PRENTICE HALL

Apple IIe Programming

A Step-by-Step Guide

Phil Robinson

BOOK ONE
Fully Illustrated in Color

PRENTICE HALL

PROGRAMMING SERIES

APPLE IIe PROGRAMMING

A Step-by-Step Guide

Never has there been a more urgent need for a series of well-produced, straightforward, practical guides to learning to use a computer. It is in response to this demand that The Step-by-Step Programming Series has been created. It is a completely new concept in the field of teach-yourself computing. And it is the first comprehensive library of highly illustrated, machine-specific, step-by-step programming manuals.

BOOKS ABOUT THE APPLE IIe

This is Book One in a series of unique step-by-step guides to programming the Apple IIe. Together with its companion volumes, it builds into a self-contained teaching course that begins with the basic principles of programming, and progresses – via more sophisticated techniques and routines – to an advanced level.

ALSO AVAILABLE IN THIS SERIES

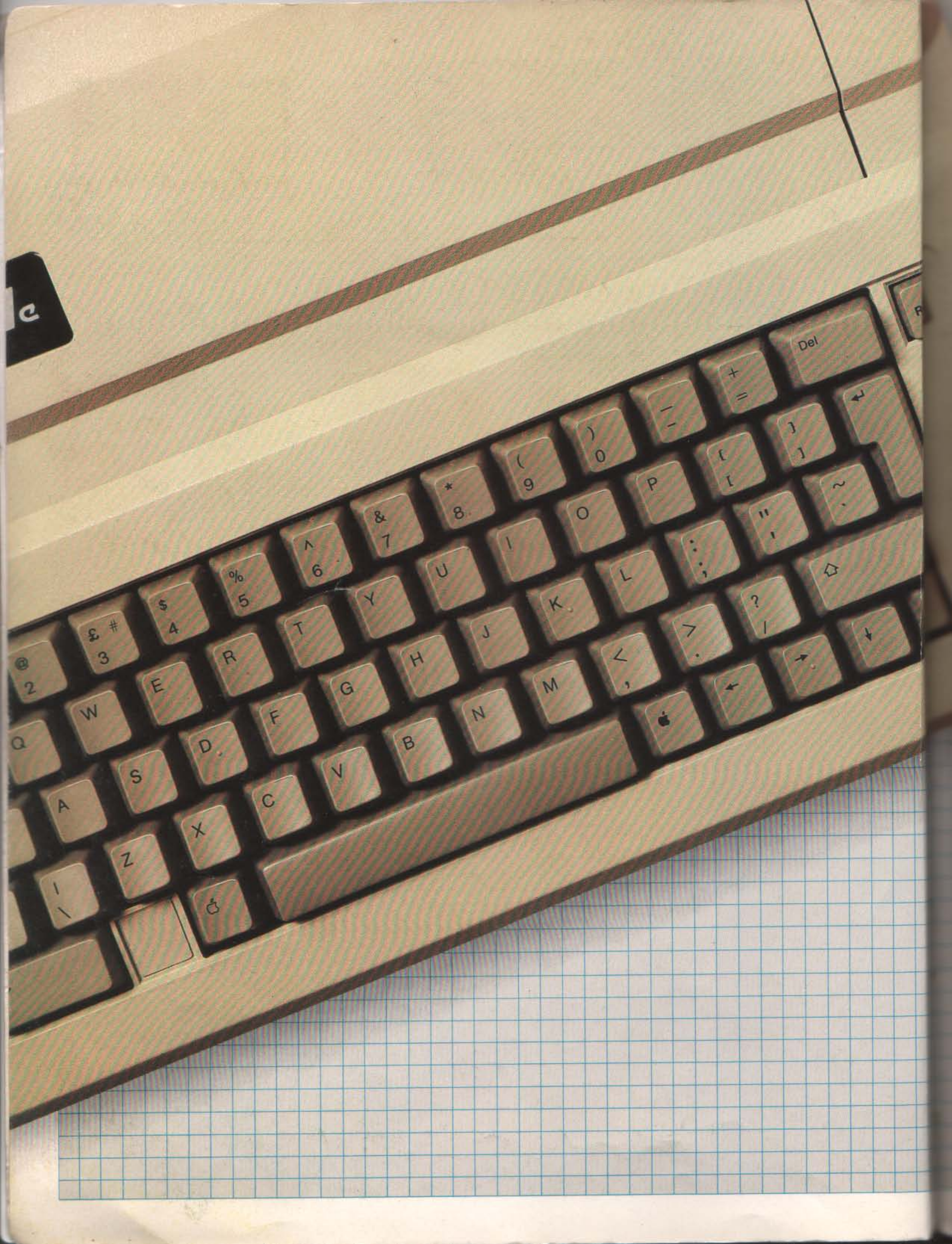
Commodore 64 Programming

IBM PCjr Programming

PHIL ROBINSON

Phil Robinson graduated from Brunel University in 1975 with a degree in Electrical Engineering. During the next four years he worked as a computer programmer and analyst on mainframe and mini computers. His work during this period involved scientific, business and games programming in BASIC, FORTRAN, COBOL and FOCAL. He transferred his expertise to microcomputers in 1979 and became a founder member of Digitus, a micro systems company based in London. Since 1979 he has written programs for the Apple, the Cromemco, the North Star, the Sirius and the IBM PC. In 1981 he became a freelance computer consultant and writer.

BOOK ONE



PRENTICE HALL

PROGRAMMING SERIES

APPLE IIe PROGRAMMING

A Step-by-Step Guide

PHIL ROBINSON

A SPECTRUM BOOK

PRENTICE-HALL INC., Englewood Cliffs, New Jersey 07632

BOOK ONE

CONTENTS

The Step-by-Step Programming Series was conceived, edited and designed by Dorling Kindersley Limited, 9 Henrietta Street, Covent Garden, London WC2E 8PS.

Editor Gillian Aspery
Designer Hugh Schermuly
Photography Vincent Oliver
Series Editor David Burnie
Series Art Editor Peter Luff
Managing Editor Alan Buckingham

First published in Great Britain in 1984 by Dorling Kindersley Limited, 9 Henrietta Street, Covent Garden, London WC2E 8PS.

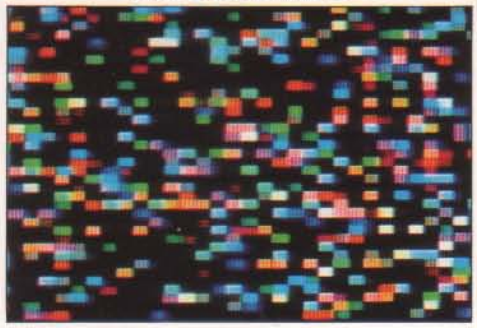
A Spectrum Book. Printed in Italy. This book is available at a special discount when ordered in bulk quantities. Contact Prentice-Hall, Inc., General Publishing Division, Special Sales, Englewood Cliffs, N.J. 07632.

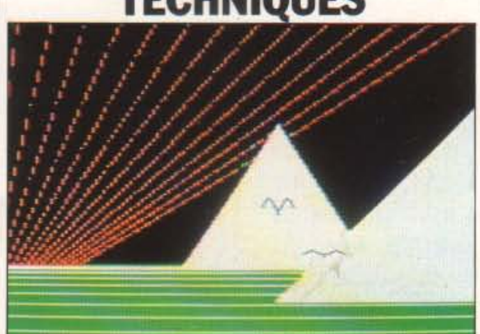
Library of Congress Cataloging in Publication Data

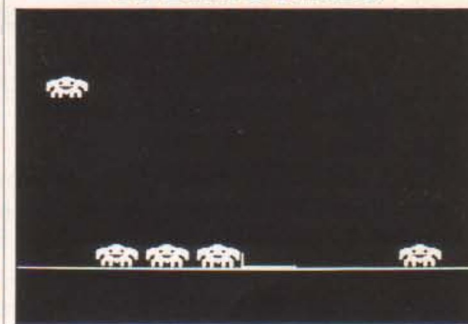
Robinson, Phil.

Apple IIe programming: a step-by-step guide: book one.
"A Spectrum Book."
Includes index.
1. Computers. 2. Apple IIe. I. Title.
ISBN 0-13-038456-9

Typesetting by Cambrian Typesetters, Frimley, Camberley, Surrey, England
Reproduction by Reprocolor Llovet S.A., Barcelona, Spain
Printed and bound in Italy by A. Mondadori, Verona







THE APPLE IIe

The Apple IIe is a member of the popular family of Apple computers. It is a versatile and friendly computer which is equally valuable in the home, the office or the classroom.

Although a wide range of software is available for the Apple IIe it is far cheaper and a lot more fun to write your own programs in Applesoft BASIC. This is Apple's version, or dialect, of BASIC – the most popular programming language for personal computers. Once mastered, Applesoft BASIC puts the full power of the Apple IIe at your fingertips.

Although the Apple IIe benefits from modern chip technology, it remains compatible with earlier models. The programs in this book will therefore work on older versions of the Apple (the original II and the II+) providing they have Applesoft BASIC in ROM. However, the keyboard and screen display differ slightly on all three models and certain procedures require different action to that explained in this book; these procedures include starting up the system and editing programs. If you own an Apple II or II+ you will therefore be advised to consult the Apple owner's manual when you reach these stages of the book.

Connectors and peripherals

From the outside, the Apple appears to be little more than a sturdy plastic case with a typewriter-style keyboard. But a closer investigation will reveal otherwise.

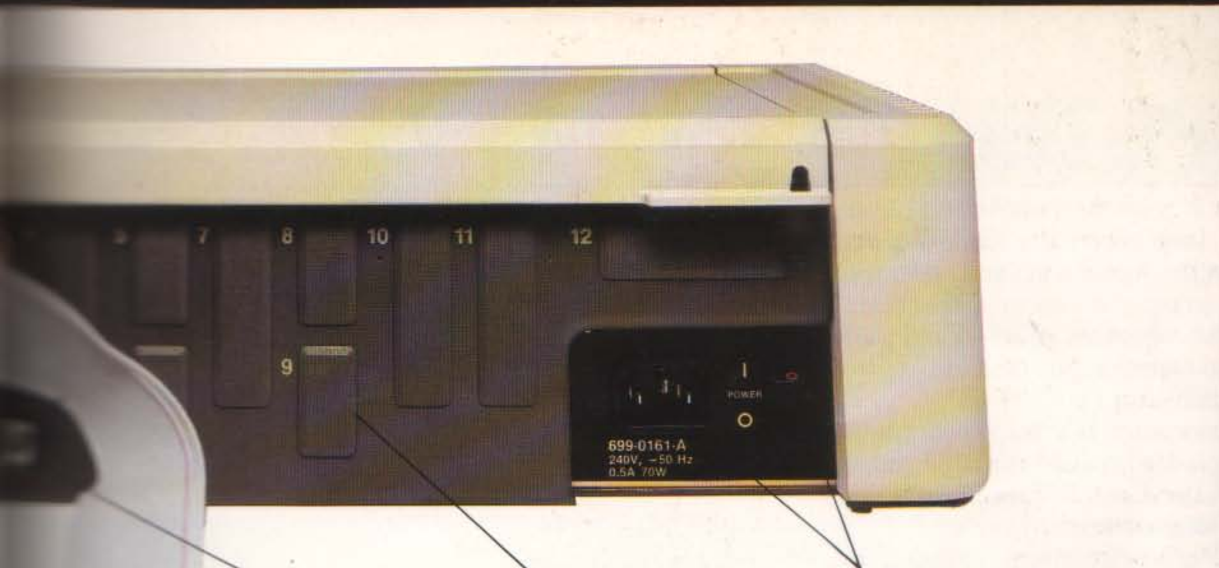
Start by turning the computer round to look at the rear panel. At the bottom left is the video output. This allows you to connect the Apple to a video monitor which will display the results of the instructions that you type into the computer.

On the right of the video jack is the cassette output and next to that, the cassette input. These allow you to save and then re-load programs into the computer using a cassette recorder. The alternative method of saving programs is on disk. A disk drive is more expensive than a cassette recorder but it is far quicker, more convenient and more reliable.

The socket next to the cassette input handles games controllers and joysticks. And the numbered rectangular openings above and to the right enable you to install expansion cards into the Apple. As your programming skills develop you can add expansion cards to make your Apple talk, recognize speech, control a light pen and play music. It is this flexibility for expansion that makes the Apple stand out from other personal computers.

Video out connects the Apple to a monitor to display text and graphics.

Cassette connecto allow you to save a load programs from cassette recorder.



Hand controller This socket connects a joy-stick to the Apple for playing games.

Connector slots (9, 19 and 25 pin.) These allow you to install sockets for optional expansion cards.

Power plug socket and on-off switch This socket holds the 3-pin plug that connects the Apple to the domestic power supply.

INSIDE THE COMPUTER

When using a computer it helps to understand a little about how it works. A look under the lid is a good place to start. Check that the Apple is off and, with the keyboard towards you, grasp the two tabs that stick out from the back of the computer cover. Then pull firmly upwards until you hear a pop and the cover comes away from the main case.

At the heart of the computer is a microprocessor. The one used in the Apple IIe is called the 6502 and it forms a crucial part of the Central Processing Unit (CPU). All computers, large or small, have a CPU. It performs all the computer's calculations, takes decisions and displays the results on the screen. But in spite of the complexity of this chip, it can only follow the instructions that it is given. Some of these instructions are pre-programmed into the Apple but you will have to enter others using the keyboard. Both types of instructions are stored in the other main component of the Apple – its memory.

There are two types of memory, RAM and ROM. Everything you type at the keyboard is stored in RAM. It can be changed easily but is lost when you turn the computer off .

ROM stands for Read Only Memory and as the name suggests it is permanent and cannot be altered. This is where the Apple stores the permanent programs that tell the CPU how to behave. The information that is stored in the ROM chips is retained even when the machine is off.

BASIC and machine code

The CPU only understands a bewildering series of electrical pulses called "binary". This system is based on only two numbers – 0 and 1, where 0 is represented by "off" (no pulse) and 1 is represented by "on" (one pulse). Each 0 or 1 conveys a unit of information and is called a "bit". The computer stores eight bits (known as one "byte") of information together. Each byte represents a different combination of 0s and 1s and stores one character of recognizable information, one letter of the alphabet or a number from 0 to 9 for example. A computer's memory is measured in kilobytes (kB or just k). One kilobyte, in computer jargon, is equivalent to 1024 bytes. The Apple has 64k of RAM and 16k of ROM.

It is very difficult to program the Apple in its own language, binary – also known as machine code. So Applesoft BASIC acts as an interpreter, converting the instructions that you enter in BASIC into machine code instructions that the CPU can follow.

Numeric keypad connector This is used to connect a special numeric keypad to the Apple. A keypad looks like a calculator and is particularly useful if you want to enter a large volume of numbers.

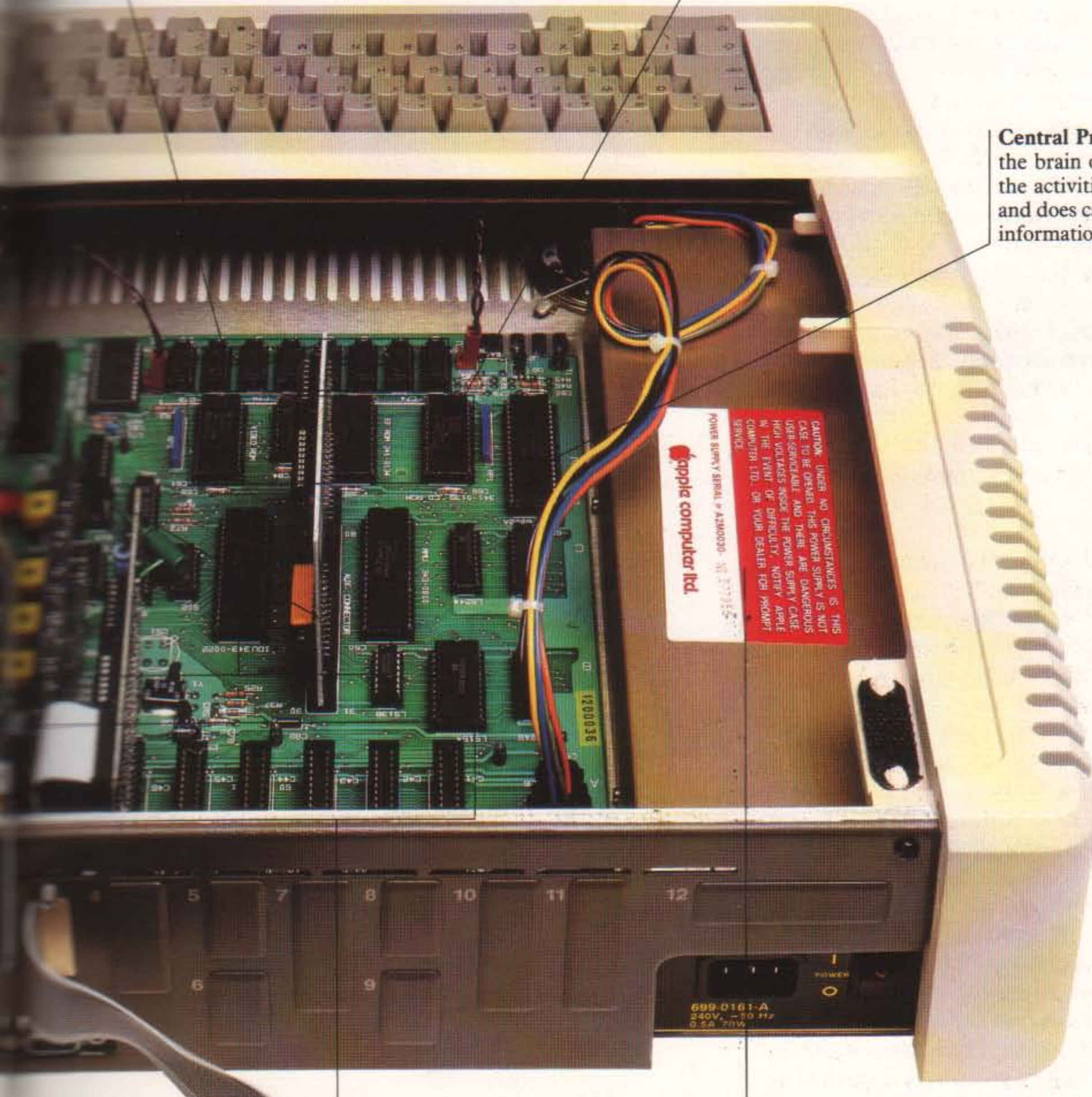
Auxiliary expansion slot This has an important function before the Apple leaves the factory. It is used to test the chips on the main circuit board. It also holds a plug-in card which can convert the Apple's display to 80 columns and add an extra 64k of RAM.

Logic chips These allow the CPU, ROM and RAM to "interface" with hardware like the keyboard and the video display.

Memory management unit and input/output unit (MMU and IOU) These two chips were specially designed for the Apple IIe. They do the work of about 50 normal logic chips making the Apple IIe cheaper to manufacture and easier to service.

Random Access Memory (RAM) These eight RAM chips provide 64k of memory and store information as it is typed into the computer. Work stored here must be saved onto disk or cassette before the computer is turned off, otherwise it is lost. If required, the auxiliary expansion slot can take a card with a further 64k of RAM.

Read Only Memory (ROM) These chips store the instructions that convert BASIC programs into a form that the CPU can understand and act upon. They also contain programs to test the Apple every time it is switched on.



Central Processing Unit (CPU) This is the brain of the computer. It organizes the activities of other parts of the Apple and does calculations using programs and information stored in ROM and RAM.

Power supply This converts the voltage from your domestic power supply to the lower level that the Apple requires.

Expansion slots These hold optional expansion cards which enhance the power of the computer. Each extra card performs a special function such as controlling a disk drive or a printer.

THE APPLE IIe KEYBOARD

The Apple has a high-quality keyboard that will suit a one-finger programmer or a fast touch typist. On first sight it looks like a typewriter but closer inspection reveals a few interesting additions.

Once the Apple is connected to a video display and switched on, try pressing a "character key" (A–Z, 0–9 and punctuation marks). The character will appear on the screen and will simultaneously be stored in the computer's RAM. Later you will use these keys to "enter" information and "commands".

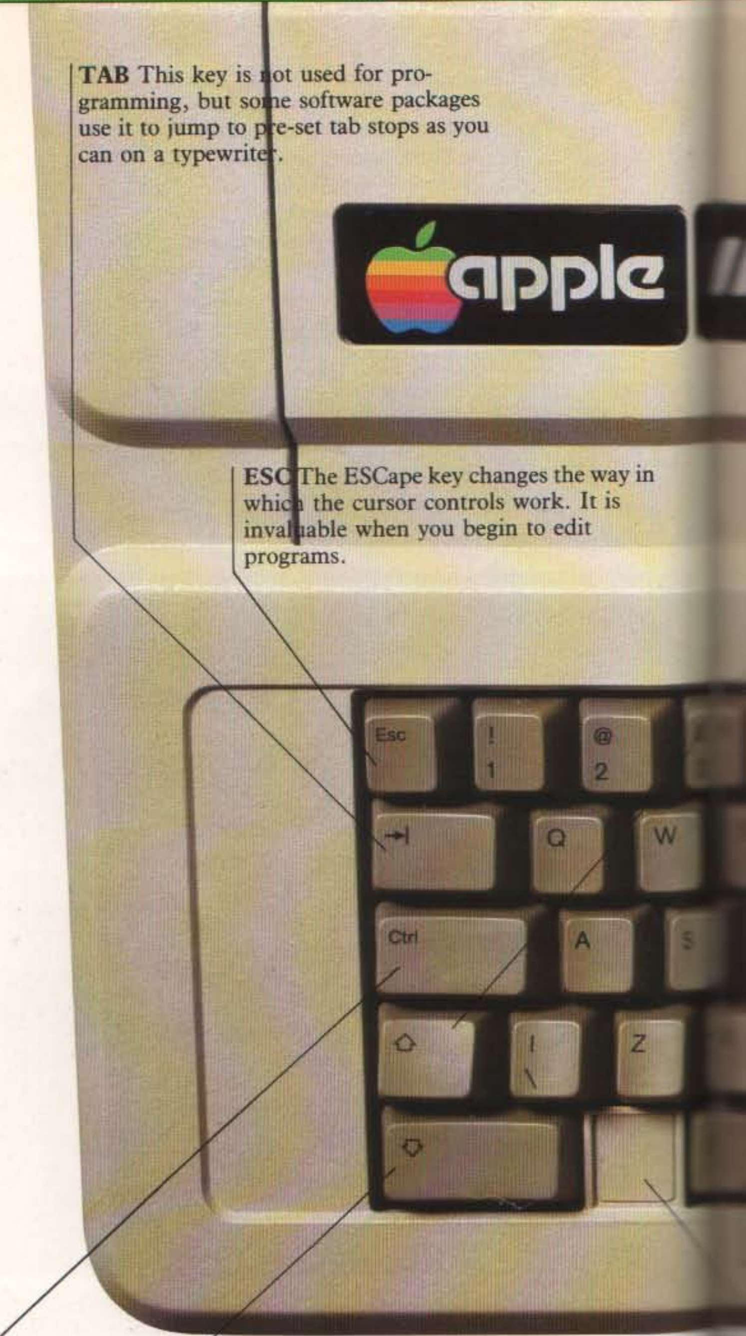
Like a typewriter the Apple can display upper- and lower-case letters. To display upper-case letters press the SHIFT key at the same time as a "letter key" (A–Z). Applesoft BASIC only accepts commands entered in upper case, so you may find it more convenient to press the CAPS LOCK key. This will click down and all subsequent characters that you type will be in upper case. You will also notice that some keys show two symbols, one above the other. These keys display the lower symbol if the key is pressed on its own, and the top symbol if the key is pressed while SHIFT is held down.

RETURN is another vital key. After you have typed an instruction, pressing the RETURN key will send it to the computer. Until that point you can make any amendments you like or even cancel the instruction completely, but after RETURN has been pressed it's more difficult (and sometimes impossible) to reverse a command.

The control key (CTRL) sends instructions direct to the CPU. But the character it sends, known as a "control character", is not displayed on the screen.

The RESET key does exactly what its name suggests – it resets the computer as if it had just been switched on.

The video display always shows a small flashing block at the point where the next character you type will appear, this is called the cursor. The cursor moves as you type but you can also move it with the cursor keys. These are the four keys marked with arrows on the bottom right of the keyboard. The arrows indicate the direction in which the cursor will move if the key is pressed, although when the normal flashing cursor is displayed only the right and left cursor keys will operate. You can alter the way the cursor keys work by pressing the escape (ESC) key. Now you can move the cursor in all four directions. By combining these two ways of moving the cursor you can change sections of a program in the computer's memory without having to re-type the entire program.



TAB This key is not used for programming, but some software packages use it to jump to pre-set tab stops as you can on a typewriter.

ESC The ESCape key changes the way in which the cursor controls work. It is invaluable when you begin to edit programs.

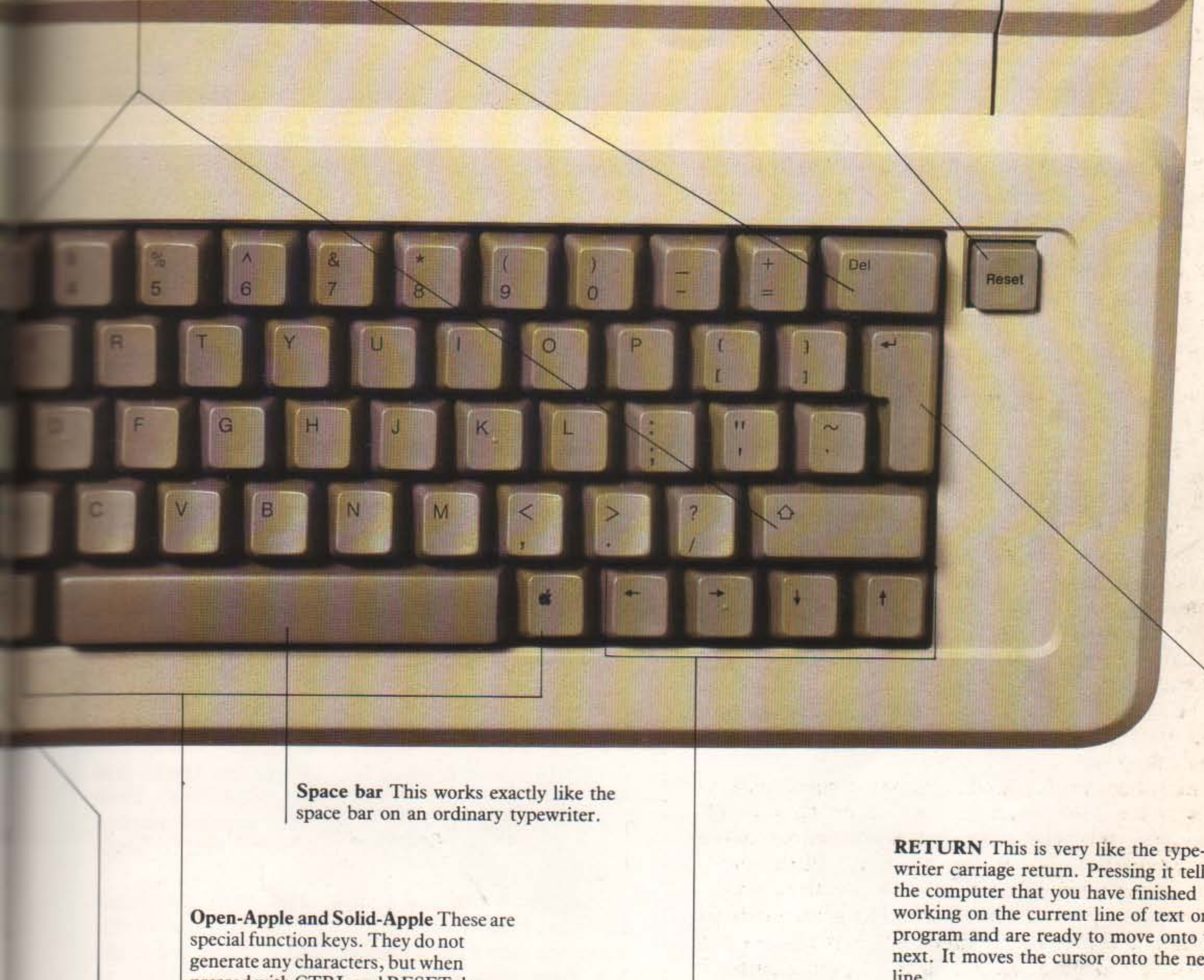
CAPS LOCK When this key is switched on all letters appear in upper case – CAPitals – until CAPS LOCK is pressed a second time.

CTRL When you press ConTRoL, together with certain letter keys, a "control character" is sent directly to the computer and gives it a command. For example, pressing CTRL and C together will stop a program running.

SHIFT Holding the SHIFT key down while pressing another key produces either an upper-case letter or, if a key has two symbols, the upper of the two. For convenience there are two SHIFT keys.

RESET This tells the computer to stop what it is doing. It is always pressed at the same time as CTRL. It RESETs the Apple ready for new instructions but saves any program that is in RAM. However, if Open-Apple is pressed with CTRL and RESET, the program in RAM will be erased. Pressing the Solid-Apple with CTRL and RESET will run a self-test program to check the Apple's main circuit board. The message "Kernel OK" will appear if the Apple is working correctly.

DEL this is not normally used for programming but some software packages use it to delete a character.



Space bar This works exactly like the space bar on an ordinary typewriter.

RETURN This is very like the typewriter carriage return. Pressing it tells the computer that you have finished working on the current line of text or program and are ready to move onto the next. It moves the cursor onto the next line.

Open-Apple and Solid-Apple These are special function keys. They do not generate any characters, but when pressed with CTRL and RESET they cause the computer to perform a special activity. For example, they can re-boot the computer while the power is on.

Cursor keys These four keys move the cursor around the screen, but their precise function varies if they are used with ESC.

Power indicator This indicates that the Apple is switched on.

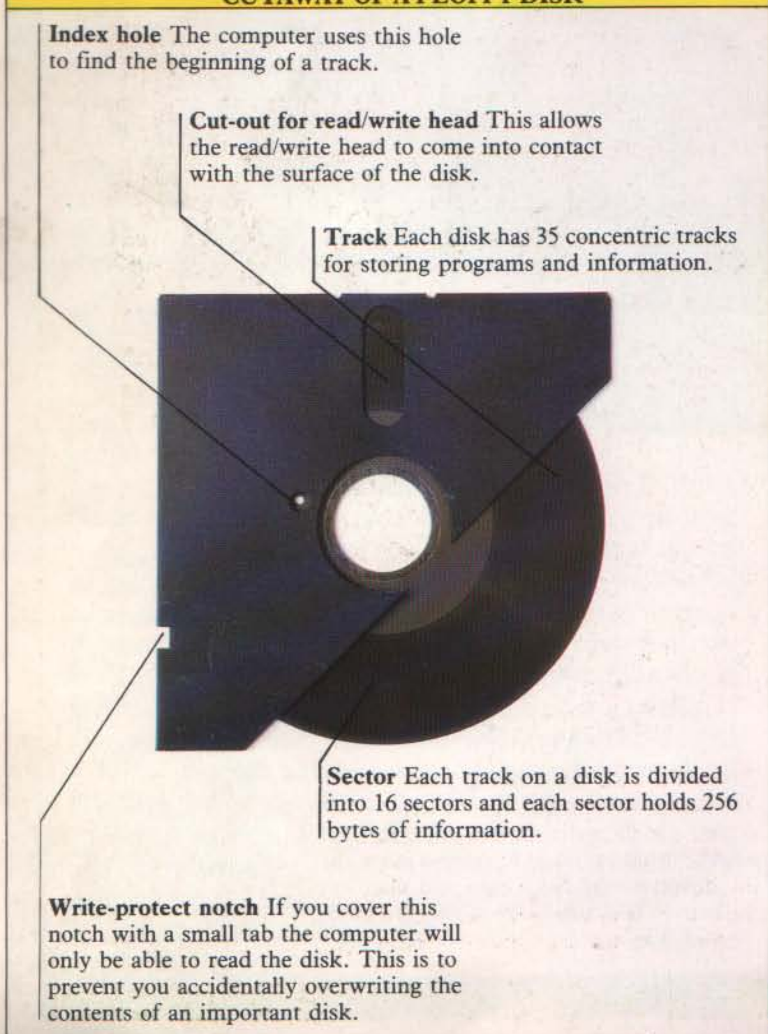
THE DISK DRIVE

The Apple IIe uses a disk drive to make permanent copies of programs and information held in RAM. Once you have saved information on a floppy disk the Apple may be switched off, and although the information will be lost from RAM you can recall the same information back to the computer from the floppy disk, and resume work on it at another time.

Floppy disks have certain things in common with both albums and cassette tapes. They use the same magnetic material as a cassette for recording information. But they rotate on a central spindle, and record information in concentric tracks, like an album.

When you buy a floppy disk it is protected in a paper sleeve and inside this by a plastic case. If you remove the paper sleeve, as if to use the disk, you will notice that there is an oblong slot in the plastic cover. This allows the "read/write" head to come into contact with the magnetic surface of the disk when it is placed in the disk drive. Floppy disks are delicate things and should always be handled carefully. Never touch the disk surface through the slot and always keep disks away from heat, dust and magnets.

CUTAWAY OF A FLOPPY DISK

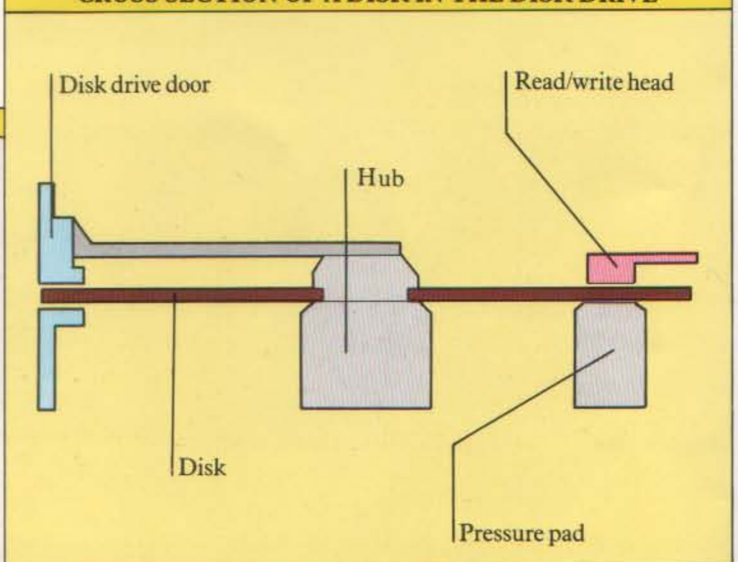


Using a disk drive

A floppy disk is placed in the disk drive through the door on the front of the drive. Normally the disk does not rotate but when the computer wants to read or write something on the disk it starts the motor running; you will hear this happen. An "in-use" light also glows red on the front of the drive when the computer is using the disk. Never open the drive while this light is on, you risk "crashing" the disk and losing everything you've stored on it.

Inside the disk drive is a circuit board and two motors: one spins the disk on a central spindle, the other is an unusual type of motor called a "stepping motor". Instead of rotating smoothly it goes round in a series of jerky steps. The read/write head is held on the arm driven by this motor. The steps occur at the same place on each rotation of the disk so that the read/write head can be positioned exactly over the required track.

CROSS-SECTION OF A DISK IN THE DISK DRIVE



An Apple's disk drive has 35 separate tracks and each track is further divided into 16 "sectors". Each sector will hold 256 bytes; this is the smallest amount of information that can be transferred to and from the Apple's RAM.

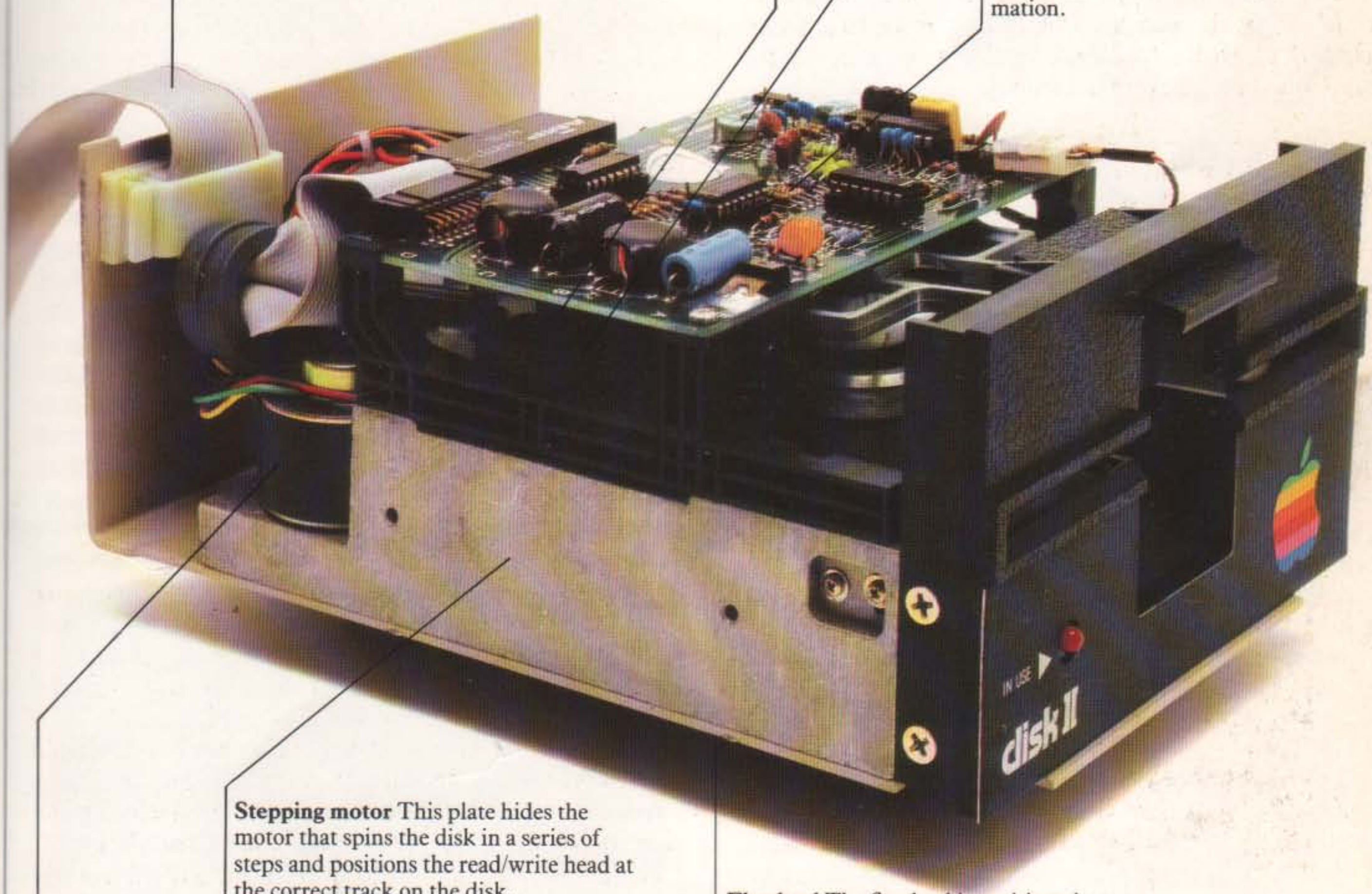
Before you can use a disk drive with the Apple you must plug a disk interface card into one of the expansion slots. The interface card is connected to the disk drive by a flat cable and allows the Apple to transfer information to and from disk, and control the two motors. When you turn the Apple on, the drive will automatically start to rotate. This is so that it can read into memory the "Operating System" (DOS 3.3 or ProDOS) which gives the computer the instructions it needs to use the disk drive. You should therefore always have a disk in the drive when you switch on.

Read/write head The read/write head is hidden by the drive control electronics, but it is similar to the record/play head in a cassette recorder. It is mounted on an arm connected to the stepping motor so that it can be positioned over the required track.

Pressure pad Like the read/write head, the pressure pad is hidden but it is positioned on the opposite side of the disk to the read/write head, and supports the floppy disk as the head comes into contact with the disk.

Interface cable As well as transferring information back and forth to RAM memory, this cable allows the Apple to control the two motors in the disk drive.

Drive control electronics These control the rotation and stepping motors and convert the electrical pulses coming from the read/write head into bytes of information.



Stepping motor This plate hides the motor that spins the disk in a series of steps and positions the read/write head at the correct track on the disk.

Flywheel The flywheel is positioned behind this plate. It smooths out any variation in the speed of rotation, much like the flywheel of a car. It has strobe markings for speed alignment and when the motor is running at the correct speed these appear stationary under artificial light.

Drive motor This rotates the floppy disk inside its sleeve.

STARTING OFF

With the help of the manual supplied when you purchased your Apple, start by connecting the computer to a TV set (or a monitor) and a disk drive.

Before you can start programming, the next step is to "load" or "boot" the computer. The instructions given below describe this start-up routine for the Apple IIe; owners of the II and II+ are recommended to consult the owner's manual, as the routine for earlier versions of the Apple is slightly different.

Booting the Apple IIe

Before you switch on the Apple IIe, the first step is to place a "Master" disk in the disk drive. This is the disk that was supplied when you bought the computer. Recent purchasers will have ProDOS User's Disk. But if you've had your Apple for a while, you will have DOS 3.3 System Master. Both Master disks fulfil roughly the same purpose, but ProDOS is the most recent development.

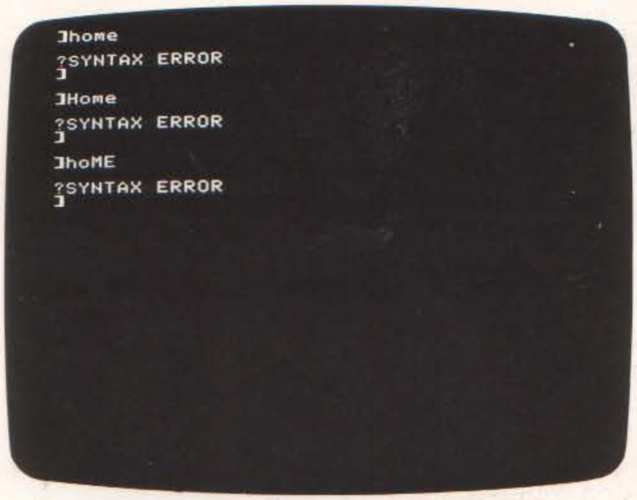
Take either disk now, place it in the disk drive and close the door. Turn on your TV or monitor and then your Apple. The disk will spin for a few seconds; wait until it stops and the light goes out on the disk drive. If you're using DOS 3.3 you're now ready to go, but if you're using ProDOS you must first type the letter B (it must be a capital B so first make sure that the CAPS LOCK is down). Now press RETURN, and you too are ready to go.

If you haven't already given in to the temptation to tap a few keys, try it now – you can't do any damage. In most cases the character you have pressed will appear on the screen.

But having successfully got the computer to display something on the screen, you will want to know how to remove it. The simplest method is to hold down the CTRL key and press RESET. This will clear the screen and reset the computer although none of the commands you may have given it are erased from its memory. If you really do want to erase a program from the memory you must press the Open-Apple key as well as CTRL and RESET. The best way of clearing the screen however is to type HOME, and then press RETURN.

It is important to remember that the computer will only obey instructions that are in upper-case letters and spelt correctly. If you type HOME and press RETURN, the screen will clear. But if you type HOme and RETURN, you will just get an "error message". This is because the computer treats capital and lower-case letters as completely different symbols. The screen shot entitled INCORRECT COMMAND ERROR MESSAGES shows how the computer responds to successive failures to type HOME correctly:

INCORRECT COMMAND ERROR MESSAGES



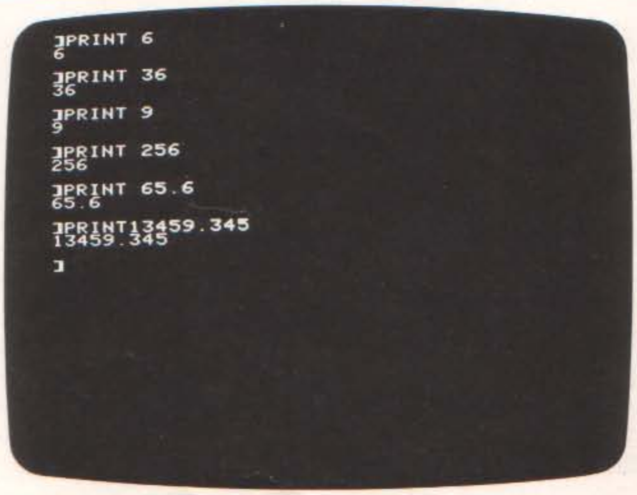
If during the following pages your computer refuses to obey your instructions, look carefully at the commands you've given it. Are they spelt correctly and in upper-case letters?

Now clear the screen and type in this line:

```
PRINT 6
```

If you press RETURN after this, the number 6 will appear on the next line of the screen; the computer has responded to your "command". PRINT has nothing to do with ink and paper – it just tells the computer to display something on the video screen. Try using this command in the same way with other numbers. It doesn't matter whether or not you leave a space for neatness between the command PRINT and the number. The computer can read characters that run together:

PRINT WITH NUMBERS



After you have tried a few different numbers, clear the screen with HOME and type this in:

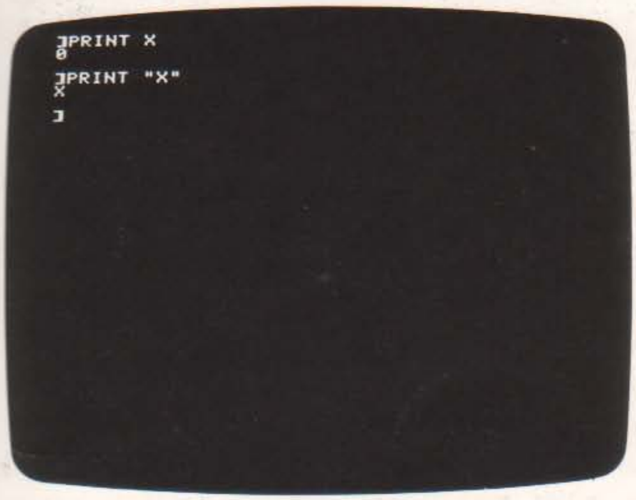
PRINT X

The computer responds by displaying the number 0. Surely it should have PRINTed the letter X? Now, using the SHIFT key to type quotation marks around the X, type:

PRINT "X"

When you press RETURN, the computer makes the correct response – it PRINTs X on the next line:

PRINTING A VARIABLE



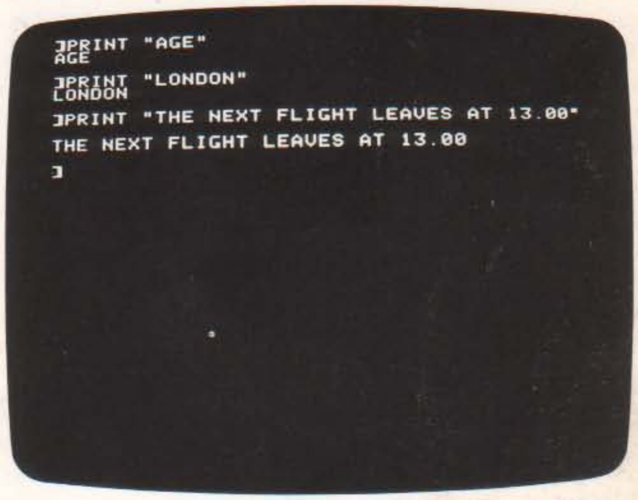
You have just discovered that, to the Apple, X and "X" mean two different things. The Apple treats any letter on its own as a variable. A variable is a label which identifies a slot in the computer's memory that can hold numbers or letters. When the computer is first switched on all of these variables have the value zero. To change the value of a variable try this. (Remember to press RETURN after each line):

USING **LET** AND **PRINT**

This time the number 14 is printed. LET is a command for changing the value of a variable slot in memory. From now on, every time you ask the Apple to PRINT X, it will display 14 – unless you change its value again using LET. As the slot X always holds a number, it is called a "numeric variable".

So X is a numeric variable, but "X" is not; furthermore, even if you substituted a number for "X", it would not become a numeric variable unless you removed the quotation marks. The computer displays everything inside quotation marks exactly as you type it. Try it, using letters, numbers, mathematical symbols and punctuation marks:

PRINTING STRINGS



Introducing string variables

In much the same way as a number can be stored inside the computer and labeled by a numeric variable, a string of characters is stored and labeled by a "string variable". String variables are always indicated by a variable name followed by a dollar sign. In the line:

LET A$="LONDON"

A$ is the string variable and LONDON is the string it labels. Once you've typed this in, type HOME to clear the screen and then, to recall the string variable A$, type:

PRINT A$

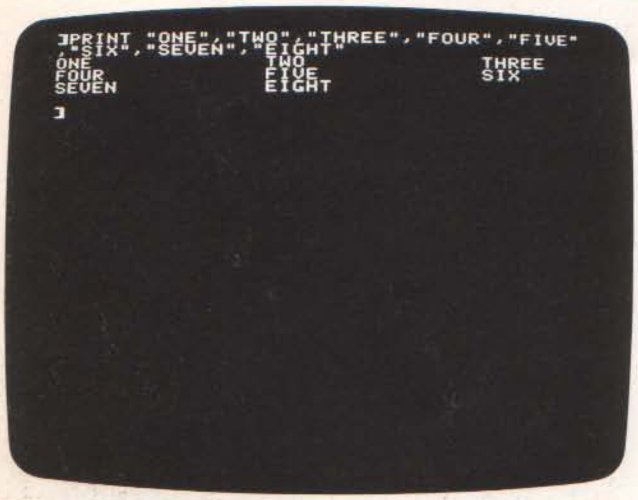
After you press RETURN, the computer will PRINT LONDON. As with numeric variables, the command LET allows you to put a string into the computer's memory. Again, you can use any letter to label a string variable, and as the computer will only remember the last version of any one string variable you can change the string held in say A$, as often as you like. Strings can be up to 255 characters long so you can PRINT several words, numbers, punctuation marks and symbols together.

MOVING AROUND THE SCREEN

The Apple's screen is divided into three invisible "fields" or columns. The first two are 16 characters wide and the last is eight wide. To see the fields type:

PRINT "ONE", "TWO", "THREE", "FOUR", "FIVE", "SIX", "SEVEN", "EIGHT"

FIELDS DISPLAY



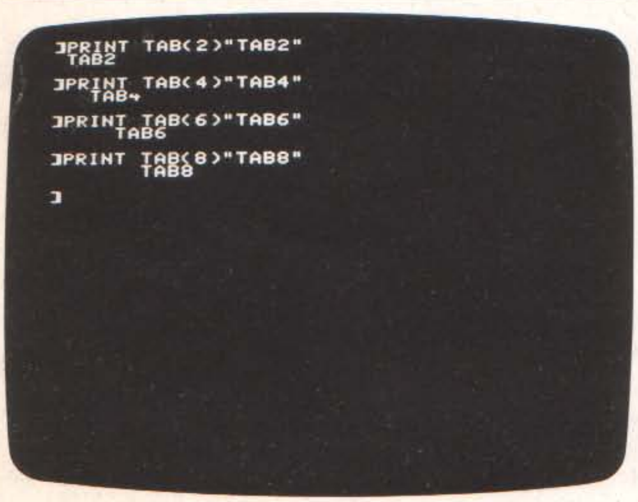
The first three strings are printed in the fields across the screen, then the computer returns to the first field on the next line to print the fourth string, and so on.

This way of PRINTing is very useful for positioning numbers, strings or variables in neat columns. But the command TAB allows you to print at any position on the screen. For instance:

PRINT TAB(2); "TAB2"

displays TAB2 two spaces in from the left. Here are some examples of the TAB command being used:

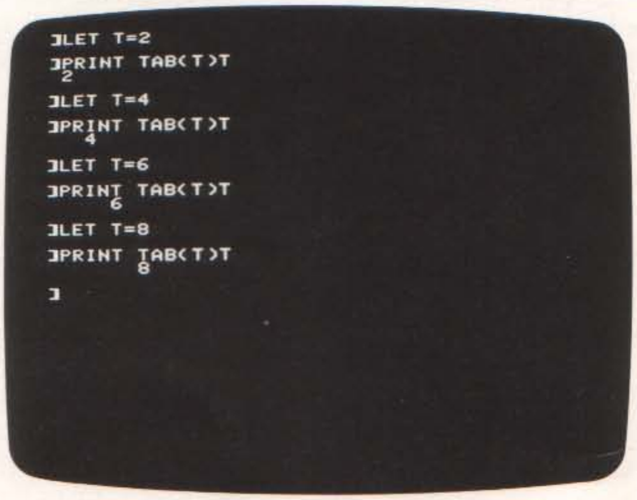
USING **TAB**



When you use TAB do not leave a space between TAB and the bracket that follows it; if you do, the Apple will not carry out the command.

In the example above a number is used in brackets to set the position of the TAB. But, as you have discovered, a variable is simply a way of labeling a number, so you can also use a variable inside the brackets, and the Apple will use its value to TAB to a column. Try this example:

TAB WITH A VARIABLE



As you can see, TAB has the same effect if you use a number or a variable. But be careful: if you try to TAB to a column you have already PRINTed beyond, the TAB will have no effect. Try the following example and notice that TAB10 appears in the wrong place because BEGINNING OF LINE is in the way.

PRINT "BEGINNING OF LINE";TAB(10); "TAB10"

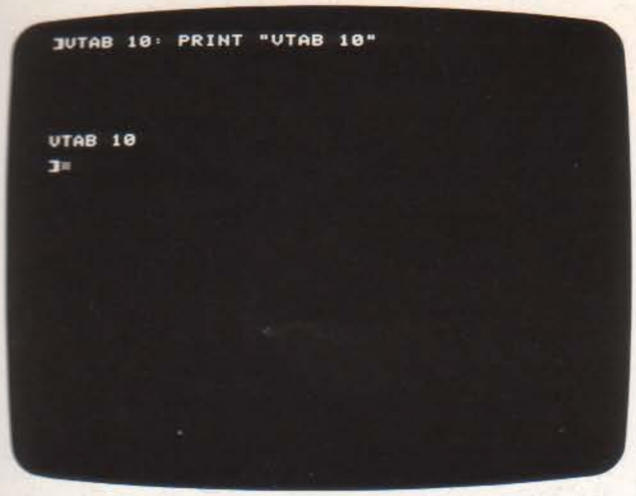
Introducing VTAB and HTAB

As well as using TAB in a PRINT command to specify the horizontal position of a number or string, you can introduce two more commands which will move the cursor to a position on the screen. These are used without PRINT. Clear the screen by typing HOME and try this:

VTAB 20

You will notice that the cursor jumps to the bottom of the screen. VTAB stands for Vertical TAB and it can change the vertical position of the cursor on the Apple's screen. It can be used before a PRINT statement to move the cursor to a particular line. Clear the screen again, type the line shown at the top of the next screen, and then press RETURN.

VERTICAL **TAB**

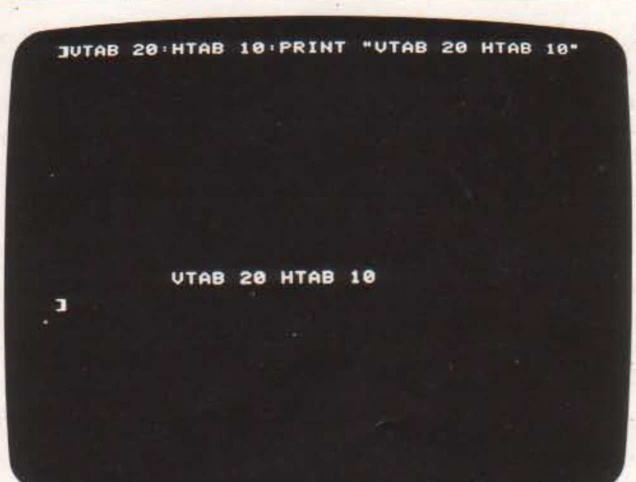


The colon is used to separate different commands typed on the same line. The computer obeys both commands before stopping for you to type another. In this example the command VTAB positions the cursor first, and the next command PRINTs at this position.

From the top to the bottom of the screen, there are 24 "lines". The first line is 1 and the last, 24. You must not use VTAB with a number outside this range.

As well as VTAB for vertical positioning, you can use HTAB (Horizontal TAB) to move the cursor to a "column" on the screen. In the same way as VTAB, you can use this as a command on its own. But if you just type HTAB with a number you will find that nothing seems to happen. In fact, the Apple obeys the HTAB command, but too quickly for you to see. The cursor moves to the column number which you gave it but then moves on immediately to the beginning of the next line to show that it has carried out your command and is ready for the next one. To see HTAB at work type the first line shown below and press RETURN:

HORIZONTAL **TAB**



This time you have given the computer three separate commands on one line. First the VTAB moves the cursor down to line 20, then the HTAB moves it across to column 10 and finally the computer PRINTs the string. The Apple has a "40-column display", so you can use numbers between 1 (left of the screen) and 40 (right of the screen) with HTAB.

If you try to VTAB or HTAB to an invalid line or column you will get an "error message". Try this for example:

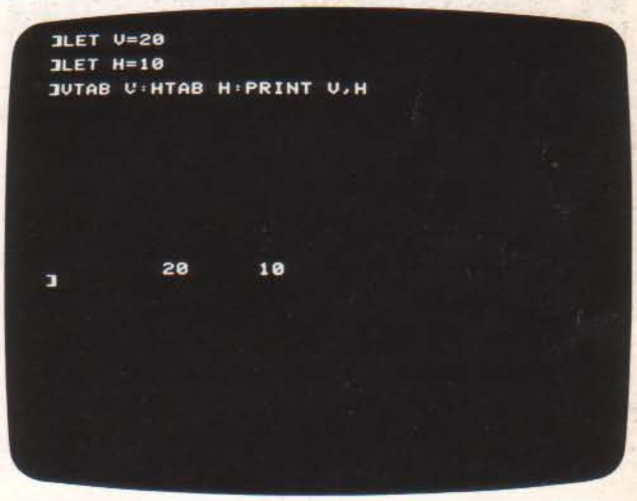
```
VTAB 50: PRINT "T"
```

The Apple will respond with "?ILLEGAL QUANTITY ERROR" to tell you that you have tried to VTAB to a line that does not exist.

Using VTAB and HTAB with variables

VTAB and HTAB are powerful commands and they can be used to move the cursor across and up and down the screen to PRINT numbers, strings and also variables. You can use variables with VTAB and HTAB in the same way as with TAB:

VTAB AND **HTAB** WITH VARIABLES



The importance of positioning

Remember that you must use VTAB, HTAB and PRINT on the same line in order to PRINT at the correct row and column on the screen. If you type them in as separate lines the Apple will move the cursor to the correct place on the screen but then the cursor will move again, ready for your next command. Also, if you forget the colons between commands the Apple won't understand the line you've typed and you will get an error message.

Now that you've begun to use the Apple you will gradually start to feel more confident. So don't be afraid to experiment, and try to work things out for yourself – you can't do any harm to the computer and it is the best way to learn.

COMPUTER CALCULATIONS

The PRINT command is not limited to simply displaying characters on the screen. You can also use it to perform calculations on your Apple.

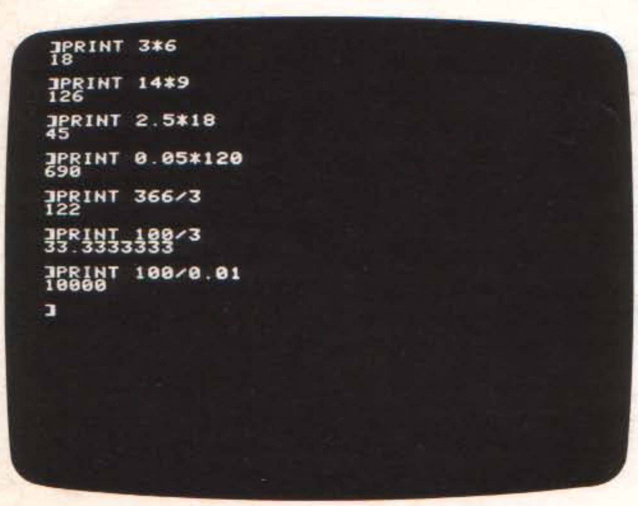
Let's take addition first. The plus (+) sign is next to the DEL key. Because it is the upper of the two symbols on the key-top, the SHIFT key must be pressed at the same time as the plus key. To add two numbers together, use PRINT followed by the calculation. Type in the following, then press RETURN:

```
PRINT 2+2
```

Subtraction is carried out in the same way. The minus sign, which doubles as a hyphen when used in text, is to the left of the plus key. It is the lower of the two symbols so there is no need to press SHIFT. The screens below show simple additions and subtractions, and multiplications and divisions:

ADDING AND SUBTRACTING

MULTIPLYING AND DIVIDING

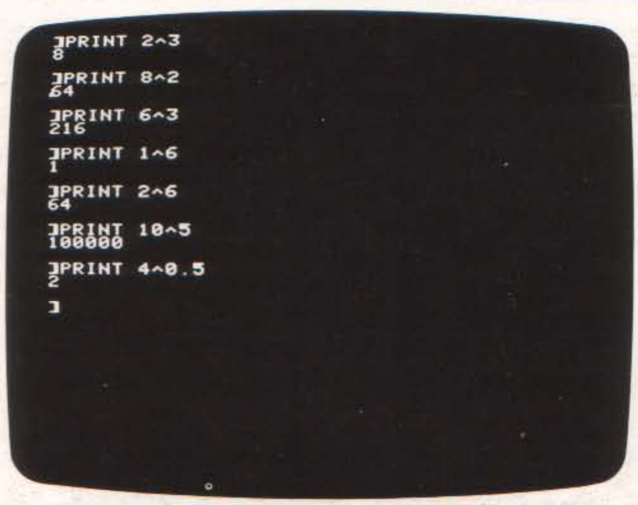


Multiplication is not carried out with the familiar "×" symbol but with an asterisk (*). The asterisk is the upper SHIFTed symbol on the number 8 key. Division uses the oblique stroke (/) next to the right-hand SHIFT key. In 24/8, for example, the left-hand number is divided by the right-hand number. You will find that you quickly get used to the computer's multiplication and division symbols.

Calculating exponents and square roots

In addition to these familiar math functions, you can multiply a figure by itself a specified number of times (called exponentiation), and calculate square roots on the Apple. For example 2∧3 is equivalent to 2 multiplied by itself three times. In other words 8. The keyboard cannot produce superscripts like the 3 in 2^3 – which is how this calculation is normally indicated – so you have to use the "up arrow" (∧) symbol. This is the upper symbol on the number 6 key, so you will need to use SHIFT. Here are some examples:

EXPONENTS



The Apple also allows you to find the square root of a number. This time there isn't a single key that carries out the calculation; instead you have to type in a command like this:

```
PRINT SQR(2)
```

Make sure that you use the round brackets on the number keys 9 and 0, and not the square or curly brackets next to the RETURN key. When you press RETURN after keying in this line, the computer will PRINT the answer. However, if you try this command with a negative number, the computer will produce an error message to let you know that you have asked for a mathematical impossibility.

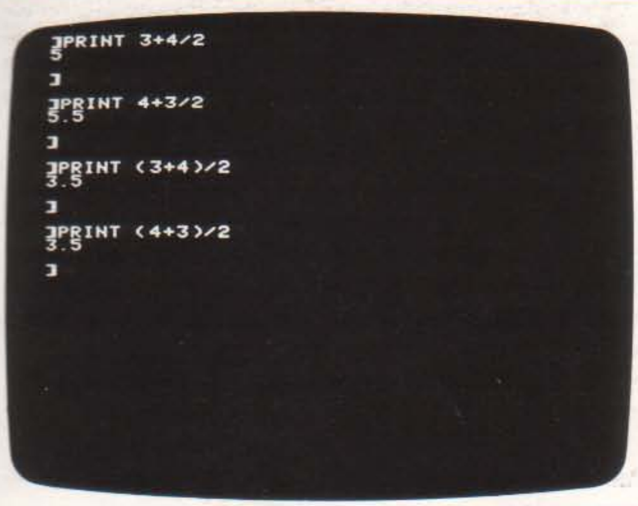
You can carry out a number of different calculations using a single PRINT command. Try experimenting with addition and subtraction. You will discover that the Apple's ability to calculate seems endless:

MULTIPLE CALCULATIONS

How to specify a sequence of calculations

You can enter the figures for each addition and subtraction in any order you like, and the result will be the same. However, when you introduce multiplication and division to the chain of calculations, unexpected things can happen. Say you want to add two numbers together and divide the result by two. Look at the next screen, and try the calculations for yourself:

THE EFFECT OF VARYING ORDER



Since 3+4 is exactly the same as 4+3, why should the computer produce three different answers when you divide it by two? The reason is that the Apple doesn't always carry out calculations in the order you type them. It performs exponentiation first, then multiplication and division, and finally addition and subtraction. So in PRINT 3+4/2, the 4 is divided by 2 before 3 is added to the result, and in PRINT 4+3/2, 3 is divided by 2 before 4 is added; so both fail to perform the task you set.

To add 3+4 and then divide the result by 2 and achieve the answer 3.5 you must change the order in which the computer performs calculations by introducing a pair of parentheses. This is shown in the third and fourth examples on the screen. Here, the addition within the parentheses is carried out first and then the result is divided by 2. So, whenever there are parentheses in a calculation, the computer works out the calculation inside the parentheses first.

What are the Apple's limits?

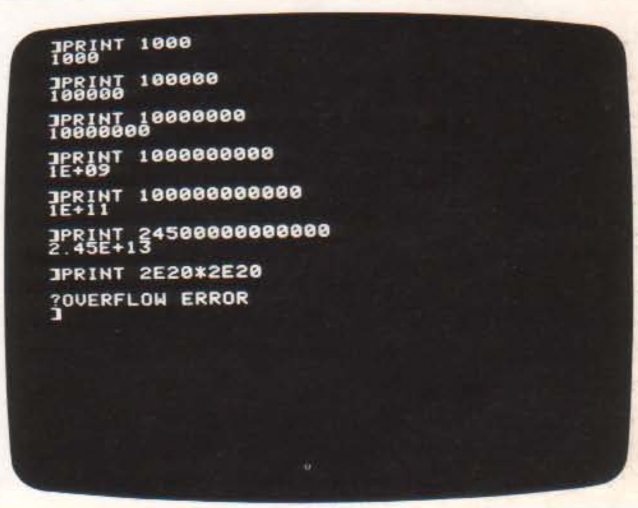
There are two limitations to the numbers that the computer can handle – size and accuracy. The size limitation is unlikely to cause you a problem. Numbers with a decimal point can have any value in the range 1×10 ∧ 38 (1 followed by 38 zeros) to 1×10 ∧ −39 (1 divided by 1 followed by 39 zeros). Whole numbers can have any value from −999999999 to 999999999.

Although the largest number that the Apple can hold is 1 followed by 38 zeros, the computer only memorizes the first nine of these digits – the rest are set to zero. This nine figure accuracy is adequate for most applications. But sometimes small errors in calculations do occur. Try:

PRINT 9 ∧ 2

The answer should of course be 81. But the computer introduces a small "rounding error" in its calculations. You may come across other similar quirks. Try typing PRINT 2000000000000; it produces 2E12 on the screen (the E stands for exponent). This is simply a shorthand way of displaying 2 followed by 12 zeros. Try entering large numbers and calculations and notice the way the computer responds to them:

PRINTING LARGE NUMBERS



WRITING YOUR FIRST PROGRAM

So far the Apple has responded immediately to the commands you have typed, and the commands have been very simple – in many cases it would have been quicker not to use the computer. However, commands on their own are not computer programs. The computer reads each command, carries it out and forgets it. A program, on the other hand, is an orderly list of instructions which the computer stores in its memory. It can carry them out as and when you wish.

When you have a task that you want your Apple to carry out, the first job is to write a program in steps that the computer can understand. The Apple uses a language called BASIC (Beginners' All-purpose Symbolic Instruction Code). BASIC is an example of a high-level language – that is, one composed of words and symbols with which you, the programmer, are already familiar. It is therefore one of the easiest programming languages to learn.

So what is a computer program? Simply a list of commands, like those you have already keyed into your computer, but with line numbers at the beginning of each line:

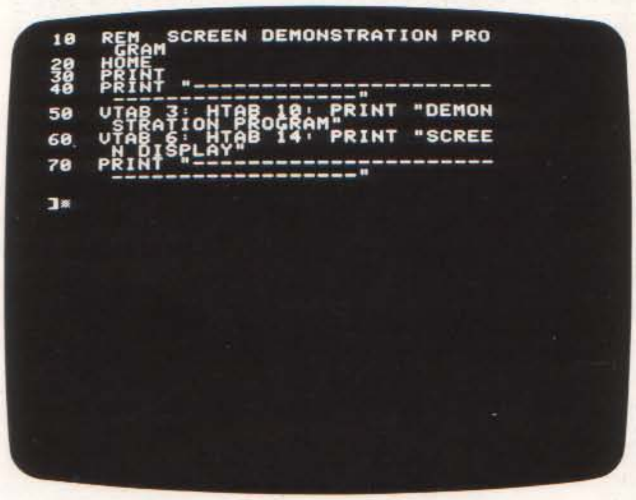
USING LINE NUMBERS

As you key the program in, you will notice that now the commands are not carried out as soon as you press the RETURN key. Instead, the program is safely stored in the computer's memory until you are ready to start it – by typing RUN, and then RETURN.

You may be wondering why the lines are numbered in steps of ten. When you are writing and testing programs, you will often find that you want to go back to an earlier stage and add a line here and there, and since the Apple runs programs in numerical order, it is not sufficient to simply add it at the next ascending line number. It has to be given a line number which reflects its correct position in the program. If you were to write the program with the lines numbered 1,2,3,4 and so on, there would be no room to insert new lines later, whereas if you begin 10,20,30,40 there's room to add several lines between each existing line.

The program you have just typed in is still in the Apple's memory, so before you try another you must erase it. To do this type NEW and press RETURN. NEW tells the computer to erase any program in its memory, ready for you to key in another. But beware, there is no opposite of NEW, so if you type NEW at the wrong time you may have to do a lot of retyping. After you have cleared the screen, type:

SCREEN DISPLAY PROGRAM



Taking it from the top, what does REM mean? REM is short for REMark. The computer doesn't do anything with a REM statement other than store it with the rest of the program. But it's a useful device for making notes to yourself about sections of a program.

As your programming ability develops you will find REM lines very valuable for reminding you how a particular program works. Other people will also be able to follow your programs more easily if you put in REM "statements" to explain precisely what you are doing at each stage of the program.

HOME you have come across already. It's a quick way of taking all the old unwanted information off the screen. Using PRINT on its own (line 30) may at first seem a little crazy. PRINT tells the computer to send whatever follows it to the screen and move on to the beginning of the next line. So here, with nothing following, it just moves to the next line and leaves a one-line space. The next PRINT gives a line of hyphens and line 70 does the same. Lines 50 and 60 contain the HTAB, VTAB and PRINT commands explained on page 16. When you've typed it in, RUN it to see what happens.

How to correct typing errors

Even in a short program like this it is easy to make a typing mistake that will prevent the program from working. But the computer only recognizes the most recently entered version of any line, so if you have made a mistake, just re-type the line correctly – with the same line number – at the end of the program, and the computer will use this version in the correct place when the program is RUN. This program uses the techniques demonstrated on pages 18–19. Remember to type NEW again, before keying it in:

CALCULATIONS PROGRAM

```
10  PRINT "3*18=";3 * 18
20  PRINT "6.25*4=";6.25 * 4
30  PRINT "179*9.8=";179 * 9.8
]
```

Now type RUN. Everything inside quotes is displayed exactly as in the program, and the result of each calculation is displayed on the same line. This is the purpose of the semi-colon; it ensures that whatever follows it is displayed on the same line. Correct spacing is also vital if you want to produce a legible display of strings and numbers on the same line. The next program, and the display it produces, demonstrate how spaces in strings appear when a program is RUN:

CONVERSIONS PROGRAM

```
10  PRINT "CONVERSIONS"
20  PRINT
30  PRINT
40  PRINT "1 FOOT=";12 * 2.54;" C
    ENTIMETERS"
50  PRINT
60  PRINT "1 POUND=";16 * 28.35;"
     GRAMS"
70  PRINT
80  PRINT "10 KILOMETERS=";10 * 5
     / 8;" MILES"
90  PRINT
100  PRINT "1 DAY=";24 * 60 * 60;
    " SECONDS"
]
```

CONVERSION DISPLAY

```
CONVERSIONS

1 FOOT=30.48 CENTIMETERS
1 POUND=453.6 GRAMS
10 KILOMETERS=6.25 MILES
1 DAY=86400 SECONDS
]
```

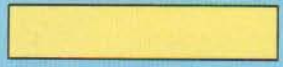
If a program is to RUN properly, it must carry out the correct operations in the right order. Drawing a flowchart is a useful way of outlining the steps involved in making the computer perform a task. The flowchart below shows how to plan a program to add up all the numbers from 1 to 1000. Each shape indicates a separate operation, and the arrows connecting the shapes show the path that the program is to follow. "NUMBER" and "TOTAL" represent figures that can be entered in a program as the numeric variables N and T. This program contains two features which you will encounter later – a program "loop" and a program "decision point". The first is explained in detail on pages 30–31 and the second is dealt with on pages 40–41.

DRAWING A FLOWCHART

This flowchart shows the individual steps needed to add together all the numbers from 1 to 1000.

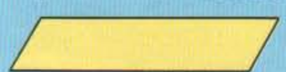
Key

Terminator Signals start and end of flowchart

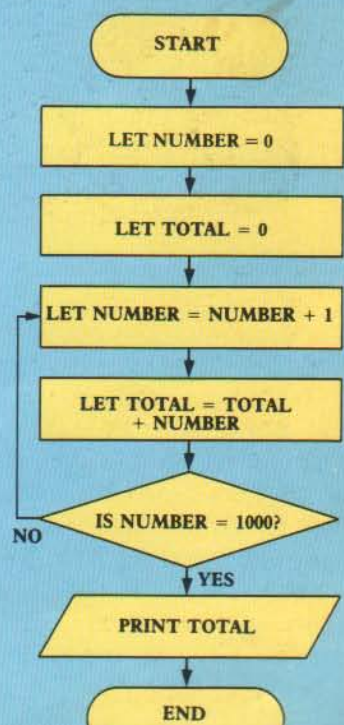


Instruction Identifies each separate operation

Decision point Instructs the computer to make a decision



Input/Output Instructs the computer to take in or give out information



DISPLAYING PROGRAM LISTINGS

As you start writing programs, you will often want to RUN a program and then refer back to the "program LISTing" in order to check something or perhaps to alter it in some way. In order to do this you must be able to recall a program to the screen after it has been RUN.

This is the function of the BASIC command LIST. After a program has been RUN, if you type LIST the Apple will recall the listing back to the screen from the part of memory where it is stored.

LISTING A PROGRAM

```
]10 LET A$="LONDON"
]20 PRINT A$
]LIST
10  LET A$ = "LONDON"
20  PRINT A$
]LIST
10  LET A$ = "LONDON"
20  PRINT A$
]
```

LISTing doesn't alter the program or remove it from the computer's memory. What you see on the screen is an exact copy of the program as it is held inside the Apple. If you want to make sure of that, type HOME to clear the screen, then type LIST again and watch the program reappear on the screen. Now key in the program shown below.

OPERATOR PROGRAM

```
]10 INPUT "WHAT IS YOUR NAME ";N$
]20 PRINT "*****************************
**********"
]30 PRINT "APPLE IIE PROGRAMMED BY ";N$
]40 PRINT "*****************************
**********"
]LIST
10  INPUT "WHAT IS YOUR NAME ";N$
20  PRINT "***********************
    *****************"
30  PRINT "APPLE IIE PROGRAMMED B
    Y ";N$
40  PRINT "***********************
    *****************"
]
```

This program will show you how to use LIST more selectively. It also demonstrates a technique that you will be using soon. Incidentally, when you RUN it remember to press RETURN after typing your name. As before typing LIST will display the whole program.

LIST is a very useful tool for developing a program. All you have to do to check that you have entered lines correctly is to type LIST and press RETURN. But LIST is not limited to displaying the entire program. In the case of a long program, which will not all fit on the screen at once, you might only want to see a few lines.

Using the previous program as an example, type LIST 10. Only line 10 will be displayed on the screen. You can also use the LIST command to display a range of line numbers. For example LIST 20,40 will display all of the lines in a program between line 20 and line 40:

PARTIAL **LIST**ING

```
]LIST 10
10  INPUT "WHAT IS YOUR NAME ";N$

]LIST 20,40
20  PRINT "***********************
    *****************"
30  PRINT "APPLE IIE PROGRAMMED B
    Y ";N$
40  PRINT "***********************
    *****************"
]
```

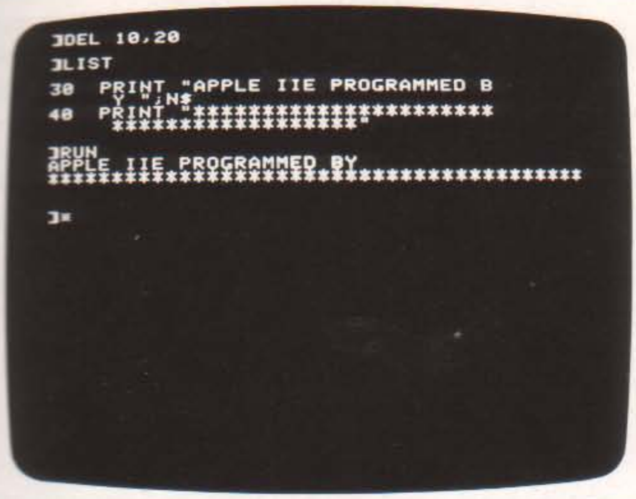
If you study the last two screens carefully, you will notice that LIST doesn't always show exactly what you typed in. Look at line 20. Some spaces have found their way into the string of asterisks. This is because LIST tries to display your program neatly on the screen. If you RUN the program the spaces will not be PRINTed because they are only in the LISTing, not in the Apple's memory. You will soon get used to the way that LIST "formats" your screen LISTings. Most of the programs in this book are shown in their LISTed form.

Removing lines from a program

Sometimes, instead of replacing a line by typing a new one, you may wish to remove a line completely. If you type the wrong line number, for instance, you will want to remove the line from the computer's memory.

LIST the OPERATOR PROGRAM again and then type 10 and press RETURN immediately. Now LIST the program once more and you will find that line 10 has been deleted. This is a good way to delete single lines, but if you want to delete several lines it is tedious to type in all the line numbers. So the command DEL – for DELete – allows you to remove a range of line numbers from the computer's memory:

DELETING LINES



But be careful – like NEW – there is no command for unDELeting lines.

How to RUN small sections of a program

As your programming skills improve you will find that your programs become progressively longer. However there are few programmers who can write lengthy programs without making one or two mistakes. On pages 24 and 25 you will discover how to go about correcting these mistakes but what if they appear right at the end of a program? Do you have to keep re-RUNning the entire program before you can observe and experiment with the problem part?

Fortunately the answer is no, because just as you can LIST sections of a program, you can also jump to and RUN any section of a program. Simply type RUN followed by the appropriate line number. You may find however that if your program is short it's just as convenient to RUN the whole program and in certain circumstances you will find that the program won't operate correctly if you try to RUN just a section of it.

In the screen that follows the OPERATOR PROGRAM has been LISTed and then followed by RUN 30. The computer goes straight to line 30, and then carries out the remainder of the program. However, because you've bypassed the INPUT at line 20, N$ will not have a value when line 30 is PRINTed. This is because every time a program, or a section of it, is re-RUN, any variables that have already been allocated are erased from the Apple's memory.

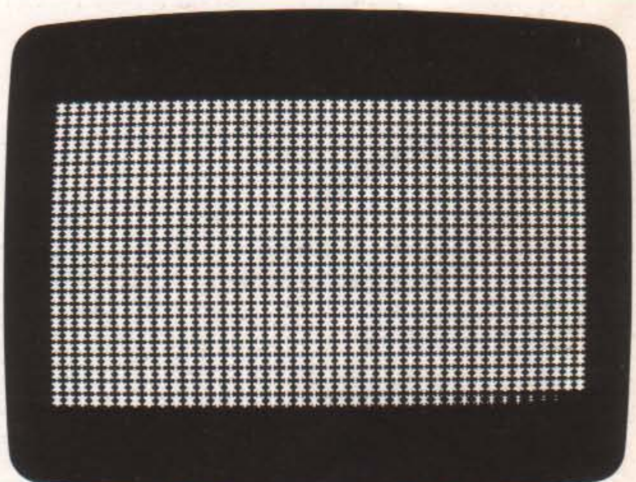
PARTIALLY **RUN** PROGRAM

Introducing GOTO

You can get exactly the same effect with the keyword GOTO. GOTO is one of the simplest and most useful commands in the BASIC language. Used without a line number in front of it, GOTO makes the computer go straight to a specified line and then RUN a program from that point. But when GOTO is actually part of a program, the results are very interesting. Key in this short program to see what GOTO can do:

```
10 PRINT "*";
20 GOTO 10
```

GOTO DISPLAY



Don't worry if you're puzzled about why this has happened; we will return to GOTO later after you've mastered a few more BASIC keywords. But in the meantime, you will find that your Apple will go on PRINTing asterisks forever – unless you stop it. Hold down the "CTRL" key and press RESET and the display will stop.

CORRECTING MISTAKES

Mistakes are unavoidable in computer programming. Programs very rarely work satisfactorily first time, and the longer they are the more difficult it is to get them right. But it's important to realize that making mistakes and correcting them is one of the most valuable exercises in program development – they are an inevitable part of the process and an aid to learning.

For instance, you can't alter the punctuation of a program without changing the sense of what you've written. As you saw on page 21, punctuation means something very precise to the computer, and if you get it wrong, a program may not work.

Once you've spotted a mistake how can you correct it? You can edit a program in two ways. First, as you have seen, you can simply retype a line, and the new version will automatically replace the old one in the computer's memory. However, if there's very little wrong with a line, especially if it's a long one, it's time-wasting to retype it completely. The alternative is to edit the existing line using the ESCape key and the four cursor keys.

The editing procedure given below applies to the Apple IIe; owners of earlier models should consult the Apple owner's manual on the appropriate procedure.

On the Apple IIe the four cursor keys are labeled with arrows at the bottom right of the keyboard and they are used with the ESC key. Here is a program that needs editing:

PROGRAM BEFORE EDITING

To correct line 30 to read:

30 PRINT "CHARLES DE GAULLE, PARIS"

you could retype the line. But try using the screen editor instead. First type in the program and RUN it. Now LIST the program on the screen. Press the ESC key at the very top left-hand corner of the keyboard.

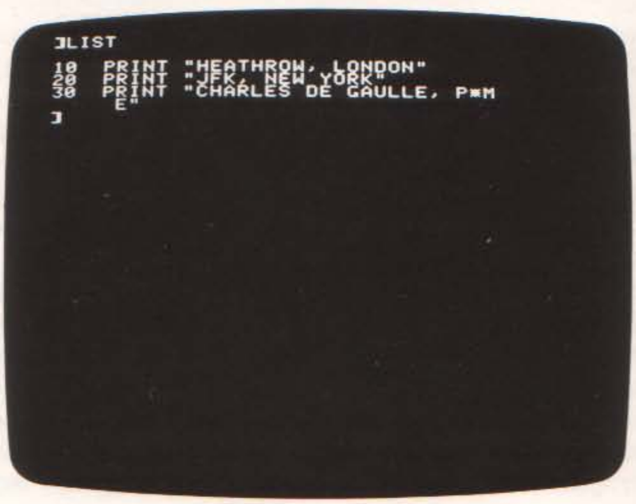
You will not see anything happen but this changes the way that the four arrow keys (called cursor keys) work. Normally you can use only the ← and → keys, but when you press ESC you can use all four.

Press the ↑ key now, until you reach the line you wish to change (line 30). Now press the ← key until the cursor is over the 3 of the line number, then press ESC for a second time. This is a vital step in the procedure as it changes the cursor keys back to their normal way of working, and enables you to make changes to a program.

In this mode, the ← key backspaces over the character to the left of the cursor. As it does so it removes the character from the Apple's memory although it still remains visible on the screen. To change a character, simply position the cursor over the character you wish to alter and type in the new one. The ← key is also useful when you first type in a program, to correct mistakes as you go along.

The → key moves the cursor to the right and copies each character it passes over into the computer's memory – as if you had just typed it on the keyboard. So, if when editing you always start from the very left of the line and keep pressing the → key until you get to the part you want to change, the old part of the line will be safely stored in memory:

PROGRAM DURING EDITING



Do this now with line 30 and then type PARIS over the top of ROME. Now press RETURN to indicate that you have finished editing this line. In this example the correction takes the cursor to the end of the line, if it had not, you would then have had to press → until you reached the end of the line before pressing RETURN.

As you will remember from page 22 LIST adds extra spaces to your program when it displays it on the screen. So when you use the → key to copy characters from the screen into your program, the extra spaces will be copied as well. Usually this won't matter, but it can cause problems if you copy extra spaces into a string. This is because spaces in a string are printed exactly as they appear, and this will spoil the appearance of your string. Fortunately there is a way to stop LIST adding spaces. This is done by reducing the screen width from 40 characters to 30. For now, don't worry about the details of this, just type POKE 33,30 – the last number refers to the width you want. To get back to the normal 40 characters, type POKE 33,40. Try doing a LIST and edit in this way:

CHANGING THE SCREEN WIDTH

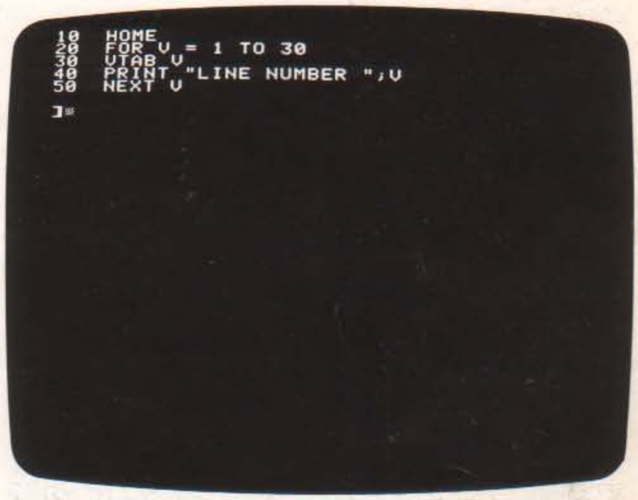
Remember to change the width back after you have edited the program or it will not RUN properly. You can change the screen to different widths but you should not make it less than 1, or more than 40; if you do, strange things will happen.

Bugs and error messages

Mistakes in programs are called "bugs", and the business of removing them is called "debugging". When you RUN a program containing a line the Apple doesn't understand it will print an "error message" and stop the program. This alerts you to mistakes.

Even if every single line of your program makes sense to the computer, the program may still not RUN properly. You may have inadvertently told the computer to do something impossible – to divide a number by zero, for instance. It responds to this by displaying an error report on the screen. In fact the Apple can display over 17 different error messages. Each report describes the type of error and gives the line number on which it occurred – for example, "DIVISION BY ZERO ERROR IN 100". Here are some slightly more advanced programs which will not work. Check the error reports they produce on the table opposite.

BUGGED PROGRAMS



ERROR TABLE

Here are some error messages that you may encounter when writing your programs.

Code	Message	Reason
16	SYNTAX ERROR	Incorrect punctuation, spelling, etc.
53	ILLEGAL QUANTITY ERROR	The number given in a command is too large.
69	OVERFLOW ERROR	A number has become too large for the computer.
90	UNDEFINED STATEMENT ERROR	A GOTO command has tried to go to a line which does not exist.
133	DIVISION BY ZERO ERROR	An attempt has been made to divide a number by zero.
163	TYPE MISMATCH ERROR	An attempt has been made to store a string in a numeric variable, or a number in a string variable.
176	STRING TOO LONG ERROR	Strings can only be 255 characters long.
191	FORMULA TOO COMPLEX ERROR	Too many brackets in a formula.

HOW TO KEEP YOUR PROGRAMS

As you know, the computer only stores the most recent program you have entered in RAM. But even this is only stored in memory for as long as the computer is left switched on. The moment you turn the power off, your program is lost. It is therefore vital to learn how to use floppy disks and the disk drive to make permanent copies of your programs.

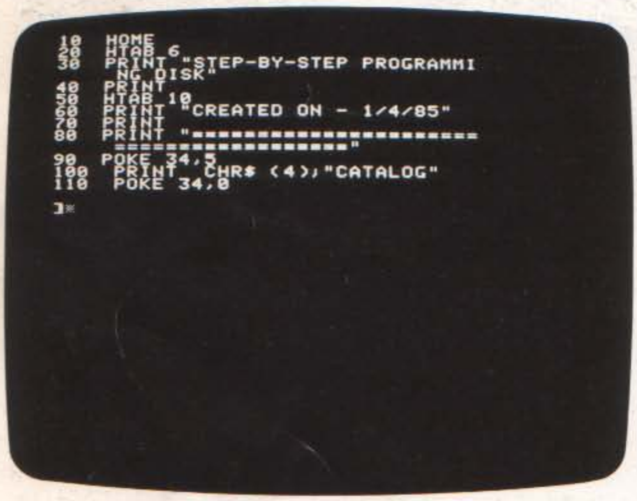
To SAVE a program on disk you will need the master disk you used on page 14. DOS 3.3 and ProDOS "format" disks in a different way, if you have both disks it is therefore vital that you select just one of these disks for the formatting procedure and use it consistently throughout. You will also need a new blank 5¼in. disk on which to store your programs.

Re-boot the Apple now with DOS 3.3 or ProDOS and then follow the appropriate instructions below according to which master disk you're using.

DOS 3.3 users start here

Carefully remove the master disk from your disk drive, replace it with a blank disk and type in this program:

HELLO PROGRAM



This is called a "hello" program. In future when you boot the new disk the Apple will greet you with the title of the disk and the date it was created. Now type the command to initialize your new disk:

INIT HELLO

The drive will come on; when it stops your new disk will be ready. You can test your new disk by "re-booting" the system: hold down CTRL, and Open-Apple and press RESET. This procedure fools the Apple into believing that you have only just turned it on, so it immediately consults the disk drive and loads the disk you have just initialized. This is what you should see:

HELLO PROGRAM DISPLAY

Now let's try to SAVE a program on your new disk – for example, the CONVERSIONS program on page 21. Enter the program into the computer and type:

SAVE CONVERSIONS

CONVERSIONS is the "filename" you've given to this program. You can give a program any filename providing it is different from all the others on the same disk. If it's not different the new program will be recorded over the previous program of the same name and you'll lose the original. To check that your program has been SAVEd, type: CATALOG. This command displays a list of all the files SAVEd on a disk. You should see the filename CONVERSIONS on the list of contents now. Once you have SAVEd a program you can take the disk out of the disk drive and switch the Apple off. When you next want to use the program you can simply recall it from disk. Just for now though, leave the Apple on and type this:

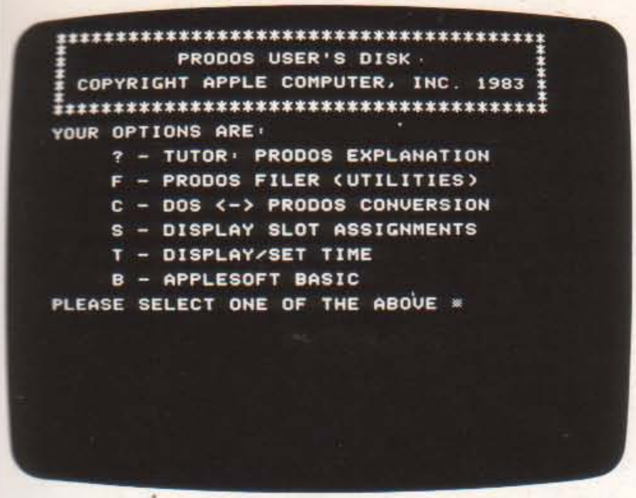
NEW
LOAD CONVERSIONS
LIST

The NEW command will clear the Apple's temporary memory (RAM) and so lose CONVERSIONS. But the next line uses the LOAD command to bring it back from the disk into RAM. The LIST command will show you that this has happened. If you want to recall a program but can't remember which filename you gave it just type CATALOG and the contents list should jog your memory.

ProDOS users start here

If you have just loaded ProDOS the screen will display this "Main Menu":

PRODOS USER'S DISK MAIN MENU



But if you loaded ProDOS some time ago, you will have to re-call this menu by typing:

RUN STARTUP

Once the main menu is on your screen type F to select the "ProDOS Filer". Then type V to select the "Volume Command Menu". Now remove the ProDOS User's disk from the disk drive, and replace it with the new, blank disk. Shut the door and type F to select the Format command. This display asks for the "slot" and "drive" numbers and for a name for the disk. The slot number is the number of the expansion slot on the Apple's printed circuit board, which holds the floppy disk interface card. You could have plugged your card into any one of several slots so the Format command needs to know which interface card to access. Similarly each card can support two drives so the Format command also needs to know which of the drives you wish to use. If you have followed these instructions however you can select the default options by simply pressing RETURN twice. Finally, type in a name for the disk – known as the "volume name" – and press RETURN once more.

The Apple will now format the disk. When it has finished it will display the Format screen again. Now take out the new disk and insert ProDOS. Then press ESC until the ProDOS Filer menu reappears. Select the "quit" option by typing Q. The Apple will ask which "pathname" you wish to use. Don't worry about the meaning of this at the moment, just press RETURN and the ProDOS Main Menu will appear.

You are now almost ready to start saving your programs on the newly formatted disk. All that remains is to leave the Main Menu and return to Applesoft BASIC. You can do this by typing B (for BASIC) and the Applesoft prompt "]" will appear. Now remove ProDOS once again and replace it with the new disk.

The next step is to type in a program and then SAVE it. Once the program is on the screen select a suitable name to label it. It can be anything you like so long as it is less than 15 characters long and starts with a letter. The remaining characters can be letters or numbers – but not spaces.

Now type SAVE, press the space bar once and then type the name of your program. Say you wanted to SAVE the CONVERSIONS program on page 21 you would enter the program and then type:

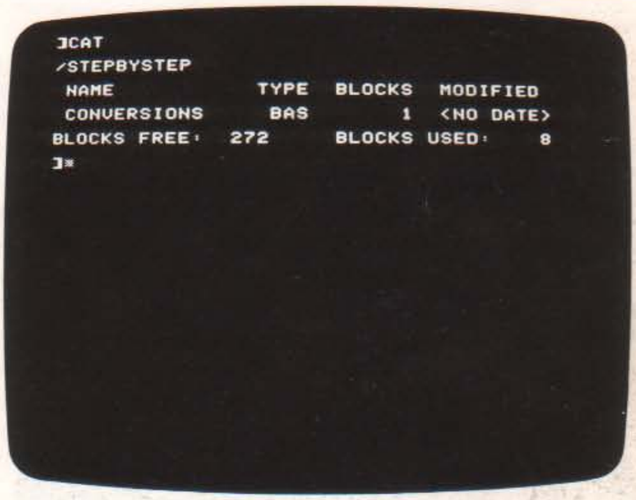
SAVE CONVERSIONS

Wait for the disk drive to stop and then, to check that it has been SAVEd, type:

CAT

This is short for CATalog and will display a list of all the programs SAVEd on a disk. You should see something like this:

CATALOG OF **SAVED** PROGRAMS



Having successfully SAVEd your program you can switch your Apple off knowing that the program is stored safely on disk. To reLOAD the program at any time type LOAD, press the space bar, and then enter the name of your program. To recall CONVERSIONS you would type:

LOAD CONVERSIONS

After LOADing a program it can be LISTed or RUN in the usual way. In ProDOS (and DOS 3.3) you can also use the RUN command to LOAD and RUN a program from the disk in one step, like this:

RUN CONVERSIONS

Always remember to use exactly the same name to LOAD a program as you did to SAVE it; you won't get a response otherwise. If you find you can't remember the name you allotted to a program, just type CAT and let the display jog your memory.

COMPUTER CONVERSATIONS

The programs you've written so far have given the computer a set of instructions and left it to carry them out. Each program has had just one outcome, which was exactly the same every time the program was RUN. But few real programs are like this; in a game, for example, the players feed the computer with new instructions every time the program is RUN and the computer responds to these instructions by changing the display accordingly.

Indeed, it's difficult to write a program of any complexity without being able to interrupt the program while it is RUNning to feed in new information.

Introducing INPUT

The BASIC command INPUT allows you to type information into a program as it RUNs. INPUT lets you carry on a "conversation" with the Apple – you "talk" to it through the keyboard and it "talks" to you through the screen.

The INPUT command tells the Apple that at a certain stage of the program it must pause until something has been entered on the keyboard and then save the entry in memory. It is always used with a variable – a numeric variable if the information entered is a number, or a string variable if the information is in string form. This variable can then be used later on in the program. Here is an example of INPUT at work:

USING **INPUT**

```
10  HOME
20  PRINT "WHAT IS YOUR NAME ?"
30  INPUT N$
40  PRINT "************************
    ****************"
50  PRINT "APPLE IIE PROGRAMMED B
    Y ";N$
60  PRINT "************************
    ****************"

]
```

The program instructs the computer to display the question "What is your name?". Line 30 then stops the program, leaving the question PRINTed on the screen. The computer is waiting for information from you. There's no need to hurry – there isn't a time limit. The computer will wait until you type in the information it needs. Type your name and press RETURN. The program then continues.

The INPUT line of the program takes your name and labels it with the string variable N$. The dollar sign shows that the computer has been programmed to expect a string. This program is similar to the one used on page 22 as an example of LIST. You can see from that earlier example that the command INPUT can also be used to PRINT:

COMBINING **INPUT** WITH **PRINT**

```
10  HOME
20  INPUT "WHAT IS YOUR NAME ? ";
    N$
30  PRINT "************************
    ****************"
40  PRINT "APPLE IIE PROGRAMMED B
    Y ";N$
50  PRINT "************************
    ****************"

]
```

Many programs use INPUT a number of times to gather different items of information. It is quite easy to do this. Just remember that you will need a separate variable for each INPUT.

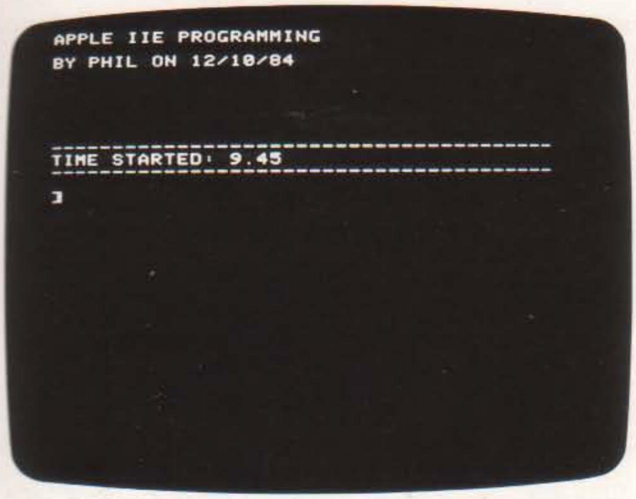
In the previous program N$ was used to label a string – in that case it was a name. But a string isn't restricted to just letters, it can include some numbers although the Apple will treat any numbers included in a string in the same way as letters.

MULTIPLE **INPUT** STATEMENTS

```
10  HOME
20  INPUT "ENTER NAME: ";N$
30  INPUT "ENTER TODAY'S DATE 00/
    00/00: ";D$
40  INPUT "ENTER TIME 00.00: ";T$

50  HOME
60  UTAB 5
70  PRINT "APPLE IIE PROGRAMMING"

80  UTAB 7
90  PRINT "BY ";N$;" ON ";D$
100  UTAB 14
110  PRINT "-----------------------
    -----------------"
120  PRINT "TIME STARTED: ";T$
130  PRINT "-----------------------
    -----------------"

]
```

In lines 30 and 90, the program labels and then uses the variable D$, which is the date. If the variable had been just D, you would have got the error message ?REENTER when you tried to type the oblique separating day, month and year. This is the Apple's way of telling you that the / character cannot be part of a number: it can however, be part of a string.

MULTIPLE **INPUT** DISPLAY



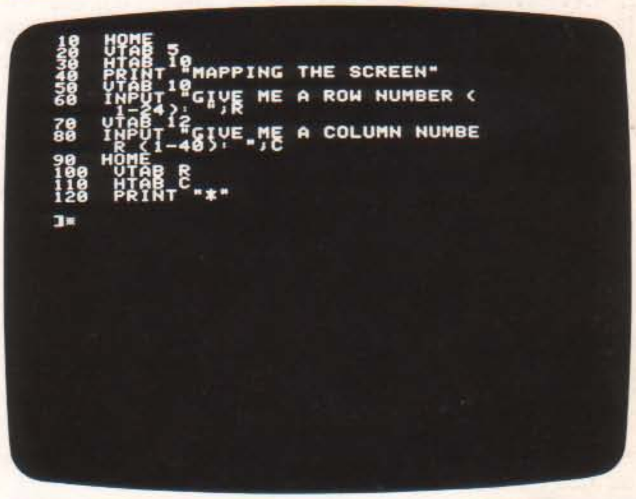
Because you can use INPUT to gather numbers as a program is RUN, the command has many practical applications. Consider, for example, the problem of converting lengths, sizes or weights from one unit of measurement to another. The conversion is always the same: 2.54 centimeters to the inch, 2.2 pounds to the kilogram, 1.6 kilometers to the mile, and so on but the numbers in each new calculation are different. Here is a simple conversion program to try out:

INPUT CONVERSION PROGRAM

The program asks you how many centimeters you want to convert into inches, waits for your response, does the conversion and then displays the result on the screen. Because the INPUT line has a numeric variable – C – you can only INPUT a number. C is then used to calculate the conversion.

In the next example, the program uses INPUT with VTAB and HTAB. See if you can work out what it does, before reading the explanation below.

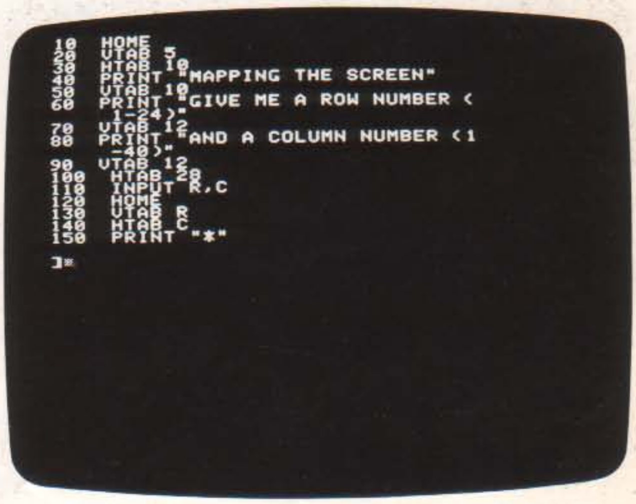
INPUT WITH **VTAB** AND **HTAB**



Line 50 moves the cursor to the 10th line on the screen before the INPUT in line 60, so the message "Give me a row number" is printed on line 10. VTAB and HTAB work with INPUT exactly as they work with PRINT. After collecting the values of the two variables R and C from you, the program PRINTs an asterisk at the position you gave it.

You can change this program to make a single line with one INPUT statement collect both figures. Type in the program below, RUN it, enter both numbers with a comma between them, then press RETURN.

COLLECTING TWO VARIABLES WITH ONE **INPUT**

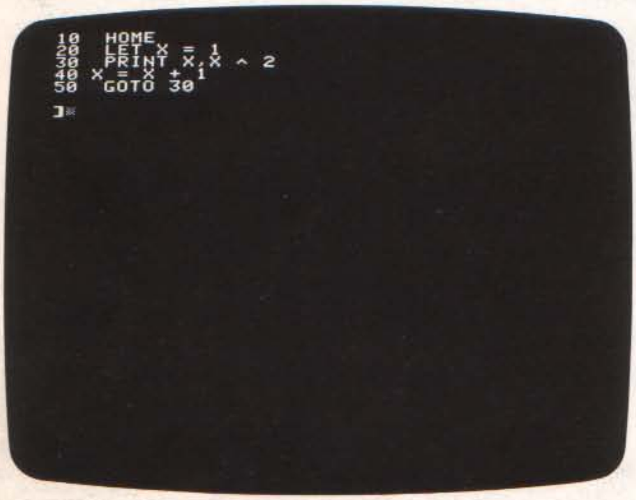


Note that before the INPUT line 90 moves the cursor back to line 12 with VTAB, and line 100 moves the cursor to the end of the message already PRINTed on screen, with HTAB.

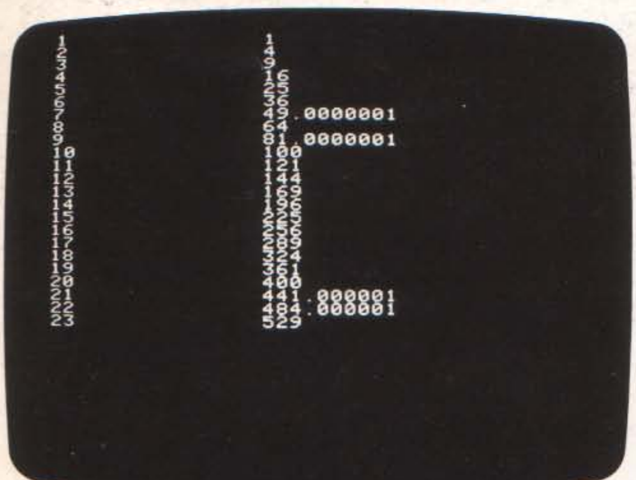
WRITING PROGRAM LOOPS

In a business environment computers are often programmed to perform a task and then repeat all or part of it as often as required. To instruct a computer to repeat itself in this way, you have to write a "loop". There are several ways of writing a loop – on page 23 you came across one which used GOTO. Here is a slightly more complex loop produced by the same method:

NEVER-ENDING LOOP PROGRAM



NEVER-ENDING LOOP DISPLAY



Note that line 40 gives X a value without using LET. The Apple allows you to use this "shorthand" method when changing the value of a variable.

RUN this program now and you will see the disadvantage of using GOTO alone – the program is never-ending and it will continue to RUN until the numbers get so big that an "OVERFLOW ERROR" message is displayed. To stop it, hold down CTRL and press the letter C.

STOPPING A LOOP WITH CTRL-C

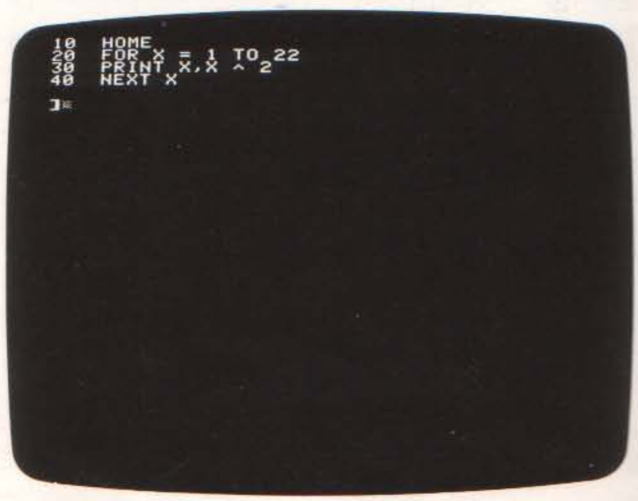
As you can see, when CTRL-C is used to stop a program the Apple displays the line number at which it was interrupted. This can sometimes be useful when you're debugging programs.

The other thing to note in this program is that line 50 does not send the computer back to line 10, the very beginning of the program. If it did, X would always be equal to 1, and the screen would clear each time the first line was PRINTed.

How to stop a loop

The way to avoid endless program loops is to use the BASIC commands FOR and NEXT. These allow you to set limits on how many times a loop is carried out. It is easy to adapt the above program to use FOR. . . NEXT and, as you can see below, this both improves and shortens the program.

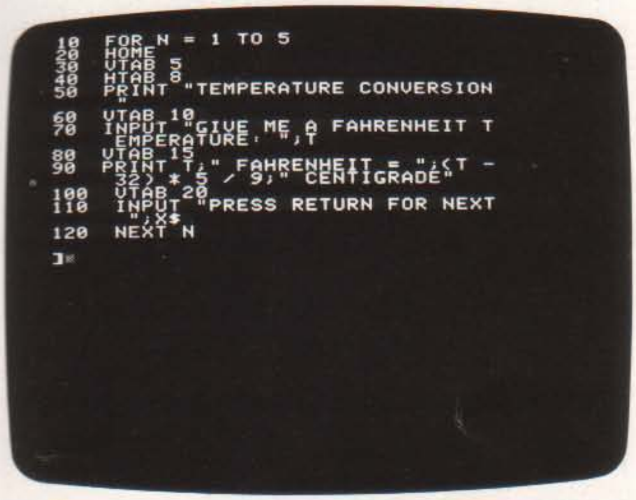
FOR. . .NEXT LOOP PROGRAM



Note that you don't have to include LET X=, or add 1 to X on each loop of the program now, because FOR. . .NEXT takes care of the increment automatically. It starts off by setting X equal to 1 and PRINTing X and X-squared. Line 40 asks for the NEXT value of X and the program is repeated again from line 20. This continues until X has a value of 22, the maximum set by line 20, when the program stops.

If necessary the program can be interrupted each time it is repeated to wait for new information. Try this program which uses INPUT in the middle of a FOR. . .NEXT loop:

FOR. . .NEXT WITH **INPUT**



This program converts Fahrenheit temperatures into Centigrade. The FOR. . .NEXT loop beginning at line 10 sets a limit of five calculations, after which you will have to RUN the program again. The INPUT statement at line 70 stops the program until you type in the Fahrenheit temperature you want to convert. Line 90 then does the calculation and PRINTs the result.

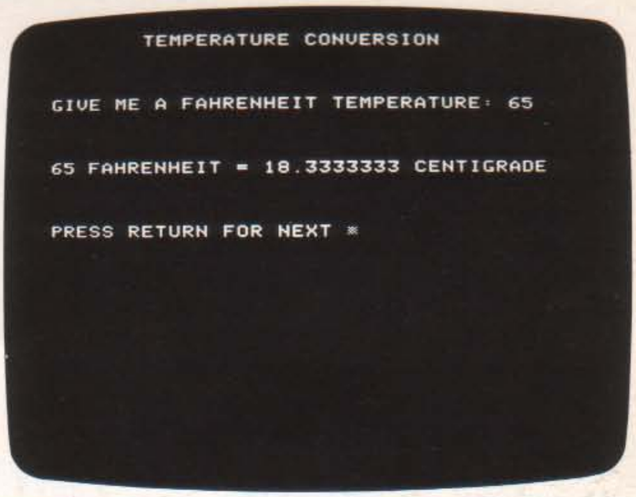
Slowing down a loop

The second INPUT statement at line 110 makes the program pause after displaying the conversion, otherwise it would return to line 20 so quickly after displaying the result, that you wouldn't be able to read it. The string variable X$ does not really label a string because you type RETURN without any other characters – its only purpose is to make the INPUT statement work, so that the computer will stop and wait for you.

How to produce round numbers

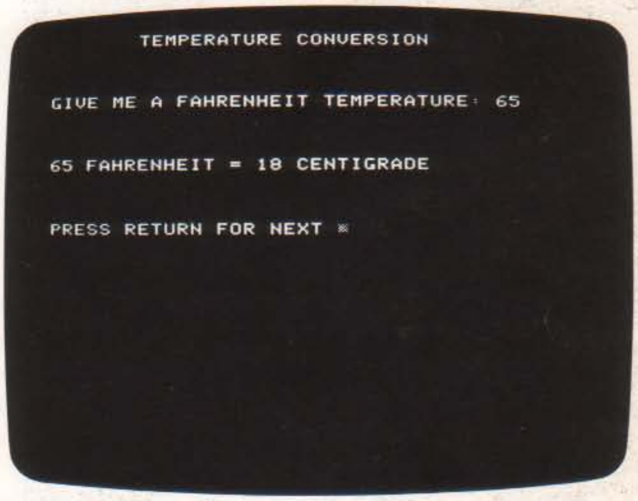
The layout of the conversion display could be improved. It's fine as long as the result of the calculation is in whole numbers, but it rarely is, and the more figures there are after the decimal point, the further "Centigrade" is pushed along the line until it splits, and part of it ends up on the next line:

TEMPERATURE CONVERSION DISPLAY



To get around this try replacing (T−32)⋆5/9 with INT((T−32)⋆5/9+0.5). INT, short for INTeger, turns a decimal number into a whole number. If the result is 18.3333333, for instance, adding INT changes that to 18. A whole number is a more sensible value for a temperature and the display looks neater:

ROUNDED-OFF CONVERSION DISPLAY



When you use INT, it always rounds downward to the next whole number, and this is why the INT line adds 0.5 to the Centigrade value. This ensures that INT always produces the nearest whole number, which is not always the same thing as the next whole number down. If you are confused, try keying in these two direct commands:

```
PRINT 3–1.1
PRINT INT(3–1.1)
```

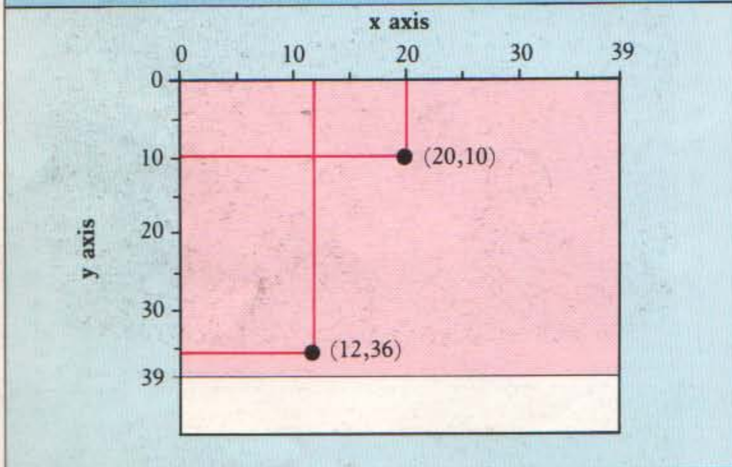
The result of the first is 1.9, and the result of the second is 1. But 1.9 is much nearer to 2 than 1. And it is to compensate for this inaccuracy that 0.5 is added to the converted temperature before the INT is used.

THE ELECTRONIC DRAWING BOARD

Your Apple's BASIC includes several commands for drawing on the screen. If you want to draw a point or line you must have some way of telling the computer where to draw. The screen is therefore divided into a grid of small boxes. Each box is numbered from 0 to 39 across the screen and from 0 to 39 down the screen. At the bottom of this grid you can display four lines of normal text. The 0,0 position is at the top left corner of the screen. The co-ordinates of a point are always given in the form x,y. The number x refers to the number of boxes across the screen from the left, and y refers to the number down from the top of the screen:

HOW TO PLOT ON A LOW-RES GRAPHICS GRID

The x-axis runs across the screen and the y-axis runs down the screen. The bottom four lines are reserved for text.



The TEXT and GRaphics modes

Try typing GR. Any text on the screen will be cleared and you will have switched the Apple into GRaphics mode. In fact, the Apple has two types of GRaphics display, high and low resolution, known as hi- and low-res. The term "resolution" refers to the level of detail which can be displayed on the screen.

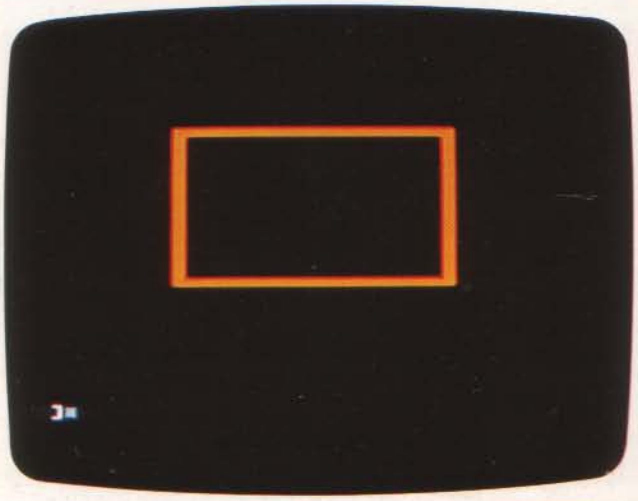
If you can't see the cursor on any of the four text lines at the bottom of the screen, keep pressing RETURN until it appears, then try this:

```
COLOR=15
PLOT 20,10
```

A small white box will appear in the upper half of the screen. PLOT is the command which lights up one of the boxes on the screen.

Once in the GRaphics mode, you must switch the Apple back to the normal text display before you can type in a program; type TEXT to do this. Using TEXT and GR you can move backwards and forwards between the two types of display. Type TEXT now, and enter this program to draw a square box:

BOX PROGRAM



The first thing you will notice is that the display is in color. You will find out more about the COLOR statement on the following pages, but don't worry about it for now. The FOR. . .NEXT loop at lines 40 to 70 draws the top and bottom of the box, and the one at lines 90 to 120 draws the sides. Each loop actually draws two lines at once by adding 20 to the x or y co-ordinate in the PLOT commands at lines 60 and 110.

Introducing VLIN and HLIN

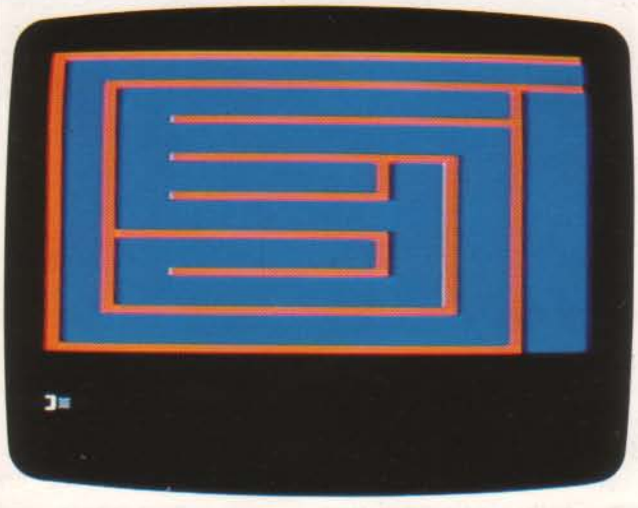
FOR. . .NEXT loops are one way to draw lines, but the Apple has two commands, HLIN and VLIN, specifically designed to draw Horizontal LINes and Vertical LINes on the screen. To draw a line in this way you must give x and y co-ordinates to indicate the beginning and end of the line.

Now return to the TEXT screen once more and type in this improved box-drawing program:

BOX PROGRAM USING **HLIN** AND **VLIN**

This program is both shorter, and easier to understand. The next program uses HLIN and VLIN to draw a maze that could be used in an adventure game:

MAZE PROGRAM

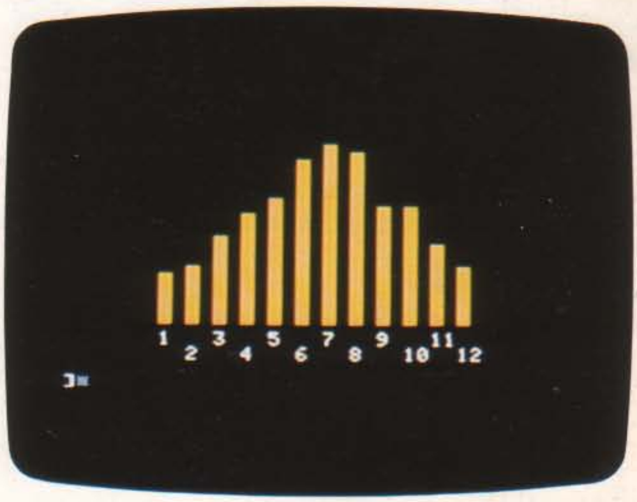


When writing programs to use the low-res screen, map out the co-ordinates on a graphics grid on paper first. You can then translate them more easily into PLOT, HLIN and VLIN commands.

Smartening up facts and figures

One important use of "computer graphics" is the presentation of facts and figures in an easy-to-understand form. In the example below the VLIN command is used to plot a vertical line to represent the average temperature for each month of the year. The numeric variable T holds the temperature.

TEMPERATURE PROGRAM



Because the y co-ordinates are numbered from the top of the screen down, the program subtracts the variable T from 38 (the y co-ordinate at the bottom of the screen) before plotting the line. This displays the graph the correct way up.

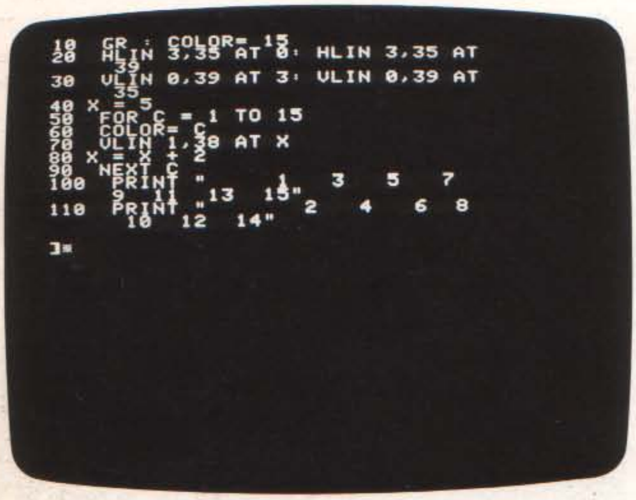
The program uses the four lines of text at the bottom of the screen to label the graph with the month numbers. You could easily change the temperatures to represent the area in which you live.

COLOR GRAPHICS

The Apple can draw on the low-res screen in 16 colors (including black and white). Each color is identified by a number which you use with the COLOR statement. After you have set a color with this command, all PLOTting and lines will be drawn in the selected color until it is changed by another COLOR statement.

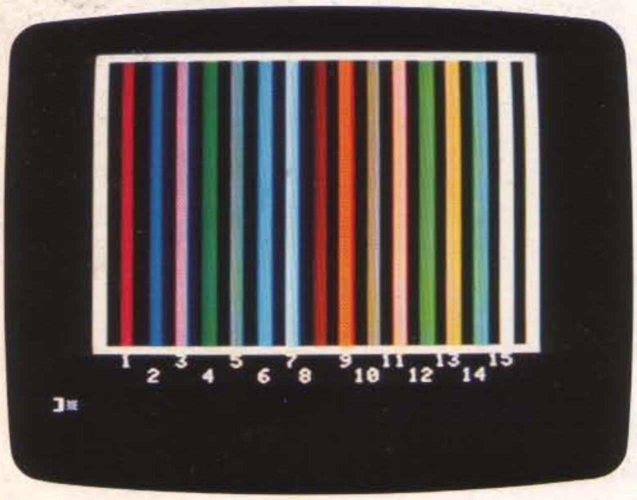
To see the colors that the Apple can produce key in this COLOR CHART PROGRAM:

COLOR CHART PROGRAM



Lines 50 to 90 contain a FOR. . .NEXT loop which draws a line in each of the colors from 1 to 15. The program will display every color, except black:

COLOR CHART DISPLAY

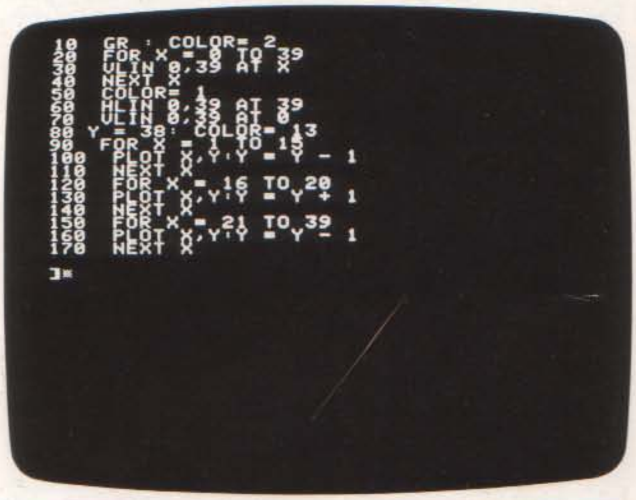


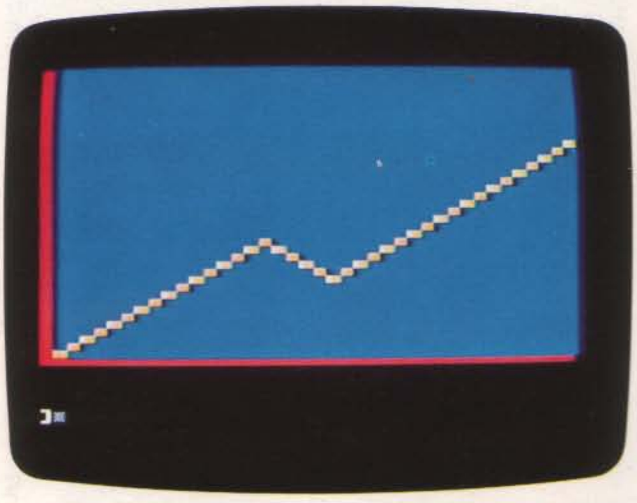
The program uses the text lines at the bottom of the screen to PRINT the color numbers. If you are using a black and white television or monitor, the different colors will show up as various shades of gray. Even in black and white, if you choose the colors carefully, you can produce attractive displays.

Introducing COLOR to a graph

The next program draws a graph on the screen. Lines 10 to 40 fill the entire display with blue, as a background color. The graph's axes are drawn in red by lines 60 and 70. The three FOR. . .NEXT loops between lines 90 and 170 plot the points of the graph in yellow. Try experimenting with this program to draw different graphs or label the x-axis using the four text lines. But remember you must always type TEXT to return to the normal display. You could SAVE a copy of this program on disk and then experiment with the program in RAM. That way you can always re-LOAD the original program and start again if your effort to amend it goes badly wrong.

COLOR GRAPH PROGRAM





Designing shapes from graphics blocks

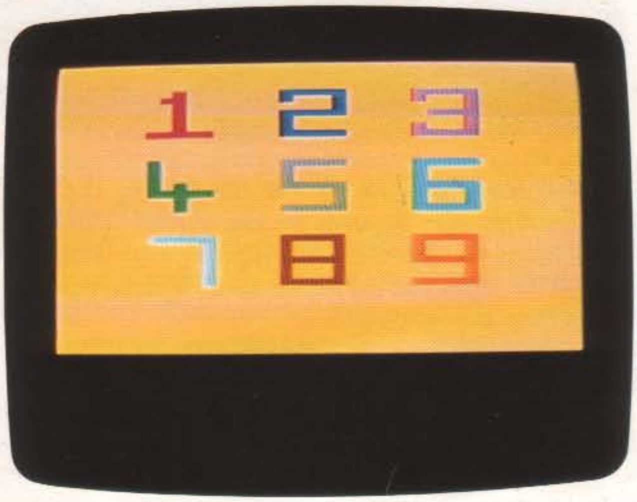
The next program is the longest you have had to key in so far, but that doesn't mean it's any more complicated.

The colorful shapes created by this program are constructed from small "graphics blocks". In fact, if you look closely at the Apple's normal letters and numbers you will see that they too are all constructed from similar, but smaller, blocks.

NUMBERS PROGRAM

```
10  GR : COLOR= 13
20  FOR V = 0 TO 39
30  HLIN 0,39 AT V
40  NEXT V
50  COLOR= 1: HLIN 7,11 AT 10: VLIN
     10,4 AT 9: PLOT 8,5
60  COLOR= 2: HLIN 17,21 AT 10: HLIN
     17,21 AT 7: HLIN 17,21 AT 4:
      VLIN 9,8 AT 17: VLIN 6,5 AT
     21: PLOT 17,5
70  COLOR= 3: HLIN 27,31 AT 10: HLIN
     28,31 AT 7: HLIN 27,31 AT 4:
      VLIN 10,4 AT 31: PLOT 27,9:
      PLOT 27,5
80  COLOR= 4: HLIN 7,11 AT 18: VLIN
     18,14 AT 7: VLIN 20,17 AT 9
90  COLOR= 5: HLIN 17,21 AT 20: HLIN
     17,21 AT 17: HLIN 17,21 AT 1
     4: VLIN 17,14 AT 17: VLIN 20
     ,17 AT 21

]
```

```
100  COLOR= 6: HLIN 27,31 AT 20: HLIN
     27,31 AT 17: HLIN 27,31 AT 1
     4: VLIN 20,14 AT 27: VLIN 20
     ,17 AT 31
110  COLOR= 7: HLIN 7,11 AT 24: VLIN
     30,24 AT 11
120  COLOR= 8: HLIN 17,21 AT 30: HLIN
     17,21 AT 27: HLIN 17,21 AT 2
     4: VLIN 30,24 AT 17: VLIN 30
     ,24 AT 21
130  COLOR= 9: HLIN 27,31 AT 30: HLIN
     27,31 AT 27: HLIN 27,31 AT 2
     4: VLIN 27,24 AT 27: VLIN 30
     ,24 AT 31

]
```



Introducing SCReeN

As well as commands to draw points and lines on the screen, the Apple has a BASIC command to find out the color of any point on the screen. Using this command you can discover if a point has already been plotted and, if so, what color it is. After RUNning the NUMBERS PROGRAM, try typing this:

```
LET CL=SCRN(10,10)
PRINT CL
```

The SCRN command is different from other commands because it "returns" a value. In the example above it gives the color of the point (10,10) on the SCReeN. Commands that return a value like this are called "functions". INT and SQR are also functions. All functions can be used in LET statements in the same way as a variable.

Next, type in these extra lines which will automatically be added to the NUMBERS PROGRAM:

COLOR CHANGING PROGRAM

```
140  FOR Y = 0 TO 39
150  FOR X = 0 TO 39
160 C =  SCRN( X,Y)
170  COLOR= 16 - C: PLOT X,Y
180  NEXT X
190  NEXT Y
200  GOTO 140

]
```

When you RUN the amended program now, the same display will appear, but then the colors will start to change. This is done by the two FOR. . .NEXT loops in lines 140 to 200. Line 160 finds the color of each point on the screen with the SCRN command. Line 170 then subtracts the color from 16 to give a new color number and re-plots the point in the new color. You can use this technique to produce interesting color patterns on the screen.

LOW-RES COLOR NUMBERS

Number	Color	Number	Color
0	Black	8	Brown
1	Magenta	9	Orange
2	Dark Blue	10	Gray
3	Purple	11	Pink
4	Dark Green	12	Green
5	Gray	13	Yellow
6	Medium Blue	14	Aqua
7	Light Blue	15	White

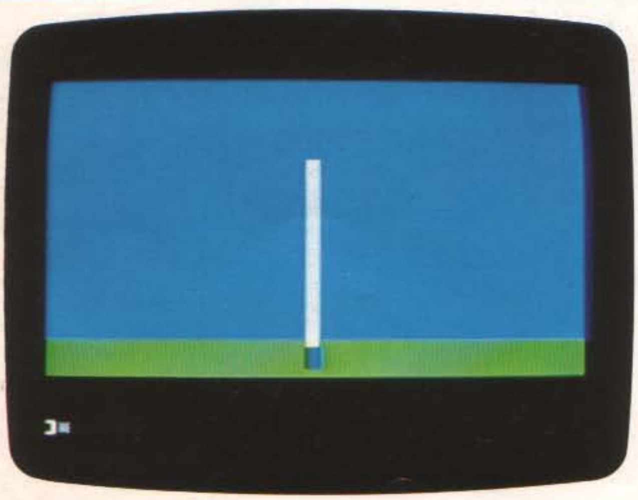
ANIMATION

Once you are confident enough to PLOT a point anywhere on the screen in different colors, you can attempt some simple animation. Animation is achieved by PLOTting a point, erasing it, and re-PLOTting it in a new position on the screen. Suppose you want a program to animate a missile launched at a target. First key in this program and RUN it:

MISSILE LAUNCH PROGRAM

Line 10 switches on the low-res graphics screen. Lines 20 to 40 draw the ground in green. Then the program fills in the blue sky at lines 50 to 70. The missile silo is drawn by line 80, and the FOR. . .NEXT loop at lines 100 to 120 launches the missile. When you RUN this program, the first thing you will notice is that it doesn't do exactly what you want it to. The Apple moves the missile up the screen, but instead of displaying a single moving block to represent the missile, it draws a vertical line:

AFTER IMAGES DISPLAY



How to remove after-images

The problem is that you haven't told the Apple to remove the old unwanted images as the missile moves upwards – so it appears to be drawing a line. But it is quite easy to remove it by PLOTting a blue-colored block behind the missile as it moves. Type in this new line and re-RUN the program:

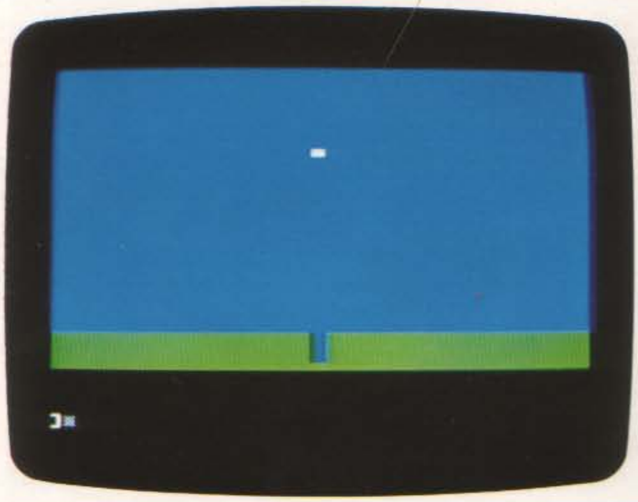
```
102 COLOR=2:PLOT 19,V+1
```

The missile now moves up the screen without leaving a trail. But there are still several improvements you can make. The missile moves very quickly. How can you scale down the speed? One way is to put in a FOR. . .NEXT loop to slow the program down:

```
112 FOR I=1 TO 50:NEXT I
```

Now the missile takes longer to move up the screen. The FOR. . .NEXT loop at line 112 doesn't do anything except occupy the Apple's time before it moves on to the next PLOT command. You can reduce the speed of the missile further still by increasing the value in the TO part of the loop. This will make the Apple go round the loop more times and so decrease the take-off speed:

ANIMATION WITH DELETIONS



You will find that the slower the speed, the more clearly the missile appears on the screen. Flickering occurs when the missile is moving quickly, because of the time it takes the Apple to erase and re-draw the missile as it moves upwards.

Adding details

Now you are ready to improve the program still further. First, you can add a yellow exhaust flame to the missile. And by drawing and erasing the exhaust in the same way as the missile, you can make it appear to follow the missile. The missile also needs a target, so you can draw a spaceship at the top of the screen:

MISSILE LAUNCH PROGRAM WITH ADDITIONS

Line 82 draws the spaceship, and line 112 has been amended to PLOT the exhaust flame as well as provide a time delay with the FOR. . .NEXT loop. Line 102 erases the path of the missile and the exhaust flame with a VLIN command. When you RUN the program now, the missile moves up the screen until it hits the spaceship:

MISSILE LAUNCH WITH SPACESHIP DISPLAY

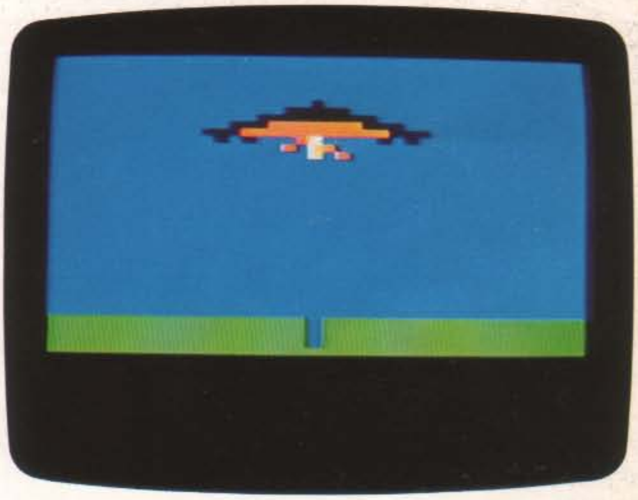
The final touch

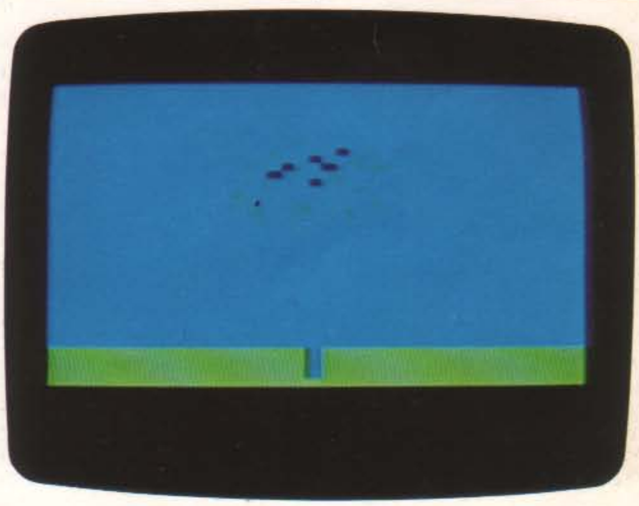
To complete your animated sequence, all you need is an explosion when the missile impacts, followed by falling debris. You can do this using the animation techniques that you used for the missile launch, just type in the EXPLOSION AND FALLING DEBRIS ADDITIONS opposite.

Line 130 plots orange blocks to represent the explosion. The FOR. . .NEXT loop at line 140 slows the Apple down so that the explosion stays on the screen. Line 150 erases the explosion and the spaceship, and lines 160 to 200 repeatedly draw and erase the debris as it falls to the ground:

EXPLOSION AND FALLING DEBRIS ADDITIONS

EXPLOSION AND FALLING DEBRIS DISPLAY

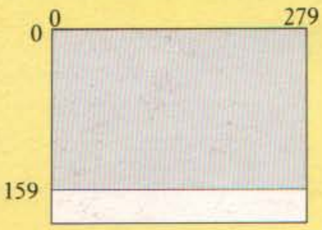




HIGH-RESOLUTION GRAPHICS

In addition to commands for drawing on the low-res graphics screen, your Apple has several BASIC commands for plotting on the hi-res graphics screen. Hi-res graphics allow you to draw and plot in far greater detail:

THE HI-RES GRAPHICS SCREEN



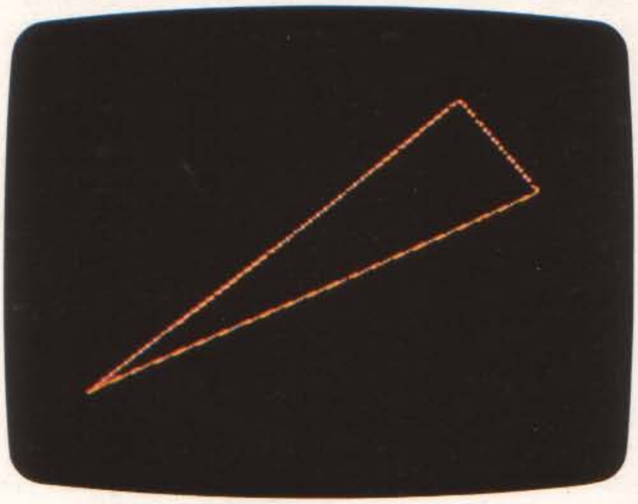
As for low-res graphics the 0,0 position is at the top left and there are four lines for text at the bottom of the screen. Many of the hi-res commands are similar to the ones you used with the low-res screen, but they are preceded by the letter H. Try typing HGR. The screen will clear and the Apple will be in the hi-res graphics mode. The available colors are defined in much the same way as low-res colors, except that only six are now available (including black and white). To plot a point on the screen, try this:

```
HCOLOR=3
HPLOT 140,80
```

A small white dot will appear in the middle of the screen. There are no hi-res commands for drawing lines. Instead the HPLOT command is used with two sets of co-ordinates giving the start and end position of a line. The next program draws a triangle on the screen. But before you can enter it you will need to type TEXT to return to the TEXT screen.

TRIANGLE-DRAWING PROGRAM

TRIANGLE-DRAWING DISPLAY



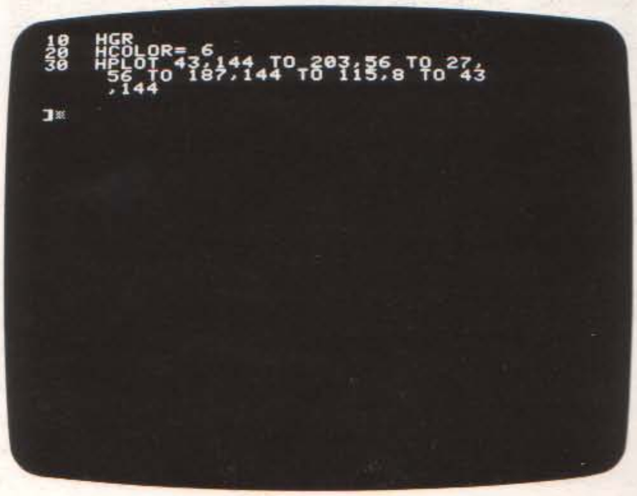
Line 10 switches on the hi-res graphics mode and line 20 sets the color to orange. Lines 30, 40 and 50 each draw one side of the triangle. Notice that the HPLOT co-ordinates on lines 40 and 50 both start where the previous HPLOT ended. In fact, you can leave out these first co-ordinates. Try typing in these lines to replace lines 40 and 50:

```
40 HPLOT TO 240,60
50 HPLOT TO 0,159
```

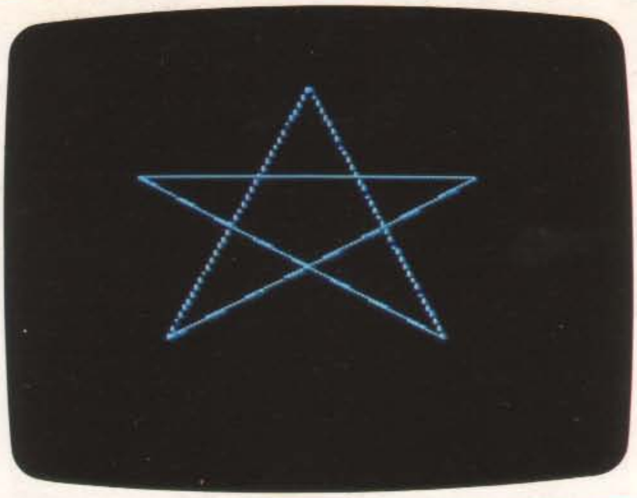
How to PLOT multiple lines

When you HPLOT without a starting position, the Apple will continue to draw from the last point plotted on the screen. HPLOT is a very versatile command. As well as plotting single points and lines, it can be used to plot several lines at once. Try this program:

MULTIPLE LINES PROGRAM



MULTIPLE LINES DISPLAY



The HPLOT on line 30 draws all the lines which make up the shape. It starts at the first co-ordinates 43,144 and draws a line TO 203,56 and then on TO 27,56 and so on until it returns to the starting point.

How to produce solid figures

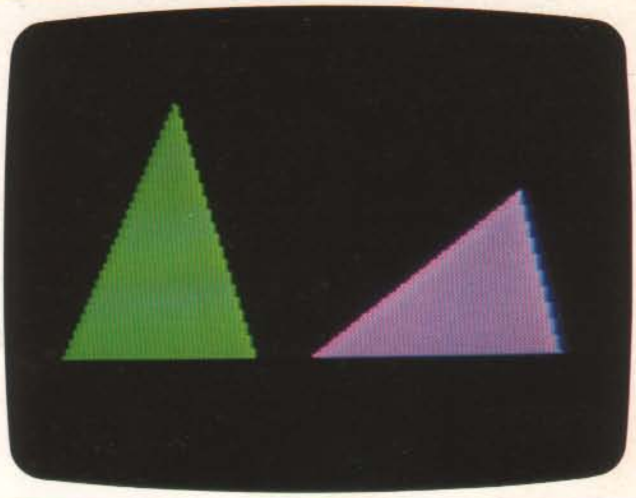
After this, it is very easy to fill in these line drawings to produce solid figures. You just HPLOT a series of lines adjacent to one another. For example, to draw a solid triangle you need to draw a series of lines from a single point (one apex of the triangle) to a gradually shifting point along the triangle's base line. This program draws two triangles in different colors:

SOLID TRIANGLE PROGRAM

```
10  HGR
20  HCOLOR= 1
30  FOR X = 1 TO 100
40  HPLOT 60,20 TO X,159
50  NEXT X
60  HCOLOR= 2
70  FOR X = 130 TO 260
80  HPLOT 240,70 TO X,159
90  NEXT X

]
```

The solid lines of color are drawn by the two FOR . . .NEXT loops. The loop at line 30 increments the value of X from 1 to 100. The HPLOT inside this loop, at line 40, draws lines from the fixed point 60,20 to a point with the y co-ordinate of 159, and a changing x co-ordinate, given by the variable X. The loop at line 70 draws the second triangle.

SOLID TRIANGLE DISPLAY



You don't even need to have fixed co-ordinates in a program. You can use INPUT to allow the person RUNning the program to set the start and end co-ordinates of a line as in the next program:

HPLOTTING WITH **INPUT**

```
10  HOME
20  HGR
30  HCOLOR= 3
40  VTAB 21
50  PRINT "X = 0-279, Y = 0-159"
60  PRINT
70  INPUT "START OF LINE (X1) ?";
     X1
80  INPUT "(Y1) ?";Y1
90  INPUT "END OF LINE (X2) ?";X2

100  INPUT "(Y2) ?";Y2
110  HPLOT X1,Y1 TO X2,Y2

]
```

On both black-and-white and color screens you will find that certain hi-res colors don't always draw vertical lines on the screen. This is because of the way hi-res colors interact with the circuitry in your TV.

HI-RES COLOR NUMBERS

There are restrictions on the number of areas on the screen in which hi-res colors can be plotted. This table indicates which colors can be used in which columns.

Number	Color	Columns available
0	Black 1	Any column
1	Green	Odd-numbered columns
2	Violet	Even-numbered columns
3	White 1	Any column
4	Black 2	Any column
5	Orange	Odd-numbered columns
6	Blue	Even-numbered columns
7	White 2	Any column

DECISION-POINT PROGRAMMING

You now know that if you want to carry out a calculation or put something on the screen 10 times you can write a loop like this:

```
FOR A=1 TO 10. . .
NEXT A
```

But there is also another way – using an IF. . .THEN statement. To take an example, let's say that you want to PRINT the multiplication tables from 1 to 10. This is how you would do it with FOR. . .NEXT:

FOR. . .NEXT LOOP

```
10  HOME
20  PRINT "-----------------------
    ----------------";
30  FOR A = 1 TO 10
40  FOR B = 1 TO 10
50  PRINT A * B; TAB( B * 4);"!";
60  NEXT B
70  PRINT "-----------------------
    ----------------";
80  NEXT A
]
```

MULTIPLICATION TABLES DISPLAY

```
---------------------------------------
1   !2   !3   !4   !5   !6   !7   !8   !9   !10 !
---------------------------------------
2   !4   !6   !8   !10  !12  !14  !16  !18  !20 !
---------------------------------------
3   !6   !9   !12  !15  !18  !21  !24  !27  !30 !
---------------------------------------
4   !8   !12  !16  !20  !24  !28  !32  !36  !40 !
---------------------------------------
5   !10  !15  !20  !25  !30  !35  !40  !45  !50 !
---------------------------------------
6   !12  !18  !24  !30  !36  !42  !48  !54  !60 !
---------------------------------------
7   !14  !21  !28  !35  !42  !49  !56  !63  !70 !
---------------------------------------
8   !16  !24  !32  !40  !48  !56  !64  !72  !80 !
---------------------------------------
9   !18  !27  !36  !45  !54  !63  !72  !81  !90 !
---------------------------------------
10  !20  !30  !40  !50  !60  !70  !80  !90  !100!
---------------------------------------
]
```

IF. . .THEN versus FOR. . .NEXT

The program which follows does the same thing using IF. . .THEN. Lines 30 and 40 set the two variables A and B to 1, which is the first value of the loop. At line 60, 1 is added to variable B ready for the next time round the loop. Line 70 is where the Apple makes a decision by examining B. The < symbol is BASIC shorthand for "less than". So, if B is less than 11, the Apple is told to GOTO line 50 and repeat the loop. If B is more than 11, the GOTO following THEN is not obeyed. Instead the Apple moves on to line 80. A similar test is performed on A at line 100, but notice that the test is A<=10. This means "if A is less than or equal to 10" and it will give the same result as "less than 11":

IF. . .THEN LOOP

```
10  HOME
20  PRINT "-----------------------
    ----------------";
30 A = 1
40 B = 1
50  PRINT A * B; TAB( B * 4);"!";
60 B = B + 1
70  IF B < 11 THEN  GOTO 50
80  PRINT "-----------------------
    ----------------";
90 A = A + 1
100  IF A < = 10 THEN  GOTO 40
]
```

You might wonder what the point of this is, as the IF. . .THEN loop produces exactly the same result as the FOR. . .NEXT loop. But the advantage of IF. . . THEN is that the Apple can respond to information that you INPUT by examining it against criteria that you have set, and taking a decision. Here is an example that illustrates this, and tests your skill at arithmetic:

MATH-TEST PROGRAM

```
10  HOME
20  PRINT "-----------------------
    ----------------";
30  FOR A = 1 TO 10: FOR B = 1 TO
    10: PRINT "??"; TAB( B * 4);
    "!";: NEXT B: PRINT "--------
    ------------------------------
    -----";: NEXT A
40  FOR V = 8 TO 13: VTAB V: HTAB
    10: PRINT  SPC( 20): NEXT V
50 A =  INT ( RND (1) * 10):B =  INT
    ( RND (1) * 10)
60  VTAB 10: HTAB 15: PRINT A;" X
    ";B;" = ";: INPUT C: VTAB 1
    0: HTAB 21: PRINT  SPC( 9);:
    PRINT "? !?? ! ??!";
70  IF A * B = C THEN  FOR I = 1 TO
    5: PRINT  CHR$ (7);: NEXT I:
    GOTO 50
80  PRINT  CHR$ (7);: GOTO 60
]
```

MATH-TEST DISPLAY

The command RND(1) which appears in line 50 is fully explained on pages 42–43; it is used here to select a RaNDom number between 1 and 10 every time the loop is repeated.

Each time the computer repeats the loop, it sets a problem and waits for your answer. It is then faced with two possible courses of action. If you type in the correct answer, the IF. . .THEN statement at line 70 "beeps" the Apple's speaker several times and GOes-TO line 50 for the next problem. If the answer is wrong, then the computer ignores the part of line 70 after THEN and moves on to line 80. This makes just one "beep" on the speaker and returns to line 60 for another attempt.

The SPC command in lines 40 and 60 is yet another way of formatting the Apple's display – SPC stands for SPaCe. The number in the brackets tells SPC how many spaces to PRINT.

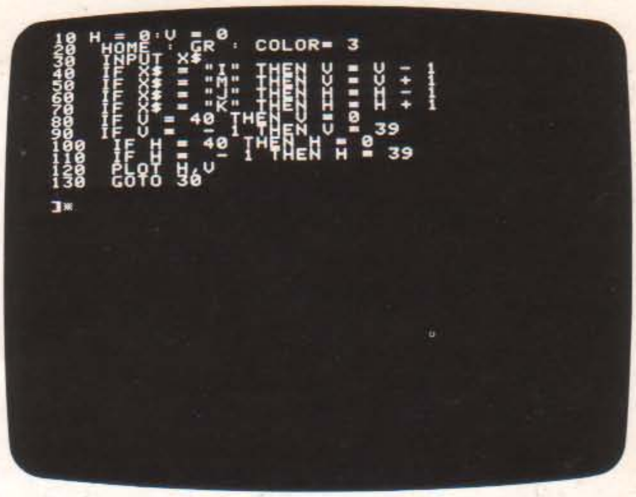
Introducing control characters

The CHR$ command in lines 70 and 80 allows you to put one of the special control characters, mentioned on page 10, into a program. You cannot type these characters directly into your program – you have to use CHR$ instead. The number of the control character you wish to PRINT is given inside the brackets – character number 7 is CTRL-G, which beeps the Apple's speaker.

IF. . . THEN conditions

You can also use IF. . .THEN, in combination with graphics commands, to turn your Apple into an electronic drawing system. All you have to do is to program the computer to respond to INPUT by PLOTting. In this simple program the four IF . . .THEN statements allow the Apple to decide what action to take. The program draws on the screen when you press the I, M, J and K keys to move up, down,

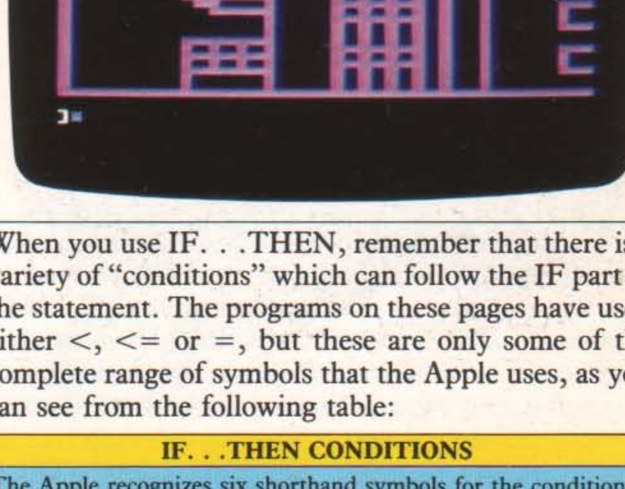
IF. . .THEN GRAPHICS PROGRAM



left and right. After each keypress remember to press RETURN. The Apple will PLOT a point and then move in the direction indicated by the key typed.

The four IF. . .THEN lines make the computer examine your INPUT, and then decide in which direction to move after PLOTting the point. Here's an example of the kind of display this program can produce but you could easily change the color statement in line 20.

IF. . .THEN GRAPHICS DISPLAY



When you use IF. . .THEN, remember that there is a variety of "conditions" which can follow the IF part of the statement. The programs on these pages have used either <, <= or =, but these are only some of the complete range of symbols that the Apple uses, as you can see from the following table:

IF. . .THEN CONDITIONS

The Apple recognizes six shorthand symbols for the conditions that can be tested by an IF. . .THEN loop:

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

UNPREDICTABLE PROGRAMS

Although computers generally work with precise information, doing exactly what you tell them to, an element of chance is necessary in certain applications. For instance, most computer games – like ordinary games – are based to some extent on luck. If you want to make something happen at an unpredictable time, or if dice are to be thrown, or coins tossed, you can't tell the computer what result to produce or the element of chance would disappear.

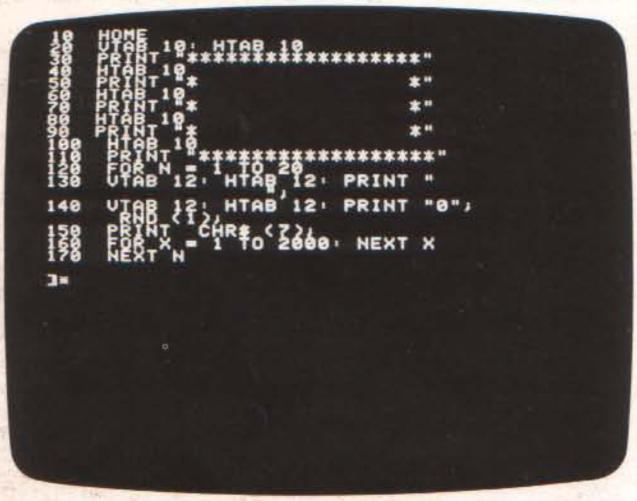
Introducing chance with RaNDom

You can build chance into a program by using RND. This command is similar to the SCRN command on page 35, as it returns a value which can be used in calculations. RND, as you've probably remembered, stands for RaNDom and it allows you to generate random numbers. You can then use these numbers to produce unpredictable sequences. The command is used like this:

```
10 A=RND(1)
```

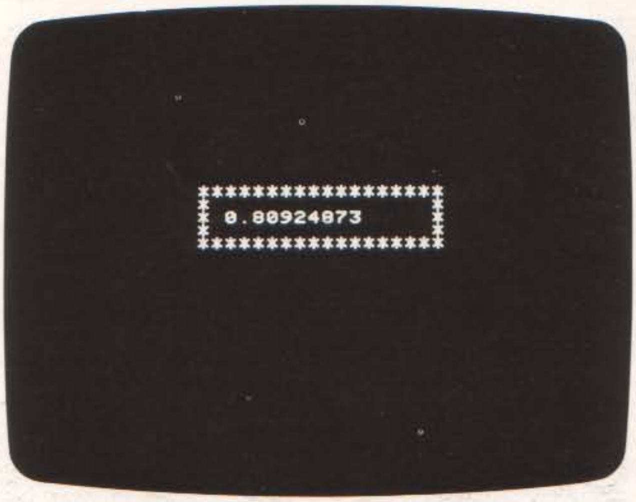
This assigns A, a value between 0 and 0.999999999. Try using RND(1) in the next program, which PRINTs numbers at random:

RANDOM NUMBER GENERATOR



This program uses RND(1) in line 140 to generate random numbers between 0 and 0.999999999, while lines 30 to 110 set up a border of asterisks to frame the numbers. Very small numbers produce the E symbol you came across on page 19. Normally, as each new number is PRINTed, it automatically erases the last number – simply by PRINTing on top of it. However, when something like E-4 appears, it is not automatically erased, so the blank spaces in line 130 take care of that. The FOR. . .NEXT loop at line 160 provides a delay between the PRINTing of numbers.

RANDOM NUMBER DISPLAY



Producing whole numbers with RND

The Apple can generate only a limited range of random numbers. To generate other numbers, you therefore have to multiply the outcome of RND(1). If you replace line 140 with:

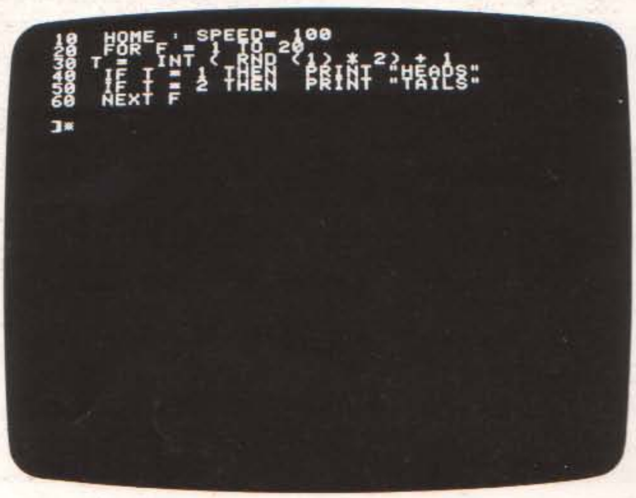
```
140 VTAB 12:HTAB 12:PRINT INT(RND(1)*10)
```

and RUN the program again, you will notice an immediate change in the display. The Apple is now generating whole numbers between 0 and 9 – a much more useful result.

"Heads or tails" on your Apple

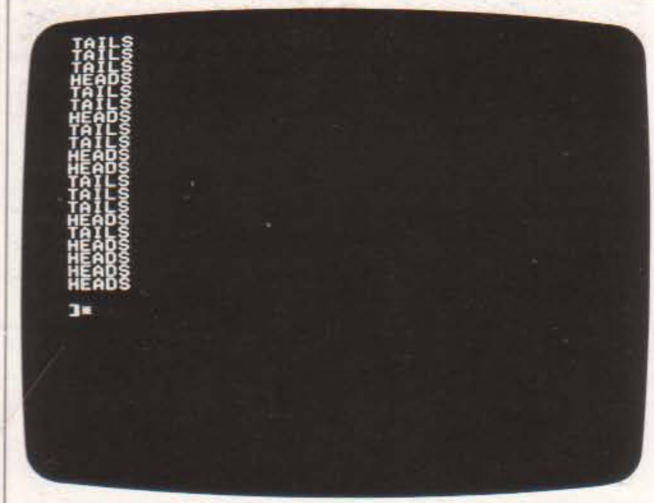
This way of using RND(1) is very useful for programming games involving chance. For example, to simulate throwing dice or tossing coins:

COIN TOSS PROGRAM



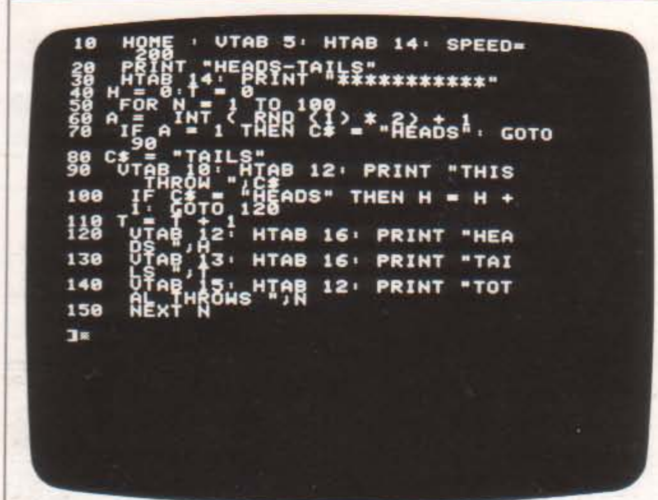
As a tossed coin can have only one of two values – heads or tails – line 30 simulates this process by producing a random number that either has the value 1 or 2. Heads are represented by 1 and tails by 2. Two IF. . .THEN lines assess the outcome and determine what is to be PRINTed and then the program continues. The SPEED command changes the rate at which the Apple PRINTs on the screen – 255 is the normal rapid display, 0 is very slow.

COIN TOSS DISPLAY



It is possible to write a program which will show just how random RND(1) is. If you use RND(1) to toss an electronic "coin" 100 times, you should get roughly 50 heads and 50 tails each RUN. You can actually test to see if this is true. Key in this program:

RANDOM TEST PROGRAM



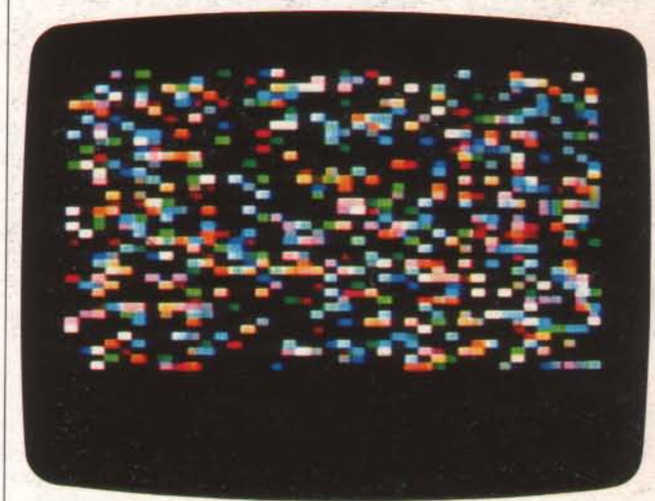
Producing random displays

You can produce some interesting effects with RND(1) by incorporating it in graphics programs so that the computer is instructed to PLOT a point at a random position on the screen. If you then make the computer repeat this process by setting up a loop, you can build up a display which will be different every time the program is RUN. Here is a program which uses RND(1) in this way:

RANDOM GRAPHICS PROGRAM

Line 20 sets COLOR to a random value, and lines 30 and 40 set random co-ordinates for the PLOT statement at line 50. RND(1) is multiplied by 40 to give numbers between 0 and 39.9999999 and INT rounds these to the integer values needed by PLOT. Line 60 sets up an endless loop with GOTO 20. (You will have to type CTRL-C to stop this program RUNning because there is no limit to the loop.)

RANDOM GRAPHICS DISPLAY



However long you let this program RUN, the screen will never completely fill with points. This is because the COLOR numbers selected include 0, which will PLOT a black point. You could write a similar program to HPLOT random points on the hi-res screen. You would need to multiply RND(1) to give co-ordinates between 0,0 and 279,159 and HCOLORs between 0 and 7.

COMPILING A DATA BANK

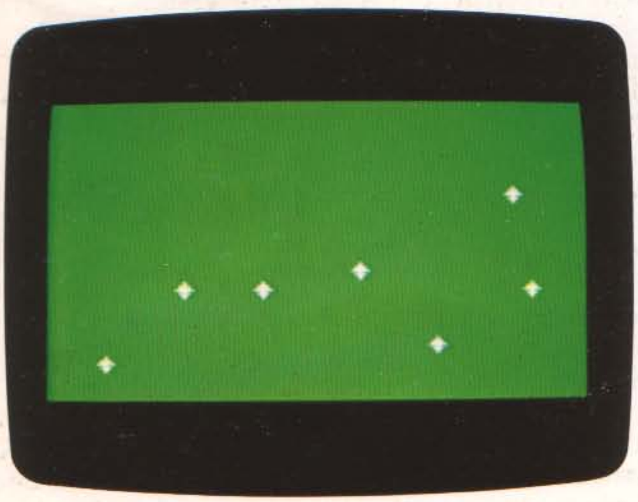
The data necessary for a program can either be collected while it is RUNning by using INPUT, or alternatively it can be written into the program itself using DATA. The commands used to store data are quite straightforward. Data is held in DATA statements and read by READ statements. This program shows these techniques at work:

CONSTELLATION PROGRAM

```
10  HGR : HCOLOR= 1
20  HPLOT 0,0: CALL 62454
30  HCOLOR= 3
40  DATA    30,140,70,100,110,100,
     160,90,200,130,250,100,240,5
     0
50 N = 7
60  FOR S = 1 TO N
70  READ X,Y
80  HPLOT X,Y - 4 TO X,Y + 4
90  HPLOT X - 4,Y TO X + 4,Y
100  HPLOT X - 2,Y + 2 TO X + 2,Y
      - 2
110  HPLOT X + 2,Y + 2 TO X - 2,Y
      - 2
120  NEXT S
]
```

When you RUN this program you should see a computer-generated map of a group of stars called the constellation Ursa Major, also known as the Great Bear or Big Dipper:

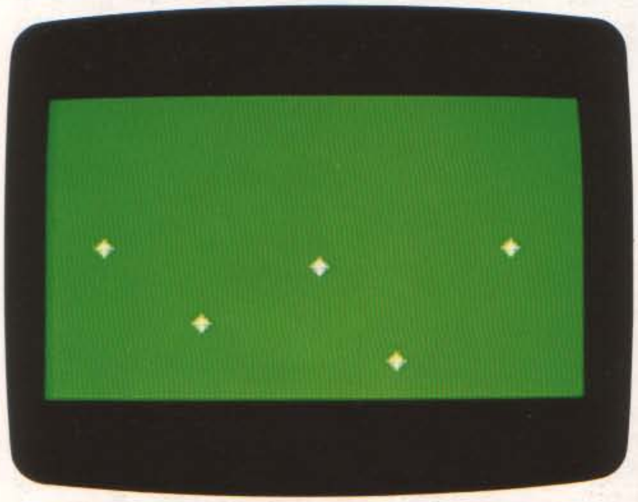
CONSTELLATION DISPLAY



The information for the display is held in line 40 in the form of 14 co-ordinates. Line 70 tells the Apple to READ the DATA in line 40, and to understand it as pairs of co-ordinates to store in the variables X and Y. Line 50 tells the Apple that there will be seven pairs of co-ordinates altogether, by setting the numeric variable N equal to 7. Lines 80 to 110 instruct the Apple to HPLOT a star at each value of X and Y, and so transform the row of DATA into a map on the screen. With a program like this it is easy to change the DATA to get the Apple to HPLOT a new map. Here is a set of line changes and the map that they produce:

```
40 DATA 30,80,80,120,140,90,180,140,240,80
50 N=5
```

CONSTELLATION DISPLAY



When you use DATA statements, it is important to tell the Apple how much DATA there is to READ. Line 50 in the CONSTELLATION PROGRAM shows you how to do this. It sets the limit for the number of pairs of co-ordinates that are to be READ, so that when the Apple has HPLOTted the final star, it stops. If you had not set a limit, the Apple would run out of DATA and the program would end with an error message.

Storing strings as DATA

Strings, as well as numbers, can be stored and READ using DATA lines. You can also hold a mixture of both numbers and strings – the names of friends and their phone numbers, for example. If you do mix numbers and strings though, it can sometimes cause a problem, because two different types of READ statement are needed to READ numbers and strings – READ A and READ A$. For this reason it is often better to store all DATA, including numbers, as strings.

How to create a telephone list

The next program holds a personal telephone list. Names and telephone numbers are held in lines 10 to 30 and lines 40 to 90 display the program title and offer a choice of functions.

TELEPHONE LIST PROGRAM

```
10  DATA  P.ROLLINS,198,S.FREZZA,
     201,D.GARTEZ,50,F.PRIGGE,78,
     W.WILLNER,363,E.LANGHORST,86
     5
20  DATA  G.SHAPIRO,765,B.TETEWSK
     Y,32,H.NAIR,886,R.NAIMAN,44,
     W.MAVIT,478,C.CLUNE,345
30  DATA D.WEDGER,321,V.CALLAMARA
     S,544,A.BLAGG,327,K.PURPURA,
     885
40  HOME
50  VTAB 5: HTAB 8: PRINT "PERSON
     AL TELEPHONE LIST"
70  VTAB 10: HTAB 6: PRINT "COMPL
     ETE LISTING....PRESS 1"
90  VTAB 12: HTAB 6: INPUT "SELEC
     TIVE LISTING...PRESS 2 ? ";C

110  HOME
120  RESTORE

]
```

```
130  IF C = 2 THEN 210
140  VTAB 5
150  FOR P = 1 TO 16
160  READ N$,M$
170  PRINT N$,M$
180  NEXT P
190  GOTO 290
200  VTAB 14: HTAB 6
210  INPUT "WHICH NAME ? ";S$
220  PRINT : PRINT
230 R$ = S$ + " - NOT FOUND"
240  FOR P = 1 TO 16
250  READ N$,M$
260  IF N$ = S$ THEN R$ = N$ + "
      " + M$
270  NEXT P
280  PRINT R$
290  PRINT : PRINT
300  INPUT "PRESS RETURN TO CONTI
     NUE ";X$
310  GOTO 40
]
```

When you RUN this program type 1 then press RETURN to PRINT the whole telephone list; you should see a screen display like the one below:

COMPLETE TELEPHONE LIST

```
P.ROLLINS           198
S.FREZZA            201
D.GARTEZ            50
F.PRIGGE            78
W.WILLNER           363
E.LANGHORST         865
G.SHAPIRO           765
B.TETEWSKY          32
H.NAIR              886
R.NAIMAN            44
W.MAVIT             478
C.CLUNE             345
D.WEDGER            321
V.CALLAMARAS        544
A.BLAGG             327
K.PURPURA           885

PRESS RETURN TO CONTINUE
```

Alternatively, you can find the telephone number for just one name by typing 2 and pressing RETURN. After displaying the required number, the Apple returns to the selection display:

TELEPHONE LIST SELECTION DISPLAY

```
        PERSONAL TELEPHONE LIST


     COMPLETE LISTING....PRESS 1
     SELECTIVE LISTING...PRESS 2 ?
```

If you type in 2 at line 90, the program follows lines 210 to 310. You are first asked to enter the initial and name. Be careful not to type any extra spaces or punctuation in this INPUT or the Apple will not recognize your entry.

If the Apple finds that the name (S$) you typed in is the same as one of the names (N$) in the DATA statements, it will give you a new string, R$: the value of N$ plus a line of dots and the telephone number (M$). If it cannot match S$, R$ is left unchanged as "NAME NOT FOUND" (set by line 230), and this is PRINTed out at the end of the program.

Because you want to add the name, a line of dots, and the telephone number together in line 260, the telephone number has to be treated as a string variable M$, instead of a numeric variable M. If you used M the program would not work because string and numeric variables cannot be added together.

Introducing RESTORE

Line 120 uses a new command, RESTORE. This tells the Apple to go right back to the beginning of the DATA statements the next time it carries out a READ command.

Without RESTORE the program would only work correctly once, because having searched all the DATA the first time it was RUN the Apple would have reached the end of the list. RESTORE is therefore important because it instructs the Apple to return to the beginning of the DATA, enabling it all to be re-READ every time you consult the telephone list.

With a little practice at compiling a DATA bank you can store your friends' birthdays, list bills and payments, or index your tape library.

INTRODUCING SHAPES

Now that you're familiar with the commands for hi-res graphics, and have seen DATA and READ statements at work, you're ready to get to grips with Shapes. A Shape is a pictorial design that can be described to the computer using "coded numbers" which represent a series of "movements". The numbers are contained in one or more lines of program called a "Shape table". A Shape can be any design – for example, a space invader, a bicycle or a bird. Once a Shape has been defined, it can be SAVEd on disk and recalled every time you want to use it in a program.

How to define a simple Shape

Shapes are only used in hi-res graphics and writing a program to define a Shape is quite a complicated procedure. But without Shapes it would be virtually impossible to draw any complex design on the Apple – because you would have to HPLOT every single point. Here is a very simple Shape program to draw a square box:

SHAPE DEFINITION PROGRAM

```
10  DATA  1,0,4,0,88,5,6,6,7,7,4,
     4,5,0
20 N = 13
30  FOR A = 768 TO 768 + N
40  READ M: POKE A,M
50  NEXT A
60  POKE 232,0: POKE 233,3
70  HGR : HCOLOR= 3: SCALE= 10: ROT=
     0
80  DRAW 1 AT 140,80

]
```

Line 10 holds the "Shape table" that defines the box. Line 20 tells the Apple how many instructions there are in the table.

The last 0 in the DATA is only there to tell the Apple that it has reached the end of the Shape definition. It is particularly important when several Shapes are defined in one table because it indicates to the computer that one Shape is now complete, and the instructions for the next are about to start.

The FOR...NEXT loop in lines 30 to 50 tells the Apple where to store this Shape table in its memory – this one will begin in the byte of memory labeled 768. The number which labels a byte is called its "address". As you learn more about your Apple you will recognize address 768 as the start of a free area of memory large enough to hold this Shape table.

Line 60 holds four special numbers that tell the computer where to find the Shape table that has been stored in memory when you give a DRAW command. Line 70 switches the computer into hi-res graphics, then sets the COLOR to 3 (white), the SCALE to 10 and the angle of ROTation to 0.

Finally, line 80 instructs the computer to DRAW Shape number 1 – the first, and in this case the only, Shape in the table – at the hi-res screen co-ordinates 140,80. If you RUN this program, a small white box will appear on the screen.

See if you can work out the relationship between the box that this program DRAWs, the SHAPE MOVEMENT INSTRUCTIONS below, and instructions five to thirteen in line 10 of the program. Ignore the first four numbers for the moment; they provide the Apple with information that enables it to use the Shape table.

SHAPE MOVEMENT INSTRUCTIONS

A Shape is defined to the computer with a series of numbers that represent movements. The computer can either be instructed to move in a given direction and then plot a point, or it can be instructed to move without plotting:

Direction	Move only	Move and plot
Up	88	4
Right	89	5
Down	90	6
Left	91	7

Don't worry if you haven't completely understood this program, as you will learn how to define your own Shapes later.

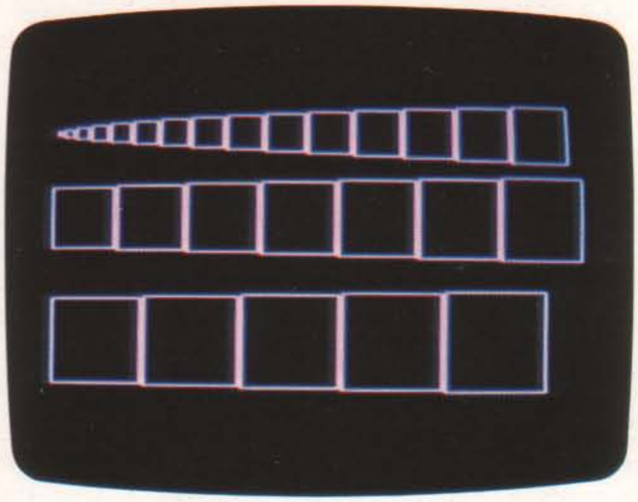
How to SCALE and ROTate a Shape

The next program demonstrates the effects of SCALE. Notice that it does not begin by re-listing the Shape table for the box. Once a Shape has been defined, in a SHAPE DEFINITION PROGRAM, it is held in memory until you define another table or switch off. All that is needed in any subsequent program is the command DRAW followed by the position of the Shape in the table: 1,2,3 or 4 and so on.

The crucial command in this program is in line 40. It enlarges the box on the screen using SCALEs of 1 to 27. You can SCALE any Shape between one and 255 times as long as it will still fit on the screen. Line 30 sets the first co-ordinate for the first box, and line 60 determines that enough space is left between the boxes as they are DRAWn to ascending SCALEs on the screen. Line 70 instructs the computer to return to the left of the screen once it has reached the right side (where X+S>279).

SCALING SHAPES

```
10  HGR
20  HCOLOR= 2: ROT= 0
30 X = 3:Y = 20
40  FOR S = 1 TO 27
50  SCALE= S: DRAW 1 AT X,Y
60 X = X + S * 2 + 3
70  IF X + S > 279 THEN X = S + 1
    :Y = Y + S * 3
80  NEXT S

]
```

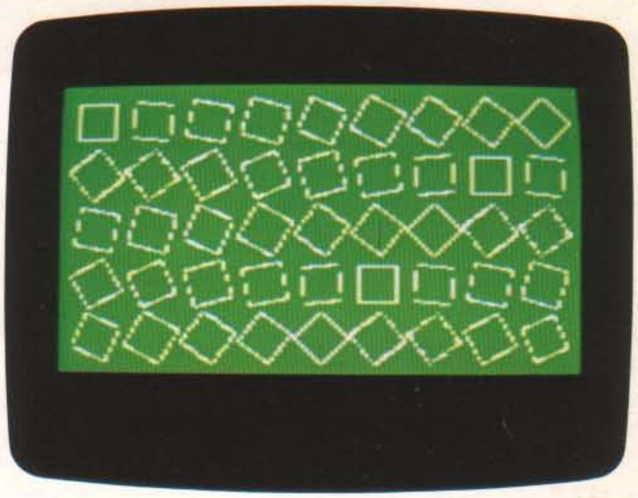


The next program uses the same box Shape and shows the effect of the command ROTate. In this program line 40 sets the first co-ordinate, line 70 determines the

ROTATING BOX PROGRAM

```
10  HGR : HCOLOR= 1
20  HPLOT 1,1: CALL 62454
30  HCOLOR= 3: SCALE= 10
40 X = 20:Y = 20
50  FOR R = 0 TO 44
60  ROT= R: DRAW 1 AT X,Y
70 X = X + 30
80  IF X > 270 THEN X = 20:Y = Y +
    30
90  NEXT R

]
```

ROTATING BOX DISPLAY

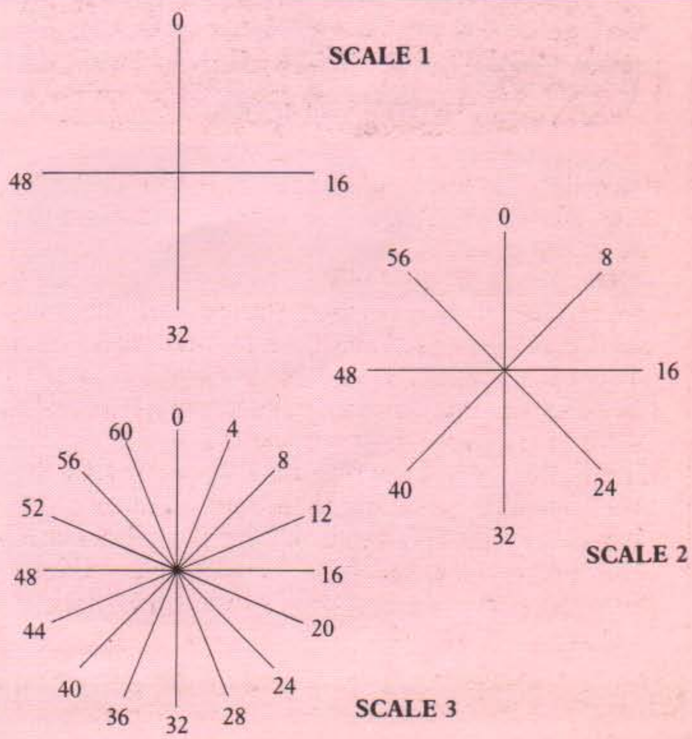


amount of space to be left between boxes and line 80 moves the "cursor" back to the left once a line has been completed. The command CALL is explained later, but here it colors the background.

But line 50 is the crucial line here: it ROTates the box between the values 0 and 44. These numbers do not refer to points of the compass they are specific to the Apple computer and are determined by the SCALE you have selected. The table, ROTATING A SHAPE, illustrates the options available with SCALEs of 1, 2 and 3.

ROTATING A SHAPE

The angles at which you can ROTate a Shape are determined by the SCALE you select: at a SCALE of 1 only four ROTation options are available but as the SCALE increases you can extend these options from 0 right up to 255. The options available at SCALEs of 1, 2 and 3 are illustrated below:



ANIMATING SHAPES

Once you have created a Shape and experimented with SCALE and ROTate, the next step is to set your Shape in motion.

The principles of animation

You already know that the principles of animation for low-res graphics are to draw an object on the screen, erase the object and then re-draw it in a new position. In hi-res graphics the method is virtually the same but simplified by the command XDRAW.

The command XDRAW is the same as DRAW, except that the colors available in this mode operate as "complementary pairs". These are the colors available:

COLORS AVAILABLE WITH XDRAW

Black (0) and White (3)
Green (1) and Violet (2)
Orange (5) and Blue (6)

This is how you use them: black and white form one pair of complementaries, so if you set the background to black and then XDRAW a Shape, the Shape will automatically be drawn in white. If you then XDRAW the same Shape again, in the same position, it will be re-drawn in black (because its immediate background is now white). This effectively erases the Shape ready for you to XDRAW it again in the next position in the animation sequence.

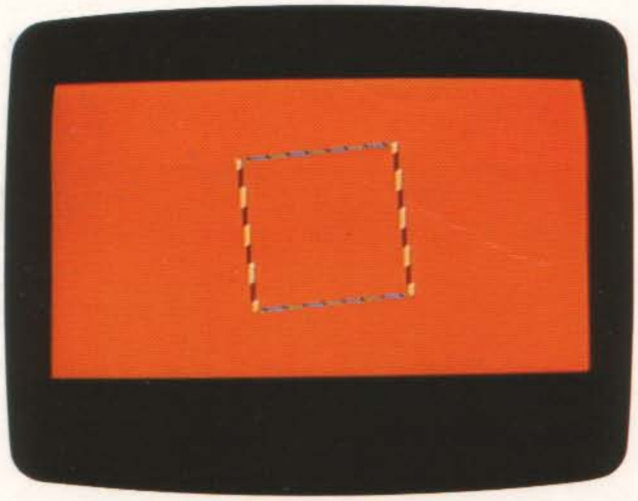
Setting the box Shape in motion

The next program uses these steps and the ROTate command to animate the box Shape from page 46. You should recognize the first few lines (10 to 70) from the SHAPE DEFINITION PROGRAM. Line 90 sets the SCALE and 100 sets the ROTation values. Then the second part of 110 XDRAWS the box. The CALL

ROTATING BOX ANIMATION

```
10  DATA  1,0,4,0,88,5,6,6,7,7,4,
    4,5,0
20 N = 13
30  FOR A = 768 TO 768 + N
40  READ M: POKE A,M
50  NEXT A
60  POKE 232,0: POKE 233,3
70  HGR : HCOLOR= 5
80  HPLOT 1,1: CALL 62454
90  SCALE= 40
100  FOR R = 0 TO 255
110  ROT= R: XDRAW 1 AT 140,80: XDRAW
     1 AT 140,80
120  NEXT R
]
```

ROTATING BOX DISPLAY



command in line 80 has already filled the screen with orange, so the first XDRAW automatically draws the box in blue – the complement of orange. The third section of line 110 then XDRAWs the box again in the same position, and as its immediate background is now blue – it re-draws the box in orange. The box therefore blends in with the background and disappears. Line 120 then calls for the next ROTation value and the process is repeated until the FOR. . . NEXT loop has XDRAWn the box at all the ROTation values set in line 100.

Designing Shapes for computer games

Try experimenting with different colors in line 70 of the ROTATING BOX ANIMATION and then type in this program. It contains a Shape table for a lunar module.

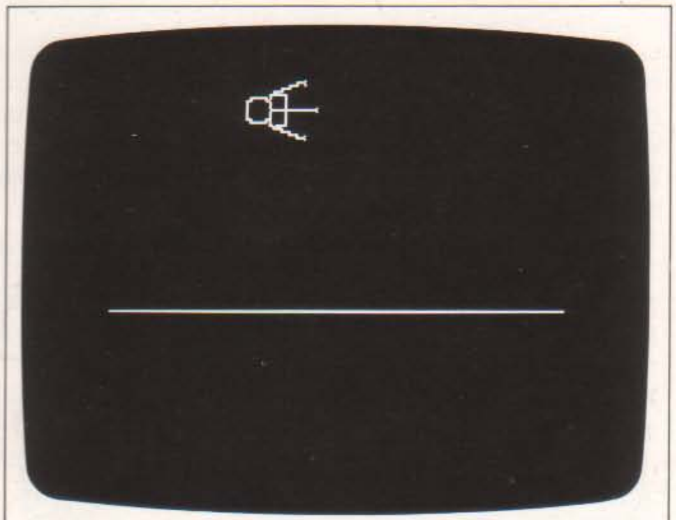
LUNAR MODULE PROGRAM

```
10  DATA 1,0,4,0,5,5,5,4,5,4,4,4,
    4,7,4,7,7,7,7,7,7,6,7,6
20  DATA  6,6,6,5,6,5,5,5,89,89,8
    9,5,5,6,6,6,6,7,7,7
30  DATA  7,7,7,7,7,7,7,4,4,4,4,5
    ,5,90,91,91,7,6,6,7
40  DATA  6,6,7,6,6,7,6,6,7,89,5,
    89,89,89,89,89,89,89,89,89
50  DATA  89,89,89,89,89,89,89,5,
    5,91,4,4,7,4,4,7,4,4,7,4
60  DATA 4,7,91,91,91,91,91,88,6,
    6,6,6,6,6,6,6,6,6,6,7,89,5
    ,0
70 N = 127: FOR A = 768 TO 768 +
   N: READ M: POKE A,M: NEXT A
80  POKE 232,0: POKE 233,3
90  HGR : HCOLOR= 3: SCALE= 2: HPLOT
    0,150 TO 279,150
100  ROT= 48: DRAW 1 AT 40,20
110  ROT= 52: DRAW 1 AT 100,25
120  ROT= 60: DRAW 1 AT 150,70
130  ROT= 64: DRAW 1 AT 170,130
]
```

Type it in, RUN it, then SAVE it on disk. Now type in these new lines to make the module move without leaving a series of after-images:

ANIMATED LUNAR MODULE

```
100 A1 =  - 0.055:A2 = 0.05:V1 =
     5:V2 = 0
110 R = 48:X = 20:Y = 20
120  FOR T = 1 TO 80
130  ROT= R: XDRAW 1 AT X,Y
140 OX = X:OY = Y
150  IF T = 50 THEN A2 =  - 0.05
160  IF T = 15 THEN R = 52
170  IF T = 30 THEN R = 60
180  IF T = 50 THEN R = 64
190 X = X + V1:Y = Y + V2:V1 = V1
     + A1:V2 = V2 + A2
200  XDRAW 1 AT OX,OY
210  NEXT T
220  DRAW 1 AT X,Y

]
```



The new line 100 sets initial values for the ship's velocity (V1 and V2) and acceleration (A1 and A2) in both horizontal and vertical directions. The FOR. . . NEXT loop in lines 120 to 210 lowers the module onto the moon's surface. Line 130 sets the ROTation value to the numeric variable R and XDRAWs the module at the co-ordinates X,Y. Line 140 saves these co-ordinates in the variables OX,OY. The IF. . .THEN statements (lines 150 to 180) set the value of the variable R depending on how far the descent has progressed. Line 190 calculates the new horizontal and vertical speeds and the new co-ordinates for X,Y. Line 200 then XDRAWs the module again, at the old X and Y co-ordinates. This erases the module and prepares the screen for the next drawing in the animation sequence. Once the sequence is complete, the DRAW command in line 220 displays the motionless ship on the surface of the moon. Now try this program:

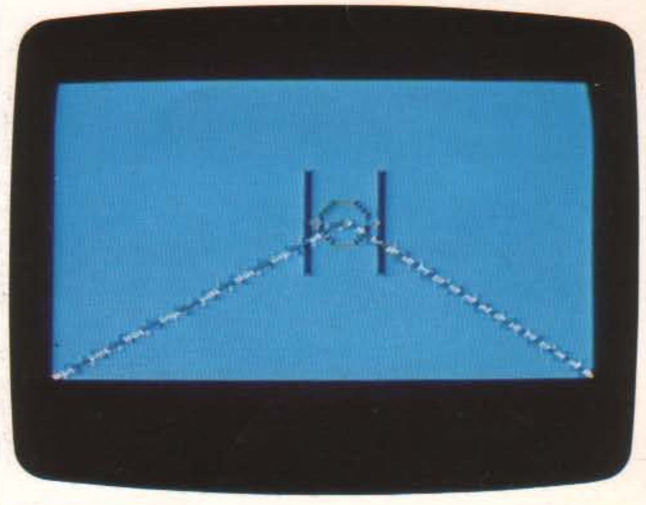
LASER ATTACK PROGRAM

```
10  DATA  1,1,1,0,1,4,1,0
20  DATA  5,88,3,7,1,6,1,7,1,6,1,
     7,1,6,1,7,2,6,3,7,13,4,1,7
30  DATA  27,6,1,5,13,4,3,5,2,6,1
     ,5,1,6,1,5,1,6,1,5,1,6,6,5
40  DATA  1,4,1,5,1,4,1,5,1,4,1,5
     ,2,4,3,5,13,6,1,5,27,4,1,7
50  DATA  13,6,3,7,2,4,1,7,1,4,1,
     7,1,4,1,7,1,4,3,7,1,0,1,-1
60 A = 768
70  READ N,M: IF M =  - 1 THEN  GOTO
     100
80  FOR X = 1 TO N: POKE A,M:A =
     A + 1: NEXT X
90  GOTO 70
100  POKE 232,0: POKE 233,3
110  HGR : SCALE= 2: ROT= 128
120  HCOLOR= 6:
130  HPLOT 0,0: CALL 62454
140 X = 140:Y = 80:R = 64

]
```

```
150  ROT= R: XDRAW 1 AT X,Y:OX =
     X:OY = Y
160 X = X + ( RND (1) - 0.5) * 20
     :Y = Y + ( RND (1) - 0.5) *
     20:R = R + ( RND (1) - 0.5) *
     2
170  IF X < 20 THEN X = 20
180  IF X > 259 THEN X = 259
190  IF Y < 20 THEN Y = 20
200  IF Y > 140 THEN Y = 140
210  IF  RND (1) < 0.8 THEN  GOTO
     240
220  HCOLOR= 1: HPLOT 0,159 TO OX
     ,OY TO 279,159
230  FOR I = 1 TO 10:J =  PEEK ( -
     16336): NEXT I
240  XDRAW 1 AT OX,OY
250  HCOLOR= 6: HPLOT 0,159 TO OX
     ,OY TO 279,159
260  GOTO 150

]
```

If you RUN this program you will see a laser firing at a spaceship. The spaceship and the laser have been programmed to move and fire at RaNDom:

LASER ATTACK DISPLAY



HOW TO WRITE A SHAPE TABLE

Now that you've seen what can be done with Shapes, all that remains is to learn how to define your own.

SHAPE TABLE CONSTRUCTION PROGRAM

```
10  HIMEM: 30208
20 SH = 31233
30  POKE 232,SH -  INT (SH / 256) * 256
40  POKE 233, INT (SH / 256)
50  INPUT "NUMBER OF SHAPES ? ";NS
60  POKE SH,NS:SH = SH + 2
70 A = SH + NS * 2
80  FOR N = 1 TO NS
90 S = A - 31233
100  HGR : HCOLOR= 3
110  FOR Y = 0 TO 160 STEP 5
120  FOR X = 0 TO 279 STEP 5
130  HPLOT X,Y
140  NEXT X
150  NEXT Y
160 X = 140:Y = 80
170  HPLOT X - 1,Y + 1: HPLOT X + 1,Y - 1: HPLOT X + 1,Y + 1: HPLOT X - 1,Y - 1

]
```

```
180  INPUT DR$
190  HCOLOR= 0: HPLOT X - 1,Y + 1: HPLOT X + 1,Y - 1: HPLOT X + 1,Y + 1: HPLOT X - 1,Y - 1: HCOLOR= 3
200 OX = X:OY = Y:M = 0
210  IF DR$ = "I" THEN M = 4:Y = Y - 5
220  IF DR$ = "M" THEN M = 6:Y = Y + 5
230  IF DR$ = "J" THEN M = 7:X = X - 5
240  IF DR$ = "K" THEN M = 5:X = X + 5
250  IF DR$ = "E" THEN M = 88:Y = Y - 5
260  IF DR$ = "X" THEN M = 90:Y = Y + 5
270  IF DR$ = "S" THEN M = 91:X = X - 5

]
```

```
280  IF DR$ = "D" THEN M = 89:X = X + 5
290  IF DR$ = "" THEN 340
300  IF M = 0 THEN  GOTO 170
310  IF M < = 7 THEN  HPLOT OX,OY TO X,Y
320  POKE A,M:A = A + 1
330  GOTO 170
340  POKE A,0:A = A + 1
350  POKE SH,S -  INT (S / 256) * 256:SH = SH + 1
360  POKE SH, INT (S / 256):SH = SH + 1
370  NEXT N
380  TEXT : HOME
390  FOR I = 31233 TO A - 1
400  PRINT  PEEK (I);",";
410  NEXT I

]
```

The first stage of the process involves a number of calculations. You could do them on paper but this is the very thing that the computer is good at, so the adjacent program has been designed to do most of the hard work for you.

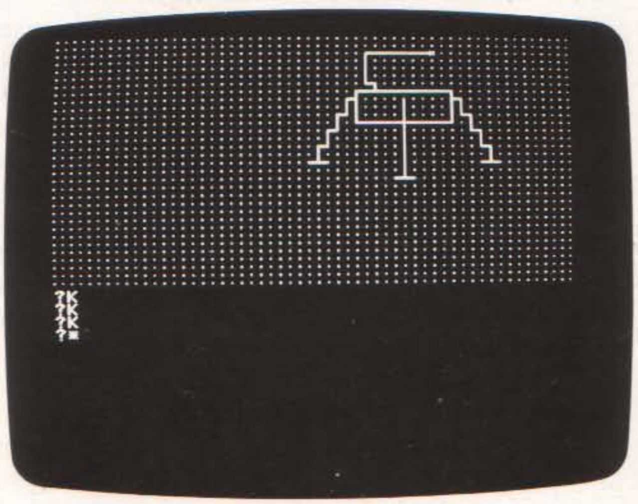
Once you've typed it in, SAVE it on disk and then RUN it. It will first ask how many Shapes you want to create. For now, just type 1; a grid of dots will appear on the screen. Use this grid as if it were a piece of graph paper to draw the design you want to reproduce on the screen. To move the cursor over the grid press the following keys:

HOW TO DESIGN ON THE SHAPE GRID

To MOVE and PLOT		To MOVE without PLOTting	
UP "I"	DOWN "M"	UP "E"	DOWN "X"
LEFT "J"	RIGHT "K"	LEFT "S"	RIGHT "D"

After each keypress, press RETURN. And when you've finished your design, press RETURN without first pressing a character to indicate that it is complete. The screen will look like this while your design is in progress:

CONSTRUCTING A SHAPE



Once it is complete the Apple will display a list of numbers. This is the Shape table for your design; write down a copy of the table for reference later.

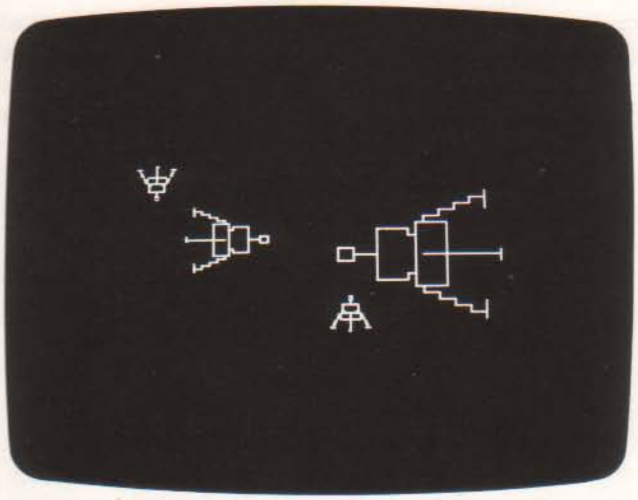
How to test your Shape

Before you go to the trouble of writing the final program for this Shape, you can now test your design by writing a simple program to see if it will achieve the desired effect. To DRAW the Shape once on the screen, try a program like this:

```
10 HGR: HCOLOR=3
20 SCALE=4: ROT=16
30 DRAW 1 AT 100,90
```

If you would like to DRAW the same Shape several times on the same screen there is no need to repeat line 10 again. Just give a SCALE, ROTation value, DRAW command and co-ordinates for every time you

TESTING A SHAPE



want the Shape repeated. The above screen illustrates the type of effect that can be achieved in this way.

Writing a Shape table into a program

When you're happy with the Shape table you've created the next step is to write it into a "Shape Definition Program". You will then have the Shape table in a form that can be SAVEd on disk and recalled when you're next writing a program that needs a design like this.

If you turn back to the SHAPE DEFINITION PROGRAM that defined a box on page 46, you should find that you can now understand and imitate lines 10,20,70 and 80 in your own program. But what of the rest? Here they are again to jog your memory:

```
30 FOR A=768 TO 768+N
40 READ M:POKE A,M
50 NEXT A
60 POKE 232, 0:POKE 233,3
```

The loop at lines 30 to 50 stores the Shape table in the Apple's memory. Line 30 selects the area of memory that it will be stored in. This one will begin at address 768. Lines 40 and 50 then instruct the Apple to POKE each of the movements (M) into memory addresses (A) between 768 and 768+N.

It will help you to understand this process if you compare it with the method the Apple uses to store variables. You can define a variable, for example Z=25, simply by typing LET Z=25. The Apple will then decide where to store this information in RAM and leave "notes" for itself so that it knows where to find the variable again when you recall it with the command PRINT Z. But the Apple cannot store Shapes in the same way. You have to tell it where to store the Shape table and also leave instructions so that the computer can find it again when you give a command to recall it.

This is why programs to define Shapes use the POKE command. The first appearance of POKE is in line 40. After the computer has READ the list of movements (M), which are given in line 10, it POKEs (which in this case means "stores") the first movement in A – the first available byte of memory (address 768). This process is then repeated until all the movements have been stored in memory.

POKE reappears in line 60. Addresses 232 and 233 are special bytes of memory that are always reserved to hold the information that tells the Apple where to re-find Shapes it has stored. In the SHAPE DEFINITION PROGRAM for the box, these addresses were POKEd with the numbers 0 and 3. They were calculated using the formulas below and tell the Apple that the Shape table begins in address 768. The numbers you have to POKE into 232 and 233 change according to the area of memory you select to store your Shape, but they can always be worked out with these formulas:

```
PRINT# – INT (#/256) * 256
PRINT INT (#/256)
```

When you have selected an area of memory to store your Shape, simply replace the # in these formulas with the first of the addresses that are available, and calculate the correct values on your Apple. Try it now with 768: you should get the answers 0 and 3.

You should now be able to write a Shape definition program for your own Shape by imitating the box definition program. You can then begin to manipulate it using SCALE, ROT and DRAW.

How to select memory to store a Shape

You may have noticed that all the Shapes you've used so far have been stored in the same section of the Apple's memory – address 768 onwards. But this is the beginning of just a small empty area, and if you wanted to store a long Shape table you could run into problems. So there is a command to reserve a larger area of memory; it is called HIMEM. The SHAPE TABLE CONSTRUCTION PROGRAM opposite contains HIMEM: 30208 in the first line. This sets the HIghest MEMory address that the Apple can use to store the subsequent program lines and variables at 30208. If you then add 1024 (one kilobyte) to this number, the number you arrive at (31233 in this case) will be the first available address into which you can POKE your Shape.

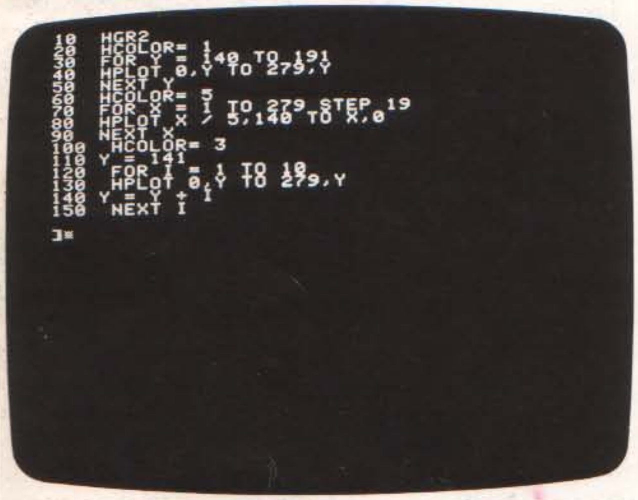
ADVANCED GRAPHICS TECHNIQUES

Once you've mastered Shape tables and hi-res color graphics, you can bring all the graphics, Shape and animation commands together in one program. The example on these two pages produces a complex picture; but if you work through the listing carefully, you should be able to write a similar program yourself.

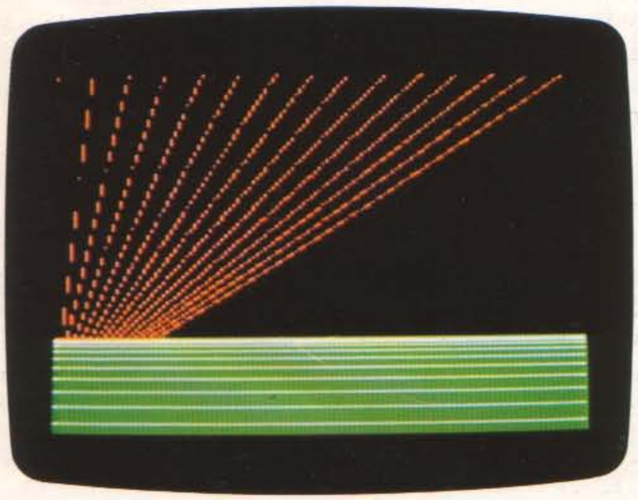
Creating a landscape

If you introduce a number of areas of color onto the screen, allowing some of them to overlap, and then HPLOT lines over certain areas, you can produce some interesting effects. The next program illustrates some of these techniques:

BASIC LANDSCAPE PROGRAM



```
10  HGR2
20  HCOLOR= 1
30  FOR Y = 140 TO 191
40  HPLOT 0,Y TO 279,Y
50  NEXT Y
60  HCOLOR= 5
70  FOR X = 1 TO 279 STEP 19
80  HPLOT X / 5,140 TO X,0
90  NEXT X
100  HCOLOR= 3
110 Y = 141
120  FOR I = 1 TO 10
130  HPLOT 0,Y TO 279,Y
140 Y = Y + I
150  NEXT I
]
```

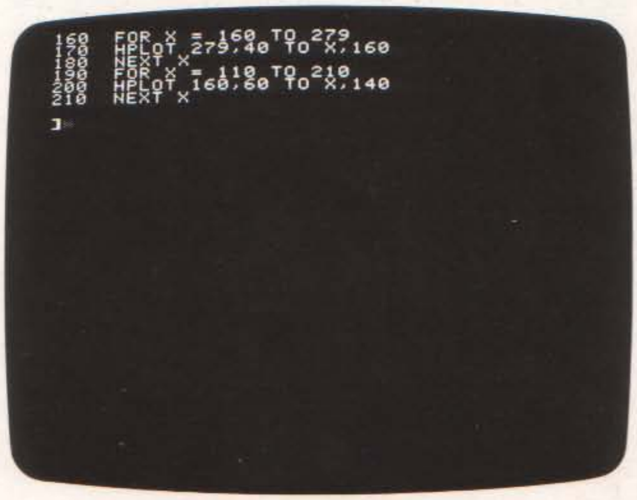


The first thing you'll notice about this program is that line 10 uses HGR2, instead of the usual HGR. In fact the Apple has two hi-res screens and the command HGR2 allows you to DRAW and HPLOT on the second screen. HGR2 functions in the same way as HGR except that it does not have four lines of text at the bottom. The y co-ordinates of HGR2 go right from 0 at the top of the screen to 191 at the bottom.

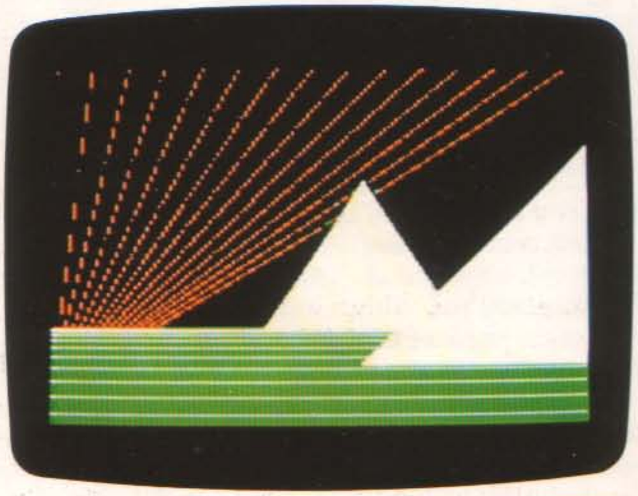
The program draws a green foreground at lines 20 to 50. Next, at lines 60 to 90, it HPLOTS orange lines that radiate from a point below the horizon, making the display look like a sunset. Notice the new command STEP, which makes the value of X increase in steps of 19 instead of 1. Finally, the perspective lines appear, at a spacing which increases the further they are from the horizon. To do this, you need to make the spacing between the y co-ordinates grow larger as you move away from the horizon. Line 140 is the crucial one; it makes Y increase by a progressively larger amount every time the loop is carried out.

When you have keyed in and RUN the first section of the program, you can improve it by HPLOTting and filling in some objects in the foreground:

ADDITIONAL LINES



```
160  FOR X = 160 TO 279
170  HPLOT 279,40 TO X,160
180  NEXT X
190  FOR X = 110 TO 210
200  HPLOT 160,60 TO X,140
210  NEXT X
]
```



Once you have RUN this, try going back over the program and altering some of the lines. The best part of graphics programs is experimenting!

Adding a Shape table

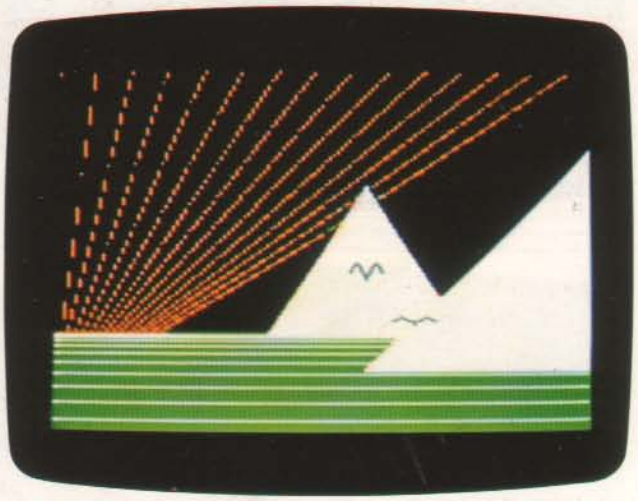
Now you can design a Shape to add to your program and use it to introduce some animation to the picture. First key in these extra lines at the end of the program:

ADDING A SHAPE TABLE

```
220  DATA 3,0,8,0,29,0,51,0
230  DATA 228,228,228,228,28,30,3
     0,246,118,73,73,137,100,100,
     100,12,14,14,118,54,0
240  DATA 228,63,7,88,63,88,247,1
     91,127,73,73,73,73,45,5,88,4
     5,88,117,173,45,0
250  DATA 228,28,191,23,23,54,30,
     54,14,118,73,9,88,88,88,88,8
     8,88,88,88,88,0,5,88,173,21,
     21,54,14,54,30,54,0
260 N = 83
270  FOR I = 768 TO 768 + N
280  READ X: POKE I,X
290  NEXT I
300  POKE 232,0: POKE 233,3
310  SCALE= 1: ROT= 0:X = 160:Y =
     110
320  XDRAW 1 AT X,Y
330  XDRAW 2 AT X + 25,Y + 25

]
```

These lines define a Shape table containing three Shapes. When you RUN the program now, two of the Shapes are drawn and show two birds in mid-flight. The third Shape is used at the next stage.

BIRD SHAPE DISPLAY



Animating the Shapes

The next phase of the program is to animate the birds so that they appear to be flying towards you. So far, your animated displays have always conveyed the impression of motion across or up and down the screen. But in order to make the bird's flight look realistic you will need to use a similar, but slightly different technique.

Animated cartoons for television and the movies are produced by photographing a series of drawings, each depicting a slightly different stage of motion. You will use a similar method here, except that you will XDRAW a Shape, XDRAW it again to erase it, and then repeat the process with the next Shape in the flying sequence.

The FOR. . .NEXT loop starting at line 320 steps through each of the bird Shapes in turn. To add realism, the two birds flap their wings out of step. This is done by subtracting the variable S (Shape number) from 4 at lines 340 and 370. Line 350 provides a short delay between XDRAWS and the bird Shapes are erased at lines 360 and 370.

ANIMATED BIRD EXTRA LINES

```
320  FOR S = 1 TO 3
330  XDRAW S AT X,Y
340  XDRAW 4 - S AT X + 50,Y + 25
350  FOR I = 1 TO 100: NEXT I
360  XDRAW S AT X,Y
370  XDRAW 4 - S AT X + 50,Y + 25
380  NEXT S
390 X = X + ( RND (1) - .5) * 8
400 Y = Y + ( RND (1) - .5) * 8
410  GOTO 320

]
```

To make the display even more realistic, the two birds in the final display move up and down and from side to side at random. This effect is achieved by the two RND(1) statements in lines 390 and 400. The last line of the program (410) then puts the program into an endless loop.

You have probably noticed that the program does not check to see if the random movement takes the birds off the edge of the screen. So if you left the program to RUN for some time this could happen, causing the program to stop and give an error message. However it would be fairly simple to insert extra lines to monitor the position of the birds and take corrective action if they stray too near the edges of the screen. These are the lines that could be inserted to test the x co-ordinates of the birds:

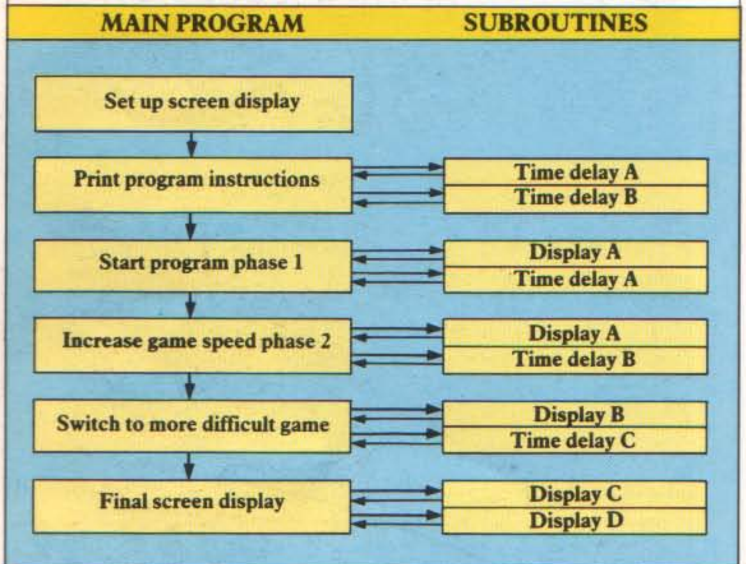
```
382 XI=(RND(1)−0.5)*8
384 IF X+XI < 0 OR X+XI >279 THEN
    GOTO 382
390 X=X+XI
```

Using the same principles amend the program to test the y co-ordinates as well.

WRITING SUBROUTINES

You will often want to repeat a few lines of a program again and again to carry out the same calculation or to display the same group of characters on the screen. To avoid writing out the same lines time after time (and using up too much of the computer's memory) you could branch off to frequently used sections of the program with GOTO. However, using GOTO for this is frowned on by many programmers because it can quickly turn your program into untidy mazes.

The easiest program to analyze and debug is one written methodically in blocks or "modules", each of which you can test independently of the others if problems arise. If you look up the listing of a good games program in a magazine, for example, you will find that it works something like this:



How to use a subroutine

As you can see from this chart, the way around this problem is to use subroutines. This is the name given to frequently used modules of programs which can be recalled by their line numbers at any time, using the GOSUB command.

GOSUB tells the computer to branch off from the main program in the same way as GOTO. But when it's finished the subroutine, it will return to exactly the point in the program from which it branched to the subroutine. The command is used like this:

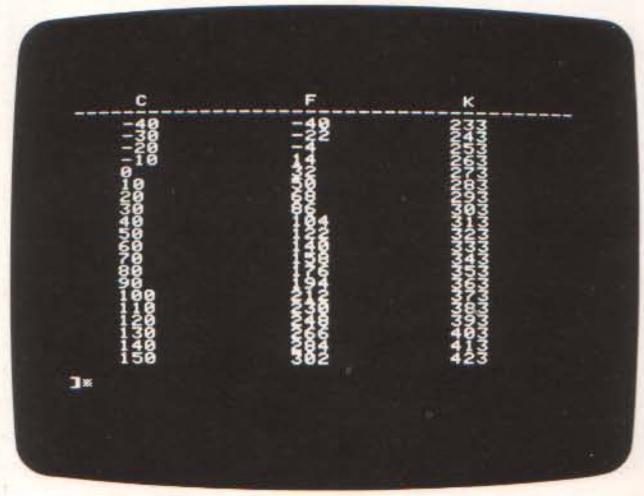
```
50 GOSUB 500
```

Following this command, a program would RUN normally until it reached line 50, and then follow the instruction to GO to the SUBroutine at line 500. After it had performed the statements in the subroutine it would return to the line after line 50 in the main program (usually line 60). Subroutines always end with RETURN. Without RETURN, your Apple would follow all of the statements from line 500 to the end of the program.

You can use GOSUB in almost any program where the Apple has to repeat an operation. The next program produces a temperature-conversion chart using Centigrade, Fahrenheit and Kelvin. The subroutine at line 80 makes the Apple PRINT out a line of the table, produce a beep on the speaker and then RETURN to line 60. The command END at line 70 stops the program from carrying on into the subroutine when the FOR. . .NEXT loop has finished. If you miss out END, the computer will reach the RETURN command at line 100 and produce an error message because it has been told to RETURN without a previous GOSUB instruction.

The subroutine in the listing below is inside a loop so that it is "called" several times.

TEMPERATURE CONVERSION PROGRAM



In the TEMPERATURE CONVERSION PROGRAM the subroutine is not actually saving any space. However, if you extended the program to carry out other functions, the subroutine could save both space and memory, and clarify the program.

Setting up "menu" displays with GOSUB

Many programs start with a "menu" and ask you to select one of the options; the choice is often programmed using GOSUB. When you enter your selection, the program goes to the appropriate subroutine and sets up the display you have chosen. Here is a simple listing to do just that:

MENU PROGRAM

```
10  TEXT : HOME
20  INPUT "DISPLAY 1 OR 2 ? ";A
30  HGR2
40  IF A = 1 THEN B = 1:G = 2:S =
     3: GOSUB 200
50  IF A = 2 THEN B = 1:G = 5:S =
     0: GOSUB 200
60  END
200  HCOLOR= B
210  HPLOT 0,0: CALL 62454
220  HCOLOR= G
230  FOR Y = 130 TO 191
240  HPLOT 0,Y TO 279,Y
250  NEXT Y
260  HCOLOR= S
270  FOR Y = 130 TO 191 STEP 5
280  HPLOT 0,Y TO 279,Y
290  NEXT Y
300  RETURN

]
```

This program can set up two simple displays. One of them is illustrated below. The colors in each display are produced by a subroutine – your INPUT following line 20 determines which colors are displayed. If you were using this subroutine in a real games program, you could GOSUB to this subroutine often with different values for the variables B, G and S.

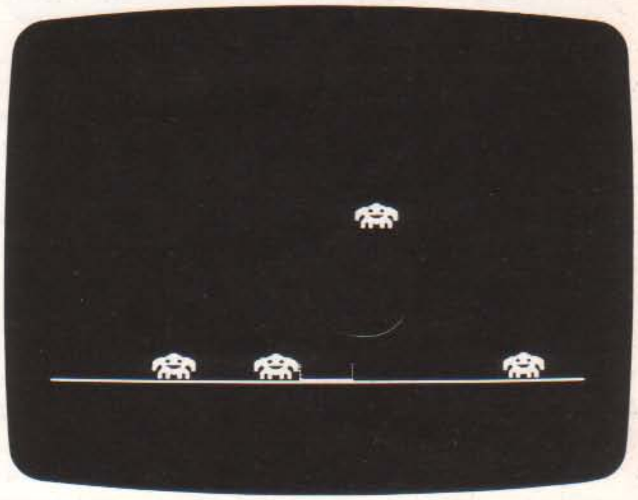
MENU PROGRAM DISPLAY

The next program HPLOTs a trap and then XDRAWs aliens falling from random points at the top of the screen. If an alien falls into the trap, line 210 directs the computer to go to the subroutine at line 230. This erases the aliens and restarts the program:

GOSUB ANIMATION PROGRAM

```
10  DATA 1,0,4,0,45,117,57,63,63
     ,63,222,27,45,45,45,45,45,45
     ,45,117,57,63,63,223,63
20  DATA 255,59,63,63,30,45,45,4
     5,77,45,109,41,45,45,53,63,6
     3,63,63,63,63,63,63,63,63
30  DATA 55,45,77,109,73,73,41,77
     ,45,62,223,59,63,223,27,63,2
     55,27,55,109,41,45,45,45
40  DATA 45,45,45,77,245,27,63,63
     ,63,63,63,63,63,55,109,41,13
     ,109,109,41,62,223,255,219
50  DATA 255,59,46,77,109,73,109,
     41,5,0
60 N = 107
70  FOR A = 768 TO 768 + N
80  READ M: POKE A,M
90  NEXT A
100  POKE 232,0: POKE 233,3
110  HGR2 : HCOLOR= 3
120  HPLOT 0,189 TO 279,189

]
```

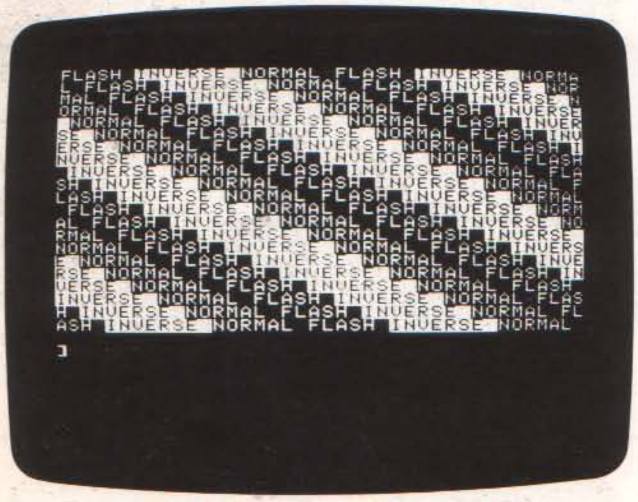
```
130  HPLOT 128,180 TO 128,188 TO
      155,188 TO 155,180
140 X =  INT ( RND (1) * 10) * 26
     + 10
150  FOR Y = 0 TO 170 STEP 5
160  XDRAW 1 AT X,Y
170  FOR T = 1 TO 5:Z =  PEEK ( -
     16336): NEXT T
180  XDRAW 1 AT X,Y
190  NEXT Y
200  DRAW 1 AT X,Y
210  IF X = 140 THEN  GOSUB 230: GOTO
     110
220  GOTO 140
230  HCOLOR= 1
240  FOR Y = 191 TO 165 STEP  - 1
250  HPLOT 0,Y TO 279,Y:Z =  PEEK
     ( - 16336)
260  NEXT Y
270  RETURN

]
```



SPECIAL SCREEN TECHNIQUES

The Apple has two commands for highlighting characters displayed on the TEXT screen. Using these commands you can display selected PRINT commands in flashing or inverse (black on a white background) characters. This program turns on the flashing mode in line 30, with the BASIC command FLASH. All characters PRINTed after this will FLASH on the screen. The command at line 50 turns on the inverse display mode with the command INVERSE. This command works in the same way as FLASH. The normal mode of operation is restored on line 70 with the command NORMAL. This statement cancels whichever of the special modes – FLASH or INVERSE – is in operation. If neither of these modes is set, then NORMAL will have no effect.

FLASH AND INVERSE PROGRAM

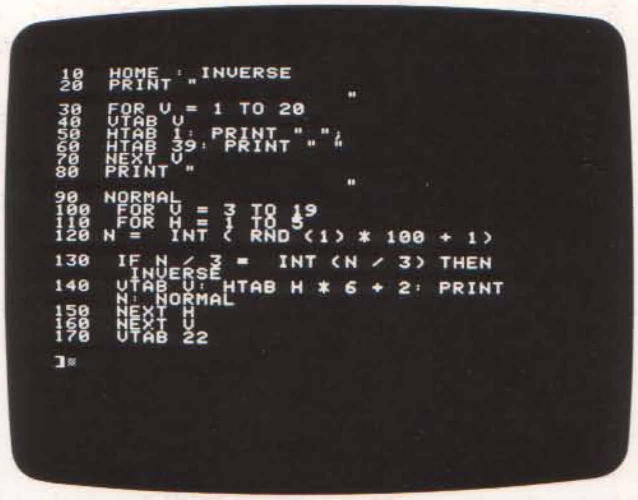


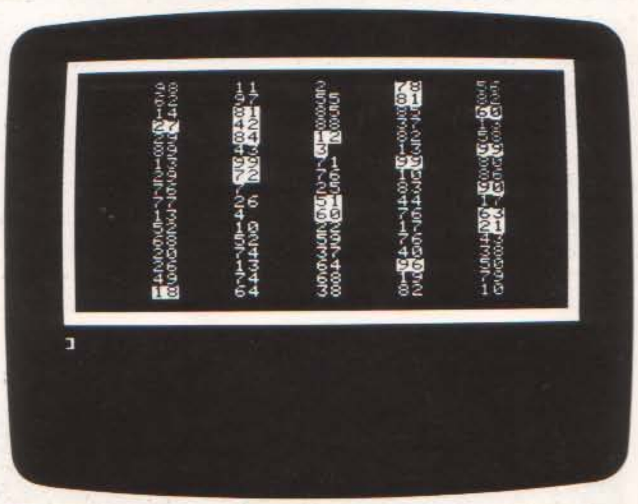
FLASH, INVERSE and NORMAL are useful for drawing attention to information displayed on the screen. For example, a FLASHing display could warn of a dangerous situation such as "Low on Fuel".

Enhancing a text display

In the next example, random numbers are generated using the RND(1) command. A check is made at line 130 to see if the number can be divided exactly by 3. If it can, the number is displayed in INVERSE.

DIVIDING BY THREE





Lines 10 to 80 PRINT a white border round the outside of the screen. The program does this by PRINTing spaces in INVERSE mode – an INVERSE space shows as a solid white block on the screen.

How to create "flicker-free" animation

You will have noticed that when you animate Shapes on the screen the display sometimes flickers. This flickering becomes more noticeable as you SCALE Shapes to larger sizes because it takes longer and longer for the Apple to XDRAW the Shape on the screen.

Fortunately, there is a way around this problem. Key in this program and then RUN it:

CITY APPROACH PROGRAM

```
10  DATA  1,0,4,0,36,36,36,36,44,
     46,46,46,46,54,54,45,45,222,
     3,32,36,36,36,44,45
20  DATA  45,54,54,54,54,6,88,56,
     63,63,40,45,45,56,63,63,40,4
     5,45,56,63,63,40,45,45
30  DATA  56,63,127,73,145,146,37
     ,36,36,44,44,44,44,53,53,53,
     53,54,54,54,246,35,36,36,36
40  DATA  36,220,27,54,54,54,54,1
     18,36,36,36,36,100,49,54,54,
     54,54,14,56,63,7,40,45,5
50  DATA 88,56,63,7,40,45,5,88,56
     ,63,7,40,45,5,88,56,63,7,0
60 N = 118: FOR A = 768 TO 768 +
     N: READ M: POKE A,M: NEXT A
70  POKE 232,0: POKE 233,3
80  HGR : HCOLOR= 3
90  HPLOT 0,160 TO 279,160
100  FOR S = 1 TO 13: SCALE= S:X =
     140 - S * 10: XDRAW 1 AT X,1
     60: XDRAW 1 AT X,160: NEXT S

]
```

You will notice that the flickering becomes quite severe and spoils the effect of the "aircraft approach" as the SCALE increases.

As you know, the Apple has two hi-res screens, selected by HGR and HGR2 and it is possible to program the computer to alternate between these screens and so eliminate the flicker. If you XDRAW on screen one whilst displaying screen two, and then display the new XDRAWing on screen one while the computer re-XDRAWs on screen two, the flicker will disappear.

A good way to write this program is to set up subroutines – one to draw on screen one whilst displaying screen two, the other to do the reverse. If you key these extra lines into the program you will see this "smooth" animation in action:

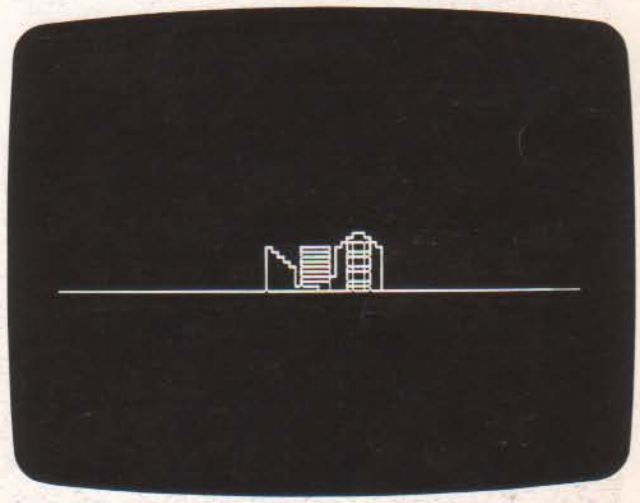
SMOOTH ANIMATION LINES

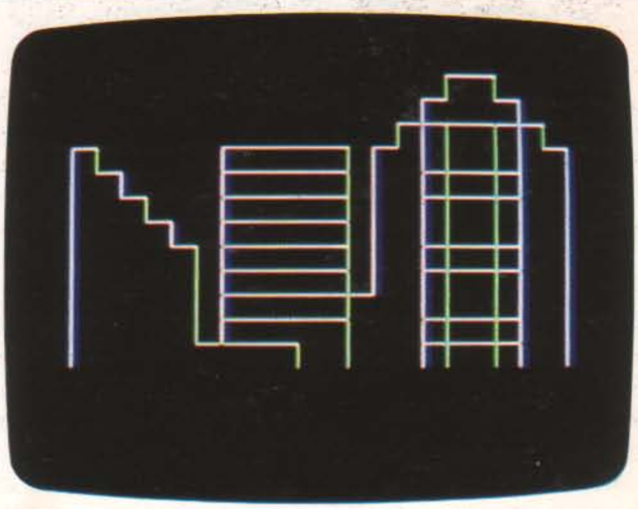
```
100  HGR2
110  HPLOT 0,160 TO 279,160
120 S1 = 1: GOSUB 500:S2 = 1: GOSUB
     600:HG = 2
150  FOR S = 2 TO 13
160  IF HG = 1 THEN  GOSUB 500:S1
     = S: GOSUB 500:HG = 2
170  IF HG = 2 THEN  GOSUB 600:S2
     = S: GOSUB 600:HG = 1
200  NEXT S
210  END
500  POKE  - 16299,0
510  POKE 230,32
520  SCALE= S1:X = 140 - S1 * 10
530  XDRAW 1 AT X,160
540  RETURN
600  POKE  - 16300,0
610  POKE 230,64
620  SCALE= S2:X = 140 - S2 * 10
630  XDRAW 1 AT X,160
640  RETURN

]
```

The subroutine at line 500 POKEs a special memory address which makes the Apple display the second hi-res screen. At line 510 it POKEs a value into yet another special address which makes all HPLOTs, DRAWs and XDRAWs take place on graphics screen one. Line 520 sets the SCALE and x co-ordinate to the value of variables S1 and X. These are reserved to store these two values for graphics screen one. Finally the subroutine XDRAWS the Shape and RETURNs. The subroutine at line 600 does the same thing, except that it displays screen one and XDRAWS on screen two.

The main program is between lines 100 and 210. First, the two subroutines are called to XDRAW the Shapes, and then the FOR. . .NEXT loop at line 150 increases the SCALE variable, S. Notice how the variable HG is used as a "switch" which the IF. . . THEN statements at lines 160 and 170 use to decide which subroutine to call. The variables S1, S2 and X are set to the correct values before calling the subroutines:

SMOOTH ANIMATION DISPLAY





PEEK, POKE AND CALL

On page 47, you saw how to use the command CALL to summon a program from the Apple's ROM; it filled the hi-res screen with color. But there are also other more useful programs in ROM which you can use with the CALL statement. All of these are "machine-code" programs. In other words, they are instructions which the Apple can obey directly – unlike BASIC which it has to "interpret". Although some programmers do write in machine code to achieve speed and compactness, machine code programs are notoriously difficult to write and debug.

Clearing the screen with CALL

Imagine a screen full of text, and your cursor somewhere in the middle. Now suppose you wanted to clear the screen from the current cursor position to the end of the screen. You could do this by writing a subroutine to PRINT space characters over all the existing text on the screen. But the result would be rather slow and you would see the cursor as it moved along over-PRINTing the text. The subroutine would also take up some space in the memory.

A better way of clearing the screen is to use a CALL command to activate one of the programs stored in ROM. Try typing CALL −958. This will clear the Apple's screen from the current cursor position to the end of the screen. A list of CALLs and their functions is given below:

TABLE OF CALLS

CALL address	Function
−936	Clears the text window
−958	Clears the text window from the current cursor position to the end
−868	Clears a line from the cursor position to the right of the text window
−992	Moves the cursor down a line
−912	Scrolls the text screen up one line
−1994	Clears the low-res graphics screen leaving four lines of text
62450	Clears hi-res graphics screen to black
62454	Clears hi-res graphics screen to the last plotted color

Introducing PEEK

Programs written in machine code cannot use variables in the way that BASIC programs do. Instead the programmer has to reserve memory addresses to store numbers and strings – as you had to with Shapes. POKE is the only way that a BASIC program can place a value at a reserved address for a machine code program to use.

The BASIC command which does the opposite of POKE is aptly named PEEK. This allows you to retrieve a value from an address and store it in a BASIC variable. Try this:

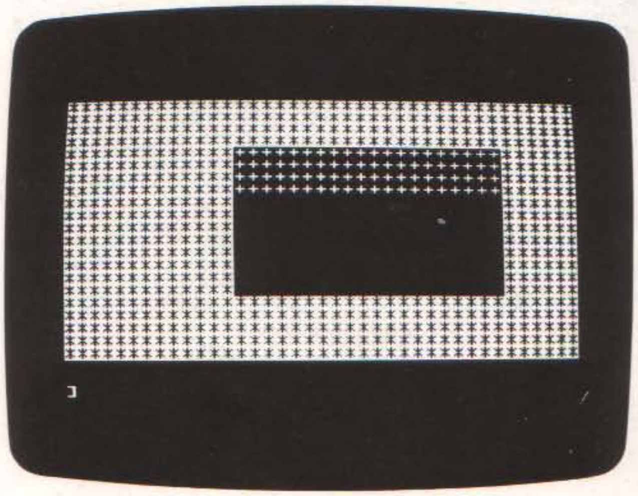
```
HTAB 20:CH=PEEK(36)
PRINT CH
```

The value stored at address 36 is the horizontal cursor column. Type in this program which illustrates the use of PEEK, POKE and CALL:

TEXT WINDOW PROGRAM

This program moves a TEXT "window" across the Apple's screen, using the FOR. . .NEXT loop beginning at line 10. First, it fills the screen with asterisks and then sets the window at lines 70 to 100. It does this by POKEing values into the addresses which are reserved for the top, bottom and left screen margins, and the display width. The CALL at line 110

TEXT WINDOW DISPLAY



clears the screen but only inside the TEXT window. The program PEEKs to find the top and bottom of the TEXT window at line 120 and then fills the window with "+" signs. CALL −958 in line 160 is then used to clear the bottom half of the window.

The table below, describes the functions of various PEEKs and POKEs:

TABLE OF PEEKS AND POKES

Address	Function
32	Left screen margin
33	Screen width
34	Top screen margin
35	Bottom screen margin
36	Cursor horizontal position
37	Cursor vertical position

Making music

As well as CALLing machine-code programs that are already in the Apple's ROM, you can POKE your own machine-code program into RAM and then CALL it. Unlike a lot of personal computers, the Apple does not have any BASIC commands for producing sounds and music on its speaker. You can produce a sort of buzzing noise using PEEK, as you did in the LASER ATTACK PROGRAM on page 49, but here is a machine-code program to make more melodic sounds:

MACHINE-CODE MUSIC PROGRAM

```
10  DATA  160,16,173,48,192,174,0
     ,3,202,208,253,136,208,244,2
     06,1,3,208,237,96
20  FOR A = 770 TO 789
30  READ M: POKE A,M
40  NEXT A
50  FOR N = 1 TO 255
60 D = 5: GOSUB 200
70  NEXT N
80  END
200  POKE 768,N: POKE 769,D
210  CALL 770
220  RETURN

]
```

The machine-code instructions are held in the DATA at line 10. Don't worry about the details of how machine code works or what the DATA represents – the advantage of CALL is that it allows you to use machine code without having to worry about its complexities.

The machine code is POKEd into RAM by the FOR. . .NEXT loop at lines 20 to 40. The program starts at address 770 and has two reserved addresses – 768 for the pitch of the sound and 769 for the duration. The FOR. . .NEXT loop at lines 50 to 70 steps through all the pitches that can be produced. The subroutine at line 200 makes the sound, by first POKEing the pitch (variable N), then the duration (variable D) and then CALLing the machine-code program.

The first thing you'll notice when you RUN this program is that the duration of the sound gets longer as the pitch decreases. A correction factor is needed to make all sounds have an equal length. This correction could be added to the machine-code program, but it would make the program longer, and less convenient to turn into DATA. The next program shows the correction factor added in BASIC, at line 200:

ADDITIONS FOR EQUAL-NOTE DURATION

```
10  DATA  160,16,173,48,192,174,0
     ,3,202,208,253,136,208,244,2
     06,1,3,208,237,96
20  FOR A = 770 TO 789
30  READ M: POKE A,M
40  NEXT A
50 D = 1
60  FOR N = 90 TO 200
70  GOSUB 200
80  NEXT N
90  END
200 X =  INT (9.56E - 4 / (N * 5E
     - 6 + 11E - 6) * 100 + 0.5)

210  POKE 771,X
220  POKE 768,N: POKE 769,D
230  CALL 770
240  RETURN

]
```

When you RUN this program it will produce sounds of equal duration over the range of pitches set by the FOR. . .NEXT loop.

It is now a simple matter to get the Apple to play a tune. The notes and their durations can be turned into DATA and played in turn:

TUNE PROGRAM

```
10  DATA  160,16,173,48,192,174,0
     ,3,202,208,253,136,208,244,2
     06,1,3,208,237,96
20  FOR A = 770 TO 789
30  READ M: POKE A,M
40  NEXT A
50  DATA  111,4,111,4,105,4,93,4,
     93,4,105,4,111,4,125,4,141,4
     ,141,4,125,4,111,4,111,6,125
     ,2,125,8,-1,-1
60  READ N,D
70  IF N =  - 1 THEN  END
80  GOSUB 100
90  GOTO 60
100 X =  INT (9.56E - 4 / (N * 5E
     - 6 + 11E - 6) * 90 + 0.5)
110  POKE 771,X
120  POKE 768,N: POKE 769,D
130  CALL 770
140  RETURN

]
```

HINTS AND TIPS

While learning to program your Apple, you will discover, through trial and error, ways to improve your technique. However, there are ways of saving time and sorting out problems which may not be immediately obvious. So here are a few hints to help you produce well-organized, bug-free programs.

Using REM as a marker

The computer ignores lines that begin with REM – this can be useful for labeling and testing parts of a program. However, when a program gets really long it is sometimes difficult to spot the REM labels among all the other lines. So it helps if you surround REMs with stars that are easily seen:

```
260 REM ****************************
270 REM SHAPE TABLE FOR ANDROID
```

Using REM as a mask

REM can also be useful in program development. It enables you to observe what happens when certain lines are omitted from a program.

You can skip sections of a program by using GOTO or RUN followed by a line number, but this won't help if you just want to miss out a few lines in the middle. The way to deal with this problem, without deleting lines, is to insert a REM command at the beginning of each of the lines you want to skip. This will "mask" or "disable" them.

How to check for tangled loops

When a program has a number of loops it is easy to get them tangled, and if they are, the program won't produce the results you want. But there is an easy way to check whether the loops are correctly "nested":

LINKING LOOPS

```
10  GR
20  FOR C = 1 TO 15
30  FOR V = 0 TO 20
40  FOR H = 0 TO 39
50  COLOR= C: PLOT H,V
60  NEXT H
70  NEXT V
80  FOR V = 21 TO 39
90  FOR H = 0 TO 39
100  COLOR= 16 - C: PLOT H,V
110  NEXT H
120  NEXT V
130  NEXT C

]
```

Write the program down (or better still use a printer) and then use a pencil to link the beginning of every loop with its end. If none of the lines overlap, as in the program opposite, then the loops are functioning correctly. If they do, you have probably found the bug in your program.

Debugging techniques

The Apple has a large repertoire of error messages which will alert you to any incorrect lines in a program. However, a program will often RUN without any hitches, only to produce an entirely different result from the one you had in mind.

If this happens and checking loops and using REMs doesn't help, try giving each variable a single fixed value instead of allowing them to run through a number of values.

Imagine, for example, that you have a graphics program which uses the command RND(1) to produce a random-color display in a loop. If it doesn't work correctly take out RND(1), and insert a fixed value instead. Then use REM to mask out the lines that start and terminate the loop and if the result of a single RUN through is not what you predicted, the display should give you some idea where the program is going wrong. Here is the RANDOM GRAPHICS PROGRAM, from page 43, edited for testing:

PROGRAM EDITED FOR TESTING

```
10  GR
20  COLOR= 15: REM  INT ( RND (1)
     * 16)
30 V = 10: REM INT ( RND (1) * 40
    )
50 H = 20: REM  INT ( RND (1) * 4
    0)
60  PLOT H,V
70  REM  GOTO 20

]
```

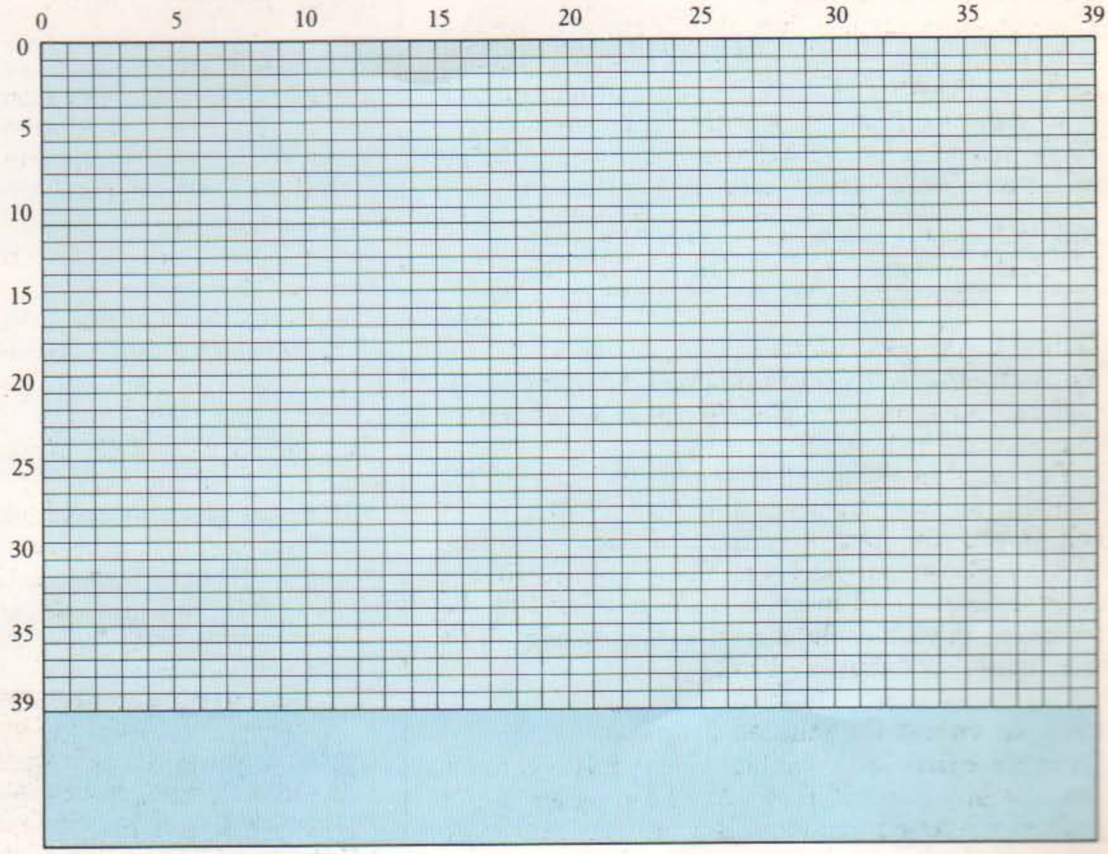
Finally, don't forget that CTRL-C can be helpful in telling you how far the Apple has got through a program. If you RUN a program which either seems to do nothing, or gets stuck at a certain point, press CTRL-C and a message will tell you where the hold up is. You can PRINT and change variables after typing CTRL-C and then use the command CONT to CONTinue RUNning the program.

GRAPHICS GRIDS

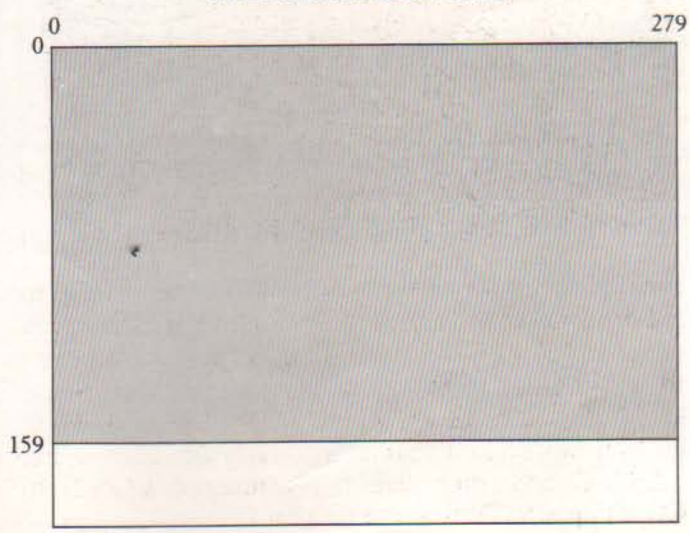
The grids below show the co-ordinates of the screen display for low-res (GR), hi-res (HGR) and hi-res 2 (HGR2) graphics. A point on the screen is identified by two co-ordinates, x and y. The first co-ordinate sets the horizontal position which is measured along from the left-hand side of the screen. The second co-ordinate sets the vertical position which is measured from the top of the screen down. The co-ordinates (30,15) therefore locate a point which is 30 places across the screen from the left and then 15 places down. On the low-res and hi-res screens, four lines at the bottom of the screen are reserved for text.

In order to produce graphics of any complexity on the Apple you have to use one of the two hi-res graphics modes – HGR or HGR2. Both screens permit you to plot tiny "points" instead of the "blocks" that are plotted on the low-res screen, but HGR2 produces a full-screen display whereas HGR leaves four lines for text at the bottom of the screen.

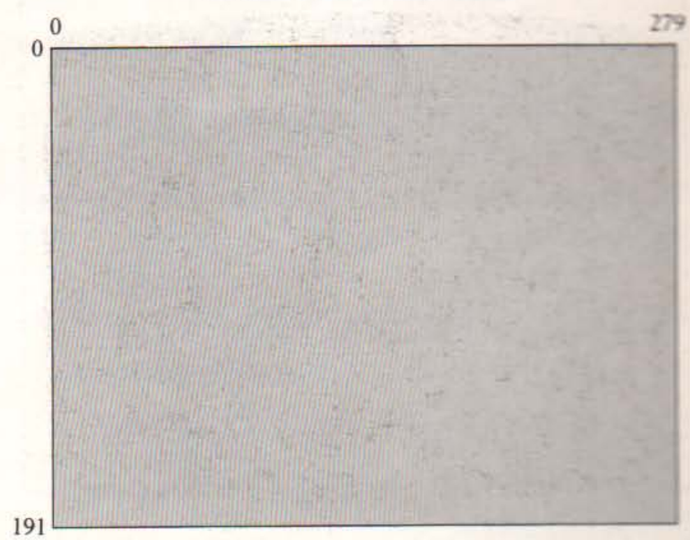
LOW-RES GRAPHICS GRID



HI-RES GRAPHICS GRID



HI-RES 2 GRAPHICS GRID



GLOSSARY

Entries in **bold type** are BASIC keywords.

Address: A number used to identify a location in the computer's memory.

BASIC: Beginner's All-purpose Symbolic Instruction Code; a high-level programming language designed to be easy to learn and use.

Binary: The counting system used by computers which uses only two numbers – 0 and 1.

Bit: A single BInary digiT, i.e. a 0 or a 1.

Bug: An error that causes a program to malfunction.

Byte: A group of eight bits.

CALL: Executes a machine-code subroutine at the specified memory address.

CAT: Short for "Catalog"; the ProDOS command which displays a list of all the files stored on a disk.

CATALOG: The DOS 3.3 equivalent of **CAT**.

Chip: One of the components that plugs into the Apple's printed circuit board and contains a complete electronic circuit. Also called an integrated circuit (IC).

CHR$: Yields the character corresponding to the subsequent ASCII code.

COLOR: Sets the display color for **PLOT**ting low-res graphics.

CONT: Resumes program execution after it has been halted by **STOP** or CTRL-C.

CPU: Central Processing Unit. The component of a computer that performs the actual computation by directly executing instructions represented in machine-code and stored in memory.

CTRL-C: When pressed simultaneously these two keys halt a RUNning program.

Cursor: A flashing symbol on the Apple's screen that shows where the next character will appear when a character key is pressed.

DATA: Creates a list of items for use by **READ** statements.

Debugging: The process of ridding a program of bugs.

DEL: Deletes specified lines from a program.

DRAW: Draws a previously-created Shape at a specified point on the Apple's hi-res graphics screen.

END: Terminates the execution of a program and returns control to the user.

ESC: A control character which changes the way in which the cursor keys work for editing programs.

FLASH: Makes any subsequent text flash on the screen.

Flowchart: A diagrammatic representation of the steps necessary to solve a problem.

FOR. . .NEXT: Marks the beginning and end of a loop which the computer repeats a specified number of times.

Function: A pre-programmed calculation that can be carried out on request from any point in a program.

GOSUB: Causes the computer to execute a subroutine beginning at the line number that follows the command.

GOTO: Makes the computer jump to the line number following the command.

GR: Converts the display to 40 rows of low-res graphics with four lines of text at the bottom.

Hardware: The physical machinery of a computer system, as distinct from the programs (software) that **RUN** on the computer and perform useful work.

HCOLOR: Sets the display color for plotting hi-res graphics.

HGR: Converts the display to 159 rows of hi-res graphics with four lines for text at the bottom.

HGR2: Converts the display to full-screen (191 rows) hi-res graphics with no text lines.

HIMEM: Sets the HIghest MEMory address that the Apple can use to store program lines and variables and therefore reserves an area of memory to store Shapes or **DATA** or machine-code.

HLIN: Draws a horizontal line in low-res graphics.

HOME: Clears all text from the text window currently in operation and moves the cursor to the top-left corner of the window.

HPLOT: Plots a point or a series of lines on the hi-res graphics screen.

HPLOT TO: Draws a line from the last plotted point.

HTAB: Moves the cursor to a specified column of the **TEXT** display.

IF. . .THEN: Prompts the computer to take a particular course of action only if the condition specified is detected.

INPUT: Instructs the computer to wait for some **DATA**, from either the keyboard or the disk, which is then used in the program.

INT: Converts a decimal number into a whole, or integer, number.

INVERSE: Makes any subsequent text appear in black-on-white on the screen.

k: Abbreviation of kilobyte (one kilobyte = 1024 bytes).

LET: Assigns a value to a variable.

LIST: Displays all, or part, of the program in memory on the screen. It can also output a program to a disk or to a printer.

LOAD: Reads a program into memory from disk.

Loop: A sequence of program statements which is executed repeatedly, or until a specified condition is fulfilled.

NEW: Clears the current program from memory and resets all variables and internal control information to their initial states so that a new program may be entered.

NORMAL: Cancels the effect of **INVERSE** or **FLASH**.

PEEK: Yields the contents of a specified location in memory.

PLOT: Plots a point at a specified position on the low-res graphics screen.

POKE: Stores a value at a specified location in memory.

PRINT: Transfers strings, numbers and variables to the current output device. This is most commonly the screen but it can also be used with the disk or printer.

RAM: Random Access Memory. The contents of RAM are erased when the Apple is switched off.

READ: Instructs the computer to take information from a **DATA** statement.

REM: The computer ignores a program line beginning with **REM**. **REM** therefore enables the programmer to insert reference REMarks.

RESTORE: Causes the next **READ** statement executed to begin **READ**ing at the first item of the first **DATA** statement in the program.

Return: The Return key, on the right hand side of the keyboard, enters a command or program line into memory after it has been typed on the keyboard.

RETURN: The **RETURN** command returns control from the subroutine to the statement following the GOSUB that called the subroutine.

RND: Yields a RaNDom number.

ROM: Read Only Memory which is programmed permanently by the manufacturer and whose contents are not lost when the Apple is switched off.

ROT: Sets the angle at which a Shape will be ROTated before it is **DRAW**n or **XDRAW**n.

RUN: Executes an Applesoft program.

SAVE: Writes the program currently in memory to a disk.

SCALE: Sets the scale factor to which a Shape will be **DRAW**n or **XDRAW**n.

SCRN: Returns the code for the color currently displayed at a designated position on the low-res graphics screen.

Shape: A Shape, described and saved in coded numbers, that can be **DRAW**n on the hi-res screen.

Software: Computer programs.

SPC: Introduces a specified number of SPaCes.

SPEED: Sets the speed at which characters are sent to the display screen. The slowest rate is 0 and the fastest is 255.

SQR: Returns the SQuaRe root of the number that follows it.

STEP: Sets the size of the increment in a **FOR. . .NEXT** loop.

STOP: Terminates the execution of a program at the point where it appears in the listing and gives a message identifying the line in which it appears.

String: A sequence of characters treated as a single item – a name for instance.

Subroutine: A part of a program that can be executed on request from any point in the program.

Syntax: The rules governing the structure of statements and commands in a programming language.

TAB: Positions the cursor to a specified position on the screen. It is used with a **PRINT** statement.

TEXT: Converts the display to 24 lines of text.

Variable: This term refers to a labeled slot in the computer's memory in which information can be stored, and also to the symbol used in a program to represent such a location.

VLIN: Draws a Vertical LINe on the low-res graphics screen.

VTAB: Moves the cursor to a specified row of the TEXT display.

XDRAW: Draws a Shape at a specified point on the hi-res screen in the complement of the color already displayed at that point.

INDEX

Main entries are given in **bold type.**

Acknowledgements
Dorling Kindersley would specially like to thank Ian Graham for his significant contribution to this series.

Thanks are also due to Apple Computer (UK) Ltd for their advice and generosity, Caxton Software Ltd for a copy of *Brainstorm* and Emily Reed for her help at every stage in the preparation of this book.

PRENTICE HALL

COMPUTERS

The original and exciting new teach-yourself programming course for Apple IIe owners.

Over 150 unique screen-shot photographs of program listings and programs in action – showing on the page exactly what appears on the screen.

Packed full of programming tips and techniques, reference charts and tables, and advice on how to get the most out of your Apple IIe.

CONTENTS INCLUDE

Getting started • Inside your computer • Computer calculations • Computer conversations • The electronic drawing-board • Introducing color • Animation • Compiling a data bank • Special screen techniques

Other volumes in the series include
Apple IIe Programming BOOK TWO
PLUS
Commodore 64 Programming
and **IBM PCjr Programming**

All programs in this book are fully compatible with the Apple II (with Applesoft BASIC), the Apple II+ and the Apple IIe.

PRENTICE HALL, INC.

3
ISBN 0-13-038456-9
0 21898 03845 6