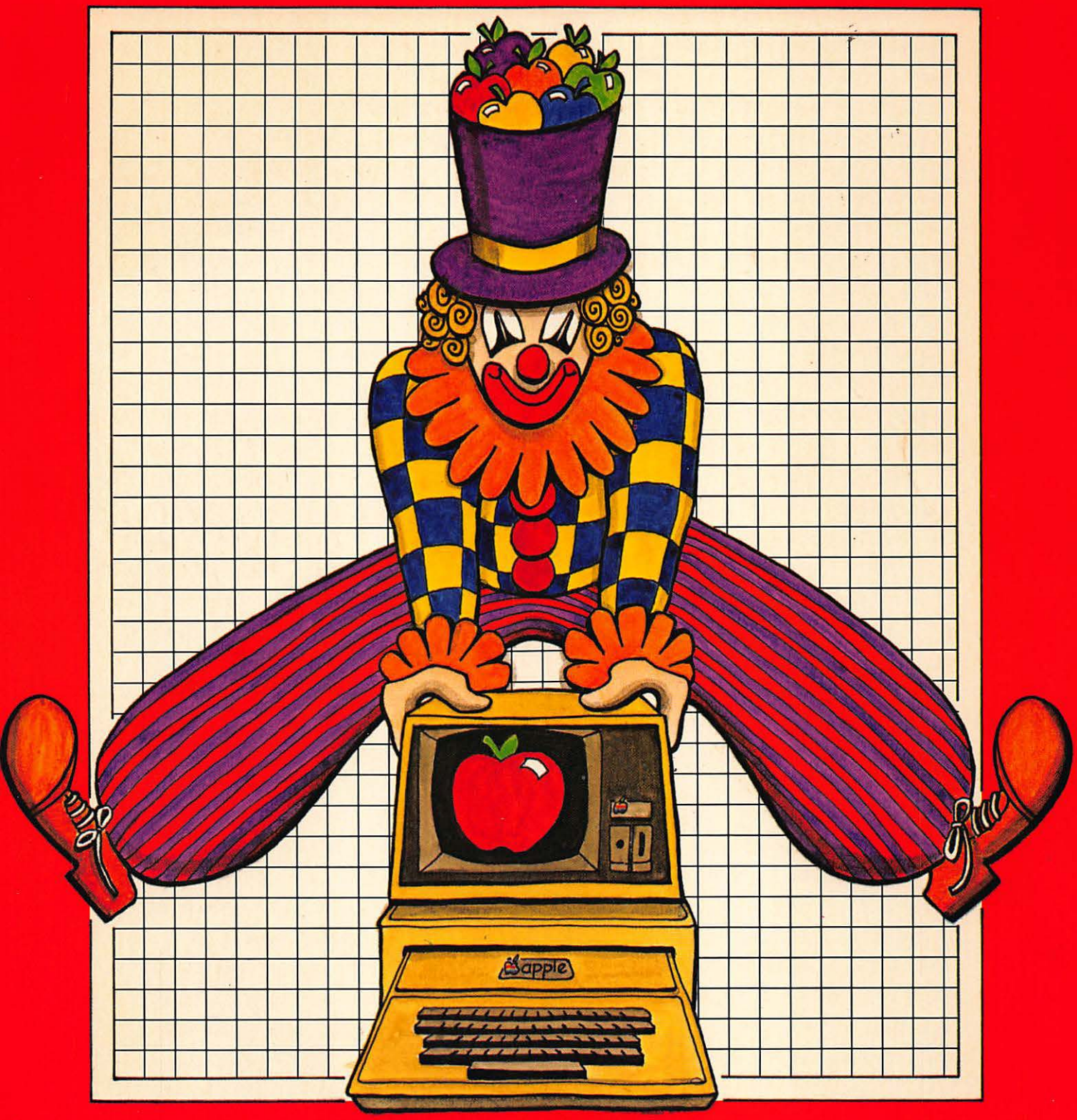
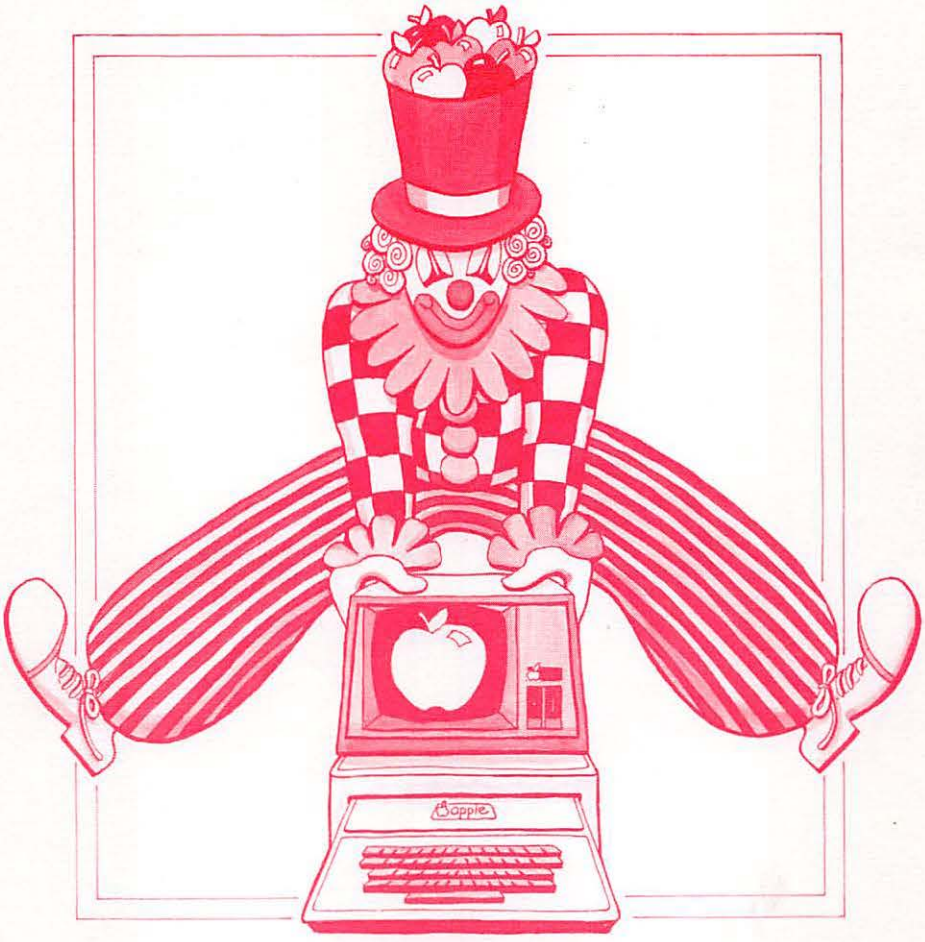
Sharon Boren

An Apple® For Kids



AN APPLE® FOR KIDS



AN APPLE® FOR KIDS

Sharon Boren

dilithium Press
Beaverton, Oregon

10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging in Publication Data

Boren, Sharon, 1956-
An Apple for kids.

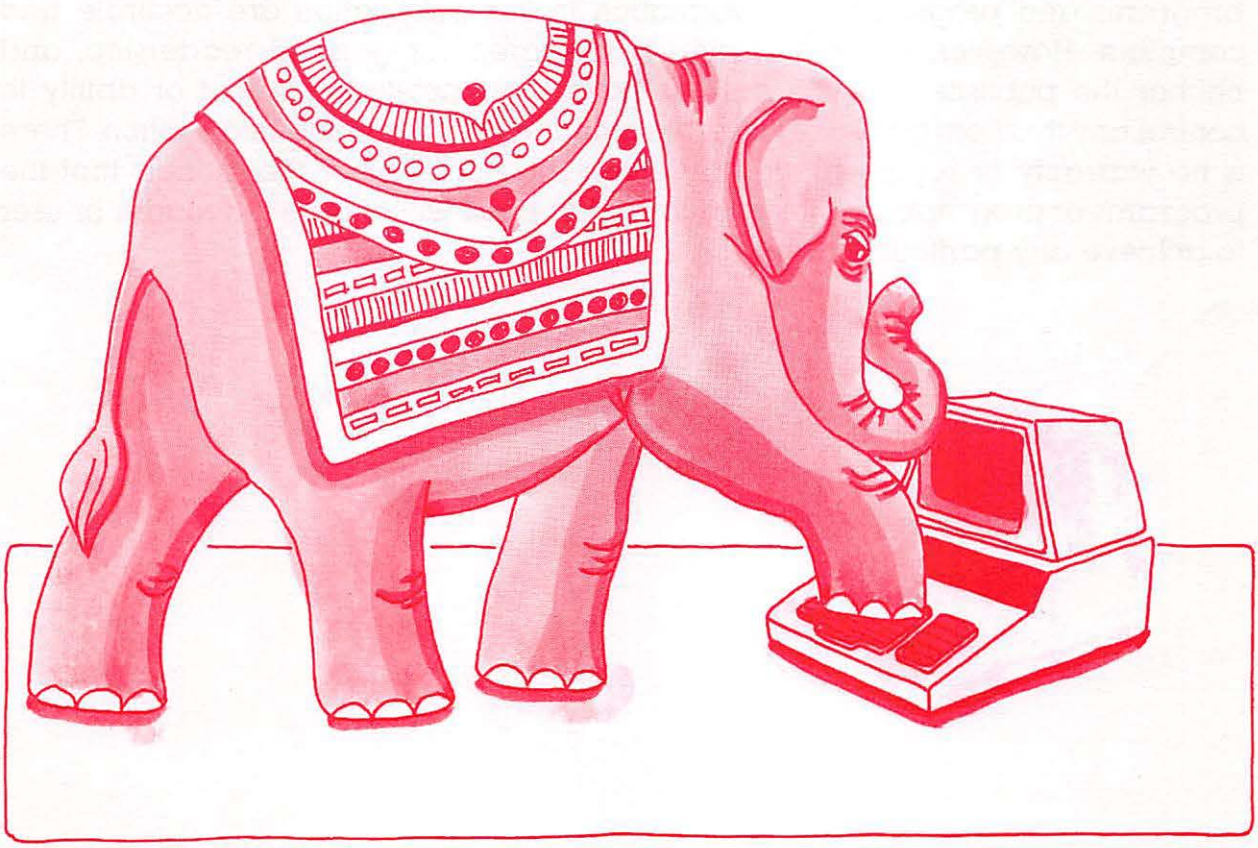
Includes index.
Summary: Introduces the essentials of Basic computer programming on the Apple computer.
1. Apple computer—Programming—Juvenile literature. 2. Basic (Computer program language) [1. Apple computer—Programming. 2. Basic (Computer program language) 3. Computers. 4. Programming (Computers)] I. Title.
QA76.8.A66B67 1984 001.64'2 83-18908
ISBN 0-88056-119-X (pbk.)

Cover and art by Marty Urman

Printed in the United States of America

dilithium Press
8285 S.W. Nimbus
Suite 151
Beaverton, Oregon 97005

TABLE OF CONTENTS



ACKNOWLEDGEMENTS

My thanks go to many people who have offered me support and inspiration in writing this book:

To Alan, my husband and "business manager," who gave me the courage and confidence to become an author.

To Jack Turner, for his expert advice.

To all of the young "computer wizards" and brave teachers for whom *An Apple in the Classroom* and *An Apple for Kids* were written.

An Important Note

The publisher and the authors have made every effort to ensure that the computer programs and programming information in this publication are accurate and complete. However, this publication is prepared for general readership, and neither the publisher nor the authors have any knowledge about or ability to control any third party's use of the programs and programming information. There is no warranty or representation by either the publisher or the authors that the programs or programming information in this book will enable the reader or user to achieve any particular result.

INTRODUCTION

An Apple for Kids is part of a three-book set designed to teach children and beginning programmers how to program a microcomputer in the BASIC computer language. Although this book is geared specifically for the Apple microcomputer, it can be easily adapted for use with other microcomputers as well.

Written at approximately a fourth grade reading level, *An Apple for Kids* consists of eight components of approximately six chapters each. You become familiar with the keyboard and Apple operation in the first components, and learn how to write your own BASIC programs as you progress through the books. By the time you have completed the last component you will have the skills needed to write game programs, simple graphics, teaching programs, and programs that solve problems. All programming techniques introduced can be easily understood by the average sixth grade student.

How to use this book: Read through the chapters and try the examples on your computer. At the end of some chapters there are notes on worksheets "**to do**." (For example, you'll see "**to do:** Programmer's Pastime #11.") Sometimes these activity worksheets are included at the end of chapters so you may try your hand at writing your own programs. These programming worksheets were taken from the second book in the set *An Apple in the Classroom: Activity Workbook*. Solutions to the activities can be found in the *Teacher's Guide* (the third book in the set) which also contains detailed lesson plans for each chapter and additional information and ideas for using this material as a computer programming curriculum. *An Apple for Kids* is the student text in this set. Both the *Activity Workbook* and the *Teacher's Guide* can be ordered from the card at the back of the book.

COMPONENT 1

CHAPTER 1 There's an Apple In Our Classroom!

It may not be the usual kind of apple that sits on the teacher's desk, but it's an apple nevertheless. It's not a type of fruit, and it wouldn't taste very good if you tried to eat it! What kind of an apple is it? Our apple is a **microcomputer** made by Apple Computer, Inc. We will begin to see more Apple microcomputers in the classroom in the future!

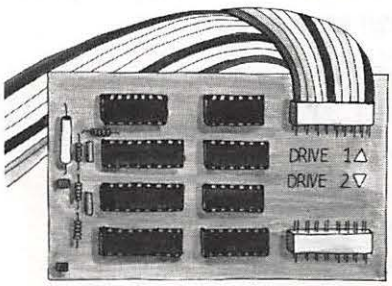
What is a microcomputer? Simply stated, it's a small, portable computer that anyone can learn to use. Microcomputers can teach you lessons in school, help you with hard assignments, or even be your partner in playing a game. What's even more important is that you can learn to control a microcomputer and make it do whatever you want! Microcomputers are a fun and valuable tool to have in a classroom.

Our Apple microcomputer has four basic parts:

1. **keyboard** (punch keys with letters, numbers, and signs)
2. **T.V. screen** (We will also call it a **monitor**.)
3. **disk drive**
4. **brain** (The Apple's insides, including its **memory**.)

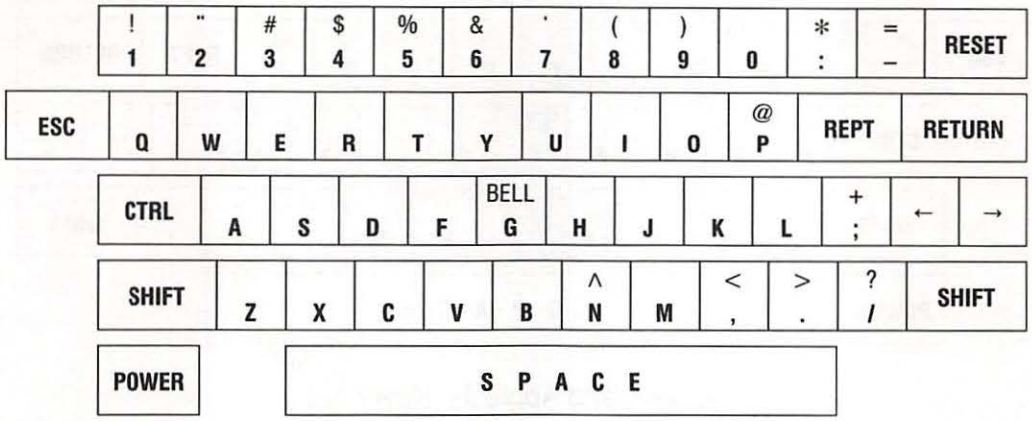
Let's learn about the parts of the Apple so we can use it in our work and play!

Inside the keyboard is the brain.

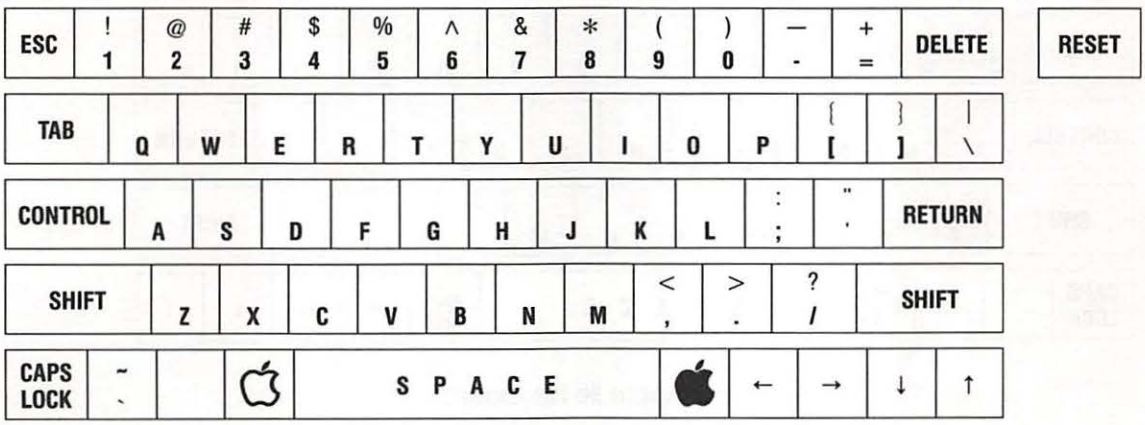


This is what part of the Apple's brain looks like—a flat metal board with many electrical circuits and little bug-like things called **chips**. Some chips are used for memory so the Apple can remember what you tell it.

CHAPTER 2 The Apple's Keyboard



Apple II and Apple II+ Keyboard

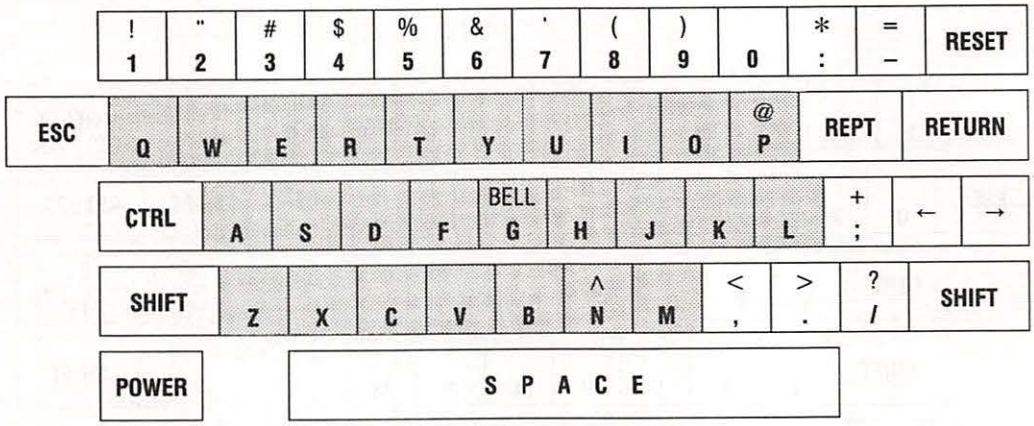


Apple IIe Keyboard

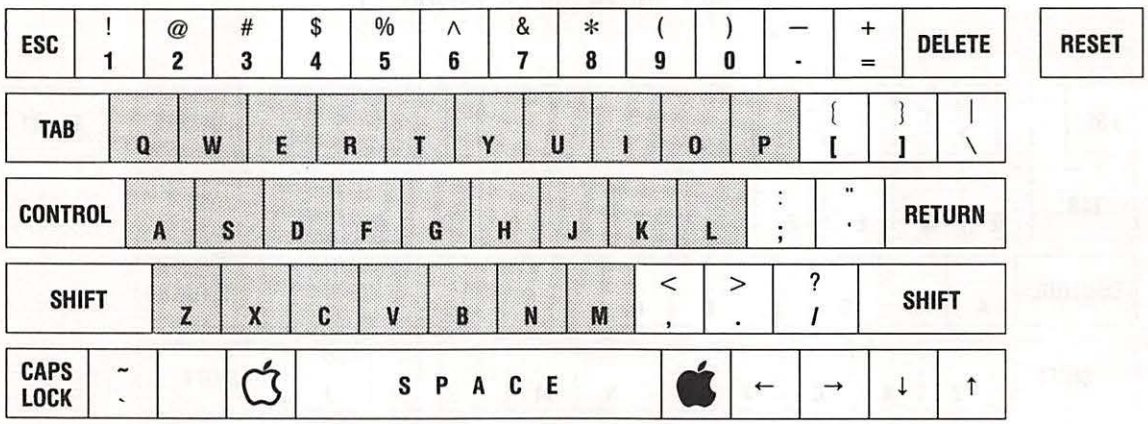
The Apple has a keyboard very much like a typewriter. You can punch:

1. letters
2. numbers
3. function keys (SHIFT , RETURN , CTRL , ESC , and more)
4. special symbols keys (+ , − , * , $, = , ! , and more)
5. edit keys (← , → , ↓ , and more)

LETTERS



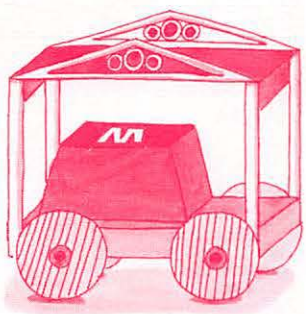
Apple][and Apple][+ Keyboard



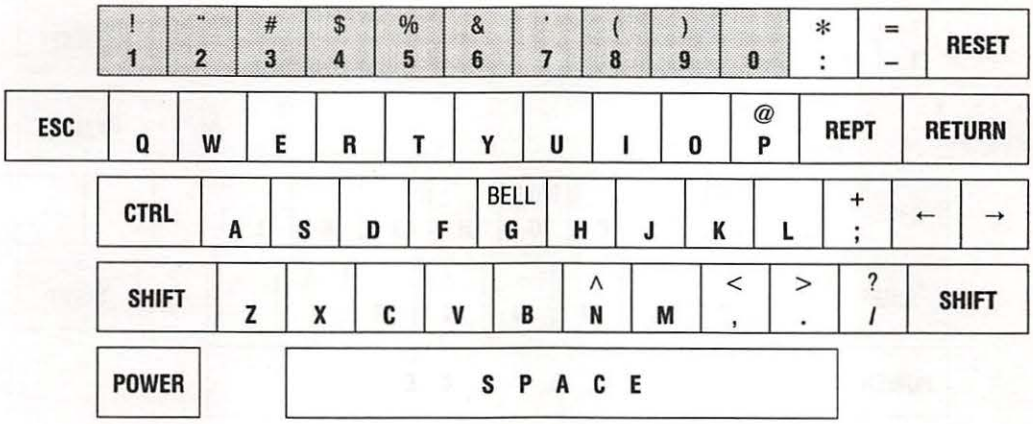
Apple IIe Keyboard

Letter keys on the Apple's keyboard are in the same places as letter keys on a typewriter. To type a letter, press the key.

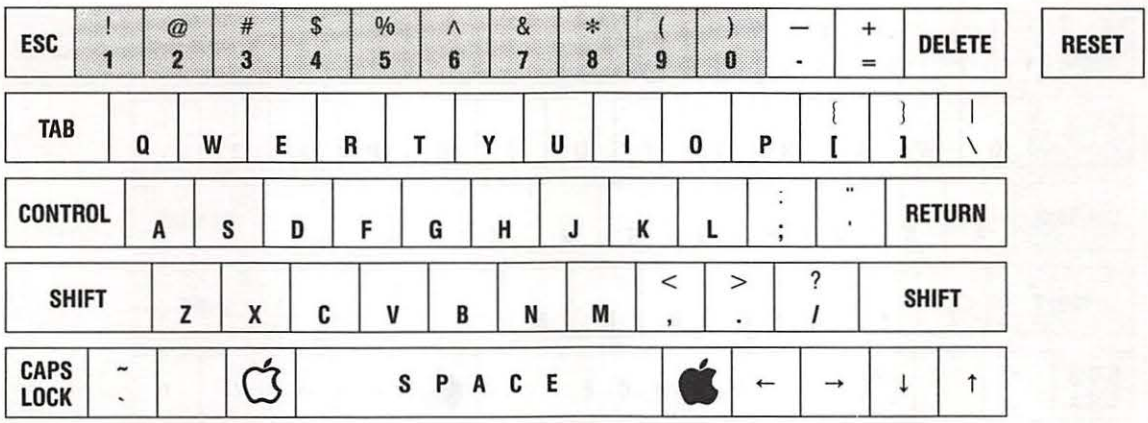
The Apple IIe keyboard has a [CAPS LOCK] key. When this key is depressed capital letters are printed. When this key is in its up position, lower case letters are printed.



NUMBERS

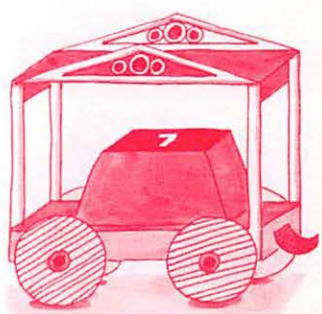


Apple II and Apple II+ Keyboard

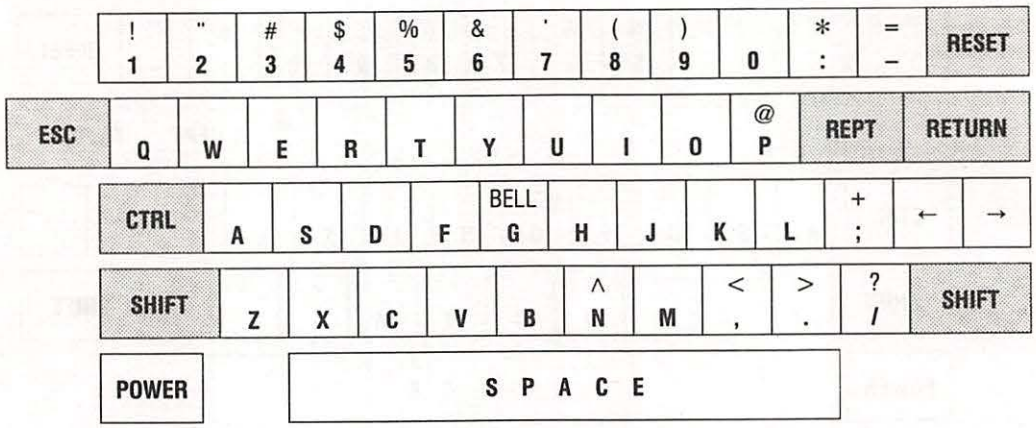


Apple IIe Keyboard

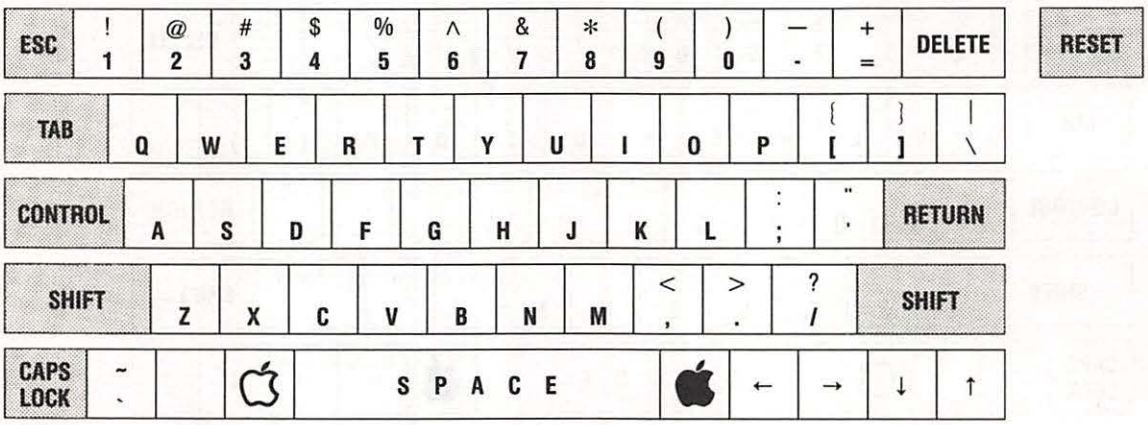
Number keys are also found in the same places as number keys on a typewriter. To type a number, press the key. When zero is typed, it is printed like this: Ø. The computer does this so it won't get zero (Ø) mixed up with the capital O.



FUNCTION KEYS

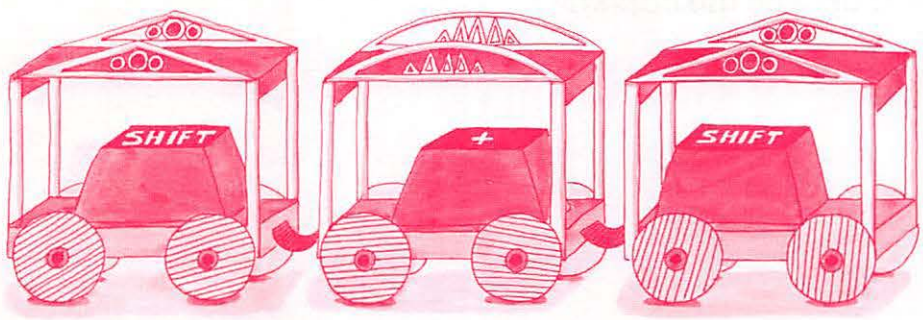


Apple II and Apple II+ Keyboard

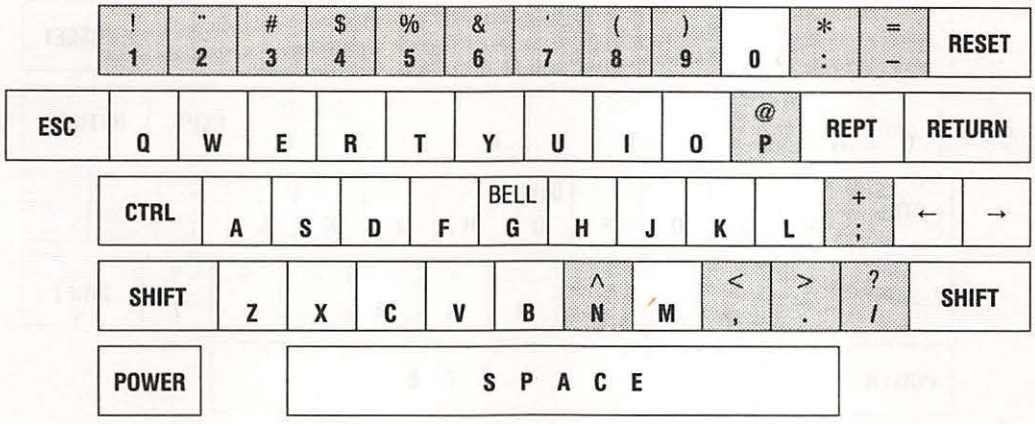


Apple IIe Keyboard

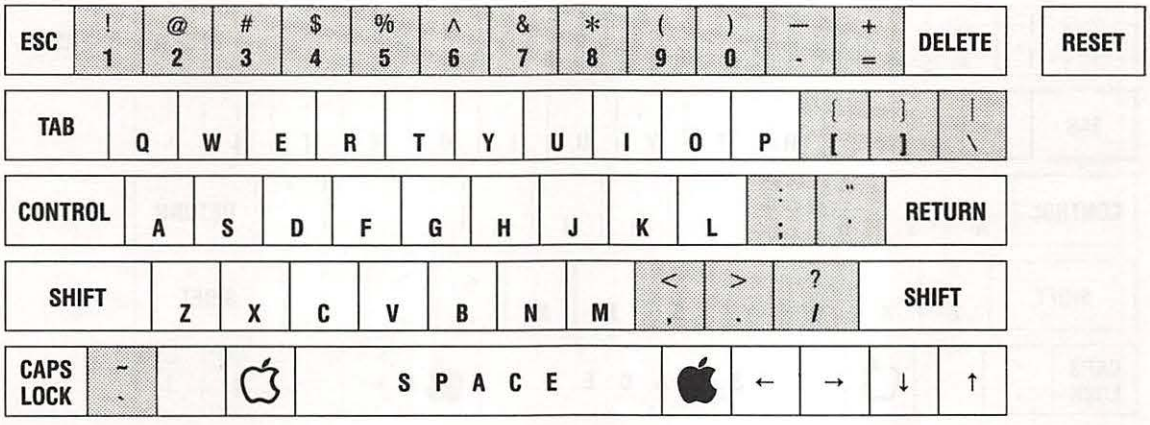
Each function key does a special job. They are very important keys. You will learn more about these keys later.



SPECIAL SYMBOL KEYS



Apple II and Apple II+ Keyboard

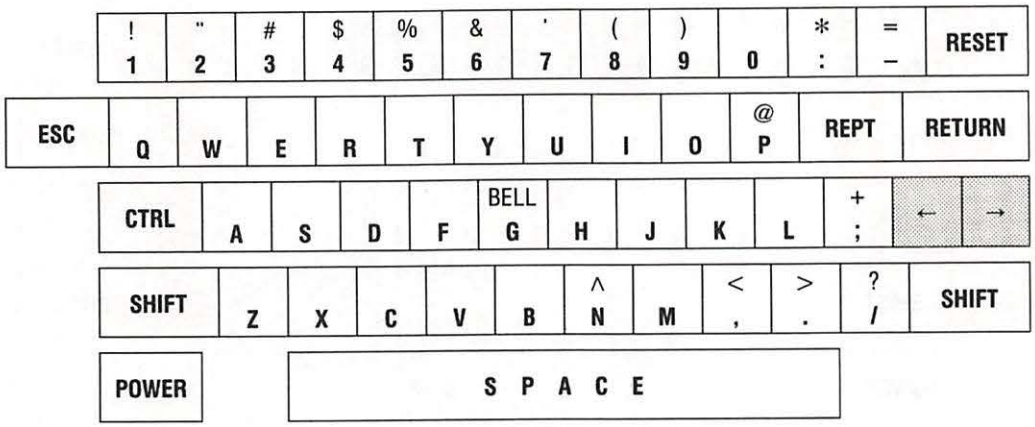


Apple IIe Keyboard

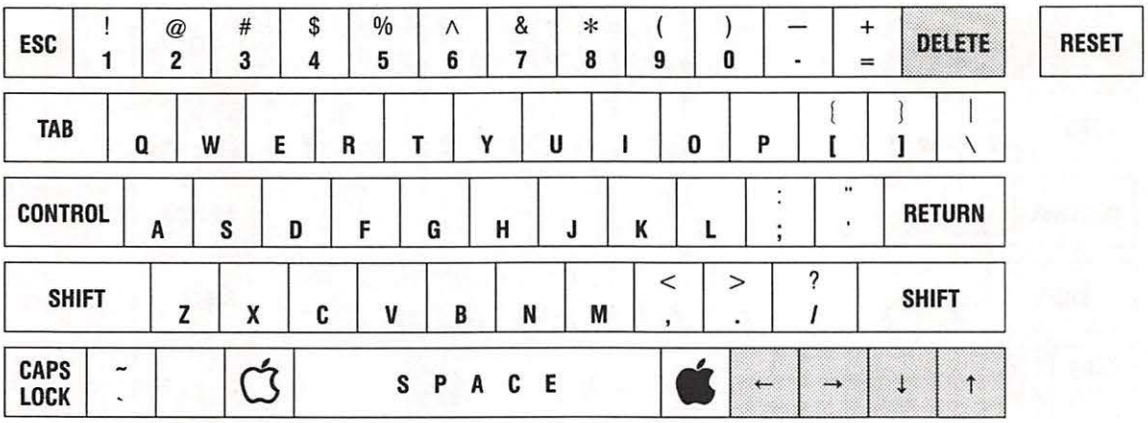
The Apple has many special symbol keys. They are used for doing math and punctuating sentences that you write. Some special symbol keys are used as shortcuts in operating the Apple. To type a special symbol, press the key and you will get the symbol at the bottom of the key.

Notice that some special symbols are found at the top of certain keys. To type these symbols, you will need to press [SHIFT] and hold it down while you press the key with the special symbol you want. The [SHIFT] key tells the Apple to print what is at the top of a key.

EDIT KEYS

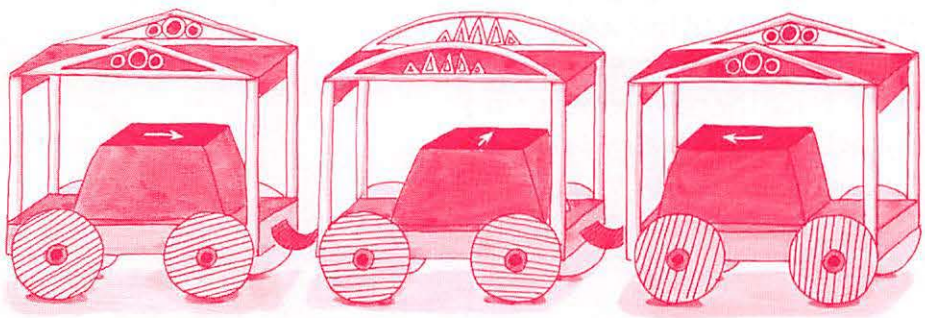


Apple II and Apple II+ Keyboard



Apple IIe Keyboard

The edit keys help us fix mistakes that are typed on the screen. You will learn how to use these keys in Chapter 5.



Before you can work and play with the Apple, you must learn how to get the Apple started. You must also learn how to use the many different keys on the Apple's keyboard.

Let's review how to type:

a number

To type 4, press [$ 4] .

a special symbol

To type $, press [SHIFT] and hold it down. Then press [$ 4] .

to do: Exploring the Apple's Keyboard #1

CHAPTER 3 Turning on the Apple

Follow these directions to turn on the Apple and get it ready to work with you.

1. Turn the volume all the way down on the monitor.
2. Turn on the monitor.
3. Flip the On-Off switch on the back of the Apple. (It's at the lower left.) You will hear a beep as the Apple is turned on. This is how it says hello and tells you it is ready. As the Apple is warming up, you may see some of the characters that the Apple can type flash across the screen. The power lamp on the left side of the Apple's keyboard should now be on.
4. The Apple's screen should say:

```
Apple ][
```

   This means the Apple you are using is called an Apple II (Apple two).
5. Press [CTRL] and hold it down as you press [RESET]. The Apple will beep again and the screen will show:

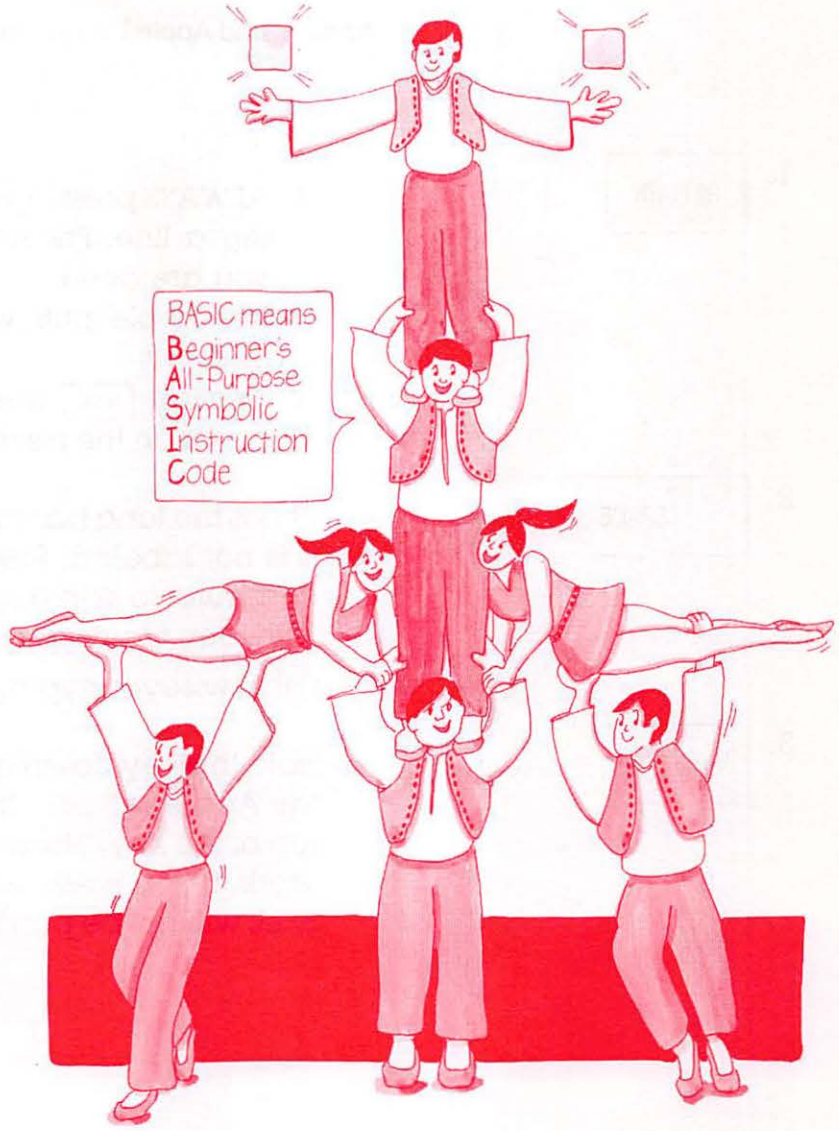
```
]
```

   The first symbol ( ]) is called the **prompt**. The prompt tells you which computer language the Apple understands. Computers can't understand English, so you will have to learn computer languages to communicate with computers.

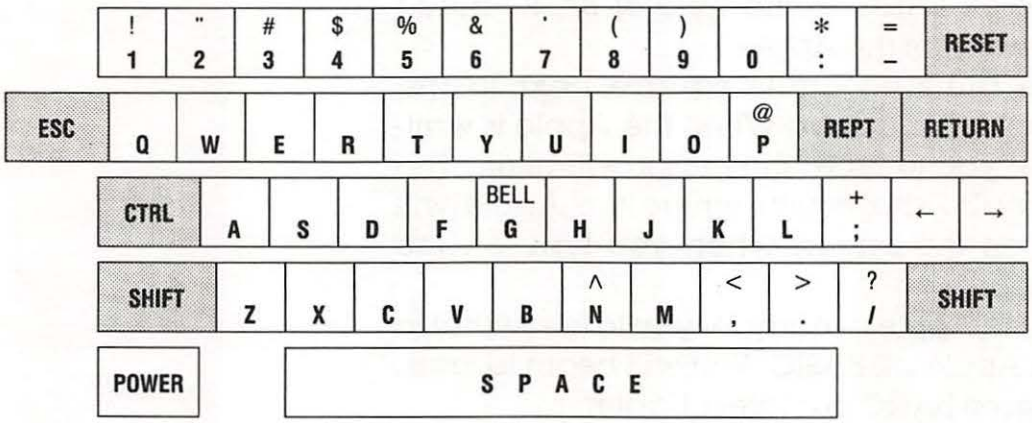
The (]) prompt means that the language the Apple II will understand now is Applesoft BASIC. All microcomputers understand BASIC. Applesoft BASIC is one type of BASIC made especially for the Apple.

The blinking white square next to the prompt is the **cursor**. When the Apple is waiting for you to type, the cursor will blink. The cursor also shows you where the Apple will print on the screen when you type on the keyboard.

6. Now the Apple is ready for you to tell it what to do in Applesoft BASIC. You will begin to learn Applesoft BASIC in a later chapter.



CHAPTER 4 Using the Apple's Special Function Keys



Apple II and Apple II+ Keyboard

1. RETURN

- ★ ALWAYS press [RETURN] when you have finished typing a line. Pressing [RETURN] tells the Apple that you are done.
- ★ The Apple puts what you have typed into its memory.
- ★ Pressing [RETURN] also tells the Apple to put the cursor on the next line of the screen.

2. SPACE

This is the long bar at the bottom of the keyboard. It is not labeled. Pressing the space bar tells the computer to skip a space. You must press this bar between words or numbers that you type.

Otherwiseyourtypingwilllooklikethis!

3. SHIFT

Hold this key down as you press another key and the Apple will print the symbol that you see at the top of the key. There is one case where this won't work: If you press [SHIFT] and the [BELL G] key, the word BELL will not be printed on the screen.

4. CTRL

CTRL means control. This key is always used together with another key—just like the SHIFT key is always used with another key. Hold down CTRL while you press another key, and something special will happen. You will learn how to use CTRL with certain keys later.

5. ESC

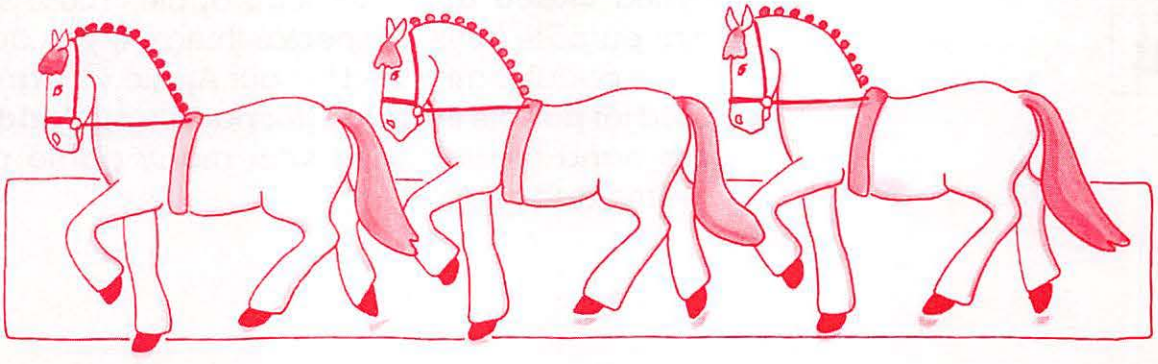
ESC stands for escape. The very first computers that were built used this key to "escape" or break out of a program that the computer was doing. This key would make the computer stop. The Apple's ESC key does not do this, but it can do many other things which you will learn about later. Unlike SHIFT and CTRL, the ESC key is never held down while pressing another key. Always press ESC and let go before you press another key.

6. REPT

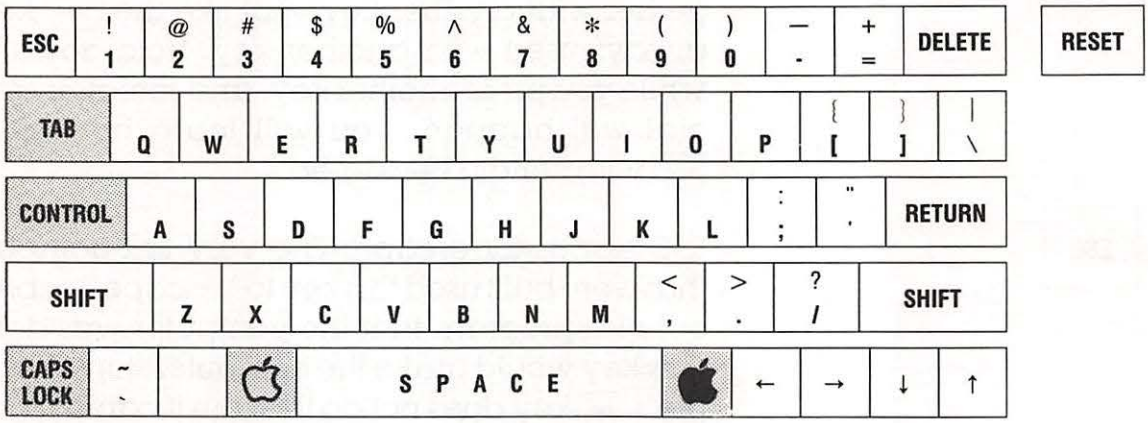
REPT means repeat. Hold the REPT key down while you hold down another key. The Apple will repeat the symbol on the other key by printing it over and over. To stop the repeated printing, let go of one or both keys. For example, if you want the Apple to quickly print a line of Z's, hold down both the REPT and Z keys and watch it go!

7. RESET

RESET is a very important key. Whatever the Apple may be doing, if you press CTRL and RESET, everything will stop. When the computer is doing a program, *it* has control. You have to wait for it to finish. By pressing CTRL and RESET, the program will stop and *you* will again have control over the computer.



For the Apple IIe



Apple IIe Keyboard

The Apple IIe has some extra function keys that the Apple II computers don't have.

The TAB key only works if the program you are running lets you use it. This key, when pressed, will move the cursor eight spaces to the right.

CONTROL is the same as CTRL on the Apple II.

The CAPS LOCK key was explained in Chapter 2. When this key is up, letters typed will be printed in lower case. The SHIFT key must be pressed to print capital letters. When CAPS LOCK is depressed, only capital letters will be printed. It is a good idea to always keep this key in the down position. Many programs only recognize capital letters.

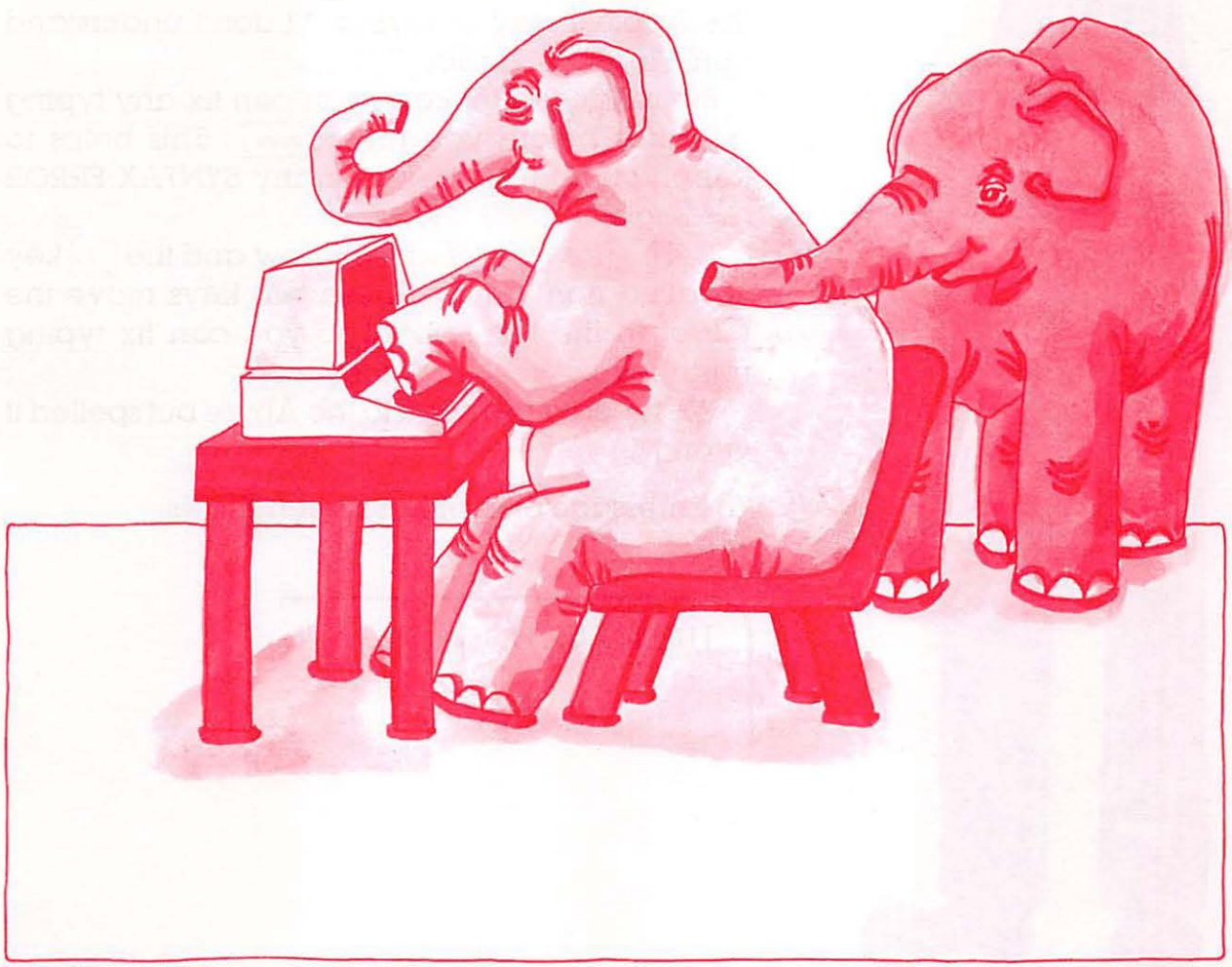
The key is called **open apple**. Likewise, is called **closed apple** or solid apple. These special purpose keys do special things. If you don't have paddles attached to your Apple, can be used for paddle #0 and for paddle #1. **Paddles** are hand controls used with many game programs. (see chapter 8)

By pressing [ ] [CONTROL] [RESET] (all together) the Apple will be restarted when the power is already on. This is called a **system reset** or **warm boot**. If you can't get the Apple to stop what it is doing by pressing [CONTROL] [RESET], try these three keys.

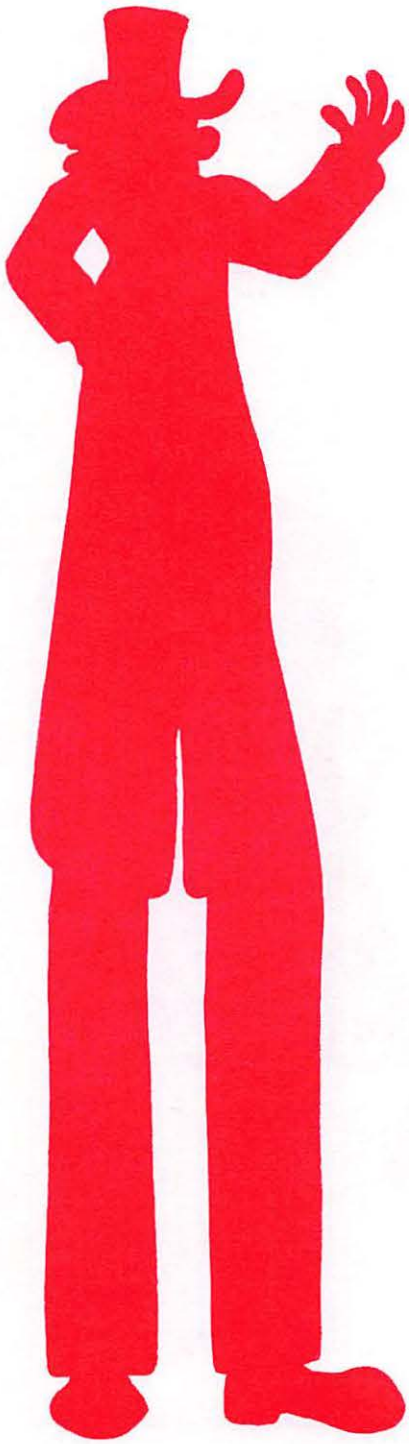
Note

The Apple IIe does not have a [REPT] key. Instead, every key automatically does repeated printing if you hold it down for more than one second.

to do: Exploring the Apple's Keyboard #2



CHAPTER 5 Fixing Typing Mistakes



If you type something wrong, the Apple won't understand you. That's why it's important to correct your typing mistakes.

The Apple lets you know when it doesn't understand you. If you spell a word wrong or forget to speak in BASIC, the Apple will beep and the screen will say:

```
? SYNTAX ERROR
```

SYNTAX ERROR is an **error message**. There are many types of error messages. SYNTAX ERROR is the Apple's way of saying, "I don't understand you. Please try again."

By using the edit keys, you can fix any typing mistakes before you press [RETURN]. This helps to keep you from getting so many SYNTAX ERROR messages.

The [←] key is called left-arrow and the [→] key is called right-arrow. These two keys move the cursor to the left or right so you can fix typing mistakes. Let's see how they work.

We typed a message to the Apple but spelled it wrong.

The message on Apple's screen shows:

```
] HELLO APZLE □
```

The message in Apple's memory says:

HELLO APZLE

We need to change the Z in APZLE to a P. We would press [←] three times to make the cursor move backward three spaces to the Z.

Screen

```
]HELLO AP[Z]LE
```

When we backed the cursor over the LE to get to the Z, the letters L and E did *not* get erased from the screen. We still see them printed on the screen.

BUT the letters L and E *did* get erased in the Apple's memory. If we were to look inside the memory, we would see:

Memory

HELLO APZ

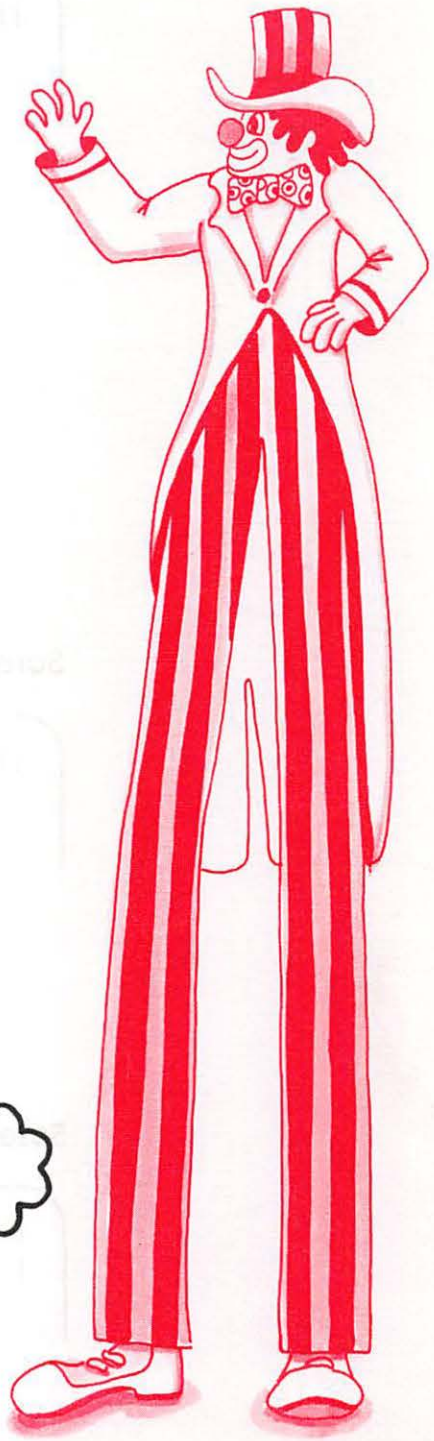
To correct the mistake, type a P over the Z. Now the screen and memory would show:

Screen

```
]HELLO APP[L]E
```

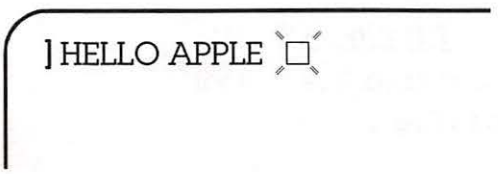
Memory

HELLO APP



To put the LE back into the Apple's memory we use the [→] key. This key moves the cursor to the right. If we press [→] two times, the cursor will type over the LE and put it back into the Apple's memory.

Screen



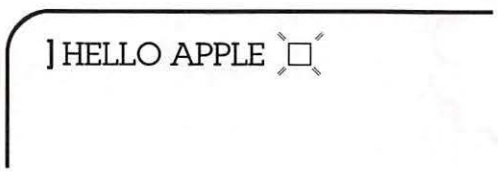
Memory

Now our mistake has been corrected both on the screen *and* inside the memory. Always remember to correct your mistakes this way.

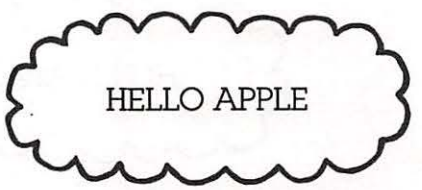
There are more tricks in correcting typing mistakes. If you use [REPT] with [←], you will be able to backspace faster. If you use [REPT] with [→], the cursor will type over what is printed more quickly.

Use [CTRL] [X] to erase a whole line from Apple's memory. Let's say we typed HELLO APPLE, but decided to change it to HOWDY PARTNER instead. The screen and memory would show:

Screen

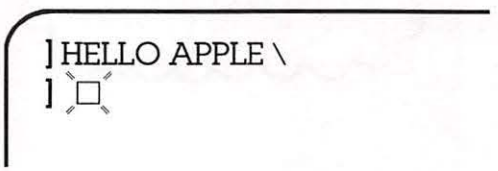


Memory

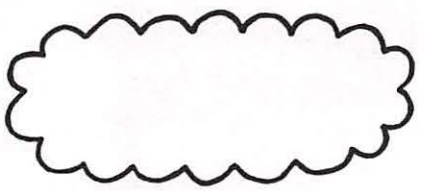


Press [CTRL] [X] and the cursor will go to the beginning of the next line. We will still see our message on the screen, but it will be erased from the memory.

Screen



Memory



The Apple printed a \ after HELLO APPLE. This \ means that we want to forget about this message and type something else on the next line.

Now type the right message:

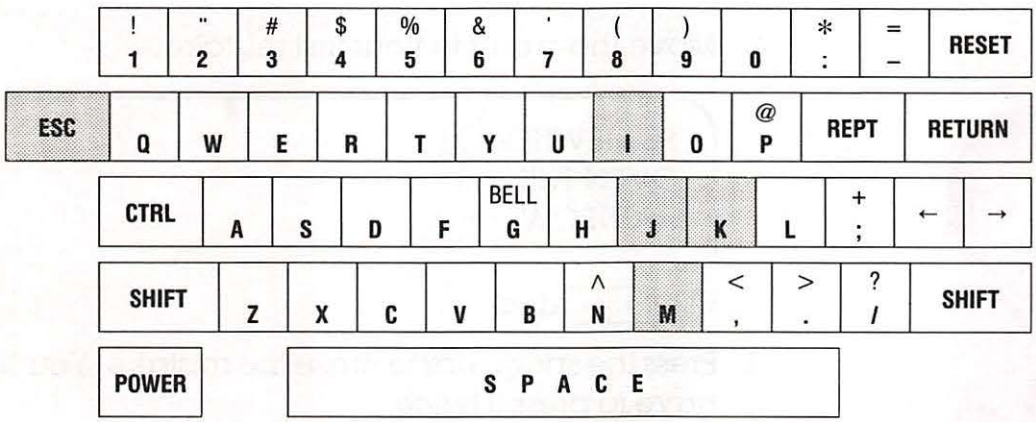
Screen

```
]HELLO APPLE \
]HOWDY PARTNER
```

Memory

By using [ESC] with four keys, you can move the cursor anywhere on the screen without erasing any writing from the screen or from the memory. We call these keys CURSOR CONTROL KEYS.

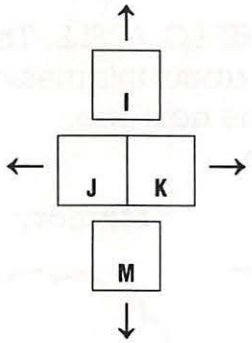
CURSOR CONTROL KEYS



Apple II and Apple II+ Keyboard

The four keys you will use with [ESC] are [I], [J], [K], [M]. Notice how these keys are arranged on the keyboard. They form a type of directional keypad.

Since I is on top, [ESC] [I] moves the cursor up.
Since M is on the bottom, [ESC] [M] moves the cursor down.
Since J is on the left, [ESC] [J] moves the cursor to the left.
Since K is on the right, [ESC] [K] moves the cursor to the right.



Always remember to press [ESC] and let it go before you press the next key.

Here are some ways you can use the cursor control keys to correct a typing mistake:

1. Find your mistakes on the screen.

```
SOMEWHEREX
OVEN THE
RAINBOW □
```

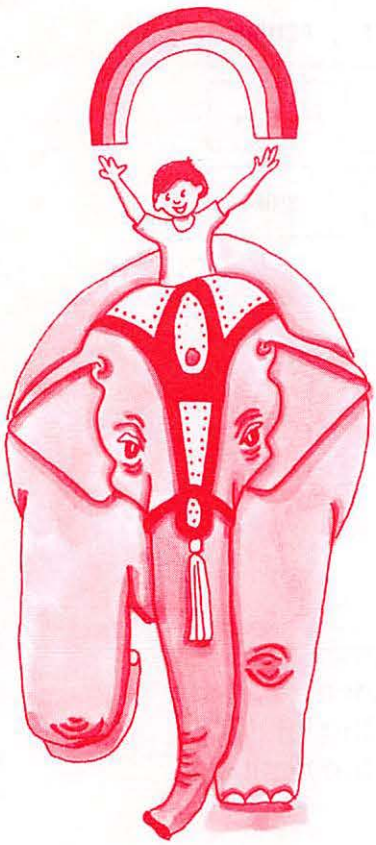
2. Move the cursor to your first mistake.

```
SOMEWHERE[X]
OVEN THE
RAINBOW→→
```

Use [ESC] [K] and [I] .

3. Press the space bar to erase the mistake. You'll have to press it twice.

```
SOMEWHERE □
OVEN THE
RAINBOW
```



4. Press [ESC] and move the cursor to your second mistake.

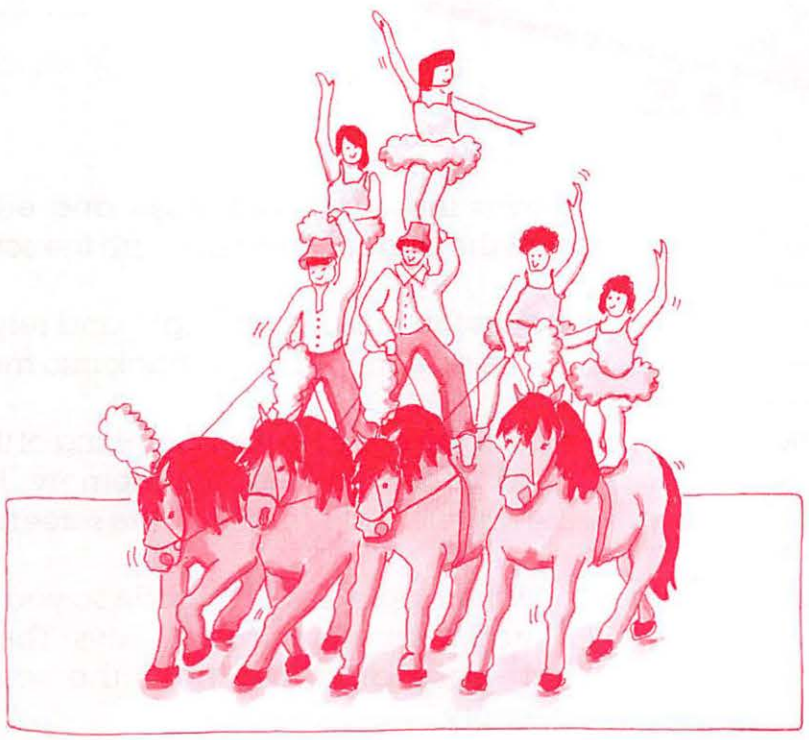
SOMEWHERE
OVE[N] ← THE ↵
RAINBOW
Use [ESC] [M] and [J] .

5. Type over your mistake twice.

SOMEWHERE
OVER □ THE
RAINBOW

6. Press [ESC] and move the cursor back to where you will type next.

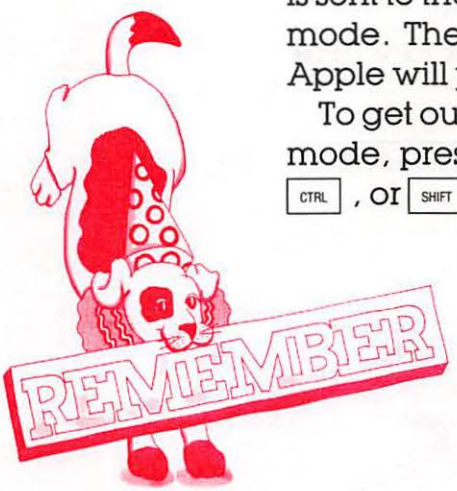
SOMEWHERE
OVER ↓ THE
RAINBOW
□ ← ↓



When you turn on the Apple, it is in **direct** or **immediate mode**. A mode is a way of acting—somewhat like a mood. (For example, if you are in a tired mode, you may yawn a lot.) When the Apple is in direct or immediate mode, you have direct control over it. It will do what you tell it to do immediately.

When you press [ESC], the Apple goes into **edit mode**. Edit means to correct mistakes. When the Apple is in edit mode, it is ready to move the cursor around the screen with I, J, K, and M so you can correct mistakes. You must press a key twice to type over a mistake. The first time you press the key, nothing happens on the screen. A message is sent to the Apple that tells it you want out of edit mode. The second time the key is pressed, the Apple will print on the screen.

To get out of the edit mode and back into direct mode, press any key except [ESC], I, J, K, M, [REPT], [CTRL], or [SHIFT].



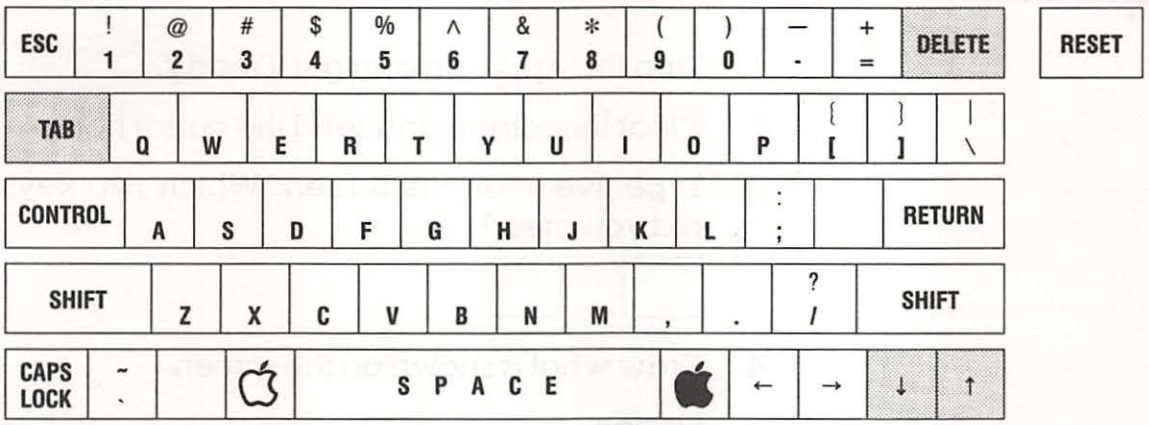
[←] moves the cursor to the left and erases writing from the memory, but not from the screen.

[→] moves the cursor to the right and retypes what is on the screen and puts it back into memory.

[CTRL] [X] moves the cursor to the beginning of the next line. It erases the old line from memory, but not from the screen. It leaves a \ on the screen.

[ESC] puts the Apple into edit mode so you can use I, J, K, and M as cursor control keys. These keys will not erase anything from the screen or the memory.

For the Apple IIe



Apple IIe Keyboard

The [↓] key is called the **down arrow**. It moves the cursor down one line on the screen without erasing any typing. The [↑] key is the **up arrow**. It moves the cursor up one screen line, again without erasing any typing. The [↑] key only works when you are running certain programs.

The [↓] key can be used in place of [ESC] and [M] .

The [DELETE] key deletes unwanted characters but only works in certain programs. If you press [DELETE] while typing on the screen, a checkerboard cursor pattern will be printed.

The [TAB] key is also called a cursor control key because it moves the cursor 8 spaces to the right when pressed.

You may use [↓] when you are typing on the screen in direct mode. This key may not work if you are running a program.

[DELETE] , [TAB] , and [↑] will only work if a program you are running allows them to work. They do not work in direct mode.

to do: Exploring the Apple's Keyboard #3, #4, #5
Screen Game #1, #2, #3, #4

Screen Game #1 Computer Safari

1. Turn the Apple on and get it ready.
2. Clear the screen and send the cursor home.
3. Type five * on the screen. Which two keys did you press?

   ☐ ☐

4. Draw what is shown on the screen.

   Screen

   Pretend this is the wild forest.

5. Type: LION Type: % (Can you figure out how?)
   What did you press to type the % ?

   ☐ ☐

6. Draw what is shown on the screen.

   Screen

   Pretend the % are the lion's eyes watching you.

7. Type 12 more *.

8. Press [ESC] . Press [M] once. Press [J] 22 times. The cursor should go to the beginning of the next line.

9. Type 10 *. Type: ELEPHANT Type: % Type three more *.

10. Go to the beginning of the next line by using [ESC] and [M] and [J] . (Follow Step 8 to help you remember.)

11. Type three *. Type: TIGER Type: % Type 13 more *.

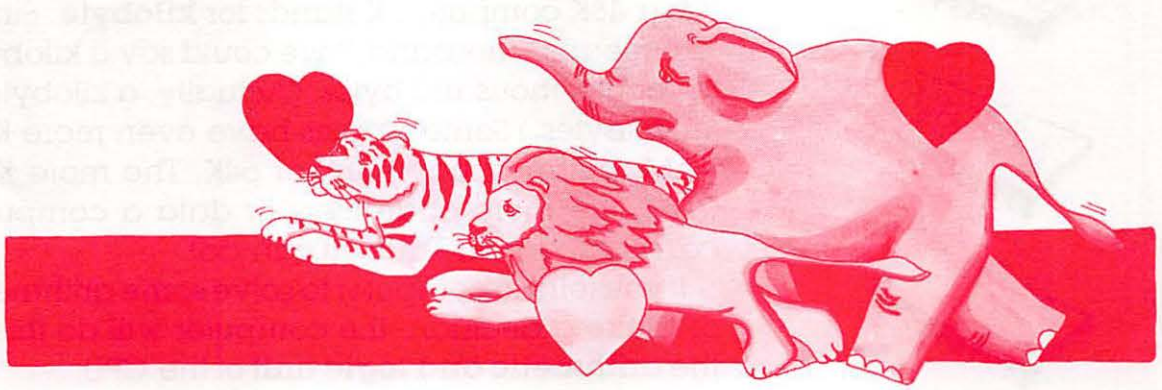
12. Go to the beginning of the next line.

13. THE HUNT BEGINS!!! Without erasing anything, move the cursor to the eyes (%) of each animal. Then erase the %.

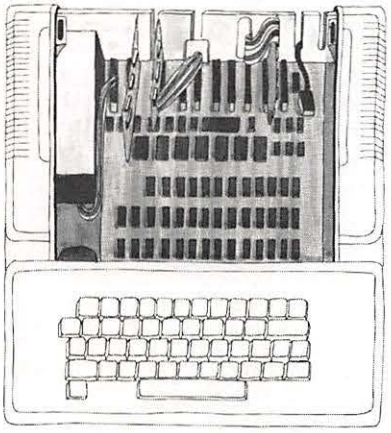
    If you erase anything besides the %, you get eaten and lose!

14. Which animals did you capture? _____

    _____



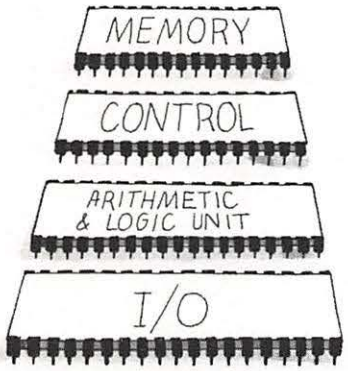
CHAPTER 6 Inside the Apple



Inside the keyboard is the **CPU** (central processing unit) or "main brain" of the Apple. You can look at the brains of your Apple by removing the keyboard lid (only with permission, of course!). You will see many flat metal circuit boards with small black **chips** sitting on them. These chips may look like strange insects with many legs, but they are really for storing information and carrying out tasks that you ask the computer to do.

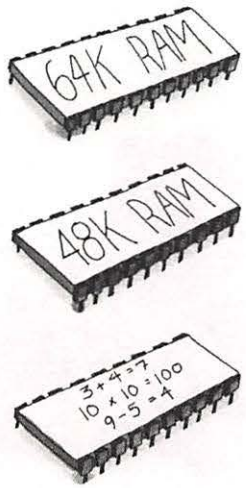
The CPU is made up of four main parts:

1. **Memory**
2. **Control**
3. **Arithmetic and Logic Unit**
4. **I/O (input/output) pathways**



When you type something on the keyboard, the characters you type are stored in the keyboard memory. If you press RETURN, those characters will be taken out of keyboard memory and put into one of the Apple's main memory chips called **RAM**. RAM stands for **Random Access Memory**.

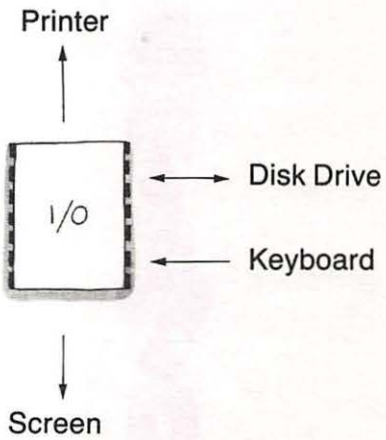
Memory is measured by **bytes**. A byte is the space it takes to store one **character**. A character can be a letter, number, special symbol, or even a blank space. Your Apple's RAM may be able to store from 49,952 to 65,536 or more characters. This is about as many letters as you would find in a 50-page book.



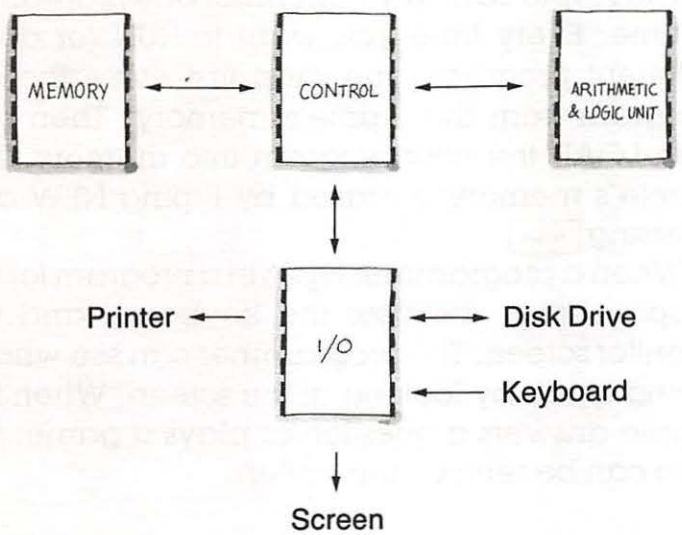
If a computer's RAM holds 49,952 bytes, we say it is a 48K computer. K stands for **kilobyte**. Since kilo means "thousand," we could say a kilobyte is about a thousand bytes. (Actually, a kilobyte is 1,024 bytes.) Some Apples have even more K of RAM, but most have 48K or 64K. The more K of RAM, the more characters or data a computer can hold, and the more it can do!

If you tell the computer to solve some arithmetic or make a decision, the computer will do this in the **arithmetic and logic unit** of the CPU.

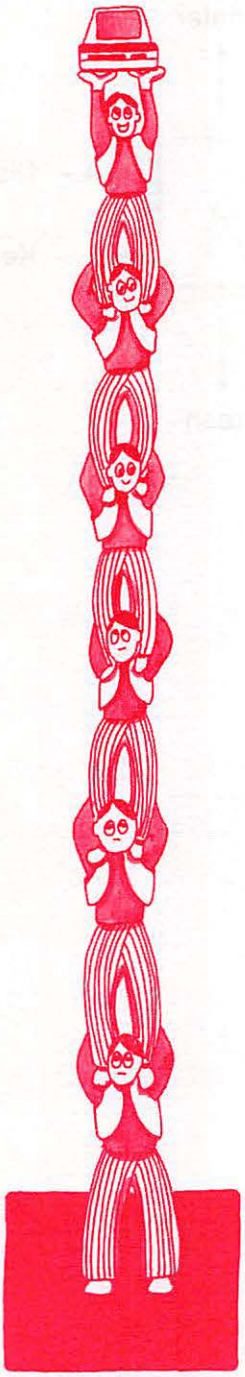
Input is any information that you put into the computer. When you type on the keyboard, you are putting input into the computer. When you run a program from a disk, the program becomes input because it is put into the computer from the disk drive. **Output** is information that the computer gives you, or puts out. You may need to know the answer to a math equation. First you type the equation on the keyboard (give the computer the input). Then the computer gives you the answer, which is the output. You can see the computer's output on the screen, or the computer can print it on a printer. The **I/O pathways** (Input/Output pathways) send messages from the computer to the screen or disk drive or printer, or vice versa. These messages will usually be some form of input or output.



The fourth part of the CPU is the **control**. The control is like a policeman directing traffic. When you type information or **data** on the keyboard, the control sends it to the memory or to the arithmetic and logic unit. After the arithmetic and logic unit has solved a problem, the control sends the answer through the I/O pathways to the screen or some other output device such as a printer or a disk drive.



CHAPTER 7 The Apple's Monitor and Disk Drive



When you learn how to play a game you must follow a set of directions. The same is true for computers. The Apple cannot work or play with you, or even communicate with you, unless it has directions to follow.

We call the set of directions computers use a **program**. Computer programs are written by people. People who write computer programs for a living or as a hobby are called computer **programmers**. As you work through this book, you will learn how to program the Apple, and perhaps some day you'll become a computer programmer!

The Apple learns programs in two ways:

1. You type the program on the keyboard. The Apple copies the program from the keyboard and stores it in memory. In this way, it understands and remembers the program.
2. The Apple can also get the program from a disk on which the program is recorded. It copies the program from the disk and stores it in memory where it will understand and remember the program.

The Apple can only remember one program at a time. Every time you want to RUN (or do) a different program, you must first erase the old program from the Apple's memory. Then you can LOAD the new program into memory. The Apple's memory is erased by typing NEW and pressing RETURN .

When a programmer types in a program for the Apple, he or she uses the keyboard and the monitor screen. The programmer can see what is being typed by looking at the screen. When the Apple answers a question or plays a game, this also can be seen on the screen.

Does the Apple's monitor remind you of your T.V. at home? Both screens are made with a **cathode ray tube**. This tube lets you see numbers, words, and images on the screen. The Apple's monitor may sometimes be called a **CRT** (cathode ray tube) because of its screen.

Because each microcomputer can hold only one program at a time in its memory, programs must be stored elsewhere. One of the best places to store microcomputer programs is on a **disk** (also called a **diskette**). A disk is small and flat and looks like a floppy record. It is made out of magnetic material and can store many computer programs at once. One disk can store more than 143,000 bytes of information.

In order for the Apple to copy programs from a disk, a disk drive is needed. The disk is slipped into the disk drive where it spins like a record. Inside the disk drive is a **head** (think of it as being like the needle on a record player), which can read and write information to and from the disk. The computer can move the head to any point on the disk to **access** any program, just like you can move the needle on a record player to play any song on the record. A special program called the **disk operating system** (or **DOS**) controls the disk drive.

The type of disk that the Apple uses is a 5¼-inch floppy disk or diskette. (We will usually just call it disk for short.) You never really see the disk because it is enclosed in a protective envelope. You should **NEVER** remove the disk from this envelope! A fingerprint, cigarette smoke, or even a speck of dust can ruin a disk and all the programs that were stored on it. You can see part of the disk through openings in the envelope. Be careful that you never put your fingers near the openings. These openings allow the disk drive head to reach the surface of the disk. You will learn how to use disks and operate the disk drive later.

CHAPTER 8 Apple's Peripherals

Many different types of equipment can be attached to a computer to do many types of jobs. These pieces of equipment for computers are called **peripherals**.

A disk drive is a peripheral because it stores and reads extra programs for the Apple. A **cassette tape recorder** does the same thing. You could attach a special cassette tape recorder to the Apple and read and store extra programs on cassette tapes.

Another important peripheral is a **printer**. Whenever the Apple answers a question or prints something on the screen, it is displaying **output**. Output is information that computers give to people. Screen output is only temporary. It has to be erased from the screen so the computer can show new output or so the programmer can type on the keyboard. To save output on paper, a peripheral such as a printer is needed. The computer prints output on printer paper so the programmer can keep it forever. Output printed on paper is called a **hard copy** because it can be kept forever.

A **graphics tablet** is a fun peripheral to have. A **graphic** is a picture or design that you can draw. The graphics tablet lets you create pictures and designs on the Apple in color (if you have a color monitor). In this way you can create beautiful computer art.

If you play games on your Apple, then you might have game control peripherals called **paddles** or **joysticks**. These peripherals help you move game pieces around the screen and shoot at objects in the program. Game controls may also be used in programs that create computer art.

There are many other types of peripherals to use with the Apple. Some are music synthesizers that help you write songs. Other peripherals can help control your living environment. They can be hooked up to a heating source in your house, such as your furnace, and control the inside temperature. Some peripherals—called **modems**—are hooked into the telephone lines and can "call up" other computers. In the future, you will see many other peripherals that allow computers to do more and more incredible things.

to do: Component 1 Fun Page

COMPONENT 2

CHAPTER 9 Play a Game with the Apple

Playing a game with the Apple can be both fun and challenging. Most game programs are stored on disks. The first thing you will do is **LOAD** the program into the Apple's RAM. Depending on the program and the disk, there are two ways to do this.

First Way

If the disk has only one program on it, follow these directions:

1. Open the disk drive door.
2. Carefully insert the disk. The label should be facing up and you should have your thumb on the label. Make sure you put the disk in very straight.
3. Close the door to the disk drive.
4. Turn down the volume on the monitor.
5. Turn on the monitor.
6. Turn on the Apple.
7. The disk drive will make whirring and clicking noises as the program is loaded.
8. After the program has been loaded, the game should begin.

Second Way

If the disk has many programs stored on it, follow these directions:

1. Open the disk drive door.
2. Carefully insert the disk with your thumb on the label. Make sure the label is facing up and that you put the disk in very straight.
3. Close the door to the disk drive.
4. Turn the volume down on the monitor.
5. Turn on the monitor.

6. Turn on the Apple.
7. The disk drive will make a whirring and clicking noise as the disk is getting the computer ready.
8. When it is ready, the screen should say:

```
DOS VERSION 3.3  08/25/80
APPLE II PLUS OR ROMCARD  SYSTEM MASTER

(LOADING INTEGER INTO LANGUAGE CARD)
]
```

9. Type: CATALOG [RETURN]
   The disk drive will again whir and click. The screen will show a **catalog** (listing) of the programs and information that are stored on the disk. If there isn't a space between the cursor and the last program listed, press any key and the rest of the programs in the catalog will be displayed on the screen.
10. Look for a program called MENU. If the catalog does not list MENU, then read through instruction number 12 and skip to number 15.
11. The screen might show something like this:

```
DISK VOLUME 254
* A006 HELLO
* B 050 INTBASIC
* I  010 MENU
* I  019 HAMMURABI
* A047 LEMONADE
]
```

12. You will see asterisks (*) and the letters A, B, or I, as well as numbers next to the name of each program.

    * means that the program is **locked** on the disk and can't be accidentally erased.

    A means that the program is written in Applesoft BASIC.

    B means that the program is written in **binary** machine language. Computers understand this language better than people because it is made mostly of numbers and symbols.

    I means that the program was written in **integer BASIC**. This was the first BASIC written for the Apple.

    006 The numbers tell how much space each program is taking up on the disk.

13. Type: RUN MENU [RETURN]
14. The disk drive will whir and click and the screen will show:

```
 INTEGER
3 MENU
4 HAMMURABI
 APPLESOFT
1 HELLO
5 LEMONADE
 BINARY
2 INTBASIC
```

The menu organizes the catalog programs into the languages in which they were written. The numbers stand for the order in which the programs are stored on the disk.

At the bottom of the screen are the instructions for using the menu. They say:

```
C—CATALOG ANOTHER DISK   ESC—EXIT

L4—LOAD PROGRAM #4
R4—RUN PROGRAM #4        WHICH?
```

If you don't want any of the programs on this disk, then you would press [C] [RETURN] .

If you want to leave (exit) the menu, press [ESC] and start over again.

If you want to load program number 4, which is HAMMURABI, you would type L4 [RETURN] . HAMMURABI would be loaded into RAM. Then you would have to type: RUN [RETURN] for the program to begin.

If you want to load *and* run program 4 (HAMMURABI), type: R4 [RETURN] . The program will be loaded *and* will run (begin).

Before the Apple can do a program, the program must be loaded into the Apple's RAM. Then the Apple will know the directions for doing the program. When the Apple starts doing the program instructions, we say the Apple is RUNNING the program. You can see that it is easier to run a program, than to load and then run it.

Now you know how to use the menu. Using the menu is only one way of loading and running a program.

15. If your catalog does not list a menu, just run the program you would like the Apple to do. Let's say you would like to run the program called LEMONADE. Type: RUN LEMONADE [RETURN] .

    The program called LEMONADE will be loaded into RAM and will begin.

Third Way

If you plan to write your own programs and then save them on a disk (which you will learn to do later), you must first get the disk drive set up and ready. To set up the disk drive, a special program called the DISK OPERATING SYSTEM (DOS) must be in the Apple's memory. The DOS controls all of the disk drive activities. Putting DOS into memory is called **booting** the disk. This is done to get the Apple and disk drive ready to work together. Some disks with programs already have DOS on them. (They load in DOS with the program.) If you are writing your own programs, you must first boot the disk with the special DOS program that is on the disk called the **system master**. Here is how you boot the disk with DOS:

1. Open the disk drive door.
2. Carefully insert the System Master disk.
3. Close the door to the disk drive.
4. Turn down the volume on the monitor.
5. Turn on the monitor.
6. Turn on the Apple.
7. The disk drive will whir and click and the screen will say:

```
DOS VERSION 3.3   08/25/80
APPLE II PLUS OR ROMCARD   SYSTEM MASTER

(LOADING INTEGER INTO LANGUAGE CARD)
]
```

8. Now the Apple and the disk drive are ready to work together, and you may start typing your programs.

Caution

Once a program has been loaded from the disk, you may remove the disk from the disk drive and put it back into the envelope. The program will stay in RAM until you type NEW or load a new program or turn off the computer. **NEVER** remove the disk if the red light on the disk drive is ON!

1. Insert the disk with the game you want to play or the program you want to run.
2. Turn on the monitor and Apple.
3. If the game/program doesn't start right away, type CATALOG.
4. Type: RUN (name of program).

CHAPTER 10 Teaching the Apple to Do Your Homework

Have you ever dreamed of having a computer that could do your homework for you? It's possible! You can teach the Apple to do your math assignments for you. The Apple can figure out your math and give you the correct answers about a billion times faster than you can!

Here are six kinds of arithmetic that the Apple can do for you:

	Our Symbol	Applesoft BASIC Symbol	Example
1. addition	+	+	2+2=4
2. subtraction	−	−	4−2=2
3. multiplication	×	*	2*3=6
4. division	÷	/	6 / 2=3
5. square root	10^2	∧	10∧2=100
6. powers	$\sqrt{25}$	SQR( )	SQR(25)=5

CHAPTER 11 The Apple as a Calculator

You can use the Apple as a calculator to do the six kinds of arithmetic mentioned in Chapter 10. You will ask the Apple to print the answer to any arithmetic equation you type on the keyboard. The BASIC command you will use is **PRINT**.

1. To add 25+35, type: PRINT 25+35 RETURN . (Always remember to press RETURN after you have typed an equation.)

```
] PRINT 25+35
60
]
```

2. To subtract 60−12, type: PRINT 60−12 RETURN .

```
] PRINT 60-12
48
]
```

3. To multiply 50×5, type: PRINT 50*5 RETURN .

```
] PRINT 50*5
250
]
```

4. To divide 90÷3, type: PRINT 90/3 RETURN .

```
] PRINT 90/3
30
]
```

5. To find the square root of 100, type: PRINT SQR(100) RETURN .

```
] PRINT SQR(100)
10
]
```

6. To find the second power of 4 (4^2), type: PRINT 4∧2 RETURN .

```
] PRINT 4∧2
16
]
```

Notice that you do *not* type an equal sign (=) when you do arithmetic on the Apple.

When you use the Apple as a calculator, it is called direct mode programming because the Apple gives you the answer *directly* after you press RETURN .

to do: Programmer's Pastime #1

CHAPTER 12 Arithmetic with Many Numbers

The Apple can do more than one kind of arithmetic in the same equation. When the Apple deals with many numbers and many types of arithmetic at once, it follows a certain order. Let's look at an equation to see what that order is.

EQUATION: 5*(6−2)+9/3∧2

	what the Apple does
1. PARENTHESES are done first.	6−2=4
2. POWERS are next.	3∧2=9
3. MULTIPLICATION, DIVISION, and SQUARE ROOTS are third. (The numbers to the left are done first, then the numbers to the right.)	5*4=20 9/9=1
4. ADDITION and SUBTRACTION are performed last, together. (The numbers to the left are done first, then the numbers to the right.)	20+1=21

5*(6−2)+9/3∧2 PARENTHESES
(6−2): 1st

5* 4 +9/3∧2 POWERS
3∧2: 2nd

5* 4 +9/9 MULTIPLICATION and DIVISION (left to right)
5*4: 3rd; 9/9: 4th

20 + 1 ADDITION (and SUBTRACTION) (left to right)
20+1: last

answer = 21

Here's a saying that might help you remember the order in which the Apple does arithmetic.

Please	P stands for PARENTHESES
Pay	P stands for POWERS
My **D**ear	M stands for MULTIPLICATION D stands for DIVISION
Square	remember—SQUARE ROOTS are done third too!
Aunt **S**ally	A stands for ADDITION S stands for SUBTRACTION

There is a shortcut for getting the answers to many short equations. If you want answers to:

1. 99*66
2. 74+47
3. 89−78
4. 4^2

Type: PRINT 99*66, 74+47, 89−78, 4^2 RETURN

Use commas to separate each equation.

You can type up to 255 characters in one PRINT statement. (Remember that spaces and commas are characters too!) The Apple will beep when you type the 248th character to warn you that you are approaching the limit. If you type more than 255 characters for one PRINT statement, the Apple will print a \ and cancel the line and you must start over!

to do: Programmer's Pastime #2, #3, #4, #5

PROGRAMMER'S PASTIME #2

In what order does the Apple perform arithmetic in equations?

_____ are done first.

_____ are second.

_____ and _____ are

_____ are done third, (left to right).

_____ and _____ are done last (left to right).

How would you type each equation to get answers from the Apple?

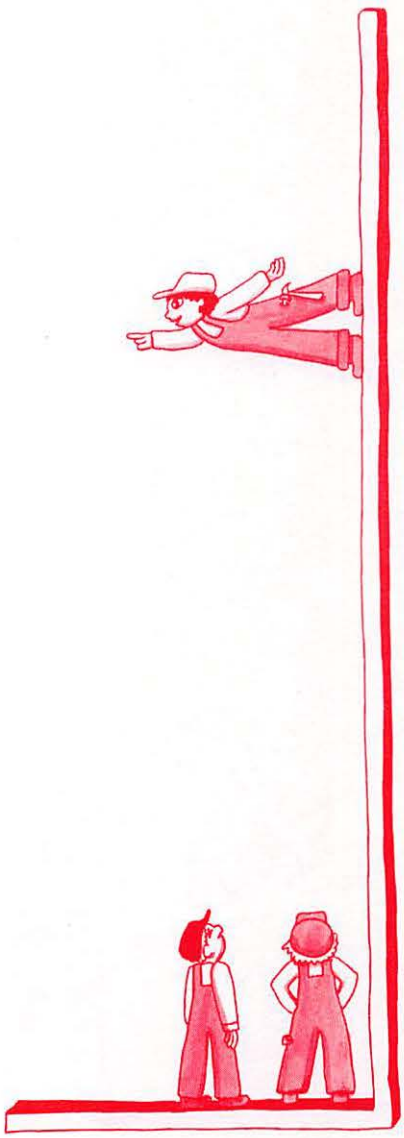
1. $457+99\times6$ _____
2. $\sqrt{64}-2$ _____
3. $26\div2^2$ _____
4. $777\times555\div222$ _____
5. 8^3-16 _____
6. $\sqrt{22}\div88$ _____
7. $\sqrt{49}+765$ _____
8. $98\div88\times66\div2^4$ _____

Show how you would type the equations above using only one PRINT statement.

_____

_____

_____



CHAPTER 13 Teaching Your Apple Simple Tricks

You can use the PRINT command to teach your Apple some simple tricks. First, teach the Apple how to print its own name.

Type: PRINT ''Apple II'' RETURN

```
] PRINT ''APPLE II''
APPLE II
] □
```

Notice that the data you want the Apple to print must be put inside quotation marks. Since you want the Apple to print APPLE II, the words Apple II must have quotation marks around them.

The Apple is in direct mode because it obeyed your command and printed Apple II *directly* after you pressed RETURN .

Let's teach the Apple to get tricky and flash on and off when it prints its name. Use the **FLASH** command before the PRINT command like this.

Type: FLASH
PRINT ''Apple II''

```
] FLASH
] PRINT ''APPLE II''
APPLE II
] □
```

Remember to press RETURN after each line.

To get the screen to stop flashing, use the command **NORMAL**. This will make the screen normal again.

```
]FLASH
]PRINT "APPLE II"
APPLE II
]NORMAL
]
```

When you type on the keyboard, the Apple prints white characters on a black background. You can make the Apple print the **inverse** (or opposite) of this by using the **INVERSE** command. INVERSE will make the Apple print black characters on a white screen.

```
]INVERSE
]PRINT "APPLE II"
APPLE II
]
```

To get the screen back to normal printing, type the command NORMAL.

```
INVERSE
]PRINT "APPLE II"
APPLE II
]NORMAL
]
```

to do: Programmer's Pastime #6

PROGRAMMER'S PASTIME #6

1. Write down what you would type to make the Apple print your name.

   _____

   Try it on the Apple to make sure it works.

2. What would you type to get the Apple to print your name in inverse?

   _____

   _____

   Try it on the Apple.

3. What would you type to make the Apple print your name and make it flash?

   _____

   _____

   Try it.

4. Make the Apple flash a message, then print it normally, and then print the message a third time in inverse. Write down the directions for doing this.

   _____

   _____

   _____

   _____

   _____

   _____

   _____

   _____

CHAPTER 14 Printing Whole Equations

You have learned how to use the PRINT command to make the Apple print the answer to an arithmetic equation.

In this chapter, you will learn how to teach the Apple to print both the equation *and* the answer.

To make the Apple print an equation, put quotation marks around the equation like this:

```
PRINT ''4+3=''
```

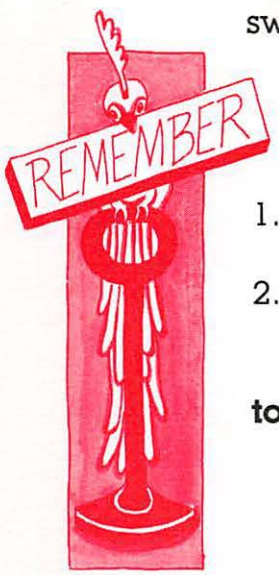
The Apple will print exactly what is inside the quotation marks.

```
]PRINT ''4+3=''
4+3=
]
```

Let's make the Apple print the equation *and* the answer. Type: PRINT ''4+3='' 4+3

```
]PRINT ''4+3='' 4+3
4+3=7
]
```

Notice that the answer-side of the equation 4+3 does not have quotation marks. When quotation marks are *not* used, the Apple will print the answer (which, in this case, is 7).



1. When you use '' '', the Apple will print what is inside.
2. When you print an equation without '' '', the Apple will print the answer.

to do: Programmer's Pastime #7
Component 2 Fun Page

COMPONENT 3

CHAPTER 15 A First Program

You have learned that computers can't think or act for themselves. They must be told what to do by people. A computer needs to follow a set of clearly written directions in order to complete even a very simple task. The set of directions that a computer must follow is called a **program**. Computer programs are written by people in computer languages such as BASIC.

There can be many steps in a computer program. The steps must be in the right order or the program will not work correctly. Each step in a program is written as one **line** on the screen. The beginning of each line must have a **line number** to help the programmer and the computer know what is to be done first, second, and so on.

It is best to use line numbers in steps of 10 like this:

```
10
20
30
40
etc.
```

This way, if you forget to put in a step, there are nine numbers between each line number in which to add the missing step. Example:

```
10
20
30
15
```

You can add line number 15 for the missing step if the step needs to come second in the program. It's OK to put step 15 last because the Apple sorts through all of the line numbers and puts them into the right order.

Never label your line numbers like this:

1
2
3
etc.

You wouldn't be able to fix your program if any steps were forgotten.

When typing a program on the keyboard, be sure to press RETURN after every line. This enters the line into memory, and moves the cursor to the next screen line.

You have learned how to make the Apple print the answer to math problems, and you know how to make the Apple print whole math equations as well. The BASIC command that tells the Apple to write something is PRINT. There is a shortcut for the PRINT command. Instead of typing the word PRINT, you need only type a question mark.

Command	**Shortcut**
PRINT	?

When you tell the Apple to print a message, you must type quotation marks around the message, like this:

```
? "THIS IS A MESSAGE"
```

The Apple will print whatever you put inside quotation marks. If you put gobbledygook inside quotation marks, the gobbledygook will be printed.

```
]? "GOBBLEDYGOOK"
GOBBLEDYGOOK
]
```

So far, you have used the PRINT command in direct mode. After pressing RETURN , the Apple prints the message on the screen. When you put a line number in front of the PRINT command, the Apple goes into **programming mode**. After pressing RETURN , nothing happens. To make the Apple print your message, you must also type the command RUN and press RETURN .

Direct Mode

```
]?"GOBBLEDYGOOK" RETURN
GOBBLEDYGOOK
]
```

Programming Mode

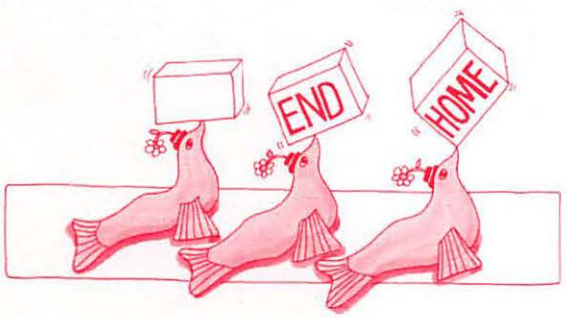
```
]10?"GOBBLEDYGOOK" RETURN
]RUN RETURN
GOBBLEDYGOOK
]
```

The **RUN** command tells the computer to do your program. Whenever line numbers are used, the Apple goes into programming mode and you must remember to use the RUN command to make the program happen. **NEVER** put a line number in front of the RUN command.

There are three other BASIC commands that should be used in the programs you write: **NEW**, **HOME**, and **END**.

You may remember that NEW is the command that erases the Apple's memory. You should type NEW and press RETURN before typing a new program. This will cause any old programs that were once in memory to be erased. NEW is **NEVER** used with a line number. Use NEW only in direct mode.

HOME is the BASIC command that erases the screen and sends the cursor "home." The cursor's



home is the top left corner of the screen. A prompt appears with the cursor so the Apple is ready to accept commands or a program. HOME works in direct mode *and* programming mode. When you type HOME in direct mode, the screen is erased and the cursor is sent home directly after you press

When you use HOME in a program after a line number, the screen is not cleared until the Apple runs the program.
program.

Direct Mode

```
] HOME RETURN
```

the Apple prints

```
]
```

Programming Mode

```
] 10 HOME RETURN
* RUN RETURN
```

the Apple prints

```
]
```

The **END** statement comes at the end of every program that you write. It tells the Apple that the program is over and there are no more instructions to follow. END should be the last statement of your program. It is only used in programming mode.

This is how you would write a simple program using NEW, HOME, PRINT, END, and RUN.

You type

```
] NEW
] 10 HOME
] 20 ? "GARBAGIO"
] 30 END
] RUN
```

The Apple Prints

```
GARBAGIO
]
```

to do: Programmer's Pastime #8, #9, #10, #11

PROGRAMMER'S PASTIME #11

Sounding Off

You learned how to make the Apple beep by pressing CTRL G. Now you can write programs that make the Apple beep by using the PRINT statement. In your program, type:

(line number) PRINT " "

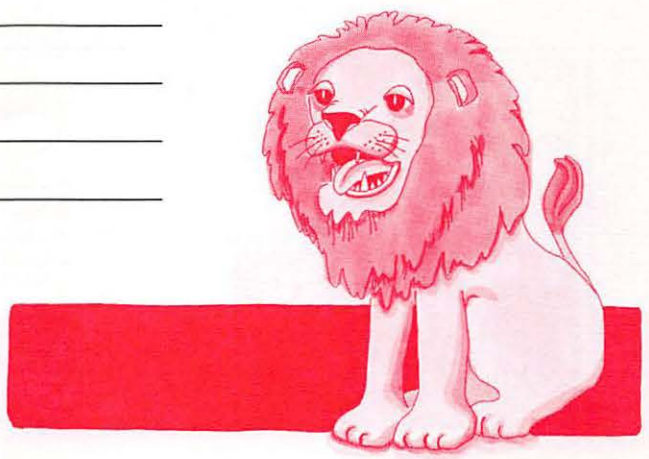
in between the quotation marks, press CTRL G.

1. Run this program to see how it works.

```
NEW
10 HOME
20 PRINT "I CAN MAKE THE APPLE BEEP"
30 PRINT " CTRL G"
40 END
```

2. Write a program that will make the Apple print the following message. Run your program on the Apple.

MARY HAD A LITTLE
(beep)
LITTLE
(beep)
LITTLE
(beep)
MARY HAD A LITTLE
(beep)
ITS FLEECE WAS WHITE AS SNOW



CHAPTER 16 Easy Graphics

It's fun to watch the Apple draw pictures on the screen. You can write programs to make the Apple draw pictures and graphics by using the PRINT statement. For example:

Program

```
NEW
10 HOME
20 ?"OOOOOO    K      K"
30 ?"O     O    K   K  "
40 ?"O     O    K K    "
50 ?"O     O    K   K  "
60 ?"OOOOOO    K      K"
70 END
```

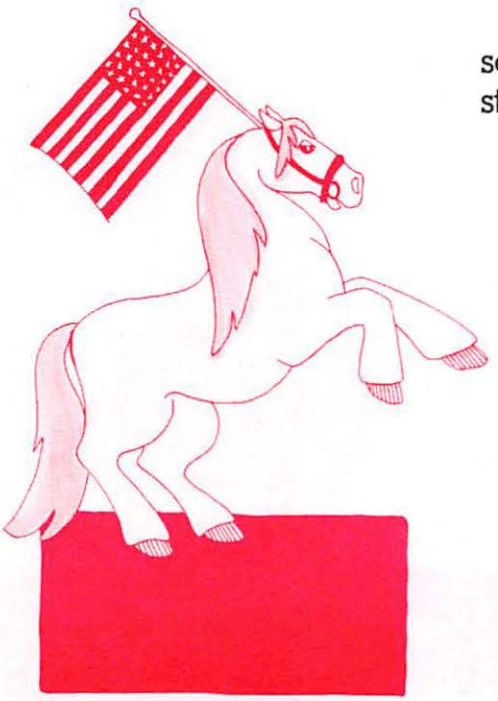
Output

```
OOOOOO    K      K
O     O   K   K
O     O   K K
O     O   K   K
OOOOOO    K      K
]
```

Type the program on the keyboard so it appears on the screen the way you would like it to look. You will have to put the spaces in the right places. If you want the Apple to make one whole line of the screen blank, type a PRINT statement with nothing after it, like this:

(line number) PRINT or (line number) ?

Study this program, which prints a flag that resembles the American flag. Notice how the PRINT statements are used to make whole lines blank.



Program

```
10  HOME
20  ?``* * * * * * * * * * * ##########################################''
30  ?`` * * * * * * * * * * *                                          ''
40  ?``* * * * * * * * * * * ##########################################''
50  ?``* * * * * * * * * * *                                           ''
60  ?`` * * * * * * * * * * ##########################################''
70  ?
80  ?`` ######################################################################''
90  ?
100 ?`` ######################################################################''
110 ?
120 ?`` ######################################################################''
130 ?
140 ?`` ######################################################################''
150 END
```

Output

```
* * * * * * * * * * ##################################################
* * * * * * * * * *
* * * * * * * * * * ##################################################
* * * * * * * * * *
* * * * * * * * * * ##################################################

##########################################################################

##########################################################################

##########################################################################

##########################################################################

]
```

to do: Programmer's Pastime #12, #13

CHAPTER 17 Formatting Screen Output

A **format** is a plan for the arrangement of something. Formatting screen output on the Apple means writing programs so the screen output is arranged a certain way when the program is run.

In Chapter 12, you learned how to make the Apple print the answer to more than one equation in one PRINT statement by using commas like this:

Type

```
]PRINT 2+2, 3+3, 4+4 RETURN
```

Output

```
4           6           8
]
```

The Apple prints the answers on one screen line.

This works the same way with words. You must remember to put quotation marks around the words you want the Apple to print.

Type

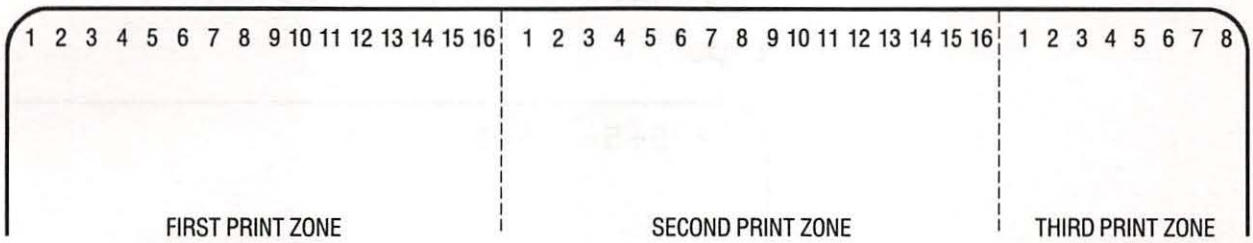
```
]?"READY", "SET", "GO" RETURN
```

Output

```
READY       SET         GO
]
```

Notice that each word in the output is separated on the screen by many spaces. The commas cause this to happen. The Apple's screen has three **print zones**. The first and second print zones can hold 16 characters. The third print zone holds 8 characters. Because 16+16+8=40, this means you can type 40 characters on one line across the Apple's screen.

Screen

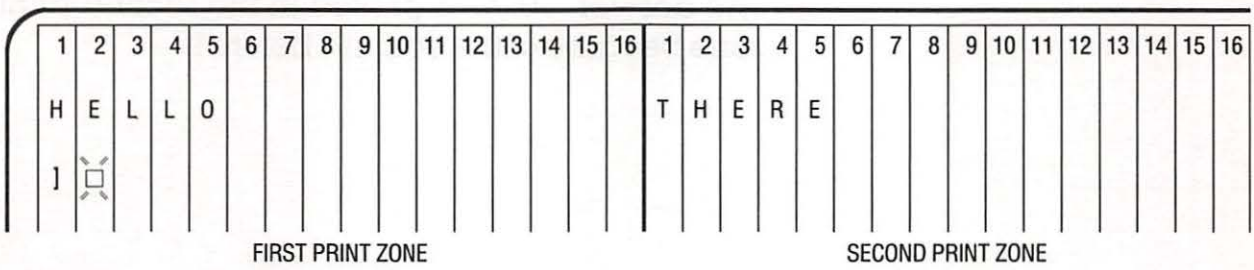


When commas are used in a PRINT statement, each piece of output will be printed in a separate print zone.

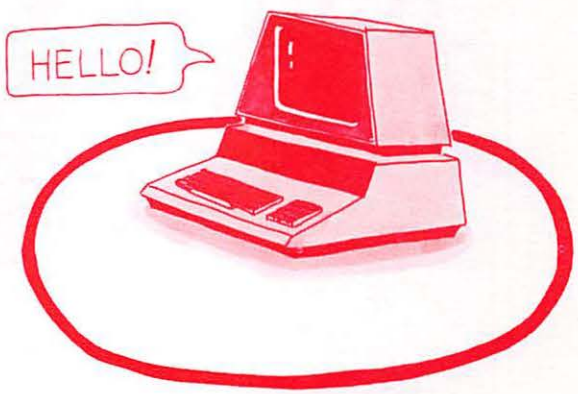
Type

```
]?"HELLO","THERE"
```

Output



The word HELLO was printed in the first print zone because it is the first word in the PRINT statement. The second word, THERE, was printed in the second print zone because of the comma before it.



Commas in PRINT statements will affect numbers and equations in the same way.

Type

```
]?"5+5=",5+5
```

Output

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
5	+	5	=													1	0														
]																															

FIRST PRINT ZONE — SECOND PRINT ZONE

The equation 5+5= was printed in the first print zone because it comes first in the PRINT statement. The answer, 10, was printed in the second print zone because of the comma before it.

Instead of using commas and making the screen output spaced out, you can use semi-colons (;) in a PRINT statement to make the output different. When semi-colons are used in a PRINT statement, the output is *not* spaced out in separate print zones. The Apple prints the output in the *next* screen column.

Type

```
]?"HELLO"; "THERE"
```

Output

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 | 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
H E L L O T H E R E
]
```

FIRST PRINT ZONE SECOND PRINT ZONE

Type

```
]?"5+5="; 5+5
```

Output

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 | 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
5 + 5 = 1 0
]
```

FIRST PRINT ZONE SECOND PRINT ZONE

Remember that a PRINT statement with nothing after it leaves a blank line on the Apple's screen.

Type

```
]NEW
]10 HOME
]20 ?"2*2=", 2*2
]30 ?
]40 ?"2*2=", 2*2
]50 END
]RUN
```

Output

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
2 * 2 = 4
2 * 2 =                                 4
]
```

FIRST PRINT ZONE SECOND PRINT ZONE

1. A comma tells the Apple to go to the next print zone and then begin printing.
2. A semi-colon holds the cursor at the end of the last thing printed. Then it prints the next thing in the very next column.

to do: Programmer's Pastime #14, #15

PROGRAMMER'S PASTIME #15

Ralph wants to run a program that will make Apple print the following output:

Output

```
***
SMASHEDTOGETHER

OR

SPACED                    APART
???
] □
```

He can't get his program lines in the right order to make the program work! Put the program lines below in the correct order with the correct line numbers so the program will print what Ralph wants.

Ralph's Program	**Correct Order**
?''???''	10
?	20
HOME	30
?''SMASHED'';''TOGETHER''	40
?''OR''	50
END	60
?'' *** ''	70
?''SPACED'',''APART''	80
?	90

CHAPTER 18 A Shortcut

As a computer programmer, you should always be looking for useful shortcuts that will make writing programs easier.

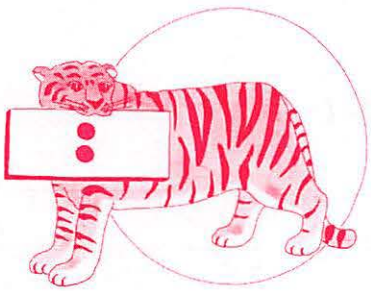
You can use colons (:) to shorten your programs. Put colons between statements so you can have many statements on one program line.

Type

```
] 10 HOME
] 20 ? "ONE" : ? "TWO" : ? "BUCKLE MY" :
     ? "SHOE"
] 30 END
```

Output

```
ONE
TWO
BUCKLE MY
SHOE
]
```



Using this shortcut, you can take a long program such as this:

```
10 HOME
20 FLASH
30 ? "FLASH GORDON"
40 INVERSE
50 ? "STRIKES AGAIN!"
60 NORMAL
70 END
```

and shorten it to this:

```
10 HOME
20 FLASH: ? "FLASH GORDON":INVERSE:
   ? "STRIKES AGAIN!":NORMAL
30 END
```

Output

```
FLASH GORDON
STRIKES AGAIN
]
```

to do: Programmer's Pastime #16, #17

PROGRAMMER'S PASTIME #16

Using what you have learned about commas, semi-colons, and colons, write a program that will print a Christmas tree. Give the Christmas tree ornaments in inverse and rows of lights that flash.

You might begin your program like this:

Type

```
] 10 HOME
] 20 ? ""          *          ""
] 30 ? ""         XXX         ""
] 40 ? ""        XXXXX        ":FLASH
] 50 ? ""       XXXXXXX       ":NORMAL
```

Output

```
        *
       XXX
      XXXXX
     XXXXXXX
```

Write your program on the lines below. Run your program on the Apple to make sure it works.

CHAPTER 19 Getting Out the Bugs

Writing a computer program can be a long process. It often takes many tries before a programmer gets a program to work properly. This is because there can be **bugs** in the program. No, there aren't little insects climbing around inside the computer. Bugs are mistakes that you and computer programmers can make in writing a program.

Some examples of bugs might be:

1. Forgetting to type a punctuation mark.

   Typing 10 ? HI'' instead of 10 ? ''HI''

2. Spelling a command wrong.

   Typing 30 ED instead of 30 END

3. Putting the steps of your program in the wrong order.

	instead of	
10 ? ''WRITE''		10 HOME
20 HOME		20 ? ''WRITE''
30 END		30 END

It is very important to check your program for bugs before you run it or save it on a disk. To check your program instructions, type:

LIST [RETURN]

The Apple will list all of the instruction lines in your program on the screen. Now you can check each instruction for bugs. If you find a bug, you can get rid of it by fixing the instruction.

Once you find a bug in your program retype the line with the bug the correct way and press [RETURN] . The old line will be replaced with the new line. List the program again to make sure the bug was fixed.

Example:

Type

```
] LIST
```

Output

```
] LIST
10 HOME
20 ? "A MESSTAKE"
30 END
] □
```

Type

```
] 20 ? "A MISTAKE" RETURN
] LIST
```

Output

```
] LIST
10 HOME
20 ? "A MISTAKE"
30 END
] □
```

There is another way that you can fix program bugs on the Apple. If you want to erase a whole program line, type the line number and press RETURN

Example:

Type

```
] LIST
10 HOME
20 ? "DEBUG"
30 ERASE THIS LINE
40 END
] □
```

Type

```
] 30 RETURN
```

Check by Listing Again

```
] LIST
10 HOME
20 ? "DEBUG"
40 END
] □
```

Even after you think you have corrected all of the bugs in your program, you may find more bugs after you run your program on the Apple. The best way to find and fix all of the bugs in a program is to take turns running *and* listing the program. If you find more bugs in the program listing, use the tricks you just learned to fix them.

If you are working on a long program, you may not want to list the whole thing. You can list certain parts of a program by typing:

LIST 40,60 RETURN

In this example, the Apple will list lines 40 through 60.

Type

```
] LIST 40,60 RETURN
```

Output

```
] LIST 40,60
40 ? "TO FETCH"
50 ? "A PAIL"
60 ? "OF WATER"
] □
```

To list only one program line, type:

LIST 70 RETURN

Type

```
] LIST 70 RETURN
```

Output

```
] LIST 70
70 ? "THE END"
] □
```

To list all of the programs *up to* a certain point, type:

LIST ,40 RETURN

Type

```
] LIST ,40 RETURN
```

Output

```
] LIST ,40
10 HOME
20 ? "JACK & JILL"
30 ? "WENT UP THE HILL"
40 ? "TO FETCH"
] □
```

To list all program lines *past* a certain point, type:

LIST 40, RETURN

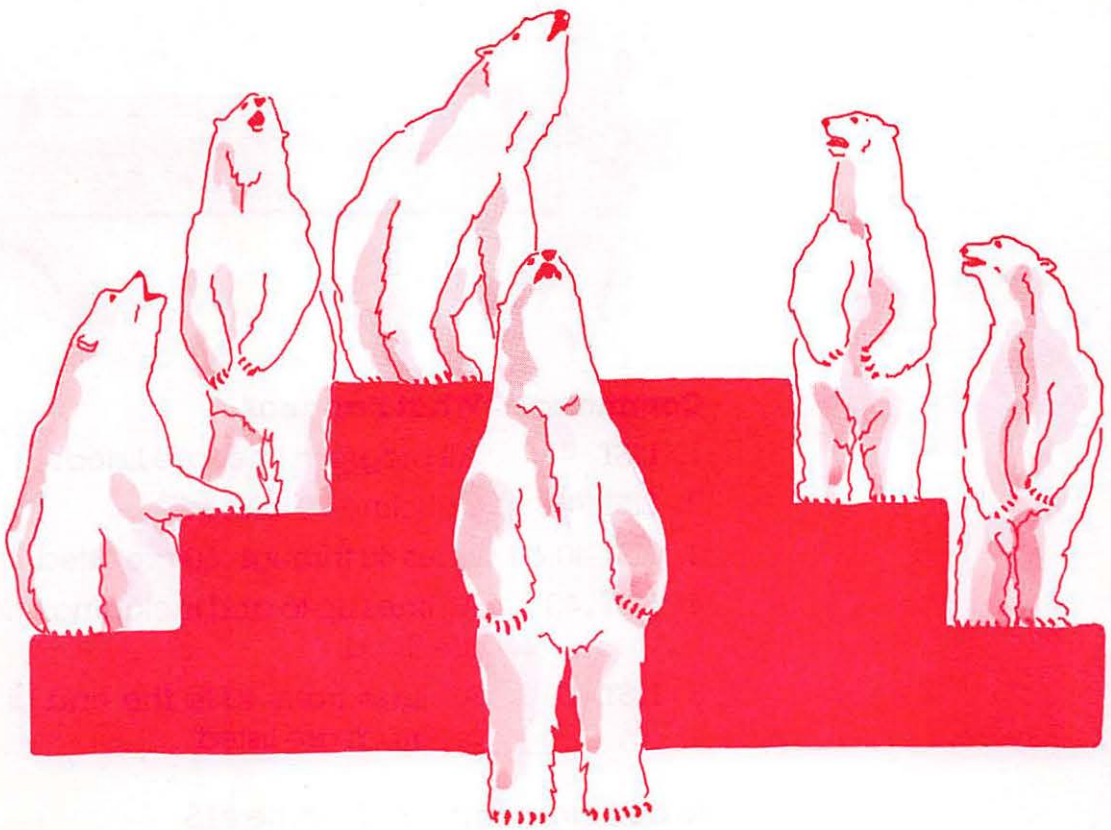
Type

```
] LIST 40, RETURN
```

Output

```
] LIST 40,
40 ? "TO FETCH"
50 ? "A PAIL"
60 ? "OF WATER"
70 ? "THE END"
80 END
]
```

The entire process of getting rid of program bugs is called **debugging**. Don't let program bugs "bug you" because now you know how to fix them!



Computer Errors

There are three types of errors you may run across as you work with computers. Sometimes the Apple will tell you what your error is. Other times you will have to figure out yourself what the error is and where it happened.

User Errors. A user error happens when you—the user—make a typing mistake or fail to communicate with the Apple in Applesoft BASIC.

Program Errors. A program error occurs when there are bugs in your program. You will have to debug your program to correct the errors.

Computer Errors. A computer error could happen if not all of the computer's equipment is hooked up properly. These errors can be very complicated, but they rarely occur.

To find out what certain errors mean, turn to Appendix B at the back of this book.

Command	What Happens
1. LIST	All program lines are listed.
2. LIST 70	Only line 70 is listed.
3. LIST 40,60	Lines 40 through 60 are listed.
4. LIST ,40	All lines up to and including line 40 are listed.
5. LIST 40,	All lines from 40 to the end of the program are listed.

to do: Programmer's Pastime #18

CHAPTER 20 Using the Disk Drive

Now that you are writing some interesting programs, you will want to **SAVE** them so you can run them and enjoy them over and over. The place to store programs is on a disk. Once a program is saved on a disk, you can load the program into the computer and run it whenever you like.

If you are saving a program on a brand new disk, turn to Appendix A, Initializing New Disks. New disks have to be **initialized** or set up so programs can be saved on them.

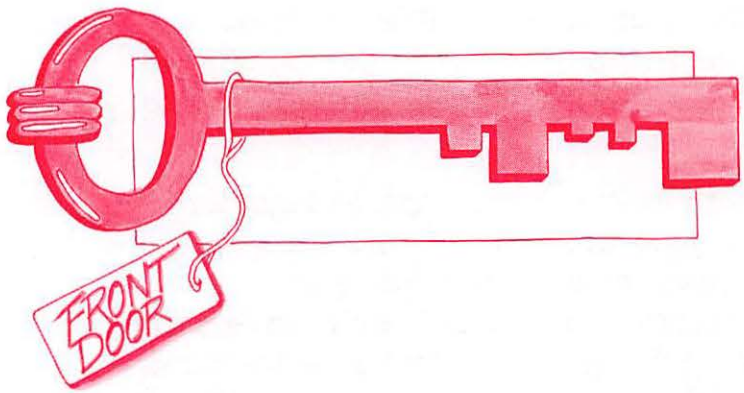
If your disk is already initialized or contains a few programs, you may be able to store additional programs if the disk isn't totally full. Follow these instructions:

1. Type your program on the Apple's keyboard.
2. Run your program to make sure it works. Debug the program to fix any mistakes.
3. Put the disk on which you want to save your program in the disk drive and close the door.
4. Type: SAVE _____ RETURN .
   (Type the name of your program in the blank.)
5. The disk drive will whir as the program is saved on the disk. When the red light on the disk drive goes out and the whirring stops, the saving process will be finished.
6. Type: CATALOG RETURN .
7. If the saving process worked properly, you should see the name of your program listed in the catalog with all of the other programs that are stored on the disk.

Erasing a Program

If you decide that you no longer want to keep a program on a disk, you may erase the program from the disk. Follow these instructions:

1. Insert the disk with the program you want to erase into the disk drive. Close the door.
2. Type: DELETE _____ RETURN .
   (Type the name of the program here.)
3. The disk drive will erase the program.
4. Type: CATALOG to make sure your program got erased. If it did, you will not see it listed in the catalog.



Lock

You may save some very important programs on a disk someday. You will want to make sure that these programs can't get accidentally erased from the disk. Use the **LOCK** command to protect these programs. To lock a program on a disk, follow these instructions:

1. Insert the disk with the program you want to lock into the disk drive. Close the door.
2. Type: LOCK _____ RETURN .
   (Type the name of the program here.)
3. The disk drive will lock the program.
4. Type: CATALOG. Look for your program in the catalog listing. If your program name now has an asterisk (*) in front of it, you will know that it is locked safely on the disk.

Renaming

You can change the name of any program that is saved on a disk. To rename a program, first make sure that it is not locked. Then follow these instructions:

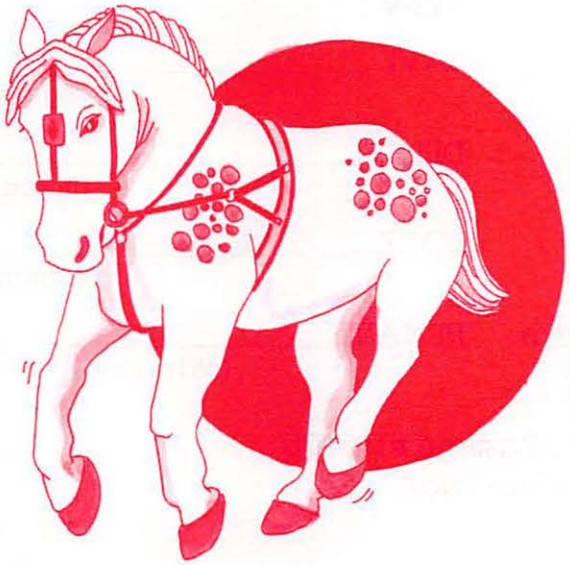
1. Insert the disk with the program you wish to rename. Close the door.
2. Type: RENAME _____, _____ RETURN .
   first blank second blank
   Type the old name in the first blank. Type the new name that you wish the program to have in the second blank.
3. The disk drive will rename the program.
4. Type: CATALOG. Make sure your program has been correctly renamed.

Naming Rules

There are certain rules you must follow when naming a program.

A program name:

1. can be up to 30 characters long (this includes numbers and spaces between words);
2. must start with a letter;
3. cannot have a comma in it.



Review

To	Type
1. Save a program:	SAVE _____ (program name)
2. Erase a program:	DELETE _____ (program name)
3. Lock a program:	LOCK _____ (program name)
4. Rename a program:	RENAME _____ (old name) , _____ (new name)

to do: Programmer's Pastime #19, #20
Component 3 Fun Page

COMPONENT 4

CHAPTER 21 Remarks

As you begin writing more complicated programs, you will want to make sure they can be easily read and understood by others who may read them. Writing your programs so they are easy to read is good programming style.

One style technique is the use of **REMARK** statements—REM for short. This is called **program documentation**, which means noting what is happening in your program. For example:

```
10 REM PRINT MY NAME
20 HOME
30 ? "EGBERT"
40 END
```

Each REM statement describes the purpose of the statements following it. Since line 30 prints a name, the REM statement in line 10 says:

```
10 REM PRINT MY NAME
```

When the program is run, the Apple will *ignore* the REM statement. REM tells the computer to ignore what is written on that line and go on to the next line number. The Apple will list REM statements, but it will not run them.

Use REM statements at the beginning of your programs to tell the name of your program or what it does. You can also show that you are the author of the program.

For example:

```
10 REM SPACE ATTACK
20 REM TRY TO SHOOT DOWN THE ALIENS
30 REM WRITTEN BY JOE COOL, 1980
```

It is also helpful to use REM statements to describe each main section of your program. For example:

```
10 REM PRINTING EQUATIONS
20 REM WRITTEN BY CHARLIE BROWN, 1981
30 REM ADDITION
40 ?"6+6=" 6+6
50 ?"6+7=" 6+7
60 REM MULTIPLICATION
70 ?"6*6=" 6*6
80 ?"6*7=" 6*7
90 END
```

In the program above, we used REM statements to:

1. introduce the program;
2. show the beginning of the addition section of the program;
3. show the beginning of the multiplication section of the program.

Be careful that you do not use too many REM statements in your programs. Too many can clutter the program, making it *more* difficult to read. Too many can also waste screen space and use up memory. As you practice writing programs you will become more aware of where REM statements should be placed.

to do: Programmer's Pastime #21

CHAPTER 22 Color on the Screen

If your Apple has a color monitor, you will enjoy drawing screen pictures with little colored blocks. This is called **lo res graphics**.

Remember that the Apple's screen has 40 **columns**. This means you can type 40 characters across one line of the screen. The Apple's screen also has 40 **rows** on which graphics can be drawn. The screen columns and rows are labeled from 0 to 39.

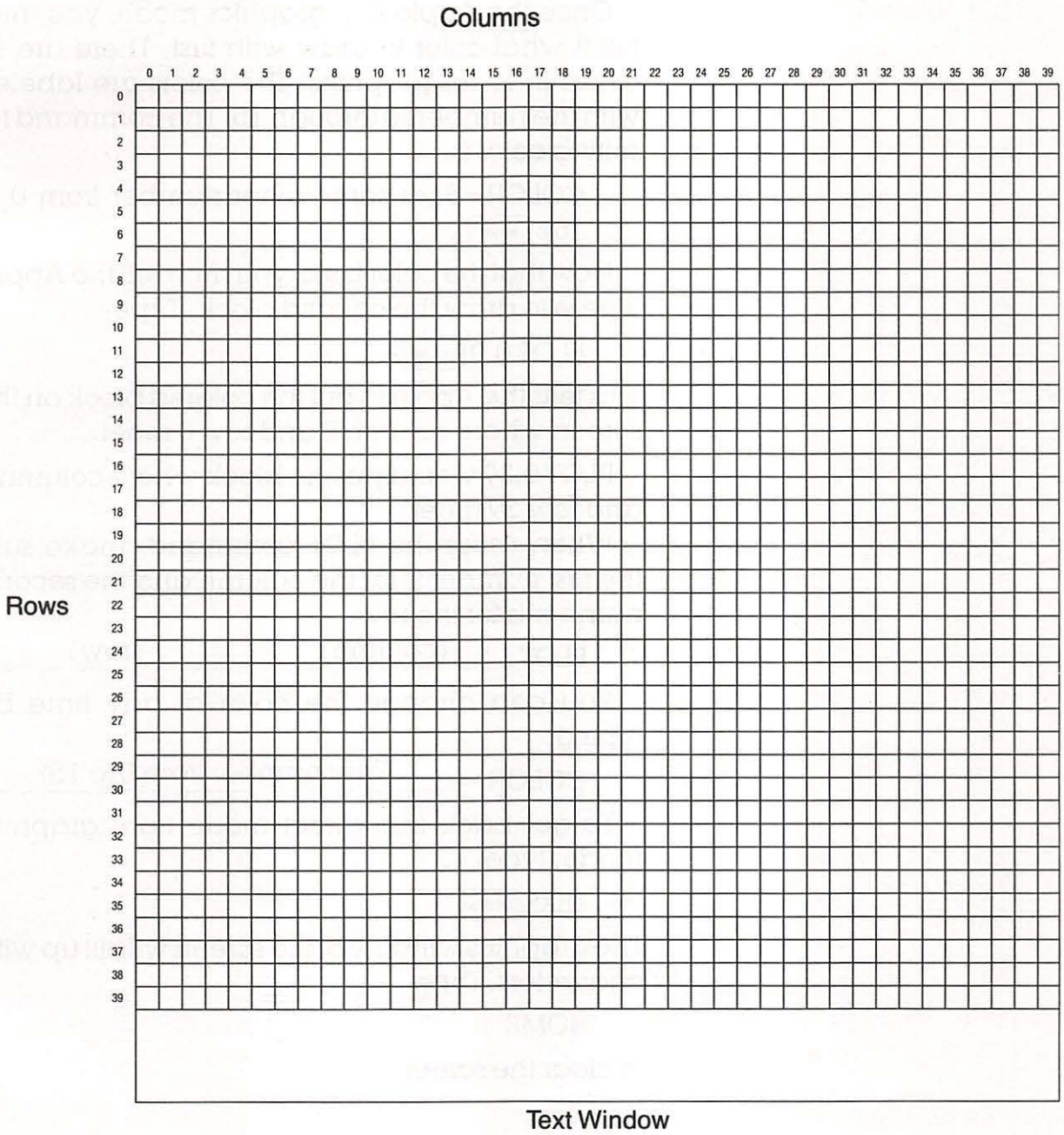
Think of the Apple's screen as a big grid of rows and columns. Each little grid block can be colored to make a picture.

To build pictures with colored blocks, first type:

GR RETURN

GR stands for graphics. Typing GR and pressing RETURN puts the Apple into **graphics mode**.

When the Apple is in graphics mode, all of the screen can be used for making pictures except the very bottom of the screen. The Apple saves the bottom four rows for writing. When you type in graphics mode, the writing will appear on the bottom four screen lines. This space is called a **text window**.


Columns
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
Rows
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
Text Window

Once the Apple is in graphics mode, you must tell it what color to draw with first. There are 16 colors in lo res graphics. The colors are labeled with the numbers 0 through 15. The command for setting color is:

COLOR=5 (or some other number from 0 to 15), RETURN .

Now that the color is set, you must tell the Apple *where* to draw the colored block. Type:

PLOT 0,0 RETURN

This tells the Apple to put the colored block on the screen where column 0 and row 0 meet.

PLOT 0,39 would put the block where column 0 and row 39 meet.

When using the PLOT command, make sure the first number is for the column and the second number is for the row.

PLOT (column) , (row)

You can change the color at any time by typing:

COLOR = (any number from 0 to 15)

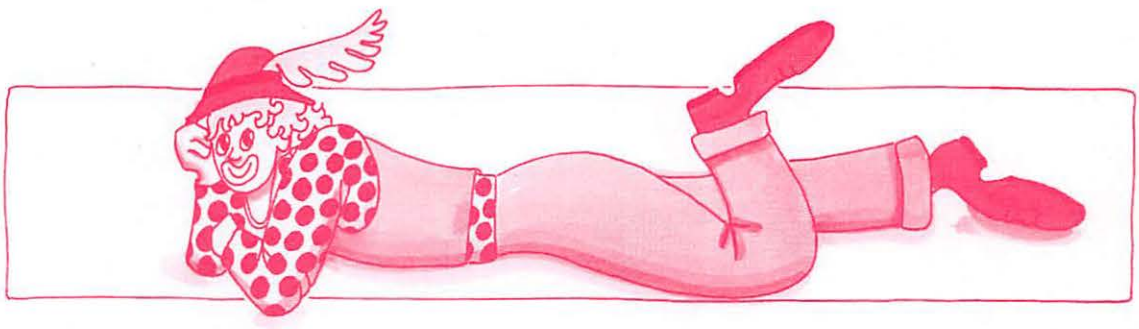
To get back into direct mode from graphics mode, type:

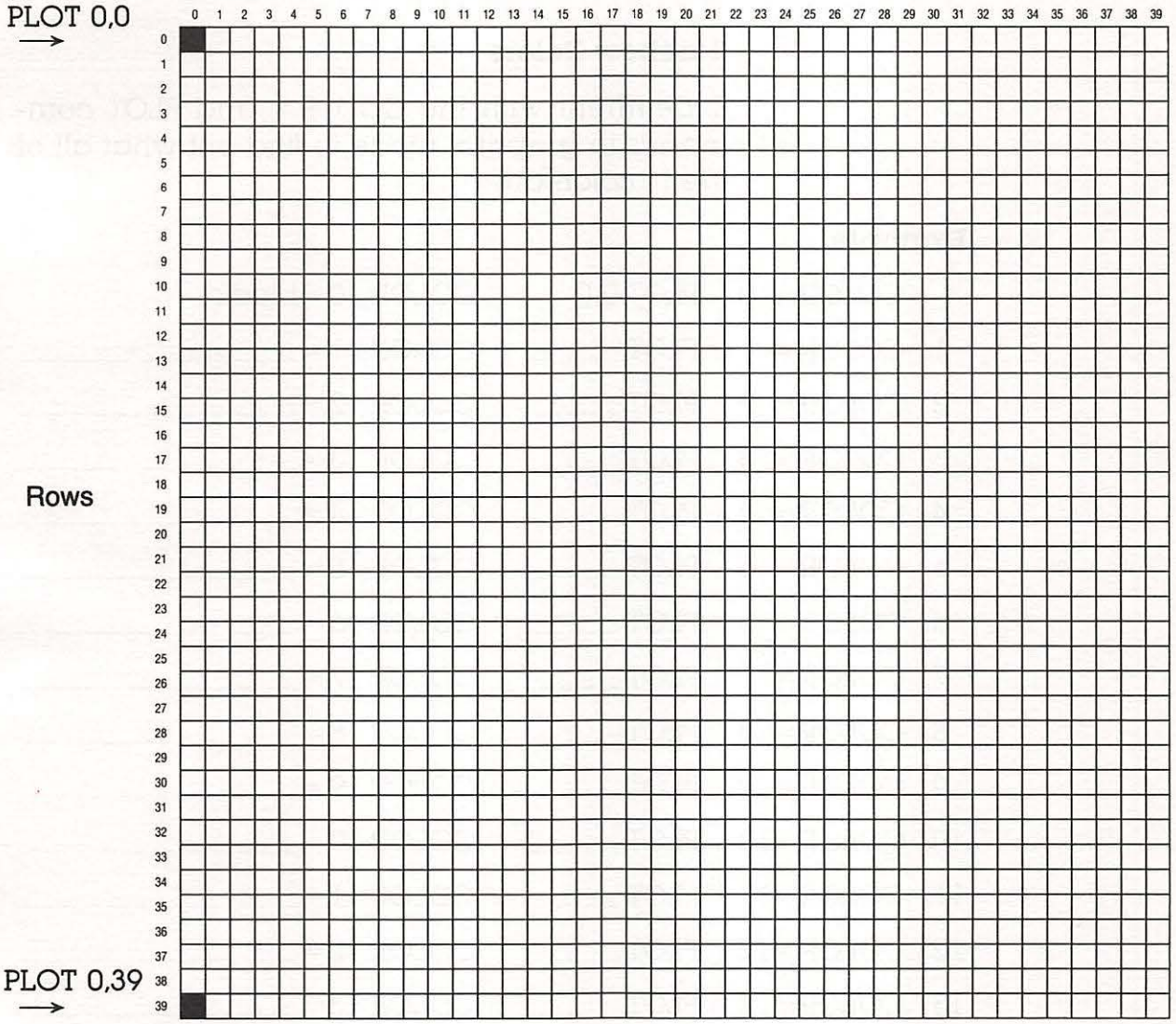
TEXT RETURN

The graphics window of the screen will fill up with characters. Type:

HOME

to clear the screen.





Review

Command	What Happens
GR	The Apple goes into graphics mode.
COLOR=(number from 0 to 15)	The color is set.
PLOT (column),(row)	The colored block is placed on the screen.
TEXT	The Apple goes back to direct mode.

to do: Programmer's Pastime #22, #23

PROGRAMMER'S PASTIME #22

Rainbow Colors

Experiment with the COLOR= and PLOT commands in graphics mode to find out what all of the 16 colors are.

Example:

	COLOR= 0	PLOT 0,0	COLOR 0= black
1.	COLOR= 1	PLOT ___,___	COLOR 1= _____
2.	COLOR= 2	PLOT ___,___	COLOR 2= _____
3.	COLOR= 3	PLOT ___,___	COLOR 3= _____
4.	COLOR= 4	PLOT ___,___	COLOR 4= _____
5.	COLOR= 5	PLOT ___,___	COLOR 5= _____
6.	COLOR= 6	PLOT ___,___	COLOR 6= _____
7.	COLOR= 7	PLOT ___,___	COLOR 7= _____
8.	COLOR= 8	PLOT ___,___	COLOR 8= _____
9.	COLOR= 9	PLOT ___,___	COLOR 9= _____
10.	COLOR=10	PLOT ___,___	COLOR 10= _____
11.	COLOR=11	PLOT ___,___	COLOR 11= _____
12.	COLOR=12	PLOT ___,___	COLOR 12= _____
13.	COLOR=13	PLOT ___,___	COLOR 13= _____
14.	COLOR=14	PLOT ___,___	COLOR 14= _____
15.	COLOR=15	PLOT ___,___	COLOR 15= _____

CHAPTER 23 Colored Lines

You can make colored graphics on the Apple screen much easier if you draw with lines instead of with the colored blocks.

Let's say you decided to draw an orange line across the screen at row 10. You could type:

```
]GR
]COLOR=9
]PLOT 0,10
]PLOT 1,10
]PLOT 2,10
    .
    .
    .
```

and so on until you typed PLOT 39,10. This would take a long time to do!

There is an easier way to draw the same orange line across row 10. Use a graphic command called **HLIN** to draw across the screen. H in HLIN stands for **horizontal**. LIN stands for line. Horizontal means across. Type:

```
]GR
]COLOR=9
]HLIN 0,39 at 10
```

After typing these three statements, the Apple will quickly print an orange horizontal line across the screen. HLIN 0,39 at 10 means draw a line across the screen from column 0 to 39 at row 10.

You can also make lines that go up and down on the screen. These lines are called **vertical** lines. Vertical means up and down. Use the **VLIN** command to draw these lines. V stands for vertical. LIN stands for line. To draw a vertical line down the center of the screen type:

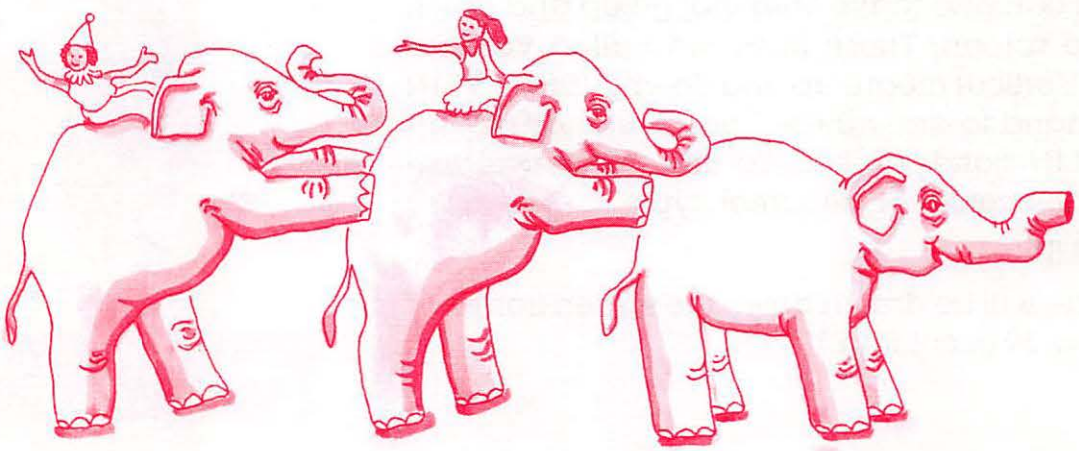
```
VLIN 0,39 at 19
```

The line will be drawn down the screen from row 0 to row 39 at column 19.

Columns

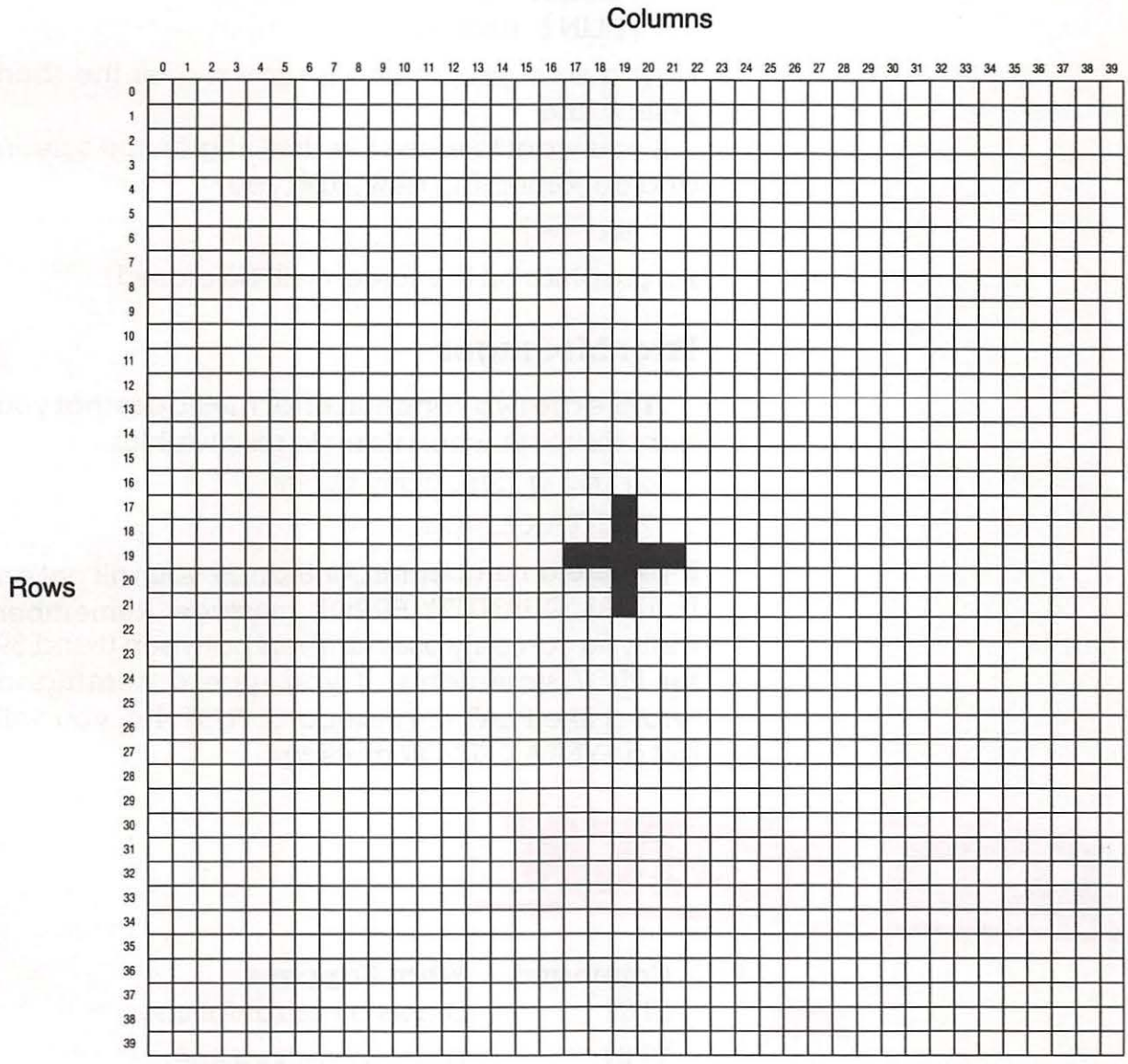
Rows

HLIN 0,39 AT 10
VLIN 0,39 AT 19



Using HLIN and VLIN you can make the Apple draw lines of any length up and down and across the screen. To make a small pink cross in the center of the screen, type:

```
] GR
] COLOR=11
] HLIN 17,21 at 19
] VLIN 17,21 at 19
```



HLIN 17,21 AT 19
VLIN 17,21 AT 19

If you try to put a new color over another color already on the screen, the new color takes over. If you type:

```
] COLOR=13
] HLIN 5,10 at 20
```

a short yellow line will be drawn.

If you change the color and type the same thing, the new color will take over.

```
] COLOR=3
] HLIN 5,10 at 20
```

Now a short purple line is drawn over the short yellow line.

If you want to erase the drawing on the screen and do something new, just type:

GR RETURN

All graphics on the screen will be erased.

Error Messages

There are two common error messages that you may discover when using lo res graphics:

1. ?ILLEGAL QUANTITY ERROR
2. ?SYNTAX ERROR

If you use a number larger than 39 you will get an ILLEGAL QUANTITY ERROR message. Remember that you can only use numbers between 0 and 39 for PLOT statements. If you type a command wrong like PLAT 4,4 instead of PLOT 4,4, you will get a SYNTAX ERROR message.

Command	What Happens
HLIN	Draws a horizontal line.
VLIN	Draws a vertical line.
GR	Erases the graphics screen. (Also puts the Apple into graphics mode.)

to do: Programmer's Pastime #24, #25, #26, #27

PROGRAMMER'S PASTIME #25

Use HLIN and VLIN to make a colored border around the Apple's screen. Inside the border, make the Apple draw a large, colored set of your initials. Write the instructions in program form on the lines below. Use REM statements.

CHAPTER 24 Flow Diagramming

When you learn how to play a game, you must read a set of instructions. These instructions are written in a clear and orderly step-by-step manner. If the instructions are mixed up and out of order, you won't understand how to play the game.

The same is true for computers. When you write a program to teach the Apple a trick or to solve a problem for you, the instructions in your program must be in a clear, step-by-step order. If you don't plan your program steps carefully, the Apple will not understand what to do.

There is a process that you can use when you write a program that will help you write your steps clearly and in the correct order. This process is called **flow diagramming**.

An **algorithm** (al' go rith m) is a step-by-step method you use to solve a problem. Every problem has a certain algorithm that you can use to solve it. For example:

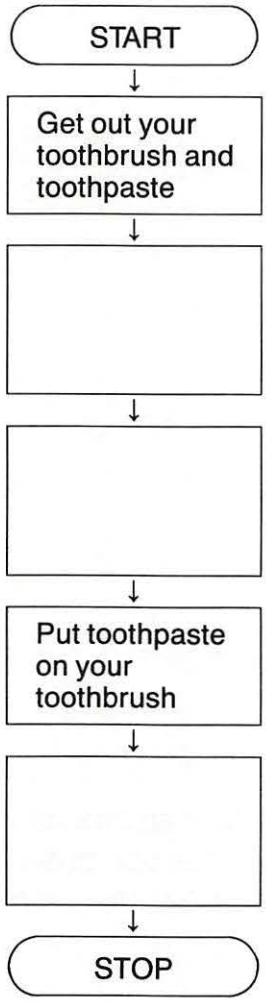
Problem	**Algorithm**
Your front door is locked.	1. Find your key. 2. Put the key into the door lock. 3. Turn the key. 4. Open the door.

By following the algorithm, you can solve the problem of being locked out of your house.

When you do flow diagramming, you must show how the algorithm works by putting it into flow chart form. Here is how you could write an algorithm in a flow chart.

A **flow chart** is a diagram that shows all of the steps of an algorithm in the correct order. The arrows in a flow chart show how the steps are connected.

Below is a flow chart that shows an algorithm on how to brush your teeth. Think about which steps at the side of the flow chart would fit in the blank boxes.



Missing Steps

Brush your teeth.

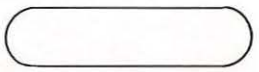
Wet your brush.

Unscrew toothpaste cap.

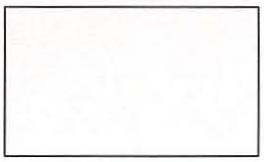
Notice that the boxes in a flow chart have different shapes. What shape are the START and STOP boxes? We usually begin a flow chart with a (START) instruction and end with a (STOP) instruction.

The boxes that tell you to do something are shaped like rectangles. They are called **processing boxes**.

Shape



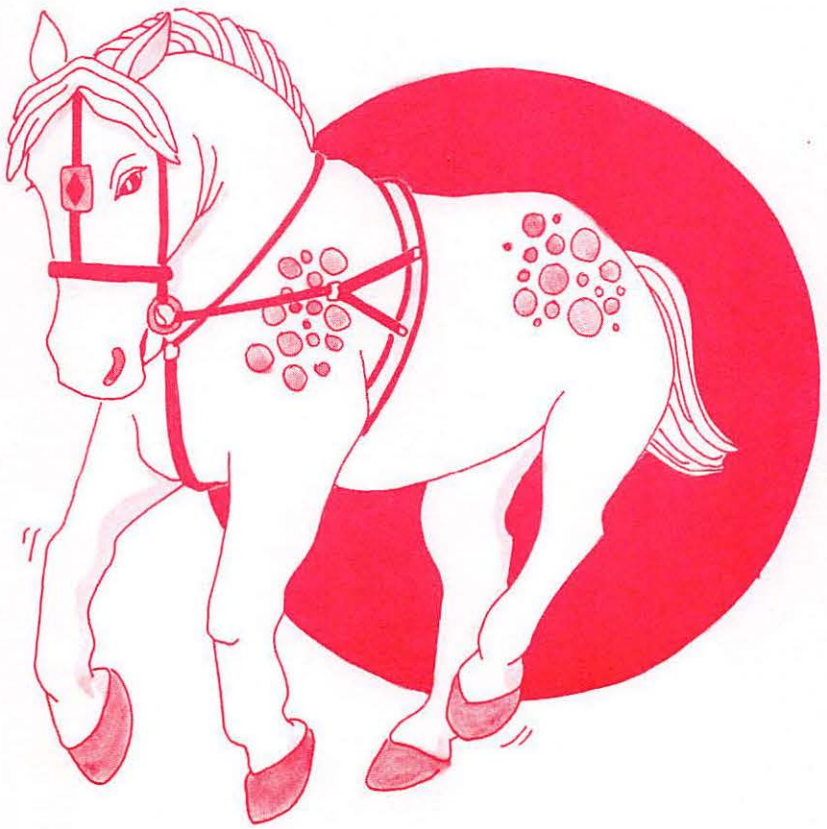
START or STOP box



PROCESSING box

Let's practice writing algorithms and putting them into flow chart form.

to do: Programmer's Pastime #28, #29, #30



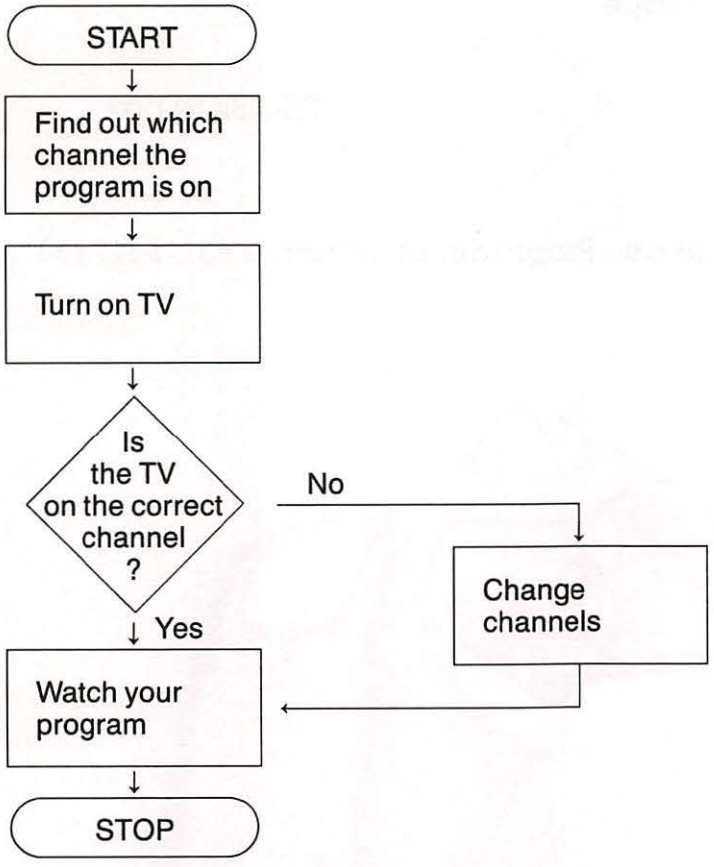
PROGRAMMER'S PASTIME #30

Design an algorithm for making a peanut butter and banana sandwich. Write your algorithm in flow chart form.

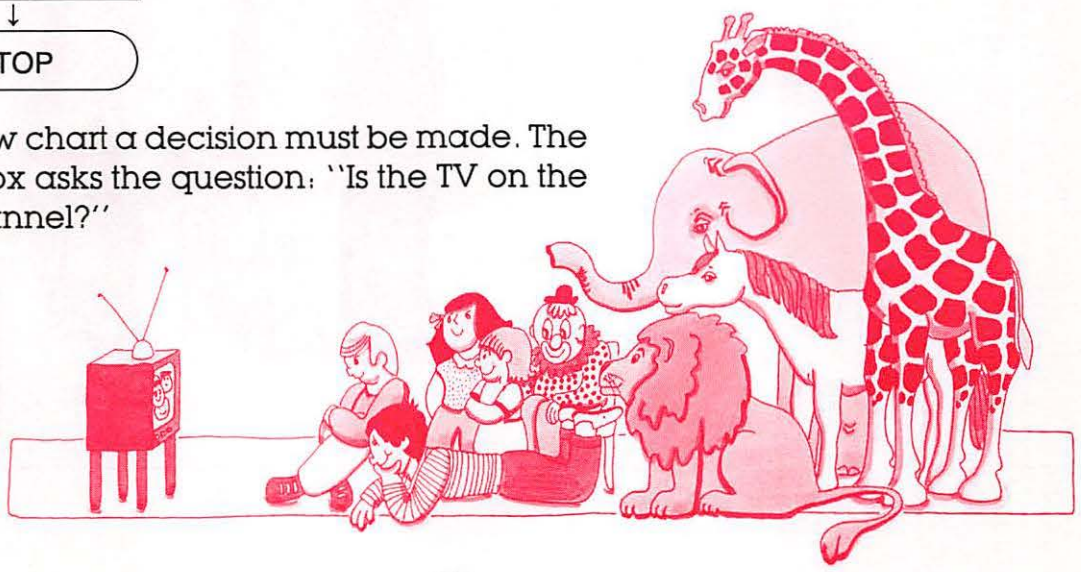
CHAPTER 25 More About Flow Charts

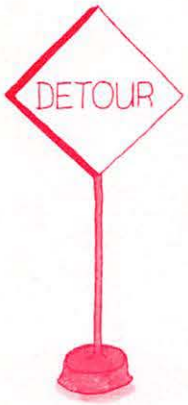
Sometimes there will be a step in a flow chart that asks a question. A question in a flow chart is written in a diamond-shaped box. This is called a **decision box**.

Algorithm/Flow Chart on How to Watch a TV Program



In this flow chart a decision must be made. The decision box asks the question: "Is the TV on the correct channel?"

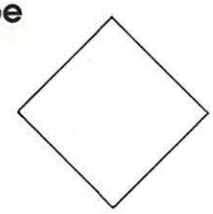




If the answer is yes, you will follow the main path of the flow chart. If the answer is no, you will take a detour and follow a different path. During the detour, there is another task to do—change the channel. Notice how the detour comes back to the main part of the flow chart before it ends.

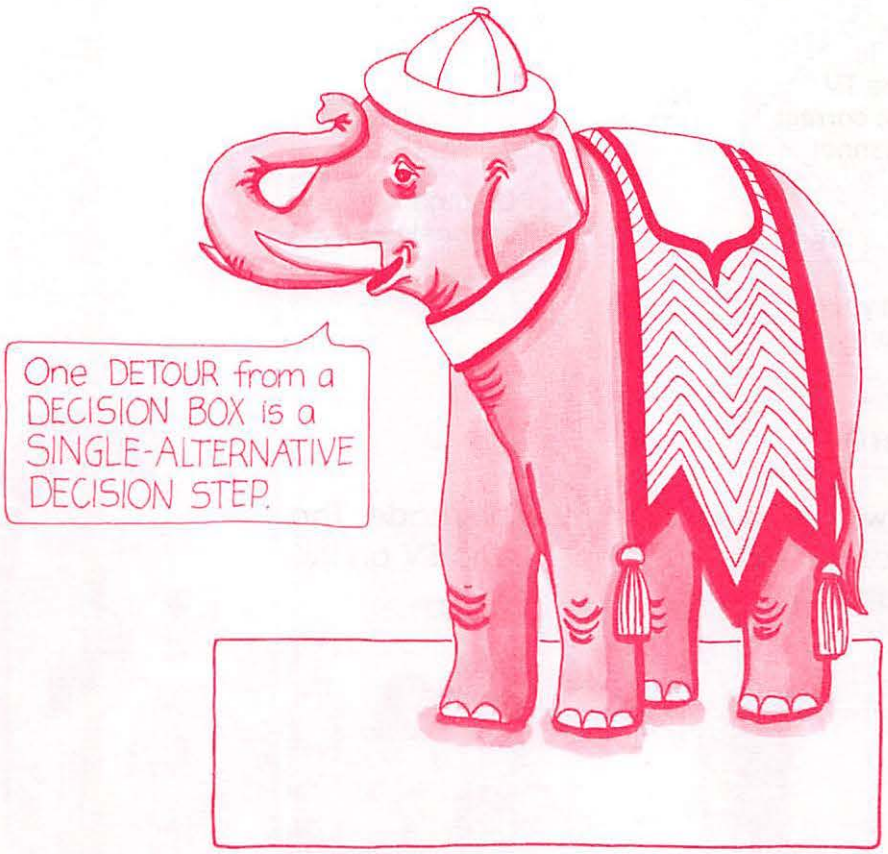
When there is *one* detour from a decision box in a flow chart, the flow chart is said to have a **single-alternative decision step**.

Shape



DECISION box

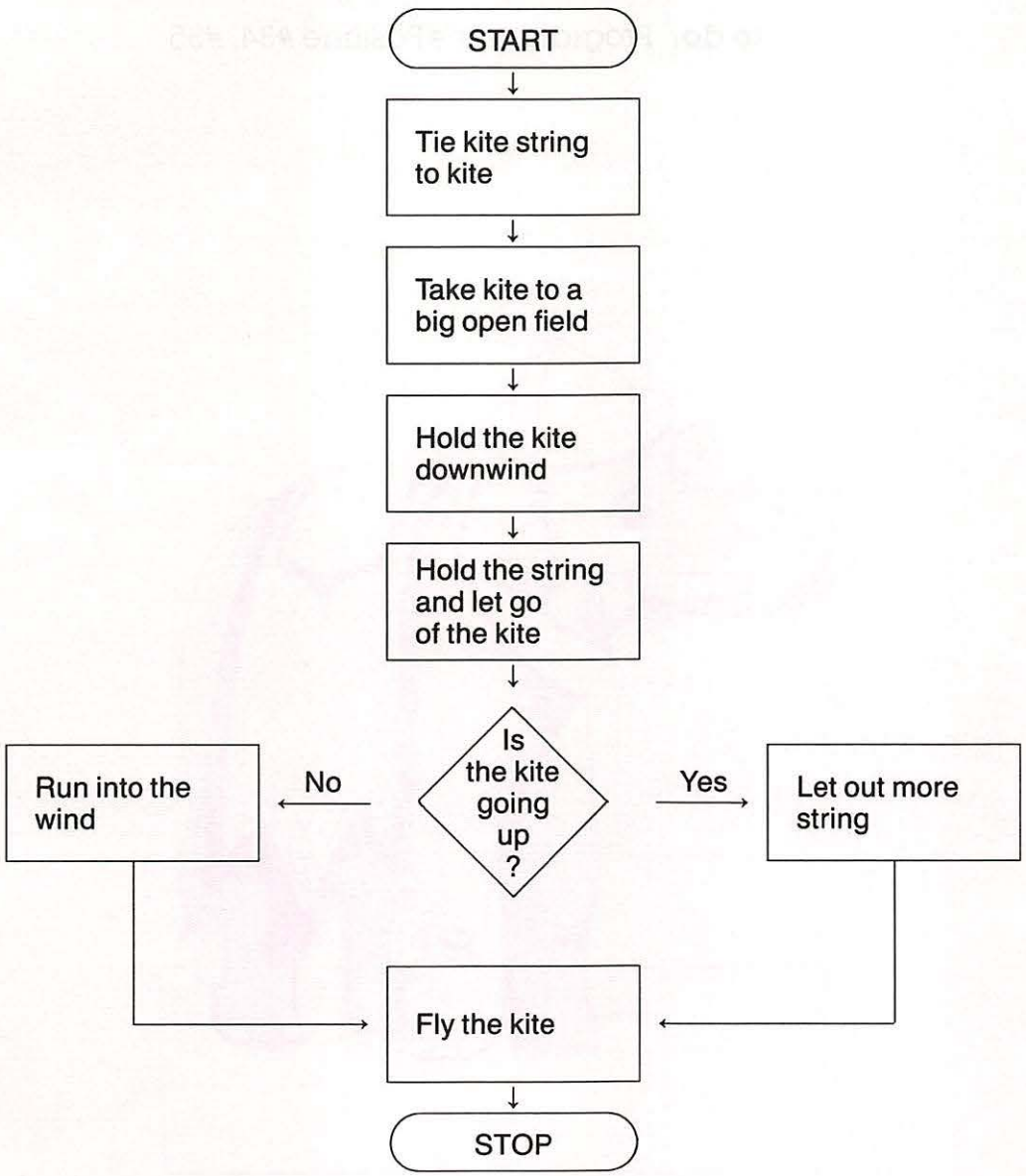
to do: Programmer's Pastime #31, #32, #33



CHAPTER 26 Double Detours

Sometimes a flow chart will have a decision box that has a detour for both the yes and no answers. If the answer is yes, a certain task is done. If the answer is no, a different task is done.

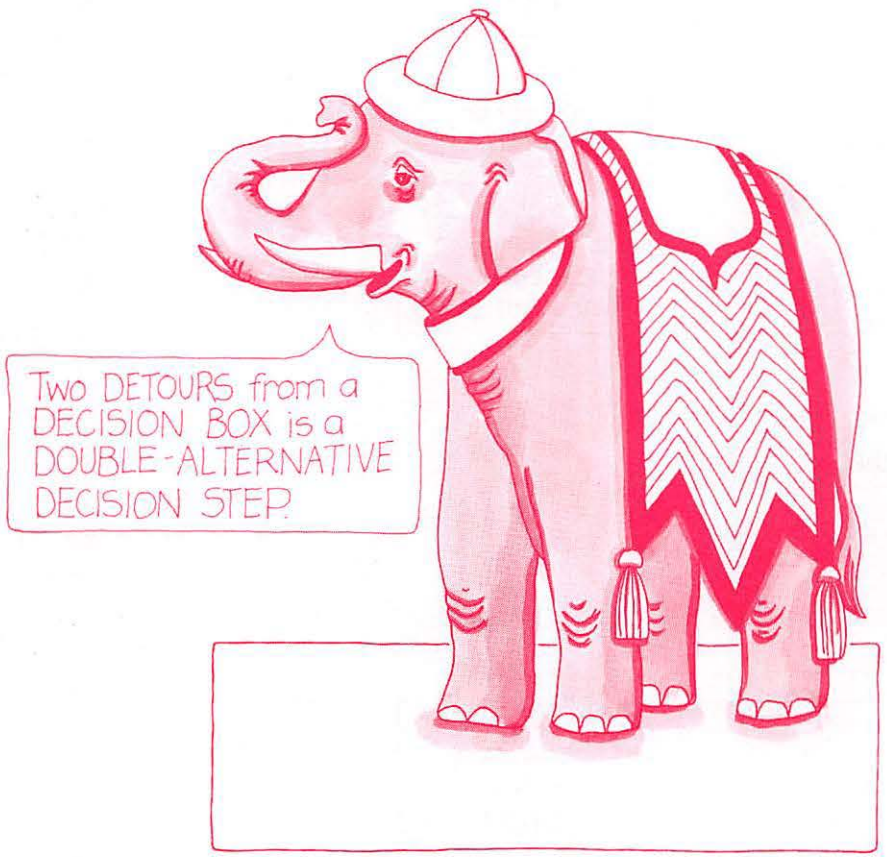
Algorithm/Flow Chart on How to Fly a Kite



This flow chart asks the question, "Is the kite going up in the air?" If the answer is yes, you take a detour that tells you to "Let out more string." If the answer is no, you will take a different detour that tells you to "Run into the wind."

Whenever there are *two* detours from a decision box in a flow chart, the flow chart is said to have a **double-alternative decision step**.

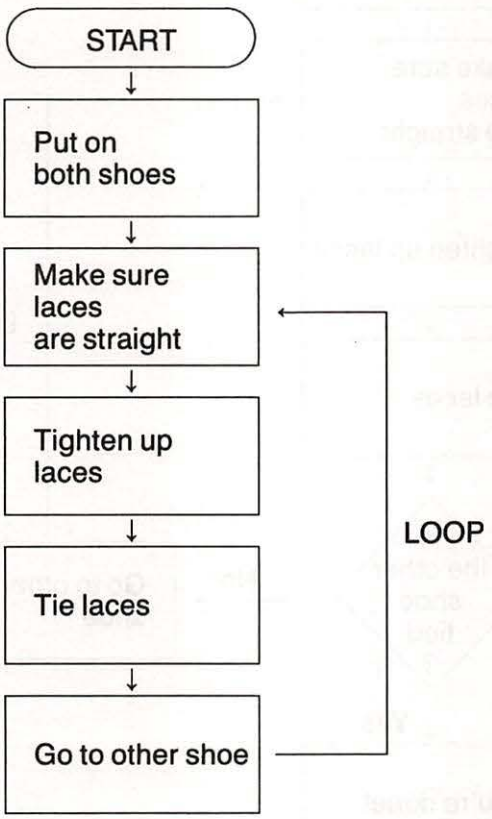
to do: Programmer's Pastime #34, #35



CHAPTER 27 Loop de Loop

Sometimes you will need to use an algorithm that repeats a certain step over and over. When you make a flow chart for such an algorithm, use a **loop** arrow to show that the step is repeated.

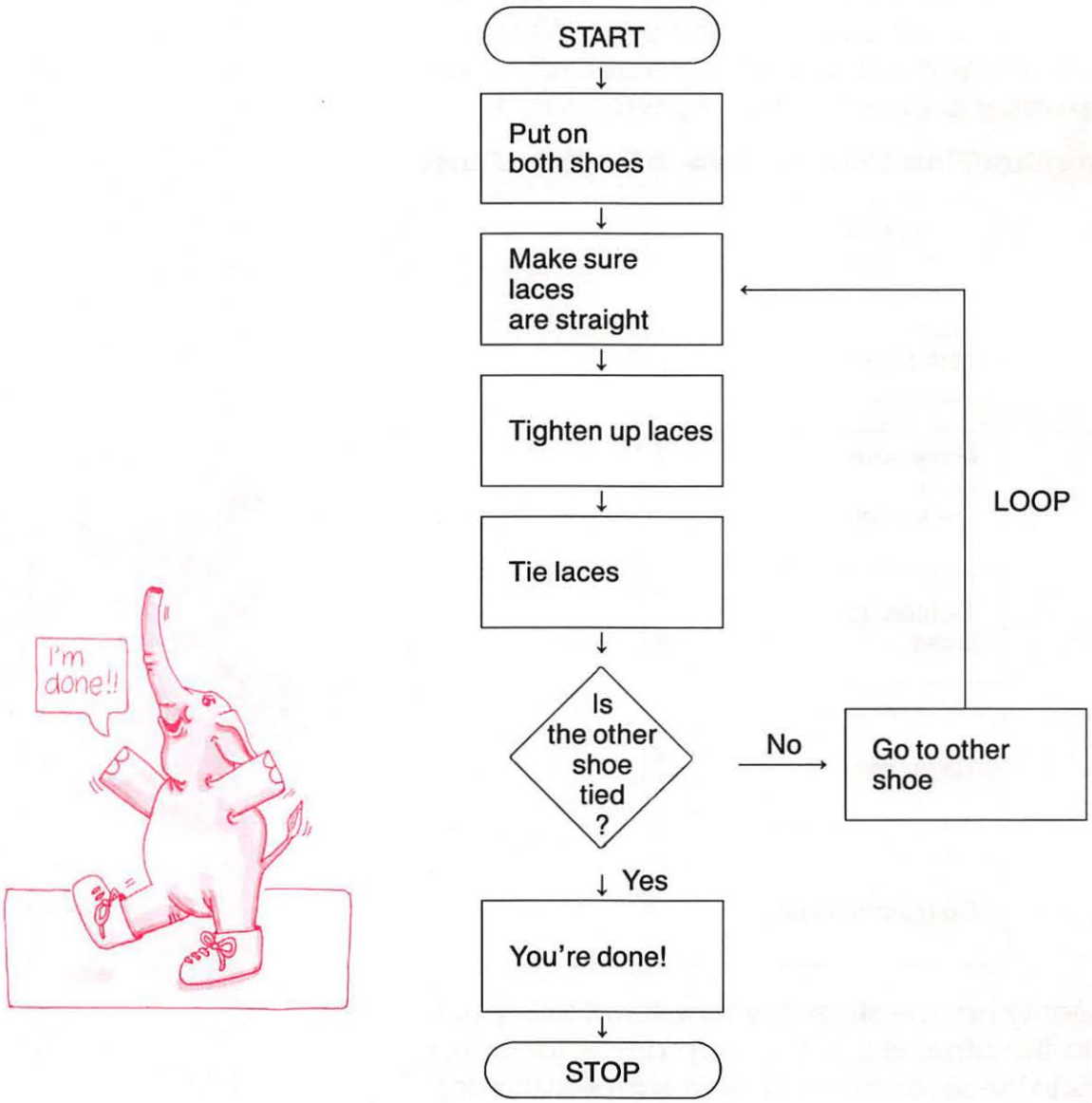
Algorithm/Flow Chart on How to Tie Your Shoes



After tying one shoe, the flow chart tells you to go to the other shoe. The loop arrow takes you back to the second step. Now you repeat the steps as you tie the other shoe. The problem with this flow chart is it will never end! You are told over and over to keep going back to the other shoe to retie it!

Looping is handy because it helps to keep the flow chart short. Imagine how long this flow chart would be if a loop wasn't used.

Looping also works nicely with a decision step. This flow chart can be improved by using a single-alternative decision step.



Now you go through the flow chart twice. Once to do the first shoe and again to do the other shoe. The first time through the flow chart the answer is no, and you follow the loop detour. The second time through the flow chart the answer is yes, and you are done.

to do: Programmer's Pastime #36, #37

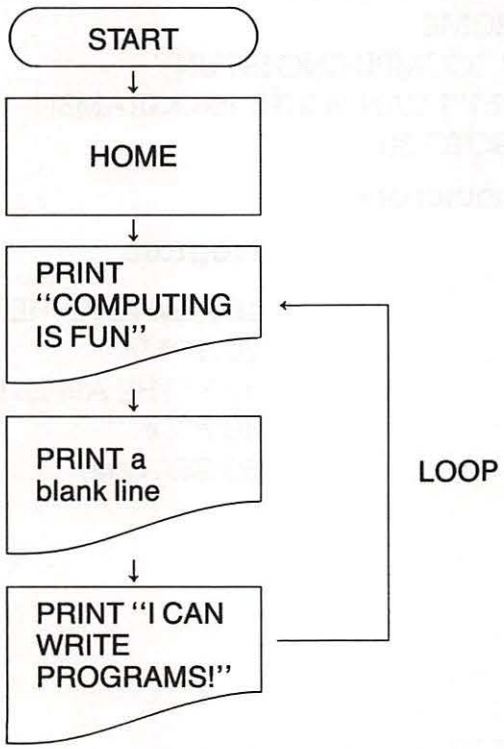
CHAPTER 28 Putting it all Together

Now that you know how to change an algorithm into a flow chart, you must learn how to change a flow chart into a program that the Apple can understand.

Tell the Apple to print over and over:

```
COMPUTING IS FUN
I CAN WRITE PROGRAMS!
```

The algorithm and flow chart will look like this:



Because we want the Apple to print something over and over again we will need to use a loop. Notice that this flow chart never stops; the loop goes on forever.

This is how you would write the flow chart as a BASIC program:

```
10 REM USING A LOOP
20 HOME
30 ? "COMPUTING IS FUN"
40 ?
50 ? "I CAN WRITE PROGRAMS!"
60 GOTO 30
```

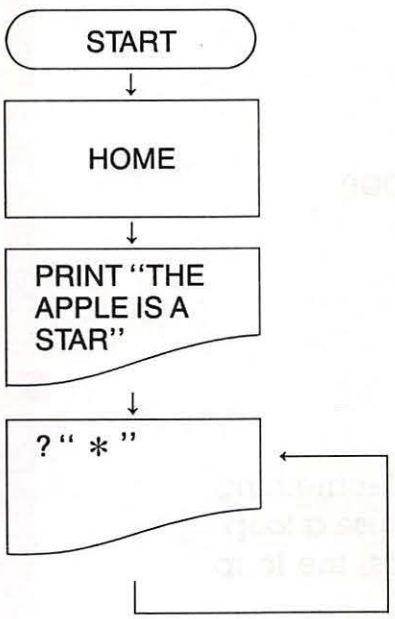
Line 60 is where the loop happens. The command to loop in this program is **GOTO**. After the command GOTO is the number of the line that you want the Apple to go back to.

A shorter way to write the program is:

```
10 REM USING A LOOP
20 HOME
30 ? "COMPUTING IS FUN"
   :?:?"I CAN WRITE PROGRAMS!"
40 GOTO 30
```

Try another one:

Flow chart



Program

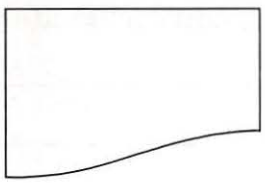
```
10 REM ANOTHER LOOPER
20 HOME
30 ? "THE APPLE IS A STAR"
40 ? " * "
50 GOTO 40
```

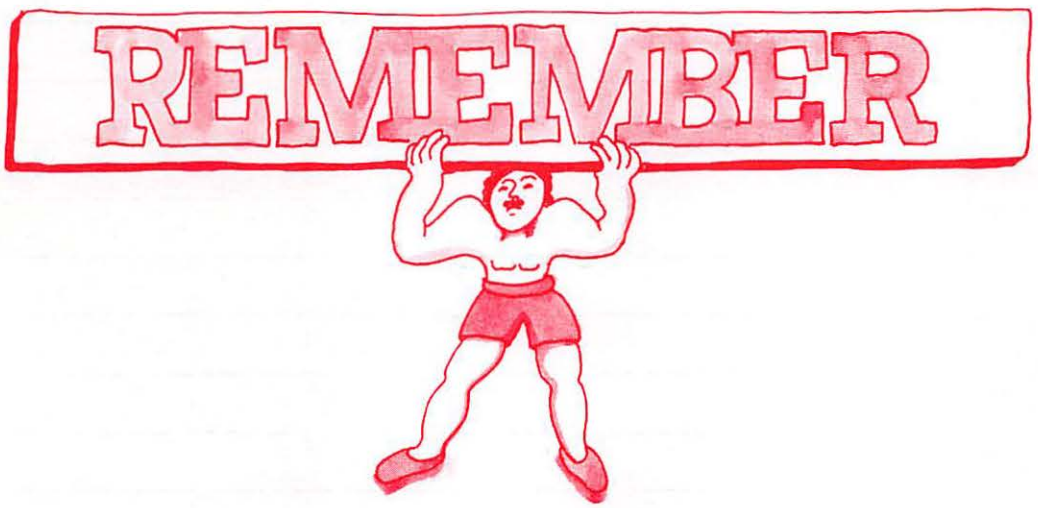
Programs with a GOTO loop will never end once they are run. As long as the Apple is plugged in and is being fed electricity, it will keep doing the GOTO loop over and over and over. To get the Apple to stop a GOTO loop, press CTRL C. The Apple will stop the loop and print:

BREAK IN 40 (or some other number)

The **BREAK** message means that the program was broken into and stopped when the Apple was performing the instruction in line number 40.

The flow charts in this chapter used an instruction step in a box of a different shape. PRINT instructions or statements should be put in a special box. PRINT boxes look like this:



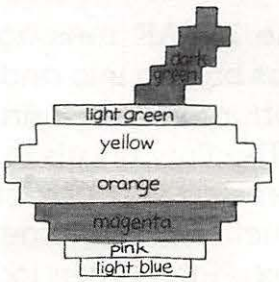


1. GOTO tells the Apple to loop to a certain line in the program.
2. Press CTRL C to stop the run of a program with a GOTO loop.

to do: Programmer's Pastime #38, #39, #40, #41
Component 4 Fun Page

PROGRAMMER'S PASTIME #41

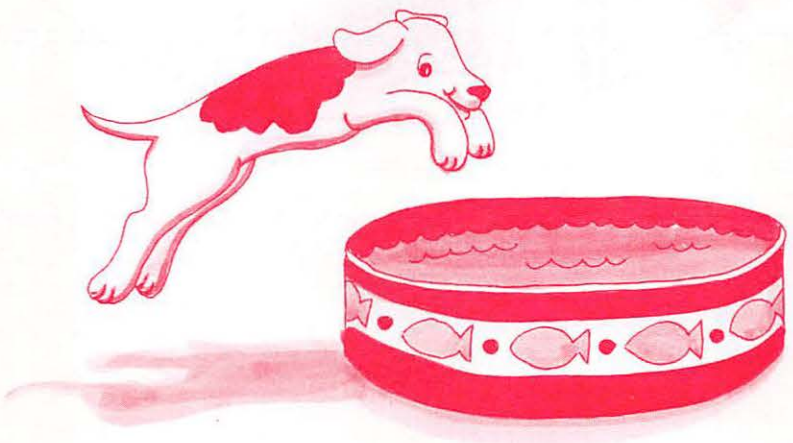
Write a lo res graphics program that will make the Apple draw the Apple logo, clear the screen, draw it again, clear the screen, and so on, over and over. Use the grid on the next page to help you. The logo looks like this:



Write your program on the lines below.

Columns

Rows



COMPONENT 5

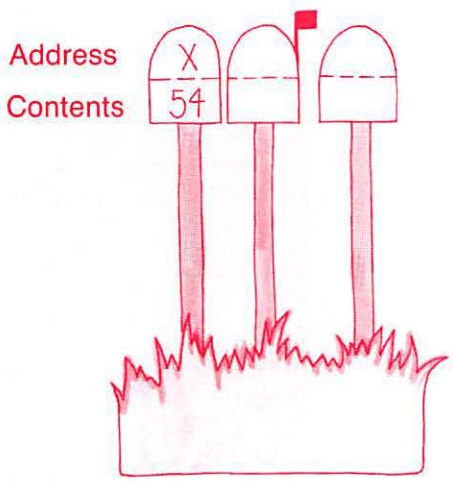
CHAPTER 29 More About Memory

In Chapter 6 you learned that the Apple has a memory. The memory is what makes the Apple powerful. Without a big memory, the Apple wouldn't be much more than the average calculator.

You learned that information "put into" the Apple (called INPUT) is stored in RAM (random access memory). The bigger the RAM, the more input the Apple can hold, and the more it can do. The input is usually made up of programs or **files**. Files are not usually programs, but lists of information that you want the computer to store and use. A file might contain a list of names and addresses of all your friends. The computer could take addresses from the file and print address labels when you write letters to your friends.

How does the Apple store input in its memory (RAM)? Think of the Apple's memory as having thousands of tiny electronic mailboxes. Each mailbox has its own address, and can store information. The information can be a number, letter, word, or even a sentence.

When you write programs in BASIC, it is helpful to store information in the memory mailboxes. When you know where information is being stored in memory, you can refer to it and use it at any time. One way to store information in memory mailboxes is to use the **LET** statement.




```
10 HOME
20 LET X=54
```

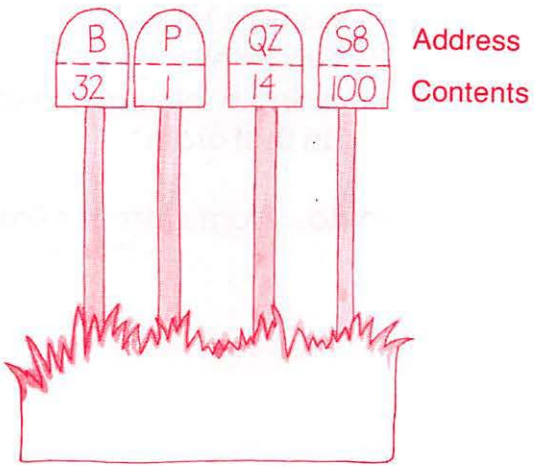
The LET statement tells the Apple to pick an empty mailbox in RAM and call it X. X becomes the **address** of the mailbox.

This LET statement also tells the Apple to put the number 54 inside the mailbox. Thus, 54 becomes the **contents** of the mailbox.

The number 54 is stored safely away in mailbox X. It will stay there until you change it to something different, or erase the memory by typing NEW or turning off the computer.

You can use many different letters or even letters and numbers as the address of a memory mailbox. For example, you can type:

```
10 HOME
20 LET B=32
30 LET P=1
40 LET QZ=14
50 LET S8=100
```



Because the mailbox address can have so many various names and contents, the address is called a **variable**. In the program above, B, P, QZ, and S8 are all variables. Each variable address stores a number as the contents of the mailbox.

To store a letter or word as the contents of a mailbox, you will use a different type of variable address. You'll learn about these variables later.

There are three different ways to write variables that store numbers:

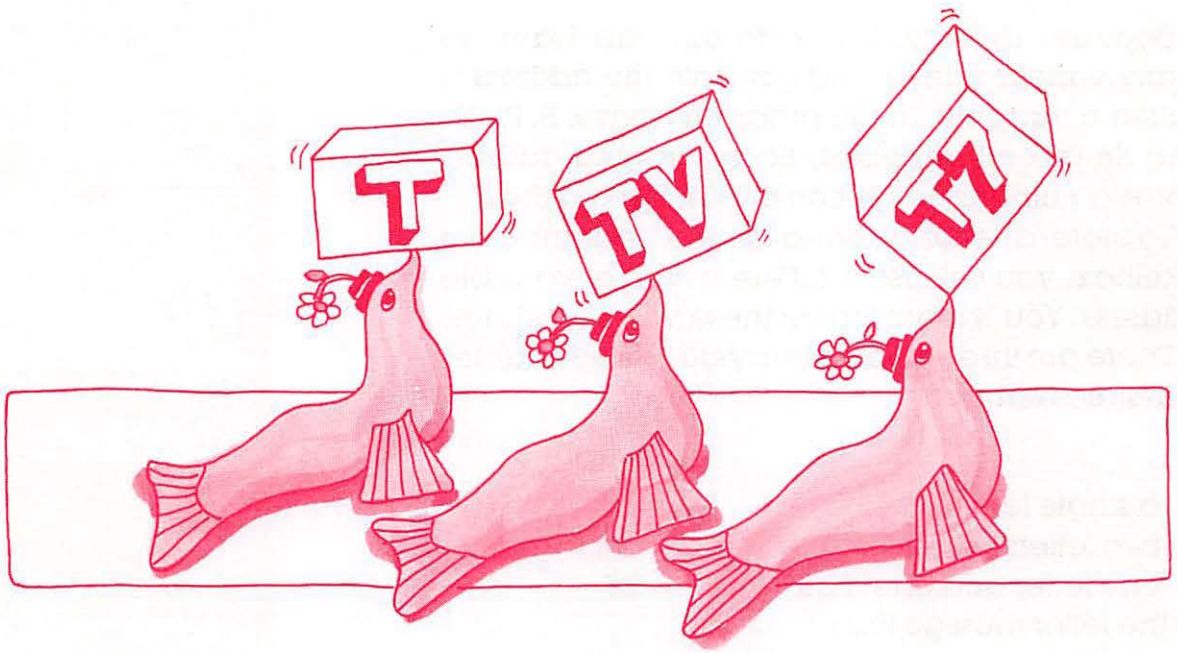
	Example
1. a single letter	Z
2. two letters	AZ
3. one letter and one digit (the letter must go first)	B9

Sometimes you might see variables that have longer names. When the Apple reads a variable, however, it only looks at the first two characters. You could use the word FUN as a variable, but the Apple would read it as FU. If you used two different variables, JK1 and JK2, the Apple would read both as just JK. Using long variable names can become confusing and mess up your program. It is best to use only the three types of variables listed below.

Safe variables to use:

1. a single letter	T
2. two letters	TV
3. a single letter and a single digit (in that order)	T7

to do: Programmer's Pastime #42



CHAPTER 30 Using Variables

Variables are very handy to use in a program. They allow you to store information or **data** and then refer back to it later in the program. For this reason, you will be using variables when you write programs. Since the contents of variables can be easily changed, this is another good reason to use them in programs.

In the program below, two variables are defined. The program refers back to the variables to have their contents printed.

Program

```
10 REM USING VARIABLES
20 HOME
30 LET X=5
40 LET Y=7
50 ? X
60 ?''IS THE CONTENTS OF X''
70 ? Y
80 ?''IS THE CONTENTS OF Y''
90 END
```

What Happens

5 is assigned as the contents of X.
7 is assigned as the contents of Y.
PRINT the contents of X.

PRINT the contents of Y.

Output

```
5
IS THE CONTENTS OF X
7
IS THE CONTENTS OF Y
]
```

If you tell the Apple to ? X (PRINT X), the Apple will print 5 because 5 is the contents of mailbox X.

If you tell the Apple to ? "X" (PRINT "X"), the Apple will print X because X is inside quotation marks.

You can use commas and semi-colons to change how the output will look.

Program

```
10 REM COMMAS & SEMI-COLONS
   WITH VARIABLES
20 HOME
30 LET X=5
40 LET Y=7
50 ? X,
60 ?"IS THE CONTENTS OF X"
70 ? Y;
80 ? "IS THE CONTENTS OF Y"
90 END
```

Output

```
5             IS THE CONTENTS OF X
7 IS THE CONTENTS OF Y
]
```

You can use colons with LET statements the same way you would use them with PRINT statements to shorten a program.

Program

```
10 REM A SHORTCUT
20 HOME
30 LET X=5 : LET Y=7
40 ? X, : ? "IS THE CONTENTS OF X"
50 ? Y; : ? "IS THE CONTENTS OF Y"
60 END
```

Output

```
5             IS THE CONTENTS OF X
7 IS THE CONTENTS OF Y
]
```

You should always try to write your programs so they are as short as possible and easy to read. You should also make sure that the output is easy to read.

Use a blank space inside quotation marks in a PRINT statement when you also use a semi-colon. (ø will mean blank space. Make a blank space by pressing the space bar when you see a ø.) For example:

Program

```
50 ? Y; : ? ''IS THE CONTENTS OF Y''
60 ? Y; : ? ''øIS THE CONTENTS OF Y''
```

Output

```
7  IS THE CONTENTS OF Y
7 IS THE CONTENTS OF Y
```

This last version will make the program short, and both the program and the output will be easy to read.

Program

```
10 REM A BETTER VERSION
20 HOME
30 LET X=5 : LET Y=7
40 ? X; :? ''øIS THE CONTENTS OF X'': ?
50 ? Y; :? ''øIS THE CONTENTS OF Y''
60 END
```

Output

```
5 IS THE CONTENTS OF X
7 IS THE CONTENTS OF Y
]
```

to do: Programmer's Pastime #43, #44

PROGRAMMER'S PASTIME #44

Rewrite each program to make it shorter.

1.
```
10 HOME
20 LET C=10
30 LET D=5
40 ?C
50 ?"IS TWICE AS MUCH AS"
60 ?D
70 END
```

2.
```
10 HOME
20 LET S=1
30 LET T=2
40 LET U=3
50 ?"COUNTING"
60 ?S
70 ?T
80 ?U
90 END
```

3.
```
10 HOME
20 LET V1=15
30 LET V2=30
40 ?V1
50 ?"IS HALF OF"
60 ?V2
70 END
```

CHAPTER 31 Using Variables in Equations

You can use variables in programs to do math equations.

Program

```
10 HOME
20 LET A=5: LET B=6
30 ? A+B
40 END
```

Output

```
 11
]
```

The Apple adds the contents of A to the contents of B and prints the answer.

You can use quotation marks and a semi-colon to make the Apple print the whole equation.

Program

```
10 HOME
20 LET A=5: LET B=6
30 ? "A+B=";A+B
40 END
```

Output

```
A+B=  11
]
```

OR

Program

```
10 HOME
20 LET A=5: LET B=6
30 ? A "+" B "=";A+B
40 END
```

Output

```
5+6=  11
]
```

Using variables in equations can be very helpful, especially if you need to do many equations with the same numbers.

Program

```
10 HOME
20 LET X=3: LET Y=9: LET Z=12
30 ? "X+Y+Z=";X+Y+Z
40 ? "Z-Y-X=";Z-Y-X
50 ? "X*Z/Y=";X*Z/Y
60 END
```

Output

```
X+Y+Z=   24
Z-Y-X=   0
X*Z/Y=4
]
```

To print the equations using the number values instead of the variables, use quotation marks differently.

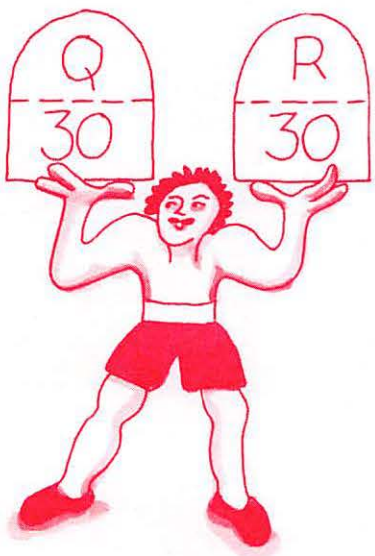
Program

```
10 HOME
20 LET X=3: LET Y=9: LET Z=12
30 ? X"+"Y"+"Z"=";X+Y+Z
40 ? Z"-"Y"-"X"=";Z-Y-X
50 ? X"*"Z"/"Y"=";X*Z/Y
60 END
```

Output

```
3+9+12=   24
12-9-3=   0
3*12/9=   4
]
```

You learned that a variable can have a number value. A variable can also have another variable's value as its value *if* the other variable has already been introduced by a LET statement in the program.




```
10 HOME
20 LET Q=30
30 LET R=Q
```

The contents of R will be the same as the contents of Q.

A variable can also have an equation as its contents.

Program

```
10 HOME
20 LET F=7+8
30 ? F
40 END
```

Output

```
15
]
```

A variable can have an equation *and* a variable as its contents.

Program

```
10 HOME
20 LET W=10
30 LET V=W+5
40 ? V
50 END
```

Output

```
15
]
```

1. The LET statement assigns a value to a variable.
2. ? "X" will print X.
3. ? X will print the value or contents of X.

to do: Programmer's Pastime #45, #46

CHAPTER 32 Important Information

There are some important things to remember about using LET statements.

1. The variable must always come before the value (contents) in the LET statement.

   10 LET S=40 is correct.
   10 LET 40=S is wrong. The Apple will not understand.

2. In a program, you must always put a LET statement *before* the statement that tells the Apple to print the variable.

   10 LET S=40
   20 ? S is correct.

   10 ? S
   20 LET S=40 is wrong. The Apple will print 0.

If the Apple sees a variable in a program that has not been introduced by a LET statement, the Apple will automatically give that variable a value of zero.

In the second program above, line 10 tells the Apple to print the value of S. Since there was no LET statement before line 10 to introduce S, the Apple gives S a value of zero. Even though the next line in the program tells the Apple that S=40, the Apple will still think that the value of S is zero because the PRINT statement comes before the LET statement.

Program

```
10 HOME
20 LET U=10 : LET V=20
30 ? U+V
```

is correct.

Output

```
30
]
```

Program

```
10 HOME
20 ? U+V
30 LET U=10 : LET V=20
```

is wrong.

Output

```
Ø
]
```

When you introduce the same variable more than once in a program, the Apple will always remember the last thing it was told.

Program

```
10 HOME
20 LET K=1
30 LET K=2
40 ? K
50 END
```

Output

```
2
]
```

This program used a LET K statement two times. The Apple only remembers that K=2 because it was the *last* LET K statement. The order of the statements told the Apple to change the value of K from 1 to 2.

Program

```
10 HOME
20 LET K=1
30 ? K
40 LET K=2
50 ? K
60 END
```

Output

```
1
2
]
```

In this program the Apple printed the first value of K and then the second value.

to do: Programmer's Pastime #47, #48

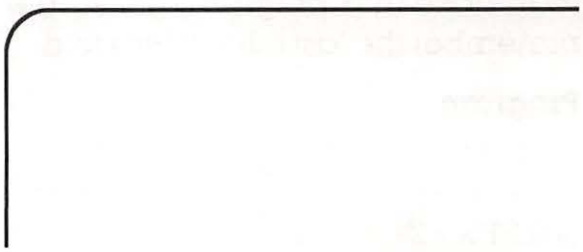
PROGRAMMER'S PASTIME #47

Read each program. Then write what the Apple would print as the output. Check your answers by running the programs.

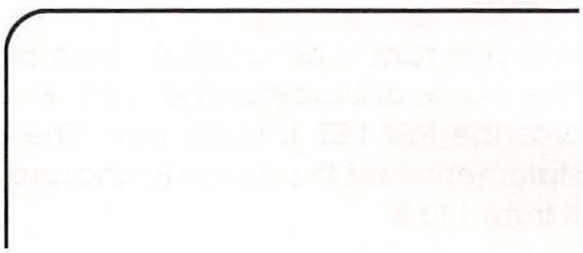
Program

1.
```
10 HOME
20 LET PJ=17: LET J2=34:
   LET J4=PJ+J2
30 ? J4
40 END
```

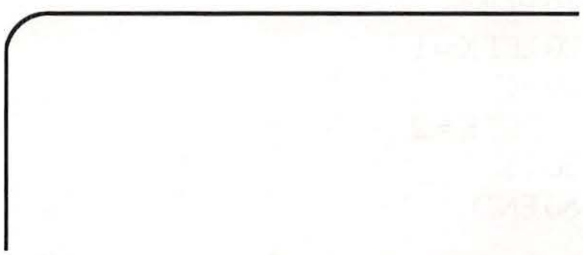
Output



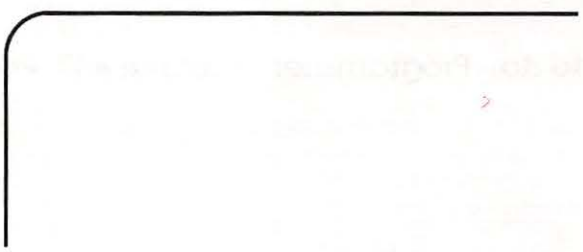
2.
```
10 HOME
20 LET B=2
30 ? B
40 LET B=100
50 ? B
60 END
```



3.
```
10 HOME
20 LET E6=3: LET E7=12
30 ? "PRODUCT", "QUOTIENT"
40 ? E6*E7 , E7/E6
50 END
```



4.
```
10 HOME
20 LET M=16 : LET N=14
30 ? M+N
40 LET N=12
```



CHAPTER 33 Strings

Until now, the variables you have been using in programs have had numbers as their value or contents. For example, X=42.

A variable like X is called a **numeric variable** because its value is a number. You learned that there are three ways to safely use a numeric variable in a program:

1. a single letter — X
2. two letters — XY
3. one letter and one digit — X6

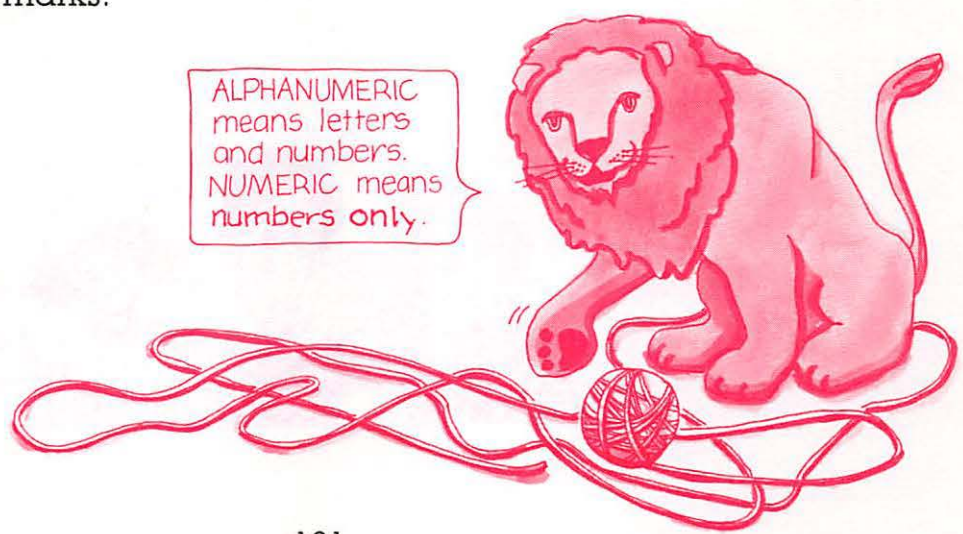
You are now ready to store numbers with letters, words, special characters, and even whole sentences in a variable. This type of variable is called an **alphanumeric** or **string variable**. A string variable can also be written safely in three ways:

1. a single letter followed by a $ — A$
2. two letters followed by a $ — CC$
3. a letter and a digit (in that order) followed by a $ — D7$

You will also introduce a string variable with a LET statement like this:

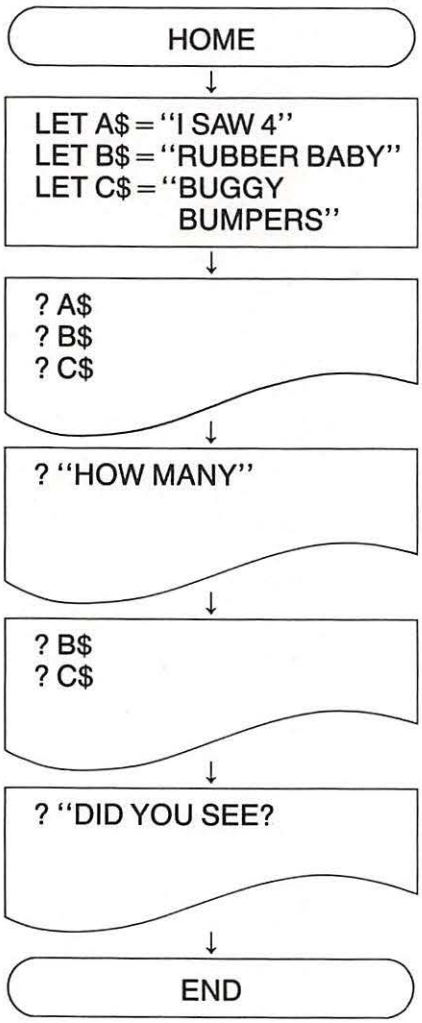
```
10 LET G$ = "HEY YOU!"
```

The contents of a string variable *must* be enclosed in quotation marks.



This program shows how you can use string variables.

Flow Chart



Program

```
10 HOME
20 LET A$ = "I SAW 4"
30 LET B$ = "RUBBER BABY"
40 LET C$ = "BUGGY BUMPERS"
50 ? A$ : ? B$ : ? C$
60 ? "HOW MANY"
70 ? B$ : ?C$
80 ? "DID YOU SEE?"
90 END
```

Output

```
I SAW 4
RUBBER BABY
BUGGY BUMPERS
HOW MANY
RUBBER BABY
BUGGY BUMPERS
DID YOU SEE?
]
```

to do: Programmer's Pastime #49, #50

PROGRAMMER'S PASTIME #50

Each program contains one or more mistake(s). Find the mistake(s), circle the line number where you found the mistake(s), then write the statement the correct way in the space to the right.

Program

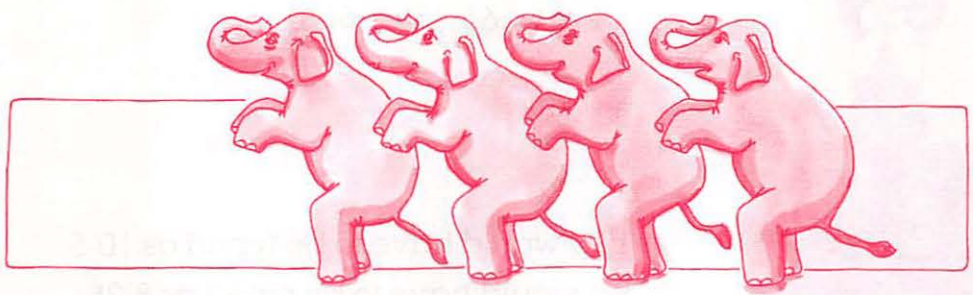
1. 10 LET AZ$ = "YES"
   20 LET BY$ = NO
   30 ? AZ$, BY$
   40 END

2. 10 HOME
   20 LET T$ = "THE TIME"
   30 LET U$ = "IS NOW"
   40 ? T , U
   50 END

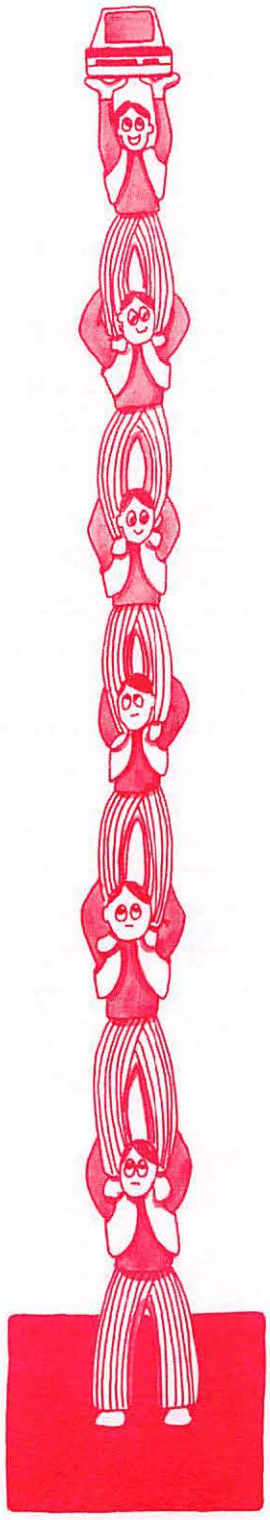
3. 10 HOME
   20 ? J$: ? K$
   30 LET J$ = "UP, UP" :
   LET K$ = "AND AWAY"
   40 END

4. 10 HOME
   20 LET "PARTRIDGE IN" = P$
   30 LET "A PEAR TREE" = T$
   40 ? P$, T$
   50 END

Correction



CHAPTER 34 What Types of Numbers Does the Apple Like?



So far, you have probably asked the Apple to work mainly with **whole numbers** (0, 1, 2, 3 . . .). The Apple can also handle **negative numbers** (−1, −2, −3 . . .).

The Apple can work with **decimals** (0.09, 1.25, etc.) but cannot understand **fractions** (½, ¼). If you need the Apple to do some math that involves fractions, you must change the fractions into their decimal equivalents *or* write them as division equations.

For example:

Change ½ to its decimal equivalent by dividing or just type it as 1/2.

```
   .5
2)1.0 = .5     ½ = .5
  1 0
  ---
    0
```

Change ¼ to its decimal equivalent by dividing or just type it as 1/4.

```
   .25
4)1.00 = .25     ¼ = .25
  8
  --
  20
  20
  --
   0
```

Change ⅔ to its decimal equivalent by dividing or just type it as 2/3.

```
   .66
3)2.00 = .66     ⅔ = .66
  1 8
  ---
  20
  18
  --
   2
```

10½ would have to be typed as 10.5.

8¼ would have to be typed as 8.25.

6⅔ would have to be typed as 6.66.

The Apple lets you use numbers with up to nine digits. If you use a number with more than nine digits, the Apple may not be able to work with it accurately. It may give you a wrong answer. This means that the *smallest* numbers the Apple can use accurately are 0.000000001 and −999999999. The *largest* number the Apple can use accurately is 999999999.

If you want to use numbers that are larger or smaller than this, the Apple will change them into something called **E notation**. This stands for **exponential notation** and it is the same thing as **scientific notation**.

If you want to use a number that has 12 digits, such as 420,000,000,000, the Apple will print it as 4.2E+11. The E+11 means that the decimal point belongs 11 more places to the *right*.

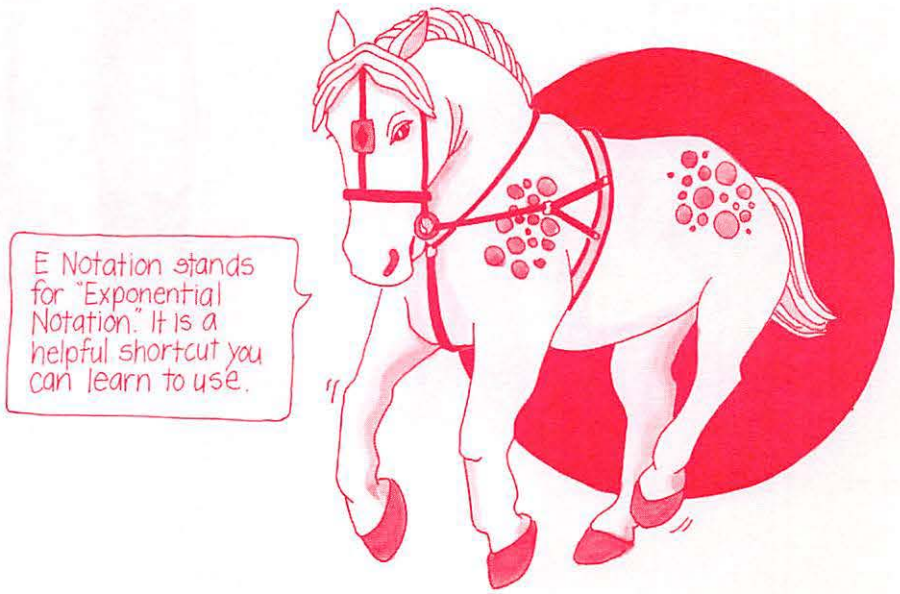
4.2E+11 means 4 . 2 0 0 0 0 0 0 0 0 0 0
1 2 3 4 5 6 7 8 9 10 11

The decimal point should go here.

The number 0.0000009876 would be printed as 9.876E−07. The E−07 means that the decimal point belongs 7 more places to the *left*.

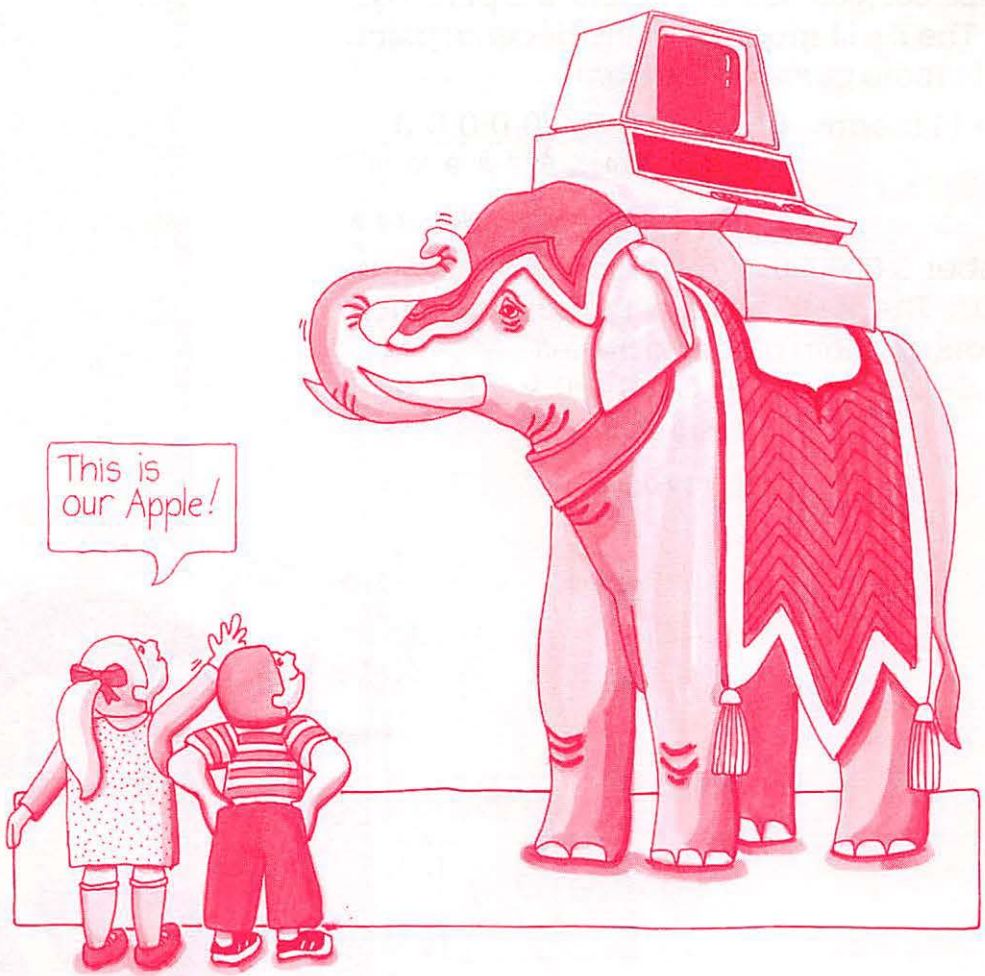
9.876E−07 means 0 0 0 0 0 0 9 . 8 7 6
7 6 5 4 3 2 1

The decimal point should go here.



Don't get worried about E notation because you will only have to use it when you are dealing with numbers that have more than 9 digits.

to do: Programmer's Pastime #51
Component 5 Fun Page

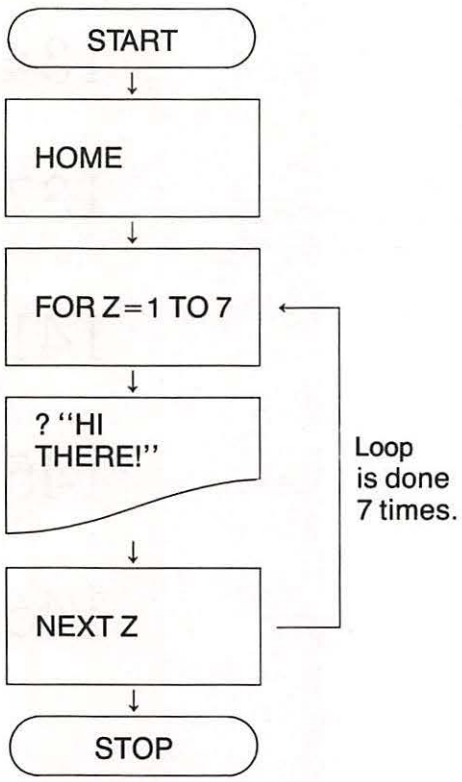


COMPONENT 6

CHAPTER 35 FOR–NEXT Looping

Another type of loop you will use in programming is the **FOR–NEXT** loop. It is used to create **counter-controlled loops** in a program. A counter-controlled loop allows you to repeat program instructions a certain number of times. For example:

Flow Chart



Program

```
10 REM 7 TIMES
20 HOME
30 FOR Z=1 TO 7
40       ?"HI THERE!"
50 NEXT Z
60 END
```

Output

```
HI THERE!
HI THERE!
HI THERE!
HI THERE!
HI THERE!
HI THERE!
HI THERE!
]
```

The FOR–NEXT loop tells the Apple to count to seven and print HI THERE! each time. The loop part of the program is:

```
30 FOR Z=1 TO 7
40      ?"HI THERE!"
50 NEXT Z
```

This loop is done 7 times.

The variable Z does the counting. Its first value is 1. NEXT Z means go back to the beginning of the loop and give Z the next value, which is 2. The Apple keeps doing the loop until Z is 7, and it has printed HI THERE! for the seventh time. Then the loop is over and the Apple goes on to the next program line.

Any statements in between the FOR statement and the NEXT statement are called the body of the loop. These statements are done each time the program loops.

```
30 FOR Z=1 TO 7
40      ?"HI THERE!"      Loop Body
50 NEXT Z
```

Loop

Notice that the body of the loop is always indented. This is good programming style because it makes the loop easier to read and understand. Please practice this when you write programs with FOR–NEXT loops.

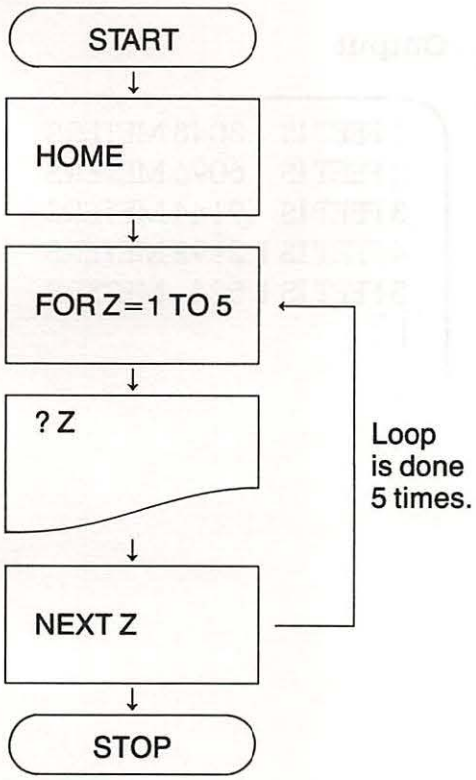
Let's trace the program to see exactly how the FOR-NEXT loop works.

Program Trace

	Program	What Happens	Contents of Z
	10 REM 7 TIMES	REM is ignored.	
	20 HOME	Screen is cleared.	
First Loop	30 FOR Z=1 TO 7	Z is 1.	
	40 ? "HI THERE!"	HI THERE! is printed once.	
	50 NEXT Z	Go back to line 30.	
Second Loop	30 FOR Z=1 TO 7	Z is 2.	
	40 ? "HI THERE!"	HI THERE! is printed again.	
	50 NEXT Z	Go back to line 30.	
Third Loop	30 FOR Z=1 TO 7	Z is 3.	
	40 ? "HI THERE!"	HI THERE! is printed a third time.	
	50 NEXT Z	Go back to line 30.	
Fourth Loop	30 FOR Z=1 TO 7	Z is 4.	
	40 ? "HI THERE!"	HI THERE! is printed a fourth time.	
	50 NEXT Z	Go back to line 30.	
Fifth Loop	30 FOR Z=1 TO 7	Z is 5.	
	40 ? "HI THERE!"	HI THERE! is printed a fifth time.	
	50 NEXT Z	Go back to line 30.	
Sixth Loop	30 FOR Z=1 TO 7	Z is 6.	
	40 ? "HI THERE!"	HI THERE! is printed a sixth time.	
	50 NEXT Z	Go back to line 30.	
Seventh Loop	30 FOR Z=1 TO 7	Z is 7.	
	40 ? "HI THERE!"	HI THERE! is printed a seventh time.	
	50 NEXT Z	Since Z is 7, go on to the next program line.	
	60 END	End the program. Put the prompt and cursor back on the screen.	

You can also write the program so Z prints its contents each time the loop is done.

Flow chart



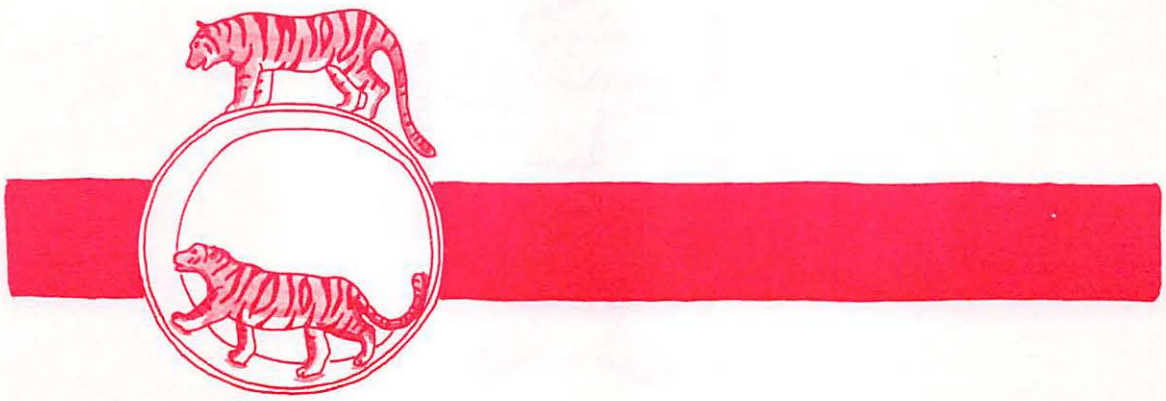
Program

```
10 REM PRINT Z EACH LOOP
20 HOME
30 FOR Z= 1 TO 5
40       ? Z
50 NEXT Z
60 END
```

Output

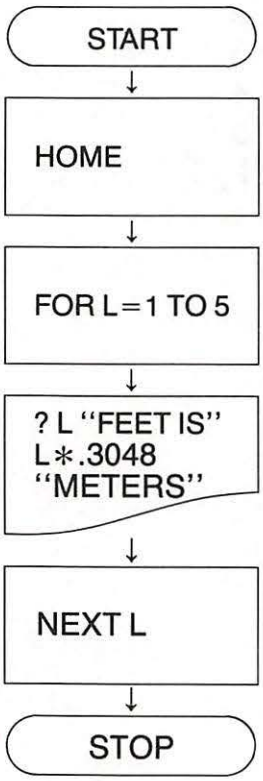
```
1
2
3
4
5
]
```

The variable used in a FOR–NEXT loop can be any kind of numeric variable.



A FOR–NEXT loop is handy to use in a program that **converts** or changes one type of measurement into another. The following program converts feet into meters.

Flow Chart



Program

```
10 REM CONVERT FEET
   INTO METERS
20 HOME
30 FOR L=1 TO 5
40       ?L "FEET IS"
   L*.3048 "METERS"
50 NEXT L
60 END
```

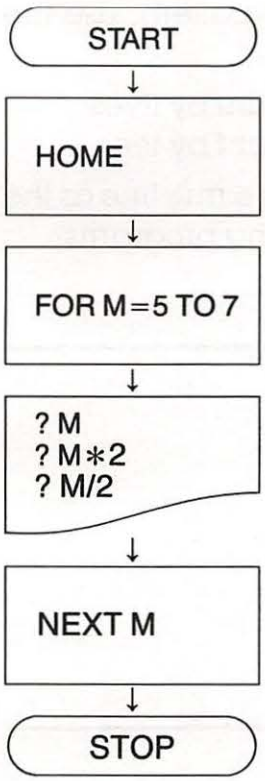
Output

```
 1 FEET IS   .3048 METERS
 2 FEET IS   .6096 METERS
 3 FEET IS   .9144 METERS
 4 FEET IS 1.2192 METERS
 5 FEET IS 1.524  METERS
]
```

Each time the loop is done in this program, the Apple multiplies the current value of L by 0.3048. The current value of L stands for feet and the answer to the multiplication stands for meters.

A FOR–NEXT loop also allows a program to do arithmetic and use a new number each time the loop is done. The loop in the following program causes the numbers 5, 6, and 7 to be printed, multiplied by 2, and divided by 2.

Flow Chart



Program

```
10 REM LOOP ARITHMETIC
20 HOME
30 FOR M=5 TO 7
40      ?M
50      ?M*2
60      ?M/2
70 NEXT M
80 END
```

Output

```
5
10
2.5
6
12
3
7
14
3.5
]
```

In a loop, every FOR statement *must* have a NEXT statement after it somewhere in the program.

to do: Programmer's Pastime #52, #53, #54, #55

CHAPTER 36 Stepping

When you were younger you learned to count in patterns such as: 5, 10, 15, 20. . .(by fives), or: 10, 20, 30, 40. . .(by tens).

The Apple can learn this trick too. If you want the Apple to count in a certain pattern, use the **STEP** statement. For example:

STEP 5 tells the Apple to count by fives.
STEP 10 tells the Apple to count by tens.

The STEP statement goes on the same line as the FOR statement. Study the following programs:

Program

```
10 REM COUNT BY FIVES
20 HOME
30 FOR W=0 TO 25 STEP 5
40       ? W
50 NEXT W
60 END
```

Output

```
0
5
10
15
20
25
]
```

```
10 REM COUNT BY FIVES
20 HOME
30 FOR W= 1 TO 25 STEP 5
40       ? W
50 NEXT W
60 END
```

```
1
6
11
16
21
]
```

How are the two programs different? If you want the Apple to count by fives, you must make the FOR statement say:

FOR W=0 to 25 STEP 5

When the loop begins, W=0. 5+0=5, so the first number the Apple will print is 0 and the next number is 5.

In the second program, W=1. When the Apple starts printing, 1 will be printed first. Then the Apple adds 5 to 1 and prints 6 as the next number. In this program the Apple is not counting by fives, but is adding 5 to each number beginning with 1. The last number printed was 21. Because 21+5=26, which is more than 25, the Apple won't print 26.

The Apple can also count backwards.

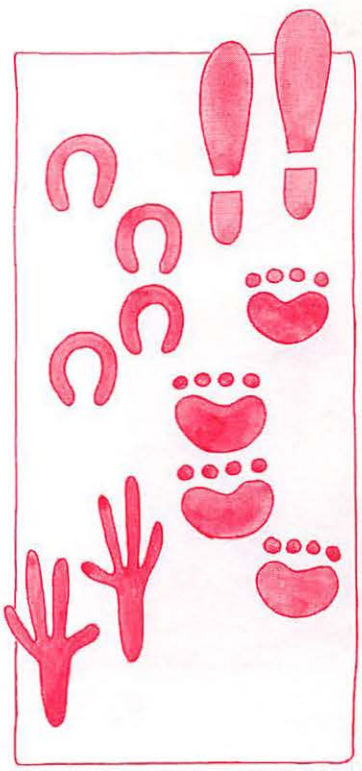
Program

```
10 REM COUNTING BACKWARDS
20 HOME
30 FOR R=5 TO 1 STEP -1
40      ?R
50 NEXT R
60 END
```

Output

```
5
4
3
2
1
]
```

R starts counting at 5. The step of −1 makes R count backwards, subtracting 1 each time.



You can write some fun programs by using the STEP statement.

Program

```
10 REM BLAST OFF
20 HOME
30 ? "STAND BY FOR BLAST OFF"
40 FOR D=5 TO 1 STEP -1
50       ? D; "ØSECONDS"
60 NEXT D
70 ? "BLAST OFF!"
80 END
```

Output

```
STAND BY FOR BLAST OFF
5 SECONDS
4 SECONDS
3 SECONDS
2 SECONDS
1 SECONDS
BLAST OFF!
]
```

Backward stepping can also be used to print words a certain number of times.

Program

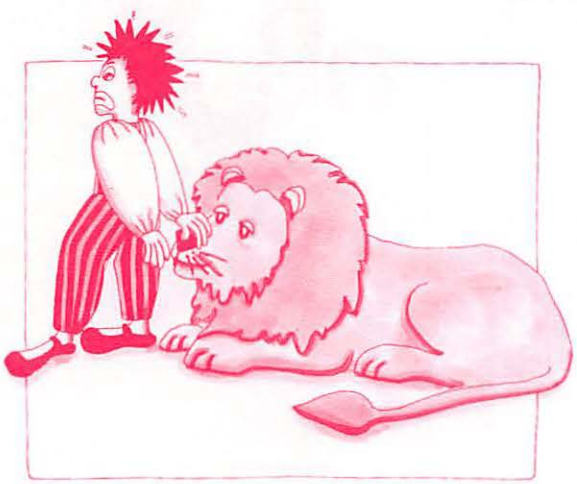
```
10 REM PRINTING WORDS
20 HOME
30 FOR P=20 TO 5 STEP -5
40       ? "GOING BACKWARDS"
50 NEXT P
60 END
```

Output

```
GOING BACKWARDS
GOING BACKWARDS
GOING BACKWARDS
GOING BACKWARDS
]
```

GOING BACKWARDS is printed four times because it takes four runs of the loop to go from 20 to 5 in steps of −5.

to do: Programmer's Pastime #56, #57



CHAPTER 37 A Counter

Sometimes it is handy to use a **counter** in your program to help you keep track of how many times you have done a loop. For example:

Program

```
10 REM LOOPING
20 HOME
30 LET N=0
40 ? "BUZZ OFF"
50 ? N
60 GOTO 20
```

Output

```
BUZZ OFF
0
BUZZ OFF
0
BUZZ OFF
0
.
.
.
```

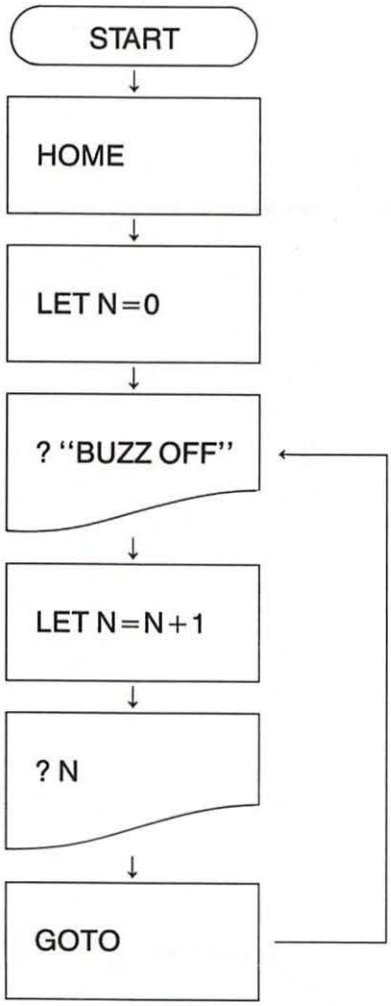
This program has a never-ending loop that prints BUZZ OFF and 0 over and over. If you could get the Apple to print:

```
BUZZ OFF
1
BUZZ OFF
2
BUZZ OFF
3
.
.
.
```

you would know how many times the Apple has done the loop and printed BUZZ OFF. To do this, you must put a counter in the program. The counter is a variable.

In this program, the counter is the variable N.

Flow chart



Program

```
10 REM A COUNTER
20 HOME
30 LET N=0
40 ? "BUZZ OFF"
50 LET N=N+1
60 ? N
70 GOTO 40
```

Output

```
BUZZ OFF
1
BUZZ OFF
2
BUZZ OFF
3
BUZZ OFF
4
.
.
.
```

Program Trace

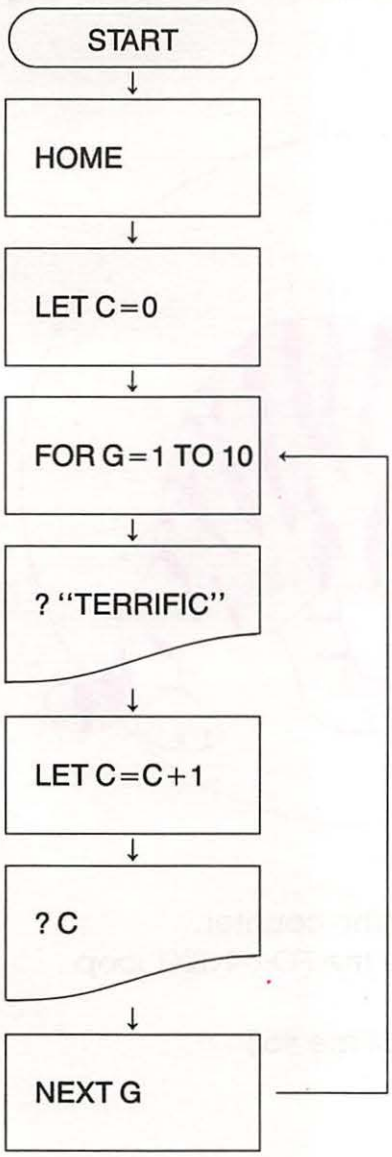
Loop	Line Number	What Happens	Contents of N
1	30	N is introduced as 0.	0
1	40	BUZZ OFF is printed once.	0
1	50	Counter adds 1 to N.	1
1	60	Value of N is printed. (1)	1
1	70	Go to line 40.	1
2	40	BUZZ OFF is printed a second time.	1
2	50	Counter adds 1 to N.	2
2	60	Value of N is printed. (2)	2
2	70	Go to line 40.	2

. . . and so on.

The statement that makes the value of N increase by 1 each time the loop is done is: 50 LET N=N+1. This statement *must* be in the loop body. It is called the counter. After running the program, you will need to press C to stop the run. At the end of the run, you can look at the last number printed and know how many times the Apple has printed BUZZ OFF.

You can also use a counter in a FOR–NEXT loop.

Flow chart



Program

```
10 REM FOR-NEXT
   COUNTER
20 HOME
30 LET C=0
40 FOR G=1 TO 10
50       ?"TERRIFIC"
60       LET C=C+1
70       ?C
80 NEXT G
90 END
```

Output

```
TERRIFIC
1
TERRIFIC
2
TERRIFIC
3
.
.
.
.
.
TERRIFIC
10
]
```

You must be very careful when you use more than one variable in a program. In the previous program, the variable C stands for the counter. The variable G stands for the FOR-NEXT loop. It is important to keep these variables separate so you can better understand what the program is doing.

Another word of warning: some BASIC words are **reserved**. This means you cannot use the first two letters of these words as variables. Some of these BASIC words are: GR, IF, and TO. If you use these commands as variables, the Apple will print a SYNTAX ERROR.

1. Use the variable C for the counter.
2. Use the variable FL for the FOR–NEXT loop.

to do: Programmer's Pastime #58

CHAPTER 38 Timing It

You have learned how to tell the Apple to clear the screen in both direct and program modes by using the HOME command. In your programs, you have used HOME as one of the very first program commands. You can also use HOME in the middle of a program or toward the end. For example:

Program

```
10 REM CLEAR IT TWICE
20 HOME
30 ? "MY NAME IS APPLE"
40 HOME
50 ? "WHAT'S YOURS?"
60 END
```

Output

```
MY NAME IS APPLE

(screen is cleared)
```

```
WHAT'S YOURS?
]
```

When you run this program, you will notice that the Apple writes MY NAME IS APPLE and then clears the screen so fast that you can barely read it. Computers work thousands of times faster than people. This is usually very helpful, but sometimes people want them to slow down a bit.

Use a FOR–NEXT **time loop** in line 35 to use up time and make the Apple wait before going on to the next program instruction in line 40.

Program

```
10 REM MAKE IT WAIT
20 HOME
30 ? "MY NAME IS APPLE"
35 FOR TL = 1 TO 1000: NEXT TL
40 HOME
50 ? "WHAT'S YOURS?"
```

Output

```
MY NAME IS APPLE
(Apple counts to 1000)
(screen is cleared)
```

```
WHAT'S YOURS?
]
```

The time loop in line 35 makes the program stop running while the Apple counts to 1000. When the Apple has finished counting, the program continues.

The colon shortcut is used to write a FOR–NEXT time loop. The colon separates FOR from NEXT so the time loop can be written on one program line. In a FOR–NEXT time loop, there is no loop body.

If you want the Apple to wait longer, change 1000 in the time loop to a larger number. If you want the Apple to move faster and not wait so long, change 1000 to a smaller number.

Speeding

Besides making the Apple wait in the middle of a program, you can make it print on the screen more slowly. The SPEED statement is used to slow down the printing.

Program

```
10 REM CONTROLLING THE SPEED
20 HOME
30 ? "NORMAL SPEED"
40 SPEED=3
50 ? "VERY SLOW"
60 SPEED=255
70 END
```

Output

```
NORMAL SPEED
VERY SLOW
]
```

In this program, the message NORMAL SPEED is printed at the Apple's normal fast speed. It looks like both words are printed on the screen at the same time. The SPEED statement in line 40 changes the printing speed and slows it down. When the message in line 50 is printed, each letter is slowly printed on the screen. Line 60 is very important. It puts the printing speed back to normal when the program is over.

The printing speed can be set using any number from 0 to 255. 255 is the normal printing speed; 0 is the slowest.

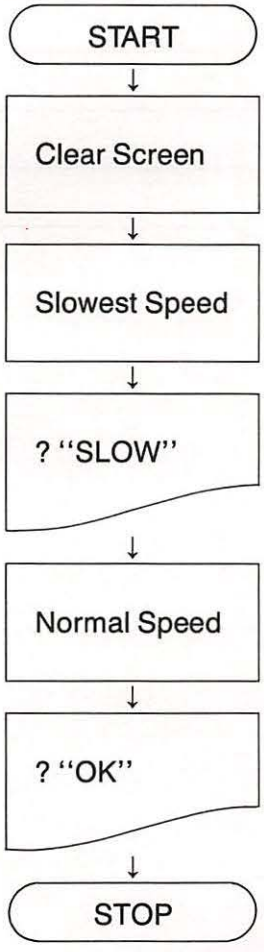
to do: Programmer's Pastime #59, #60

PROGRAMMER'S PASTIME #60

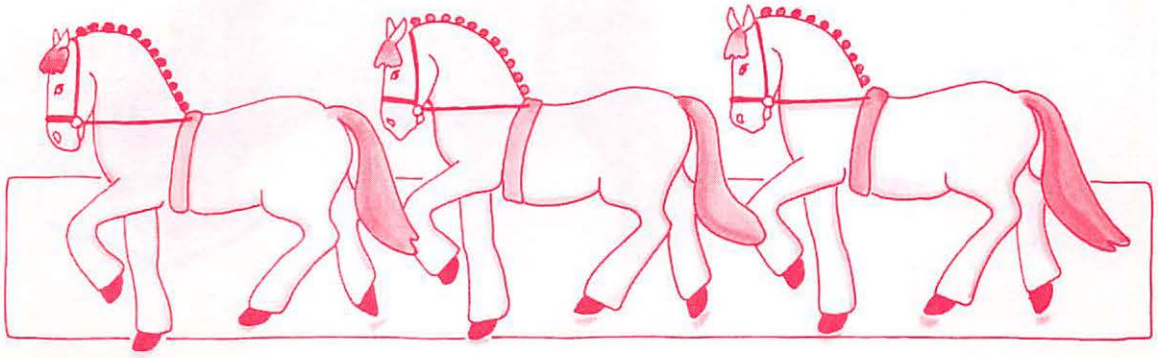
Read each flow chart. Using a SPEED statement, write a program for each flow chart.

Flow chart

1.

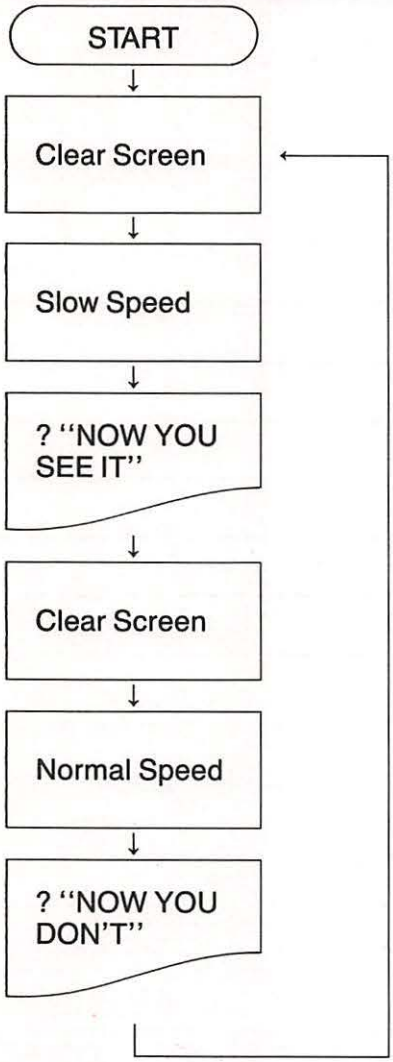


Program

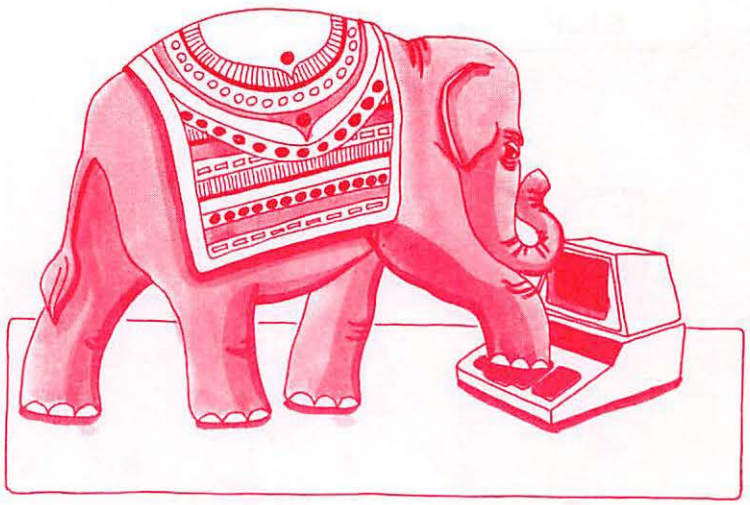


Flow Chart

2.



Program



CHAPTER 39 Blinkers

You can use the FOR–NEXT time loop to make things blink on and off the Apple's screen. For example:

Program

```
10 REM BLINK
20 HOME
30 FOR TL = 1 TO 500: NEXT TL
40 ? "WOW"
50 GOTO 20
```

Output

WOW

The secret to the blinking is in lines 30 and 50. In line 30, FOR TL = 1 TO 500 makes the Apple wait a short time. In line 40, WOW is printed. In line 50, the Apple goes back to line 20 and clears the screen.

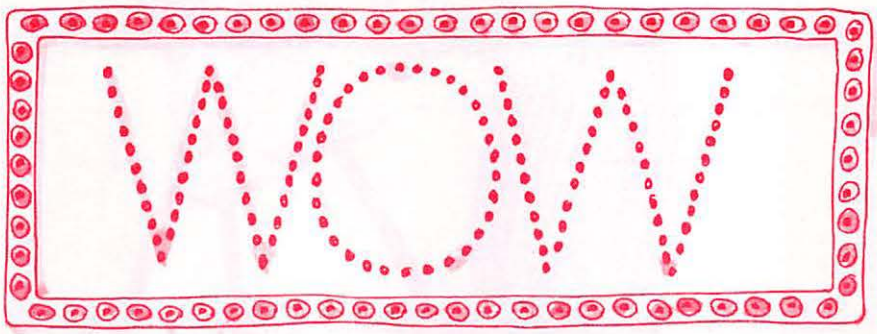
BLINK OFF: lines 20 and 30
BLINK ON: line 40

To make output blink, you must have a FOR–NEXT time loop and a GOTO or FOR–NEXT loop.

You can make something blink faster by changing 500 to a smaller number. You can make output blink more slowly by changing 500 to a larger number.

You can also make output blink in a lo res graphics program. Try it!

to do: Programmer's Pastime #61, #62



CHAPTER 40 Fast Graphics

FOR–NEXT loops can make programming lo res graphics faster and easier. The program below quickly fills the graphics screen with one color. If you are drawing a screen picture, this could be used as your background.

```
10 REM FILLING THE BACKGROUND
20 GR
30 COLOR=9
40 FOR BG=0 TO 39
50      HLIN 0,39 AT BG
60 NEXT BG
70 END
```

The FOR–NEXT loop draws horizontal lines across the screen until the whole screen is filled with color. The loop variable, BG, stands for the rows. Its value starts at 0 (for row 0), and increases by one each time the loop is done. For example:

	BG
FIRST time through the loop, line 50 reads	HLIN 0,39 AT 0
SECOND time through the loop, line 50 reads	HLIN 0,39 AT 1
THIRD time through the loop, line 50 reads	HLIN 0,39 AT 2

and so on.

The next program uses FOR–NEXT loops in the same way to make grass and sky.

```
10 REM GRASS
20 GR
30 COLOR=12
40 FOR G=18 TO 39
50      HLIN 0,39 AT G
60 NEXT G
70 REM SKY
80 COLOR=6
90 FOR S=0 TO 17
100     HLIN 0,39 AT S
110 NEXT S
120 END
```

Grass is drawn on the screen from row 18 to row 39.

The sky is drawn on the screen from row 0 to row 17.

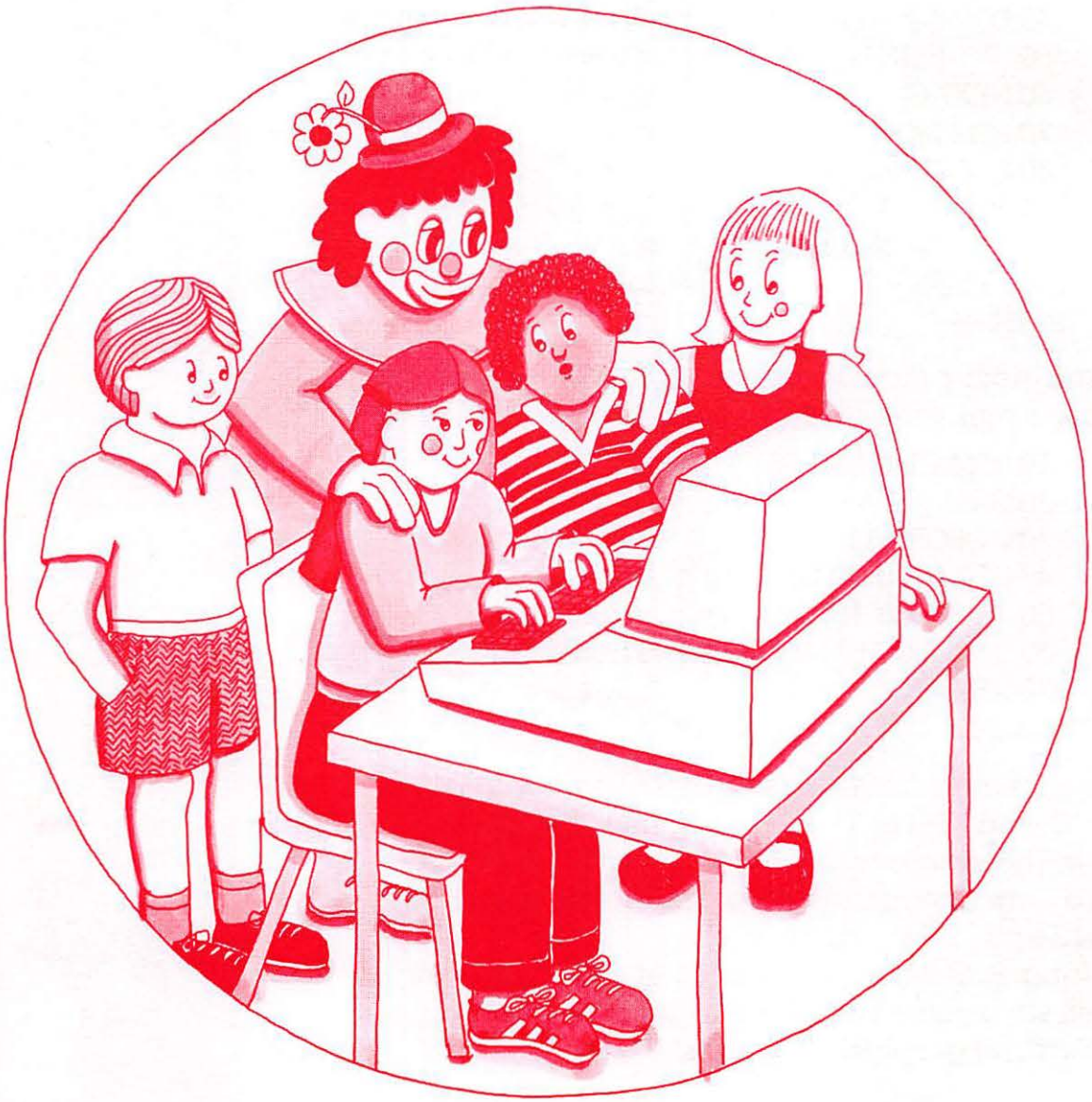
The next program uses a FOR–NEXT loop to draw a pink cross on the screen:

```
10 REM PINK CROSS
20 GR
30 COLOR=11
40 FOR I=10 TO 20
50      PLOT 15,I       (Draws vertical line.)
60      PLOT I,15       (Draws horizontal line.)
70 NEXT I
80 END
```

The first time through the loop, the Apple will PLOT 15,10 and 10,15. The second time through the loop, the Apple will PLOT 15,11 and 11,15. The third time through the loop, the Apple will PLOT 15,12 and 12,15. And so on.

Using FOR–NEXT loops in your programs will often save you a lot of time, especially when you are writing graphics programs!

to do: Programmer's Pastime #63
Component 6 Fun Page



COMPONENT 7

CHAPTER 41 Input

In your dealings with the Apple so far, you have typed programs on the keyboard and then sat back and watched them run. The only way you have given input to the computer is by typing programs on the keyboard.

By using an **INPUT** statement in your program, you can interact with the program *while* it is running. The INPUT statement makes the computer stop the program and ask you for information or data. When the INPUT statement is used, the program becomes an interactive program because the user can now interact with the computer.

Put an INPUT statement in your program at a point where you want the Apple to stop the program and ask for data or information. The Apple will stop the program at the INPUT statement and print a ? and flashing cursor. This means that the computer expects you to type something on the keyboard. What you type will probably be the answer to a question. After you type the input, the Apple will continue running the program.

An INPUT statement usually comes after a question has been asked in the program. For example:

Program

```
10 REM INPUT
20 HOME
30 ? "HOW OLD ARE YOU"
40 INPUT A
```

Output

```
HOW OLD ARE YOU
? □
```

Line 40 is the INPUT statement: 40 INPUT A. A is the variable where the number you answer with is stored.

Program

```
10 REM INPUT
20 HOME
30 ? ''HOW OLD ARE YOU''
40 INPUT A
50 ? ''YOU AREþ '' A ''þYEARS OLD''
60 END
```

Output

```
HOW OLD ARE YOU
? 12                    ← you type
YOU ARE 12 YEARS OLD
]
```

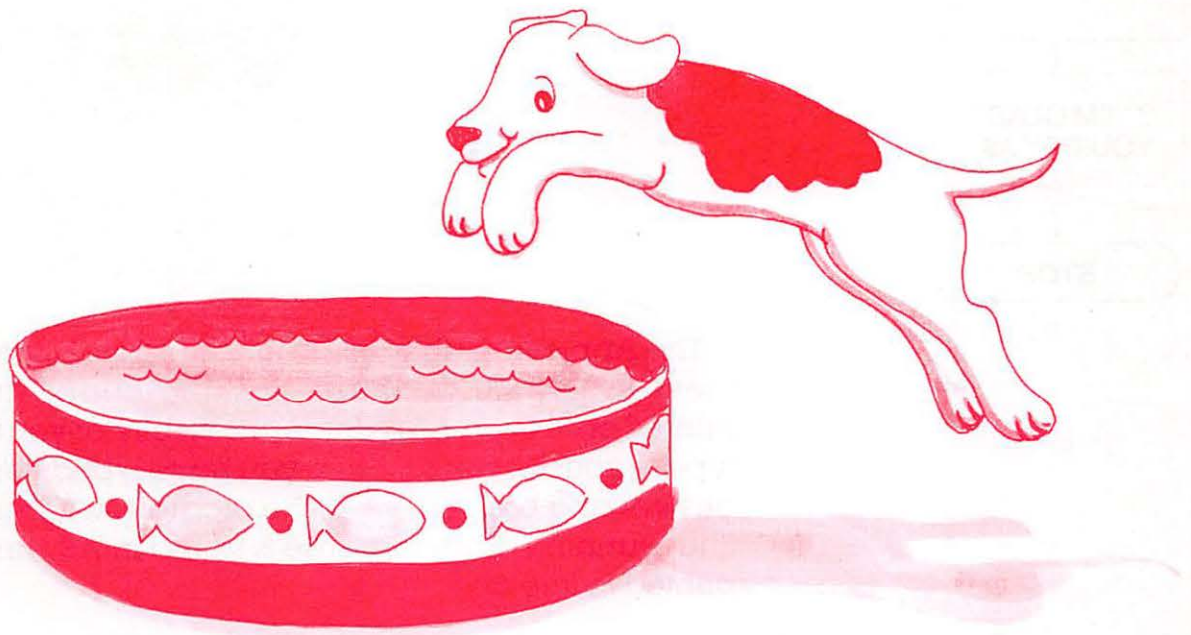
If the answer to the question is to be a word or alphanumeric data, a string variable must be used. For example:

Program

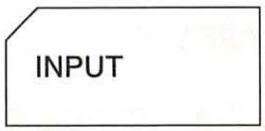
```
10 REM INPUT
20 HOME
30 ? ''WHAT IS YOUR NAME''
40 INPUT N$
50 ? ''HI THEREþ '' N$
60 END
```

Output

```
WHAT IS YOUR NAME
? HARRY                 ← you type.
HI THERE HARRY
]
```

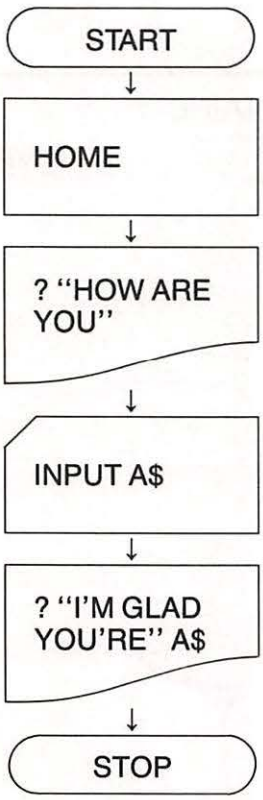


When you make a flow chart for a program with an INPUT statement, you will use a new shape:



Be sure to write the word INPUT inside this box because the box will also be used for another statement, which you will learn about later.

Flow chart



Program

```
10 REM INPUT
20 HOME
30 ? "HOW ARE YOU"
40 INPUT A$
50 ? "I'M GLAD
   YOU'RE" A$
60 END
```

Output

```
HOW ARE YOU
? FINE                    ← you type.
I'M GLAD YOU'RE FINE
]
```

The Apple stops the program at the INPUT statement in line 40. A ? and flashing cursor are printed on the screen as the Apple waits for you to type in your response. Once you have typed your answer and pressed RETURN, the program will continue running. Your response is stored in A$, and printed in line 50.

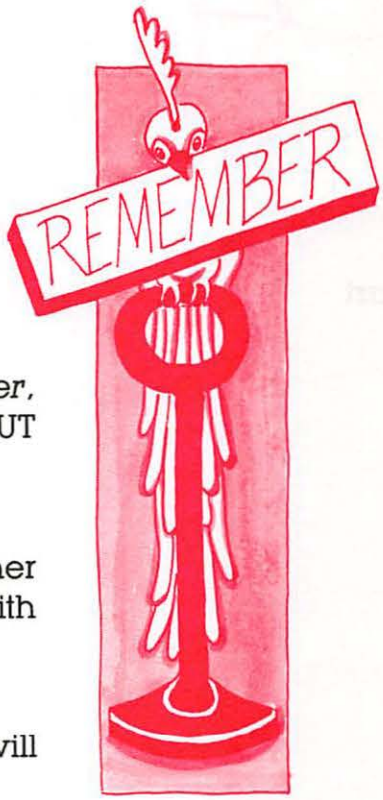
Notice that the question mark is printed on the screen line after the question. There is a trick to make the ? be printed after the question on the same line: use a semi-colon after the question. Remember that a semi-colon holds the cursor after the last thing printed, and then prints the next thing in the very next column.

Program

```
10 REM INPUT WITH ;
20 HOME
30 ? "HOW ARE YOU";
40 INPUT A$
50 ? "YOU'RE" A$
60 END
```

Output

```
HOW ARE YOU? TERRIBLE          ← you type.
YOU'RE TERRIBLE
]
```



1. If the answer (the input) will be a *number*, use a numeric variable with your INPUT statement:

   ```
   INPUT X
   ```

2. If the answer (the input) will be a *word* or other alphanumeric data, use a string variable with your INPUT statement:

   ```
   INPUT X$
   ```

   If you use the wrong variable, the Apple will type:

   ```
   ?REENTER
   ```

3. Put a semi-colon after your question.

   ```
   PRINT "DO YOU LIKE ME";
   ```

to do: Programmer's Pastime #64, #65, #66

PROGRAMMER'S PASTIME #66

It is fun to use INPUT statements in lo res graphics programs. You could have the INPUT statement ask for a color or a location at which to draw something. Read the program below and run it on the Apple. The program draws a pattern on the top half of the screen. Write a program that draws a mirror image of the pattern on the bottom half of the screen.

```
10 REM PATTERN
20 GR
30 FOR J=0 TO 19
40      ?"PICK A NUMBER<1-15>";
50      INPUT N
60      COLOR=N
70      HLIN J,39-J AT J
80 NEXT J
90 END
```

Write your own program here:

Flow chart

Program

This program uses INPUT statements to ask the user where he or she wants to draw a horizontal line and what color the line should be. Run this program on the Apple to see what it does.

```
10 REM DRAW A HORIZONTAL LINE
20 HOME
30 GR
40 ? "WHAT COLOR LINE? CHOOSE A
   NUMBER <1-15>";
50 INPUT C
60 COLOR=C
70 ? "WHICH COLUMN SHOULD THE LINE
   START AT <0-39>";
80 INPUT A
90 ? "AT WHICH COLUMN SHOULD THE LINE
   END <0-39>";
100 INPUT B
110 ? "AT WHICH ROW SHOULD THE LINE BE
   DRAWN <0-39>";
120 INPUT R
130 HLIN A,B AT R
140 GOTO 40
```

Write your own program below that uses INPUT statements to ask the user where to draw a vertical line and in what color.

CHAPTER 42 IF-THEN

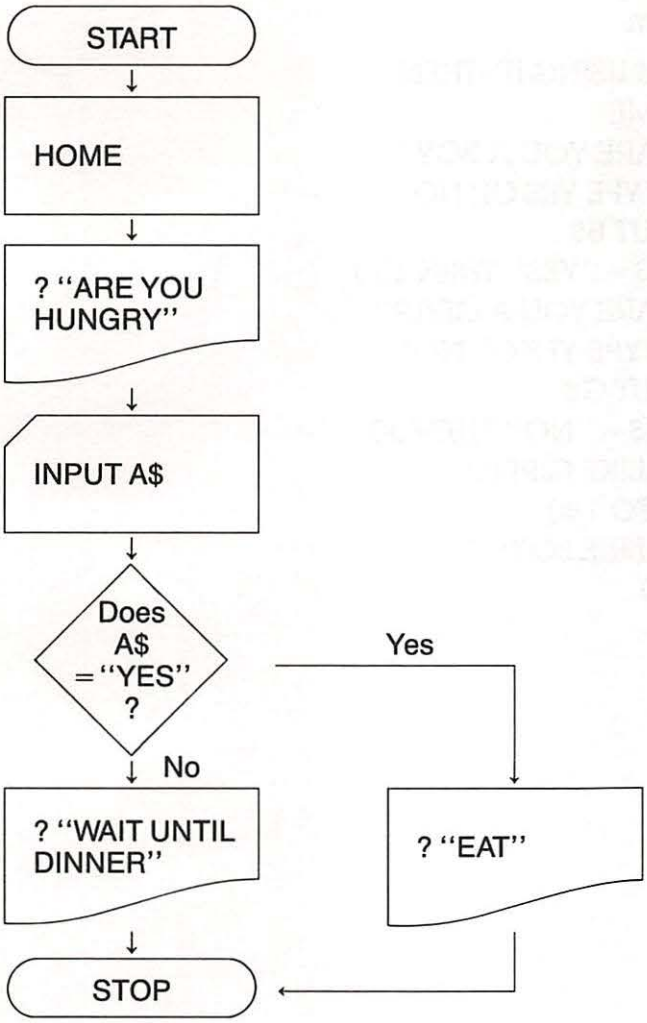
So far you have learned how to program the Apple to print things on the screen and make pictures and designs in lo res graphics. You also know how to make the Apple do math equations. Computers work mainly with numbers. They were invented to do long and tedious arithmetic thousands of times faster than humans can. In this way, computers have saved people countless hours of work.

Computers can do other things as well.

	Example
1. They can compare letters and numbers:	Does X come before Y in the alphabet? Is 97 bigger than 98?
2. They can make a decision and then do the right task:	IF X$ = "YES" THEN PRINT "HELLO"

You now have the skills to set up a flow chart for these types of problems. The flow chart on the next page will have the Apple make a decision and then do the right task based on your input:

Flow chart



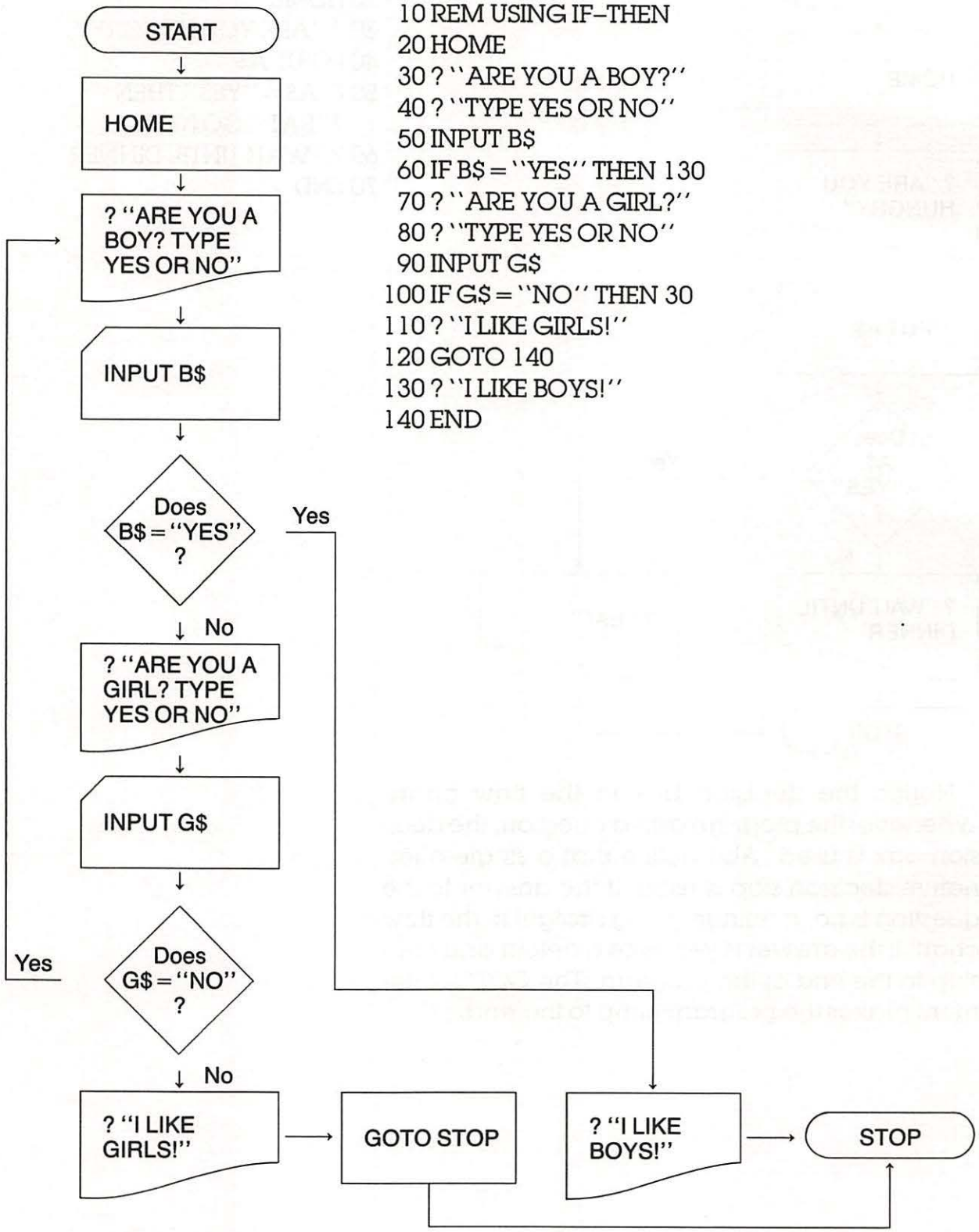
Program

```
10 REM DECISIONS, DECISIONS
20 HOME
30 ? "ARE YOU HUNGRY";
40 INPUT A$
50 IF A$ = "YES" THEN
   ?"EAT":GOTO 70
60 ? "WAIT UNTIL DINNER"
70 END
```

Notice the decision box in the flow chart. Whenever the program asks a question, the decision box is used. Also notice that a single-alternative decision step is used. If the answer to the question is no, continue going straight in the flow chart. If the answer is yes, take a detour and then skip to the end of the program. The GOTO statement makes the program jump to the end.

Here is another example:

Flow chart



Program

```
 10 REM USING IF-THEN
 20 HOME
 30 ? "ARE YOU A BOY?"
 40 ? "TYPE YES OR NO"
 50 INPUT B$
 60 IF B$ = "YES" THEN 130
 70 ? "ARE YOU A GIRL?"
 80 ? "TYPE YES OR NO"
 90 INPUT G$
100 IF G$ = "NO" THEN 30
110 ? "I LIKE GIRLS!"
120 GOTO 140
130 ? "I LIKE BOYS!"
140 END
```

Output

```
ARE YOU A BOY?
TYPE YES OR NO
? YES                          ← you type
I LIKE BOYS!
]
```

```
ARE YOU A BOY?
TYPE YES OR NO
? NO                           ← you type
ARE YOU A GIRL?
TYPE YES OR NO
? YES                          ← you type
I LIKE GIRLS!
]
```

```
ARE YOU A BOY?
TYPE YES OR NO
? NO                           ← you type
ARE YOU A GIRL?
TYPE YES OR NO
? NO                           ← you type
ARE YOU A BOY?
TYPE YES OR NO   } program goes
?                } back to the
                   beginning
```

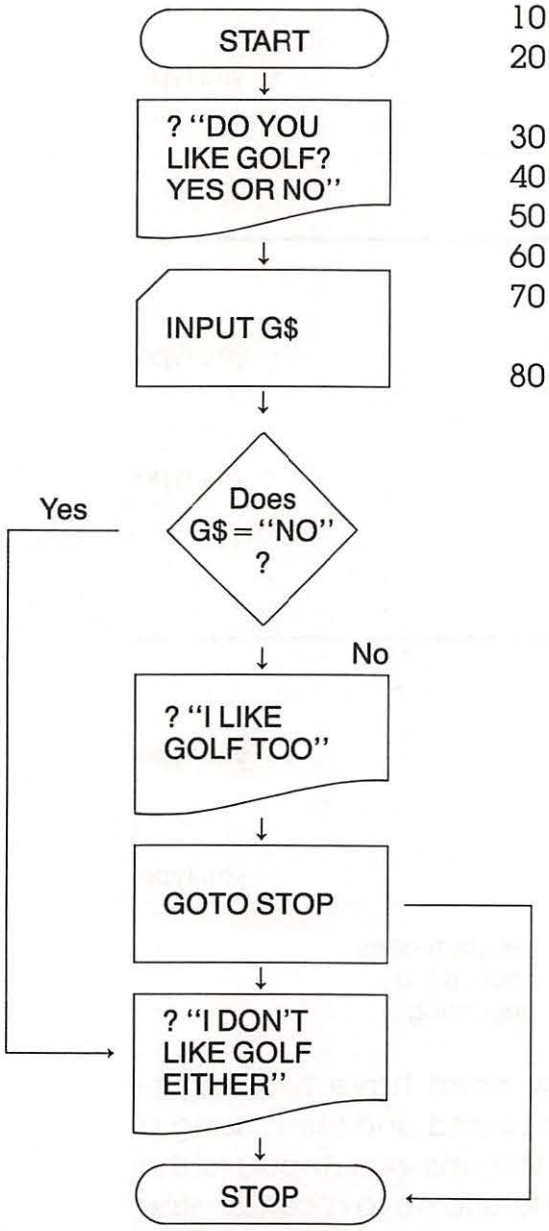
This program and flow chart have two single-alternative decision steps and one jump using a GOTO. Depending on the answers (input) to the questions, the Apple is told to go to a certain line in the program in the IF–THEN statement.

IF B$ = "YES" THEN 130 means if B$ is YES, then skip to line 130.
IF B$ is NO, then go on to the next program line.

IF G$ = "NO" THEN 30 means if G$ is NO, then go back to line 30.
IF G$ is YES, then go on to the next program line.

Here is another example:

Flow chart



Program

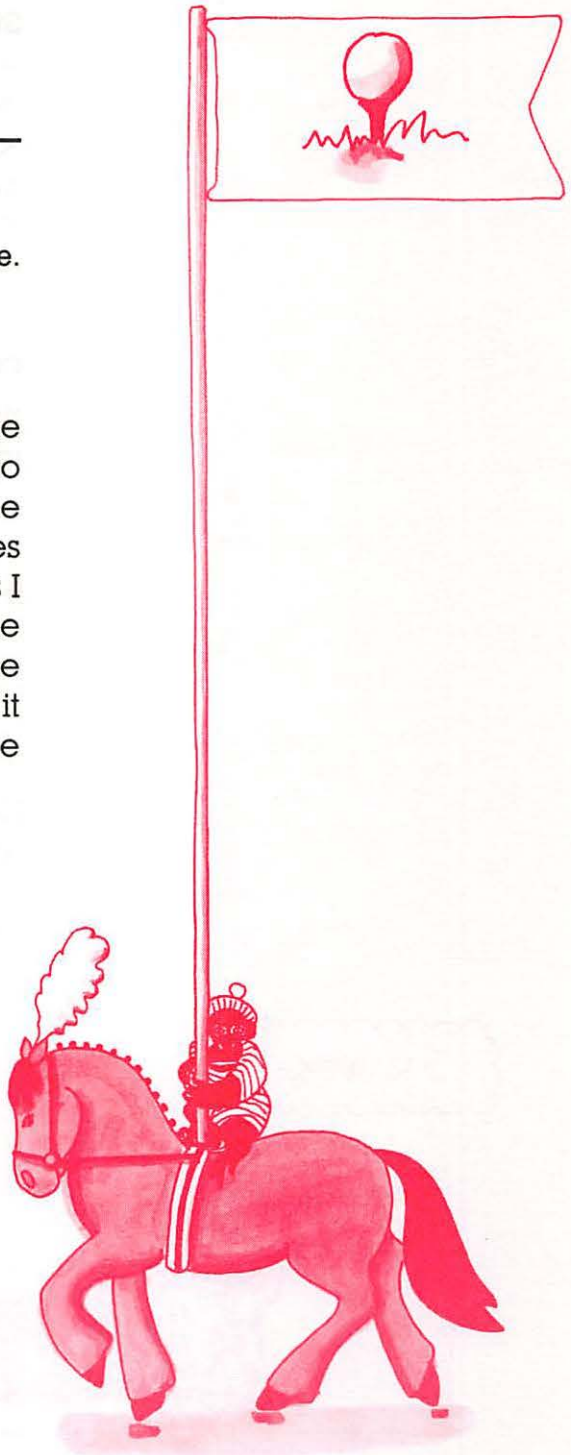
```
10 REM GOLF ANYONE?
20 ? "DO YOU LIKE GOLF?
   YES OR NO"
30 INPUT G$
40 IF G$ = "NO" THEN 70
50 ? "I LIKE GOLF TOO"
60 GOTO 80
70 ? "I DON'T LIKE
   GOLF EITHER"
80 END
```

Output

```
DO YOU LIKE GOLF?
YES OR NO
? YES                    ← you type.
I LIKE GOLF TOO
]
```

```
DO YOU LIKE GOLF?
YES OR NO
? NO                     ← you type.
I DON'T LIKE GOLF EITHER
]
```

If the answer (input) to the question, "Do you like golf?" is no (G$=NO), then the Apple jumps to line 70 and prints I DON'T LIKE GOLF EITHER. If the answer to the question is yes, then the Apple goes on to the next program line (line 50) and prints I LIKE GOLF TOO. The next line (line 60) tells the Apple to jump to line 80 (END); otherwise, the Apple will also print I DON'T LIKE GOLF EITHER if it goes on to line 70. GOTO 80 is needed to make the Apple skip over line 70.



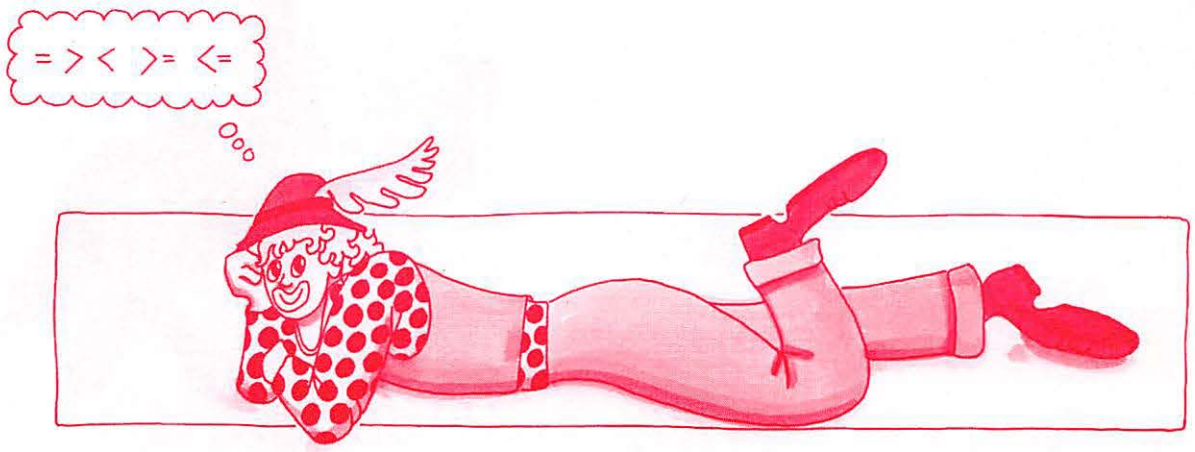
When the Apple sees an IF–THEN statement, it is told to make a comparison. You can use many different signs other than the equal sign when asking the Apple to compare two things. Here is a list of comparison signs and what they mean:

Sign	Meaning	Example
=	equal	4+5=6+3
>	greater than	88>2
<	less than	6<46
>=	greater than or equal to	33>=32
<=	less than or equal to	4<=4
<>	not equal to	65<>800

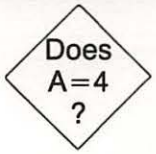
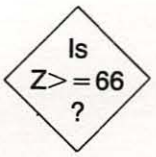
Use the IF–THEN statement when making a comparison in a program. For example:

IF–THEN Statement	Meaning
IF A>B THEN ? A	If the value of A is greater than the value of B, then print A.
IF A$<=S$ THEN 20	If the contents of A$ are less than or equal to the contents of S$, then GOTO line 20 in the program.

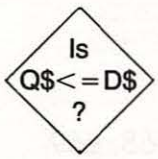
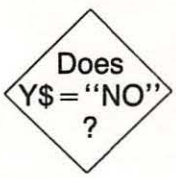
Notice that both IF and THEN are written in the same statement on the same line.



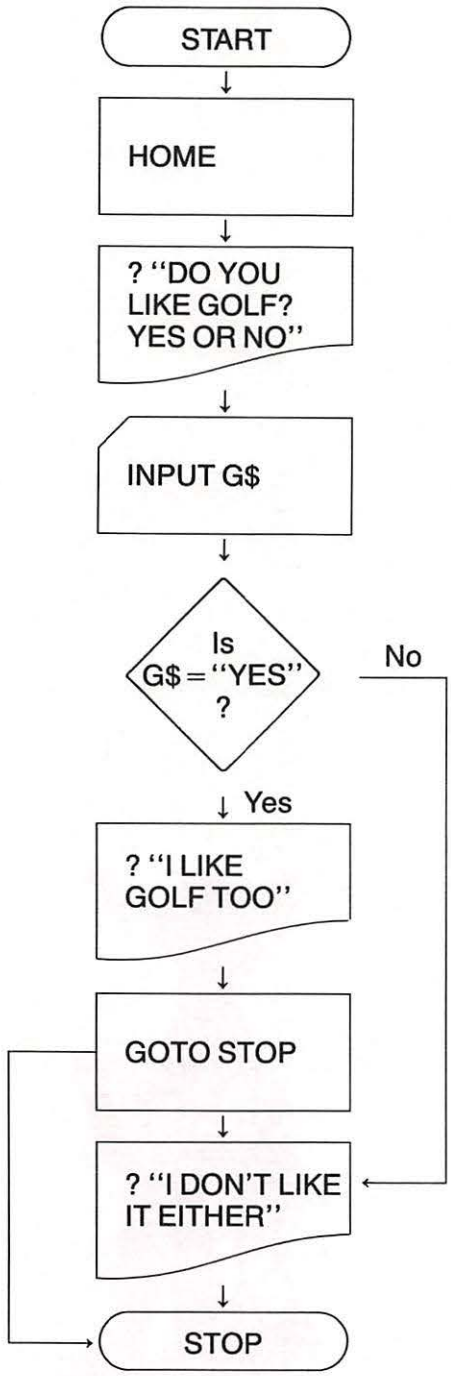
Questions in flow chart decision boxes must be changed into IF-THEN statements for the program.

Flow chart	**Program**
	IF A=4 THEN _____
	IF Z>=66 THEN _____

Sometimes you may have to write the **complement** (opposite) of the flow chart question for the IF–THEN statement. In this case you will use the sign that has the opposite meaning. Example:

Flow chart	**Complement in the program**
	IF E<>F THEN _____
	IF Q$>=D$ THEN _____
	IF Y$ <> "NO" THEN ___

Flow chart



Program

```
10 HOME
20 ? "DO YOU LIKE GOLF?
   YES OR NO"
30 INPUT G$
40 IF G$ < > "YES" THEN 70
50 ? "I LIKE GOLF TOO"
60 GOTO 80
70 ? "I DON'T LIKE IT EITHER"
80 END
```

Output

```
DO YOU LIKE GOLF? YES OR NO
? YES                          ← you type.
I LIKE GOLF TOO
]
```

```
DO YOU LIKE GOLF? YES OR NO
? NO                           ← you type.
I DON'T LIKE GOLF EITHER
]
```

to do: Programmer's Pastime #67, #68, #69

CHAPTER 43 Alphabetizing

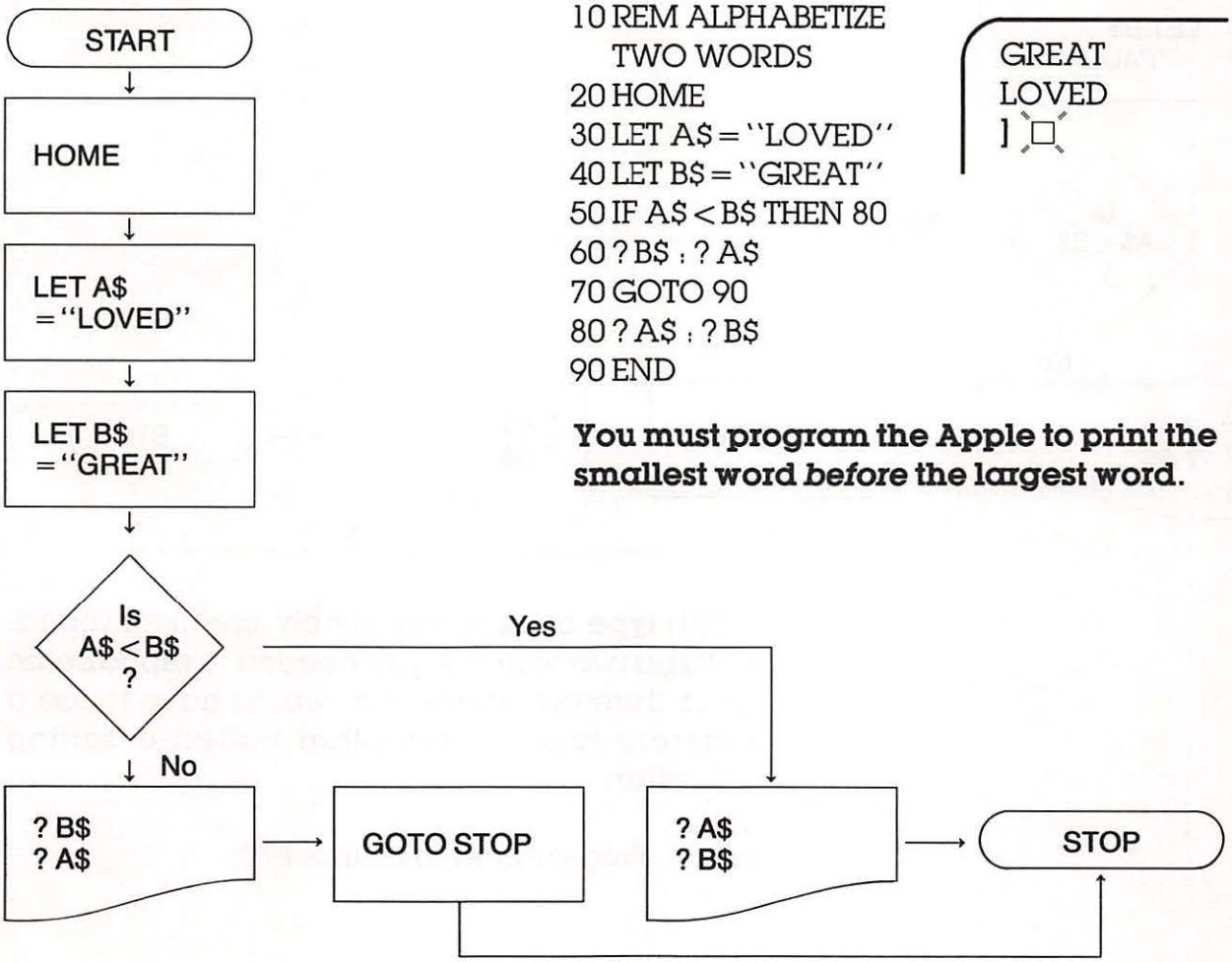
Did you know that the Apple has the ability to compare letters in string variables and alphabetize the words? The Apple already understands that:

A<B<C . . . <Y<Z

A is smaller than B, which is smaller than C, which is smaller than D, and so on, all the way to Z. In other words, A is the smallest letter because it comes first in the alphabet, and Z is the largest letter because it comes last. A word that begins with A is smaller than a word that begins with Z.

Keeping this in mind, you can write a short program to alphabetize two words. For example:

Flow chart



Program

```
10 REM ALPHABETIZE
   TWO WORDS
20 HOME
30 LET A$ = "LOVED"
40 LET B$ = "GREAT"
50 IF A$ < B$ THEN 80
60 ? B$ : ? A$
70 GOTO 90
80 ? A$ : ? B$
90 END
```

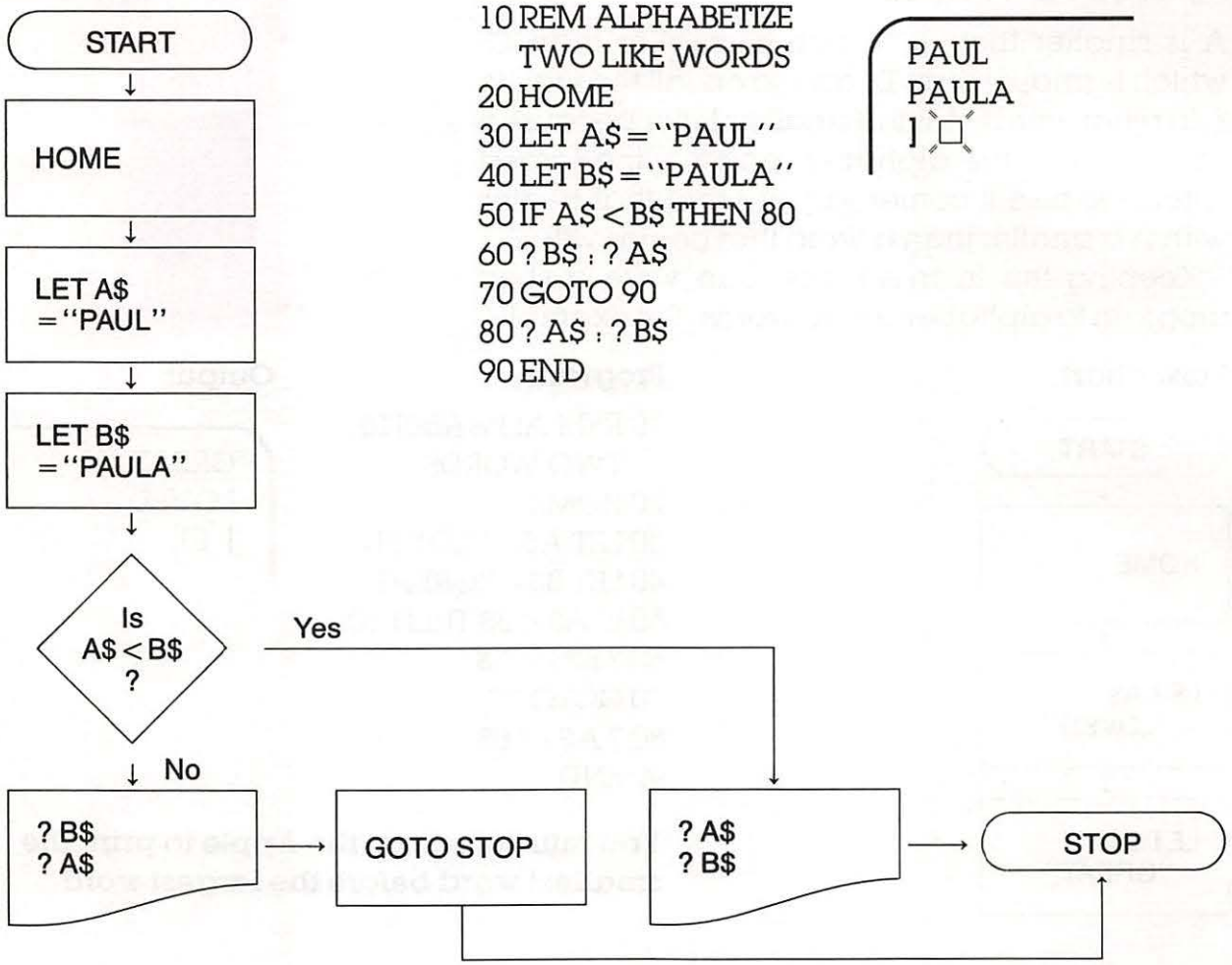
Output

```
GREAT
LOVED
]
```

You must program the Apple to print the smallest word *before* the largest word.

The Apple can also alphabetize words that have the same letters, such as PAUL and PAULA. Both words begin with P-A-U-L, but PAULA has an extra letter at the end. The Apple knows that the rule for this situation is: The shortest word comes first. For example:

Flow chart



Program

```
10 REM ALPHABETIZE
   TWO LIKE WORDS
20 HOME
30 LET A$ = "PAUL"
40 LET B$ = "PAULA"
50 IF A$ < B$ THEN 80
60 ? B$ : ? A$
70 GOTO 90
80 ? A$ : ? B$
90 END
```

Output

```
PAUL
PAULA
]
```

This type of algorithm is only useful for alphabetizing two words. If you needed to alphabetize more than two words, you would have to use a different type of algorithm called a sorting algorithm.

to do: Programmer's Pastime #70

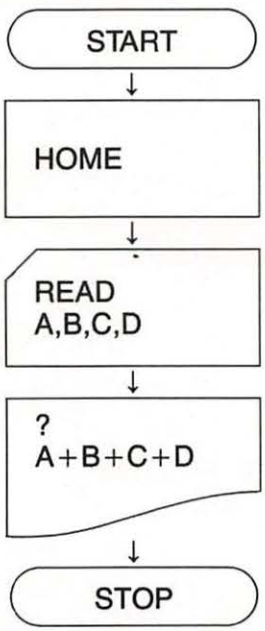
CHAPTER 44 READ–DATA

Another programming trick that can save you and the computer time is the use of **READ–DATA** statements.

The READ statement and the DATA statement go together in a program. These two statements make it possible for you to place data in your program *as* you type it on the keyboard, or even while you are running the program.

This is handy because you can use the same program many times. Instead of writing and typing the program over again for different data, you merely change the information in the DATA statement. The program below adds four numbers:

Flow chart



Program

```
10 REM ADD 4
   NUMBERS
20 HOME
30 READ A,B,C,D
40 DATA 6,7,8,9
50 ? A+B+C+D
60 END
```

Output

```
30
]
```

If you want to use the same program to add four different numbers, just change the DATA statement in line 40. Example:

```
40 DATA 10,11,12,13
```

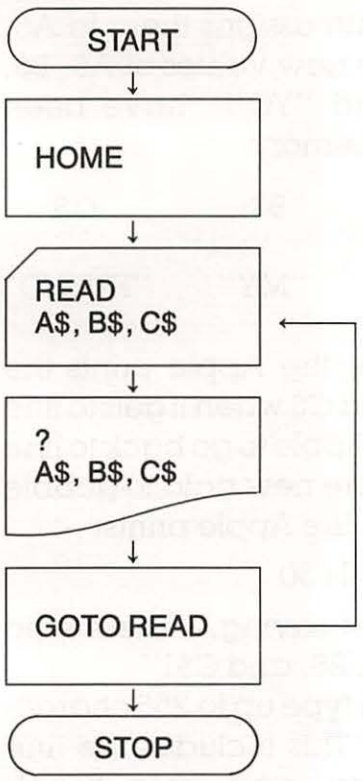
Output

```
46
]
```

The READ box looks just like the INPUT box in a flow chart. You must label the box as READ or INPUT so it is not confusing.

You can also add more data to a DATA statement. Example:

Flow chart



Program

```
10 REM DATA WITH
   WORDS
20 HOME
30 READ A$,B$,C$
40 ? A$, B$, C$
50 GOTO 30
60 DATA "I", "LIKE",
   "YOU", "YOU'RE",
   "MY", "FRIEND"
70 END
```

Output

```
I              LIKE      YOU
YOU'RE         MY        FRIEND

?OUT OF DATA ERROR IN 30
]
```

In line 30 the Apple is told to READ enough data to fill up the three variables A$, B$, and C$. The Apple looks for a DATA statement in the program and finds one in line 60. It "gobbles" up the first three pieces of data it finds ("I", "LIKE", "YOU") and assigns them to A$, B$, and C$.

```
30 READ        A$,        B$,        C$

60 DATA        "I"    ,   "LIKE"    ,   "YOU"
```

You can think of this data as being used up.

In line 40, the Apple prints the contents of A$, B$, and C$. Line 50 tells the Apple to go back to line 30 and read three *new* pieces of data. The Apple finds "YOU'RE", "MY", "FRIEND" in the DATA statement and again assigns them to A$, B$, and C$. These are the new values of A$, B$, and C$. "I", "LIKE", and "YOU" have been erased from the Apple's memory.

```
30 READ          A$,        B$,        C$

60 DATA . . .    "YOU'RE" , "MY" , "FRIEND"
```

The second time through, the Apple prints the new contents of A$, B$, and C$ when it gets to line 40. Again, line 50 tells the Apple to go back to line 30. Because there is no more new data to gobble up in the DATA statement, the Apple prints:

```
?OUT OF DATA ERROR IN 30
```

This is the Apple's way of saying, "There's no more data to read into A$, B$, and C$!"

Remember that you can type up to 255 characters for one line number. This includes the line number, statement, data, and even blank spaces and commas. If you have more data for a DATA statement than 255 characters, you may use more than one DATA statement in your program.

The Apple treats all of the data in a program as one big list. The READ statement has a **pointer** that goes through this data list and gobbles up any new data.

The DATA statement can be placed anywhere in a program. There is only one thing you must look out for. You *must* have the data in your DATA statement in the correct order. For example, if you want the program to print:

```
HI THERE PAL
```

the data "HI" "THERE" and "PAL" must be in this order in the DATA statement. If they are out of order, this is what might happen:

Program

```
10 REM DATA OUT OF ORDER
20 HOME
30 READ L$, M$, N$
40 ? L$, M$, N$
50 DATA "PAL", "THERE", "HI"
60 END
```

Output

```
PAL             THERE           HI
]
```

You must also make sure that the data in the DATA statement is separated by commas. Any data not separated by commas will be lumped together as one piece of data.

If you have three variables to read, the Apple will gobble up data in groups of three. Any left-over pieces of data will not be printed. For example:

Program

```
10 REM LEFTOVERS
20 HOME
30 READ X, Y, Z
40 ? X, Y, Z
50 GOTO 30
60 DATA 2,4,6,8,10,12,13,14
70 END
```

Data used

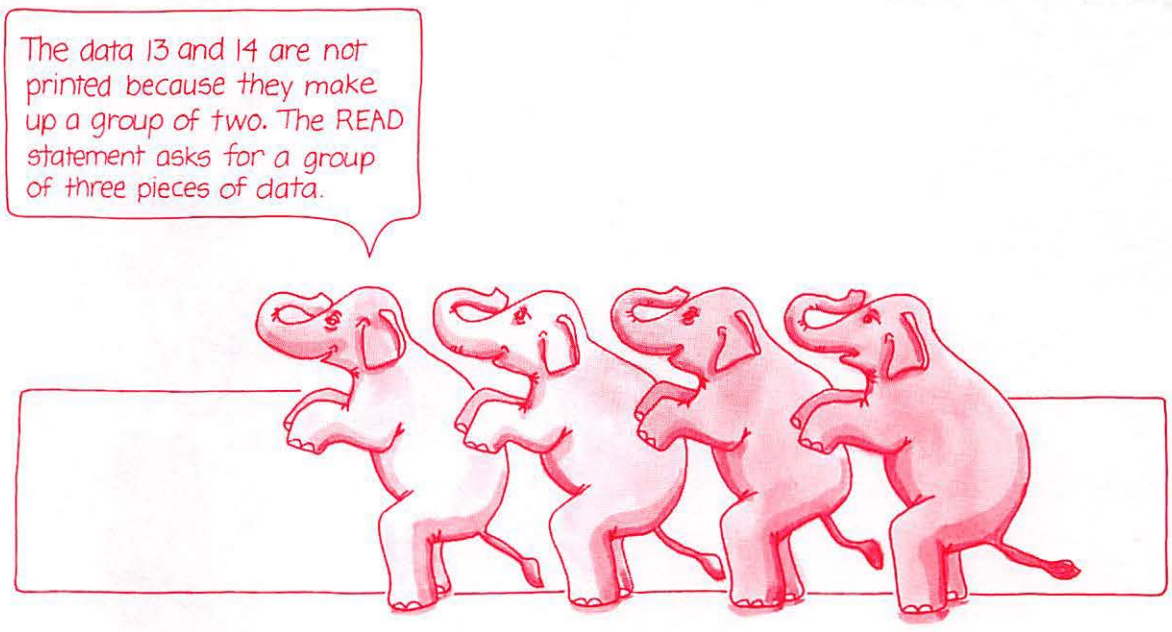
1st time: 2, 4, 6
2nd time: 8, 10, 12
left over: 13, 14

Output

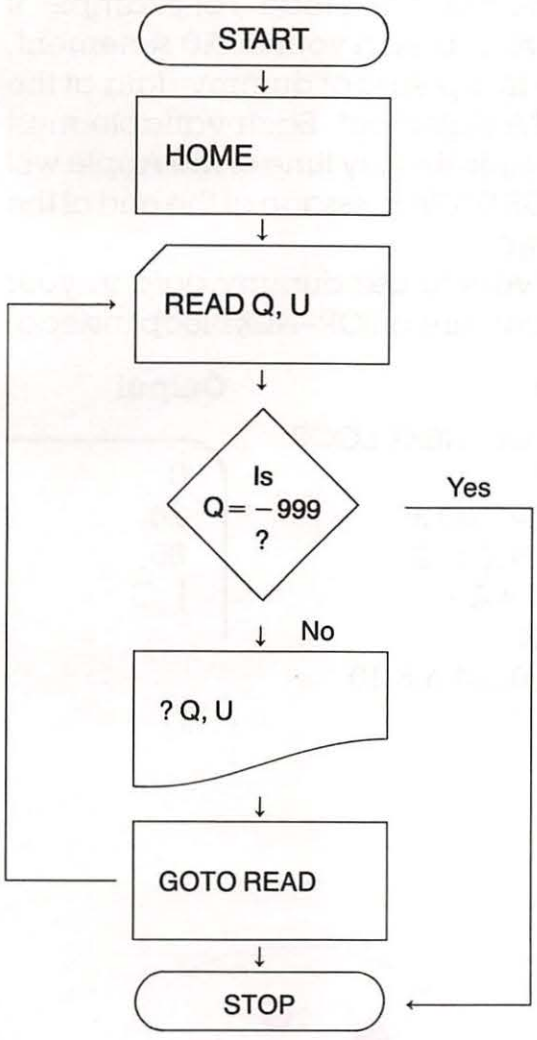
```
2       4       6
8       10      12

?OUT OF DATA ERROR IN 30
]
```

You may ask, "Is there any way I can get the program to end without printing the ?OUT OF DATA error message?" The answer is "Yes!" You need to:

1. Put some **dummy** data at the end of your DATA statement. (Dummy data is data that you want the Apple to read as a signal that the pointer is at the end of the data list.)
2. Use an IF–THEN statement that directs the Apple to the end of the program as soon as it READS the dummy data.



Flow chart



Program

```
10 REM DUMMY DATA
20 HOME
30 DATA 48,6,8.5,9,
   -999,-999
40 READ Q,U
50 IF Q=-999 THEN 80
60 ? Q,U
70 GOTO 40
80 END
```

Output

```
 48          6
 8.5         9
]
```

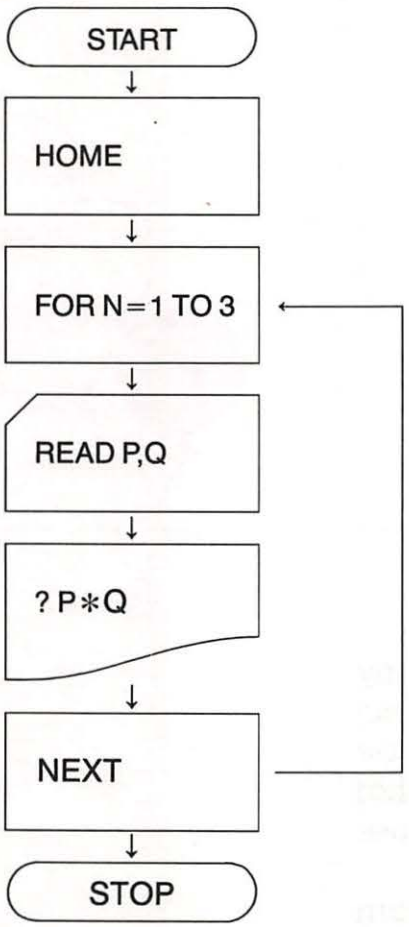
In the program, −999 was used as the dummy data. When you choose dummy data, select something you know you probably won't be using for data. For example, it is very unlikely that −999 would be data that you would want to use in a program.

The IF–THEN statement in line 50 of the program asks, "Does Q=−999?" after each gulp of data is read. When Q finally equals −999, the Apple is directed to the END of the program.

It is important to have dummy data for each variable that the Apple will read. For example, if you have five variables in your READ statement, you must have five pieces of dummy data at the end of the DATA statement. Each variable must have data read into it every time or the Apple will print the ?OUT OF DATA message at the end of the program's output.

If you don't want to use dummy data in your program, you can use a FOR–NEXT loop instead.

Flow chart



Program

```
10 REM FOR-NEXT LOOP
20 HOME
30 FOR N=1 TO 3
40       READ P,Q
50       ?P*Q
60 NEXT N
70 DATA 0,2,4,6,8,10
80 END
```

Output

```
0
24
80
]
```

The FOR–NEXT statements make the program loop three times. During the first loop, 0 and 2 are read into P and Q. In the second loop, the Apple reads 4 and 6, and during the third loop 8 and 10 are read. Because the loop is done only three times, the computer goes to line 80 and the program ends.

1. It's OK to have both numeric *and* string variables in the same READ statement. As a good programming practice, just make sure any data in the DATA statement that goes with the string variables has quotation marks around it.

```
Example: 30 READ     C,    D$,    E,    F$

         40 DATA     10  ,"KEN",  3   , "MITSY"
```

   If the data doesn't match up to your variables, you'll get a SYNTAX ERROR message.
2. You can't use an equation like 5 – 2 as data in a DATA statement. You must list only single numbers (5, 18, 343, etc.) or the Apple will give you a SYNTAX ERROR message.

to do: Programmer's Pastime #71, #72, #73, #74, #75

PROGRAMMER'S PASTIME #75

Use what you know about READ–DATA statements to write programs for the following tasks.

1. Write a program that multiplies three numbers.

Flow chart

Program

2. Write a program that lists the names of your friends.

Flow chart

Program

CHAPTER 45 Problem-Solving Programming

By now you have discovered that the Apple is a friend who can keep you company when you are bored, entertain you, and help you do your work. The most important thing the Apple can do for you, however, is to help you solve difficult problems.

So far you have learned how to program the Apple to do many things. You have learned most of the BASIC commands and algorithms necessary to write problem-solving programs. In this chapter you will learn how to put all of these valuable tools to use in order to teach the Apple to solve problems.

Before the Apple can give you the answer to a problem, there are many things that you must plan for in writing a good program.

Problem

Joe went to the store to buy some goldfish. He has $4.83 to spend. The fish bowl costs $2.25. Sand for the bottom of the bowl costs 49¢ a bag. Fish food is 60¢ for 4 ounces. The goldfish cost 80¢ each or two for $1.35. If Joe buys all of the supplies, how many fish can he afford to buy?

1. Think about the problem:

a. What exactly *is* the problem?	Can Joe buy one or two goldfish?
b. Do I understand the problem?	
c. What kind of answer do I want?	The answer should be one or two goldfish.
d. What do I need to know in order to find out the answer?	I need to know how much money Joe will have left over after buying the supplies. Then I will know how many fish he can buy.

2. Make a **data table:**

   a. What variables will I need to use in the program and what will they stand for?
   b. **Input variables** are variables that you already know the value of.
   c. **Output variables** are the answers that the Apple will give you.
   d. **Program variables** are variables that are used in the program to do other things.

Data Table

Input Variables

T=total $ that Joe can spend	=4.83
FB=cost of fish bowl	=2.25
S=cost of sand	= .49
FF=cost of fish food	= .60
G1=cost of 1 goldfish	= .80
G2=cost of 2 goldfish	=1.35

Output Variables

TC=total cost of FB+S+FF
L=money left after buying the supplies

There are no program variables in this program.

3. Algorithm:

   a. Break the problem into smaller parts.
   b. Figure out the step-by-step process you will use to solve the problem. Decide what operations you will use (+, −, /, and so on).

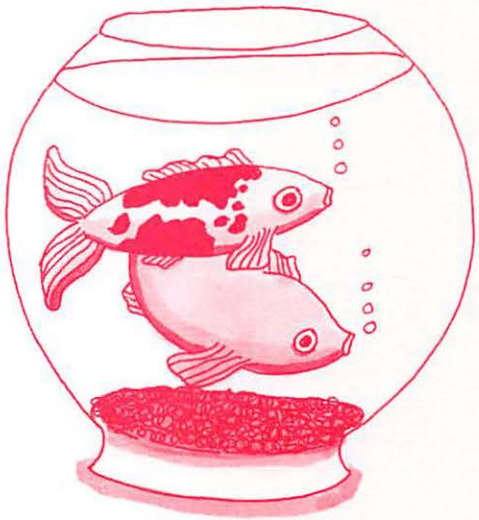
1. Find out the TC by adding FB+S+FF.
2. Find out L by subtracting T−TC.
3. Find out if L is enough to buy one or two goldfish. Ask:

   Is L>=G1?
   Is L>=G2?

4. Tell how many goldfish Joe can buy and how much money he would have after buying both the supplies *and* the goldfish.

4. Flow chart:

   a. Write the algorithm in flow chart form.



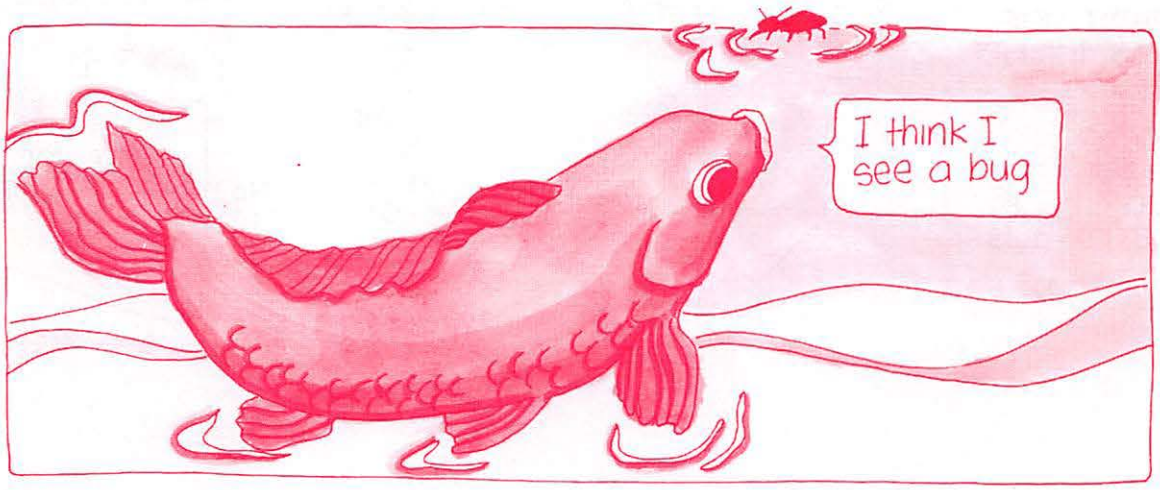
5. Coding:

   a. Write a BASIC program for the flow chart.

```
 10 LET T=4.83
 20 ? "JOE HAS" T "DOLLARS"
 30 LET FB=2.25 : LET S=.49 : LET FF=.60
 40 LET TC=FB+S+FF
 50 ? "FISH SUPPLIES COST" TC
 60 LET L=T-TC
 70 ? "JOE HAS" L "LEFT OVER"
 80 LET G1=.80 : LET G2=1.35
 90 IF L>=G2 THEN 130
100 IF L>=G1 THEN 150
110 ? "JOE CAN'T AFFORD ANY FISH"
120 GOTO 160
130 ? "JOE CAN BUY 2 GOLDFISH" : ? "AND
    HAVE" L-G2 "DOLLARS LEFT"
140 GOTO 160
150 ? "JOE CAN BUY 1 GOLDFISH" : ? "AND
    HAVE" L-G1 "DOLLARS LEFT"
160 END
```

6. Debugging:

   a. Pretend you are a computer. Follow the directions in your program to make sure it works. This is called **tracing** the program.
   b. Run the program on the Apple to check for bugs.
   c. Does the program do what you wanted it to do?



7. Revising:

a. Is there a better or shorter way to write your program?	Yes. We can write the program using READ–DATA statements.
b. Can you use better programming style?	Yes. We can use REMARKS.
c. Can you design your output better?	Yes. We can clear the screen and leave spaces between the printing.

```
10 REM CALCULATING PURCHASE OF GOLDFISH & SUPPLIES
20 HOME
30      READ T, FB, S, FF, G1, G2
40      DATA 4.83, 2.25, .49, .60, .80, 1.35, 99, 99, 99, 99, 99, 99
50      IF T=99 THEN 60
60      ?"JOE HAS" T "DOLLARS"
70      LET TC=FB+S+FF
80      ?
90      ?"FISH SUPPLIES COST" TC
100     LET L=T-TC
110     ?
120     ?"JOE HAS" L "LEFT OVER"
130     IF L>=G2 THEN 170
140     IF L>=G1 THEN 190
150     ?"JOE CAN'T AFFORD ANY FISH"
160     GOTO 200
170     ?"JOE CAN BUY 2 GOLDFISH AND HAVE" L-G2 "DOLLARS LEFT"
180     GOTO 200
190     ?"JOE CAN BUY 1 GOLDFISH AND HAVE" L-G1 "DOLLARS LEFT"
200     END
```

Using the READ–DATA statements may be the best way to write this program. Why? If the price of goldfish or supplies goes up, you can change the DATA statement and the program will be updated.

You can write good problem-solving programs for the Apple to solve if you follow these seven steps.

1. THINK about the problem
2. DATA TABLE for input, output, and program variables
3. ALGORITHM—How can I solve the problem, step by step
4. FLOW CHART
5. CODE the flow chart into a BASIC program
6. DEBUG
7. REVISE the program to make it the best

to do: Programmer's Pastime #76

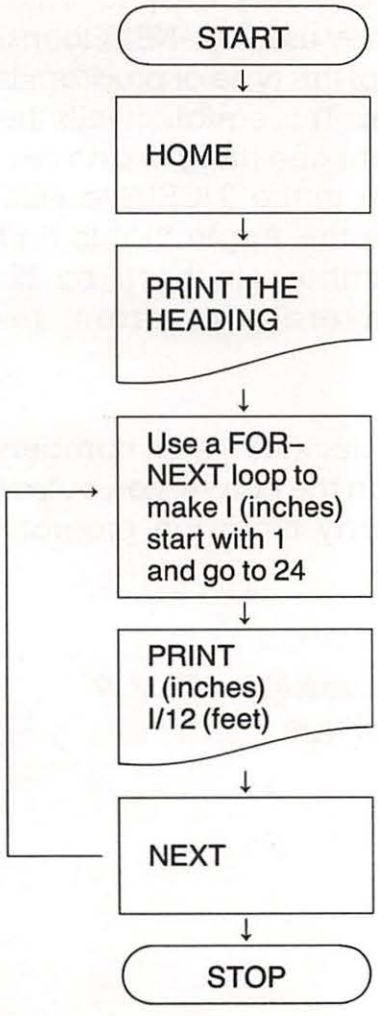
CHAPTER 46 Conversions

The Apple can be especially good at running a program that helps you **convert** one thing to another. Convert means to change, so a conversion is changing information to a different type. For example, you can convert:

inches to feet	decimals to fractions
feet to meters	miles to kilometers

You can program the Apple to make the conversion and then print a table that shows how the two types of conversions are equal to each other. For example:

Flow chart



Program

```
10 REM CONVERTING INCHES
   TO FEET
20 HOME
30 ? "INCHES", "FEET" : ?
40 FOR I=1 TO 24
50       ?I,I/12
60 NEXT I
70 END
```

Output

```
INCHES  FEET
1       .083333
2       .16666667
3       .25
4       .333333
5       .41666667
6       .5
7       .583333
8       .66666667
9       .75
10      .833333
11      .91666667
12      1
.       .
.       .
.       .
24      2
]
```

The output of this program shows how inches compare to feet. You can tell from the program that:

1 inch =0.083333 of a foot
3 inches=0.25 or ¼ of a foot
6 inches=0.5 or ½ of a foot
12 inches=1 foot

Line 30 prints the **heading** for the output. The heading of a program is usually printed first in the output. It explains the meaning of the numbers that follow. The heading in the program is:

INCHES FEET

The heading tells you that the numbers listed under INCHES are inches, and the numbers listed under FEET mean feet.

Conversion programs are very easy to write. They are short because they use FOR–NEXT loops. The most important part of this type of program is the **conversion equation**. This equation tells the Apple how to convert from one thing to another. The conversion equation in the INCHES to FEET program is I/12. This tells the Apple that to find feet, it must divide the number of inches (I) by 12.

To write a good conversion program, remember to include:

1. a heading
2. a FOR–NEXT loop that decides which numbers to start and end with on the conversion output *and* decides how many times the program will loop.
3. conversion equation

to do: Programmer's Pastime #77, #78, #79
Component 7 Fun Page

COMPONENT 8

CHAPTER 47 TAB

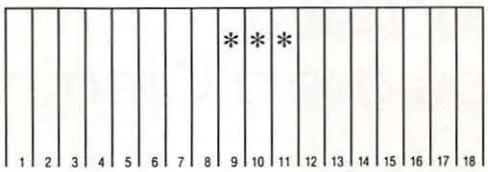
You have learned how to control where something is printed on the Apple's screen by using commas and semi-colons. There is a function, called **TAB**, that you can use to control the screen output. A function is an operation that the computer does automatically, like a small built-in program.

The TAB function is used only in PRINT statements, like this:

```
10 ? TAB(10); "***"
```

You have learned that the Apple's screen has 40 rows and 40 columns labeled from 0 to 39. This is true only when the Apple is in graphics mode. When the Apple is in direct mode it still has 40 columns, but it has only 24 rows. The columns are labeled 1 through 40 and the rows are labeled 1 through 24.

The TAB function tells the Apple to move the cursor across the screen to column 10 and begin printing in column 10.



Columns

```
PRINT TAB(20); "***"
```

will make the Apple begin printing in the 20th screen column.

The TAB function above tells the Apple to move the cursor across the screen to column 10 and begin printing in column 10.

When you use the TAB function in a PRINT statement, you must remember to put a semi-colon after the last parenthesis.

```
PRINT TAB(25); "$"
```

This function will cause a $ to be printed in column 25.

The number inside the parentheses of a TAB function can be a variable. You must, however, introduce the variable in the program before you use it in a TAB function. For example:

```
10 HOME
20 LET Z=33
30 ? TAB(Z); "!"
```

You may also use more than one TAB function in a PRINT statement. For example:

```
20 ? TAB(10); "**"; TAB(15); "%%"
```

This function causes the Apple to print an * in columns 10 and 11 and a % in columns 15 and 16. Notice how the two TAB functions are separated by a semi-colon.

You can have lots of fun designing output by using TAB.

to do: Programmer's Pastime #80, #81

CHAPTER 48 Moving Around the Screen

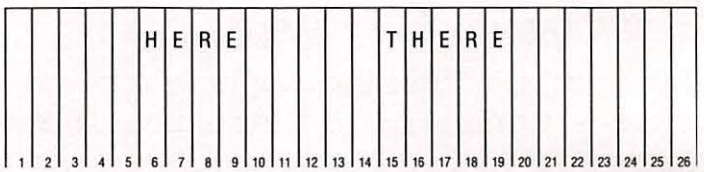
There is another function that works very much like the TAB function. **SPC** is the space-over function. It is also used in a PRINT statement, like this:

```
10 ? SPC(10); "PRINT HERE"
```

The SPC function tells the Apple to make 10 blank spaces on the screen and then print what is inside the quotation marks. Notice that the SPC function also must have a semi-colon after the second parenthesis.

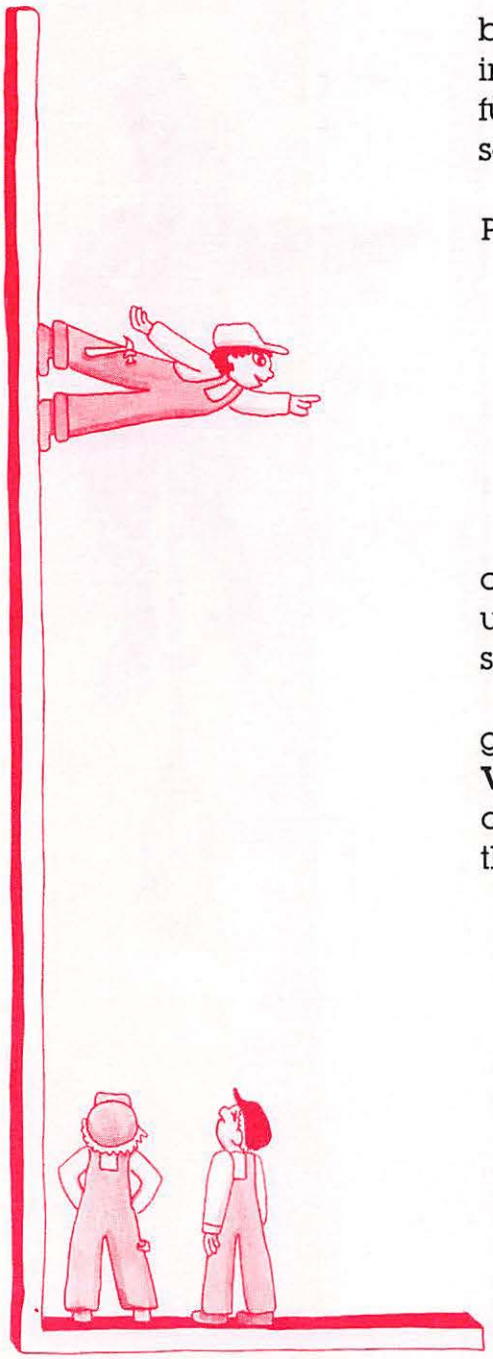
You can use more than one SPC function in a PRINT statement. For example:

```
10 ? SPC(5); "HERE"; SPC(5); "THERE"
```



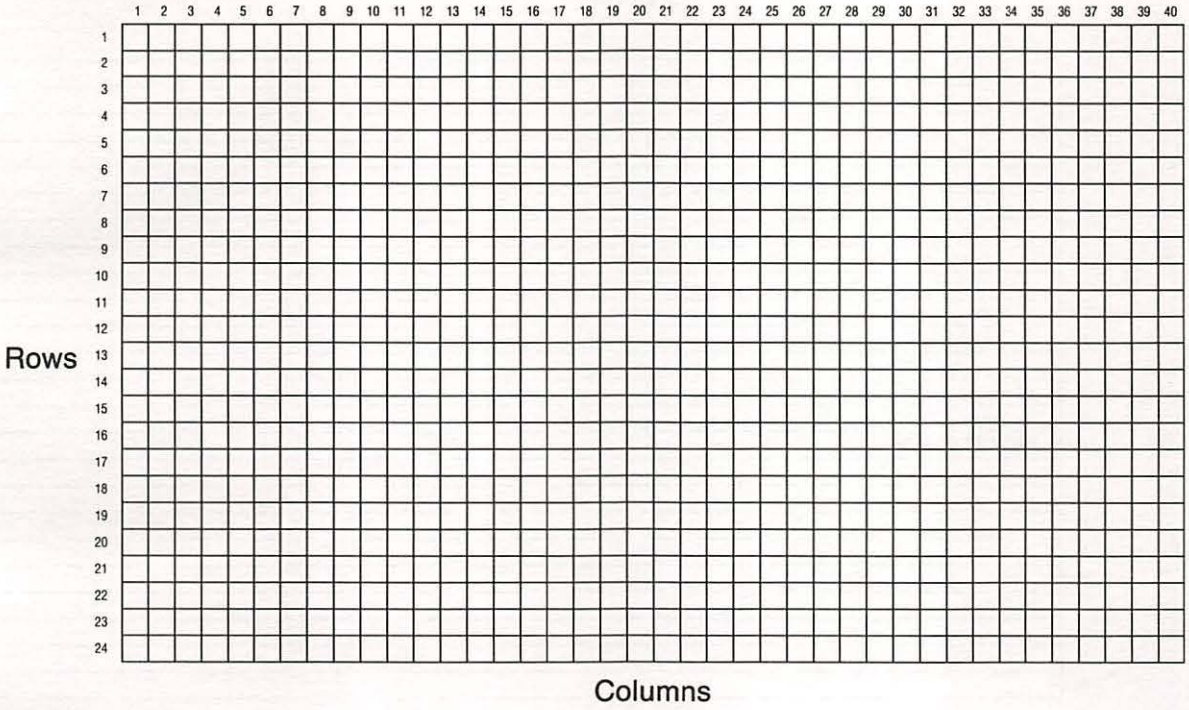
The SPC functions above tell the Apple to space over five columns and print HERE in the next column. Then the Apple must make five screen spaces before printing THERE.

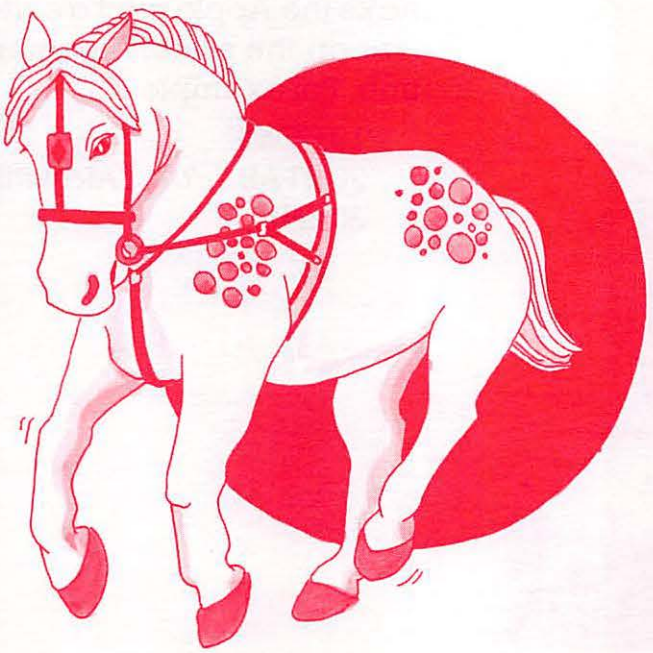
There are two other statements that let you program the Apple to print anywhere on the screen. **VTAB** moves the cursor vertically (up and down), and **HTAB** moves the cursor horizontally (across the screen).



Remember that when the Apple is in direct mode, the screen is made up of 40 columns labeled 1 through 40 and 24 rows labeled 1 through 24.

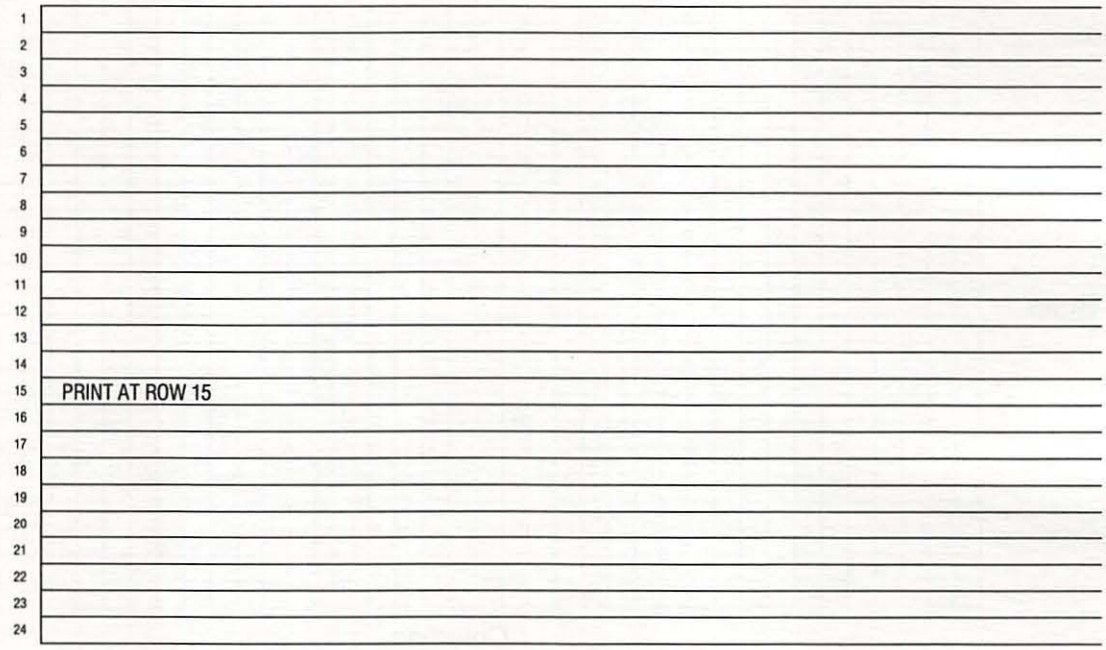
The Apple's Screen in Direct Mode





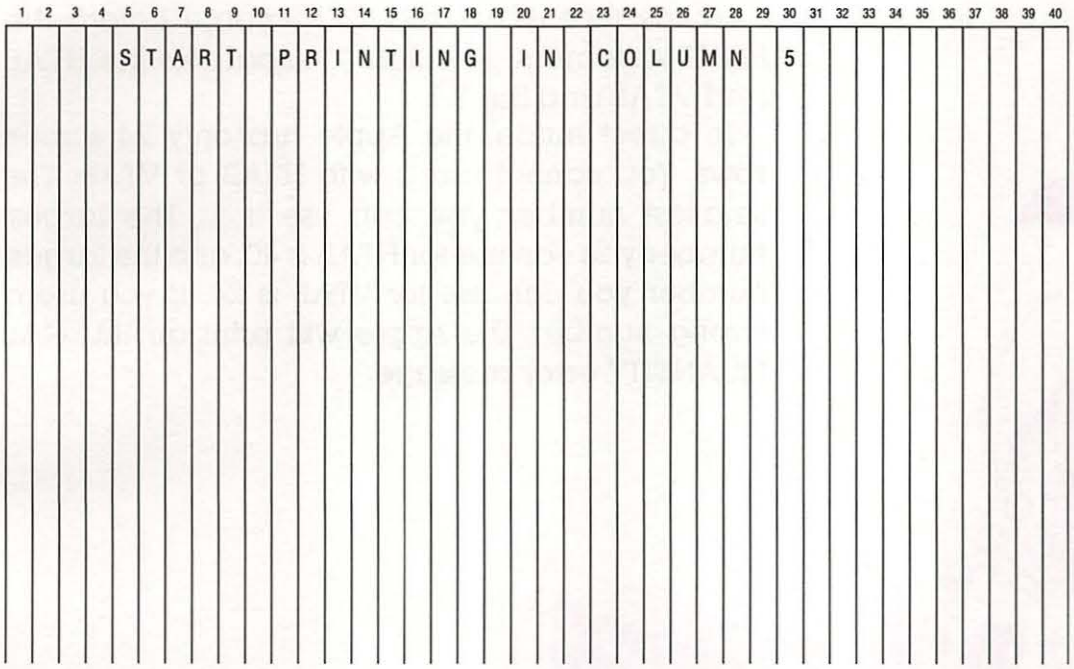
You can use VTAB with a PRINT statement to make the Apple print on a certain screen line (at a certain row). For example:

```
10 HOME
20 VTAB 15: ? "PRINT AT ROW 15"
30 END
```



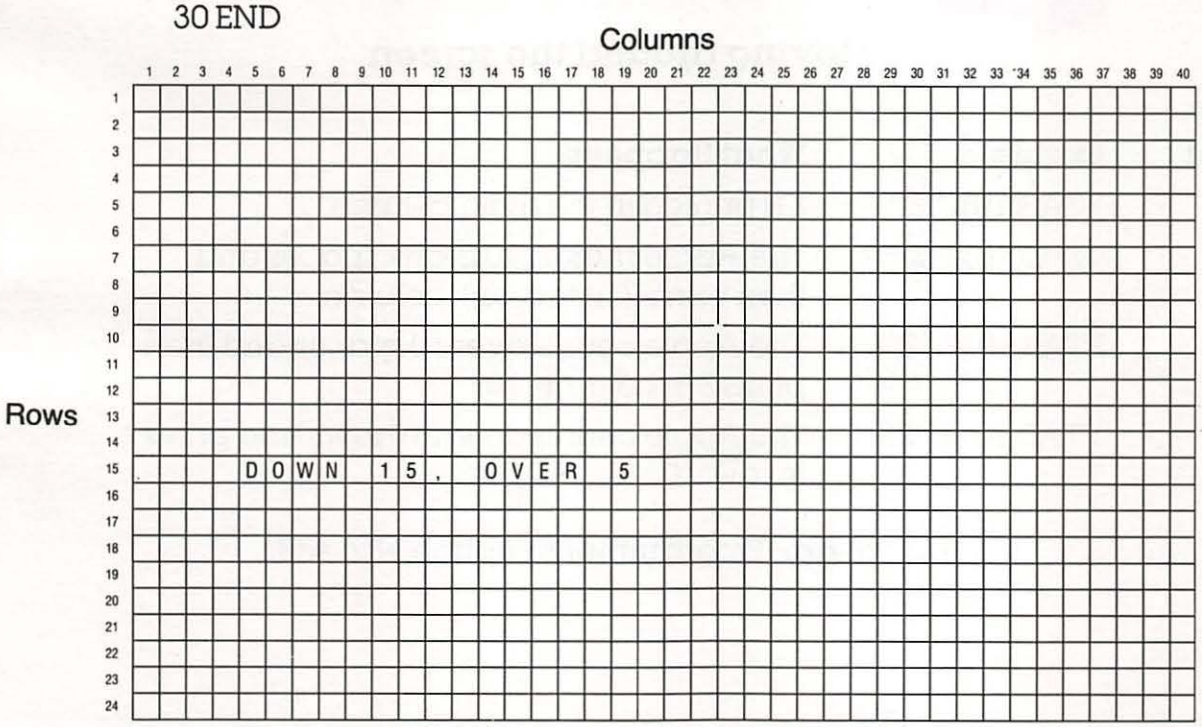
You can use HTAB with a PRINT statement to make the Apple print a certain number of spaces over on the screen—beginning in a certain column. For example:

```
10 HOME
20 HTAB 5: ? "START PRINTING IN COLUMN 5"
30 END
```



You can use both VTAB and HTAB in a PRINT statement to make Apple print at a certain location on the screen. For example:

```
10 HOME
20 VTAB 15: HTAB 5: ? "DOWN 15, OVER 5"
30 END
```



Notice that HTAB and VTAB come *before* the PRINT statement. A colon (:) separates the HTAB and VTAB and the ? .

In direct mode, the Apple has only 24 screen rows. You cannot use 0 with HTAB or VTAB. The smallest number you can use is 1. The largest number you can use for HTAB is 40, and the largest number you can use for VTAB is 24. If you use a wrong number, the Apple will print an ILLEGAL QUANTITY error message.

Moving around the screen

Function/ Statement	Example	What Happens
TAB	?TAB(10);"Z"	Z is printed in the ninth column.
SPC	?SPC(10);"Z"	The Apple makes 10 blank spaces and then prints Z in the next column.
HTAB	HTAB 10: ?"Z"	The Apple counts over 10 spaces and then prints Z in column 10.
VTAB	VTAB 10: ?"Z"	The Apple counts down 10 rows and prints Z in row 10.

to do: Programmer's Pastime #82, #83

CHAPTER 49 Motion Pictures

You can have lots of fun writing programs that move graphics across the screen. An easy way to do this is to use strings and add them together. Run the program below to see how this works:

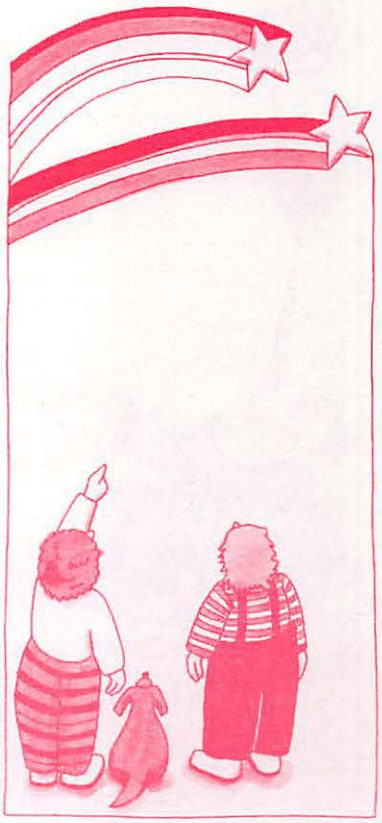
```
 10 REM A MOVING STAR
 20 H OME
 30 LET B$ = " " : LET S$ = "*"
 40 VTAB 12
 50 FOR L = 1 TO 39
 60       HTAB L
 70       FOR TL = 1 TO 100: NEXT TL
 80       ? B$ + S$;
 90 NEXT L
100 END
```

A blank space is stored in B$, and the * is stored in S$. The VTAB 12 statement makes the star move across the screen at row 12. The FOR–NEXT loop does many things. The value of L increases from 1 to 39. The HTAB L statement makes the * move from column 1 to column 39 across the screen. The FOR–NEXT time loop slows down the movement. The trick to the movement is in line 80. In order to make a graphic move, it must be erased after it is printed, and then printed again in the next column. This can be done by adding the two strings together. B$ + S$ means that B$ and then S$ will be printed in that order each time the loop is done. This is what moves the * (in S$) across the screen.

The example below shows how the movement is created.

```
*
ø*
  ø*
    ø*
      ø*
        ø*
```

The ø is the blank that erases the star as it moves along.



You can make a word move across the screen by changing the contents of S$. Make the following change and run the program again.

```
30 LET B$="ø" : LET S$="MOVIN' ON"
```

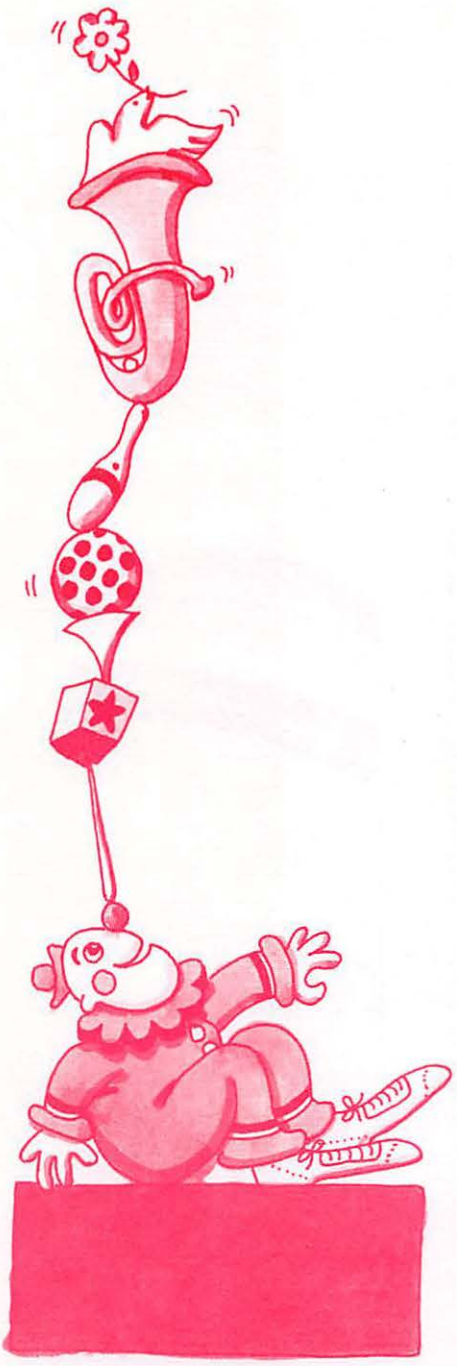
You can change how fast the graphic moves by changing the FOR–NEXT time loop.

The following program makes the word HI move down the screen from the top to the bottom.

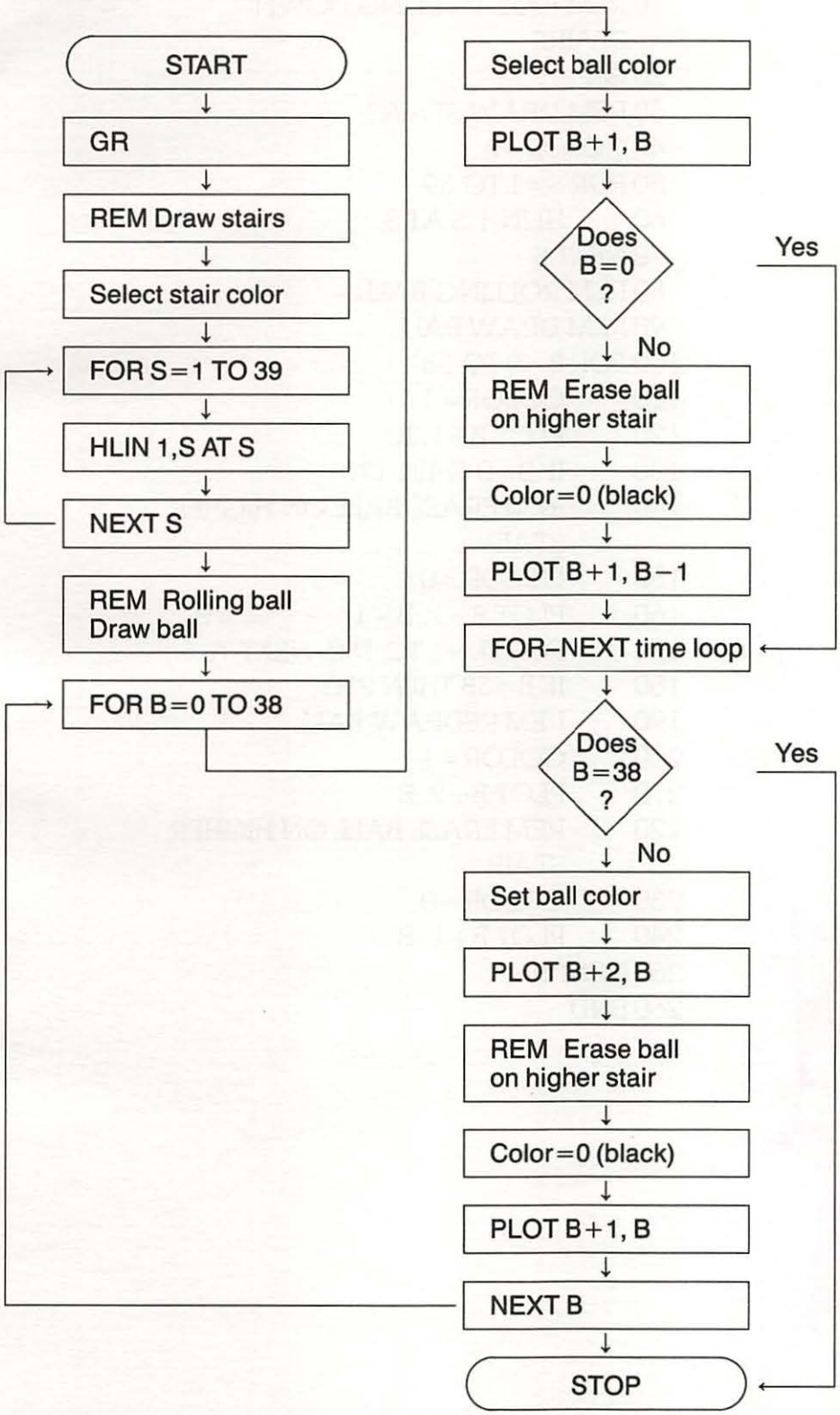
```
10 REM FALLING
20 HOME
30 LET B$="øø": LET M$="HI"
40 FOR L=1 TO 24
50      ?B$
60      HTAB 10
70      VTAB L
80      ?M$;
90      HTAB 10
100     FOR TL=1 TO 100: NEXT TL
110 NEXT L
120 END
```

The contents of B$ must have two blank spaces because the word HI in M$ is made of two letters.

You can also make moving pictures in lo res graphics. Instead of using a string variable that contains a blank space, you will use the command COLOR=0 to make the screen black and erase the graphic. The following program makes a ball roll down a set of stairs. Run it on the Apple.



Flow chart


START
GR
REM Draw stairs
Select stair color
FOR S=1 TO 39
HLIN 1,S AT S
NEXT S
REM Rolling ball
Draw ball
FOR B=0 TO 38
Select ball color
PLOT B+1, B
Does B=0 ?
Yes
No
REM Erase ball on higher stair
Color=0 (black)
PLOT B+1, B-1
FOR-NEXT time loop
Does B=38 ?
Yes
No
Set ball color
PLOT B+2, B
REM Erase ball on higher stair
Color=0 (black)
PLOT B+1, B
NEXT B
STOP

Program

```
 10 REM BALL ROLLING DOWN
    STAIRS
 20 GR
 30 REM DRAW STAIRS
 40 COLOR=6
 50 FOR S=1 TO 39
 60       HLIN 1,S AT S
 70 NEXT S
 80 REM ROLLING BALL
 90 REM DRAW BALL
100 FOR B=0 TO 38
110       COLOR=1
120       PLOT B+1, B
130       IF B=0 THEN 170
140       REM ERASE BALL ON HIGHER
          STAIR
150       COLOR=0
160       PLOT B+1, B-1
170       FOR TL=1 TO 150: NEXT TL
180       IF B=38 THEN 250
190       REM REDRAW BALL
200       COLOR=1
210       PLOT B+2, B
220       REM ERASE BALL ON HIGHER
          STAIR
230       COLOR=0
240       PLOT B+1, B
250 NEXT B
260 END
```

To make moving graphics in direct mode:

Use string variables. One string variable must contain a blank space(s).

To make moving graphics in lo res graphics mode:

Use the color commands. COLOR=0 erases a graphic.

to do: Programmer's Pastime #84

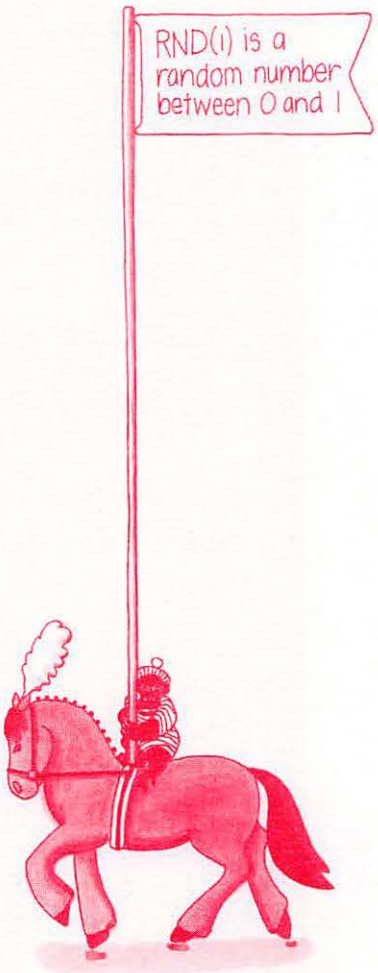
CHAPTER 50 Random Numbers and Integers

The word **random** means "having no pattern or special purpose." Therefore, **random numbers** are a list of numbers that are not in any particular order or for any particular purpose. An example of a list of random numbers might be: 7, 43, −6, 0.7, 413. There is no order or number pattern in this list, and the numbers listed have no special purpose or meaning.

Random numbers are often used in two types of computer programs:

1. teaching programs, also called CAI (Computer-Aided Instruction)
2. games and simulations*

You will use the **RND** function to create random numbers in a program. For example:




```
10 REM CREATE RANDOM NUMBERS
   BETWEEN 0 AND 1
20 HOME
30 FOR L=1 TO 10
40      LET X=RND(1)
50      ?X
60 NEXT L
70 END
```

The program you just read tells the Apple to print any number between 0 and 1 ten times. The Apple will pick numbers randomly each time. There will be no order to the numbers. Each time you run the program, the Apple will print a different list of numbers.

If you want the Apple to print a list of numbers between 0 and 10 randomly, you would change the RND function to:

```
LET X=RND(1)*10
```

*A simulation is a real-life "game." It imitates something the way it would really happen.

If you want the Apple to print a list of random numbers between 0 and 100 you would change the RND function to:

```
LET X=RND(1)*100
```

If you are writing a game program, you will probably not want 0 to be a random number—especially if the game is simulating the roll of a die. To print any random number between 1 and 101, change the RND function to:

```
LET X=1+RND(1)*100
```

This causes the lowest possible number to be 1.00000001 and the highest possible number to be 100.999999.

An **integer** is a whole number. Numbers like 0.25 and 6.32 are not whole numbers—they are decimals. You can use the **INT** function to create whole numbers or integers in a program.

For example:

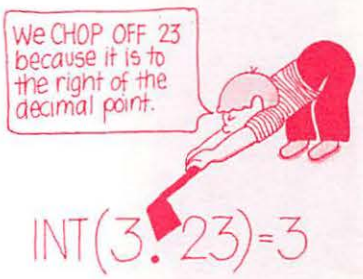
Program

```
10 REM CONVERTING DECIMALS TO
   INTEGERS
20 HOME
30 ? "DECIMAL", "INTEGER"
40 FOR X=1 TO 5 STEP .5
50       ? X, INT(X)
60 NEXT X
70 END
```

Output

DECIMAL	INTEGER
1	1
1.5	1
2	2
2.5	2
3	3
3.5	3
4	4
4.5	4
5	5

Notice that the integer for the decimal 1.5 is 1. The integer for the decimal 2.5 is 2, and so on. The INT function **rounds** the decimal **down** to the nearest integer. You could also say that the INT function "chops off" or **truncates** anything to the right of the decimal point.



Sometimes you will want the Apple to print random numbers that are only integers. To do this, you will use both the RND and INT functions. For example:

```
10 REM ROLL IT
20 HOME
30 FOR L=1 TO 10
40      LET X=INT(1+RND(1)*6)
50      ?X
60 NEXT L
70 END
```

This program tells the Apple to print a random integer between 1 and 6. The smallest possible number would be 1 and the highest possible number would be 6.

Let's say you want the Apple to print a random integer between and including 2 and 12. The INT and RND function should say:

```
LET X=INT(2+RND(1)*11)
          ↑
```

The smallest number that will be printed.

To create random integers between and including 50 and 85, use:

```
LET X=INT(50+RND(1)*36)
          ↑
```

smallest number 85−50=35 35+1=36

The formula for creating random integers between A and B (where A is the smallest integer and B is the largest) is:

```
INT(A+RND(1)*(B−A+1))
```

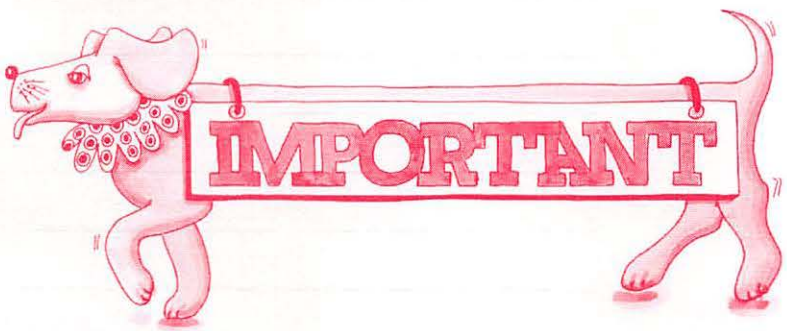
To create random integers between 26 and 77, use the formula like this:

```
LET X=INT(A+RND(1)*(B−A+1))
          ↑          ↑  ↖
LET X=INT(26+RND(1)*(77−26+1))
                    ↙
LET X=INT(26+RND(1)*52)
```

Here is an example of how to use the INT and RND functions in a CAI program that gives the student practice in adding integers.

Program	**What happens**
10 REM PRACTICE ADDING	
20 HOME	
30 LET A1 = INT(RND(1) * 100)	A random integer for A1 is created.
40 LET A2 = INT(RND(1) * 100)	A random integer for A2 is created.
50 ? A$ "+" A2 "=" ;	The equation for the student to do is printed.
60 INPUT S	The student types his or her answer.
70 LET T = A1 + A2	The Apple calculates the answer to the equation.
80 IF T = S THEN 110	The student's answer is compared to the correct answer. If S=T go to line 110.
90 ? "NOPE. TRY AGAIN"	If the student's answer is wrong, the Apple tells the student.
100 GOTO 50	The program goes back to line 50 and the same equation is given to the student.
110 ? "RIGHT ON!"	The Apple tells the student he or she is right.
120 GOTO 20	Goes to the beginning of the program, picks new random integers, and starts all over again.

Run this program to see how it works on the Apple.



The placement of parentheses in RND and INT functions is very important.

If the parentheses are in the wrong places, the program won't run properly.

to do: Programmer's Pastime #85, #86, #87, #88, #89, #90

PROGRAMMER'S PASTIME #89

Make a flow chart and write a program that will print 10 random decimals between 1 and 100 and then print the integer for each.

Flow chart

Program

CHAPTER 51 Writing Game Programs

Playing computer games can be an enjoyable recreational experience. One of the rewards of learning how to program a computer is being able to write game programs.

There are basically three types of computer games:

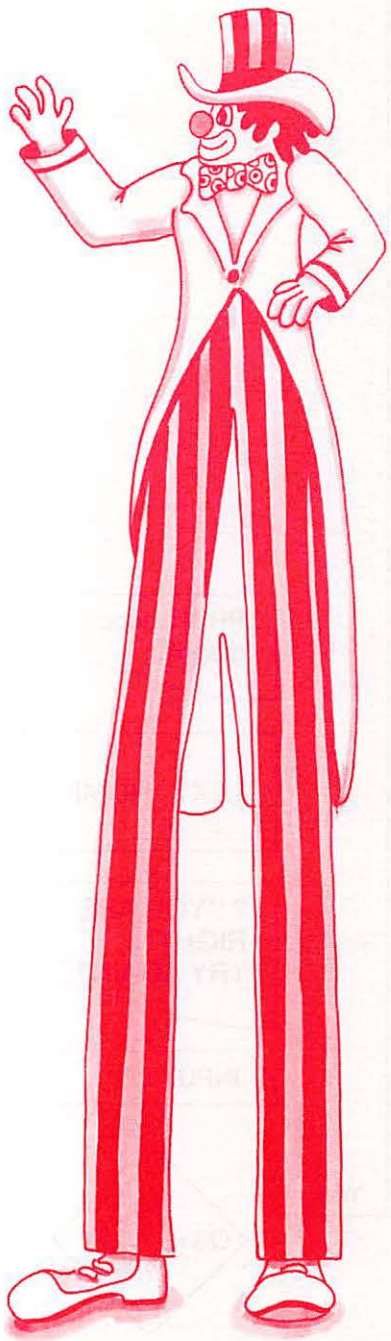
1. **mathematical games:** games involving numbers and/or solving arithmetic or mathematical problems.
2. **recreational games:** many different games could fall in this category. I think of *Space Invaders* and *Dungeons and Dragons* as recreational games.
3. **simulations:** games that imitate real-life situations. For example, *Sell Lemonade*.

In writing a game program, you must be sure the program will be **user friendly**. This means that the program is easy for anyone to use.

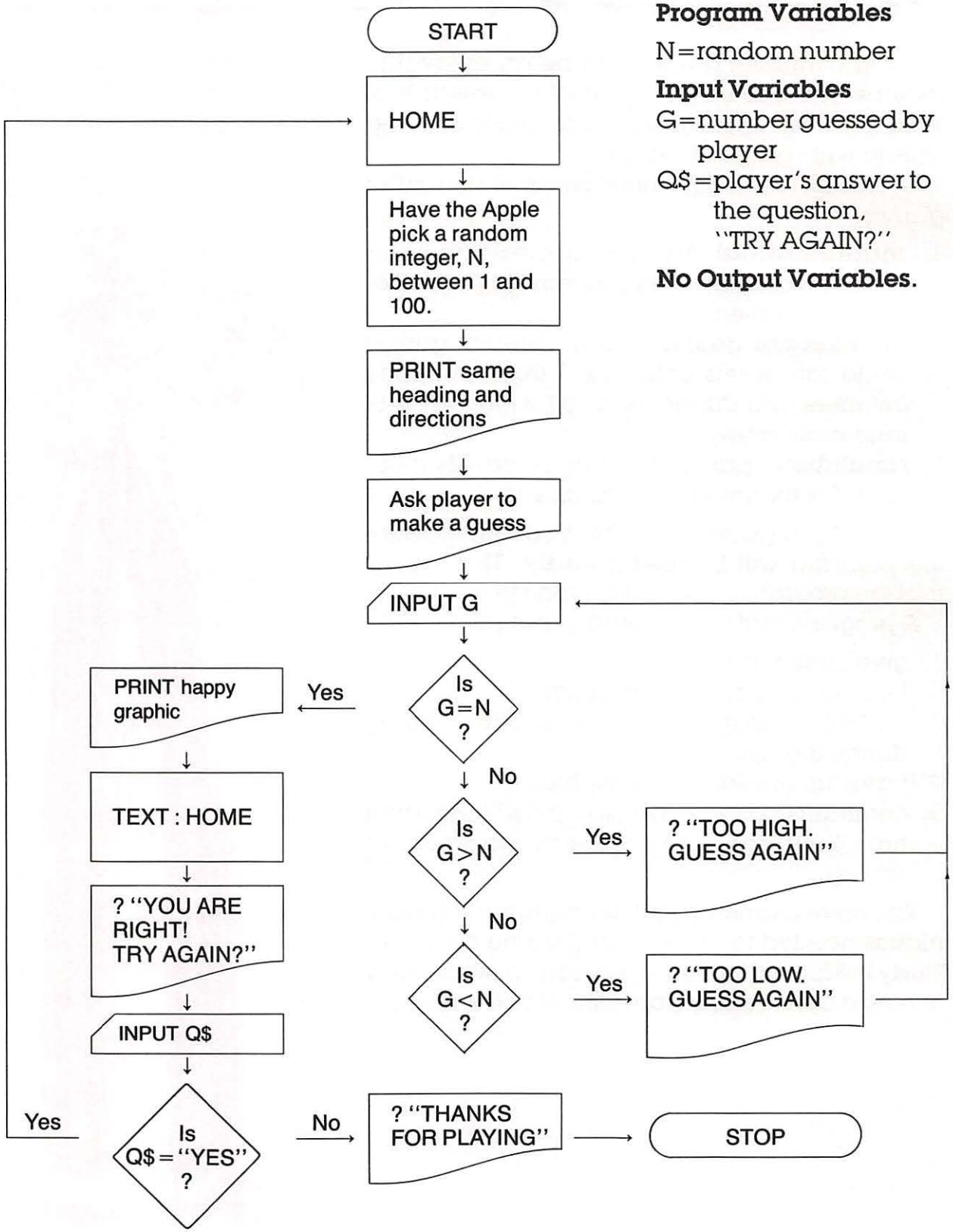
A program that is user friendly should:

1. give clear directions;
2. have easy-to-read screen output;
3. be free of bugs and not be "broken" easily during the run;
4. have fun or interesting graphics;
5. communicate with the player (tell the player how they are doing through messages or scores).

You have learned all of the programming techniques needed to write a good game program. Study the following game program to get an idea of how a user-friendly game should be written.



Flow chart



Data table

Program Variables

N=random number

Input Variables

G=number guessed by player

Q$=player's answer to the question, "TRY AGAIN?"

No Output Variables.

Program

```
 10 REM ** GUESS A NUMBER GAME **
 20 HOME
 30 REM ** CHOOSE A RANDOM NUMBER **
 40 LET N=INT(1+RND(1)*100)
 50 REM ** BEGIN GAME **
 60 HTAB(10): VTAB(5): ? "GUESS A NUMBER
    GAME"
 70 HTAB(2) : VTAB(8): ? "GUESS A NUMBER
    BETWEEN 1 AND 100"
 80 INPUT G
 90 IF G=N THEN 120
100 IF G>N THEN ? "TOO HIGH. GUESS AGAIN.":
    GOTO 80
110 IF G<N THEN ? "TOO LOW. GUESS AGAIN.":
    GOTO 80
120 REM ** CORRECT GUESS **
130 REM ** GRAPHIC **
140 GR: COLOR=2
150 PLOT 22,17 : PLOT 24,17
160 COLOR=13
170 PLOT 23,19
180 COLOR=11
190 PLOT 20,20 : PLOT 21,21 : PLOT 22,22 : PLOT
    23,22 : PLOT 24,22 : PLOT 25,21 : PLOT 26,20
200 FOR T=1 TO 1000: NEXT T
210 REM ** CONGRATULATE THE PLAYER **
220 TEXT: HOME
230 HTAB 5: VTAB 5:? "YOU ARE RIGHT! TRY
    AGAIN";
240 INPUT Q$
250 IF Q$="YES" THEN 20
260 HTAB 8: VTAB 8:? "THANKS FOR PLAYING."
    PLAYING."
270 END
```

Does this program have the five elements of a good program?

1. Clear Directions: lines 60–70
2. East-To-Read Output: The HTAB and VTAB statements do this.
3. Free of Bugs: There is one possible bug. Look at lines 230 through 250.

   If the user types yes, the game will start over again. If the user types no or even a mistake (like QYES) the program will end. The program should be written so that if something other than yes or no is typed, the Apple will go back to line 230 and print the question, TRY AGAIN another time instead of ending the program. This technique is called *accident proofing* user responses.

4. Fun, Interesting Graphics: A happy face is printed when the number is guessed.
5. Messages to the Player: Lines 100 and 110 tell the player if the guess is too high or too low.

   Line 230 congratulates the player for guessing correctly.

   Line 260 thanks the player for playing.

Run this program so you can see firsthand how it works. Maybe you will have some suggestions on how to make the program even better!

to do: Programmer's Pastime #91

CHAPTER 52 You Are a Creative Programmer!

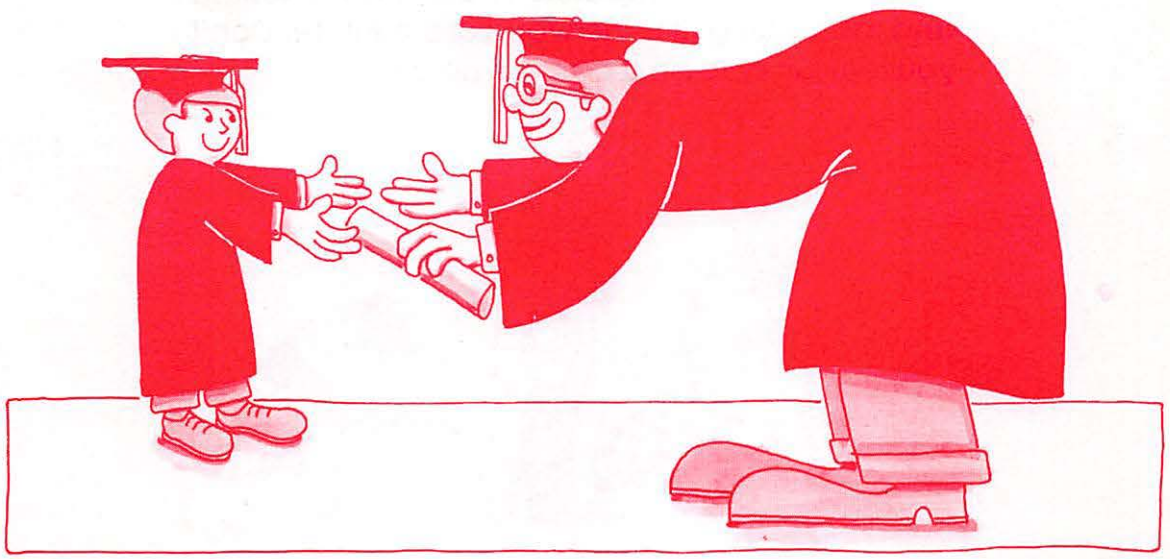
You have learned how to use the Apple as a calculator and a problem-solving tool. You know that the Apple can also help you with your creative projects. Computer art and design can be amazing. You have the basic skills needed to create interesting graphics.

Another creative outlet for computers is animation and sound generation. Did you know that people program computers to make music and even to talk?

Now that you know how to create visual pictures and designs, it is hoped that you will continue to learn more about computer animation and sound. The possibilities of what you can do with your Apple are endless!

Use your imagination . . . explore . . . try new things! Your Apple is your friend, a tool, and a key to your future!

to do: Component 8 Fun Page



AFTERWORD

Congratulations!

You are now a veteran computer programmer! You've come a long way!

You now have the skills needed to write programs in BASIC to control a computer. You know how to use the computer to solve your problems (problem-solving programming) and to entertain yourself and others (recreational programming). The skills you have learned enable you to create designs and new ideas on the computer (creative programming). You should be very proficient at programming the computer to do just about anything!

Sure, there are still many more BASIC programming techniques to learn. Some of them are complicated but others are shortcuts that will make your programming easier!

Once you are a pro at communicating in BASIC, there are other computer languages waiting for you—PASCAL, LOGO, and PILOT, to name just a few.

The world of computers is certainly exciting and fascinating. It is the world of the future. Don't you feel lucky to be a part of it now?

APPENDIX A

Initializing New Disks

1. Boot the disk with the system master disk.
2. Type the following greeting program on the keyboard:

```
10 REM HELLO
20 REM GREETING PROGRAM
30 ? "(student's name)'S DISK"
40 ? "INITIALIZED ON (date) FOR
   (memory size)K APPLE II"
50 NEW
```

3. Put the new uninitialized disk into drive 1.
4. Type: INIT HELLO
5. The disk drive will make grinding noises as the disk is initialized and prepared for storing programs.
6. Any time CATALOG is typed, the Apple will display the names of the programs currently stored on the disk.

APPENDIX B

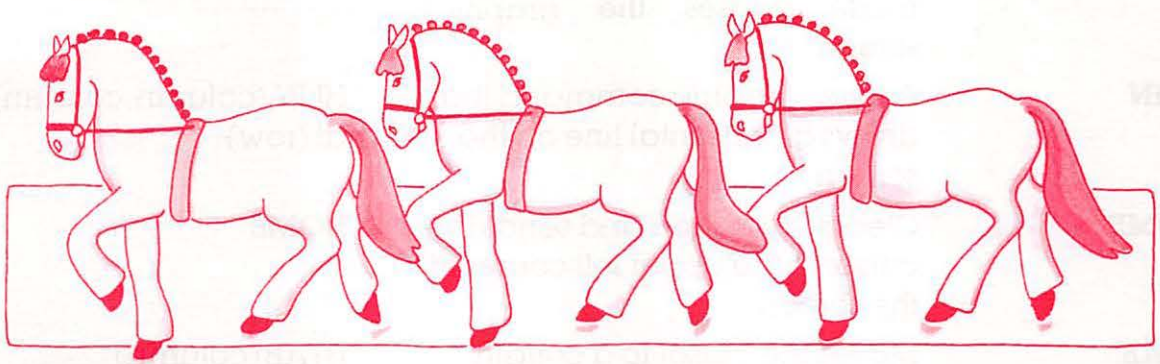
Common Error Messages

Apple Error Messages:

1. **SYNTAX ERROR:** This common error message is caused by misspelled words, incorrect punctuation, extra characters, and so on. It also occurs when a BASIC word is not used.
2. **OUT OF MEMORY ERROR:** This occurs when all of the available RAM memory is used up. The program entered may be too long.
3. **TYPE MISMATCH ERROR:** This error message will occur when you try to input a number into a string variable, or a word or letter into a numeric variable.
4. **UNDEF'D STATEMENT ERROR:** In your program, the Apple was told to go to a line number that does not exist.
5. **CAN'T CONTINUE ERROR:** You have tried to continue the program (using CONT) when no program existed, after an error happened, or after a change has been made in the program.
6. **DIVISION BY ZERO ERROR:** The Apple cannot divide a number or expression by zero.
7. **FORMULA TOO COMPLEX ERROR:** A program line may have more than two IF–THEN statements.
8. **ILLEGAL QUANTITY ERROR:** A number value is too big or too small.
9. **NEXT WITHOUT FOR ERROR:** The programmer forgot to put a FOR statement that matches the NEXT statement in the loop.
10. **OUT OF DATA ERROR:** No more data is available for the READ statement.
11. **OVERFLOW ERROR:** The number entered or calculated is too large or small.
12. **STRING TOO LONG ERROR:** The user tried to add strings that together had more than 255 characters.

DOS Error Messages:

1. **DISK FULL:** The disk is full and no more programs or information can be stored on it.
2. **FILE LOCKED:** You have tried to save, delete, or rename a locked program.
3. **FILE NOT FOUND:** You tried to load or run a program that does not exist on the disk. Often you have merely misspelled the name of the program.
4. **I/O ERROR (INPUT/OUTPUT ERROR):** You have tried to save a program to the disk or load or run a program from the disk and it is not working properly. This is often caused by the disk drive door being left open, the disk not being initialized, or the disk being defective.
5. **LANGUAGE NOT AVAILABLE:** You have tried to load or run a program that was written in a language that the Apple does not have in memory. For example, to run a program written in Integer BASIC, the system master disk must first be booted.
6. **SYNTAX ERROR:** A command to the disk drive was misspelled or incorrectly written.
7. **WRITE PROTECTED:** You have tried to save or delete a program on a disk that is write protected. Disks are write protected so you will not accidentally write over valuable programs. Remove the tab that covers the opening on the side of the disk.



APPENDIX C

BASIC Commands, Statements and Functions Used in This Book

Command, Statement, or Function	Purpose	Example
BRUN	Runs a program written in binary machine language from the disk.	BRUN (program name)
CATALOG	Shows a listing of all programs stored on a disk.	CATALOG
COLOR=	Assigns the color for lo res graphics.	COLOR=(number between 0 and 15)
DATA	Holds data for the variables in the READ statement.	DATA 4,72, "Y"
DELETE	Erases a program from the disk.	DELETE (program name)
END	Makes a program stop at the end.	END
FLASH	Makes the output flash on the screen.	FLASH
FOR–NEXT	Creates a loop in a program.	FOR Z=1 TO 10 NEXT Z NEXT Z
GOTO	Tells the computer to go to a certain location in the program. One way to jump or create a loop.	GOTO (line number)
GR	Puts the Apple in lo res graphics mode. Erases the graphics screen.	GR
HLIN	A lo res graphic command that draws a horizontal line on the screen.	HLIN (column,column) at (row)
HOME	Clears the screen and sends the cursor to the upper left corner of the screen.	HOME
HTAB	Moves the cursor to a certain column on the screen.	HTAB (column)

Command, Statement, or Function	Purpose	Example
IF–THEN	Conditional transfer. If something, then do something else.	IF Z=10 THEN ? ''HI'' IF Z=11 THEN 500
INIT	Initializes a disk.	INIT HELLO
INPUT	Tells the computer to ask the user to type in input.	INPUT A, B$
INT	Tells the Apple to print a whole number (integer).	? INT(P) ? INT(4.69)
INVERSE	Causes the output to be printed in inverse.	INVERSE
LET	Assigns a value to a variable.	LET P=100
LIST	Tells the computer to list the statements of the program in memory.	LIST
LOAD	Loads a program from the disk.	LOAD (program name)
LOCK	Protects a program on a disk from being accidentally erased.	LOCK (program name)
NEW	Erases the memory.	NEW
NEXT	See **FOR–NEXT**	
NORMAL	Changes flash or inverse screen modes back to normal.	NORMAL
PLOT	Displays a point on the lo res screen.	PLOT (column,row)
PRINT	Tells the computer to print output.	PRINT A$ OR ? ''HI''
READ–DATA	Tells the computer to use data from the DATA statement for the value of certain variables.	READ Z$, X
REM	Allows remarks or documentation to be written into the program without affecting how the program runs.	REM ADDING NUMBERS

Command, Statement, or Function	Purpose	Example
RENAME	Changes the name of a program that is already stored on a disk.	RENAME (old name, new name)
RND	Tells the computer to pick a random number.	LET R=1+RND(1)*10
RUN	Executes the program in memory.	RUN
SAVE	Stores the program in memory on the disk.	SAVE (program name)
SPC	Moves the cursor over a certain number of spaces before printing.	? SPC(5); "HELLO"
SPEED	Changes the speed with which output is printed.	SPEED=(number between 0 and 255)
TAB	Moves the cursor to a certain column on the screen before printing.	? TAB(9); "HELLO"
TEXT	Returns the screen mode to direct mode from graphics mode.	TEXT
VERIFY	Checks a program to make sure it has been correctly copied and saved from the program in memory.	VERIFY (program name)
VLIN	A lo res graphic command that draws a vertical line on the screen.	VLIN (row,row) at (column)
VTAB	Moves the cursor to a certain row on the screen.	VTAB (row)

APPENDIX D

Reserved Words in Applesoft BASIC

You cannot use any of these words or abbreviations as variables.

A
AND
ASC
AT
ATN

C
CALL
CHR$
CLEAR
COLOR=
CONT
COS

D
DATA
DEF
DEL
DIM
DRAW

E
END
EXP

F
FLASH
FN
FOR
FRE

G
GET
GOSUB
GOTO
GR

H
HCOLOR=
HGR
HGR2
HIMEM:
HLIN
HOME
HPLOT
HTAB

I
IF
IN#
INPUT
INT
INVERSE

L
LEFT$
LEN
LET
LIST
LOAD
LOG
LOMEM:

M
MID$

N
NEW
NEXT
NORMAL
NOT
NOTRACE

O
ON
ONERR
OR

P
PDL
PEEK
PLOT
POKE
POP
POS
PRINT
PR#

R
READ
RECALL
REM
RESTORE
RESUME
RETURN
RIGHT$
RND
ROT=
RUN

S
SAVE
SCALE=
SCRN(
SGN
SHLOAD
SIN
SPC(
SPEED=
SQR
STEP
STOP
STORE
STR$

T
TAB(
TAN
TEXT
THEN
TO
TRACE

U
USR

V
VAL
VLIN
VTAB

W
WAIT

X
XPLOT
XDRAW

APPENDIX E

Lo Res Graphics Colors

0 Black
1 Magenta
2 Dark blue
3 Purple
4 Dark green
5 Gray 1
6 Medium blue
7 Light blue
8 Brown
9 Orange
10 Gray 2
11 Pink
12 Light green
13 Yellow
14 Aquamarine
15 White

GLOSSARY

A

Access: Getting information from a certain place.

Address: A label that tells where information is stored in the computer's memory.

Algorithm: A step-by-step method used to solve a problem.

Alphanumeric or string variable: A variable that stands for letters, numbers, or special characters. It is labeled like a numeric variable but must be followed by a dollar sign ($).

ALU (Arithmetic and Logic Unit): The part of the CPU (computer's "brain") where arithmetic and logical decisions are made.

Animation: Programming the computer to make graphics and figures move across the screen.

Apple II: A microcomputer made by Apple Computer, Inc.

Applesoft BASIC: A version of the BASIC computer language. It is the language used most widely with the Apple II.

B

BASIC: (Beginner's All-Purpose Symbolic Instruction Code): A fairly simple, popular computer language used mainly with microcomputers.

Binary machine language: A computer language made up of numbers and symbols. It is easy for computers, but difficult for people, to understand.

Booting: Putting DOS (Disk Operating System) into the computer's RAM (memory) is called booting the disk.

Brain: The central processing unit (CPU) and memory bank, which make up the internal circuitry of the computer.

BREAK message: The message that is displayed on the screen after the run of a program has been stopped. The message tells you at which program line the run was stopped or "broken."

Bugs: Mistakes found in a program that were made when the program was written.

Byte: The space it takes to store one character of information in the computer's memory.

C

Calculator: A mechanical or electronic device that carries out logical and arithmetic calculations. It is not as powerful, nor does it have as many capabilities, as a computer.

CAPS LOCK: A key found on the Apple IIe keyboard. When this key is depressed capital letters are printed. When this key is in its up position lower case letters are printed.

Cassette tape recorder: A device that can be attached to a computer to read and store programs to and from cassette tapes. Disk drives are often used in place of cassette recorders because they are faster and more reliable.

CATALOG: A disk command that causes the Apple to list all of the programs stored on the disk.

Cathode ray tube (CRT): A tube found in a television screen or monitor that allows the viewer to see images on the screen. Some mini- and microcomputers are called CRTs because of their screen.

Character: A letter, number, special symbol, or even a blank space.

Chip: A integrated circuit on a wafer slice that does certain jobs in the CPU. Different chips do different things, such as storing information in memory, sending messages, and doing arithmetic.

Closed apple: (also called solid apple) This key on the Apple IIe keyboard is used with the CONTROL and RESET keys to activate the system's built-in self-test.

Coding: Writing the BASIC program from a flow chart.

COLOR=: The BASIC command that tells the computer which color to use when in lo res graphics mode.

Complement: The opposite of a question or sign. For example, the opposite of > is <.

Computer-Aided Instruction (CAI): Using computers for teaching purposes.

Computer error: An error or problem in the computer system or hardware.

Computer language: Sets of words and symbols used to communicate with a computer.

Contents: The data stored at a memory address.

Control: The part of the CPU that makes sure all of the program statements are done in the right order.

CONTROL: This Apple IIe key functions in the same manner as the CTRL key on the Apple II.

Conversion equation: A program equation that converts one type of information to another.

Convert: Change one type of measurement or information into another type so a comparison can be made.

Counter: 1. A variable whose value increases consecutively in order to count how many times a certain instruction is done. A counter is often found inside a loop and controls how many times a loop is done. 2. A program technique that is used to keep track of the number of times a loop is done.

Counter-controlled loop: A programming loop that is done a certain number of times.

CPU (Central Processing Unit): The circuitry that makes up the "brain" of the computer.

CTRL: The CONTROL key. Holding this key down while pressing another key will cause a certain function to occur.

Cursor: The blinking square on the computer screen. It tells you that the computer is waiting for you to give it information or instructions. It also shows you where the next character will be printed on the screen.

Cursor control keys: The keys that allow the cursor to be moved around the screen without changing the writing on the screen or information that is in the memory.

D

Data: Information.

Data table: A table that helps the programmer identify the different variables that will be used in a program. This is important because it helps the programmer remember what the variables stand for and what they do in the program.

Debugging: The process of finding and correcting program bugs (errors).

Decision box: The diamond-shaped box in a flow chart that represents a decision to be made.

DELETE: The disk drive command that erases a program from the disk, and a key found on the Apple IIe keyboard which deletes mistakes if allowed by the program which is currently running.

Direct mode: The mode the Apple is in when it is first turned on. A command is immediately carried out by the computer after it is typed and RETURN is pressed. This mode is also called immediate mode or typing mode.

Disk: A flat, floppy object made of magnetic material on which programs and information are stored. The disk itself is covered by a flat plastic cover, which protects it.

Disk drive: A device used to store computer information and programs on floppy disks. It is also used to send information and programs from a disk to the computer memory.

Disk Operating System (DOS): Computer instructions that control the operation of the disk drive.

Double-alternative decision step: A situation in a flow chart in which there are two detours from a decision box.

Dummy data: Data that is read as a signal to the computer that the pointer is at the end of the data list.

E

E (Exponential) notation (also called scientific notation): A short way to represent very large or very small numbers.

Edit keys: The left and right arrow keys, which move the cursor to the left or right across the screen.

Edit mode: A screen mode that allows the user to move the cursor around the screen with the cursor control keys. Pressing ESC puts the Apple into edit mode.

END: The last statement in a program.

Error messages: The Apple's way of telling you that it did not understand the input.

ESC: The ESCAPE key. Pressing this key puts the Apple into the screen editing mode whereby the cursor can be moved around the screen without affecting the screen output or memory. There is also a way to clear the screen and send the cursor "home" using the ESC key.

F

Files: Lists of information that the computer has stored in its memory or on a disk. Sometimes programs are called files.

FLASH: The BASIC command that causes the screen output to flash.

Flow chart: A diagram that shows all of the steps of an algorithm in the correct order.

Flow diagramming: The process of illustrating parts of programs in a clear, step-by-step manner.

Format: A plan for the arrangement of something. Formatting screen output means writing programs so the screen output is arranged a certain way.

FOR–NEXT: Two BASIC programming statements that work together to allow counter-controlled loops to be made.

FOR–NEXT time loop: A FOR–NEXT loop with no body that is used to make the computer pause in the printing of output on the screen.

Function: An operation that the computer does automatically, like a built-in small program.

Function keys: Keys that control the mechanical operation of the keyboard such as [SHIFT], [CTRL], [ESC], [REPT], [RESET], and [RETURN].

G

GOTO: The BASIC statement that tells the computer to go to a certain location in the program. It is used to create a program jump or loop. It can be written as GOTO or GO TO.

GR: The BASIC command that puts the Apple into lo res graphics mode.

Graphic: A picture or design made by a computer.

Graphics mode: The screen mode that allows you to plot blocks and lines of color on the screen. Forty columns and rows of the screen are available for making graphics.

Graphics tablet: A device (peripheral) that can be attached to a computer to allow you to draw freehand graphics.

H

Hard copy: Output printed on paper by a printer.

Head: Part of the disk drive that reads and gets information from the disk.

Heading: Program output that labels or explains the information that follows it.

HLIN: A BASIC command for lo res graphics that draws a horizontal line on the screen.

Home: The upper left corner of the screen is called the cursor's home.

HOME: The BASIC command that clears the screen and sends the cursor to the "home" position. This command may be used in either direct mode or programming mode.

Horizontal: A horizontal line goes across the screen from left to right.

HTAB: The BASIC command that moves the cursor to a certain column on the screen.

I

IF–THEN: A BASIC program statement used to make comparisons and decisions.

Illegal Quantity Error: An error that indicates that a number too big or too small was used in a command or program statement.

Initialize: Setting up a new blank disk so programs can be saved on it.

I/O Pathways (Input/Output Pathways): Channels with which the computer transfers information and instructions.

Input: Any information that is put into the computer.

INPUT: A BASIC program statement that allows data to be typed into the program while the program is running.

Input variables: Variables that the programmer already knows the value of before the program is run on the computer.

INT: The program function used to create whole numbers (integers) in a program.

Integers: Whole numbers (no fractions or decimals).

Integer BASIC: The first type of BASIC that was written for the Apple microcomputer. Most programs nowdays are written in Applesoft BASIC.

Interactive program: A program that allows you to interact with the computer by typing data into the program while the program is running. In this type of program, the computer usually asks questions, and you must type in the answers.

Inverse: Reversed in order or nature.

INVERSE: The BASIC command that causes screen output to be printed in inverse (black characters on a white background instead of white characters on a black background).

K

K: Kilobyte.

Keyboard: The part of the computer used to type in information (input) to the computer memory.

Keyboard memory: Memory that stores characters typed on the keyboard. The characters are transferred to RAM when RETURN is pressed.

Kilobyte: One thousand bytes. The quantity by which computer memory is measured.

L

LET: The program statement that assigns a value to a variable.

Letter keys: The keys that cause a letter of the alphabet to be typed on the screen.

Line number: Any number between 1 and 63999 that comes before a program statement.

LIST: The BASIC command that causes the computer to list all of the statements of the program that is currently in memory.

Lo res graphics (low resolution graphics): Using a low resolution screen mode to plot colored blocks and lines on a 40-row by 40-column screen.

LOAD: The BASIC command used to bring programs from a disk into the computer's RAM (memory).

LOCK: The disk drive command that protects a program on a disk from being accidentally erased.

Locked: Locking a program on a disk keeps it from being accidentally erased.

Loop: A program situation whereby a sequence of steps are repeated. A loop is represented in a flow chart by an arrow that shows a jump to another location.

Loop body: The program statements between the FOR and NEXT statements in a loop.

M

Memory: A part of the CPU that is used for storing data—or information—and program instructions.

Menu: A program on a disk that organizes the catalog listing of programs by the languages in which they were written.

Microcomputer: A small, portable computer that is inexpensive and easy to use.

Modem: A device (peripheral) that can be attached to a computer to allow communication between computers in different locations through the telephone lines.

N

NEW: The BASIC command that erases or clears the computer's memory.

NORMAL: The BASIC command that changes a flashing or inverse screen mode back to the normal direct mode.

Number keys: The keys that cause the numbers (0-9) to be printed on the screen.

O

Open apple: A key found on the Apple IIe keyboard which can be used as paddle control #0 or as a system reset with the CONTROL and RESET keys.

Out of Data Error: An error message caused by a READ–DATA statement with which the computer is telling you that there is no more data for the READ statement to read.

Output: Information that the computer puts out.

Output variables: Variables that will hold the answers that the computer calculates in the program. The values of these variables are not known until the program has been run.

P

Paddles and joysticks: Game control devices that can be attached to the computer.

Peripheral: A piece of equipment that can be attached to a computer to do a certain job.

PLOT: The BASIC command that displays a point on the lo res graphics screen.

Pointer: An electronic device that marks the location of the data being read from a data list.

Powers (also called exponents): Using exponentiation in mathematics.

PRINT: The BASIC statement that tells the computer to print something on the screen. The computer will print information inside quotation marks exactly as they appear. A question mark (?) may be used as a shortcut instead of typing the word *PRINT*.

Printer: A device that can be attached to a computer to print output on paper.

Print zones: The three sections that make up the Apple's screen area.

Processing box: The rectangular-shaped box in a flow chart that represents something to be done.

Program: The set of instructions written in a computer language that tells the computer what to do.

Programmer: A person who writes computer programs.

Program documentation: A good programming technique in which REM (REMARK) statements are used to note and clarify what is happening in a program.

Program errors: An error in a computer program.

Programming mode: A state of computer operation in which statements typed on the computer's screen are placed in the RAM (memory) when RETURN is pressed. These statements must have line numbers and are stored in memory as part of a program until the **RUN** command is typed.

Prompt: The symbol that appears at the beginning of new screen lines after RETURN is pressed. It tells which computer language the computer is operating in.

R

RAM (Random Access Memory): A type of computer memory. When the computer is first turned on, RAM is empty and the user may store programs and information there. When the computer is turned off, all information and programs in RAM are lost because the RAM is erased.

Random numbers: Lists of numbers that are in no particular order and have no particular purpose.

READ–DATA: Two BASIC programming statements that work together to cause the computer to place data in a program as it is typed on the keyboard. This feature allows you to use the same program over and over again with different data.

REM: The REMARK statement, which allows comments to be placed in the program for program documentation. These statements are ignored by the computer and are used only to note what is happening in the program.

RENAME: The disk drive command that changes the name of a program already stored on a disk.

REPT: The REPEAT key. Holding this key down while pressing another key will cause repeated keystrokes to occur.

Reserved words: Some BASIC commands and statements are reserved. This means that you cannot use these words or even the first two letters as variables in a program. See Appendix D for a list of the reserved words.

RESET: This key, when pressed, stops any computer activity and immediately returns control to you in direct mode. Sometimes CTRL and RESET must be pressed together to make this happen.

RETURN: The key that makes the cursor move to the next screen line and enters any information from the previous line into memory (RAM).

RND: The program function used to create random numbers in a program.

RUN: The BASIC command that tells the computer to "do" the program.

Run: What happens when the computer "does" a program.

S

SAVE: The disk command that copies the program in RAM and transfers it to the disk to be stored.

Screen: The display portion of a television or monitor in which information from the computer (output) and instructions typed on the keyboard (input) are shown.

SHIFT: The key that when pressed while holding down another key will print the symbol at the top of the key that is being held down.

Simulation: A computer program that imitates a real-life situation.

Single-alternative decision step: A situation in a flow chart in which there is one detour from a decision box.

Sorting algorithm: An algorithm that can sort and alphabetize a list of more than two words.

SPC (space-over function): The BASIC function that moves the cursor a certain number of spaces before printing.

Special symbol keys: Symbol keys used for doing math and punctuating sentences. For example +, ! , = , # , etc.

SPEED = : The BASIC command that changes the speed at which output is printed on the screen.

Square root: Using the square root function in mathematical equations.

STEP: A program statement that allows counter-controlled loops to be counted in a certain pattern (for example, by fives, by tens, and even in reverse order).

Style: Using a variety of techniques to develop easy-to-read programs.

Syntax error: A type of error message that tells you there is a word spelled wrong, a word that the computer does not recognize (not a BASIC word), or incorrect punctuation.

System master: A disk that comes with the Apple and contains the DOS program plus many other helpful programs.

System reset: (see warm boot)

T

TAB: A BASIC program function used to control screen output, and an Apple IIe key that, when pressed, moves the cursor right eight spaces if the program being run allows this.

TEXT: The BASIC command that returns the screen mode from lo res graphics mode to direct mode.

Text window: The eight rows (four screen lines) set aside at the bottom of the lo res graphics screen for text. When you type, the input will appear in the text window.

Trace: The act of working through a program in the same way that the computer would to see exactly how the program works.

Truncate: To remove any numbers to the right of the decimal point, thus changing the number from a decimal to an integer.

U

User error: An error you make when you make a mistake or forget to communicate with the computer in BASIC.

User-friendly: A program that is easy and enjoyable to use.

V

Variable: A name given to a value that is also the memory address of where the value is stored in memory. A variable's value can be changed (varied).

Vertical: A vertical line goes up and down the screen from top to bottom.

VLIN: A BASIC command for lo res graphics that draws a vertical line on the screen.

VTAB: The BASIC command that moves the cursor to a certain row on the screen.

W

Warm boot: Restarting the Apple IIe computer system while the power is still on. This is accomplished by pressing open apple, CONTROL, and RESET together.

INDEX

MORE!

An Apple in the Classroom Activity Workbook

Complete with 91 tear-out worksheets to go with each chapter of the student text, *AN APPLE FOR KIDS,* this workbook provides practice and reinforcement for skills learned. Most of the activities can be done as seat work without the computer, and students will only need the computer to check their work.

ISBN 0-88056-120-3 91 worksheets 175 pages/170 illustrations

$5.95

An Apple in the Classroom Teacher's Guide

The teacher's guide features 91 worksheets, complete with answers. Additional information and hints for teachers are provided. Information on how to convert the material to other brands of microcomputers and how to use the curriculum is also discussed.

ISBN 0-88056-118-1 $14.95 90 pages

NOTE: For every 25 copies of AN APPLE FOR KIDS student texts ordered, receive one teacher's guide free!

BILL TO:

Name: _____

School: _____

Address: _____

City, State Zip: _____

SHIP TO: (if other than bill to)

Name: _____

School: _____

Address: _____

City, State, Zip: _____

Phone number _____ Date _____ P.O. No. _____

_____ An Apple for Kids, Student Text $9.95
ISBN 0-88056-119-X

_____ An Apple in the Classroom Activity Workbook $5.95

_____ An Apple in the Classroom Teacher's Guide $14.95

_____ Check here if your order is over 25 copies of *An Apple for Kids* to receive a free copy of the teacher's guide.

Mail order to: **dilithium Press**
P.O. Box 606
Beaverton, OR 97075

To expedite your order, phone 800-547-1842
or (inside Oregon) 646-2713

PLACE
STAMP
HERE

dilithium Software
P.O. Box 606
Beaverton, OR 97075

An Apple For Kids is written by a teacher who wants to teach enthusiastic kids computer operation and programming in BASIC. Using an individualized, self-paced approach, this book encourages kids to be creative programmers as well as learn good programming techniques.

Full of illustrations and activities to make the learning process fun, *An Apple For Kids* is written for the 3rd to 8th grade student. It focuses on problem solving, higher order thinking skills and creativity. Approximately 80 activity worksheets are included to make this a fresh, instructive and fun approach to learning programming.

Teachers manuals and student workbooks are available to accompany An Apple For Kids.

0 45078 00995 56119

ISBN 0-88056-119-X

>>$9.95

NOW 4/95

DILITH. 3/84

dilithium Press