to safeguard the Licensed Product and to ensure that no unauthorized person shall have access to it and that no unauthorized copy of any part of it in any form shall be made.

6. Limited Warranty

If You are the original consumer purchaser of a disk and it is found to be defective in materials or workmanship (which shall not include problems relating to the nature or operation of the Licensed Product) under normal use, MCP will replace it free of charge (or, at MCP's option, refund your purchase price) within 30 days following the date of purchase. Following the 30-day period, and up to one year after purchase, MCP will replace any such defective disk upon payment of a $5 charge (or, at MCP's option, refund your purchase price), provided that the Limited Warranty Registration Card has been filed within 30 days following the date of purchase. Any request for replacement of a defective disk must be accompanied by the original defective disk and proof of date of purchase and purchase price. MCP shall have no obligation to replace a disk (or refund your purchase price based on claims of defects in the nature or operation of the Licensed Product).

The software program is provided "as is" without warranty of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with You. Should the program prove defective, You (and not MCP) assume the entire cost of necessary servicing, repair or correction.

Some states do not allow the exclusion of implied warranties, so the above exclusion may not apply to You. This warranty gives You specific legal rights, and You may also have other rights which vary from state to state.

MCP does not warrant that the functions contained in the program will meet your requirements or that the operation of the program will be uninterrupted or error free. Neither MCP nor anyone else who has been involved in the creation or production of this product shall be liable for any direct, indirect, incidental, special, or consequential damages, whether arising out of the use or inability to use the product, or any breach of a warranty, and MCP shall have no responsibility except to replace the disk pursuant to this limited warranty (or, at its option, provide a refund of the purchase price).

No sales personnel or other representative of any party involved in the distribution of the Licensed Product is authorized by MCP to make any warranties with respect to the disk or the Licensed Product beyond those contained in this Agreement. Oral statements do not constitute warranties, shall not be relied upon by You, and are not part of this Agreement. The entire agreement between MCP and You is embodied in this Agreement.

7. General

If any provision of this End User License Agreement is determined to be invalid under any applicable statute or rule of law, it shall be deemed omitted and the remaining provisions shall continue in full force and effect. This End User License Agreement is to be governed and construed in accordance with the laws of the State of New York.

Macmillan Computer Publishing and the authors make no warranties, express or implied in connection with the software, and expressly exclude all warranties of fitness for a particular purpose. Macmillan Computer Publishing and the authors shall have no liability for consequential, incidental, or exemplary damages.
Assumes no previous knowledge of C++!

Symantec C++ Programming for the Macintosh, Second Edition, is a tutorial for the most popular programming language for the Mac environment. This book is divided into three, easy-to-follow sections. Get up and running in the Symantec C++ environment, master the concepts and techniques of object-oriented programming, and learn how to shorten your development time with C++ and application frameworks. In addition, appendixes include portability considerations, a complete listing of source code, and a reference manual on the Think Class Library. With this practical, hands-on approach, you'll be writing, editing, and compiling your own programs with proficiency.

- Explore the Symantec C++ environment, from debugging a program and using resource utilities to building applications and creating projects.
- Design programs for compatibility with multiple application frameworks.
- Learn how to use the new Visual Architect, Inspector, and templates.
- Make your programs easier to write, to modify, and to maintain by designing them as a collection of objects.
- Give your applications more features and faster execution with application frameworks.

The Companion Disk Includes:

- C++ Calc, a pop-up calculator
- Tick-Tock, a clock application
- TextTyper, a text-editing program
- PICTPeeker, a graphics-viewing program
- RandomRectangle, a drawing program

SAMS PUBLISHING

Category: Programming
User Level: Beginning-Intermediate-Advanced
Covers: All versions through 6.5 for Macintosh

$45.00 USA / $61.95 CAN

ISBN 1-56686-155-1

9781566861557