THE APPLESOFT
TUTORIAL
>LIST
100 REM SET GRAPHICS MODE
110 GR
120 REM CHOOSE A RANDOM
130 COLOR= RND (16)
140 REM PICK A RANDOM POS.
150 X= AND (40)
160 Y= AND (40)
170 REM PLOT THE
180 PLOT X,Y
190 REM DO IT ALL AGAIN
200 GOTO 130
apple II
apple computer inc.

NOTICE

Apple Computer Inc. reserves the right to make improvements in the product described in this manual at any time and without notice.

DISCLAIMER OF ALL WARRANTIES AND LIABILITY

APPLE COMPUTER INC. MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL OR WITH RESPECT TO THE SOFTWARE DESCRIBED IN THIS MANUAL, ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. APPLE COMPUTER INC. SOFTWARE IS SOLD OR LICENSED "AS IS". THE ENTIRE RISK AS TO ITS QUALITY AND PERFORMANCE IS WITH THE BUYER. SHOULD THE PROGRAMS PROVE DEFECTIVE FOLLOWING THEIR PURCHASE, THE BUYER (AND NOT APPLE COMPUTER INC., ITS DISTRIBUTOR, OR ITS RETAILER) ASSUMES THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR, OR CORRECTION AND ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES. IN NO EVENT WILL APPLE COMPUTER INC. BE LIABLE FOR DIRECT, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE SOFTWARE, EVEN IF APPLE COMPUTER INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF IMPLIED WARRANTIES OR LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

This manual is copyrighted and contains proprietary information. All rights are reserved. This document may not, in whole or part, be copied, photocopied, reproduced, translated or reduced to any electronic medium or machine readable form without prior consent, in writing, from Apple Computer Inc.

©1979 by APPLE COMPUTER INC.
1Ø26Ø Bandley Drive
Cupertino, California 95Ø14
(4Ø8) 996-1Ø1Ø

All rights reserved.

Reorder APPLE Product #A2LØØ18
(Ø3Ø-ØØ44-ØØ)

THE APPLESOFT TUTORIAL

Based on the Apple II Basic Programming Manual by Jef Raskin.
Rewritten for Applesoft by Caryl Richardson.

TABLE OF CONTENTS

WELCOME

CHAPTER 1
GETTING STARTED

2 Introduction
3 What you will need
4 Hooking up the TV
4 Plugging in the game controllers
4 The Disk II
5 The cassette recorder
5 The Apple keyboard: the RESET, SHIFT and ESC keys
9 Keyboard notation
10 Control, and other characters: the CTRL and REPT keys
11 Setting the tape recorder
13 The usual procedure for loading tapes
14 Listening to a computer tape: a helpful hint
15 Using a disk drive
17 The Menu
17 Stopping the computer
18 Setting the TV color
19 Playing Little Brick Out

CHAPTER 2
BEGINNING APPLESOFT

22 A first look at the PRINT statement
25 Applesoft's format for numbers
26 More about RETURN
27 Easy editing features: the arrow keys
29 Putting colors on the screen: GR, TEXT, COLOR= and PLOT
31 PLOT error messages
32 Drawing lines
34 The game controls: PDL
35 Pigeonholes : an introduction to variables
39 Precedence among arithmetic operators, or who's on first?
41 How to avoid precedence

CHAPTER 3
ELEMENTARY PROGRAMMING

44 Deferred execution: NEW, LIST, RUN and HOME
48 Elementary editing: DEL
5Ø Elementary aerobatics: GOTO loops
5Ø Some more things that make life easier: more editing tips
52 The moving cursor: editing with the ESC key
53 A word about learning Applesoft BASIC
54 An accident about to happen
55 The truth: arithmetic and logical assertions
59 Order or precedence for operations
59 The IF statement
6Ø Saving programs on diskette: SAVE, CATALOG, RUN and LOAD
61 Saving programs with a cassette recorder: SAVE
62 More graphics programs: REM
64 FOR/NEXT loops
67 A wrong program
67 A last example of nested loops
68 Getting flashy: INVERSE, FLASH and NORMAL
68 PRINTs charming: comma, semi-colon, TAB, HTAB and VTAB

CHAPTER 4
LOTS OF GRAPHICS

74 Talking to a program on the RUN: INPUT and a bouncing ball
78 Off the walls: a program with lots of bounce
79 Making sounds: PEEK(-16336)
81 Noise for the bouncing ball
81 For higher notes
82 Random numbers: RND and INT
84 Simulating a pair of dice
85 Subroutines: drawing horses using GOSUB and RETURN
87 Traces: TRACE, NOTRACE and END
89 A better horse-drawing subroutine
91 High-resolution graphics: HGR, HCOLOR= and HPLOT

CHAPTER 5
STRINGS AND ARRAYS

1ØØ Stringing along: LEN, LEFT$, RIGHT$, MID$ and CLEAR
1Ø5 Concatenation got your tongue?: putting strings together
1Ø5 More string functions: VAL and STR$
1Ø8 Introducing arrays: DIM
11Ø Array error messages
111 Conclusion

APPENDICES

114 Appendix A: Summary of Commands

126 Appendix B: Reserved Words in Applesoft

128 Appendix C: Editing Features
128 Left and right arrow keys
128 Pure cursor moves
129 Deleting program lines
129 Clearing the screen
130 Summary of edit features

131 Appendix D: Firmware Applesoft versus Cassette or Diskette Applesoft
131 Introduction
132 General discussion
133 An important note
133 Part 1: The Applesoft II firmware card
134 Part 2: Diskette Applesoft
136 Part 3: Cassette tape Applesoft
137 Part 4: Differences between diskette or cassette Applesoft and firmware Applesoft
140 Part 5: Memory locations used by DOS and by Applesoft BASIC

143 Appendix E: Error Messages

147 Appendix F: The Old Monitor ROM
147 Using the old monitor ROM
148 Recovering from accidental RESETs

WELCOME

This manual is designed for people who want to learn to program in Apple's Applesoft BASIC. With this manual, and an Apple computer, and a bit of your time and attention, you will find that there is nothing difficult about learning how to program a computer. At first, as with anything new, programming will be unfamiliar, but this manual was designed to alleviate any apprehension you might have. First of all, there are no hidden secrets that you have to know before you can read this manual. Everything is revealed, and only one thing at a time is explained. If you start at the beginning, try everything as it comes along, and make up your mind to take your time, it is pretty much guaranteed that you will learn how to program.

The real secret is taking your time and trying everything. You cannot learn programming by merely reading this or any other book. Like learning to ride a bicycle or play the fiddle, you must learn by doing. You must make mistakes and correct them, and not feel too bad when you do make mistakes.

If you already know how to program, a quick run through this book will make you familiar with the features of Applesoft BASIC. We suspect that you will be quite impressed with the ease of doing high resolution graphics and using the other features that make the Apple a fine computer for a wide variety of applications.

This book is a tutorial manual. For reference purposes, the companion volume, the Applesoft BASIC Programming Reference Manual (Apple product number A2LØØØ6) should be used. Its extensive indices, handy reference card, and many technical details make it the book you will use after you have learned the basics from this volume. One last thing--if you have read the Apple II BASIC Programming Manual (Apple product number A2LØØØ5X) then this manual may seem a bit familiar. If you think so, you are right. It closely follows that manual, which (we are happy to report) has been very well recieved by customers and reviewers alike. We hope you enjoy using this manual as much as we have enjoyed writing it.

CHAPTER 1
GETTING STARTED

2 Introduction
3 What you will need
4 Hooking up the TV
4 Plugging in the game controllers
4 The Disk II
5 The cassette recorder
5 The Apple keyboard: the RESET, SHIFT and ESC keys
9 Keyboard notation
10 Control, and other characters: the CTRL and REPT keys
11 Setting the tape recorder
13 The usual procedure for loading tapes
14 Listening to a computer tape: a helpful hint
15 Using a disk drive
17 The Menu
17 Stopping the computer
18 Setting the TV color
19 Playing Little Brick Out

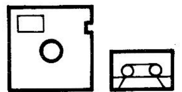
INTRODUCTION

This manual will show you how to plug in your Apple (easy) and will be a guide as you learn to program it (also easy). If you are an Old Hand at programming, you will find some new features and conveniences in Applesoft BASIC that make programming a lot more fun. If you are a Newcomer to programming, you will also find many features and conveniences in Applesoft BASIC that make programming a lot of fun. But, if you are a Newcomer, be warned that programming, though not difficult, can only be learned by doing. More will be said on this topic later, but remember--this is a book to be used, not merely perused.

If you purchased your Apple from an authorized Apple dealer, the dealer will be willing to let you set your Apple up in the shop to make sure you know how to set it up at home. If you received it as a gift or through the mail, it is not difficult to hook up--it is as easy as setting up a stereo system, and no technical knowledge is needed at all.

If you have not already done so, please take a few minutes to complete and mail your OWNER/WARRANTY REGISTRATION CARD. This Registration Card will register your Apple with the factory, give you membership in the Apple Software Bank, and include you in our list of Apple owners. If you don't send us this card you will not receive any newsletters, information about new accessories for your Apple, nor any of the other information that is frequently mailed to Apple owners. So please mail in the completed card.

The Apple described in this manual has the Applesoft BASIC computer language and the Autostart ROM installed on the main board. If your system differs from this one, for instance, if you have an Apple II with Integer BASIC and the Old Monitor ROM, you can find the information you need in one or more of the appendices in the back of this book. If you have Applesoft on cassette or diskette watch for the



symbol. This symbol indicates information that is of special interest to cassette and diskette Applesoft users. If these sections apply to you, be sure to read them carefully. If you don't, you may lose your program or part of the Applesoft program itself.

Another symbol to watch for is the

The purpose of this symbol is to alert you to an unusual Applesoft feature. The situation described may cause you to lose your program.

WHAT YOU WILL NEED

This manual was in the accessory box. This box should also contain

1. The power cord (the cord that plugs into the outlet on the wall).
2. A set of two game controllers (the black boxes with buttons and knobs, connected with a cord).
3. A cable to connect the Apple to a tape recorder. This cable has two plugs on each end.
4. Some cassette tapes. These tapes contain programs for the Apple.

In addition to the Apple itself and the contents of the accessory box, you will need two more items chosen from the options below (none of these items are supplied).

1. You will need one of the following items (it's useful to have both, but only one is necessary).
   a. A cassette recorder.

   OR

   b. The Apple Disk II disk drive with a controller card.

2. You will also need one of the following items:
   a. A color TV monitor and a cable that has a phono plug (also called a male RCA-type connector) at one end and something to match the monitor at the other end. The dealer that sells you the monitor can supply the cable.

   OR

   b. An ordinary home color TV and an "RF Modulator" with the connecting cables. The RF Modulator changes the signal put out by the Apple so that it matches what your TV expects. A number of Modulators are available. There is one made especially for the Apple called the SUPERMOD II. Your computer dealer can probably sell you one, or, if not, it can be ordered from

      M&R Enterprises
      P.O. Box 61Ø11
      Sunnyvale, CA 94Ø88

The Modulator comes with instructions on how to hook it up. Your TV's ability to receive normal programs will not be diminished (or enhanced) by having the Apple hooked up to it.

A black and white monitor or TV will work fine, but will not let you take advantage of Apple's ability to generate color pictures. Colors described in this manual will appear as different shades of grey on a black and white monitor or TV.

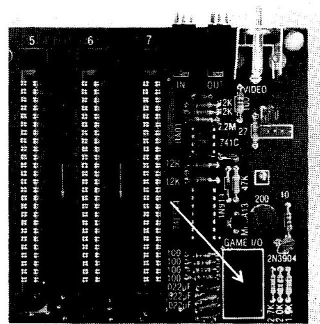
HOOKING UP THE TV

If you have a color (or black and white) monitor, just connect the appropriate cable from the jack marked VIDEO OUT (on the rear of the Apple) to the input of the monitor.

If you have an ordinary TV, you will have to install an RF modulator. Open the top of the Apple by pulling straight up on the back of the lid using both hands, one on each side. Then install the modulator following the directions that come with it.

PLUGGING IN THE GAME CONTROLLERS

With the lid open, plug the controllers' rather delicate plug into the GAME I/O socket located in the right-rear corner (front view) of the Apple board. Be very careful and make sure that all the pins go into the socket. The plug's white dot should be toward the front (keyboard end) of the computer.



THE DISK II

If you have a disk drive, unpack it carefully. Then read the preface and the first chapter (pages 2 through 8) of the Disk Operating System (usually called DOS) manual that came in the Disk II package. Those pages will give you complete instructions on how to set up your disk drive.

THE CASSETTE RECORDER

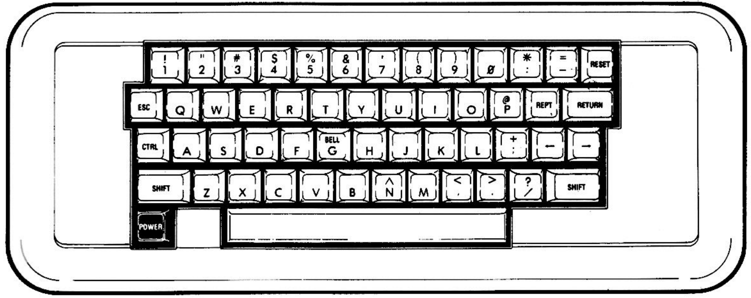
(if you are not using a disk drive, or
if you are going to use both recorder and disk)

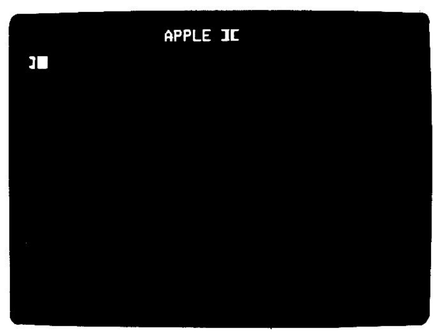
Use the supplied cable (the one with two plugs on each end) to connect the Apple to your cassette tape recorder. Connect one black plug to the MIC or MICROPHONE jack on the recorder, and the other black plug (on the opposite end of the cable) to the jack on the back of the computer marked CASSETTE OUT. Connect the grey plug on the recorder end of the cable to the recorder's EAR or EARPHONE or MON or MONITOR jack on the recorder (different brands use different words). Connect the grey plug on the computer end of the cable to the jack marked CASSETTE IN. "OUT" means "out of the computer" and "IN" means "into the computer." All that remains is to plug the cassette recorder's power cord into a wall outlet, and it will be ready to use.

Now close the top of the Apple. Plug the Apple end of the computer's power cord into the Apple (on the rear of the Apple, next to the power switch), and the other end into a three-prong grounded wall outlet. Now the Apple is completely set up, and you have only to read on to begin exploring the fascinating world of personal computing.

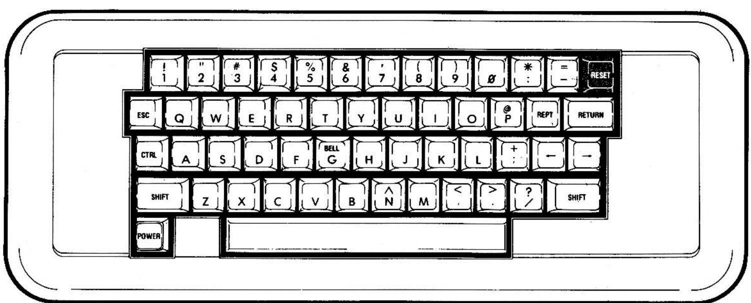
THE APPLE KEYBOARD

The first thing to do, now that all the connections have been made, is to turn the Apple on. The switch is on the back of the computer next to where the power cord plugs in. Push it into the upward position. You will be rewarded by the "POWER" light at the bottom of the keyboard coming on. The POWER light is not a key, and cannot be depressed. The title "APPLE II" should also appear at the top of the screen along with a] and a blinking square called the "cursor" to the far left.





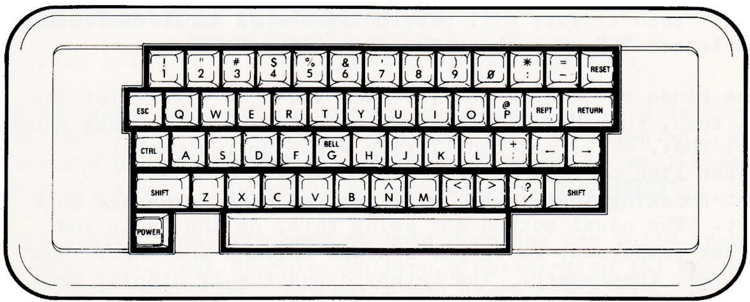
If your screen display doesn't conform to the description, don't worry. If you have the Applesoft II Firmware Card that plugs into the Apple's main board (Apple Part Number A2BØØØ9X), make sure the switch on the back of the card is in the up position. Then turn your Apple off and turn it on again. If your screen display still doesn't look right, or if you don't have Applesoft on a card, press the key marked RESET in the upper right-hand corner of the keyboard. The Apple should go "beep" when the RESET key is released.



If your Apple doesn't seem to be responding correctly to your instructions (you'll find out what correct responses are as you become familiar with this manual), a press of the RESET key will usually remedy the problem. If that doesn't work, turning the Apple off and then turning it back on again will probably do the trick.

If you have a disk drive, turning your Apple on will give the following results. A few clacking noises will come from it, followed by a soft whirring sound, and a red light labelled "IN USE" will come on. The disk drive will whir and whir until it seems that it will never stop. As a matter of fact, it won't: until you stop it by pressing the RESET key. Do that now. The title "APPLE II" disappears, and the prompt and cursor appear at the bottom left of the screen.

Study the keyboard. If you are familiar with standard typewriters, you will find a few differences between the Apple keyboard and a typewriter keyboard. First, there are no lower case letters. You can get only capital letters on the Apple. This is all you need for programming in Applesoft BASIC.



Using the diagram, locate the two SHIFT keys on the keyboard. The reason the keyboard has the SHIFT keys is to allow for nearly twice as many characters with the same number of keys. A keyboard with a separate key for each character would be very large, making it hard to find any desired key.

If you press a key which has two symbols on it, the lower symbol will appear on the screen. If you press the same key while holding down either of the SHIFT keys, the upper symbol will appear on the screen. You will find that the SHIFTed comma and the SHIFTed period are < and > respectively. You will also find other symbols on the Apple keyboard that are not on a standard typewriter. Feel free to try operating any of these keys.

If there is no upper symbol on a key, then holding down the SHIFT while the key is pressed has no effect. There are two exceptions: the M and the G .

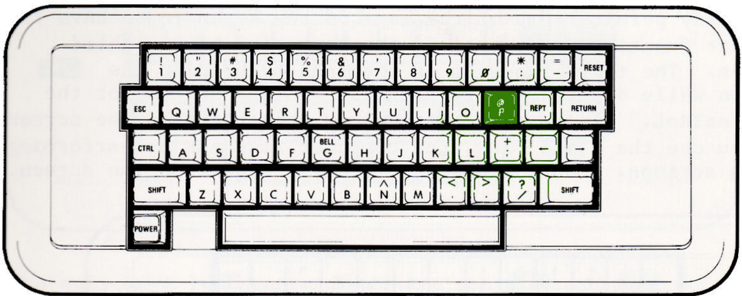
The SHIFTed M key gives a right hand square bracket (]). The G key has the word "BELL" above the "G". But SHIFT G does not put a bell on the screen, it just puts a "G" there. The meaning of the word "BELL" on the G key will be explained later.

An important difference between using the Apple keyboard and most typewriters is that you cannot employ a lower case "L" for the number "1". Of course, there is no lower case "L" on the Apple, but some typists will have to break the habit of reaching for the letter "L" when they mean the number "1".

When the Hindu mathematicians invented the open circle for the numeral zero, they didn't use the Roman alphabet. So they chose a symbol that, while not conflicting with their alphabet, looks just like our letter "O". The computer (and any straight-thinking individual) will want to keep zeros and oh's distinct. The usual method for doing this, on the Apple and many other computers, is to put a slash through the zero. Now you can tell them apart. The keyboard and the TV display both make the distinction clear. Try them.

After a bit of typing, the screen tends to get full of stuff. To clear the screen, you need to use the key marked ESC . ESC stands for the word "ESCape." The ESC key does not show up on

the Apple's screen. Press ESC , and then type an "at" sign (@) by holding down either SHIFT key and pressing the key marked "P". Notice that the ESC key, unlike the SHIFT key, does not need to be held down while typing another key. You have to operate three keys to clear the screen. First press ESC and release it. Then, while holding down SHIFT , press P. Instant gratification: the contents of the screen promptly disappear.



KEYBOARD NOTATION

At this point we will introduce a simple notation.

As you have seen, when a key is to be pressed, such as the key for the letter "H", that key's symbol will be shown: H. To indicate pressing several keys in succession, we will simply list the keys in the order to be pressed: H E L L O

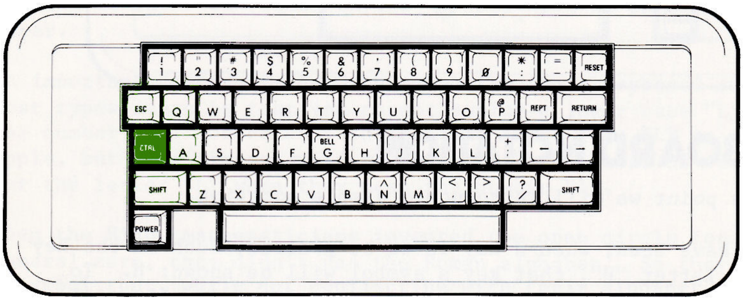
On occasion, you will need to hold down one key while pressing another key. For example, to type a dollar-sign ($) you must hold down the SHIFT key while you press the 4 key. Whenever this dual action is required, we will show the symbols for both keys, one above the other.

The upper key is to be held down while the lower key is pressed. Here's how to clear the screen, using the new notation:

Try it.

CONTROL, AND OTHER UNSAVORY CHARACTERS

When you press the 5 key, the numeral 5 appears on the TV screen. You probably believe this is true, but try it anyway. If you hold the SHIFT key down while pressing the 5 key, a percent sign (%) should appear on the screen. Does it? The SHIFT key permits some of the keys on the keyboard to have two different functions. Several of the keys also have a third function. The third function is obtained by holding the CTRL key down while other keys are pressed. "CTRL" stands for the word "ConTRoL." Instead of putting new characters on the screen when you use the CTRL key, the computer responds by performing certain actions. Control characters never appear on the screen.



Hold the CTRL key down and press G

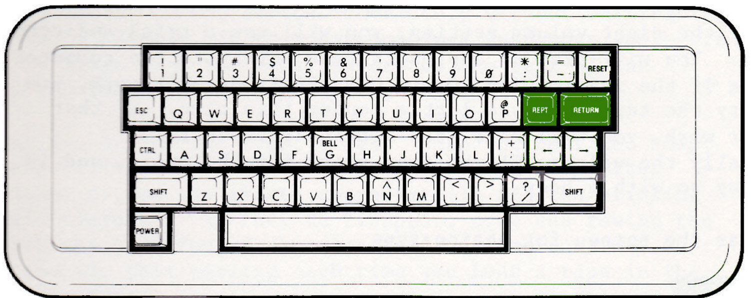
It doesn't go "ding", but it does go "beep." Whenever the computer wishes to call your attention to something, it will sound the beeper. CTRL G is called "BELL" for historical reasons: the present keyboard design is based on that of the Teletype. On that venerable machine, CTRL G rings a real bell.

Another key that is not usually found on typewriters is the REPT which stands for "REPeaT." Holding down the REPT key while you press any other key just makes that key's character appear repeatedly on the sceen. You must first press and hold the key for the character you wish repeated and then hold down the REPT key. Experiment with it.

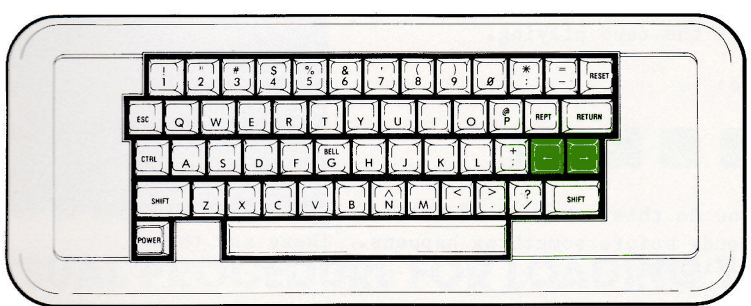
There is also a key marked RETURN on the keyboard. On machines in the past, this was the "carriage return" key. On the Apple, it causes the blinking cursor to "return" to the screen's left edge, but it is also a special message to the computer. More about this message later. If you happen to press RETURN , you will sometimes get a "beep" and the message

?SYNTAX ERROR

will appear on the screen. For the time being, ignore this message.



The only keys left unmentioned are the ← and → keys. They move the cursor to the left and the right. They will be explained more fully later. Test out these keys and any others you can find. There is nothing you can do by typing at the keyboard that can cause any damage to the computer. Unless you type with a hammer. So feel free to experiment. With your fingers.



SETTING THE TAPE RECORDER

(if you are not using a casette recorder,
skip to the section called "USING A DISK DRIVE".)

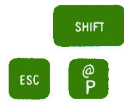
Now press the RETURN key. The right hand square bracket] and the blinking cursor that show on the screen's left edge let you know that you are "in Applesoft" or have Applesoft "up" (as they say). Now you are ready to set the volume control on the tape recorder.

When you play a tape recorder, it is usually with the intent of making sounds that you can hear. If it is too soft, you miss some of the words or music. If it is too loud, it is annoying.

When you play the tape recorder into the Apple, it is with the intent of putting the tape's information into the computer. If the volume setting is too soft, the Apple will miss some of the information, and it will complain by giving an error message. If the volume setting is too loud, the Apple will also complain.

To find the right volume setting, you will use a trial-and-error method. You will play an Applesoft tape softly to the computer and see if the information got in OK. If it doesn't work, you will try the tape again, a little louder this time. If that doesn't work, you will make it a little louder still. Eventually the volume will be just right for the Apple, and it will say so with a beep.

To clear the screen for action type



Place the tape marked COLOR DEMOSOFT into your recorder. For each position of the volume control you are going to do the following:

1. Rewind the tape to the beginning.

2. Start the tape playing.

3. Type:

When you do this, the cursor will disappear. It may take up to 15 seconds before something happens. There are these possibilities:

a. The message ?SYNTAX ERROR appears.

b. Nothing at all happens.

c. The message ERR or ERRERR appears (with or without a beep).

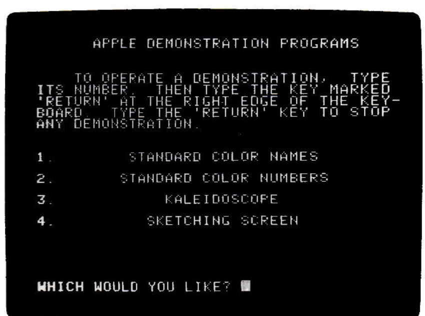
d. The computer goes "beep" and nothing appears.

In case a, do not reset the volume control, but go back to step 1 where you rewind the tape.

In cases b and c, make sure you waited for 15 seconds before giving up. If there is no prompt character or cursor, and the Apple does not respond to its keyboard, press RESET , set the volume control a bit higher and go back to step 1. Once in a great while the LOAD command may not work properly, and the cursor will appear on the screen immediately without waiting for the tape to be LOADed. If this happens just turn your Apple off and then on again with the power switch on the rear of the computer, and then try LOADing the tape again.

In case d, you are on the right track. When you hear the beep, wait another fifteen seconds. Either you will get an error message (case c), or the prompt character (]) and the blinking cursor will reappear. If they do reappear, stop and rewind the tape. Mark the position of the recorder's volume control, so that you can use this setting each time you LOAD a tape in the future. Then type

The screen should look like this:



THE USUAL PROCEDURE FOR LOADING TAPES

(once the recorder's volume control has been set correctly)

1. Rewind the tape.

2. Start the tape playing.

3. Type LOAD

   After you press RETURN the cursor will disappear. Nothing happens from 5 to 2Ø seconds, and then the Apple beeps. This means that the tape's information has started to go into the computer. After some more time (depending on how much information was on the tape, but usually less than a few minutes) the Apple beeps again and the prompt character and the cursor reappear.

4. Stop the tape recorder and rewind the tape. The information has been transferred, and you are finished with the tape recorder for the time being.

5. Type RUN and press RETURN , and your program will begin to execute.

If your Apple is in the Applesoft BASIC computer language, the tape you are LOADing must be in Applesoft too. Trying to LOAD a tape in the wrong computer language gives results that are pretty much unpredictable. Strange error messages and odd characters may appear on your TV screen, you may loose keyboard control, or any number of other odd things can happen. If this happens to you, turn your Apple off and then back on again to get everything back to normal.

Computerniks use many different words to describe the process of taking information from a tape and putting the information into the computer. The computer is said to "read" (pronounced "reed") the tape. The information on the tape is said to be "entered" or "read" (pronounced "red") into the computer. The act of reading a tape is also called "loading" a tape into the computer and the information on the tape is said to be "loaded into" the computer. All these expressions are ways of saying the same thing.

A HELPFUL HINT

What is it that the computer finds so interesting about these tapes? Listen to one of them. It's not music to your ears. Yet you can recognize some of the sounds the computer listens for. The information starts with a steady tone. Then there is a short "blip" followed by more of the steady tone. The tone is at 1ØØØ cycles per second. This pitch is just below the C two octaves above middle C. After the tone comes a burst of sound rather reminiscent of a rainstorm.

When you are used to the sound of a good tape, you can quickly check a tape by ear to see if it is a computer tape or not. If you can tell what the tape contains by listening to it, you are a mutant, and will go far in the computer world.

USING A DISK DRIVE

(Skip this section if you are not using a disk drive.)

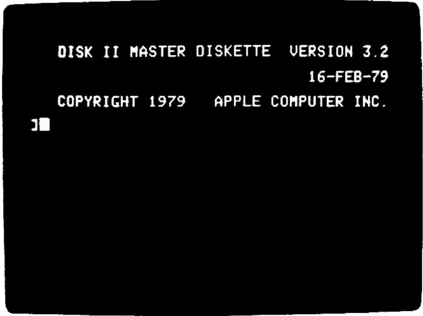
A disk drive is much quicker and easier to use than a cassette recorder, however, diskettes and disk drives are delicate creatures, and some care must be taken to protect them. You will find information on their care and feeding on pages 5 through 6 in your DOS manual in the section called CARE OF THE DISK II AND DISKETTES. Read that section carefully if you haven't already.

The last section in the first chapter of the DOS manual is called INSERTING AND REMOVING DISKETTES. Get the System Master diskette from its package and insert it with the label facing up and the oval cutout toward the back of the disk drive, as described in the DOS manual.

One of the features that make the Disk II so easy to use is its ability to store and retrieve several different groups of information. The groups of information are filed on the disk under names called file names. A program that keeps track of addresses, for instance, might be called ADDRESSES on the diskette.

The programs that keep track of files, save and retrieve them, and do lots of other housekeeping tasks are what make up the Disk Operating System or DOS. The process of adding the DOS capabilities to Applesoft (or to any other language used by your Apple) is called "booting DOS" or "booting the system".

There are several ways to boot DOS. One way is to simply turn your Apple off and turn it on again. The Disk drive's red "IN USE" light will come on again, and the Disk II will make the same whirring and clacking noises it made when the Apple was turned on for the first time. This time the disk drive will stop whirring on its own. When the whirring stops and the red light goes off, the title APPLE II will disappear and a message will come on the screen.



When you get this message, you know that DOS is booted. The right-hand square bracket and the blinking cursor that appear at the bottom left corner of the screen indicate that the Apple is in the Applesoft language (or has Applesoft "up", as they say) and is ready for instructions.

Another way to boot DOS is to type

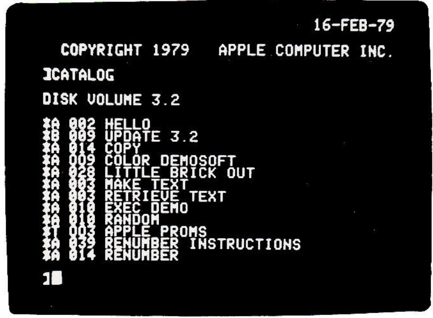
on your Apple. If the controller card is not plugged into slot number six then type

followed by the number of whatever slot the card is plugged into and then RETURN .

The System Master is a very special diskette. It contains programs you'll need in order to get the most out of this manual as well as many other useful programs. To see what programs are on the diskette, use the CATALOG command. Simply type

CATALOG RETURN

and a list of filenames will appear on the screen.



The first program you need is called COLOR DEMOSOFT. Locate the name COLOR DEMOSOFT in the catalog. Now type

RUN COLOR DEMOSOFT

and then press RETURN . The screen should look like the photograph on the next page.

```
         APPLE DEMONSTRATION PROGRAMS

    TO OPERATE A DEMONSTRATION,  TYPE
ITS NUMBER.  THEN TYPE THE KEY MARKED
'RETURN' AT THE RIGHT EDGE OF THE KEY-
BOARD.  TYPE THE 'RETURN' KEY TO STOP
ANY DEMONSTRATION.

1.          STANDARD COLOR NAMES
2.         STANDARD COLOR NUMBERS
3.              KALEIDOSCOPE
4.            SKETCHING SCREEN


WHICH WOULD YOU LIKE?
```

THE MENU

Computerniks call this list of numbered descriptions a "menu." It works like a menu at a roadside cafe. If you want scrambled eggs with hash brown potatoes, toast, jelly and coffee you can just say, "I'll have a number 5." Try selecting one of the color demonstrations by typing its number (followed by a RETURN, of course). When you are viewing one of the demos, just press the RETURN to get back to the "menu."

STOPPING THE COMPUTER

To stop the computer, use

This will cause the prompt character and blinking cursor to appear. The prompt character tells you that it is OK to proceed with typing information to the computer. That is why it is called the prompt character: it "prompts" you to type something.

Once the computer is stopped, it may be started again by typing

RUN

(and, of course, RETURN, but you hardly need to be told that anymore. In fact, you won't be from now on.)

Use

to stop the computer, and

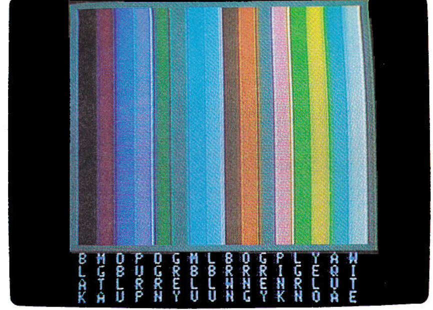
RUN

to start it again. Try this a few times.

SETTING THE TV COLOR

If the "menu" is not on your screen, boot DOS and RUN the program called COLOR DEMOSOFT if you are using a disk drive. Or, if you are using a cassette recorder, follow the Usual Procedure for loading the tape marked "COLOR DEMOSOFT". One of the items on the menu is called STANDARD COLOR NAMES. We will use this DEMO to set the TV color. Type the number of the COLOR NAMES DEMO, 1, and press RETURN . A number of bars of light (perhaps in color) will appear. Under each bar is a four letter abbreviation of a color name. The full names are:

Ø BLACK
1 MAGENTA (a slightly bluish red)
2 DARK BLUE
3 PURPLE (a light purple, lavender)
4 DARK GREEN
5 GREY
6 MEDIUM BLUE
7 LIGHT BLUE
8 BROWN
9 ORANGE
1Ø GREY
11 PINK
12 GREEN
13 YELLOW
14 AQUA
15 WHITE



If you have a black-and-white television or monitor, adjust the brightness and contrast until you are pleased. Of course, if the picture is flipping over, stop it the way you would for any TV show. If you have a color set, a bit more work is necessary.

These colors will be different in Europe and some other parts of the world.

Remember that this color business is quite subjective, and that you can do whatever you want with the color. The following instructions will give the picture that we like, using the standard colors. But it's your eyes you must please. Besides, the optimum settings will vary with different amounts of room light.

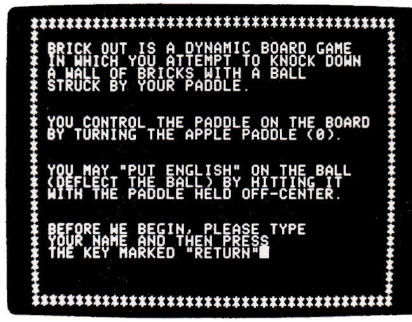
Turn off any Automatic Color switch. On some sets it is marked "AUTO COLOR" or simply "AUTO". Turn the TV set volume control all the way down (but don't turn the set off). Four controls are now important: Picture, Brightness, Color and Hue. Some sets have a knob marked "Contrast" rather than "Picture," but it does the same thing. Turn the Picture control to its dimmest position, and then turn down the Brightness until the background just goes completely dark. Turn the Color control to the middle of its range. Now turn up the Picture control to make things brighter. Do not make it so bright that the colors "spill" off the edges of the bars too much.

Now adjust the Color knob. At one extreme, all color is lost and the picture is black and white. This setting is handy when you are just showing text on the screen. Adjust the Color control until the colors are intense but not "blooming" or spilling into one another. Lastly, adjust the Hue knob until all the colors agree with their names. Purple, Pink and Yellow are especially sensitive indicators. Also, make sure that the three Blues are distinct.

When the TV set's colors are OK, press the RETURN key and the menu will reappear. Now try DEMO 2, which shows the color bars with their code numbers. Also try the other demonstrations. You'll never believe how talented your TV is until you replace the local stations with your Apple.

PLAYING LITTLE BRICK OUT

RUN the program called LITTLE BRICK OUT from your diskette. Or, if you have a cassette recorder, put the tape labeled "LITTLE BRICK OUT" into your recorder, and use the Usual Procedure for getting the tape loaded. The screen will look like the photo on the left when you RUN the program. Then, when you press the space bar, a description of the game will appear on the screen.



When asked, type your name, and then press RETURN . We will type, for example, (as it appears on the screen):

J. APPLESEED

The Apple will respond by drawing the game board complete with paddle. The numbers just below the graphics display are the number of points you can get for hitting bricks from the appropriate columns. The farther you get through the brick "wall", the more points you get per brick.
If you type a name that is more than 12 characters long, the program will truncate the name to a length it is more comfortable with (12 characters).

Sometime you may accidentally press the RESET key instead of the RETURN key (it can happen); the screen will light up. Don't panic. Just type

RUN

Don't forget that we are no longer mentioning RETURN every time it is necessary.

Try deliberately making some errors, such as "accidentally" pressing the RESET key, so that you can get some confidence in your ability to recover from errors.

Meanwhile, back at the LITTLE BRICK OUT program, the Apple instructs you to

PUSH PADDLE BUTTON TO BEGIN GAME

so grab the game control and begin the game.

"Which control?" you ask. Try them both and see. Your Apple will tell you if you have chosen the wrong one.

CHAPTER 2
BEGINNING APPLESOFT

22 A first look at the PRINT statement
25 Applesoft's format for numbers
26 More about RETURN
27 Easy editing features: the arrow keys
29 Putting colors on the screen: GR, TEXT, COLOR= and PLOT
31 PLOT error messages
32 Drawing lines
34 The game controls: PDL
35 Pigeonholes : an introduction to variables
39 Precedence among arithmetic operators, or who's on first?
41 How to avoid precedence

BEGINNING APPLESOFT

If you are in Applesoft, the square bracket prompt character (]), followed by the blinking cursor, will appear at the left edge of the screen each time you press RETURN . Get into Applesoft and, if you have a disk drive, boot DOS.

A FIRST LOOK AT THE PRINT STATEMENT

Now that you have the Applesoft prompt character (]) and the blinking cursor on the screen, (and your diskette is booted if you have a disk drive) you are ready to begin using the Applesoft language. Type

```
PRINT "HELLO"
```

and the computer will print the word

```
HELLO
```

on the next line. If it didn't, ask yourself this question: "Did I forget the RETURN ?" If you misspell the word "PRINT", you will get this error message:

```
?SYNTAX ERROR
```

If you forget either the first quote or both quotes, the computer will print a zero (you can tell it's a zero by the slash):

```
Ø
```

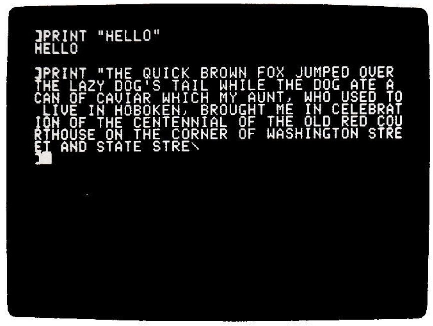
If the final quote is the last character before the RETURN , you don't have to type it: the word "HELLO" will be printed with or without it. It's a good idea to put the end quote in anyway, though. The habit of putting in the final quote will become important later. This manual will assume that you use the final quote.

The statement

```
PRINT "HELLO"
```

is an instruction to the computer telling it to display on the screen all the characters between the quotes, in this case a word of greeting. You can use the PRINT statement to tell the computer to display any message you wish. However, if you type

much beyond 24Ø characters, the computer will start to beep, then give you a backward slash and let you start over again.



Now try the statement

```
PRINT "15Ø"
```

The computer obediently prints the number 15Ø on the next line, as expected. But type

```
PRINT 15Ø
```

and the computer again prints the number, without any fuss or error message about the missing quotation marks. In fact, the Apple will let you PRINT any number at all without enclosing it in quotes.

Without further study, the Apple can be used as a simple-minded desk calculator.

Try this on your Apple:

```
PRINT 3 + 4
```

The answer, 7, appears on the next line. The Apple can do six different elementary arithmetic operations:

1. ADDITION. Indicated by the usual plus sign (+).

2. SUBTRACTION. Uses the conventional minus sign (-).

3. MULTIPLICATION. Many people use an "X" to represent multiplication. This could be confused with the letter "X". Some people use a dot (.), but this could be confused with a period or a decimal point. So the Apple uses an asterisk (*). To find 7 times 8 (in case you don't remember the answer), just type

```
PRINT 7 * 8
```

and have your memory jogged.

4. DIVISION. As is customary, use a slash (/). To divide 63 by 7, type

```
PRINT 63 / 7
```

and the correct answer will appear.

Try dividing 3 by 2. The answer is one and one half. The Apple gives the answer to you in the decimal form: 1.5.

One thing we should point out here is that you can do more than one arithmetic operation in the same instruction. For example, it is legal to say

```
PRINT 3 + 5 + 9 + 4
```

The exact rules governing such usage will be given later, but you can experiment with it now if you wish.

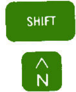
6. EXPONENTATION. It is often handy to multiply a number by itself a given number of times. Instead of bothering to write

```
PRINT 4 * 4 * 4 * 4 * 4
```

you can substitute the shorthand

```
PRINT 4 ^ 5
```

The upward pointing arrow is typed:



There is nothing special about exponentiation. It is just an abbreviation for repeated multiplication! In non computer-notation, this would be written with a superscript five, like this: 4^5

```
150
]PRINT 150
150
]PRINT 3 + 4
7
]PRINT 7 * 8
56
]PRINT 63 / 7
9
]PRINT 3 + 5 + 9 + 4
21
]PRINT 4 * 4 * 4 * 4 * 4
1024
]PRINT 4 ^ 5
1024
]
```

APPLESOFT'S FORMAT FOR NUMBERS

Type

```
PRINT 45.34Ø
```

Your computer responded with

```
45.34
```

and didn't PRINT the trailing zero. The Apple does not PRINT leading or trailing zeros, that is, zeros that are at the beginning of a number and to the left of the decimal, or zeros that are at the end of a number and to the right of the decimal.

Very, very small numbers (between about .ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ3 and -.ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ3) will be converted to zero by the Apple. (We hope that was the right number of zeros.) An easier way to write these numbers is 3 * 1Ø^ - 39 and -3 * 1Ø^ - 39. Don't take our word for it. Try it yourself.

Now type

```
PRINT 985788.6898
```

Surprise! The last two digits are lost, and the number left behind is the closest approximation the computer can think of. This process is called "rounding". Try typing

```
PRINT 788.6898
```

Your computer did not round the number, but PRINTed it just the way you typed it. Madness you say? Ah, but there is a method to this seeming madness. Numbers are rounded only if they have more than nine digits. Any number that has fewer than ten digits will not be rounded. The computer does the best it can, but it only has nine digits to work with.

If you type a PRINT statement with a long number like

```
123456789Ø
```

the Apple responds with

```
1.23456789E+Ø9
```

The numbers 1234567890 and 1.23456789E+09 have the same value. Really. The number PRINTed by your computer is in "scientific notation". If you need numbers like this you probably know how to read them. The Applesoft BASIC Programming Reference Manual has more information if you are curious about this strange notation.

Try some more numbers. How many digits can a number without a decimal point have before the Apple changes it to scientific notation? If scientific notation seems complicated, don't worry. You probably won't be wanting to use numbers that require it for some time yet. Remember that any number will be PRINTed just the way you type it if the number is surrounded by quotes. However, the Apple can't use numbers in quotes for arithmetic operations. For more information on scientific notation and other types of number formatting used by the Apple, refer to the Applesoft II BASIC Programming Reference Manual.

MORE ABOUT RETURN

So far, you have been pressing RETURN after every line, like a zombie. We thought we might tell you why this key gets so overworked. The reason is simple: without the RETURN , the computer does not know when you have completed the instruction. For example, you might start typing

```
PRINT 4 + 5
```

If the computer immediately jumped in and printed a 9, you might be upset because you had planned to type

```
PRINT 4 + 5 + 346
```

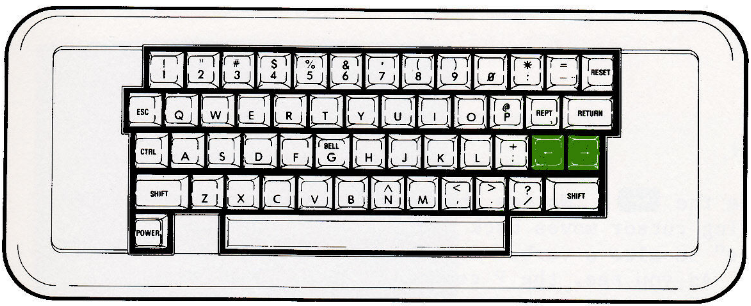
which would have given a different answer entirely. Since the computer can't tell when you have finished typing an instruction, you must tell the computer. You do this by pressing the RETURN key. Since you always have to do this after typing an instruction, we have (as you know) stopped mentioning RETURN after every instruction. Pressing RETURN after each instruction should be a habit by now, if you have been doing all the examples.

We really hope you have been trying all the examples. Learning to program is very much like learning to ride a bicycle, play the piano, or throw a baseball. You can read all the books in the world on the subject of bicycle riding, and be a great "paper expert." But all this book-learning is of little help when you actually get on a bicycle for the first time. Once you

have learned to ride, through experience (which can be a bit painful), you can go almost anywhere. The same is true of programming. You can read this manual and think you understand it. But you won't be able to program. Only if you do each example, as it is given, will you learn to program. That's the truth.

EASY EDITING FEATURES, OR: WHAT TO DO BEFORE YOU HIT RETURN

No one is a perfect typist. We make mysteaks (Oops! See what I mean?). The Apple has several features that aid in correcting errors, thereby saving you the effort of retyping a whole line for each goof. This is where the left-pointing and right-pointing arrows on the keyboard come in.



The ← key is rather like the backspace key on a typewriter so we will call it the "backspace key". A few experiments will make this clear. Type (exactly as shown) the statement:

```
PRINT COMPUTER"
```

and, as usual, press the RETURN key. The computer will reply

```
ø
```

because of the missing quote. Now if we had typed

```
PRINT "COMPUTER"
```

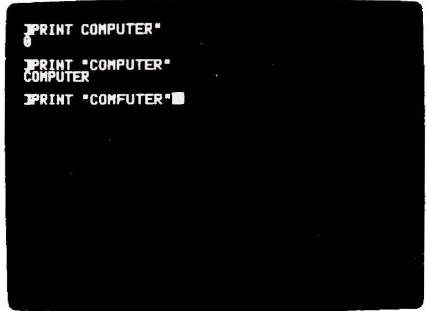
the computer would have responded with

```
COMPUTER
```

Don't believe this manual. Try it. Now, without pressing RETURN , type the "mistaken" instruction:

```
PRINT "COMFUTER"
```

Since you haven't pressed RETURN , nothing has happened yet. As shown in the photograph, the cursor is sitting to the right of the last quote. (Sorry, we can't make the photo blink)



To change

COMFUTER

into

COMPUTER

we can use the ← . Notice that each time you press this key, the blinking cursor moves back (to the left) one space. "Backspace" is also a verb. So backspace the cursor to the F. Type a P. As you see, the P replaces the F. Now press RETURN . You got

COMP

from the computer? That is because you backspaced over "UTER". Any character in the line you are currently typing that is backspaced over is not sent to the computer when you press RETURN . One solution would be to correct the F by backspacing to it, and then to type

Try it.

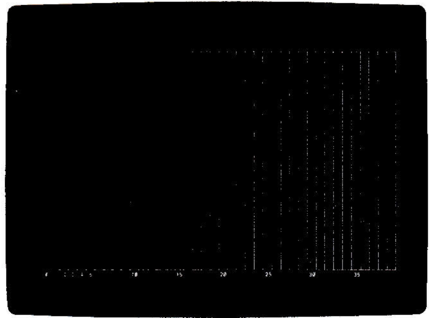
It works! There is, however, an easier way. When you press the → key, the cursor moves to the right. As the cursor moves to the right across a character, it has the same effect as if that character had been retyped. We call the → key the "retype" key. Again type

PRINT "COMFUTER"

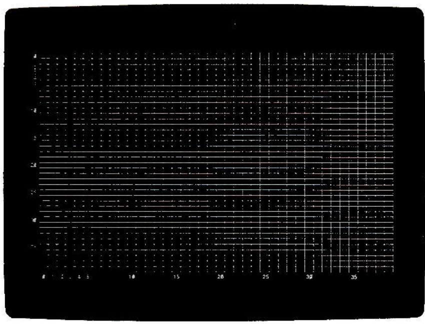
then backspace to the F and change it to P. To complete the correction simply press the retype key five times, and then press RETURN . Does it all work? The use of the backspace and retype keys will save you a lot of time. Make a point of using them a number of times on your own "mistakes," so that these keys become familiar.

PUTTING COLOR ON THE SCREEN

To put color graphics on the screen, we need a way to describe which of the 16 available colors we want, and where we want it. To specify where a color goes, we divide the screen into 4Ø vertical columns, numbered Ø through 39. The Ø column is at the leftmost edge of the screen, and the numbers increase to the right. You may wonder why the numbers don't go from 1 through 40 instead of Ø through 39. As you get more programming experience you will find that the choice we have made is somewhat handier, even though it may not seem that way at first.



The screen is also divided into 4Ø horizontal rows, again numbered Ø to 39. The horizontal rows start with row Ø at the top of the screen and increase to row 39 at the bottom. These rows cut across the columns, partitioning each column into 4Ø "bricks" numbered Ø (the top brick) through 39 (the bottom one). Those who like formal terminology will recognize that this is merely a system of rectangular Cartesian co-ordinates. Those who don't like fancy talk can just think in terms of columns of bricks.



To use the screen colorfully, type the following instruction:

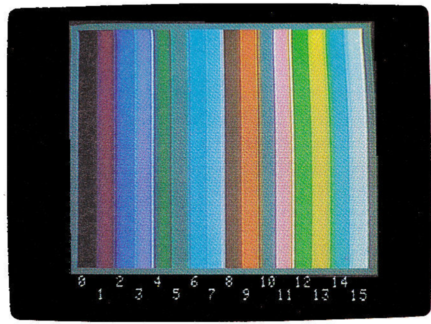
```
GR
```

You remembered the RETURN , no doubt. When you use this command the screen wipes itself clean, leaving only four lines for text at the bottom. The "GR" stands for GRaphics. To get back to things as they were (before you typed GR) you use the command

```
TEXT
```

When you type this command the screen will suddenly change to a lot of "at" signs (@). This is normal. Try typing the TEXT instruction, and then getting back to graphics by typing the GR instruction.

Before you can place a dot of color on the screen, you must tell the computer which color you want. There are sixteen colors available. You have seen them before: they are numbered from Ø to 15, as shown in COLOR DEMOSOFT 2.



Suppose you want to put a green dot somewhere. You must first type the GR command, and then type

```
COLOR = 12
```

This means that any dot (or spot or brick) of color that you place will be green. In fact, until otherwise instructed, everything the computer puts on the screen will be green. Except, of course, for the small area reserved at the bottom of the screen for your instructions. To put a spot of color in the upper left-hand corner of the screen (leftmost or zeroth column, top or zeroth brick), you type

```
PLOT Ø,Ø
```

To put a spot of the same color in the upper right-hand corner, you must specify column 39, brick Ø. So type

```
PLOT 39,Ø
```

Notice that you always give the column first. Now put an orange brick at the lower left-hand corner. First change the color. Remember--you should really be doing these exercises, not just thinking about them. So, put out your fingers and type

```
COLOR = 9
```

Nothing happens on the upper, graphic portion, of the screen (even if you did remember to press RETURN). But the computer remembers that when you next PLOT something, it will be in orange, not in green. Now that you have chosen the color, you can put a dot in the lower left-hand corner. That's column Ø, and brick 39:

```
PLOT Ø,39
```

Did it work? Did you forget to press RETURN ? Is orange your favorite color?

Now put a magenta dot in the lower right-hand corner. Figure it out for yourself.

PLOT ERROR MESSAGES

There are two error messages that can easily turn up when you are using the PLOT statement. You already know that if you typed

```
PLAT
```

or

```
PLOP
```

instead of

```
PLOT
```

you would get the message

```
?SYNTAX ERROR
```

A new error message occurs when you plot a number higher or lower than those permitted for coordinates in a PLOT command. Type

```
PLOT 13,85
```

and you get the message

```
?ILLEGAL QUANTITY ERROR
```

This message means that you have tried to plot a point out of range and off the screen. The highest numbers you can use in a PLOT statement are 39 for the first coordinate, and 47 for the second. Use of numbers over 39 for the second coordinate, as in a statement such as

```
PLOT 2Ø,45
```

will just give you peculiar characters in the text area at the bottom of the screen.

Trying to use negative values in a PLOT command is another way to get the

```
?ILLEGAL QUANTITY ERROR
```

message.

DRAWING LINES

Suppose you want to draw a light blue horizontal line from column 5 to column 9 at the level of brick 14. You could type

```
COLOR = 7
PLOT 5,14
PLOT 6,14
PLOT 7,14
PLOT 8,14
PLOT 9,14
```

Notice that the joints between adjacent bricks do not show, and they form a continuous line. However, there is an easier way to do horizontal lines. There had better be. Suppose you want to draw a dark green horizontal line across the middle of the screen. Using the long way, it would take forty typed statements:

```
COLOR = 4
PLOT Ø,2Ø
PLOT 1,2Ø
PLOT 2,2Ø
```

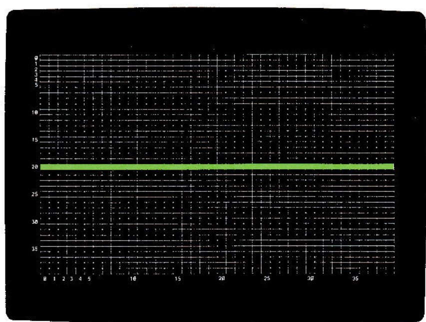
and so on, until

```
PLOT 39,2Ø
```

The easier way is this. Just type

```
COLOR = 4
HLIN Ø,39 AT 2Ø
```

Press the RETURN key, and there you have it: an instant Horizontal LINe from column Ø to column 39 at the level of brick 2Ø.



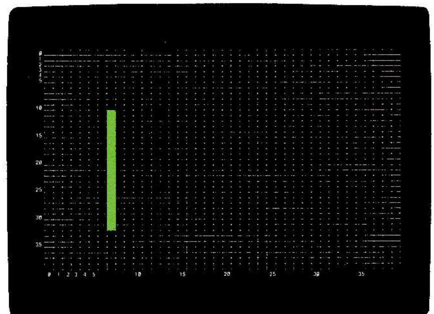
Now try to place a purple line from column 19 to column 28 at the level of brick 18. Try a few others. Doing about 6 different horizontal lines should give you the hang of it.

Notice that when you put a colored dot or line at the same location as an existing dot or line, the new color takes over, and the old color disappears. To clear the screen of all graphics at once, use the GR command.

There is a provision for automatic vertical lines similar to that for horizontal lines. To draw an orange Vertical LINe in column 7 from brick 12 to brick 33, we write:

```
COLOR = 9
VLIN 12,33 AT 7
```

Try this statement.



Practice making several more vertical lines by changing the numbers for the rows and columns. You can test your proficiency with both horizontal and vertical lines by drawing a magenta border around the screen in five statements. Then put a green cross on the screen. Try drawing some lines with COLOR = Ø. Play with PLOT, HLIN and VLIN for a while. This manual's usefulness to you will self-destruct in five seconds if you don't experiment with these commands. Pfffsssss.

THE GAME CONTROLS

Grab the game control that you used in playing LITTLE BRICK OUT. With the other hand type

```
PRINT PDL(Ø)
```

and a number should appear. Move the control a bit. Now type

```
PRINT PDL(Ø)
```

again. Experiment with moving the control and typing

```
PRINT PDL(Ø)
```

If the number never changes, you've got the wrong game control. What are the highest and lowest numbers you can get? What is the smallest change you can make?

You can discover the position of the other control by PRINTing PDL(1). The abbreviation "PDL" comes from the word "PADDLE" since these controls are often used to control "paddles" in games. There are many other uses for these controls.

PDL is a <u>function</u>. A function, in Applesoft, is something that takes one or more numbers and then performs some operation on them to yield a single value. The numbers that the function uses are called its <u>arguments</u> and are always put in parentheses after the function name. PDL is a function that has one argument. The number the function finds is said to be <u>returned</u> to the program.

PIGEONHOLES AND MORE CALCULATOR ABILITIES

On many simple calculators you can save a number for later reference or use. To do this, you put the number into a special place in the calculator--a place we shall call, for now, a pigeonhole. Usually this is done by pressing a key marked "M" for "Memory." On the Apple you can do the same thing. For instance, to save the value 77, you type

```
M = 77
```

The value, 77, is not printed, just stored in the pigeonhole called M. If you now type

```
PRINT M
```

the computer will print the value of M. Try typing the two statements.

Now type

```
M = 324
```

and PRINT the value of M. It is 324, right? What happened to the 77? It is gone forever. The pigeonhole can hold only one value at a time. When you put a new value in M, the old value is erased.

Type

```
PRINT "M"
```

What happens? There is a big difference between

```
M
```

and

```
"M"
```

It is just like the difference between these two statements in English:

MICE HAVE FOUR FEET.

"MICE" HAS FOUR LETTERS.

In one case we are referring to little furry things with long tails. In the other case we are referring to the word itself. This is how quotes are used in computerese. When we say

```
PRINT "M"
```

we mean to print the letter itself. When we say

```
PRINT M
```

we mean to print what the letter stands for. You would never confuse the name of someone you love with the actual person that name stands for.

You can store the result of a computation in a pigeonhole. For example:

```
M = 4 + 5
```

You can see that the answer has been stored by PRINTing the value of M.

You can also use the value of M in further computation. For example, try this on your Apple:

```
PRINT M + 2
```

Is the answer what you expected? Try some other calculations using M.

A simple calculator has one pigeonhole. Computers have hundreds of pigeonholes (Applesoft has 936). The formal term for pigeonholes is variables. But this term is somewhat misleading since pigeonholes don't behave like "variables" in mathematics. They are much simpler. Each one is merely a place where one value is stored. But we will defer to common usage. Just forget the math you've learned. In the Apple all variables have the value of zero until you put something into them.

A pigeonhole, or variable, can have almost any name that you like, so long as it starts with a letter. For example:

```
SUM = 56 + 34 + 1523 + 8
GAMEPOINTS = 45
PLAYER2 = 9
```

Some names are not allowed because they include a word that has a special meaning to the Apple. These are known as reserved words. One of these words is "COLOR". Thus a variable's name must not have the word "COLOR" in it. Try typing

```
THISCOLOR = 6
```

or

```
COLORFUL = 9
```

All you get for your pains is an error message. Whenever a variable name gives you the ?SYNTAX ERROR message, it means that you have unwittingly included a reserved word in the name. Don't worry. Just choose another name.

A list of "reserved" words that cannot be used as variables or as part of variable names can be found in Appendix B in the back of this manual.

When you are choosing names, make them reflect the use to which they are being put. This will make them easier to remember.

Try typing

```
BIRD = 11
```

and then

```
PRINT BIRD
```

Did you get what you expected? Now type

```
PRINT BITE
```

What happens? Try

```
PRINT BILL
```

and

```
PRINT BILLOW
```

If you study the names you will notice that they all begin with "BI". Applesoft uses only the first two characters of each variable name to distinguish it from other variable names. So the name

```
BIRD
```

refers to the same variable as

```
BITE
```

and

```
BILLOW
```

and so on.

Here is a useful trick. Let's say that you had some value in the variable PRICE, and you wanted to increase this value by 5. One way you could do this would be to PRINT the value of PRICE, then add 5 to that value, and finally store the resulting value back in PRICE. For instance:

```
PRICE = 28
PRINT PRICE
PRINT 28 + 5
PRICE = 33
```

But see how much easier it is to type

```
PRICE = 28
PRICE = PRICE + 5
```

Try the statements on the next page in order:

```
PRICE = 2
PRINT PRICE
PRICE = PRICE + 3
PRINT PRICE
PRICE = PRICE * 6
PRINT PRICE
PRICE = PRICE / 10
PRINT PRICE
```

At the end of this sequence of statements, you will probably have the value 3. Is this correct? Is this what you expected? Try this sequence:

```
APPLES = 55
BANANAS = 11
QUOTIENT = APPLES / BANANAS
PRINT QUOTIENT
```

First think what answer you expect, then see if you are right. If you are not, find out why. Lastly, try these statements:

```
HELLO = 128
PRINT "HELLO"
HELLO = HELLO / 2
PRINT "HELLO"
HELLO = HELLO / 2
PRINT HELLO
```

What did you expect? What did you get?

PRECEDENCE, OR WHO'S ON FIRST?

At certain old-fashioned banquets, the people were served their food according to a strict plan: first the guest of honor, then the female guests (in order of the rank of their husbands), then the male guests (in order of rank), and finally the host. No matter where they were seated, the waiter went among them choosing the appropriate persons to be served next. We could say there was a certain precedence among the diners. In a simple calculation like

```
PRINT 4 + 8 / 2
```

you can't tell whether the answer should be 6 or 8, until you know in which order (or precedence) to carry out the arithmetic. If you add the 4 to the 8, you get 12. If you then divide 12 by 2, you get 6. That's one possible answer. However, if you add 4 to eight-divided-by-two, you have 4 plus 4, or 8. This is another possible answer. Eight is the answer your Apple will give. Here's how the Apple chooses the order in which to do arithmetic:

1. When the minus sign is used to indicate a negative number, for example

```
-3 + 2
```

the Apple will first apply the minus sign to its appropriate number or variable. Thus -3 + 2 evaluates to -1. If the Apple did the addition first, -3 + 2 would evaluate to -5. But it doesn't. Another example is

```
BRIAN = 6
PRINT -BRIAN + 10
```

The answer is 4. (Notice, though, that in the expression 5-3 the minus sign is indicating subtraction, not a negative number.)

2. After applying all minus signs, the Apple then does exponentiations. The expression

```
4 + 3 ^ 2
```

is evaluated by squaring three (three times three is nine), and then adding four, for a grand total of 13. When there are a number of exponentiations, they are done from left to right, so that

```
2 ^ 3 ^ 2
```

is evaluated by multiplying 2 by itself three times (2*2*2) which is eight, and then multiplying that by itself (8). The answer is 64.

3. After all exponentiations have been calculated, all multiplications and divisions are done, from left to right. Arithmetic operators of equal precedence are always evaluated from left to right. Multiplication (*) and division (/) have equal precedence.

4. Lastly, all additions and subtractions are done, from left to right. Addition (+) and subtraction (-) have equal precedence.

Let's summarize the Apple's order of precedence for carrying out mathematical operations:

First: - (minus signs used to indicate negative numbers)
Second: ^ (exponentiations, from left to right)
Third: * / (multiplications and divisions, from left to right)
Fourth: + - (additions and subtractions, from left to right)

Below, you will find some arithmetic expressions to evaluate. With each one, first do it in your head (or with the help of a hand-held calculator, or pencil and paper), and then try it on the Apple. If your own answer is different from the Apple's answer, try to find out why. We will give only the expressions here. You will have to put a PRINT in front of each one to get its value from the computer.

Unless you have a lot of experience with the way computers evaluate expressions, you should actually do these examples. Don't do them all at once and then check with the computer. Do an example by hand and then do it on the computer. Then go on to the next one, and so on.

```
3 + 2
4 + 6 - 2 + 1
8 * 4
4 ^ 2 + 1
6 / 4 + 1
5 - 4 / 2
4 / 2 - 2
6 * -2 + 6 / 3 + 8
4 + -2
2 ^ 2 ^ 3 + 1
2 * 2 * 3 + 1
2 * 2 + 1 * 3
2 * 2 * 1 + 3
8 / 2 / 2 / 1
8 * 2 / 2 + 3 * 2 ^ 2 * 1
20 / 2 * 5
```

No answers are given in this book. Your Apple will give you the correct answers.

HOW TO AVOID PRECEDENCE

Suppose you want to divide 12 by four-plus-two. If you write

```
12 / 4 + 2
```

you will get 12-divided-by-four, with two added on. But this is not what you wanted. To accomplish what you wanted in the first place, you can write

```
12 / (4 + 2)
```

The parentheses modify the precedence. The rule the computer follows is simple: do what is in parentheses first. If there are parentheses within parentheses, do the innermost parentheses first. Here is an example:

```
12 / (3 + (1 + 2) ^ 2)
```

In this case, doing the innermost parentheses, you first add 1 + 2. Now the expression is, effectively,

```
12 / (3 + 3 ^ 2)
```

But you know that 3 + 3 ^ 2 is 3 + 9 or 12 so the expression has now been simplified to 12 / 12, which is one.

In a case like (9 + 4) * (1 + 2), where there is more than one set of parentheses, but they are not "nested" one inside the other, you just work from left to right. This expression becomes 13 * 3, or 39.

Here are some more expressions to evaluate. Again, if you are not familiar with computers, the few minutes you spend actually working these out and trying them on the Apple will be very valuable. You will be well repaid for your efforts by being able to use the computer more effectively. Incidentally, most of these rules for precedence and parentheses hold good for most computer systems anywhere in the world, not just the Apple.

```
44 / (2 + 2)
(44 / 2) + 2
3 + (-2 * 2)
(3 + -2) * 2
1ØØ / (2ØØ / (1 * (9 - 5)))
32 / (1 + (7 / 3) + (5 / 4))
```

CHAPTER 3
ELEMENTARY PROGRAMMING

44 Deferred execution: NEW, LIST, RUN and HOME
48 Elementary editing: DEL
5Ø Elementary aerobatics: GOTO loops
5Ø Some more things that make life easier: more editing tips
52 The moving cursor: editing with the ESC key
53 A word about learning Applesoft BASIC
54 An accident about to happen
55 The truth: arithmetic and logical assertions
59 Order or precedence for operations
59 The IF statement
6Ø Saving programs on diskette: SAVE, CATALOG, RUN and LOAD
61 Saving programs with a cassette recorder: SAVE
62 More graphics programs: REM
64 FOR/NEXT loops
67 A wrong program
67 A last example of nested loops
68 Getting flashy: INVERSE, FLASH and NORMAL
68 PRINTs charming: comma, semi-colon, TAB, HTAB and VTAB

DEFERRED EXECUTION

No, this section is not on last minute reprieves for condemned criminals. Up to now, when you typed

```
PRINT 3 + 4
```

and pressed the RETURN key, the computer would do what you told it to do, immediately. When a computer performs according to the statement you have given it, it is said to execute that statement. Thus, you have been using the computer to do immediate execution of each statement you have typed on the Apple's keyboard.

You are about to learn how to store statements for execution at a later time (deferred execution). To make sure that the computer's memory is cleared of any previous programs, type

```
NEW
```

Like almost everything else you have seen, NEW has to be followed by a RETURN . To tell the computer to store a statement, just type a number before typing the statement. For example, if you type

```
100  PRINT 3 + 4
```

nothing seems to happen, even if you press RETURN . The Apple has stored the statement. To see that it has stored the statement, you type the instruction

```
LIST
```

Try it. Unless you mistyped something (and probably got a

```
?SYNTAX ERROR
```

for your effort),

```
100  PRINT 3 + 4
```

appears on the screen. Now type the statement

```
RUN
```

and the answer

```
7
```

appears on the screen.

Typing RUN caused your stored statement to be executed, but the computer has not forgotten the statement. You can RUN the same statement as many times as you like. Try it.

What's more, the computer does not forget the stored statement when you clear the screen. Here's a new way to clear the screen:

```
HOME
```

The HOME command has the same effect as

that you learned earlier, but it can be used in deferred execution as well as immediate execution. To try this out type

```
100  HOME
```

Now when you type

```
RUN
```

the computer faithfully executes the stored statement and clears the screen. Type

```
NEW
```

and then

```
LIST
```

and see what happens. Typing NEW has caused the stored statement to be lost permanently. Type

```
RUN
```

and nothing appears on your screen. That is because your old statement has been erased by the NEW command.

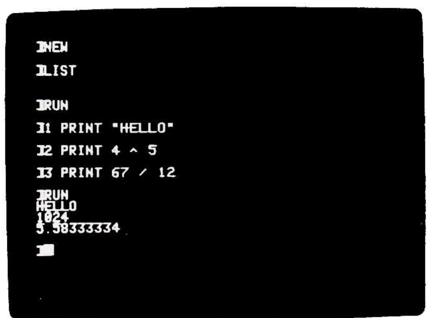
It is possible to store many statements by giving each of them a different number. Try typing this:

```
1  PRINT "HELLO"
2  PRINT 4 ^ 5
3  PRINT 67 / 12
```

Nothing much has happened so far. But now type

```
RUN
```

and watch the answers appear.



The numbers that we put in front of statements, in order to tell the computer to store them, are called line numbers. The computer stores and executes statements in order of increasing line number. To see this in action, erase the statements you stored by typing

```
NEW
```

and then type these statements:

```
1  PRINT "P"
Ø  PRINT "A"
3  PRINT "E"
2  PRINT "L"
```

Notice that zero is an allowed line number. The highest line number that you can use is 63999. Now RUN these instructions. The results should look like this:

```
A
P
L
E
```

To see what has happened inside the Apple, type

```
LIST
```

Notice that you do not have to LIST a set of instructions before you RUN them. It is, however, a good idea to do so.

A set of instructions that is executed when you type RUN is called a program.

The program was meant to print

```
A
P
P
L
E
```

but, it seems, a PRINT statement was left out. How can you add it in? Only by retyping the statements with line numbers 2 and 3 as statements 3 and 4, and adding a new line number 2. To make the corrections type this:

```
2  PRINT "P"
3  PRINT "L"
4  PRINT "E"
```

To see what has happened, LIST the program.

Notice that in whatever order statements are entered, the Apple stores them with their line numbers in numerically ascending order. Now RUN this program.

It was a bother to have to retype those statements in order to merely add one in the middle. It is, therefore, good programming practice to leave some line number room between lines, and before the first line. Type

```
NEW
```

to eliminate that program and put in this one:

```
1ØØ  PRINT "C"
11Ø  PRINT "T"
```

When you RUN this program it doesn't quite print the word "CAT" vertically. But now you can go back and type

```
1Ø5  PRINT "A"
```

LIST and RUN this program. From now on this book will start all programs at a reasonably high line number and leave plenty of room between successive line numbers so there will be adequate room for inserting statements.

ELEMENTARY EDITING

Earlier, you discovered that the instruction

```
PRINT PDL(Ø)
```

would print a number corresponding to the present position of one of the game controls. It took quite a number of PRINTs to discover very much about the control. Now that you can write programs, life is much easier. Clear the computer with a

```
NEW
```

and type

```
1ØØ  PRINT PDL(Ø)
```

Now, each time you type RUN, this short program is executed and you can see the position of the game control.

For doing something more than once, the stored program is already saving you some work. Before, you had to retype a whole statement or group of statements. Now, you merely retype

```
RUN
```

Deferred execution confers another advantage. You can modify part of a program and leave the rest the same, without having to retype the whole thing. For example:

```
NEW
2ØØ P =  PDL (Ø)
21Ø  PRINT P
22Ø  PRINT "MOVE THE GAME CONTROL"
23Ø  PRINT "TO A NEW POSITION"
```

RUN this program a few times, changing the game control's setting between RUNs. Check to see if the program responds to both game controls. It should work for only one of them. You might take this opportunity to mark this control with the number zero.

This same program can be used, with a slight change, to look at the other game control. List the program the way it is now, then type

```
2ØØ P = PDL(1)
```

When you type a statement with the same line number as one that already exists in a program, the new line replaces the old one. LIST the program to see how it has changed. RUN it a few times to see what happens. Move the other game control between RUNs. Does this program respond to both controls? Mark a number one on the control to which this program responds.

Modifying a program in this way is one example of editing a program. Another way is to use what you have just learned to delete lines you no longer want in your program. If you wanted to erase line 23Ø in the preceding program you would type

```
23Ø
```

and then press

You could also have used the DELete instruction. To DELete line 23Ø from your program you would type

```
DEL 23Ø,23Ø
```

The advantages of the DELete are not apparent until we reveal to you that whole sections of programs can be erased with instructions like

```
DEL 2ØØ,23Ø
```

which DELetes every statement whose line number is 2ØØ or greater, but less than or equal to 23Ø. Try these commands, and LIST the program to see what they do to it. The ability to DELete blocks of line-numbered statements will be handy when you are writing large programs.

As you have seen, there are several commands that help you deal with whole programs. They are

```
NEW
```

which erases programs,

```
LIST
```

which displays programs, and

```
RUN
```

which executes programs, beginning with the statement having the lowest line number. It is also possible to start execution elsewhere, and to LIST only part of a program. These capabilities will be covered later.

ELEMENTARY AEROBATICS

At this point you are beginning to fly, so this section will discuss loops.

The best way to see how the PDL function works--and to understand program "loops"--is to use a statement we haven't discussed, until now. It's very simple. Type the following lines (after typing NEW, to erase any old programs that might be around):

```
11Ø  PRINT PDL(Ø)
12Ø  GOTO 11Ø
```

Line 11Ø of this program PRINTs the number representing the current value of the game control. Line 12Ø does just what it seems to say: it causes program execution to go to line 11Ø. What happens then? The program PRINTs the current value of the game control. Then it executes line 12Ø, which says to do line 11Ø over again, and so on. Forever. This is a loop. A loop is a program structure that exists when the program includes a command to return to a statement executed previously. RUN the program. Play with the game control. In the next section, we will tell you how to stop this program. Meanwhile, admire the fact that--if you typed RUN when instructed to do so, three sentences back--your Apple has executed the statement PRINT PDL(Ø) a few hundred times already. Now the power of a stored program begins to increase significantly over what you can do by hand. Your abilities with the computer will increase dramatically in the next few sections, now that a good groundwork has been established.

SOME MORE THINGS THAT MAKE LIFE EASIER

But first, you are probably wondering how to stop the paddle program. You have already noticed how the numbers ripple up the screen as you move the paddle. This is because the numbers are printed at the bottom of the screen and as each new number is printed, all the rest of them are moved up one line. This is called "scrolling" and you've been seeing it all along, but at a much slower rate. To stop running the program, simply use

The CTRL C command lets you know where the program execution was stopped by printing

```
BREAK IN 11Ø
```

or whatever line number the program was stopped at. (Try it). By the way, this is an exception to the rule about pressing RETURN after every command. Pressing RETURN is usually not necessary when a program is stopped with CTRL C. You can also use the RESET key to stop programs if you like, but, with RESET you will not get the message that tells you the line number at which the program was stopped.

When you stop a program with CTRL C or RESET, you can resume its execution by typing the instruction

```
CONT
```

which stands for "CONTinue." Try it, and then try this program:

```
NEW
10Ø X = PDL(Ø)
11Ø  PRINT "GAME CONTROL ZERO IS"
12Ø  PRINT X
13Ø Y = PDL(1)
14Ø  PRINT "AND CONTROL ONE IS"
15Ø  PRINT Y
```

Earlier we said that when you type RUN, the program starts executing at the lowest numbered line. True. However, if you want to start RUNning at some other line, such as line 13Ø, you simply type

```
RUN 13Ø
```

You can specify line numbers in the LIST statement, as well. If you type

```
LIST 13Ø
```

the Apple will LIST line 13Ø (if there is one, of course). If you type

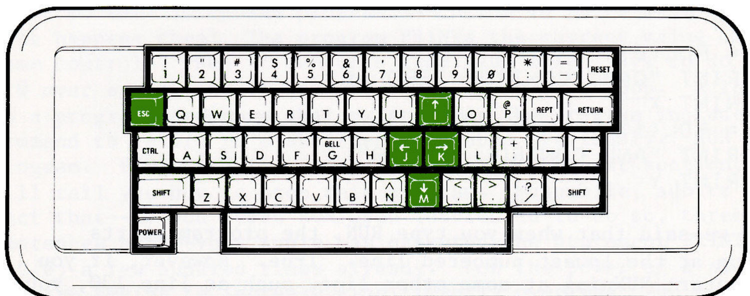
```
LIST 11Ø,13Ø
```

the Apple will LIST all the lines of your program starting at line 11Ø and continuing through line 13Ø. This feature is not available with the RUN instruction.

THE MOVING CURSOR
HAVING WRIT
CAN ERASE OR COPY
ANY OF IT

When the backspace and retype keys are pressed, they move the cursor. But they also either erase or retype characters, as you learned in Chapter 2. It is possible to move the cursor without affecting anything at all (except the cursor position). You do this by using <u>pure</u> <u>cursor</u> <u>moves</u>.

Five keys are used in executing pure cursor moves. They are: ESC , I , J , K , and M . Here's how to use them.



First you put the Apple in edit mode by pressing and releasing the ESC key, then use I to move the cursor up, J to move it left, K to move it right, and M to move it down. To move the cursor repeatedly, hold down one of the cursor direction keys (I , J , K , or M) and then hold down the REPT key at the same time. The cursor will zip along while both keys are held down. If the cursor reaches the top of the screen, it will stop. If the cursor reaches the bottom of the screen, it will stop, and the screen display will move upwards, one line at a time. If it reaches the right edge of the screen, the cursor will disappear and "wrap around" to the beginning of the next line. Practice moving the cursor around the screen with these five keys.

When you get tired of pure cursor moves just press the space bar, and you will find yourself in normal typing mode.

The pure cursor moves caused by the J and K keys, when seen on the screen, appear the same as the moves caused by the backspace and retype keys, but the effect is different as you will see when you LIST the results. The pure cursor moves don't cause any changes in the text they go over, whereas the backspace and retype keys erase and retype the characters they go over.

These pure cursor moves do have an application, so they are not so pure after all. For example, type:

```
130  PRUNT "THE QUALITY OF MERCY"
140  PRINT "IS NOT STRAINED"
```

pressing RETURN after each line. Your Apple seems to accept the statements, but when you try to RUN the program, you get the message

```
?SYNTAX ERROR IN 130
```

for your pains.

To make the needed correction you can use this trick to effectively retype the entire statement. First LIST the program and then press ESC . Now type I enough times to move the cursor up to the line with the incorrect statement. Press the J key to move the cursor to the beginning of the line, and then use the retype key to retype all the characters preceeding the "U" in "PRUNT". Type an "I" over the "U", and continue with the retype key to the end of the line. Then press RETURN and LIST the program to see that the line is properly corrected. The computer does have mercy on poor typists.

When you must retype a portion of a line that appears somewhere on the screen, the pure cursor moves and the backspace and retype keys can be used to speed the retyping. A few minutes of playing with this feature now will save you much work later.

The backspace key only works on the line you are currently typing. If you type a program line and then execute a pure cursor move before you press RETURN , the program line you just typed, not the characters the backspace key is going over, will be affected by the backspace key moves. The retype key, however, does retype the characters it actually goes over.

A WORD ABOUT LEARNING APPLESOFT BASIC

Many times there are questions you can ask about the Applesoft BASIC language that are not answered directly in this book. For instance, in the statement

```
PRINT "HELLO"
```

do you have to put a space after the word "PRINT"? Rather than give you the answer, we recommend that you simply turn to your Apple and try it both ways. Usually a simple experiment will answer your question and, since you have taken the time to try it yourself, you will remember it far better than if you had merely read it.

AN ACCIDENT ABOUT TO HAPPEN

Earlier in this chapter you learned to delete a line by typing its line number and pressing the RETURN . This is a favorite way of introducing errors into your program. Suppose you wanted to eliminate line 11ØØ from your program, but you slipped and typed

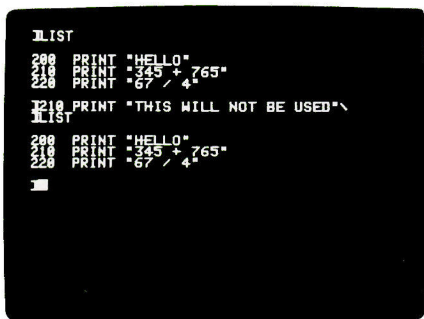
```
11Ø
```

RETURN . Congratulations, you have just wiped out line 11Ø. This happens. Or you are about to fix up line 45Ø so you type

```
45Ø
```

and think about it and decide not to change the line after all. Don't press RETURN . Either backspace over the line number, or use the special "forget this line" command,

Using CTRL X places a backslash at the end of the line you are typing, and it will be as if you never typed it at all.



THE TRUTH

The Apple can distinguish between what is true and what is false. Since this is more than most of us can do, a few words of explanation are in order. The symbol > means "greater than". The assertion 6 > 2 (which is read "six is greater than two") is certainly true. The Apple uses the number 1 to indicate truth.

If you type

```
PRINT 6 > 2
```

the computer will reply with a one. The assertion 55 > 78 is false. The Apple uses the number Ø to indicate falsehood. If you type

```
PRINT 55 > 78
```

the computer will reply with a zero.

The symbol < means "less than", and you can make statements using it as well. Here is the full set of symbols used in making assertions:

```
>    greater than
<    less than
=    equal to
>=   greater than or equal to
<=   less than or equal to
<>   not equal to
```

To type the symbols for "greater than or equal to" and "less than or equal to" on your Apple keyboard, you must first type either a < or a > and then type an =. To type the symbol for "not equal to" you must type a < and then a >.

Think about and then test to see which of these assertions are true, and which are false.

```
5 <> 5
6 > 2
8 > 8
8 <= 8
9534 = 4359
5 < 8
45 >= -4
-8 < -7
-2 >= -5
9 <> -9
```

Assertions can include variables and expressions as well as numbers. For example

```
PRINT (45 * 6) <> (45 + 6)
```

will print the value 1 since 27Ø is not equal to 51 (remember that 1 means the assertion is true).

You have seen that the Apple can tell truth from falsehood in simple assertions about numbers. However, an assertions such as ABLE > BAKER may be true or false, depending on the value of the two variables, ABLE and BAKER. If

ABLE = 5
and
BAKER = 9
then the assertion
ABLE > BAKER
is false. But if
ABLE = -8
and BAKER = -15
then the assertion
ABLE > BAKER
is true.

Assertions have the numerical values of zero or one. They can be used in arithmetic expressions instead of ones and zeros. For example,

```
PRINT 3 + (4 > 2)
```

will print the value 4. The statement

```
T = 4 <> 3
```

gives T the value 1, since 4 does not equal three, and thus the assertion 4 <> 3 has the value 1. The statement

```
HOT = 67 = 19
```

looks very confusing at first, but it is easily understood. Since 67 does not equal 19, the assertion 67 = 19 is false and has the value zero. The value of Ø is given to the variable "HOT."

As we have seen, the Apple uses 1 to mean true, and Ø to mean false. If something is not true, it is false. If something is not false, it is true. This may not always be the case in real life, but it is always the case with computers. Try this on the Apple:

```
PRINT NOT 1
```

and then try

```
PRINT NOT Ø
```

The computer agrees: not true is false and not false is true. Of course, you can use expressions instead of ones and zeros. For example

```
PRINT NOT (45 > 3)
```

The sentence
TRIANGLES HAVE THREE SIDES.
is true. And the sentence
THIS BOOK IS IN ENGLISH.
is true. Consider the sentence
TRIANGLES HAVE THREE SIDES AND THIS BOOK IS IN ENGLISH.
Is this sentence true or false? It is true. Consider the sentence
TRIANGLES HAVE EIGHT SIDES AND THIS BOOK IS IN ENGLISH.
This sentence, as a whole, is false. Lastly, consider the sentence
TRIANGLES HAVE EIGHT SIDES AND THIS BOOK IS IN SWAHILI.
This sentence is also false. In general, when you combine two sentences, or assertions, by joining them with the word AND, you find that

a. The new sentence is true if both original sentences were true.
b. The new sentence is false if at least one of the original sentences was false.

The Apple knows how to determine whether an assertion containing the connecting word AND is true or false. Test your computer with the following instructions; try to predict each answer:

```
PRINT 1 AND 1
PRINT 1 AND Ø
PRINT Ø AND 1
PRINT Ø AND Ø
PRINT (3 > 2) AND Ø
PRINT (NOT Ø) AND (4 = 5)
```

Is this sentence true or false?

A TRIANGLE HAS THREE SIDES <u>OR</u> THIS BOOK IS IN LATIN.

It's true. A triangle does have three sides, even if this book isn't in Latin, so the sentence as a whole is true. Quod erat demonstrandum. In general, when you combine two sentences by joining them with the word OR, you find that

a. The new sentence is true if one or both of the original sentences were true.
b. The new sentence is false if both of the original sentences were false.

The Apple can also determine if an assertion containing OR is true or false. Try each of these on your Apple--after figuring out what the answer should be.

```
PRINT 1 OR 1
PRINT 1 OR Ø
PRINT Ø OR 1
PRINT Ø OR Ø
PRINT (4 <> 5) OR (4 = 5)
PRINT 1 OR (Ø AND 1)
PRINT ((3 > 4) OR (54 < 337)) AND (NOT Ø)
```

AND, OR, and NOT will become very useful in the next section.

You have already found that in the statement

```
PRINT 1 OR Ø
```

the computer regards 1 as true and Ø as false. Now try this:

```
PRINT 23 OR Ø
```

and this:

```
PRINT -247 AND 327Ø7.61
```

In assertions, the Apple regards not only 1, but any number which is not zero, as true. However, when the computer figures out the value of an assertion, that value will always be either Ø or 1.

While the following box gives the precedence rules for AND, OR, and NOT, we strongly recommend that you use parentheses to make your statements clear.

ORDER OR PRECEDENCE FOR OPERATIONS USED SO FAR IN THIS TEXT:

1. ( )
2. NOT - (for negative values)
3. ^
4. * /
5. + -
6. > < = >= <= <>
7. AND
8. OR

THE IF STATEMENT

Suppose you want to print out integers from 1 through 1Ø, one number to a line. An obvious way to do this is

```
NEW
21Ø  PRINT 1
22Ø  PRINT 2
23Ø  PRINT 3
```

and so on. But this would require 1Ø statements, and if you wanted to print the integers from 1 through 2ØØ this way, it would require 2ØØ statements. Using what you have already learned, you can PRINT integers from 1 on up in just four statements by using a loop:

```
2ØØ N = 1
21Ø  PRINT N
22Ø N = N + 1
23Ø  GOTO 21Ø
```

There is a way to control how long a loop runs. What you want is a statement that does a GOTO if N is, for example, less than 11, but doesn't do the GOTO if N is greater than 11. The answer to your wishes is the IF statement. If a condition is met, the computer will skip the GOTO instruction and execute the instruction on the next line. If there is no next line, the program will end.

Here is a program that counts from 1 to 1Ø and then stops:

```
2ØØ N = 1
21Ø  PRINT N
22Ø N = N + 1
23Ø  IF N < 1Ø THEN GOTO 21Ø
```

In general, the IF statement works like this:

IF arithmetic expression THEN any statement

First, the arithmetic expression is evaluated. If it evaluates to zero (false) all the rest of that program line is ignored, and the computer goes on to the next line. If the arithmetic expression is not zero (true) the remaining portion of that program line is executed.

The IF statement is a very powerful one, and it will appear in almost every program you write. For the fun of it, try this program:

```
NEW
400  GR
410 ROW = 1
420  COLOR= ROW
430  HLIN 0,39 AT ROW
440 ROW = ROW + 1
450  IF ROW < 16 THEN GOTO 420
```

SAVING PROGRAMS ON DISKETTE

(Skip this section if you are not using a disk drive.)

At this point, you may wish to save on diskette some of the programs you have been using. Simply type in the program (after typing NEW) and then type

```
SAVE
```

followed by the name you want to use when refering to the program, and then, of course, a RETURN . For instance, if you wanted to SAVE the program above and call it STRIPES, you would type

```
SAVE STRIPES
```

and the program would be saved on the diskette under the name STRIPES. Once the program is SAVEd, type

```
CATALOG
```

followed by RETURN to see the name of your program listed with the other programs on the diskette. You would then be able to RUN the program called STRIPES from that diskette anytime you wished by typing

```
RUN STRIPES
```

Try SAVEing the program of your choice. If you accidentally mistype the command (and probably get ?SYNTAX ERROR from the Apple) simply retype the line correctly.

Sometimes it is desirable to load a program into the Apple's memory without actually RUNning the program. For example, you may wish to modify the program before you RUN it. The LOAD command is useful in this instance. To use it, simply type

```
LOAD
```

followed by the name of the program you wish to LOAD. For instance if you wanted to LOAD the program called STRIPES, you would type

```
LOAD STRIPES
```

If you change the program and then wish to RUN the new version which is in the Apple's memory but not SAVEd on the diskette, remember to type only

```
RUN
```

If you forget and type

```
RUN STRIPES
```

the old version stored on the diskette will be re-LOADed, erasing the new version in memory.

You can use the LOAD and SAVE commands to move programs from one diskette to another by LOADing a program from one diskette and SAVEing the program to another diskette. Practice using the LOAD and SAVE commands.

SAVING PROGRAMS WITH A CASSETTE RECORDER

(Skip this section if you are not using a cassette recorder.)

To SAVE on cassette tape a program you wish to use later, first insert a blank cassette into your recorder and rewind the tape to the beginning, where your recorded program will be easy to find. On the recorder, hold down the PLAY lever while pressing down the RECORD lever. Both should stay down. Back at the Apple, type

```
SAVE
```

When you press RETURN , the blinking cursor will disappear. After 1Ø or 15 seconds, the computer will give a "beep" to let you know the recording has begun. Another "beep" will sound when the recording is completed, and the cursor will reappear. Push the STOP lever on the recorder, rewind the tape to the beginning, and you are ready to go back to programming. Your program in the computer has not been affected in any way by SAVEing it.

MORE GRAPHICS PROGRAMS

Earlier, you put four colors at the corners of the screen. Now type this program:

```
NEW
19Ø  GR
2ØØ  COLOR= 9
21Ø  PLOT Ø,Ø
22Ø  PLOT Ø,39
23Ø  PLOT 39,39
24Ø  PLOT 39,Ø
```

LIST the program to check that you typed it in correctly, and then RUN it. Quick, isn't it? To change the colors, just change line 2ØØ, and RUN the program again. Try to LIST the program. Notice that the listing slips through the narrow window at the bottom of the screen. This will continue to happen unless you type

```
TEXT
```

to get out of GRaphics mode before you try to LIST.

The following program makes the entire screen a solid color.

```
NEW
2ØØ  GR
21Ø  COLOR= 9
22Ø COLUMN = Ø
23Ø  VLIN Ø,39 AT COLUMN
24Ø COLUMN = COLUMN + 1
25Ø  IF COLUMN < 4Ø THEN GOTO 23Ø
```

Here's a blow-by-blow explanation of what happens when you RUN this program. Line 2ØØ sets GRaphics mode. The color is chosen

in line 21Ø. The program is to start in the screen's column Ø and work its way over to column 39. Line 22Ø makes sure the program starts in column Ø. At line 23Ø, a vertical line is drawn in column Ø. Now that column Ø is filled with the desired color, line 24Ø increments the column by one. The value of COLUMN is now 1. Line 25Ø checks to see if the new value of COLUMN is less than 4Ø. If it is less than 4Ø, the program goes back to line 23Ø to draw a new vertical line in that column. However, when the value of COLUMN reaches 4Ø (on the Apple screen, the rightmost column is column 39, the program does not go back to line 23Ø, but "drops through" (as we say) and stops executing because it has reached the end of the program.

To eliminate the need to type RUN each time you wish to fill the screen with color, type

```
26Ø  GOTO 21Ø
```

Observe what happens. When will this program stop? LIST the program and make sure you understand what it does before going further in this book.

When you are finished playing with the solid color program, clear the computer and try the following program. It uses a new and very important instruction: the REM statement. "REM" stands for "REMark". This statement allows you to put commentary in a program. The computer ignores any REM statements; they are strictly for the benefit of humans. See how easy it is to follow this program where REMs are used liberally.

```
2ØØ  REM SET GRAPHICS MODE
21Ø  GR
22Ø  REM CHOOSE A COLOR
23Ø  COLOR= 1
24Ø  REM READ PADDLE ZERO
25Ø X = PDL(Ø)
26Ø  REM DIVIDE BY 7 SO MAXIMUM VALUE OF X IS 36
27Ø X = Y / 7
28Ø  REM READ PADDLE ONE
29Ø Y = PDL(1)
3ØØ  REM LIMIT RANGE TO KEEP Y ON THE SCREEN TOO
31Ø Y = Y / 7
32Ø  REM PLOT THE POINT
33Ø  PLOT X,Y
34Ø  GOTO 25Ø
```

After you type RUN, operate the game controls. This program is called the "Etch-a-sketch" (TM) after a device that behaves

similarly. The division by seven is necessary since the PDL function gives values between Ø and 255, whereas the screen can only accept column and row values from Ø to 39. Dividing by seven gives you values from (Ø / 7) = Ø to (255 / 7) = 36.4285715. Then GRaphics mode automatically rounds coordinate values down to the nearest integer whose value is less than or equal to the given value. In other words, the X and Y coordinates are rounded down to integers from Ø to 36 so that the X and Y coordinates can be PLOTted. This method does not utilize the full height or width of the screen. To get the full width of the screen, instead of

```
27Ø X = X / 7
```

you could use the two lines

```
27Ø  IF X > 239 THEN X = 239
275 X = X / 6
```

To get the full height of the screen, you could do the same thing using the Y coordinate.

The IF statement limits the value of X to 239. In the Apple's low-resolution GRaphics mode, 239 / 6 is rounded down from 39.833333 to 39. This use of the IF statement to limit the range of a variable is very common.

FOR/NEXT LOOPS

Loops, whether executed by airplanes or computer programs, have a top and a bottom. In the program

```
NEW
1ØØ NUMBER = Ø
11Ø  PRINT NUMBER
12Ø NUMBER = NUMBER + 1
13Ø  IF NUMBER <= 12 THEN GOTO 11Ø
```

line 11Ø is the top of the loop, and 13Ø is the bottom. The program prints the integers from Ø to 12 inclusive. The number 12 is the limit of the loop. Another way to write a loop is to use a statement we have not discussed yet: the FOR statement. We can use this statement to rewrite the previous program.

```
2ØØ  FOR NUMBER = Ø TO 12
21Ø  PRINT NUMBER
22Ø  NEXT NUMBER
```

Use

```
RUN 2ØØ
```

to execute this program. If you just type RUN, the program at line 1ØØ (the lowest line number around) will be executed.

Line 2ØØ contains the new FOR statement. It starts by setting NUMBER to the value Ø. This is exactly the same task that line 1ØØ performed. Then line 21Ø is executed. The bottom of the FOR loop is in line 22Ø. The variable NUMBER is increased by 1 and then compared to the upper limit specified in the FOR statement: 12. If NUMBER is not over the limit, execution continues at the statement immediately following the FOR. If the variable is over the limit, the program drops through (out of the loop) to the statement after the NEXT. In this case, the program drops through and, not finding any more lines, terminates the program.

The most obvious advantage of the FOR/NEXT method of constructing loops is that it saves a statement. The most important advantage is that you don't have to think so hard when writing a loop if you use a FOR/NEXT loop. If you wanted to draw a series of horizontal lines on the screen, using each of the 15 colors on the screen, you could type

```
3ØØØ  GR
3Ø1Ø  FOR N = Ø TO 15
322Ø  COLOR= N
3Ø3Ø  HLIN Ø,39 AT N
3Ø4Ø  NEXT N
```

Another advantage is that it is much easier to read a single FOR statement than to look through three statements to figure out what a loop is doing. To find the bottom of a FOR/NEXT loop, all you have to do is look for a NEXT which has the same variable as the FOR.

It might be well to mention that, although you should know how the FOR statement works, you don't have to use it. It doesn't add any new abilities to those you already have. It just makes some programs easier to write (for some people).

At this point, if you have been following along on your Apple II, you should remove the portion of the programs between lines 3ØØØ and 3Ø4Ø, inclusive. So type

```
DEL 3ØØØ,3Ø4Ø
```

and then LIST the program to check the results.

To PRINT just the even numbers from Ø to 12, you could use the program

```
1ØØ THING = Ø
11Ø  PRINT THING
12Ø THING = THING + 2
13Ø  IF THING <= 12 THEN GOTO 11Ø
```

The secret is in line 12Ø, where 2 is added to THING. We say that the loop steps by two. To step by two in a FOR loop, you would type

```
2ØØ  FOR THING = Ø TO 12 STEP 2
```

The rest of the program would look like lines 21Ø through 22Ø on the previous page except that the name NUMBER would have to be changed, wherever it occurs, to the name THING. Try it. The STEP may be any number in the range of the Apple. It can even STEP backward, for example

```
2ØØ  FOR THING = 39 TO 15 STEP -3
```

Type this line and try it by typing

```
RUN 2ØØ
```

You should play with the FOR statement for a while, if you wish to learn to use it. A number of the example programs from this point on will use the FOR statement.

Along with the convenience of the FOR statement come some limitations. For example, FOR/NEXT loops may be nested, but may not cross. Here are a few examples which demonstrate the idea.

```
NEW
3ØØ  GR
31Ø  FOR HUE = 1 TO 15
32Ø  COLOR= HUE
33Ø  FOR ROW = Ø TO 39
34Ø  HLIN Ø,39 AT ROW
35Ø  NEXT ROW
36Ø  COLOR= HUE - 1
37Ø  FOR COLUMN = Ø TO 39
38Ø  VLIN Ø,39 AT COLUMN
39Ø  NEXT COLUMN
4ØØ  NEXT HUE
```

This program is an example of two-level nesting. Think about it and RUN this program before going on to the next. Remember, when writing programs using FOR statements, each FOR must have a matching NEXT.

A WRONG PROGRAM

```
NEW
500  FOR N = 10 TO 20
510  PRINT N
520  FOR J = 30 TO 40
530  PRINT J
540  NEXT N
550  NEXT J
```

This program won't work. Its loops are crossed, which not only gives an error message, but doesn't make any sense. Whenever you find yourself writing crossed loops, your thinking has gotten tangled. If you are sure you know what you are doing, and still want to cross loops, use loops made with IF statements. You can cross those all you want, for what good it will do you.

A LAST EXAMPLE OF NESTED LOOPS:

```
NEW
300  GR
310 HUE = 0
320  FOR COLUMN = 0 TO 35 STEP 5
330  FOR LINE = 0 TO 30 STEP 10
340 HUE = HUE + 1
350  IF HUE > 15 THEN HUE = 0
360  COLOR= HUE
370  FOR ROW = LINE TO LINE + 9
380  HLIN COLUMN,COLUMN + 4 AT ROW
390  NEXT ROW
400  NEXT LINE
410  NEXT COLUMN
```

This program has three-level nesting and draws quilts. Note that COLOR can't be used as a FOR/NEXT variable. COLOR is a reserved word in Applesoft. Try running the program in text mode by removing line 300. What happens?

GETTING FLASHY

If you're bored with plain old white print on a black background, you will especially enjoy this section. Type

```
INVERSE
```

and take a look at the Applesoft prompt and cursor. The prompt character should be black on a white background. Now type a simple program such as

```
NEW
100  PRINT "BLACK AND WHITE IN COLOR"
```

and RUN it. Isn't that more exciting? Now type

```
FLASH
```

and RUN the program again. Now that is flashy.

Notice that INVERSE and FLASH affect only the computer's output. Characters which appear on the screen as you type them are not changed. These commands can be used in both immediate and deferred execution. Experiment with them. After using the INVERSE and FLASH commmands for a while you may decide that white on black is not so boring after all. If you do decide you prefer white on black, type

```
NORMAL
```

to return to normal text mode.

PRINTS CHARMING

As an experiment, type this program and see what it does when you RUN it.

```
NEW
100  PRINT "HELLO"
110  GOTO 100
```

Stop the program with CTRL C. Then change line 100 by just one symbol

```
100  PRINT "HELLO",
```

and RUN the program again. As you can see, this PRINTs the word in columns. Now substitute a semicolon (;) for the comma (,)

```
100  PRINT "HELLO";
```

and RUN the program again. This time the output is packed. This means that there are no spaces between what you told the computer to PRINT. It prints HELLO after HELLO, until the screen is quickly filled.

Change the program by adding this statement

```
90 V = 99
```

and changing line 100 to read

```
100  PRINT V
```

RUN this program. Now change line 100 to

```
100  PRINT V,
```

and RUN it again. Then change line 100 to

```
100  PRINT V;
```

and observe that the comma and semicolon can also be used with numerical values. The ability to place numbers one after the other without intervening spaces is sometimes quite useful. Commas and semicolons can be used within a PRINT statement. Clear the old program with NEW, and type

```
100 STRIKES = 2
110 BALLS = 3
120  PRINT STRIKES,BALLS
```

You can make clearer output by including messages in the PRINT statement. For example, change line 120 into

```
120  PRINT "THE STRIKES AND BALLS ARE ";STRIKES,BALLS
```

Notice that you probably want to have a space after the word ARE lest the number of strikes gets printed too close to it. If you don't think that the large space between the number of strikes and balls looks nice, you could use the statement

```
120  PRINT "THE STRIKES AND BALLS ARE  ";STRIKES;"
     ";BALLS
```

In this version, a blank is put between the numbers of strikes and balls. Perhaps the prettiest way of doing this (are you trying all of these on your Apple?) is

```
120  PRINT "STRIKES  ";STRIKES;"     BALLS  ";BALLS
```

This gives you a scoreboard-like display.

Let's say that you wanted to PRINT the word HERE starting in the 1Øth column (the screen is 4Ø columns across, by the way), you could use this statement

```
120  PRINT "         HERE"
```

(You have to take our word for it that there are nine blanks before the word HERE). Or you could use the TAB feature. Just as on typewriters, you can set a tab on the Apple. The statement

```
120  PRINT TAB(10)"HERE"
```

has the same effect as putting 9 blanks in the quotes, as we did above. Try it, you'll like it.

By combining the TAB with the FOR loop you can program some nice visual effects. For example:

```
NEW
200  FOR N = 1 TO 24
210  PRINT TAB(N)"X"
220  NEXT N
```

There are 24 (not 4Ø) horizontal printing lines. That, by the way, is why the upper limit in the loop in the program above is 24. TAB cannot be used to move backwards (to the left) on a line. Only forward moves are carried out. To print on a particular line, you can vertical tab (VTAB) to that line. The top line is line 1, and the bottom line is line 24. VTAB, unlike TAB, is not used within a PRINT statement.

You can tab horizontally with HTAB if you don't want to use a PRINT statement. HTAB works like TAB except that it is not used within a PRINT statement. Also, HTAB can cause printing to begin either to the left or to the right of the current printing position. The leftmost character on a line is in position 1, while the rightmost character is in position 4Ø. On the next page is a short program that demonstrates the use of HTAB and VTAB:

```
NEW
590  HOME
600  FOR X = 1 TO 24
610  FOR Y = 1 TO X
620  HTAB X
630  VTAB Y
640  PRINT "APPLE"
650  NEXT X
660  NEXT Y
670  GOTO 600
```

Before you RUN this program, try (it ain't easy!) to figure out what it will do. It's both surprising and pretty.

TAB works for immediate execution mode, but you can only use VTAB and HTAB in programs. While TAB, HTAB and VTAB act somewhat like the co-ordinates in PLOT, there are some differences. The 40 columns for the TAB instruction are numbered from 1 to 40, as they would be on a typewriter, while the first co-ordinate of a PLOT instruction can run from 0 to 39, which is more convenient for programming graphics. Since characters are taller than the "bricks" we build graphics with, there is room for only 24 lines of printing on the screen. Therefore VTAB's limits are 1 and 24. A zero or a number that is too large or too small for TAB, VTAB or HTAB will give you an

```
?ILLEGAL QUANTITY ERROR
```

The largest value for VTAB is 24, but the largest value for TAB or HTAB is 255. Both TAB and HTAB will tab past the length of the screen line and "wrap around" to the next line. To see this in action, type

```
NEW
300  FOR K = 1 TO 255
310  PRINT TAB(K) K
320  NEXT K
```

Then try replacing lines 310 and 320 with

```
310  HTAB K
320  PRINT K
```

and adding

```
330  NEXT K
```

What happens to the program when you replace HTAB with VTAB?

Downloaded from www.Apple2Online.com

CHAPTER 4
LOTS OF GRAPHICS

74 Talking to a program on the RUN: INPUT and a bouncing ball
78 Off the walls: a program with lots of bounce
79 Making sounds: PEEK(-16336)
81 Noise for the bouncing ball
81 For higher notes
82 Random numbers: RND and INT
84 Simulating a pair of dice
85 Subroutines: drawing horses using GOSUB and RETURN
87 Traces: TRACE, NOTRACE and END
89 A better horse-drawing subroutine
91 High-resolution graphics: HGR, HCOLOR= and HPLOT

TALKING TO A PROGRAM ON THE RUN

Here is a program that makes a dot of color move across the screen, bouncing off the right and left sides.

```
NEW
4ØØ  REM  CHOOSE BALL COLOR
42Ø BALL = 9
44Ø  REM  SET GRAPHICS MODE
46Ø  GR
48Ø  REM  STARTING POSITION
5ØØ XOLD = 2Ø
52Ø  REM  MOVE THE BALL BACK AND FORTH
54Ø XMOVE = 1
56Ø  REM  NEW X POSITION
58Ø XNOW = XOLD + XMOVE
6ØØ  REM  IS BALL ON THE SCREEN?
62Ø  IF (XNOW >  = Ø) AND (XNOW < 4Ø) THEN  GOTO 72Ø
64Ø  REM  CHANGE XMOVE DIRECTION
66Ø XMOVE =  - 1 * XMOVE
68Ø  GOTO 58Ø
7ØØ  REM  PLOT THE NEW BALL
72Ø  COLOR= BALL
74Ø  PLOT XNOW,2Ø
76Ø  REM  ERASE OLD BALL
78Ø  COLOR= Ø
8ØØ  PLOT XOLD,2Ø
82Ø  REM  SAVE NEW BALL
84Ø XOLD = XNOW
86Ø  REM  MOVE AGAIN
88Ø  GOTO 58Ø
```

You should always give some thought to the naming of variables. It may seem that XNEW would be a more convenient name than XNOW until you remember that NEW has a special meaning in Applesoft and is a reserved word. The reason that the variable XMOVE was called "XMOVE" will be evident if you change its value. Try XMOVE = 2 for example. If you set XMOVE too high, the ball will appear to jump wildly across the screen, with no trace between positions.

This kind of program is the basis for many typical TV games. It is worthwhile to spend some time playing with the program, changing this and that, just to see what can be done with it. It would be a good idea to SAVE this program so you won't have to retype the whole thing if you make a fatal change.

When you LIST this program it doesn't all fit on the screen at once, and the program lines scroll upward too quickly to be read easily. One way to see all the program lines is to LIST the program in portions. For example, you could type

```
LIST 400,680
```

and only the lines with numbers greater than 400 and less than 680 would be printed. Then you could LIST the rest of the program. You can also use

to interrupt the program listing. Try LISTing the program and then quickly typing

before the program lines scroll up beyond the top of the screen. To start the listing again, type CTRL S again. CTRL C can also be used to stop the listing. However, CTRL C will abort the listing so that it cannot be continued from where it left off.

You are now in a position to understand the bouncing ball program, but you might have friends who aren't. Suppose you wanted a friend to be able to choose the color of the ball. You could explain how to change line 420, but you'd also have to explain the possible error messages, and what to do if ...well, it would take a bit of explaining. It would be better to let your friend interact with the program. To do this, you can use an INPUT statement. Change line 420 to read

```
420  INPUT BALL
```

When the program executes this statement, a question mark (?) will appear on the screen, followed by the blinking cursor. The Apple will then wait until someone types a number and presses RETURN The number typed will become the value of BALL and the program will resume execution. It might be a good idea to have the computer tell your friend what is expected. You could put in PRINT statements such as

```
280  REM SET TEXT MODE
300  TEXT
320  PRINT "TO SELECT A COLOR FOR THE BOUNCING BALL,"
340  PRINT "TYPE A NUMBER FROM 1 TO 15"
360  PRINT "AFTER THE QUESTION MARK."
380  PRINT "THEN PRESS THE KEY LABELLED 'RETURN'."
```

You may also incorporate a message into the INPUT statement:

```
420  INPUT "WHAT COLOR WOULD YOU LIKE THE BALL TO BE
(1-15)? "; BALL
```

Notice that in an INPUT statement the message must be in quotes and that there must be a semicolon between the message and the variable name. When the INPUT statement contains a message, no question mark is added after it. If you want a question mark to appear, you must include it in the INPUT message.

Your friends can use the backspace and retype keys to correct mistakes in typing, but if they make a mistake and then press RETURN, they will get an error message. If the character entered is not a number,

```
?REENTER
?
```

will appear on the screen. If too great or small a number is entered, the program will either let the ball move to the right side of the screen and then stop, or the message

```
?ILLEGAL QUANTITY ERROR IN 720
```

will appear on the screen, and the program will stop. For the most part, the user will not know how to restart the computer--and shouldn't have to. Therefore you should make the program check that all numbers typed by the user are correct. These lines will do it:

```
424  REM IS BALL BETWEEN 1 AND 15?
428  IF (BALL > 0) AND (BALL < 16) THEN GOTO 460
432  PRINT "THAT WASN'T BETWEEN 1 AND 15."
436  GOTO 420
```

Are you beginning to see why we advised you to leave so much room between line numbers?

It is good programming practice to make a program as foolproof as possible. You have advanced to the point where you are writing error messages for others to read. It may be all right for a programmer like you to read jargon such as "?SYNTAX ERROR", but it is most definitely not all right to force an innocent user to deal with such nonsense.

Each time you use an INPUT statement, your program must check that what the user types is within certain limits, so that the program won't "blow up" or fail in any way. Dealing with the untutored user (and you must assume that users are not programmers) is an art in itself. Use of clear English sentences and careful checking of what the user types are always required.

By the way, you can INPUT several values with one INPUT statement. The statement

```
3000  INPUT X,Y,Z
```

would display a question mark as usual, and then wait for three numbers to be typed in. The first number would be stored in the variable named X, the second number in the variable named Y, and the third in the variable named Z. The three numbers must be separated by RETURN 's or commas, and the last number must be followed by a RETURN .

SAVE your best version of the bouncing ball program, just in case. Then, if you have not done so already, try to add vertical motion to it. Use the new variables Y1, Y2 and YMOVE. A solution is given on the next page, but try to work this out yourself before you look.

When you have this program running the way you want it to, SAVE it on your diskette or on your tape cassette. We will use it again later on.

OFF THE WALLS

Here is one way to make the ball bounce off all four walls. The statements in black are the ones that have been added to or changed from the program which bounced the ball between two walls.

```
280  REM  SET TEXT MODE
300  TEXT
320  PRINT "TO SELECT A COLOR FOR THE BOUNCING BALL,"
340  PRINT "TYPE A NUMBER FROM 1 TO 15"
360  PRINT "AFTER THE QUESTION MARK,"
380  PRINT "THEN PRESS THE KEY LABELLED 'RETURN'."
400  REM  CHOOSE BALL COLOR
420  INPUT "WHAT COLOR WOULD YOU LIKE? ";BALL
424  REM  IS BALL COLOR 1-15?
428  IF (BALL > Ø) AND (BALL < 16) THEN  GOTO 46Ø
432  PRINT "THAT WASN'T BETWEEN 1 AND 15."
436  GOTO 42Ø
440  REM  SET GRAPHICS MODE
460  GR
480  REM  STARTING POSITION
5ØØ XOLD = 2Ø
51Ø YOLD = 38
52Ø  REM  MOVE BALL BACK AND FORTH
54Ø XMOVE = 1
545  REM  MOVE BALL UP AND DOWN
55Ø YMOVE = 1
56Ø  REM  NEW X POSITION
58Ø XNOW = XOLD + XMOVE
6ØØ  REM  IS BALL ON THE SCREEN?
62Ø  IF (XNOW >  = Ø) AND (XNOW < 4Ø) THEN  GOTO 686
64Ø  REM  MOVE BALL LEFT
66Ø XMOVE =  - 1 * XMOVE
68Ø  GOTO 58Ø
684  REM  NEW Y POSITION
686 YNOW = YOLD + YMOVE
688  REM  IS BALL ON THE SCREEN?
69Ø  IF (YNOW >  = Ø) AND (YNOW < 4Ø) THEN  GOTO 72Ø
692  REM  MOVE BALL UP
694 YMOVE =  - 1 * YMOVE
699  GOTO 686
```

```
7ØØ  REM  NEW BALL POSITION
72Ø  COLOR= BALL
74Ø  PLOT XNOW, YNOW
76Ø  REM  ERASE THE OLD BALL POSITION
78Ø  COLOR= Ø
8ØØ  PLOT XOLD, YOLD
82Ø  REM  SAVE BALL POSITION
84Ø XOLD = XNOW
85Ø YOLD = YNOW
86Ø  REM  MOVE AGAIN
89Ø  GOTO 58Ø
```

As you will see when you RUN this program, the result is a bit repetitive. You can alter the pattern of bouncing by changing the starting values of XOLD and YOLD (lines 5ØØ and 51Ø), but here is a change you might like better:

```
58Ø XNOW = XOLD + XMOVE * PDL(Ø) / 7Ø
686 YNOW = YOLD + YMOVE * PDL(1) / 7Ø
```

To see what this does, play with the paddles.

One more suggestion. Why not have another INPUT, giving a value to a variable called BACKGROUND? Fill the screen with the color BACKGROUND once, at the beginning of your program (right after GR). Then, to erase the old ball position, use

```
78Ø  COLOR= BACKGROUND
```

or even

```
78Ø  COLOR= BACKGROUND + 3
```

SAVE your favorite version of this program.

MAKING SOUNDS

Clicks, ticks, tocks, and various buzzes are easily generated. You can make sounds on your Apple if you tap it, scratch your fingers across it or drop it, but the sounds covered in this manual are produced by programming it. So go to a quiet place, and try working through this section.

To construct any sound-producing program on the Apple, you will need this magic formula:

```
150 SOUND = PEEK(-16336)
```

There is no easy explanation for this formula. The number, -16336, is related to the "memory address" of the Apple's loudspeaker, and was built into the electronics of the computer. You are just going to have to look this number up when you need it.

PEEK returns the numerical code stored at a certain location in the computer. At most locations PEEK only returns a numerical value, but at some locations, such as -16336, it can cause something to happen. In this case, it causes the speaker to make a click. Every time the program executes this statement, the Apple will produce a miniscule "click." RUN the program, and listen to your computer closely.

Now add this line:

```
160  GOTO 150
```

and RUN the program. No problem hearing this!

To make your program beep for a limited period of time, add statements such as

```
140  FOR BEEP = 1 TO 100
160  NEXT BEEP
```

Try it.

A tone is generated by a rapid sequence of clicks. Any program that uses PEEK(-16336) repeatedly will generate some sort of noise. Since -16336 is such a bother to type, we will insert another statement that will allow us to substitute a symbol which is easier to type. Enter the statement

```
100 S = -16336
```

To produce a nice, resonant click, change line 150 to

```
150 SOUND = PEEK(S) - PEEK(S) + PEEK(S) - PEEK(S) +
PEEK(S) - PEEK(S)
```

Different numbers of PEEKs in the statement will produce different quality clicks. Try RUNning some variations. For

more buzzy tones, put one of your variations into a loop. In general, the faster the loop, the higher the pitch.

Now, to use these sounds, LOAD the bouncing-ball program called Off the Walls back into the computer. Try adding a "bounce" sound each time the ball rebounds from a wall.

One possible solution is given on the next page, but try to work it out for yourself, first. (Hint: a bounce occurs whenever either XMOVE or YMOVE changes value.)

NOISE FOR THE BOUNCING BALL

Here is one way to make the bouncing audible. Add these lines to the Off the Walls program:

```
240  REM SET S TO ADDRESS OF SPEAKER
260 S = -16336
663  REM BOUNCE NOISE
665  FOR B = 1 TO 5
670 BOUNCE = PEEK(S) - PEEK(S) + PEEK(S) - PEEK(S)
675  NEXT B
695  REM BOUNCE NOISE
696  FOR B = 1 TO 5
697 BOUNCE = PEEK(S) - PEEK(S) + PEEK(S) - PEEK(S)
698  NEXT B
```

Now try your own sounds. Why not make a different sound off each wall?

FOR HIGHER NOTES, MULTIPLE STATEMENTS ON ONE LINE

To get still higher tones, another feature of Applesoft BASIC can be introduced. It is possible to put more than one statement on the same line. Try this one-line program:

```
NEW
50 S = PEEK(-16336) : GOTO 50
```

The colon (:) can be used to separate statements in any program where you wish to have more than one statement on a line. However, only the first statement on the line has a statement number, so you can only branch to the first statement with a GOTO.

Now add

```
40  FOR PAUSE = 1 TO 2500 : NEXT PAUSE
```

The advantages of multiple statements with a common line-number are these:

1. The statements are executed faster. (This is an advantage only if you need more speed.)
2. More of your program can fit on the screen.
3. It can save some typing.
4. You can group statements together that collectively perform one function, such as the pause in line 40 above.
5. It requires less memory. (This is an advantage only if you are running out of space, and the computer gives you an ?OUT OF MEMORY or a PROGRAM TOO LARGE message while you are entering a program.)

There are also some disadvantages:

1. The program is harder to read.
2. It is harder to modify or correct the program.
3. You can branch only to the first statement in a line.
4. It is very discouraging to type in a long, multiple statement only to have it return a ?SYNTAX ERROR when the program is RUN, making it necessary to retype the whole statement.

RANDOM NOTES

Try this short program on your Apple.

```
NEW
100  PRINT RND(1)
110  GOTO 100
RUN
```

The RND in line 100 stands for RaNDom. The RND function returns RaNDom numbers. Stop the program with

The numbers generated by this program were RaNDom decimal fractions between zero and one.

Change line 100 to

```
100  PRINT RND(6)
```

and RUN the program.

Again stop the program with CTRL C. Hmmm, interesting. The random numbers are still all between zero and one. Write down the last number that was generated so you'll remember it.

Now change line 1Ø0 again, this time to

```
1ØØ  PRINT RND(Ø)
```

and RUN it. Stop the program and compare what you wrote down with what is now on the screen. RND(Ø) returns the last RaNDom number that was generated.

Random decimal fractions between one and zero can be a little clumsy. Often integers (numbers like 3,6 and 1Ø) are easier to use. To get random integers from Ø to 9 we have to add a few more lines to the program. Type

```
NEW
 9Ø  REM ASSIGNS RND NUMBER TO X
1ØØ X = RND(1)
11Ø  REM MULTIPLIES X BY 1Ø
12Ø X = X * 1Ø
13Ø  REM CHOPS OFF THE FRACTION
14Ø X = INT(X)
15Ø  PRINT X
16Ø  GOTO 1ØØ
```

Line 14Ø introduces the INT function. The statement INT(X) gives the largest integer that is less than or equal to the value of X. For instance, if the value of X is 3.6754, then INT(X) is equal to 3. The parentheses following INT can contain any arithmetic expression or numeric variable.

Now RUN the program. Does it work the way you expected? To change the program so that it generates numbers from one to ten instead of from zero to nine just add one to the value of X by adding this line to your program:

```
145 X = X + 1
```

Try it.

The program may seem a little complicated at first. To see step by step what happens, you can add lots of PRINT statements. Modify your program to look like this one.

```
 9Ø  REM ASSIGNS RND NUMBER TO X
1ØØ X = RND(1) : PRINT "X=RND(1)", X
1Ø5  PRINT
11Ø  REM MULTIPLIES X BY 1Ø
12Ø X = X * 1Ø : PRINT "X=X*1Ø", X
125  PRINT
13Ø  REM CHOPS OFF THE FRACTION
14Ø X = INT(X) : PRINT "X=INT(X)", X
145  PRINT
15Ø  REM ADDS 1 TO THE VALUE OF X
16Ø  X = X + 1 : PRINT "X=X+1"
17Ø  PRINT X
175  PRINT
18Ø  FOR PAUSE = 1 TO 2ØØØ : NEXT PAUSE
19Ø  GOTO 1ØØ
```

RUN this program, and see what it does.

If you want to get fancy, you can condense this program to just one line:

```
1ØØ  PRINT INT (1Ø * RND(1)) + 1 : GOTO 1ØØ
```

Do you know how line 1ØØ works?

SIMULATING A PAIR OF DICE

You can use what you've learned about random numbers to write a program that pretends to be a pair of dice.

```
NEW
1ØØ  PRINT "WHITEDICE",
11Ø  PRINT INT (6 * RND(1)) + 1
12Ø  PRINT "REDDICE",
13Ø  PRINT INT (6 * RND(1)) + 1
```

This program generates random integers from one to six for each die. To reroll the dice, reRUN the program. Can you write a program that uses these "dice" to play a game? Try it.

Try writing a one-line program that generates random numbers from 1 through 5Ø. From Ø through 25. Make up your own numbers. Remember to add (1) to the random number if don't you want to generate zeros.

Here's a colorful way to use RaNDom integers.

```
NEW
200  GR
210  REM CHOOSE A RANDOM COLOR
220  COLOR= INT (16) * RND(1))
230  REM CHOOSE A RANDOM POINT
240 X = INT (40 * RND(1))
250 Y = INT (40 * RND(1))
260  REM PLOT THE RANDOM POINT
270  PLOT X,Y
280  REM DO IT AGAIN
290  GOTO 220
```

Try using RND in other programs. Can you write a program that draws lines in RaNDom colors across the screen?

SUBROUTINES

Imagine that there is a game for which you need a piece that looks like a blue horse with orange feet and a white face. Here is a program that draws such a piece:

```
NEW
1000  REM PROGRAM TO DRAW A BLUE HORSE WITH WHITE FACE
      AND ORANGE FEET
1010  GR
1020  COLOR= 7 : REM LIGHT BLUE
1030  PLOT 15,15
1040  HLIN 15,17 AT 16
1050  COLOR= 9 : REM ORANGE
1060  PLOT 15,17
1070  PLOT 17,17
1080  COLOR= 15 : REM WHITE
1090  PLOT 14,15
```

There is nothing wrong with this program; it does draw a blue horse with orange feet and a white face. Now, suppose you needed to draw another horse somewhere else on the screen. You could rewrite this program with new values for X and Y, but that is a bother. There should be some way of using the same program to put a figure anywhere on the screen without having to rewrite it each time.

The key to doing this begins with the observation that you can move a point which is at coordinates (A,B) to the right by adding to the value of the first coordinate, A in this case. For example, the point (4,17) moves 10 columns to the right if you add 10 to the first coordinate, making the point (14,17).

Likewise, a point moves left if you subtract from the first coordinate (or add a negative value). A simple experiment will show you that adding to, and subtracting from, the second coordinate moves points down and up, respectively.

With these facts in mind, you can rewrite your program to "center" the horse at almost any point (X,Y) on the screen. Why "almost" any point? Because, if you choose a center point at an edge of the screen, the horse will go off the screen, and this might give you an ?ILLEGAL QUANTITY ERROR IN 1Ø3Ø (or some other line number) message. Here is an improved program.

```
NEW
1ØØØ  REM PUT A HORSE ANYWHERE ON THE SCREEN
1Ø1Ø  COLOR= 7 : REM LIGHT BLUE
1Ø2Ø  PLOT X,Y - 1
1Ø3Ø  HLIN X,X + 2 AT Y
1Ø4Ø  COLOR= 9 : REM ORANGE
1Ø5Ø  PLOT X,Y + 1
1Ø6Ø  PLOT X + 2,Y + 1
1Ø7Ø  COLOR= 15 : REM WHITE
1Ø8Ø  PLOT X - 1,Y-1
```

You notice that the GR has been left out. We want to use this part of the program to put several horses on the screen. A GR here would clear the screen before each new horse was drawn.

This program can't be run just as it is. First you must set GRaphics mode, and choose X and Y. A good first try at using the horse program might be:

```
2Ø  GR
3Ø  REM FIRST HORSE CENTER
4Ø X = 12
5Ø Y = 35
```

If you try to RUN this, you do get a horse at the desired location, but the program ends there. We want to put two horses on the screen. What if you could write

```
6Ø
7Ø  REM SECOND HORSE CENTER
8Ø X = 33
9Ø Y = 2
1ØØ
```

Do the portion of the program at line 1ØØØ again and then end.

Wouldn't that be nice and easy? You know that the computer can't read those strange instructions at lines 6Ø and 1ØØ. It can, however, read

```
GOSUB 1ØØØ
```

in Applesoft. A program such as the one starting at line 1ØØØ is called a subroutine. GOSUB 1ØØØ tells the computer to GO to the SUBroutine beginning at line 1ØØØ and start executing at that statement. It also tells the computer to come back to the line that follows the GOSUB statement when it is finished with the subroutine. The computer knows the subroutine is finished when it encounters a RETURN statement. To make your horse-drawing partial-program into a complete subroutine, add the line

```
1Ø9Ø  RETURN
```

Now you can write that "what if you only could" program:

```
2Ø  GR
3Ø  REM FIRST HORSE CENTER
4Ø X = 12
5Ø Y = 35
6Ø  GOSUB 1ØØØ
7Ø  REM SECOND HORSE CENTER
8Ø X = 33
9Ø Y = 2
1ØØ  GOSUB 1ØØØ
```

Now RUN the program. You get an error message:

```
?RETURN WITHOUT GOSUB ERROR IN 1Ø9Ø
```

but otherwise the program seems to RUN fine. In effect, you have added a new statement to Applesoft: a horse-drawing statement. Now you can use the statement

```
GOSUB 1ØØØ
```

to draw one of these special horses at whatever X,Y location you have chosen.

TRACES

The portion of the program from line 1ØØØ to 1Ø9Ø is called a subroutine or subprogram. The portion of the program from line 2Ø to line 1ØØ is called the main program.

To see the program's flow, or path of execution, you can invoke a special feature called TRACE. This special feature can show you why the Apple gave an error message when the horse-drawing program was executed. Add this line to the main program.

```
1Ø  TRACE
```

and, for a moment, delete line 2Ø. Put the Apple into TEXT mode and RUN the program.

The numbers you see on the screen are the line-numbers of each statement as it is executed. You can see how the program begins at line 1Ø, continues through the main program until the subroutine call, then executes the subroutine, goes back to the main program, executes the subroutine again, and, not finding any smaller line numbers, goes to line 1ØØØ and executes the subroutine again. This is where the problem occurs. Do you understand the error message now?

To remedy the problem, add this new line to the program:

```
11Ø  END
```

When the program gets to line 11Ø, it will do just what the line says: end. RUN the program once more. No more error message. As you have just seen, TRACE is very handy when you are having problems with a program. If you want to TRACE only part of a program, you can use the NOTRACE statement. Add this line:

```
65  NOTRACE
```

and the program will be TRACEd only up to the execution of line 65.

TRACE can also be issued in the immediate mode. Simply type

```
TRACE
RUN
```

and your program will be TRACEd.

Once you have issued the TRACE command, whether in immediate mode or as a statement in your program, your program will be TRACEd every time you RUN it, from then on. To stop TRACE, you must issue a NOTRACE command, either in a line of your program, or in immediate mode.

A BETTER HORSE-DRAWING SUBROUTINE

Subroutines should be written so that problems from possible errors do not arise when the program is RUN. One problem with our horse-drawing subroutine is that some values of X and Y will cause the horse to go off the edge of the screen. This can be prevented by a set of statements such as:

```
1Ø12  IF X < 1 THEN X = 1
1Ø14  IF X > 37 THEN X = 37
1Ø16  IF Y < 1 THEN Y = 1
1Ø18  IF Y > 38 THEN Y = 38
```

(Why should the maximum Y value be 38, while X must be limited to 37?)

If there is any attempt to locate a horse off the screen, the horse will be moved to the nearest edge. There are other possible strategies, such as giving an error message and stopping the program. However, our choice has the advantage that it doesn't stop the program, and you can see that something is happening.

Sometimes you want to be able to change the values in a subroutine for different program GOSUBs. For example, a second player may want to place a piece, and that should be a horse of a different color. One way to do this would be to type the whole subroutine again, with different colors. However, let's try using variables rather than numbers. Instead of line 1Ø1Ø saying COLOR = 7, it could say

```
1Ø1Ø  COLOR= BODY
```

Similarly, you could write

```
1Ø4Ø  COLOR= FEET
1Ø7Ø  COLOR= FACE
```

Then the main program could go like this:

```
2Ø  GR
3Ø  REM 1ST PLAYER'S HORSE COLOR
4Ø BODY = 7 : REM LIGHT BLUE
5Ø FEET = 9 : REM ORANGE
6Ø FACE = 15 : REM WHITE
7Ø  REM 1ST PLAYER'S HORSE CENTER
8Ø X = 15
9Ø Y = 3Ø
1ØØ  GOSUB 1ØØØ
```

and so on (be sure to follow with an END statement, before you try to RUN it). That's a lot of statements each time you want a horse, but it is still fewer than if you had to type out the entire horse program each time. For additional programming ease, a rather subtle trick is to have a subroutine that assigns the colors for each player's horse--and have each of those subroutines call the horse-drawing subroutine, in turn.

```
2000  REM DRAWS BLUE HORSE WITH ORANGE FEET AND WHITE FACE
2010 BODY = 7 : REM LIGHT BLUE
2020 FEET = 9 : REM ORANGE
2030 FACE = 15 : REM WHITE
2040  GOSUB 1000
2050  RETURN

2500  REM DRAWS ORANGE HORSE WITH PINK FEET AND GREEN FACE
2510 BODY = 9 : REM ORANGE
2520 FEET = 11 : REM PINK
2530 FACE = 12 : REM GREEN
2540  GOSUB 1000
2550  RETURN
```

Now all you need, to put a blue horse with white a face and orange feet at (10,11), is

```
30  REM 1ST PLAYER'S HORSE
40 X = 10
50 Y = 11
60  GOSUB 2000
```

To put an orange horse at (19,2) all you need is

```
70  REM 2ND PLAYER'S HORSE
80 X = 19
90 Y = 2
100  GOSUB 2500
```

Both the horse-drawing subroutines on the previous page, beginning with lines 2000 and 2500, call another subroutine that begins at line 1000. Things get to be quite efficient at this stage. Once you have written a good subroutine that checks for errors and uses variables that you can set in the calling program (which may be the main program or another subroutine), then you can pyramid other subroutines upon it. This makes main programs much easier to write. Using the three subroutines, it is very easy to put up an attractive display of horses.

But first, another handy routine:

```
3ØØØ  REM CHOOSES RANDOM X,Y
3Ø1Ø X = INT (RND(1) * 37) + 1
3Ø2Ø Y = INT (RND(1) * 38) + 1
3Ø3Ø  RETURN
```

And, now for the main program.

```
1Ø   REM SET GRAPHICS MODE
2Ø   GR
3Ø   REM CHOOSE A RANDOM POINT
4Ø   GOSUB 3ØØØ
5Ø   REM PUT A BLUE HORSE THERE
6Ø   GOSUB 2ØØØ
7Ø   REM CHOOSE ANOTHER RANDOM POINT
8Ø   GOSUB 3ØØØ
9Ø   REM PUT AN ORANGE HORSE THERE
1ØØ   GOSUB 25ØØ
11Ø   REM DO IT ALL AGAIN
12Ø   GOTO 3Ø
```

This is how a main program should look if you are a good programmer: mostly REMs and GOSUBs. The work should be done in relatively short subroutines, each of which is easy to write, and complete in itself. Feel free to use TRACE to see how this sample program does its stuff.

HIGH-RESOLUTION GRAPHICS

We call the kind of graphics you have been using so far, low-resolution graphics. In this section you will learn to use another kind of graphics called high-resolution graphics. This new kind of graphics lets you draw with much more detail than you could with the 4Ø by 4Ø low-resolution grid. The new high-resolution graphics screen is 28Ø by 16Ø plotting points. The horizontal coordinates start with Ø at the left of the screen and end with 279 at the right. Likewise the vertical coordinates go from Ø at the top of the screen to 159 at the bottom.

High-resolution graphics are not difficult to understand. Often high-resolution graphics commands are the same as the corresponding low-resolution graphics commands except for the addition of an H (for High-resolution). A thorough knowledge of low-resolution graphics will be helpful to you in this section.

Type

HGR

to get into high-resolution graphics mode. This command clears the screen to black, leaving four lines at the bottom for text. As with low-resolution graphics, high-resolution graphics allow you to use vertical coordinates that would be in the text area (192 is the maximum), but these points are not shown on the screen at all. If the cursor is not visible, press RETURN a few times until it appears near the bottom of the screen.

The HGR command is not available with cassette or diskette Applesoft. High-resolution graphics are not available in cassette and diskette Applesoft unless your Apple has at least 24K of memory. If you have at least 24K of memory and you wish to use high-resolution graphics with cassette or diskette Applesoft, see Appendix D for more information.

High-resolution graphics are truly wonderful, but you do have to make some sacrifice in order to use them. There are fewer colors available in this new kind of graphics mode. The high-resolution colors go from Ø (black) to 7 (white). The colors are

Ø	black1	4	black2
1	green	5	orange
2	violet	6	blue
3	white1	7	white2

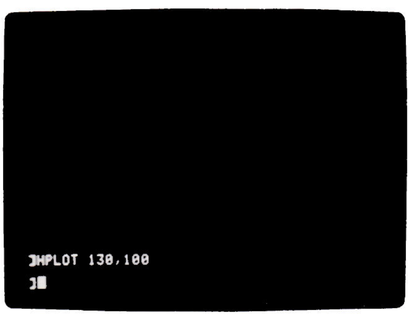
These colors will vary from TV to TV and according to their positions on the screen. A high-resolution dot plotted with color number 3, for example, will be blue if the horizontal coordinate of the dot is even, green if the horizontal coordinate is odd, and white only if both the even and the odd horizontal coordinates are plotted. This is due to the way home TVs work.

If you have an older Apple (prior to S/N 6ØØØ) colors in the second column appear identical to those in the first column.

The only instruction for PLOTting in high-resolution graphics is HPLOT. To try this out, once you have issued the HGR command, type

```
HCOLOR= 3
HPLOT 130,100
```

The last line will plot a white, high-resolution dot at point X = 13Ø, Y = 1ØØ.



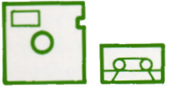
Drawing lines is even easier in high-resolution graphics than it is in low-resolution graphics. You simply HPLOT from one point on the screen TO another point. To draw a line along the top edge of the screen, type

```
HPLOT 0,0 TO 279,0
```

If you want to draw a line from the corner at point 279, Ø to the next corner of the screen all you have to do is type

```
HPLOT TO 279,159
```

and a line appears along the right edge of the screen. When you use this last statement, the new line takes its starting point and its color from the point previously plotted (even if you have issued a new HCOLOR command since that point was plotted). You can "chain" these commands and HPLOT several lines in one statement if you wish.

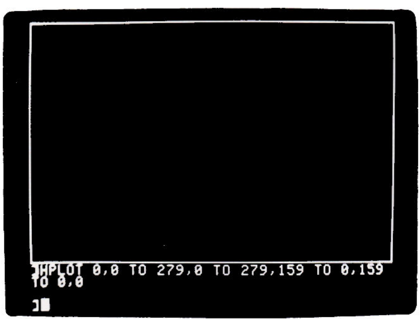


Applesoft on diskette or cassette cannot use this "chaining" feature. If you have Applesoft on diskette or cassette, you must use HPLOT statements specifying at most, two points in order to draw lines.

Clear the screen with HGR, and try this on your APPLE:

```
HPLOT Ø,Ø TO 279,Ø TO 279,159 TO Ø,159 TO Ø,Ø
```

There should be a line around the edge of the screen. If there isn't a continuous line around the edge of the screen, first check that your typing was correct. If the line still isn't there or isn't continuous, change HCOLOR and try again. Some parts of the screen only show up in certain colors on certain TVs.



Not only is drawing lines on your Apple easy, but diagonal lines are just as easy to draw with high-resolution graphics. To draw a line from the top left corner of the screen to the bottom right corner just type

```
HPLOT Ø,Ø TO 279,159
```

Practice drawing high-resolution lines of varying length and color.

Here is a program that makes your Apple into a high-resolution sketching screen.

```
NEW
2ØØ  HGR
21Ø  HCOLOR= 3
22Ø X = PDL(Ø)
23Ø Y = PDL(1)
24Ø  IF Y > 159 THEN Y = 159
25Ø  HPLOT X,Y
26Ø  GOTO 22Ø
```

Line 24Ø is included because the PDL function can return game-controller values up to 255, and the Y coordinate would be off the screen if its value were larger than 159. RUN this program.

This program works, but it would be improved if it were easier to draw a solid line. Those gaps between plotted points are not always desirable. You can improve the program by typing the following lines:

```
220  GOSUB 1000
230  HPLOT X,Y
240  GOSUB 1000
250  HPLOT TO X,Y
260  GOTO 240
1000 X = PDL(0) / .913
1010 Y = PDL(1) / 1.6
1020  RETURN
```

LIST the program and check it carefully to make sure you typed everything correctly. Here's what the program does. High-resolution graphics and HCOLOR are set, and then the program goes to the subroutine beginning at line 1000. The subroutine determines the value of the X and Y coordinates. The game controls each return a maximum value of 255. Because high-resolution graphics uses horizontal coordinates from 0 through 279, the values returned by PDL(0) are divided by .913 to expand their range to the full screen width. Similarly, the values returned by PDL(1) are divided by 1.6 to compress their range down to the range of high-resolution vertical coordinates: 0 to 159. As with low-resolution graphics, the coordinate actually plotted is the nearest integer value less than or equal to the given value. Line 1020 RETURNs to the main program, and then line 230 plots the point X,Y. Next the program goes back to the subroutine, gives X and Y new values and RETURNs to line 250 in the main program where a very short line is drawn from the old point X,Y to the new X,Y. The GOTO in line 260 repeats all of the program except the HGR and HCOLOR instructions, getting new values for X and Y from the position of the game controls, and then PLOTting the new X,Y position. Use CTRL C to stop the program.

There is a reason for drawing lines instead of plotting each point separately. It takes a certain amount of time to plot a point, and when the Apple plots points one at a time it can't always keep up with the game controls. That's why there were spaces between dots when you moved the knobs on the game controls quickly in the first sketching program. Drawing a short line to each new position specified by the game control knobs remedies this: in drawing a line from one point to another, all the points in between are plotted automatically and much more quickly than if they had been plotted one at a time.

Now SAVE the program, and then RUN it.

Here's a program that draws pretty "moire" patterns on your screen.

```
NEW
90  HOME
100  VTAB 24: REM  MOVE CURSOR TO BOTTOM LINE
120  HGR : REM  SET HI-RES GRAPHICS MODE
140 A =  RND (1) * 279: REM  PICK AN "A" FOR CENTER
160 B =  RND (1) * 159: REM  PICK A "B" FOR CENTER
180 N =  INT ( RND (1) * 4) + 2: REM  PICK A STEP SI
     ZE
200  HTAB 15: PRINT "STEPPING BY  ";N;
220  FOR X = 0 TO 278 STEP N: REM  STEP THROUGH A VA
     LUES
240  FOR S = 0 TO 1: REM  2 LINES, FROM X AND X + 1
260  HCOLOR= 7 * S: REM  FIRST LINE BLACK, NEXT WHIT
     E
280  REM  DRAW LINE THROUGH "CENTER" TO OPPOSITE SID
     E
300  HPLOT X + S,0 TO A,B TO 279 - X - S,159
320  NEXT S,X
340  FOR Y = 0 TO 158 STEP N: REM  STEP THROUGH B VA
     LUES
360  FOR S = 0 TO 1: REM  2 LINES, FROM B AND B + 1
380  HCOLOR= 7 * S: REM  FIRST LINE BLACK, NEXT WHIT
     E
400  REM  : DRAW LINE THROUGH "CENTER" TO OPPOSITE S
     IDE
420  HPLOT 279,Y + S TO A,B TO 0,159 - Y - S
440  NEXT S,Y
460  FOR PAUSE = 1 TO 1500: NEXT PAUSE: REM  DELAY
480  GOTO 120
```

This is a rather long program; type it carefully and LIST it in portions (LIST 0,320 for instance) to check your typing. When you are sure it is correct, RUN the program.

If you have Applesoft on diskette or cassette, you will have to type the plotting chains in lines 300 and 420 as separate HPLOT statements. Here's an example of how to do this:

```
300  HPLOT X + S,0 TO A,B
310  HPLOT TO 279 - X - S,159

420  HPLOT 279,Y + S TO A,B
430  HPLOT TO 0,159 - Y - S
```

As you saw in lines 32Ø and 44Ø, one instruction can provide the NEXT for more than one FOR statment. Be careful that you list the NEXT variables in the right order, though, to avoid crossed loops.

To go back to programming, stop the pattern by typing

and then

TEXT

Can you think of ways to change the program? After SAVEing this version on your diskette or cassette recorder try making the value of HCOLOR change randomly. Try drawing first orange then blue lines, or only blue lines.

There is much more to high-resolution graphics than is presented here. When you feel confident using the high-resolution graphics commands presented in this section, refer to chapters eight and nine of the Applesoft BASIC Programming Reference Manual for more information on high-resolution graphics capabilities.

CHAPTER 5
STRINGS AND ARRAYS

100 Stringing along: LEN, LEFT$, RIGHT$, MID$ and CLEAR
105 Concatenation got your tongue?: putting strings together
105 More string functions: VAL and STR$
108 Introducing arrays: DIM
110 Array error messages
111 Conclusion

STRINGING ALONG

Would you like to see your name spelled backwards? So far we have played with graphics and numbers, but computers can also manipulate letters and symbols. Your computer can deal with a single character, or it can handle a whole string of characters at a time. This will seem fairly natural, since we humans also usually deal with characters in bunches. Variables which contain character strings, like numeric variables, have names. String variable names follow the same rules as numeric variable names except that they end with a dollar sign ($). Here are some examples of string variable names:

```
MYNAME$
A$
SENTENCES$
```

The variable A is different from the variable A$, and both can be used in the same program.

If you wish the string variable called NAME$ (pronounced "NAME-dollar") to contain the letters "HARRY S. TRUMAN" you can type

```
NAME$ = "HARRY S. TRUMAN"
```

Notice that the characters that you put into a string variable must be enclosed in quotes. The statement

```
PRINT NAME$
```

will print the contents of the variable NAME$: in this case, the name of the 33rd President of the United States. Thus, when you have a string of characters that you need often, you can store the string in a variable with a short name.

There are several more Applesoft instructions that manipulate strings. Suppose you want to know the length of a string (how many characters it contains). You can type

```
PRINT LEN("HARRY S. TRUMAN")
```

or you can type the equivalent statement,

```
PRINT LEN(NAME$)
```

and the Apple will PRINT the LENgth of the string, in this case 15. Notice that spaces count as characters.

The number of characters in a string may range from Ø to 255. If you try to use more than 255 characters in a string you will just get the ?SYNTAX ERROR or ?STRING TOO LONG ERROR message. A string with Ø characters is called a null string. Refer to the Applesoft BASIC Programming Reference Manual for more information on null strings.

On some occasions you may want to PRINT only a part of NAME$. To do this you can utilize three very handy functions: LEFT$, RIGHT$ and MID$.

If, for instance, you want to PRINT the first five letters in NAME$ you can type

```
PRINT LEFT$(NAME$,5)
```

and

```
HARRY
```

should appear on the screen. If you type

```
PRINT RIGHT$(NAME$,5)
RUMAN
```

will appear. For each program you write that uses string variables, you must assign the string value within the program. Each time you RUN a program, all numeric variables are first set to the value Ø and all string variables are set to contain the null string. Here's a short program that uses the functions LEN and LEFT$.

```
NEW
9Ø NAME$ = "HARRY S. TRUMAN"
1ØØ  FOR N = 1 TO LEN(NAME$)
11Ø  PRINT LEFT$(NAME$,N)
12Ø  NEXT N
```

RUN this program. The RIGHT$ command is just like the LEFT$ command except that it uses the rightmost characters in the string. Now write another program substituting RIGHT$ for LEFT$. What happens when you RUN it?

If you want to use characters starting from the middle of the string instead of the beginning or end, the MID$ function is what you need.

Type

```
PRINT MID$(NAME$,7)
```

Your computer replies with

```
S. TRUMAN
```

since S is the seventh character in the string. Now try this program.

```
NEW
190 NAME$ = "HARRY S. TRUMAN"
200  FOR N = 1 TO LEN(NAME$)
210  PRINT MID$(NAME$,N)
220  NEXT N
```

Do you get what you expect when the program is RUN?

Suppose you want to PRINT just the "Y S. T" from the string called NAME$. To do this you add another argument to the MID$ function.

```
PRINT MID$(NAME$,5,6)
```

The first number (5) specifies the string character space at which the Apple is to begin PRINTing. The second number (6) tells it how many character spaces to PRINT. Thus the instruction is interpreted by the Apple as "find the fifth character space in NAME$, and PRINT six character spaces beginning at the fifth and moving to the right." Change line 210 of the previous program as follows, and RUN it.

```
210  PRINT MID$(NAME$,N,6)
```

Don't go any further in this book until you've thoroughly tested the LEFT$, RIGHT$ and MID$ functions. Or else.

Consider this program.

```
NEW
200 A$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
210  PRINT
220  PRINT "TYPE A NUMBER, FROM 1 THROUGH ";LEN(A$);" "
230  PRINT "AND I WILL TELL YOU WHICH LETTER HAS THAT
     POSITION IN THE ALPHABET.  ";
240  INPUT P
250  IF P > LEN(A$) OR P < 1 THEN GOTO 210
260  PRINT
270  PRINT MID$(A$,P,1);" IS LETTER NUMBER ";P;
     " IN THE ALPHABET."
280  PRINT : PRINT
290  PRINT "TYPE A LETTER, AND I WILL TELL YOU"
```

```
300  INPUT "WHERE IT IS IN THE ALPHABET.  ";X$
310  FOR N = 1 TO LEN(A$)
320  IF MID$(A$,N,1) = X$ THEN GOTO 370
330  NEXT N
340  PRINT
350  PRINT "THAT IS NOT A LETTER OF THE ALPHABET." : PRINT
360  GOTO 290
370  PRINT
380  PRINT X$," IS LETTER NUMBER ";N;" IN THE ALPHABET."
390  PRINT
400  GOTO 210
```

This program illustrates some common programming practices. Notice how it finds the position of a character in a string. This method of using a loop to scan through a string, one position at a time, is very common. Also notice the function of the blank spaces in the PRINT statements. What would happen to the output without these blank spaces? Finally, observe that the program limits are always set by LEN(A$), rather than the actual number of characters in the alphabet. This allows the program to work even if you specify a different "alphabet" in line 200. Try it and see.

You can substitute one string for another with a replacement statement such as

```
X$ = A$
```

This statement copies the contents of A$ into X$. However, you cannot use partial string notation on the left side of a replacement statement. For example, the statement

```
MID$(X$,3,3) = "XYZ"
```

is illegal, but the statement

```
X$ = MID$(A$,24,3)
```

is OK. Only a variable can be on the left side of a replacement statement.

Oh yes--still want to see your name spelled backwards? The program on the next page will do just that.

```
NEW
100  REM PROGRAM TO SPELL YOUR NAME BACKWARDS
110  INPUT "TYPE YOUR NAME AND I WILL SHOW IT TO YOU SPELL
     BACKWARDS  ";N$
120  REM REVERSE ORDER OF LETTERS
130  FOR T = LEN(N$) TO 1 STEP -1
140 R$ = R$ + (MID$(N$,T,1))
150  NEXT T
160  PRINT : PRINT "YOUR NAME SPELLED BACKWARDS IS ";R$
170  PRINT : PRINT
180  GOTO 110
```

RUN this program, trying several different names. After the program executes itself a few times you will notice that there is a something wrong. Line 140 is the key to the problem. If your name is Sally, for instance, you would type it when asked and thus set N$ to SALLY and R$ to YLLAS. Perhaps your friend Joe is there with you and wants to see his name spelled backwards too. The next time the program asks for a name he would type his name, setting N$ to JOE. Line 140 would then set R$ to the old R$ plus N$ spelled backwards, in other words, YLLASEOJ. What is needed is a command that resets string variables to zero, so R$ can be refilled with characters after each GOTO.

Fortunately there is such a command in Applesoft. It is the CLEAR command. CLEAR resets all variables of every size, shape and color to 0. Add this line to your program.

```
175  CLEAR
```

Now RUN the program again.

The CLEAR command can also be used in immediate execution. Type

```
N = 254
PRINT N
```

Now type

```
CLEAR
```

and then

```
PRINT N
```

again. Did your Apple give 0 as the value of N?

CONCATENATION GOT YOUR TONGUE?

It is possible to add a second string onto the end of an existing string using the plus (+) sign. This process is called concatenation. Try the following on your Apple.

```
C$ = "GOOD MORNING"
D$ = C$ + " " + "BILL"
PRINT D$
```

Your Apple will respond with

```
GOOD MORNING BILL
```

Concatenation is especially useful if you wish to take a string apart and then put it back together with slight modifications. For instance, if you wanted to create a new string that was the same as D$ except that the spaces between words would be substituted with dashes, you could type

```
E$ = RIGHT$(D$,4) + "-" + LEFT$(D$,4) + "-" + MID$(D$,6,7)
PRINT E$
```

and

```
BILL-GOOD-MORNING
```

would appear on your screen.

Here's a program that uses concatenation.

```
NEW
100  INPUT "GIVE ME ABOUT HALF OF A SENTENCE.  ";HALF$
110  INPUT "NOW GIVE ME THE SECOND HALF OF THE SENTENCE.
     ";OTHERHALF$
120 WHOLE$ = HALF$ + OTHERHALF$
130  PRINT
140  PRINT WHOLE$
150  PRINT : PRINT : PRINT : GOTO 100
```

And that's how you can do concatenation.

MORE STRING FUNCTIONS

Strings can be made up of almost any kind of character, including numbers. However, like items in a PRINT statement, the characters between the quote marks in a string cannot be interpreted arithmetically even if they are numbers. To see what happens when you try, type

```
C$ = "123"
PRINT C$ + 7
```

Your Apple will confusedly print

```
?TYPE MISMATCH ERROR
```

and not be able to deal with the last statement. We need the help of the

```
VAL
```

(short for VALue) function to alleviate this problem.

The VAL function returns the VALue of the contents of a string as opposed to its actual contents. Type

```
PRINT C$
```

and then type

```
PRINT VAL(C$)
```

Both commands apparently get the same result; however, appearances can be deceiving. You already know that if you type

```
PRINT C$ + 5
```

your Apple will respond with

```
?TYPE MISMATCH ERROR
```

Try typing

```
PRINT VAL(C$) + 5
```

and

```
128
```

appears on the screen. Notice that the string variable name which is the argument of the VAL function must be enclosed in parentheses.

What if you want to put the value of C$ minus 21 into an ordinary (non-string) variable? Simple. Just type

```
Q = VAL(C$) - 21
```

Now type

```
PRINT Q
```

and see what you get. Are the contents of Q as you expect? You can even use VAL to add the numerical value of two different strings. To try this, create a new string

```
K$ = "12"
```

and then type

```
P = VAL(C$) + VAL(K$)
PRINT P
```

Try VAL with different strings, including strings that begin or end with letters.

Sometimes it is necessary to change a number into a string. The STR$ function, which works much like the VAL function in reverse, can be used to make this change. Suppose you want to change the numeric variable P to a string variable. Typing

```
P$ = STR$(P)
PRINT P$
```

will show you how STR$ works. Here is a program that uses STR$ and VAL.

```
300  INPUT "TYPE A NUMBER FROM 1 THROUGH 999999999.   ";N$
310 N = VAL(N$)
320  IF N < 1 OR N > 999999999 THEN GOTO 300
330 N$ = STR$(N)
340  FOR T = LEN(N$) TO 1 STEP -1
350 P$ = P$ + MID$(N$,T,1)
360  NEXT T
370  PRINT : PRINT "ORIGINAL",N$
380  PRINT : PRINT "REVERSED",P$
390 P = VAL(P$)
400  PRINT : PRINT "ORIGINAL + REVERSED = ";N + P
410  CLEAR
420  PRINT : PRINT : GOTO 300
```

Do you understand how the program works? Why are there commas in lines 370 and 380? Try deleting line 330 to see what its effect is. The first four lines of this program demonstrate the

first steps toward making a truly "bomb proof" input routine. See what inputs can still stop this program, and then devise ways to catch those inputs before they can cause the program to stop.

INTRODUCTION ARRAYS

In this section on arrays we use examples from mathematics, but they are from recreational mathematics and require nothing beyond elementary arithmetic.

Arrays enable you to select any element in a table of numbers, and the programming power they give you more than compensates for the bit of thinking and experimenting you must do to become familiar with them.

An array is a table of numbers. The name of this table, called the array name, is any legal variable name: A, for example. The array name A is distinct and separate from the simple variable A.

To create an array, you must first tell the computer the maximum number of elements you want the array to accommodate. To do this you use a DIM statement (DIM stands for DIMension). The elements in an array are numbered from Ø, so to DIMension an array called A that will have a maximum of 16 elements, type

```
DIM A(15)
```

The DIM statement above has given us 16 new variables. They behave exactly like the variables you have come to know and love. They are:

```
A(Ø)
A(1)
A(2)
```

and so on, down to

```
A(15)
```

Although you may find them awkward to type, they can be used just as any other variable is used. The statement

```
A(9) = 45 + A(12)
```

is perfectly correct. The number in parentheses is called a subscript, and the notation A(12) is read "A-sub-twelve." The

subscript can be an arithmetic expression or it can be represented by a variable.

Type the following program. It illustrates the use of variables in the subscript and prints out a display of the contents of each array element.

```
1ØØ  REM DIMENSION ARRAY CALLED 'DAYS', TO HOLD 7 NUMBERS
11Ø  DIM DAYS(6)
12Ø  REM FILL THE ARRAY
13Ø  FOR NUM = Ø TO 6
14Ø DAYS(NUM) = NUM + 1
15Ø  NEXT NUM
16Ø  REM PRINT THE ARRAY ELEMENTS
17Ø  FOR I = Ø TO 6
18Ø  PRINT "DAYS(";I;") = ";DAYS(I)
19Ø  NEXT I
```

If an array is used in a program before it has been DIMensioned, Applesoft reserves space for 11 elements (subscripts Ø through 1Ø). However, it is good programming practice to DIMension <u>all</u> arrays.

Suppose you want to write a program that generates the numbers from one to eight in scrambled order. To accomplish this you need to manipulate tables of data. This is just the kind of thing for which arrays are excellent. The program on the next page accomplishes this.

```
NEW
2ØØ  REM  DIMENSION THE ARRAY
21Ø  DIM GLASS(8)
22Ø  REM  FILL THE ARRAY
23Ø  FOR I = 1 TO 8
24Ø GLASS(I) = I
25Ø  NEXT I
26Ø  REM  SCRAMBLE THE ARRAY AND CHOOSE EACH ELEMENT
27Ø  FOR WINE = 1 TO 8
28Ø  REM  CHOOSE SOME OTHER ELEMENT
29Ø MILK =  INT ( RND (1) * 8) + 1
3ØØ  REM   WAS MILK DIFFERENT FROM WINE?
31Ø  REM  IF NOT, TRY AGAIN"
32Ø  IF MILK = WINE THEN  GOTO 28Ø
33Ø  REM  INTERCHANGE GLASS(WINE) AND GLASS (MILK)
34Ø TEMP = GLASS(WINE):GLASS(WINE) = GLASS(MILK):GLA
     SSMILK = TEMP
35Ø  NEXT WINE
36Ø  REM  PRINT CONTENTS OF ARRAY
37Ø  FOR C = 1 TO 8
38Ø  PRINT GLASS(C)
39Ø  NEXT C
```

Do you understand how this program works? It first fills an array with numbers and then scrambles the contents of the array. Notice that you don't have to start filling the array at zero. Here's a description of what some of the more elusive program lines do. Lines 23Ø through 25Ø fill the array and assign each array element a number corresponding to its array number (GLASS(1) = 1, etc.). Line 27Ø sets the new variable WINE to numbers 1 through 8. Line 28Ø sets variable MILK to random integers from 1 to 8. Then line 3ØØ makes sure that the value of WINE is not equal to the value of MILK at any given time. The contents of variables GLASS(WINE) and GLASS(MILK) are switched in line 31Ø. Finally the array is printed with lines 33Ø through 35Ø.

The switching that occurs in line 31Ø can be thought of like this. Lets say we have two glasses--one is a wine glass (WINE), and the other is a milk glass (MILK). Oh no, there was a mistake. The milk is in the wine glass and the wine is in the milk glass. Luckily we have an extra glass (TEMP). We can pour the milk into the extra glass, then pour the wine into the wine glass, and, finally, pour the milk into the milk glass. Now both drinks have been switched to their proper glasses.

ARRAY ERROR MESSAGES

Here are a few error messages you might generate while programming with arrays.

?REDIM'D ARRAY
This error message occurs when an array is dimensioned more than once in the same program. Often this error occurs because the default dimension has been used, and a dimension statement has been added to the program afterwards.

?BAD SUBSCRIPT ERROR
If an attempt is made to use an array element that is outside the dimension of the array, this error message will occur. For instance, if A has been dimensioned to 25 with the statement DIM A(25), referring to the element A(52) or any other element whose

subscript is less than Ø or greater than 25 will give the ?BAD SUBSCRIPT ERROR.

?ILLEGAL QUANTITY ERROR
You will get this message if you try to use a negative number as an array subscript.

These are some of the ways that you can use arrays. The arrays used here are all one dimensional arrays. You can also use arrays that have two or more dimensions. See the Applesoft BASIC Programming Reference Manual for more a more information on arrays.

CONCLUSION

This book has presented the core of Applesoft BASIC. If you now go through this book again, writing your own programs with the statements that have been presented here, you will solidify your knowledge considerably. The Apple has many more abilities, and once you have mastered those presented here, there are whole new worlds for you to explore.

APPENDICES

114 Appendix A: Summary of Commands

126 Appendix B: Reserved Words in Applesoft

128 Appendix C: Editing Features
128 Left and right arrow keys
128 Pure cursor moves
129 Deleting program lines
129 Clearing the screen
130 Summary of edit features

131 Appendix D: Firmware Applesoft versus Cassette or Diskette Applesoft
131 Introduction
132 General discussion
133 An important note
133 Part 1: The Applesoft II firmware card
134 Part 2: Diskette Applesoft
136 Part 3: Cassette tape Applesoft
137 Part 4: Differences between diskette or cassette Applesoft and firmware Applesoft
140 Part 5: Memory locations used by DOS and by Applesoft BASIC

143 Appendix E: Error Messages

147 Appendix F: The Old Monitor ROM
147 Using the old monitor ROM
148 Recovering from accidental RESETs

APPENDIX A: SUMMARY OF COMMANDS

(This appendix contains both Applesoft and DOS commands.)

The following is a summary of the commands that can be used in the Applesoft BASIC programming language. For more information on these commands, see the Applesoft BASIC Programming Reference Manual.

Arrow Keys → ←

The keys marked with right- and left-pointing arrows are used to edit Applesoft programs. The right-arrow key moves the cursor to the right; as it does, each character it crosses on the screen is entered as though you had typed it. The left-arrow key moves the cursor to the left; as it moves, one character is erased from the program line which you are currently typing, regardless of what the cursor is moving over.

CALL -151

Causes the asterisk prompt to appear indicating that the Apple is now responding to its native language called machine language. If you are not an advanced programmer, you will probably not need to use this command.

CATALOG

Displays on the screen a list of all the files on the diskette in the specified or default drive. The file type and the number of sectors occupied by the file, are indicated to the left of the file name. The file types are:

I	The file is an Integer BASIC program.
A	The file is an Applesoft BASIC program.
T	The file consists of Text: it was created by a WRITE command.
B	The file is a bit-for-bit image of a portion of Apple's memory.

The CATALOG command is a DOS (Disk Operating System) command, not an Applesoft command.

CLEAR

Sets all variables to zero and all strings to null.

COLOR = 12

Sets the color for plotting in low-resolution graphics mode. In the example, color is set to green. Color is set to zero by GR. Color names and their associated numbers are:

Ø black	4 dark green	8 brown	12 green
1 magenta	5 grey	9 orange	13 yellow
2 dark blue	6 medium blue	1Ø grey	14 aqua
3 purple	7 light blue	11 pink	15 white

CONT

If program execution has been halted by STOP, END or CTRL C, the CONT command causes execution to resume at the next instruction (like GOSUB)-- not the next line number. Nothing is cleared. CONT cannot be used if you have
    a) modified, added or deleted a program line, or
    b) gotten an error message since stopping execution.

CTRL C

Can be used to interrupt a RUNning program or a LISTing. It can also be used to interrupt an INPUT if it is the first character entered. The INPUT is not interrupted until the RETURN key is pressed.

CTRL X

Tells the APPLE to ignore the line currently being typed, without deleting any previous line of the same line number. A backslash (\) is displayed at the end of the line to be ignored.

DEL 23,56

Removes the specified range of lines from the program. In the example, lines 23 through 56 will be DELeted from the program. To DELete a single line, say line 35Ø, use the form DEL 35Ø,35Ø or simply type the line number and then press the RETURN key.

DIM NAME$(5Ø)

When a DIM statement is executed, it sets aside space for the specified array with subscripts ranging from Ø through the given

subscript. In the example, NAME$(5Ø) will be allotted 5Ø + 1 or 51 strings of any length. If an array element is used in a program before it is DIMensioned, a maximum subscript of 1Ø is allotted for each dimension in the element's subscript. Array elements are set to zero when RUN or CLEAR is executed.

END

Causes a program to cease execution, and returns control to the user. No message is printed.

ESC I or ESC J or ESC K or ESC M

The Escape key may be used in conjunction with the letter keys I or J or K or M to move the cursor without affecting the characters moved over by the cursor. To move the cursor, first press and then release the ESC key to enter edit mode. Then press the appropriate letter key once for each move in the desired direction. The REPT key can be used to speed the moves by pressing the appropriate letter key and then pressing the REPT key while holding down the letter key.

command	moves cursor one space
ESC I	up
ESC J	left
ESC K	right
ESC M	down

FLASH

Sets the video mode to "flashing", so the output from the computer is alternately shown on the TV screen in white characters on black and then reversed to black characters on a white background. Use NORMAL to return to a non-flashing display of white letters on a black background.

FOR W = 1 TO 2Ø ... NEXT W
FOR Q = 2 TO -3 STEP -2 ... NEXT Q
FOR Z = 5 TO 4 STEP 3 ... NEXT Z

Allows you to write a "loop" to perform any instructions between the FOR command (the top of the loop) and the NEXT command (the bottom of the loop) a specified number of times. In the first example, the variable W counts how many times to do the

instructions; the instructions inside the loop will be executed for W equal to 1, 2, 3, ...2Ø, then the loop ends (with W = 21) and the instruction after NEXT W is executed. The second example illustrates how to indicate that the STEP size as you count is to be different from 1. Checking takes place at the end of the loop, so in the third example, the instructions inside the loop are executed once.

GOSUB 25Ø

Causes the program to branch to the indicated line (25Ø in the example). When a RETURN statement is executed, the program branches to the statement immediately following the most recently executed GOSUB.

GOTO 25Ø

Causes the program to branch to the indicated line (25Ø in the example).

GR

Sets low-resolution GRaphics mode (4Ø by 4Ø) for the TV screen, leaving four lines for text at the bottom. The screen is cleared to black, the cursor is moved into the text window, and COLOR is set to Ø (black).

HCOLOR = 4

Sets high-resolution graphics color to the color specified by HCOLOR. Color names and their associated values are:

Ø black1	4 black2
1 green	5 orange
2 violet	6 blue
3 white1	7 white2

On older Apples, prior to S/N 6ØØØ, the colors in the second column will look the same as those in the first column.

HGR

Only available in the firmware version of Applesoft. Sets high-resolution graphics mode (28Ø by 16Ø) for the screen,

leaving four lines for text at the bottom. The screen is cleared to black, and page 1 of memory is displayed. Neither HCOLOR nor text screen memory is affected when HGR is executed. The cursor is not moved into the text window.

HGR2

Sets full-screen high-resolution graphics mode (28Ø by 192). The screen is cleared to black and page 2 of memory is displayed. Text screen memory is not affected.

HLIN 1Ø, 2Ø AT 3Ø

Used to draw horizontal lines in low-resolution graphics mode, using the color most recently specified by COLOR. The origin (x = Ø and y = Ø) for the system is the top leftmost dot of the screen. In the example, the line is drawn from x = 1Ø to x = 2Ø at y = 3Ø. Another way to say this: the line is drawn from the dot (1Ø,3Ø) through the dot (2Ø,3Ø).

HOME

Moves the cursor to the upper left screen position within the text window, and clears all text in the window.

HPLOT 1Ø,2Ø
HPLOT 3Ø,4Ø TO 5Ø,6Ø
HPLOT TO 7Ø,8Ø

Plots dots and lines in high-resolution graphics mode using the most recently specified value of HCOLOR. The origin is the top leftmost screen dot (x = Ø, y = Ø). The first example plots a high-resolution dot at x = 1Ø, y = 2Ø. The second example plots a high-resolution line from the dot at x = 3Ø, y = 4Ø to the dot at x = 5Ø, y = 6Ø. The third example plots a line from the last dot plotted to the dot at x = 7Ø, y = 8Ø, using the color of the last dot plotted, not necessarily the most recent HCOLOR.

HTAB 23

Moves the cursor either left or right to the specified column (1 through 4Ø) on the screen. In the example, the cursor will be positioned in column 23.

```
IF AGE < 18 THEN A = Ø: B = 1: C = 2
IF ANS$ = "YES" THEN GOTO 1ØØ
IF N < MAX THEN GOTO 25
```

If the expression following IF evaluates as true (i.e. non-zero), then the instruction(s) following THEN in the same line will be executed. Otherwise, any instructions following THEN are ignored, and execution passes to the instruction in the next numbered line of the program. String expressions are evaluated by alphabetic ranking.

```
INPUT A
INPUT "TYPE AGE THEN A COMMA THEN NAME. "; B, C$
```

In the first example, INPUT prints a question mark and waits for the user to type a number, which will be assigned to the numeric variable A. In the second example, INPUT prints the optional string exactly as shown, then waits for the user to type a number (which will be assigned to the variable B) then a comma, then string input (which will be assigned to the string variable C$). Multiple entries to INPUT may be separated by commas or RETURNs.

```
INT(NUM)
```

Returns the largest integer less than or equal to the given argument. In the example, if NUM is 2.389, then 2 will be returned; if NUM is -45.123345 then -46 will be returned.

```
INVERSE
```

Sets the video mode so that the computer's <u>output</u> prints as black letters on a white background. Use NORMAL to return to white letters on a black background.

```
LEFT$("APPLESOFT",5)
```

Returns the specified number of leftmost characters from the string. In the example, APPLE (the 5 leftmost characters) will be returned.

Left Arrow

See "Arrow Keys".

LEN("AN APPLE A DAY")
LEN(B$)

Returns the number of characters in a string, between Ø and 255. In the first example, 14 will be returned.

A = 23.567
A$ = "DELICIOUS"

The variable name to the left of = is assigned the value of the string or expression to the right of the = .

LIST
LIST 2ØØ,3ØØØ
LIST 2ØØ-3ØØØ

The first example causes the whole program to be displayed on the TV screen; the second and third examples cause program lines 2ØØ through 3ØØØ to be displayed. To list from the start of the program through line 2ØØ, use LIST -2ØØ ; to list from line 2ØØ to the end of the program, use LIST 2ØØ- . LISTing is aborted by CTRL C, and the CONT command cannot be used. To stop the program temporarily at some point in the listing, use CTRL S. Use CTRL S again to resume the listing.

LOAD

Reads an Applesoft program from cassette tape into the computer's memory. No prompt is given: the user must rewind the tape and press "play" on the recorder before LOADing. A beep is sounded when information is found on the tape being LOADed. When LOADing is successfully completed, a second beep will sound and the Applesoft prompt character (]) will return. Only RESET can interrupt a LOAD.

LOAD DOW JONES

Attempts to find a program file with the name DOW JONES on the diskette in the specified or default drive. If the program is found, it will be LOADed into the Apple's memory. LOAD erases any program in the Apple before placing the new program in memory. This command, when followed by a file name, is a DOS command.

MID$("AN APPLE A DAY",4)
MID$(DAY$,4,9)

Returns the specified substring. In the first example, the fourth through the last characters of the string will be returned: APPLE A DAY. In the second example, the nine characters beginning with the fourth character in the string will be returned: APPLE A D

NEW

Deletes current program and all variables.

NEXT

See the discussion of FOR...TO...STEP.

NORMAL

Sets the video mode to the usual white letters on a black background for both input and output.

NOTRACE

Turns off the TRACE mode. See TRACE.

PDL(1)

Returns the current value, a number from Ø through 255, of the indicated game control paddle. Game paddle numbers Ø through 3 are valid.

PLOT 1Ø, 2Ø

In low-resolution graphics mode, places a dot at the specified location. In the example, the dot will be at x = 1Ø, y = 2Ø. The color of the dot is determined by the most recent value of COLOR, which is Ø (black) if not previously specified.

PRINT
PRINT A$; "X = "; X

The first example causes a line feed and RETURN to be executed on the screen. Items in a list to be PRINTed should be separated by commas if each is to be displayed in a separate tab field. The items should be separated by semi-colons if they are to be printed right next to each other, without any intervening space. If A$ contains "CORE" and X is 3, the second example will cause

COREX = 3

to be printed.

REM THIS IS A REMARK

Allows text to be inserted into a program as remarks.

REPT

If you hold down the repeat key, labeled REPT, while pressing any character key, the character will be repeated.

RETURN

Branches to the statement immediately following the most recently executed GOSUB.

RIGHT$("SCRAPPLE",5)
RIGHT$(S$,2)

Returns the specified number of rightmost characters from the string. In the first example, APPLE (the 5 rightmost characters) will be returned.

Right Arrow

See "Arrow Keys".

RND(5)

Returns a random real number greater than or equal to Ø and less than 1. RND(Ø) returns the most recently generated random number. Each negative argument generates a particular random number that is the same every time RND is used with that argument, and subsequent RND's with positive arguments will always follow a particular, repeatable sequence. Every time RND is used with any positive argument, a new random number from Ø to 1 is generated, unless it is part of a sequence of random numbers initiated by a negative argument.

RUN 5ØØ

Clears all variables, pointers, and stacks and begins execution at the indicated line number (5ØØ in the example). If no line number is specified, execution begins at the lowest numbered line in the program.

RUN ANNUITY

LOADs the file called ANNUITY from the specified or default drive and then RUNs the program LOADed. When followed by a file name, RUN is a DOS command, not an Applesoft command.

SAVE

Stores the program currently in memory, on cassete tape. No prompt or signal is given. The user must press "record" and "play" on the recorder before SAVE is executed. SAVE does not check that the proper recorder buttons are pushed; "beeps" signal the start and end of a recording.

SAVE ADDRESSES

SAVEs the file currently in memory. If no file called ADDRESSES is found on the diskette in the specified or default drive, a file is created on that diskette and the program currently in memory is stored under the given file name. If the diskette contains a file with the specified file name, and in the same language, the original file's contents are lost and the current program is SAVEd in its place. No warning is given. SAVE, when followed by a file name, is a DOS command.

STR$(12.45)

Returns a string that represents the value of the argument. In the example, the string "12.45" is returned.

TAB(23)

Must be used in a PRINT statement; the argument must be between Ø and 255 and enclosed in parentheses. For arguments 1 through 255, if the argument is greater than the value of the current cursor position, then TAB moves the cursor to the specified printing position, counting from the left edge of the current cursor line. If the argument is less than the value of the current cursor position, then the cursor is not moved. TAB(Ø) puts the cursor into position 256.

TEXT

Sets the screen to the usual non-graphics text mode, with 4Ø characters per line and 24 lines. Also resets the text window to full screen.

TRACE

Causes the line number of each statement to be displayed on the screen as it is executed. TRACE is not turned off by RUN, CLEAR, NEW, DEL or RESET. NOTRACE turns off TRACE.

VAL("-3.7E4A5PLE")

Attempts to interpret a string, up to the first non-numeric character, as a real or an integer, and returns the value of that number. If no number occurs before the first non-numeric character, a Ø is returned. In the example, -37ØØØ is returned.

VLIN 1Ø,2Ø AT 3Ø

In low-resolution graphics mode, draws a vertical line in the color indicated by the most recent COLOR statement. The line is drawn in the column indicated by the third argument. In the example, the line is drawn from y = 1Ø to y = 2Ø at x = 3Ø.

VTAB(15)

Moves the cursor to the line on the screen specified by the argument. The top line is line 1; the bottom line is line 24. VTAB will move the cursor up or down but not left or right.

APPENDIX B: RESERVED WORDS IN APPLESOFT

The following list contains all of the reserved words in Applesoft BASIC. Although most of these words are not covered elsewhere in this manual, the list is handy as a guide for naming variables. Refer to the Applesoft BASIC Programming Reference Manual to find out how to use more of these commands.

&

ABS
AND
ASC
AT
ATN

CALL
CHR$
CLEAR
COLOR=
CONT
COS

DATA
DEF
DEL
DIM
DRAW

END
EXP

FLASH
FN
FOR
FRE

GET
GOSUB
GOTO
GR

HCOLOR=
HGR
HGR2
HIMEM:
HLIN
HOME
HPLOT
HTAB

IF
IN#
INPUT
INT
INVERSE

LEFT$
LEN
LET
LIST
LOAD
LOG
LOMEM:

MID$

NEW
NEXT
NORMAL
NOT
NOTRACE

ON
ONERR
OR

PDL
PEEK
PLOT
POKE
POP
POS
PRINT
PR#

READ
RECALL
REM
RESTORE
RESUME
RETURN
RIGHT$
RND
ROT=
RUN

SAVE
SCALE=
SCRN(
SGN
SHLOAD
SIN
SPC(
SPEED=
SQR
STEP
STOP
STORE
STR$

TAB(
TAN
TEXT
THEN
TO
TRACE

USR

VAL
VLIN
VTAB

WAIT

XPLOT
XDRAW

Applesoft "tokenizes" these reserved words: each word takes up only one byte of program storage. All other characters in program storage use up one byte of program storage each.

The ampersand (&) is intended for the computer's internal use only; it is not a proper Applesoft command. This symbol, when executed as an instruction, causes an unconditional jump to location $3F5.

XPLOT is a reserved word that does not correspond to a current Applesoft command.

Some reserved words are recognized by Applesoft only in certain contexts.

COLOR, HCOLOR, SCALE, SPEED, and ROT
: parse as reserved words only if the next non-space character is the replacement sign, = . This is of little benefit in the case of COLOR and HCOLOR, as the included reserved word OR prevents their use in variable names anyway.

SCRN, SPC and TAB
: parse as reserved words only if the next non-space character is a left parenthesis, (.

HIMEM: must have its colon (:) to be parsed as a reserved word.

LOMEM: also requires a colon (:) if it is to be parsed as a reserved word.

ATN is parsed as reserved word only if there is no space between the T and the N. If a space occurs between the T and the N, the reserved word AT is parsed, instead of ATN.

TO is parsed as a reserved word unless preceded by an A and there is a space between the T and the O. If a space occurs between the T and the O, the reserved word AT is parsed instead of TO.

Sometimes parentheses can be used to get around reserved words:

```
          100 FOR A = LOFT OR CAT TO 15
LISTs as  100 FOR A = LOF TO RC AT TO 15
     but  100 FOR A = (LOFT) OR (CAT) TO 15
LISTs as  100 FOR A = (LOFT) OR (C AT ) TO 15
```

APPENDIX C: EDITING FEATURES

The Left and Right Arrow Keys ← →

The left-pointing arrow key, also called the backspace key, moves the cursor back (left) one space, erasing the character it passes over. If you haven't pressed RETURN at the end of the last line you typed, the backspace key only affects the characters on that line.

Pressing the right-pointing arrow key, also referred to as the retype key, makes the cursor move forward (right), retyping the character it passes over. If you retype a line with the retype key, then press RETURN, the Apple behaves as if you had retyped the line by hand.

You can cause the cursor to move more quickly by pressing the REPT key while pressing one of the arrow keys.

Pure Cursor Moves

The ESC, I, J, K, and M keys are used to move the cursor without affecting any of the characters on the screen. Imagine arrows drawn on the letter keys as illustrated:

Pressing ESC gets you into edit mode. Once you are in edit mode, pressing one of the above-mentioned letter keys will cause the cursor to move one character in the direction of the corresponding arrow. You can use these keys to move the cursor anywhere on the screen.

For faster cursor motion, hold down one of these keys and then hold down the REPT key. The cursor will zip along while both keys are held down. If the cursor reaches the top of the screen, it will stop. If the cursor reaches the bottom of the

screen, it will stop, and the lines will start to scroll upward. If it reaches the right edge, the cursor will disappear and reappear at the left edge, but on the next line. At the left edge, it will reappear on the right, one line up. To return to normal mode, press the space bar once.

Deleting Program Lines

An easy way to delete a program line is simply to type the line number of the line you wish to delete and press RETURN . If you have more than one line to delete, you may wish to utilize the DELete command. To delete, for instance, lines 100 through 200 you would type

```
DEL 100,200
```

All program lines from 100 to 200 inclusive, should then be deleted.

Typing

will delete the line you are currently typing. This is useful when you realize you have made a mistake before you have pressed RETURN .

Clearing the Screen

The following commands affect only what you see on the screen, not what is stored in the Apples memory. Pressing the ESC key once puts you in edit mode.

To clear the screen, type ESC followed by an "at" sign.

The cursor will appear at the top left-hand corner of the screen without the prompt. The prompt character will appear when you press RETURN .

If you are already in edit mode you can clear the screen by simply typing an "at" sign (@). The Apple will return you to normal mode.

The HOME command will also clear the screen. Simply type

HOME

and the cursor will "home" to the top left-hand corner of the screen.

It is also possible to clear only portions of the screen. To clear from a point on the screen to the end of the screen, get into edit mode by pressing ESC . Then use the pure cursor move features to move the cursor to the first character you wish to clear. Press the F key, and all the characters from that point to the end of the screen will be cleared. To clear characters to the end of a line, you must first be in edit mode. Then move the cursor to the first character to be cleared, and press E . In both cases, the Apple will return to normal mode after the command is executed.

Summary of Edit Features

Enter edit mode	Press ESC
Exit edit mode	Press space bar
Move cursor	Press I, J, K or M
Delete a character	Press ←
Retype a character	Press →
Clear from the cursor to the end of a line	Press ESC, then E
Clear from the cursor to the end of the screen	Press ESC, then F
Clear the entire screen	Press ESC, then CTRL and @ P
Stop listing	Press CTRL and S
Resume listing	Press CTRL and S

APPENDIX D: FIRMWARE APPLESOFT VERSUS DISKETTE OR CASSETTE APPLESOFT

Introduction

You do not need to read this appendix at all unless you are using one of the following:

1. the plug-in Applesoft II Firmware Card
2. Applesoft loaded from cassette tape
3. Applesoft loaded from diskette

Some of the material in this appendix may seem highly technical, if this is your first experience with computers. Don't worry if you do not understand everything here, at first. Just read the appropriate parts, looking for information which might help you. At a later time, when you know more about your computer, you may wish to re-read this appendix for more detailed facts.

This appendix will use some special words which may be unfamiliar to you. Many of them describe the Apple's memory, which is used in a surprising number of different ways:

1. To store the diskette or the cassette version of the Applesoft programming language.
2. To store the instructions that make up your program.
3. To store your program's variables, strings, and intermediate and final results.
4. To store various information which the Apple itself needs, about the system, your program, and where different things are stored in memory.
5. To create the text and low-resolution graphics which normally show on your TV screen.
6. To create the high-resolution graphics that can be shown on your TV screen.

Each of these activities, in general, occupies a different portion of the Apple's memory. Information is placed in various memory "pigeonholes", called memory locations. A block of 1Ø24 memory locations is sometimes called 1K of memory. Each memory location has an identifying address, a number which lets the Apple find that location, and the item of information stored there, again. These items of information, which you rarely see in their raw, machine-language form, are called bytes of information. Each byte of information occupies one memory location.

The portion of Apple's memory that is used by a particular activity can be described in terms of the memory locations used, usually specified as a range of memory addresses. If a certain range of memory locations is being used to store your program, for instance, those same memory locations must not be used to create a high-resolution graphics display, or your program will be lost.

In Applesoft BASIC, memory addresses and other numbers are expressed in the usual decimal form. The computer itself uses numbers in a different form called hexadecimal. To aid advanced programmers, memory addresses are sometimes given both in the normal decimal form and in the hexadecimal form. Hexadecimal numbers are usually preceded by a dollar sign ($) and may safely be ignored.

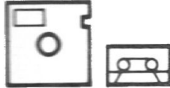
General Discussion

Apple Computer Inc. offers two versions of the BASIC programming language. Integer BASIC, described in the Apple II BASIC Programming Manual, is a very fast BASIC suited for many applications, especially in education, game playing, and graphics. The other version of BASIC is called "Applesoft" and is better suited for most business and scientific applications.

Applesoft BASIC is available in two versions: firmware Applesoft and diskette or cassette Applesoft. Firmware Applesoft comes with Applesoft in ROM (permanent, Read-Only Memory). The Applesoft ROM chips may be installed in sockets DØ through FØ on the Apple's main printed circuit board, or they may be on a plug-in Applesoft II Firmware Card (Apple Part Number A2BØØØ9X). Firmware Applesoft is instantly available when you turn your Apple on or when you type the disk command FP. This saves some time over loading the language from diskette at every use, and saves even more time over loading the language from cassette tape. Aside from this convenience, having Applesoft in ROM frees about 1ØK of Apple's memory for the use of programs.

The main body of this manual assumes your Apple has firmware Applesoft installed in sockets DØ through FØ on the Apple's main printed circuit board. PART 1 of this appendix gives more details about installing and using the plug-in Applesoft II Firmware Card.

If you are using the diskette or the cassette version of Applesoft, the



symbol points out places in this manual where your Applesoft differs from firmware Applesoft. PART 2 of this appendix discusses diskette Applesoft in more detail, and PART 3 gives more details about cassette Applesoft. PART 4 of this appendix summarizes special instructions and notes on the differences between diskette or cassette Applesoft and the firmware Applesoft described elsewhere in this manual. Finally, PART 5 of this appendix gives more technical information for the use of more advanced programmers who need to know how the Apple's memory is used by Applesoft.

An Important Note

One of the functions of the prompt character, besides PROMPTing you for input to the computer, is to identify at a glance which language the computer is programmed to respond to at that time. Here are the prompt characters you are likely to see:

- * for the Monitor program (when you type CALL -151)
- > for Apple Integer BASIC
-] for Applesoft floating-point BASIC.

By simply looking at this prompt character, you can easily tell (if you forget) which language the computer is in.

PART 1. THE APPLESOFT II FIRMWARE CARD

Installation

The Applesoft II Firmware Card simply plugs into a socket inside the Apple. Care must be exercised, however, so follow these instructions exactly:

1) Turn off the Apple's power switch: this is very important to prevent damaging the computer.

2) Remove the cover from the Apple. This is done by pulling up on the cover at the rear edge (the edge farthest from the keyboard) until the two corner fasteners pop apart. Do not continue to lift the rear edge, but slide the cover backward until it comes free.

3) Inside the Apple, across the rear of the main circuit board, there is a row of eight long, narrow sockets called "slots." The leftmost one (looking at the computer from the keyboard end) is slot #Ø; the rightmost one is slot #7. Hold the Applesoft II Firmware Card so that its switch is toward the back of the computer; insert the "fingers" portion of the card into the leftmost slot, slot #Ø. The fingers will enter the slot with some friction, and will then seat firmly. The Applesoft II Firmware Card <u>must</u> be placed in slot #Ø.

4) The switch on the back of the Applesoft II Firmware Card should protrude part way through the slot on the back of the Apple.

5) Replace the Apple's cover: first slide the front edge into place, then press down on the two rear corners until they pop into place.

7) Now turn on the Apple.

Using the Applesoft II Firmware Card

With the Applesoft II Firmware Card's switch in the <u>downward</u> position, the Apple will begin operating in Integer BASIC when you turn the computer on. You will see the prompt character > , which indicates Integer BASIC.

With the switch in the <u>upward</u> position, the Apple will begin running in Applesoft BASIC, instead of Integer BASIC, when you turn the computer on. The prompt character] tells you that you are in Applesoft.

When using the Disk Operating System (DOS), the computer will automatically choose Integer BASIC or Applesoft, as required. It does not matter in which position the switch is set on the Applesoft II Firmware Card.

PART 2: DISKETTE APPLESOFT

With each Disk II, Applesoft II BASIC is provided on the Integer BASIC System Master Diskette, in a program called APPLESOFT. In

addition to the 2K bytes used by the Apple, and the 1ØK bytes used by Applesoft BASIC loaded from diskette, the Disk Operating System (DOS) occupies another 1Ø.5K of memory. Therefore, your computer must contain at least 24K bytes of memory to use the diskette version of Applesoft BASIC.

To use diskette Applesoft, the disk must be booted and at least one disk drive must contain a diskette which has the program APPLESOFT on it (such as the System Master Diskette). Do not use the command RUN APPLESOFT. This command does not properly initialize the language: Applesoft will look as though it is running correctly, but you will be in trouble as soon as you press the RESET key or type a DOS command. Instead, use the DOS command

FP

(for Floating Point BASIC).

When you issue the DOS command

FP

your computer will attempt to load the APPLESOFT language program from the diskette in the default (last used) disk drive. If the program APPLESOFT is not on that diskette, the message

LANGUAGE NOT AVAILABLE

is given. In that case, you have two choices. You may place in that drive a diskette with the APPLESOFT program on it, and type the

FP

command again. Or, if you know that a different drive contains a diskette with APPLESOFT on it, you may issue the FP command with slot and drive parameters. For example, to load Applesoft from the diskette in drive 2 connected to a disk controller card in slot 6, you would type

FP, S6, D2

(see your DOS manual).

When you LOAD or RUN a diskette program written in Applesoft, DOS automatically switches to the correct language. If this necessitates a change to Applesoft, DOS will attempt to find the

APPLESOFT program on the diskette in the disk drive specified by the LOAD or RUN command, or on the diskette in the default disk drive if none is specified. You may, of course, use the FP command to change languages yourself, as described above.

PART 3: CASSETTE TAPE APPLESOFT

Applesoft II BASIC is provided on cassette tape with each Apple II. If your system includes firmware Applesoft or a disk drive, you will not need to use the cassette tape version of Applesoft. Applesoft BASIC loaded from cassette tape occupies approximately 1ØK bytes of memory, and the Apple uses another 2K bytes for text screens, etc. Thus, your computer must contain at least 16K bytes of memory to use the cassette version of Applesoft BASIC.

Getting Started With Cassette Tape Applesoft

Use the following procedure to load Applesoft from your cassette tape unit:

1) Start up Integer BASIC by turning on the computer. You will know you are in Integer BASIC when you see the prompt character > displayed on the TV screen, followed by the blinking square "cursor."

2) Place the Applesoft cassette tape (Part Number A2TØØØ4) in your cassette recorder and rewind the tape to the beginning.

3) Type LOAD

4) Press the recorder's "play" lever to start the tape playing.

5) Back at the Apple keyboard, press the key marked RETURN . When you do this the blinking cursor will disappear. After 5 to 2Ø seconds the Apple will beep, to signal that the tape's information has started to go into the computer. After about 1-1/2 minutes, there will be another beep and the prompt character > followed by a cursor will reappear.

6) Stop the tape recorder and rewind the tape. APPLESOFT is now in the computer.

7) Type RUN and press the key marked RETURN. The screen will display the copyright notice for APPLESOFT II and APPLESOFT's prompt character,].

Typing CTRL B from the Monitor program (prompt character *) will transfer you to Integer BASIC; this will erase Applesoft.

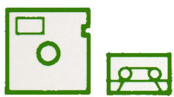
PART 4: DIFFERENCES BETWEEN DISKETTE OR CASSETTE APPLESOFT AND FIRMWARE APPLESOFT

Applesoft on diskette or on cassette tape (Part Number A2TØØØ4) does not work exactly the same as does the firmware version of Applesoft that resides in ROM on the Apple's main printed circuit board (sockets DØ through FØ) or on a plug-in Applesoft II Firmware Card (Part Number A2BØØØ9X). Most of this manual describes the firmware version of Applesoft. The following comments point out how diskette or cassette Applesoft differs from firmware Applesoft.

Firmware Applesoft does not occupy any space in the Apple's memory, and therefore may be used with Apples of almost any size memory.

Diskette Applesoft occupies approximately 1ØK bytes of memory, the Apple uses another 2K bytes for text screens and other system needs, and the Disk Operating System (DOS) occupies another 1Ø.5K bytes. Thus, diskette Applesoft cannot be used in Apples with less that 24K bytes of memory. With diskette Applesoft loaded into the Apple, the lowest memory location available to the user is 12291. See the memory map in PART 5 of this appendix.

Cassette Applesoft occupies approximately 1ØK bytes of memory and the Apple uses another 2K bytes for text screens and other system needs. Thus, cassette Applesoft cannot be used in Apples with less than 16K of memory. With cassette Applesoft loaded into the Apple, the lowest memory location available to the user is 12291. See the memory map in PART 5 of this appendix.



HGR is <u>not</u> available in diskette or cassette Applesoft. The HGR command clears "page 1" of graphics memory (the portion of Apple's memory from location 8192 to location 16383) for high-resolution graphics. Since diskette or cassette Applesoft

partly occupies this portion of memory, attempting to use HGR will erase Applesoft, and your program will be lost. The HGR command can only be used with firmware Applesoft.

The HGR2 command uses "page 2" of graphics memory (the portion of Apple's memory from location 16384 to location 24575). HGR2 can be used both in the firmware and in the cassette version of Applesoft, but is only available if your Apple contains at least 24K of memory. Therefore, in a system with less than 24K of memory, cassette Applesoft does not offer any form of high-resolution graphics.

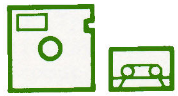
In diskette Applesoft, and in firmware Applesoft used with DOS, the HGR2 command may cause trouble when it clears "page 2" of graphics memory (location 16384 to location 24575). On systems with less than 36K of memory, this will erase a large portion of DOS. Therefore, in a system with less than 36K of memory, diskette Applesoft (and firmware Applesoft used with DOS) does not offer any form of high-resolution graphics.

The command

```
POKE -163Ø1,Ø
```

is used in Applesoft to convert any full-screen graphics mode to mixed graphics-plus-text mode. When issued after HGR2, however, the four lines of text are taken from <u>page 2</u> of text memory, and not from the usual page 1 of text memory that is displayed on the text screen. In the diskette or the cassette version of Applesoft, Applesoft itself occupies page 2 of text memory, so that mixed high-resolution graphics-plus-text is not available.

In diskette or cassette Applesoft, use CALL 11246 (instead of the CALL 6245Ø used in firmware Applesoft) to clear the HGR2 screen to black. Use CALL 1125Ø (instead of firmware Applesoft's CALL 62454) to clear the HGR2 screen to the HCOLOR last HPLOTted.



If executed <u>before</u> you issue the HGR2 command the <u>first</u> time, these CALLs may clear "page 1" of graphics memory, erasing Applesoft.

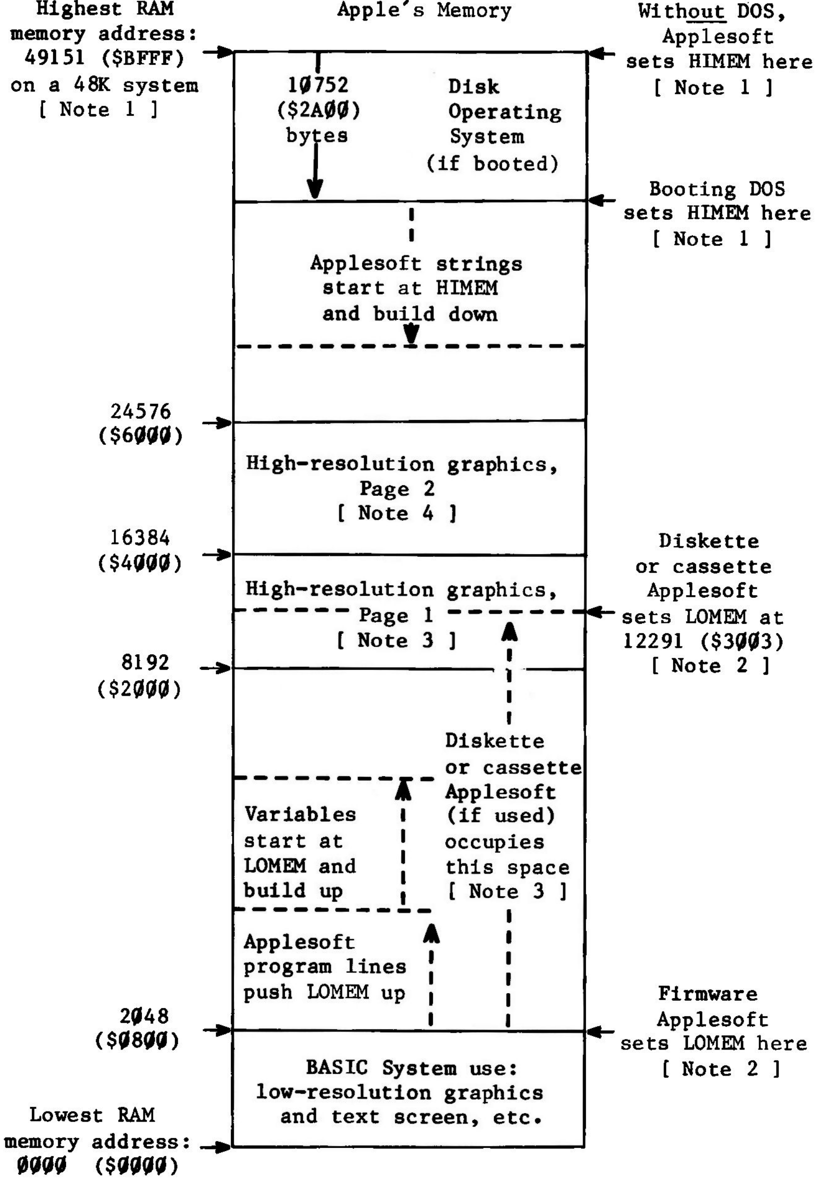
The Applesoft command HPLOT can be used in this form if you are using firmware Applesoft:

```
HPLOT X1,Y1 TO X2,Y2 TO X3,Y3 TO X4,Y4
```

If you are using diskette or cassette Applesoft, you must change such an instruction to this form:

```
HPLOT X1,Y1 TO X2,Y2
HPLOT TO X3,Y3
HPLOT TO X4,Y4
```

PART 5: MEMORY LOCATIONS USED BY DOS AND BY APPLESOFT BASIC



Note 1. HIMEM is the address of the highest memory location available to an Applesoft program. If your system is in Applesoft, the value of HIMEM can be found (low byte first, then high byte) in decimal locations 115 and 116 ($73-$74, hexadecimal). To see the current value of HIMEM, type

```
PRINT PEEK(115) + PEEK(116) * 256
```

Consult the following table for the value of HIMEM set by booting DOS, for systems with various amounts of memory. Increasing MAXFILES will move HIMEM down an additional 595 bytes for each file buffer added. For the locations of other Applesoft program pointers, consult your Applesoft II BASIC Programming Reference Manual, Appendix I.

HIMEM Value Set By Booting DOS

System	Highest RAM address		HIMEM: set by DOS boot	
size	Decimal	Hexadecimal	Decimal	Hexadecimal
16K	16383	$3FFF	5632	$1600
20K	20479	$4FFF	9728	$2600
24K	24575	$5FFF	13824	$3600
32K	32767	$7FFF	22016	$5600
36K	36863	$8FFF	26112	$6600
48K	49151	$BFFF	38400	$9600

The negative equivalent of any positive decimal address n is (n - 65536).

Note 2. LOMEM is the address of the lowest memory location available to an Applesoft program. In Applesoft, the value of LOMEM can be found (low byte first, then high byte) in decimal locations 105 and 106 ($69-$6A, hexadecimal). To see the current value of LOMEM, type

```
PRINT PEEK(105) + PEEK(106) * 256
```

Applesoft automatically sets LOMEM just after the last line of the current stored program, and the first variable starts at LOMEM.

Note 3. Using high-resolution graphics Page 1 (with HGR) erases the contents of memory locations 8192 through 16383. If you are using firmware Applesoft with DOS, an attempt to use high-resolution graphics Page 1 will erase part of DOS unless DOS

sets HIMEM to a value greater than 16383. This means that you cannot use DOS and high-resolution graphics at the same time, unless your system contains at least 32K of memory.

If you are using diskette or cassette Applesoft, an attempt to use high-resolution graphics Page 1 will always erase part of Applesoft. With diskette or cassette Applesoft, you may use high-resolution graphics Page 2, only. However, see Note 4.

Note 4. Using high-resolution graphics Page 2 (with HGR2) erases the contents of memory locations 16384 through 24575. If you are using DOS, an attempt to use high-resolution graphics Page 2 may erase part of DOS unless DOS sets HIMEM to a value greater than 24575. This means that you cannot use DOS and Page 2 high-resolution graphics at the same time, unless your system contains at least 36K of memory.

APPENDIX E: ERROR MESSAGES

All of the error messages that can be generated in Applesoft BASIC are listed here along with their descriptions. See the Applesoft BASIC Programming Reference Manual for more information on the error messages not covered in this manual.

After an error occurs, Applesoft BASIC returns to command level as indicated by the] prompt character and a blinking cursor. Variable values and the program text remain intact, but the program cannot be CONTinued and all GOSUB and FOR loop counters are set to Ø.

When an error occurs in an immediate-execution statement, no line number is printed.

Format of error messages:

Immediate-execution Statement	?XX ERROR
Deferred-execution Statement	?XX ERROR IN YY

In both of the above examples, "XX" is the name of the specific error. "YY" is the line number of the deferred-execution statement where the error occurred. Errors in a deferred-execution statement are not detected until that statement is executed.

The following are the possible error codes and their meanings.

?CAN'T CONTINUE ERROR

Attempt to continue a program when none existed, or after an error occurred, or after a line was deleted from or added to a program.

?DIVISION BY ZERO ERROR

Dividing by zero is an error.

?FORMULA TOO COMPLEX ERROR

More than two statements of the form IF "XX" THEN were executed.

?ILLEGAL DIRECT ERROR

You cannot use an INPUT, DEF FN, GET or DATA statement as an immediate-execution command.

?ILLEGAL QUANTITY ERROR

The parameter passed to a math or string function was out of range. ILLEGAL QUANTITY errors can occur due to:

a) a negative array SUBSCRIPT (e.g., A(-1) = Ø)
b) using LOG with a negative or zero argument
c) using SQR with a negative argument
d) A ^ B with A negative and B not an integer
e) use of MID$, LEFT$, RIGHT$, WAIT, PEEK, POKE, TAB, SPC, ON...GOTO, or any of the graphics functions with an improper argument.

?NEXT WITHOUT FOR ERROR

The variable in a NEXT statement did not correspond to the variable in a FOR statement which was still in effect, or a nameless NEXT did correspond to any FOR which was still in effect.

?OUT OF DATA ERROR

A READ statement was executed but all of the DATA statements in the program had already been read. The program tried to read too much data or insufficient data was included in the program.

?OUT OF MEMORY ERROR

Any of the following can cause this error: program too large; too many variables; FOR loops nested more than 1Ø levels deep; GOSUB's nested more than 24 levels deep; too complicated an expression; parentheses nested more than 36 levels deep; attempt to set LOMEM: too high; attempt to set LOMEM: lower than present value; attempt to set HIMEM: too low.

?OVERFLOW ERROR

The result of a calculation was too large to be represented in Applesoft BASIC's number format. If an underflow occurs, zero is given as the result and execution continues without any error message being printed.

?REDIM'D ARRAY ERROR

After an array was dimensioned, another dimension statement for the same array was encountered. This error often occurs if an array has been given the default dimension 1Ø because a statement like A(I) = 3 is followed later in the program by a DIM A(1ØØ). This error message can prove useful if you wish to discover on what program line a certain array was dimensioned: just insert a dimension statement for that array in the first line, RUN the program, and Applesoft will tell you where the original dimension statement is.

?RETURN WITHOUT GOSUB ERROR

A RETURN statement was encountered without a corresponding GOSUB statement being executed.

?STRING TOO LONG ERROR

Attempt was made by use of the concatenation operator to create a string more than 255 characters long.

?BAD SUBSCRIPT ERROR

An attempt was made to reference an array element which is outside the dimensions of the array. This error can occur if the wrong number of dimensions are used in an array reference; for instance, LET A(1l) = Z when A has been dimensioned using DIM A(2).

?SYNTAX ERROR

Missing parenthesis in an expression, illegal character in a line, incorrect punctuation, etc.

?TYPE MISMATCH ERROR

The left-hand side of an assignment statement was a numeric variable and the right-hand side was a string, or vice versa; or a function which expected a string argument was given a numeric one or vice versa.

?UNDEF'D STATEMENT ERROR

An attempt was made to GOTO, GOSUB or THEN to a statement line number which does not exist.

?UNDEF'D FUNCTION ERROR

Reference was made to a user-defined function which had never been defined.

APPENDIX F: THE OLD MONITOR ROM

Most of this manual assumes that your Apple contains the Autostart ROM, which instantly starts your Apple running in BASIC when you turn on the Apple's power switch. When you turn on your Apple, if it clears its own screen and prints

APPLE][

at the top (and boots your disk, if you have one) then your Apple contains the usual Autostart ROM.

On some older Apples, the RESET key must be pressed before the screen is cleared and the title APPLE][appears.

However, some Apples use a Monitor ROM which works somewhat differently. When you turn on your Apple, if it displays lots of random characters on the screen, and these characters are not cleared away, then your Apple contains the "Old Monitor ROM", and you should read this section.

Using the Old Monitor ROM

Each time you turn on your Apple, you will see an asterisk (*) prompt character at the screen's lower left, followed by the flashing cursor. This indicates that you are in the Monitor program, which advanced programmers use when working in "machine language". To begin running in BASIC after turning on your Apple, you must always go through the following magic sequence:

1. Press the RESET key (at the upper right corner of the Apple's keyboard).
2. Hold down the CTRL key (at the middle left on the keyboard) and continue to hold it down while you type the letter B .
3. Press the RETURN key (at the middle right on the keyboard).

In key-symbol notation, the sequence to begin running in BASIC looks like this:

Recovering from Accidental RESETs

If your Apple contains the Autostart ROM, pressing the RESET key causes no problems: you are immediately returned to the BASIC you were just using.

With the Old Monitor ROM, however, accidentally pressing the RESET key will suddenly throw you into the Monitor program. You will see the asterisk (*) prompt character at the screen's lower left, followed by the flashing cursor. To return to BASIC without losing any stored program, you must do one of the following:

1. Without the Disk

If you are using Integer BASIC or <u>firmware</u> Applesoft (see Appendix D), you can return to your program after an accidental or intentional press of the RESET key by typing

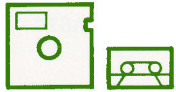
This will return you to the BASIC you were using when you pressed the RESET key, without losing your program.

If you are using cassette Applesoft, after pressing the RESET key you must type

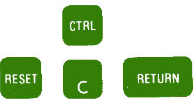
2. With the Disk

If you have booted DOS, no matter what version of Applesoft you are using, after pressing the RESET key you must type

This will return you to DOS and the BASIC you were using when you pressed the RESET key, without losing your program.



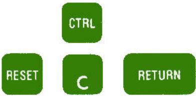
If you are using diskette <u>or</u> cassette Applesoft,



will attempt to reinstate Integer BASIC as your programming language. This may erase Applesoft and any program in memory, and the uninitialized Integer BASIC will not work correctly.

3. With the Applesoft II BASIC Programming Reference Manual

The Applesoft II BASIC Programming Reference Manual contains much more detailed information about Applesoft than this teaching manual contains. The Applesoft II BASIC Programming Reference Manual, however, was written assuming your system contains the Applesoft II Firmware Card (see Appendix D in the manual you are now reading), and no disk. Each place where the Applesoft II BASIC Programming Reference Manual says to use



diskette Applesoft users should use

and cassette Applesoft users should use

instead. Where the Applesoft II BASIC Programming Reference Manual says to use

diskette Applesoft users can do the same, but they will then be in Integer BASIC, and will have to re-boot the disk (PR#6) and then reload Applesoft from diskette (FP). Cassette Applesoft users will also find themselves in Integer BASIC, and will have to reload Applesoft from cassette tape.

INDEX TO THE APPLESOFT TUTORIAL

NOTE: The page numbers in parentheses refer to the Applesoft II BASIC Programming Reference Manual.

A

addition 23, 39-42
address 131-132 (40, 41, 43-45)
AND 57-59 (33, 36, 144)
Apple Disk II Disk Drive: see disk drive
Applesoft BASIC 2
  command summary 114-125
  error messages 143-146
  loading from cassette 136 (106-109)
  loading from diskette 134-136
  on cassette 2, 92, 93, 96, 131-133, 136-142, 148-149 (106, 108, 109)
  on diskette 2, 92, 93, 96, 131-133, 134-142, 148-149
  on firmware card 6, 117, 131-134, 137-142, 148-149 (44, 106, 107, 109)
Applesoft II BASIC Programming Reference Manual 26, 97, 101, 111, 114, 126, 141, 143, 149
Applesoft II Firmware Card 6, 117, 131-134, 137-142, 148-149 (106, 107, 109)
arguments 35
arithmetic 23-24, 59 (33, 36)
arithmetic operators 23-24, 59 (33, 36)
  precedence of 39-42
arrays 108-111 (14, 18, 32, 58)
  error messages 110-111
arrow keys 11, 27-29, 53, 114, 128 (54, 55, 110-114, 150)
arrow, upward pointing 24
assertions, true and false 55-58 (9)
Autostart ROM 2, 147-148

B

backspace key 27-29, 53, 114, 128
beep 10
  generating 78-81
  with LOAD and SAVE 12-13, 62, 123, 136
BELL 8, 10
blinking square: see cursor
booting DOS: see DOS
bouncing ball 74-79
branching
  FOR/NEXT 64-67, 116-117, 143, 144 (11-14, 20, 78, 79, 152)
  GOSUB/RETURN 87-91, 117, 143, 145, 146 (15, 16, 79, 80, 119, 153)
  GOTO 50, 59, 63-64, 117, 146 (76, 81, 153)
  IF/THEN 59-60, 119, 143, 146 (9-10, 76, 154)
bytes 131, 135, 136, 137-138

C

cable 3, 5
CALL -151 114, 133
cassette Applesoft: see Applesoft BASIC
CASSETTE IN jack 5
cassette recorder
  plugging in 5
  setting the volume 11-14
CASSETTE OUT jack 5
CATALOG 16, 60, 114
change program line: see editing
CLEAR 104, 114 (8, 52, 150)
clearing the screen 9, 12, 33, 45, 118, 129-130
colon 81 (10, 125)
COLOR= 30-34, 115 (5, 11, 24, 25, 85, 150)
color (23-27, 85, 89, 131-134)
  high-resolution charts 92, 117
  low-resolution charts 18, 115
  names and numbers 18, 92, 115, 117
  setting TV color 18-19
COLOR DEMOSOFT
  on diskette 16, 18, 30
  on cassette tape 12, 18, 30
columns
  tab fields: see tab
  with graphics 29, 62-63
comma 68-69 (6, 70)

commands 114-125 (2, 122-123)
concatenation
  strings 105, 145 (21, 71)
CONT 51, 115, 143
  (39, 40, 67, 151)
Control: see CTRL key
co-ordinates
  high-resolution 91, 95
  low-resolution 29, 64, 71, 85-86
CTRL B 137, 147, 149
CTRL C 17, 50-51, 115, 120, 148-149 (7, 10, 35, 39, 40, 107-109, 151)
CTRL key 10 (35, 144)
CTRL G 10
CTRL S 75, 120
CTRL X 54, 115, 129
  (55, 66, 69, 151)
controller card 3, 16, 135
cursor 5, 10, 11, 13, 16, 22, 136
cursor position 10, 11, 27-29, 52-53, 116, 124
  (50-52, 54, 55, 110-114, 131)

D

debug mode: see TRACE
decimal places 25 (18, 22)
deferred execution 44-45, 48, 143
  (2, 36, 134)
DEL 49, 65, 115, 129 (49, 151)
delay loop 82 (27, 41-43, 97)
delete 49, 54, 65, 115, 129
  (3, 38, 49)
DIM 108-109, 115-116, 145
  (14, 58, 152)
dimensions: see DIM
Disk II disk drive: see disk drive
disk drive 3, 4, 6, 15-16, 134-136
diskette Applesoft: see Applesoft BASIC
Disk Operating System: see DOS
division 24, 39-42
  (2, 18, 33, 36)
DOS (Disk Operating System)
  booting 15-16
  commands 16, 114, 120, 123, 135
  memory requirements 135, 137-138, 140-142
  recovering from accidental RESETs 148-149

E

EAR or EARPHONE jack 5
edit mode 52, 128
editing (54, 55, 110-114)
  arrow keys 27-29, 53, 114, 128
  changing program lines 48-49, (54, 110-114)
  CTRL X 54, 115, 129
  DEL 49, 65, 115, 129
  pure cursor moves (ESC with I, J, K, and M) 52-53, 116, 128-129
element
  arrays 108
    (14, 32, 58, 62-64)
END 88, 90, 116
  (16, 39, 118, 152)
equal sign
  as a replacement sign 36-39
    (12)
  in an assertion 55 (55)
erasing
  programs 44-46, 121 (3, 38)
  the screen 9, 12, 33, 45, 129-130 (52)
ERR or ERRERR 12
error messages 143-146
  (115-117, 167)
ESC key 8-9, 52-53, 128-130 (35)
execution 44-45, 48-49, 143
  (2, 36, 38-45)
exponentation 24, 39-42
  (4, 5, 18, 31-33)

F

firmware Applesoft 6, 117, 131-134, 137-142, 148-149
  (106, 107, 109)
FLASH 68, 116 (53, 152)
FOR/NEXT 64-67, 116-117, 143, 144
  (11-14, 20, 78, 79, 152)
format: see number format
function 35 (73, 102-104)

G

game controls 3, 4, 20, 34, 48-51, 121 (90, 134, 135)
GAME I/O socket 4
GOSUB/RETURN 87-91, 117, 143, 145, 146 (15, 16, 79, 80, 119, 153)

GOTO 50, 59, 62-64, 117, 146
(7, 76, 81, 153)
GR 30, 33, 62, 86, 117
(5, 11, 23-25, 84, 131-134, 153)
graphics
high-resolution 91-97, 137-142
(25-27, 87-100, 126, 131-134)
low-resolution 29-34, 60, 62-67, 74-79, 85-87, 89-91
(5, 10, 23-25, 83-87, 131-134)
greater than [>] 55, 59

H

HCOLOR 93-96, 117
(26, 27, 89, 134, 153)
hexadecimal 132
HGR 92, 117-118, 137 (25, 26, 84, 87, 89, 98, 99, 153)
HGR2 118, 138, 142
(25, 84, 88, 89, 99,153)
high-resolution graphics 91-97
(25-27, 87-100, 131-134)
memory map 140-142
memory range 140-142 (126)
Page 2 118, 138, 142
HIMEM: 140-142, 144 (41, 43, 44, 99, 100, 123, 127, 154)
HLIN 33, 118 (6, 25, 86, 154)
horizontal lines, plotting: see HLIN
HOME 45, 118, 130
(11, 48, 52, 154)
HPLOT 93-96, 118, 139 (26, 89, 98, 131-134, 154)
HTAB 70-71, 118 (27, 50, 51, 154)

I

I key (with ESC): see editing
IF/THEN 59-60, 119, 143, 146
(9-10, 76, 154)
ILLEGAL QUANTITY ERROR 32, 76, 144
immediate execution 44-45, 143
(2, 36)
IN USE light 6, 15
incrementing in loops: see looping
INPUT 75-77, 102, 119
(7, 9, 66, 67, 141, 154)
INT 83, 119 (19, 102, 155)
integer (2, 4)
INT function: see INT
rounding 25 (18, 31)
variables 36-38 (18, 31, 145)
Integer BASIC 2, 132, 134, 136, 148-149
interrupting execution: see CTRL C and RESET
INVERSE 68, 119 (53, 155)

J

J key (with ESC): see editing

K

K key (with ESC): see editing
keyboard 7-11 (130)
keyboard notation 9

L

LEFT$ 101-102, 119, 144
(20, 60, 124, 155)
Left-arrow key 11, 27-29, 53, 114, 128 (54, 55, 67, 110-114, 150)
LEN 100-101, 120 (19, 59, 155)
less than [<] 55, 59
lines
in a program 45-47, 81
(2, 3, 36, 118, 141)
in graphics mode 32-34, 118, 124 (86, 89, 92-97)
line number 45-47
(2, 3, 35, 49, 145)
LIST 44-46, 51, 120
(3, 4, 48, 155)
Little Brick Out 19-20
LOAD 12-14, 61, 120, 136
(38, 156)
loading
cassette Applesoft 136-137
cassette programs 11-14, 120, 136 (106-109)
diskette Applesoft 134-136
diskette programs 61, 120
LOMEM: 140-141, 144

looping 50, 59, 62-67, 116-117
(11-14, 20)
incrementing 66, 116-117
(13, 78)
low-resolution graphics: see
graphics

M

M key (with ESC): see editing
memory 131-132, 137-138
(2, 8, 40, 41)
HGR2 118, 138, 142 (88)
map 140-142
requirements 135, 136, 140-142
menu 17
MIC or MICROPHONE jack 5
MID$ 101-103, 121, 144
(20, 61, 156)
modes
debug: see TRACE
execution: see execution
modulator, RF 3-4
MON or MONITOR jack 5
Monitor program
entering 114, 133, 148
Monitor ROM 2, 147-149
monitor, TV 3-4
moving the cursor 11, 27-29, 52-
53, 114, 116, 128-129
(50-52, 54, 55, 110-114, 131)
multiple statements on a line 81-82
(10, 125)
multiplication 23, 39-42 (2, 33, 36)

N

negative numbers 39-42
nested loops 66-67, 144
NEW 44-46, 121 (3, 8, 38, 156)
NEXT: see FOR/NEXT
NORMAL 68, 121 (53, 156)
NOT 57 (33, 34, 36)
NOTRACE 88, 121 (40, 156)
null string 101
number format 25-26, 144 (4, 5,
18, 22, 31-33)

O

Old Monitor ROM 2, 147-149
one (in assertions) 55-58
OR 58-59 (33, 36)

P

paddle 34, 121
parentheses 41-42, 58, 106
pause: see delay loop
PDL 34-35, 48-51, 94-95, 121
(90, 157)
PEEK 80-81, 141, 144
(40, 131, 134-136, 157)
PLOT 30-33, 121
(5, 10, 24, 85, 157)
POKE 138, 144
power cord 3
POWER light 5
power switch 5
precedence of operators 39-42, 59
(36)
PRINT 22-23, 35, 44, 122
(2, 6, 7, 70, 71, 157)
comma 68-70, 122
semi-colon 68-70, 122
program, definition of 47
prompt character 13, 17, 22, 133,
136 (35, 84, 106, 108)
PR# 16 (72, 158)
pure cursor moves 52-53, 116, 128

Q

question mark
INPUT 75-77 (7, 66, 67)
quotation marks
INPUT 76 (66)
PRINT 22, 27, 36
strings 100, 105 (19, 34)

R

random number function: see RND
REENTER 76
REM 63, 122 (8, 10, 50, 118, 158)
Replacing lines: see editing
REPT key 10, 52, 122, 128 (55,
111-114, 158)

reserved words 37, 126-127 (7, 8, 38, 64, 87, 148)
RESET key 6, 2Ø, 51, 12Ø, 147
recovering from 148-149
stopping a program 51 (39)
RETURN 87, 122 (15, 16, 79, 8Ø, 158)
RETURN key 1Ø-11, 17, 26 (2, 3, 7, 35)
retype key 28-29, 53, 114, 128
RF modulator 3
RIGHT$ 1Ø1, 122, 144 (2Ø, 61, 158)
right-arrow key 11, 27-29, 53, 114, 128 (54, 55, 11Ø-114, 15Ø)
RND 82-85, 123 (18, 27, 1Ø2, 141, 159)
ROM-Applesoft: see Applesoft II Firmware Card
rounding 25 (4, 5, 18, 19, 31-33)
with graphics 64
rows 29
RUN 13, 14, 16, 17, 2Ø, 44-46, 51, 65, 123 (2, 8, 38, 39, 159)

S

SAVE 6Ø-62, 123 (38, 159)
saving programs
on diskette 6Ø-61, 123
on cassette tape 61-62, 123
scientific notation 25-26 (4, 5)
screen
clearing: see clearing the screen
sketching screen program 63, 94
semi-colon 69, 76 (3Ø, 33)
INPUT 76 (66-67)
PRINT 69 (6, 7Ø, 71)
setting the tape recorder 11-14
setting the TV color 18-19
SHIFT key 7
slots Ø through 7 16, 134 (71, 72)
sounds, generating 78-81
spacing 69-71
speaker 79-81 (134, 135)
square bracket 11, 16, 22
statements, multiple 81-82 (1Ø, 125)
STEP 66, 116-117 (13, 78, 152)
stopping the computer 17
listings 75, 115, 12Ø
programs 5Ø-51, 115 (7, 1Ø, 16, 38, 39)
STR$ 1Ø7, 124 (21, 22, 59, 16Ø)
strings 1ØØ-1Ø8 (18-23, 34)
concatenation 1Ø5 (21, 52, 71)
INPUT 1Ø2 (66, 67, 154, 155)
LEFT$ 1Ø1, 119 (2Ø, 6Ø, 155)
LEN 1ØØ-1Ø1, 12Ø (19, 2Ø, 59, 155)
MID$ 1Ø1-1Ø3, 121 (2Ø, 21, 61, 156)
null strings 1Ø1 (19, 6Ø, 61, 67, 69, 76, 77)
RIGHT $ 1Ø1-1Ø2, 122
STR$ 1Ø7, 124 (21, 22, 59, 16Ø)
VAL 1Ø6-1Ø7, 124 (21, 23, 59, 161)
subroutine 85-91 (16, 22, 79, 8Ø)
subscript 1Ø8, 144 (14, 15, 34, 58)
subtraction 23, 39-42
SYNTAX ERROR 11, 12, 22, 32, 37, 145
System Master diskette 15, 16, 134-135

T

tab
HTAB 7Ø-71, 118 (5Ø, 51)
TAB 7Ø-71, 124 (51, 16Ø)
VTAB 7Ø-71, 125 (5Ø)
TEXT 3Ø, 62, 88 (6, 11, 84, 16Ø)
THEN: see IF/THEN
TO: see HPLOT and GOTO
TRACE 87-88, 91, 124 (4Ø, 82, 161)
TV monitor 3-4

U

V

VAL 1Ø6-1Ø7, 124 (21, 23, 59, 161)

variables 37-38, 89 (7, 8, 31-35)
  array 108, 110 (14, 58)
  FOR/NEXT loops 64-67, 116, 117, 143, 144 (12, 13, 78, 79)
  INPUT 75-77, 102-103, 119 (7, 9, 66, 67, 71)
  integer 36-38 (18, 19, 31)
  names 36-38, 100, 120 (7, 8, 14, 18 31-35)
  string 100-104, 120 (18)
vertical lines, plotting: see VLIN
VIDEO OUT jack 4
VLIN 33-34, 124 (6, 25, 86, 161)
VTAB 70-71, 125 (27, 50, 161)

X

X coordinate 29, 64, 71, 85-87, 91, 95

Y

Y coordinate 29, 64, 71, 85-87, 91, 95

Z

zero 8, 22, 143
  in assertions 55-58

apple computer inc.
10260 Bandley Drive
Cupertino, California 95014