ISSN 0265-2919

Ⓡ 80p

# THE HOME COMPUTER COURSE 22

MASTERING YOUR HOME COMPUTER IN 24 WEEKS

An ORBIS Publication

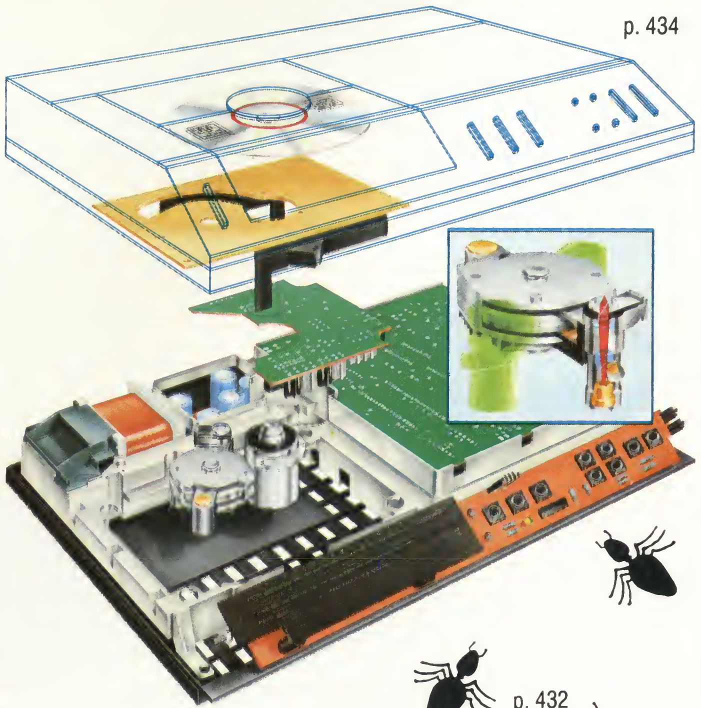
IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

# CONTENTS

## Next Week

●The QL (Quantum Leap) is the latest in a long line of outstanding technological achievements from Sinclair Research. Its CPU is as powerful as some mainframe computers

●Machine code is itself a quantum leap up from BASIC, both in terms of speed of operation and difficulty of learning. We take an introductory look at the subject


p. 434

p. 432

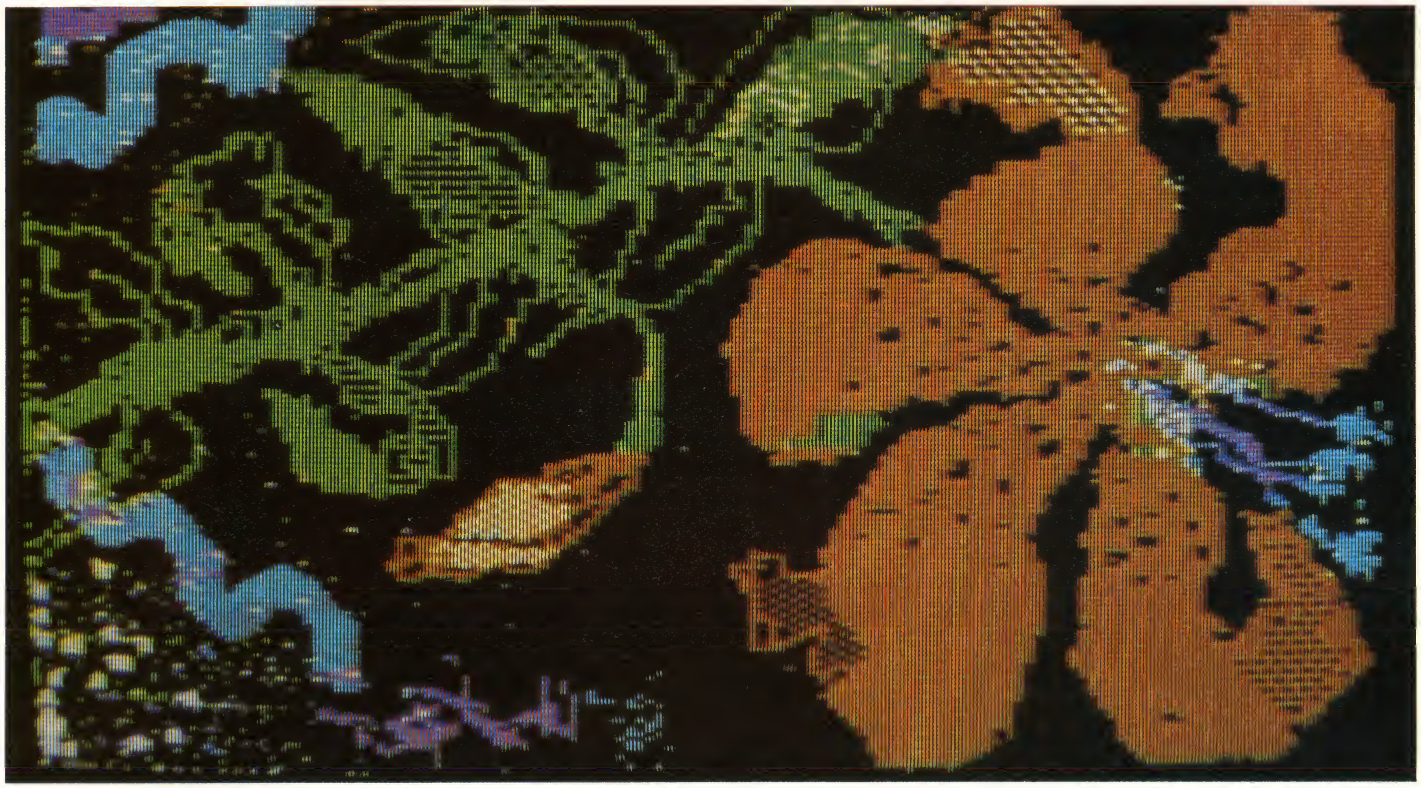
COVER PHOTOGRAPH: IAN McKINNELL

**Editor** Richard Pawson; **Consultant Editor** Gareth Jefferson; **Art Director** David Whelan; **Production Editor** Catherine Cardwell; **Staff Writer** Roger Ford; **Picture Editor** Claudia Zeff; **Designer** Hazel Bennington; **Art Assistant** Liz Dixon; **Sub Editors** Robert Pickering, Keith Parish; **Researcher** Melanie Davis; **Contributors** Tim Heath, Henry Budgett, Brian Morris, Lisa Kelly, Steven Colwill, Richard King; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Chris Cooper; **Production Co-ordinator** Ian Paton; **Circulation Director** David Breed; **Marketing Director** Michael Joyce; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 85 Charlotte Street, London W1; **© 1984 by Orbis Publishing Ltd: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

**HOME COMPUTER COURSE** – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95
**How to obtain your copies of HOME COMPUTER COURSE** – Copies are obtainable by placing a regular order at your newsagent.
**Back Numbers UK and Eire** – Back numbers are obtainable from your newsagent or from HOME COMPUTER COURSE. Back numbers, Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are obtainable from HOME COMPUTER COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE & MALTA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax.
**How to obtain binders for HOME COMPUTER COURSE** – UK and Eire: Please send £3.95 per binder if you do not wish to take advantage of our special offer detailed in Issues 4, 5 and 6. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency. In case of difficulty write to HOME COMPUTER COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.
**Note** – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

# Line Of Sight

**Computer Aided Design involves complex calculation and high-quality graphical output. Some of the same principles are employed in packages for home computers**



DIGITISER BY CONNY JUDE SCREEN SHOT BY IAN McKINNELL COURTESY OF SOFT

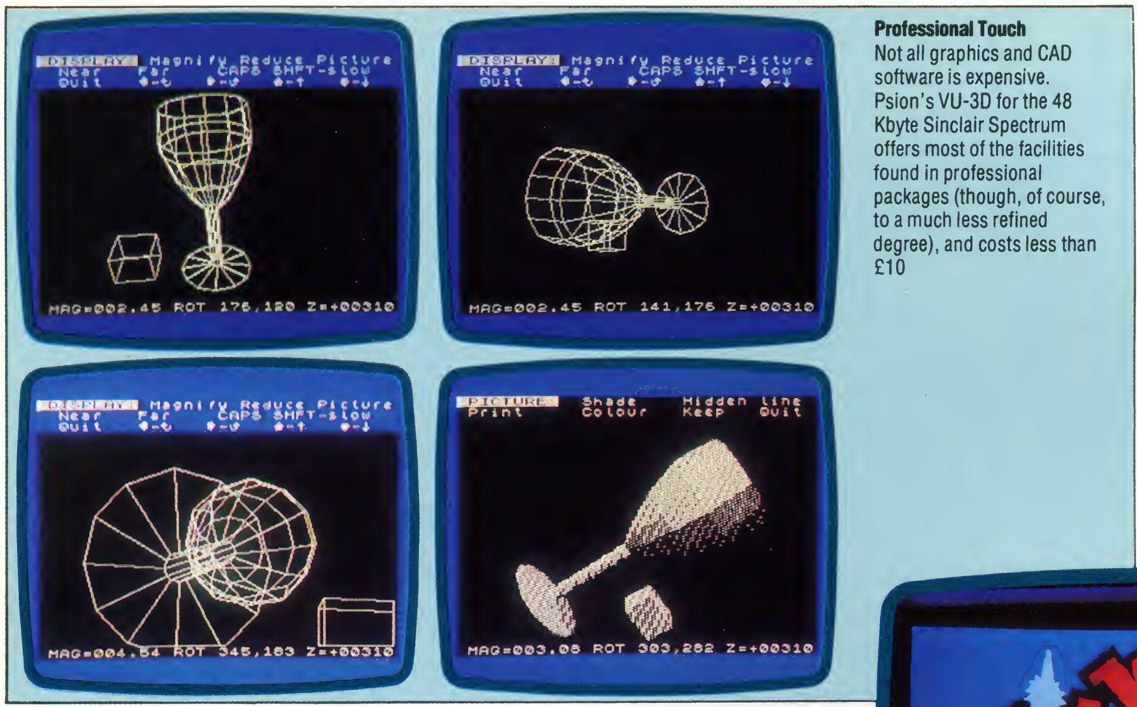
**Apple Blossom**
No matter how sophisticated the software, the most important component of a Computer Aided Design package is the designer using it. Versawriter, the results of which are shown here, runs on an Apple II and requires two disk drives and a digitiser. The flower took an experienced user some considerable time to enter into the machine

The idea of applying computers to the process of industrial design was first suggested in the early sixties (at the Massachusetts Institute of Technology). However, it was not until a decade later that computer technology enabled the designer to see and interact with a graphic representation of his work, presented on a monitor screen and accessible via a digitiser or light pen, exactly as if he were at a drawing board. These essential peripherals — the digitiser, light pen and plotter — are the basic tools of the Computer Aided Designer's art. With them it is possible to create images in just the same way as an animator does (see page 181), by 'drawing' on a digitising tablet. The designer can modify that image, perhaps using a light pen, incorporating pre-drawn components and sub-assemblies, and then produce a copy of the finished drawing on a plotter. The computer becomes a drafting system similar in principle to a word processor, but working with images instead of text.

We have seen that the quality of the image produced on a computer's monitor depends on two things: the resolving ability of the monitor (that is, the size of an individual picture element or pixel), and the power and memory size of the computer driving that monitor. When we looked at computer generated images, we saw much the same requirement — a monitor that could resolve to something like 1,000 × 1,200 pixels, and a computer capable of processing these 1.2 million picture elements in less than 1/24th of a second.

It is useful to continue the analogy between the Computer Aided Design package and the word processor. Instead of moving paragraphs, sentences or individual words around a piece of text — amending, inserting or deleting these at will — the CAD program can be used to move elements of a drawing around the page. The effect may be different, but the principle is exactly the same.

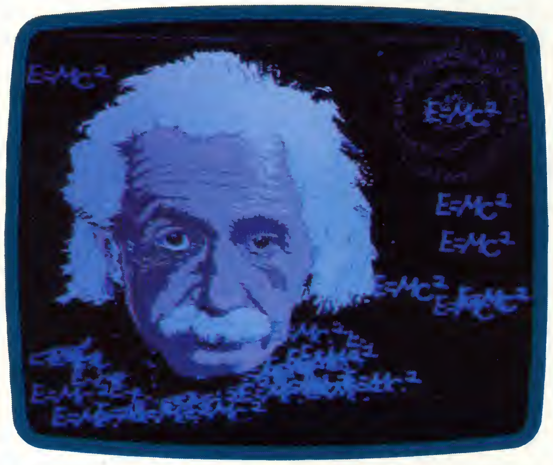
The problem for the program is: how to store this image in such a way as to allow it to be altered and manipulated. If we were to make a physical model of an object, we would use one of two basic methods — additive or subtractive. The additive method is rather like modelling in clay: building up the object piece by piece until we arrive at the final shape. The subtractive method is like that used by the sculptor, who takes away material to achieve the same result. The computer's analogy for a solid block of material is a three-dimensional


DISPLAY: Magnify Reduce Picture
Near Far CAPS SHFT-slow
Quit
MAG=002.45 ROT 176,120 Z=+00310
MAG=002.45 ROT 141,176 Z=+00310
MAG=004.54 ROT 345,183 Z=+00310
PICTURE Shade Hidden line
Print Colour Keep Quit
MAG=003.08 ROT 303,282 Z=+00310

IAN McKINNELL

**Professional Touch**
Not all graphics and CAD software is expensive. Psion's VU-3D for the 48 Kbyte Sinclair Spectrum offers most of the facilities found in professional packages (though, of course, to a much less refined degree), and costs less than £10

array. Consequently, the size and performance of the computer involved becomes important. If the array is large enough to enable one whole byte to be allocated to the definition of each pixel or element of the image, then the amount of information that we can retain about the single element is quite large (256 separate pieces when using an eight-bit processor, a great deal more for 16- or 32-bit devices). But the problems of creating this much storage space are practically insoluble, and so we are forced into a compromise



COURTESY OF IO RESEARCH

**Hot Dog**
Io Research's Pluto system brings high resolution image generation to a wide range of small microcomputers with the addition of a fast processor and extra memory. The basic system costs only £500 and gives eight fixed colours and a resolving power of 670 × 576 pixels

— but one that is generally quite acceptable. Instead of allocating one whole byte to each element, it is sufficient to allocate a single bit, if all we want to do is indicate the presence or absence of an element at this position on the model.

Computer Aided Design software shares many attributes with Computer Generated Image packages: curve smoothing, hidden element removal, shading, block filling and re-colouring, for example. It only requires the repeated solution of a simple equation for a series of values to form a curve. If we specify the starting and finishing points for a given line, and the maximum distance away from that straight line that the curve will reach, then we have provided one solution to the equation. We can work backwards from that solution to deduce the equation itself, and then proceed to solve it for the rest of the series of values, thus forming the curve.

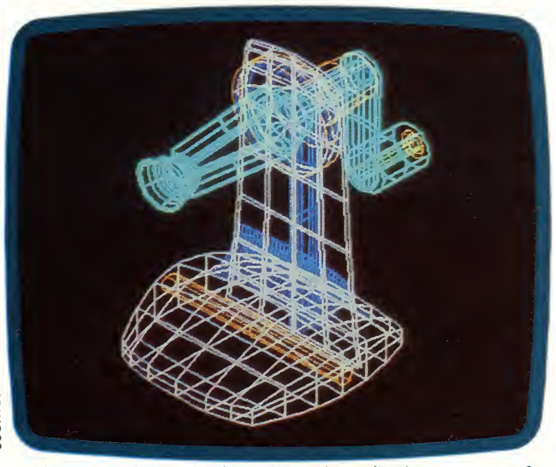
This ability to compose a drawing from standard component parts is the real strength of CAD systems. No longer is it necessary to re-draw common individual components. When they have been defined once, that definition can be recalled as often as required and incorporated into new drawings. One particularly good example of this is

the use of computers to assist with the designing of future generations of computers.

Printed circuit board design, for example, is quite complex, involving optimising techniques in order to arrange components and their interconnection paths in the most economical way possible (bearing in mind that connection paths can never cross). The designer is often forced to fall back on trial and error — and it is here that CAD packages are particularly useful. All the individual components are stored as pre-defined images, and are called up as and when required. It is a simple matter to try out a particular design on the visual display unit to see if it meets all the criteria before committing it to paper as part of a working drawing. By this method it is possible to assemble a design, and even test out the efficiency of a variety of different solutions, in the time it would otherwise take to complete a single draft.

Integrated circuits are designed in almost exactly the same way, but due to the density of components and connecting pathways, a further software feature is necessary: the ability to magnify a part of the drawing, work on it at the enlarged scale and then place it in position again within the overall design. This effect is now an important part of the CAD repertoire, and has added considerably to the efficiency of the system. By its use, the specification of a complete object can be held in just one drawing, and the scale can be adjusted to meet the viewer's requirements.

And it's not only the scale that can be varied in this way. If we take the case of a more complex object — a car, for example — we are presented with a variety of sub-systems that go together to


COURTESY OF APPLICON

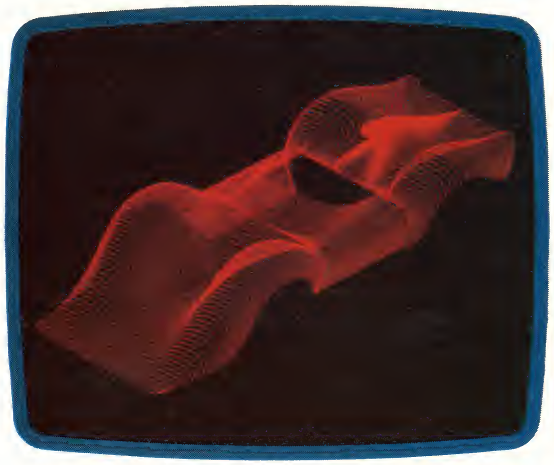
make up the whole: the electrical system, the hydraulic system, the exhaust, the suspension, and so on. While the aesthetic designer will be concerned with the overall package, individual engineers are more likely to be interested in just one sub-system. It is a simple matter to keep each sub-system in a different colour, and then extract the objects of just that one colour from the whole drawing. That is not to say that the drawing must always be a coloured mixture — the coding can be suppressed at will when not required.

It is the ability to retain the *complete* specification of an object (not just its form and

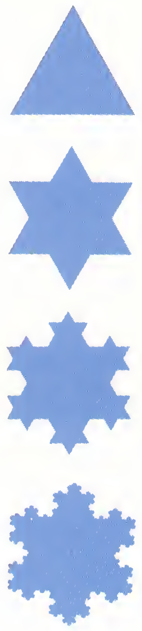
COURTESY OF SIGGRAPH

appearance, but also information on the material of which it is constructed, its weight, cost, etc.) that is the real breakthrough. Retrieving information about the shape and size of the object is only one function of the system, which can be regarded as a visually orientated database. By asking different questions of that database it is possible to: place orders to suppliers; schedule sub-assembly and component manufacture; integrate production lines to ensure that components arrive exactly where and when they are required; analyse costs; monitor manufacturing efficiency: and much more as well. It is tempting to speculate that the next step will be a regular system of direct computer control of manufacture, and with the burgeoning use of robots in the industrial process, that next step is not such a large one.

Most of the applications we have discussed here require mainframe computers or else very powerful minicomputers, but that is not to say that even small microcomputers cannot play a useful part in the design process. There is a wide variety of CAD software available for machines running under CP/M, for example, and most manufacturers offer at least one package, even for computers as relatively unsophisticated as the Sinclair ZX81. As we noted, the size and speed of the computer dictates the quality of the stored image, but the home user's requirements are unlikely to be as stringent as those of the professional designer, so it is quite feasible to achieve exciting results for a modest outlay.


COURTESY OF INTERGRAPH

**Just Imagine...**
The background to this shot from the Lucasfilms production 'Road To Point Reyes' was largely composed by fractals, a new and ingenious CAD technique. Fractals are phenomena that increase in complexity the more closely they are viewed. The hills and mountains in the background started life as simple polygons, described within the memory of a computer. Each polygon was then made successively more complex by the addition of its own shape to each of its sides, and the process repeated with a degree of randomisation. The development in shape of the snowflake, shown below, from a simple triangle, illustrates this principle



**Wired Up**
The first stage in creating an image or design in three dimensions is known as 'wire framing'. The image is defined as a series of point co-ordinates, appropriately joined by straight lines. These lines can be manipulated using the curve smoothing algorithm, hidden lines removed, and then the planes filled in with colour and shaded as necessary to increase the illusion of depth

# Getting It Taped

**The Turing machine is a purely theoretical device, used for deciding whether a problem is computable or not**

So far in THE HOME COMPUTER COURSE, we have tended to emphasise practical subjects, and things to do on your home computer. In this article, however, we're going to take a look at the theoretical side of computers: the field that is called 'computer science'. This is to computing what pure mathematics is to engineering — a highly theoretical subject, but one from which the practical ideas ultimately derive.

The Turing machine, for example, is a purely theoretical idea, developed by Alan Turing (see page 200) to assist in the study of algorithms and computability. It is really the minimal possible computer, so that if it is possible to prove that a particular problem could not be solved using a Turing machine, then that problem could be said to be 'non-computable'. Turing decided that such a minimal computer would need three facilities: an external storage for recording and storing input and output information; a means for reading from and writing to that storage; and a control unit to determine the actions to be undertaken.

A Turing machine is therefore usually defined as having a tape (if it helps, think of it as a magnetic tape), which is infinite in length (that is to say: however much tape is needed to solve a problem, there will always be enough). The tape is divided up into squares, which will either be blank or contain a symbol. A tape head mechanism that can read or write the symbols in the squares moves along the tape, receiving its instructions from a control unit that tells it what symbols to write and the direction in which to move next.

The control unit contains an execution program, and in this respect a Turing machine can be considered to have been 'built' specifically to perform one application, since there is no provision in the specification for loading or altering a program. We use the term 'built' advisedly, since the only Turing machines that have ever been physically constructed have been purely for educational purposes. However, it is a relatively simple exercise to write a BASIC program that will simulate the operation of a Turing machine on a home computer.

The control program in a Turing machine is made up of a collection of 'quintuples', or statements that contain five elements. Which quintuple is executed at any stage depends on two factors: the symbol in the square currently underneath the tape head, and the 'state' or 'condition' of the machine. This state is a purely arbitrary quality: we can specify that the machine starts off in state SA, and when it reaches the special state H then it halts, the computation being finished. In between, the state will change many times according to instructions from the quintuples. The state merely reflects what has happened in the computation so far, and serves to select which quintuple is executed next (again, if it helps, think of it as a flag variable in BASIC programming).

The five elements of each quintuple are:

1) The current state of the machine;
2) The symbol in the square of tape underneath the head;
3) The symbol to be written in that square (this is the same as 2 if no change to the data is required);
4) The state that the machine should now go into; and
5) The direction in which the tape head should move — left or right.

The quintuple (SA,5,3,SB,R), for example, will be executed whenever the machine is in state SA and the tape head reads a 5. The 5 will then be replaced by a 3, the machine changed from state SA to SB, and the tape head moved one square to the right.

Designing a theoretical Turing machine to perform a particular task involves specifying the format in which your input data will be presented to the machine on tape, the format of the output data on tape when the computation is finished (i.e. the machine is in state H), and the set of quintuples needed to execute the algorithm.

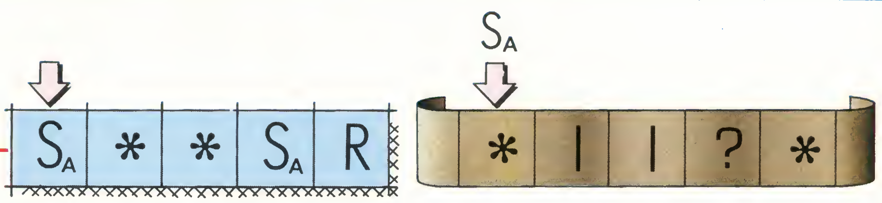
In our panel, we have designed a Turing machine to perform the AND function. We will set up the two input bits (each a 1 or a 0) in adjacent squares, followed by a question mark symbol, which is to be replaced by the answer (again a 1 or a 0, depending on the two inputs). For decorum, we have added an asterisk symbol at either end of the data area, and will start the machine going in state SA on the left-most asterisk, finishing on the right-hand one.

A total of ten quintuples are needed to specify this machine, though as you can see from the worked example (1 AND 1 = 1), only five are used for any run. If you try the same machine out for, say, 0 AND 1, you will find that a different set of quintuples will be selected from the ten.

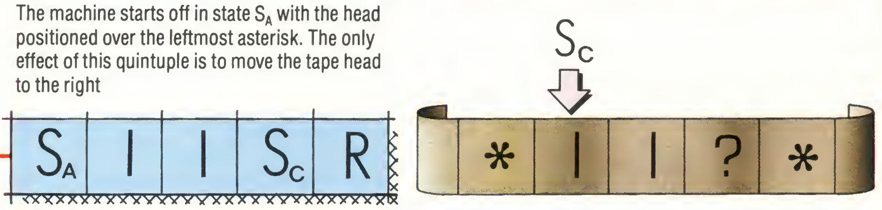
## Turing Machine

This example shows the construction of a Turing machine to perform the AND function. The two input bits are set up in adjacent squares, followed by a question mark, which will be replaced by the result. Two asterisks are placed at the ends of the data area to act as delimiters. The ten quintuples below specify the operation of the machine, though for any worked example (in this case 1 AND 1), only five of the ten will be used

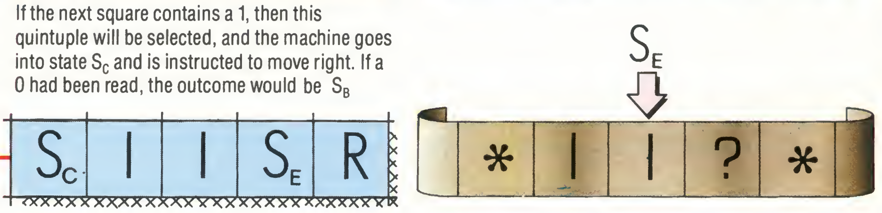
| | | | | |
|---|---|---|---|---|
| $S_A$ | * | * | $S_A$ | R |
| $S_A$ | 0 | 0 | $S_B$ | R |
| $S_A$ | 1 | 1 | $S_C$ | R |
| $S_B$ | 0 | 0 | $S_D$ | R |
| $S_B$ | 1 | 1 | $S_D$ | R |
| $S_C$ | 0 | 0 | $S_D$ | R |
| $S_C$ | 1 | 1 | $S_E$ | R |
| $S_D$ | ? | 0 | $S_F$ | R |
| $S_E$ | ? | 1 | $S_F$ | R |
| $S_F$ | * | * | H | R |



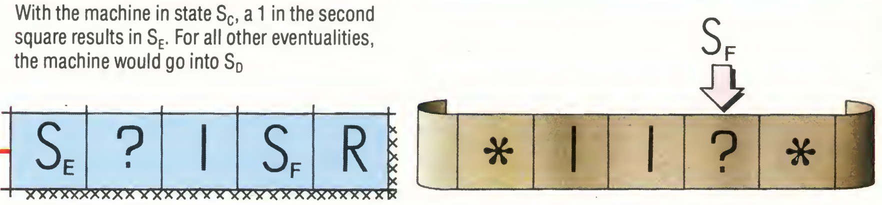
The machine starts off in state $S_A$ with the head positioned over the leftmost asterisk. The only effect of this quintuple is to move the tape head to the right



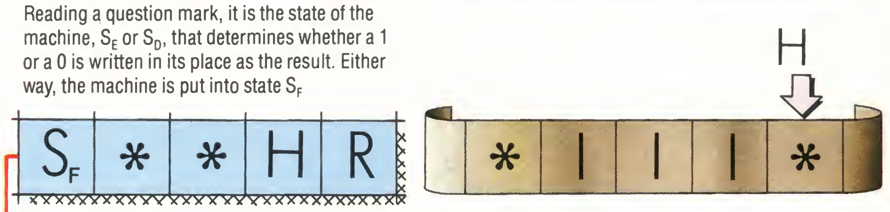
If the next square contains a 1, then this quintuple will be selected, and the machine goes into state $S_C$ and is instructed to move right. If a 0 had been read, the outcome would be $S_B$



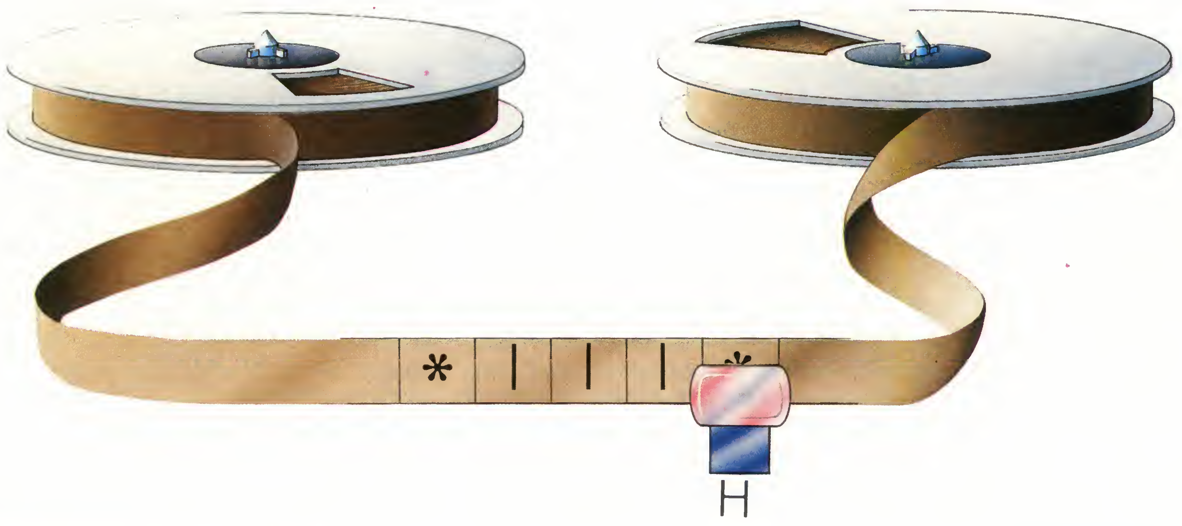
With the machine in state $S_C$, a 1 in the second square results in $S_E$. For all other eventualities, the machine would go into $S_D$



Reading a question mark, it is the state of the machine, $S_E$ or $S_D$, that determines whether a 1 or a 0 is written in its place as the result. Either way, the machine is put into state $S_F$



The machine now enters the halt state (H) over the second asterisk. You can test out the operation on paper for 1 AND 0, 0 AND 1, and 0 AND 0



KEVIN JONES

# Sound Ideas

## Continuing our look at the BBC Micro's sophisticated ENVELOPE command

In the previous instalment of the Sound And Light course we introduced the BBC Micro's ENVELOPE command. This is one of the most powerful commands available to the BASIC programmer when used with the SOUND command, discussed on page 358. We now continue our explanation of ENVELOPE by looking at 'volume envelope'.

In the following line of programming parameters, N to NS3 are concerned with the pitch envelope, and these were dealt with on page 408.

```
ENVELOPE N,T,PS1,PS2,PS3,NS1,NS2,NS3,AR,DR,
SR, RR,FAL,FDL
```

The remaining parameters are all concerned with the volume envelope, between them setting peak volumes and rate of change of volume over the duration of the note set by the associated SOUND command.

AR & DR (−127 to 127) + FAL & FDL (0 to 126)

AR sets the Attack Rate of the note. Although the software allows a negative value, in practical terms the range is 1 to 127. This relates to the number of volume changes per time step and continues to rise until the Final Attack Level (FAL) of volume is reached, which indicates the beginning of the decay phase. Decay Rate is controlled in a similar manner by DR, usually a negative value, causing volume to fall until it reaches the Final Decay Level (FDL). Although software allows a range of 0 to 126 for final volume levels, current hardware only allows 0 to 16, so a FAL value of 50 would be automatically scaled down and rounded off to a volume of 6.

SR & RR (−127 to 0)

The Sustain Rate (SR) and Release Rate (RR) also refer to volume changes per time step although both must take negative values. Sustain continues until the duration set by the SOUND command is complete. This means that if the Attack time and Decay time together are greater than or equal to the set Duration time, there will be no Sustain

# Light Waves

## Atari's graphics set a trend that other manufacturers have followed

The Atari 400 and 800 home computers are well known for their plug-in cartridge systems, but the machines themselves also have fairly sophisticated graphics facilities available in BASIC. These facilities, common to both machines, support nine levels of screen display — three text modes (offering different character sizes) and six graphics modes. The maximum resolution obtainable is 320 x 192 dots.

There are 16 colours to choose from on the Atari computers, but the maximum number that can be displayed at any one time is five. The standard ASCII upper and lower character sets are available, as well as 37 special Atari graphics characters. These characters may be used in PRINT statements to build up low resolution displays and tables. The Ataris also allow cursor movement to be controlled from a BASIC program. This is done using cursor control characters within PRINT statements to position the text that follows on the screen. The cursor control characters allow up/down or backwards/forwards movement of the cursor.

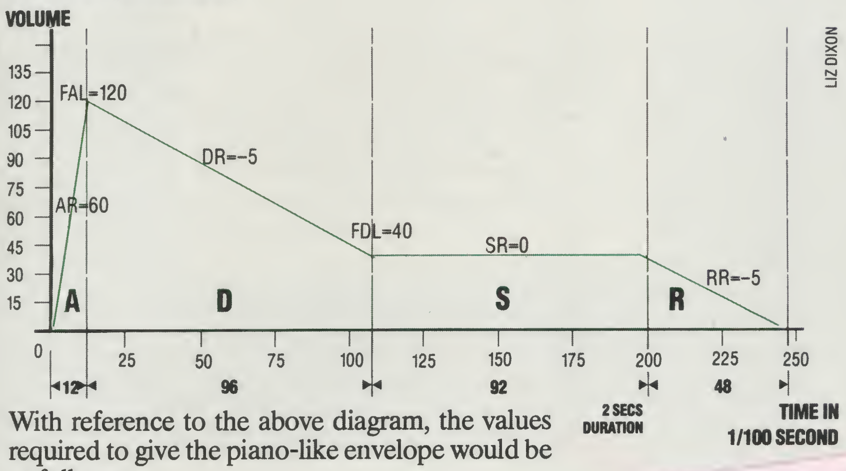
One of the most attractive features of the Ataris is their ability to use sprite-style graphics, known as 'Player-Missile' (PM) graphics, which allow the user to write fast-moving arcade games in BASIC. There are, however, no special BASIC commands to use PM graphics, and all the necessary work has to be done by manipulating the memory locations in RAM, using PEEK and POKE. Player-Missile graphics will be discussed more fully in a later part of the course.

### Display Modes

Modes 0, 1 and 2 are for text display. When the machine is switched on, the display is set to mode 0 and the screen is formatted into 24 rows, each containing 40 character spaces. In this mode the display characters are based on the standard eight by eight ASCII format. Characters PRINTed in mode 1 are twice the width of mode 0 characters, but are still the same height; whilst mode 2 characters are twice the height and width of those in mode 0.

With the exception of mode 0, all graphics modes have a split screen, the bottom few lines being reserved for miscellaneous data such as error messages. To PRINT to the main body of the screen in modes 1 and 2, a device number must be specified. PRINT#6 allows text to be PRINTed to the phase, even if it has been programmed in. Release begins when Duration is complete. Volume falls to zero at the set rate unless a new note is started on the same oscillator, which means that Release is cut off unless 'H' has been set to '1' by means of a new SOUND & command.

### Volume Envelope



With reference to the above diagram, the values required to give the piano-like envelope would be as follows:

```
T=6     AR=60    SR=0     FAL=120
        DR=-5    RR=-5    FDL=40
SOUND duration=40 (two seconds)
```

Resulting in:

```
ENVELOPE 1,6,0,0,0,0,0,0,60,-5,0,-5,120,40
```

The following program employs all the sound associated BBC BASIC commands to play a well known sequence of notes with the piano volume envelope, and a short triangular repeated pitch envelope on the final chord.

```
10 REM**COSMIC**
20 ENVELOPE 1,6,0,0,0,0,0,0,60,-5,0,-5,120,40
30 ENVELOPE 2,6,1,-1,1,1,2,1,60,-5,0,-5,120,40
40 FOR I=1TO4:READ N
50 SOUND 1,1,N,20:REM**PLAY A B G G**
60 SOUND &1001,0,0,5:NEXT I
70 SOUND &201,2,77,40:REM**FINAL**
80 SOUND &202,2,89,40:REM**D MAJOR**
90 SOUND &203,2,109,40:REM**CHORD**
100 DATA 137,145,129,85:REM**A B G G**
```

graphics part of the screen. Modes 3 to 8 are graphics modes and allow points and lines to be plotted on the screen with varying degrees of resolution and a choice of colours. This table shows the complete range of options available to the user:

| MODE | TYPE | ROWS | COLS | COLOURS |
|---|---|---|---|---|
| 0 | text | 24 | 40 | 2 |
| 1 | text | 20 | 20 | 5 |
| 2 | text | 10 | 20 | 5 |
| 3 | graphics | 20 | 40 | 4 |
| 4 | graphics | 40 | 80 | 2 |
| 5 | graphics | 40 | 80 | 4 |
| 6 | graphics | 80 | 160 | 2 |
| 7 | graphics | 80 | 160 | 4 |
| 8 | graphics | 160 | 320 | 1 |

The choice of mode will depend on how much memory there is available for screen display. Mode 5, for example, requires almost twice as much memory to support four colours as mode 4 needs to support two.

## Basic Commands

There are a number of commands in Atari BASIC to help with graphics. These commands also work in modified form in the three text modes.

SETCOLOR a,b,c

There are five colour registers to control the use of colour on the screen, but not all of them are used in every mode. SETCOLOR is used to select the colours used by these five registers. In this command a is the colour register number, 0-4; b is the colour number to be used, 0-15; and c enables each colour to be displayed in one of eight levels of brightness, by choosing an even number between 0 and 14.

COLOR n

This command works in two ways, depending on whether a text or a graphics mode has been selected. In modes 0, 1, and 2, n is a number in the range 0 to 255. In its binary form this number is made up of eight bits: the first six bits relate to the ASCII code of the character being PLOTted, and the other two bits are reserved for the colour information about the character.

In the graphics modes, n takes on a value between 0 and 3, and is used to select a particular colour control register when PLOTting a point.

PLOT x,y

The origin of the Atari screen is placed in the top left-hand corner of the screen. PLOT illuminates the graphics point with co-ordinates (x,y). Similarly, the POSITION command:

POSITION x,y

places an invisible cursor at the point (x,y) on the screen.

DRAWTO x,y

draws a straight (or as straight as is possible in the lower resolution modes) line from the old cursor position to the point (x,y). Finally the line:

X10 18,#6,0,0,"S:"

employs the Atari input/output command X10, which allows the user to fill or paint a shape drawn on the screen. It is rather complicated, but can produce some good results if used carefully. Once a closed shape has been drawn on the screen, then the cursor should be set to the bottom left-hand corner of the area that is to be coloured in. The colouring will start from the top of the shape and will fill it in, between the boundaries, until the cursor position is reached at the bottom. The colour is set by POKE 765,C where C is 1, 2, or 3, as used in the COLOR command.

**XL Size**
Atari graphics can be quite interesting but are not particularly easy to use. Limited colour choice and the lack of many of the 'standard' high resolution commands, such as CIRCLE, mean that the programmer has to work fairly hard to achieve good results. Atari does have the advantage, however, of a large range of text modes. The following program demonstrates the use of double size characters, in conjunction with the POSITION command, to PRINT a familiar message on the screen:

```
 10 REM* BIG LETTERS *
 20 GRAPHICS 2+16
 30 SETCOLOR0,3,6
 40 FOR X=19TO8 STEP-1
 50 POSITION X,1
 60 FOR J=1TO100: NEXT J
 70 PRINT#6; "HOME "
 80 NEXT X
 90 FOR X=19TO6 STEP-1
100 POSITION X,3
100 FOR J=1TO100: NEXT J
120 PRINT#6;
     "COMPUTER "
130 NEXT X
140 FOR X=13TO7 STEP-1
150 POSITION X,9
160 FOR J=1TO100: NEXTJ
170 PRINT#6; "COURSE "
180 NEXT X
190 SETCOLOR 0,5,5
200 FOR Y=9TO5 STEP-1
210 POSITION 7,Y
220 PRINT#6; "COURSE "
230 NEXT Y
240 GOTO240
```

Note that when a mode is selected, the split screen effect can be overridden by adding 16 to the mode number

# Newspeak

**The world of computers has generated some imaginative language. These 'buzzwords' often have interesting origins**

**Many of the terms that are used to describe aspects of computing have rather obscure origins. Every trade has its jargon (code words and phrases that are especially used by the people involved in that trade), and none more so than the computer industry. In fact, computer people even have a jargon word for their jargon: they call them 'buzzwords'.**

## BUZZWORDS

The word **BUZZWORD** first surfaced in the late 1960's, when someone in Honeywell's publicity department developed a game called a 'Buzzword Generator'. The game was centred on three columns of ten words each, numbered 0—9. The first column contained adjectives, and the other columns consisted of nouns that could stand in apposition. You simply thought of a three figure number, looked up the appropriate words, and there you had an utterly meaningless phrase, such as 'interactive system module'. This could then be used to pepper conversation with your friends and colleagues, in order to baffle and confuse them.

## BOOT

**BOOT** is a contraction of bootstrap: as in 'to pull oneself up by one's bootstraps'. A bootstrap loader is a routine that is automatically run whenever a computer is powered-up (N.B. for the dedicated computer user, it's not sufficient to say 'switched on'). In machines that do not have an operating system in ROM, the boot routine must contain instructions to call in that operating system from disk, or else the machine could not be used.

## BIT

**BIT** is a buzzword in its own right. Though most dictionaries declare it to be a contraction of '**BI**nary digi**T**', it seems equally likely that it is just an extension of its common meaning: 'a small piece of something'. It's worth bearing in mind, though, that in American slang a bit is also an eighth part of a dollar, and is always spoken of in twos: 'two bits', for example, is a quarter — 25 cents.

Bit often appears as a prefix: as in 'bit-slicing', a term used to explain how certain rather sophisticated microprocesssors can be constructed out of two, four, or eight bit 'building blocks', resulting in devices with capacities as large as 32 bits. Computing wisdom has it that programs left unused for a long time will develop additional and unsolvable bugs, and this imaginary phenomenon is referred to as 'bit-decay'.

## TURNKEY

When it comes to people greeting their computer system, perhaps for the first time, yet another jargon word has evolved. Many commercial organisations employ a firm of computer consultants to install hardware and software so that the client can take it over in working order. This is known as **TURNKEY** operation, because all the client has to do is turn the key and drive away.

## HARDWARE

## SOFTWARE

**HARDWARE** and **SOFTWARE** are in themselves buzzwords ('hard' meaning tangible and 'soft' the opposite), but there are two other types of 'ware' as well. **FIRMWARE** meaning software that is encapsulated in hardware (such as in the ROM or EPROM), and **LIVEWARE**, which refers to all those people fortunate enough to work with and use computers!

## BASIC

**BASIC** itself is a buzzword, standing for **B**eginners' **A**ll-purpose **S**ymbolic **I**nstruction **C**ode; though, as with so many acronyms, one suspects that the word was thought of before the phrase.

## BAUD

**BAUD** — the rate at which data is transmitted — is named after Emile Baudot, the inventor of a telegraphic code that initially rivalled the more successful one devised by Samuel Morse.

## BYTE

**BYTE** is an often encountered computer term, and though it is no more than 30 years old, its origins are already lost in obscurity. Until the eight-bit microprocessor appeared, a byte was enough bits to encode a single character — sometimes six, sometimes eight. At that time, computers rarely used a word of less than 24 bits; and some machines, chiefly those designed for scientific applications, went as high as 64 bits. The eccentric spelling of byte has led to the coining of the term **NYBBLE** — half a byte! Straining the analogy a little further, a **GULP** is a small group of bytes.

## LOGIC BOMBS

## TROJAN HORSES

The media are always very quick to latch on to imaginative pieces of jargon, and in recent years they have taken to making up some of their own. The subject of computer crime is particularly fertile ground for buzzword generation: **LOGIC BOMBS** and **TROJAN HORSES** are two of the methods supposedly used for fraudulent purposes. The former describes a piece of code that is written into an applications program but which remains dormant (has no effect) until the program has been running for a sufficient length of time for the fraud (moving money from one account to another perhaps) to go undetected. A Trojan Horse, we are led to believe, is a program which is disguised as another program in order to gain entry to the system.

## TIME BOMB

A similar expression, but one referrring to an authentic practice, is **TIME BOMB**. This describes a particularly ingenious technique for protecting business software against piracy. It is a piece of code within the package, which would normally be disabled when the system is installed by the bona fide dealer. On a pirated copy, however, the Time Bomb will wait until a certain date is reached (often April 1), by which time there is a good chance that the company will be heavily dependent on the package. The day after the bomb has 'exploded', not only will the user's files have been turned into garbage, but the copy of the program will also have been destroyed (unless the disk was protected against being overwritten).

## GARBAGE

**GARBAGE** is a word that occurs in several phrases in a computer user's dictionary of jargon. For example, the acronym **GIGO** stands for '**G**arbage **I**n, **G**arbage **O**ut', and this is really just a reminder that computers are only processing information, and therefore you can't expect accurate results if you don't feed in accurate data in the first place.

**GARBAGE COLLECTION** is the name given to an internal process that may well be used in your home computer, if it uses a version of BASIC that permits dynamic strings (i.e. strings that can change in length during a program). Every time a string increases in length, a complete new copy will be made in RAM. So if there are a lot of statements of the form `LET A$=A$+"*"` (particularly within loops) then it won't take long for the memory to fill up completely. At this point, the program execution will automatically come to a temporary halt, and a routine in ROM called the 'garbage collector' will tidy up the string area, and remove all the sections of strings that have been left over from previous manipulation. Though the program will resume when the garbage collector has finished, the process can take seconds or even minutes, during which the computer will cease all operations.

## HANDSHAKE

Many computing buzzwords derive from analogy. When a business deal has been agreed, for example, the participants may well shake hands: so in computing terms a **HANDSHAKE** is the name given to the electronic signal that signifies that an exchange of data is complete.

| | | |
|---|---|---|
| 0.Integrated | 0.Database | 0.Network |
| 1.Interactive | 1.Situational | 1.Capability |
| 2.Buffered | 2.Top-down | 2.System |
| 3.Digitised | 3.Diagnostic | 3.Algorithm |
| 4.Stochastic | 4.Addressing | 4.Processor |
| 5.Peripheral | 5.Linear | 5.Array |
| 6.Heuristic | 6.Graphic | 6.Module |
| 7.Relational | 7.Alphanumeric | 7.Facility |
| 8.Customised | 8.Image | 8.Hierarchy |
| 9.Programmable | 9.Schematic | 9.Generator |

**Generator Hum**
The term 'buzzword' was first used to describe a simple game that could create meaningless but convincing technological jargon phrases. You can devise your own 'buzzword generator' by thinking up three columns of ten words each, as we have done here. Choosing a three digit random number will 'generate' a resounding phrase

# Commodore PET 4032

**The Commodore PET was the first personal computer. Since its introduction, however, the machine's hardware has advanced considerably**

In many ways the Commodore PET (an acronym of Personal Electronic Transactor) was the machine that started the whole microcomputer boom. When it was released in 1977, it set such a high standard that it's possible to regard some more recent machines as retrograde steps in comparison. The original machine's metal casing serves as an excellent example of its superiority. Apart from the Memotech and the more expensive business machines, the cases of most recent computers are moulded from plastic, and these range in quality from the barely adequate to the shoddy. The PET's built-in power supply is another detail that separates it from many of its competitors in the home market.

CHRIS STEVENS

**PET Keyboard And Monitor**
The first PETs had a non-standard keyboard, the later ones more closely approximate the style of a typewriter and feature the graphic symbols on the front of the keys (except the business models). All PETs feature built-in monitors: the later ones have 12" (30 cm) screens, with green on black displays and a choice of 40 or 80 character columns

Although eight-bit, as well as 16-bit, machines had been available for at least two years before the PET was released, these were either kits or simple 'minimal systems' consisting only of chips on a PCB. The PET was the first readily available microcomputer that could truly be described as 'plug-in-and-go'. The very early versions of the PET had a built-in tape recorder with motor control, a built-in monitor, and ROM BASIC. All that a new user had to do to start work was to plug it in and turn it on, and almost immediately a reassuring message:

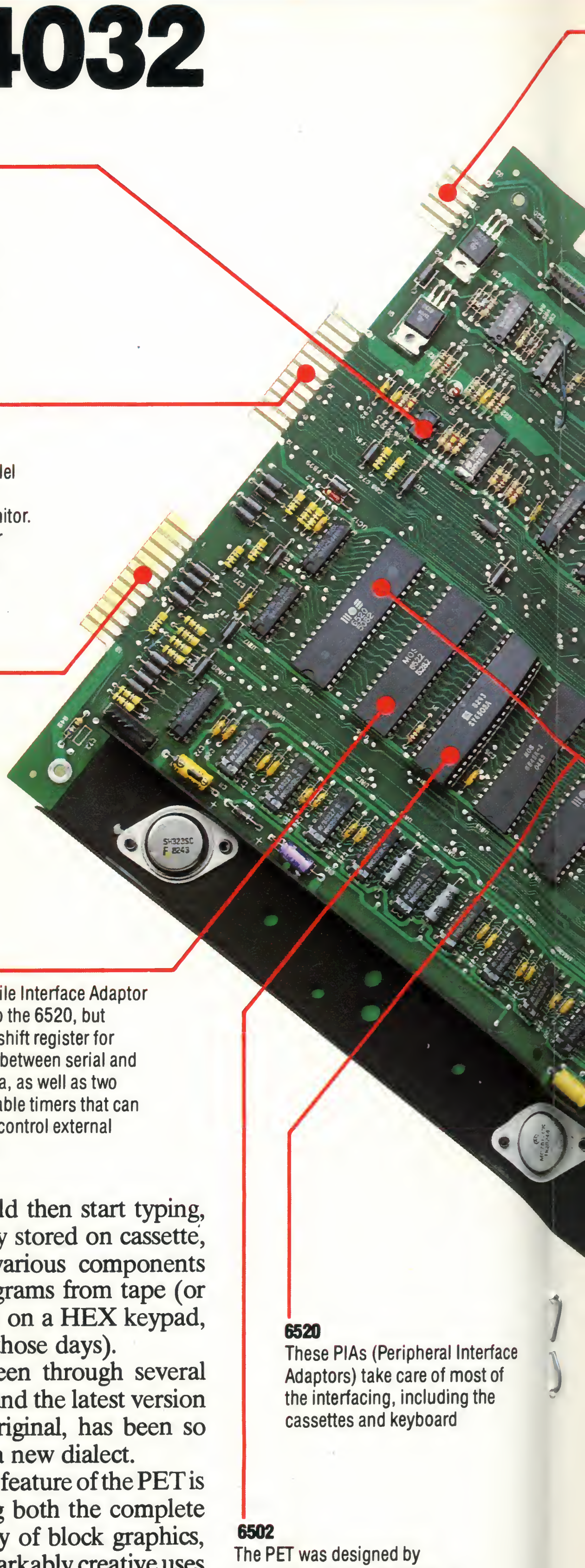
```
COMMODORE BASIC VER. 1.0
7167 BYTES FREE
READY
```

would appear. The user could then start typing, and this work could be safely stored on cassette, without the need to plug various components together or load system programs from tape (or worse, to have to enter them on a HEX keypad, which wasn't uncommon in those days).

Commodore BASIC has been through several revisions during its lifetime, and the latest version (4), though based on the original, has been so extended as to make it into a new dialect.

Another major and unique feature of the PET is the character set. Containing both the complete ASCII set and a large variety of block graphics, this has been put to some remarkably creative uses by PET owners, despite the relatively low resolution of the characters. However, a major problem of the machine was that the codes generated by the keyboard don't match the ASCII set, nor are they arranged in any standard order.

The heavy use of these block graphics has been reinforced by the availability of a range of printers



**Timer Chip**
When a computer is switched on, the circuits take a while to stabilise. This timer waits for a fraction of a second, after which it resets the microprocessor to the start of the BASIC interpreter

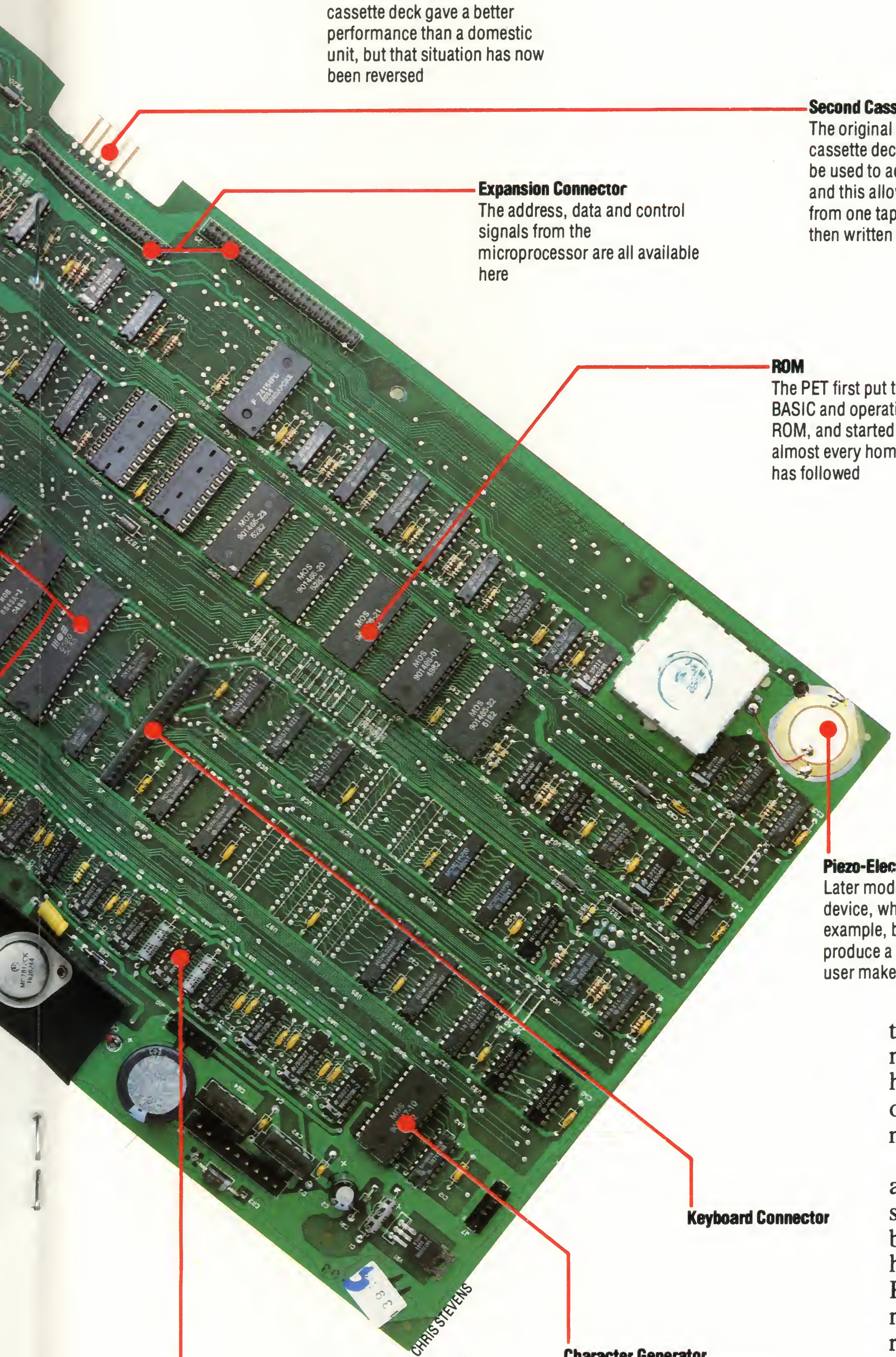
**User Port**
This interface contains a number of useful lines, including an eight-bit parallel port, and connections for interfacing an external monitor. It is particularly suitable for interfacing home-designed electronics projects

**IEEE488 Port**
The PET was the only one of the early microcomputers to include this parallel interface. Because it could address up to 15 peripherals, it was used to drive both disks and printers. The IEEE488 is also the standard used for interfacing scientific laboratory equipment

**6522**
This Versatile Interface Adaptor is similar to the 6520, but contains a shift register for converting between serial and parallel data, as well as two programmable timers that can be used to control external equipment

**6520**
These PIAs (Peripheral Interface Adaptors) take care of most of the interfacing, including the cassettes and keyboard

**6502**
The PET was designed by Commodore's Chuck Peddle, so it is hardly surprising that it is based on a 6502 microprocessor, which he also designed. Though business computers have opted for other processors, the 6502 still remains popular amongst home computers



**Cassette Port**
A specially modified Commodore cassette deck must be used. When the PET was first introduced, the Commodore cassette deck gave a better performance than a domestic unit, but that situation has now been reversed

**Second Cassette Port**
The original PETs had a built-in cassette deck. Now this port can be used to add a second unit, and this allows data to be read from one tape, modified, and then written to another

**Expansion Connector**
The address, data and control signals from the microprocessor are all available here

**ROM**
The PET first put the complete BASIC and operating system in ROM, and started a trend that almost every home computer has followed

**Piezo-Electric Speaker**
Later models incorporated this device, which could, for example, be programmed to produce a 'warble' when the user makes an erroneous entry

**Keyboard Connector**

**Character Generator**
In addition to 64 alphanumeric characters, the PET can generate 64 graphics symbols. Alternatively, text can be displayed in upper and lower case

**RAM**
PETs come with anything from 8 Kbytes to 32 Kbytes as standard. By means of a special modification this can be extended to 96 Kbytes

## COMMODORE PET

**PRICE**
From about £300

**SIZE**
480×440×300mm

**CPU**
6502

**CLOCK SPEED**
1 MHz

**MEMORY**
32 Kbytes RAM
20 Kbytes ROM

**VIDEO DISPLAY**
25 lines of 40 characters. Built-in 12″ (30 cm) green phosphor monitor. 256 displayable characters and graphics symbols, or low resolution (50×80) graphics

**INTERFACES**
IEEE488, 8-bit parallel user port, cassette (2)

**LANGUAGE SUPPLIED**
BASIC, Machine Language Monitor

**OTHER LANGUAGES AVAILABLE**
PASCAL, COMAL, LISP

**COMES WITH**
Instruction manual

**KEYBOARD**
Typewriter-style keyboard, featuring 64 individual keys with graphics symbols inscribed on the front. A separate numeric keypad includes calculator function keys

**DOCUMENTATION**
Commodore have never been acclaimed for the quality of their documentation, although this has much improved since the early days

that will reproduce them in hard copy without the need for complex bit programming of the printer head. Of course, this means that a limited number of printers are suitable for use with the PET, and most, if not all, are Commodore products.

As a result of these various idiosyncrasies, and although there is a considerable amount of software available for the machine, little of this has been translated to other machines. Few programs have been converted from other machines to the PET as well, because they generally involve too much effort to convert, and it is easier simply to rewrite them. Consequently, the machine has become somewhat 'isolated' in its own little world, and is scarcely affected by changes in the industry as a whole. Though the PET's days of glory are now over, it still remains a popular machine in schools, and home computer manufacturers would do well not to forget the features of the PET that really triggered off the microcomputer revolution.

# Subversive Elements

**With careful planning and a step-by-step approach, the time taken to de-bug a program can be dramatically reduced**

As you become more skilled at writing programs, you will also tend to become more accomplished at 'de-bugging' them. The syntactical mistakes and errors in logic, which even the most experienced computer programmers can make, become less frequent and less problematic as your experience increases. Here are some hints to help you avoid programming errors and become more efficient at de-bugging your code.

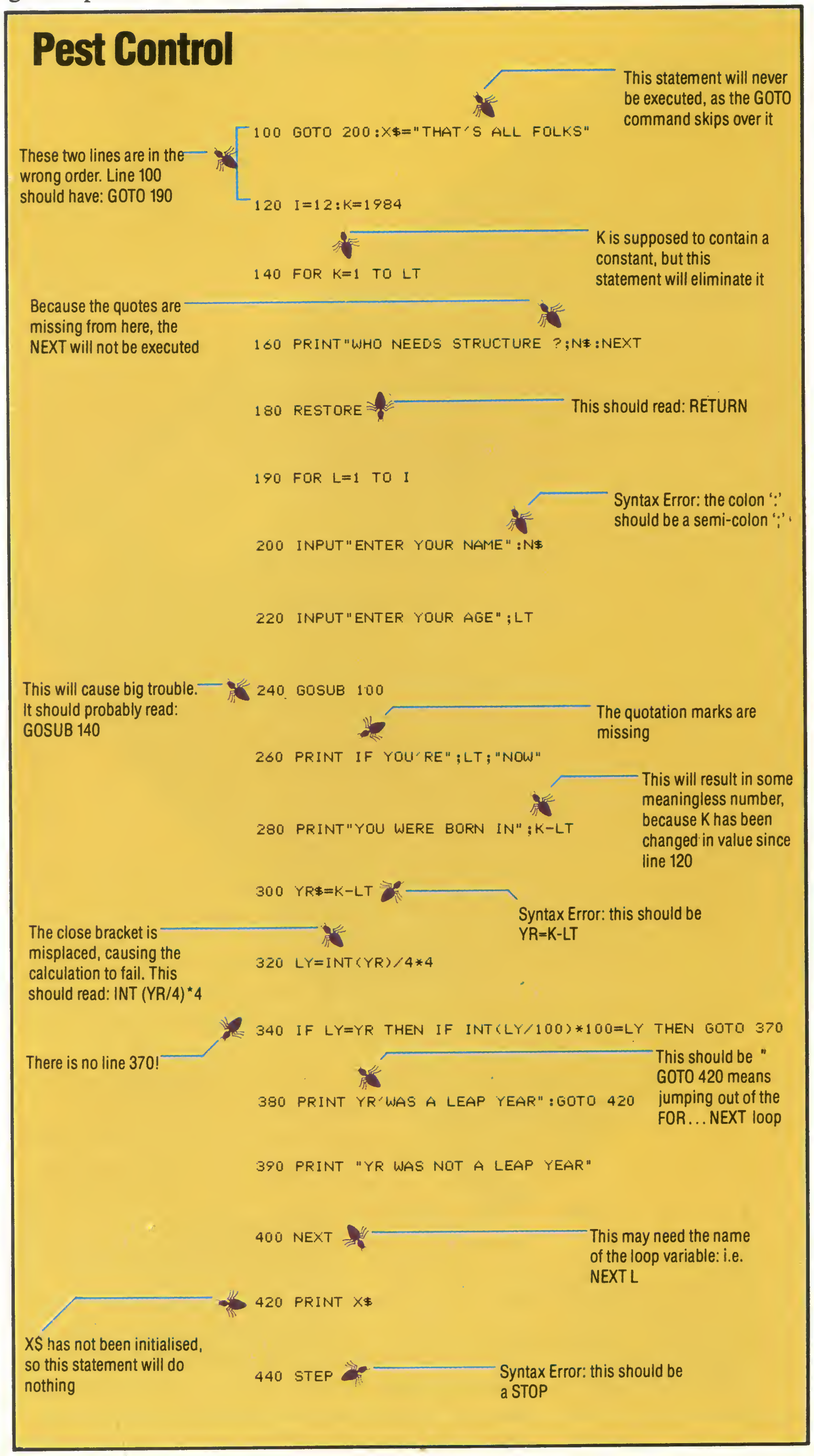
The first place to begin is at the precise point where a program begins — in your head! If the concept of a program is badly thought out at the beginning, then it is sure to be infested with bugs when it is written.

It is a far better idea to begin writing a program by first stating the problem as clearly as possible to yourself or someone else. Then divide the problem into logically complete parts — Input, Output, Algorithms, Data Structures, Processes, etc. — and consider each of those parts as a separate problem. If necessary, break down each of these problems into its component problems, and so on, until the original problem is a structured collection of sub-problems, each of which is easy for you to program. A formal approach, such as using a pseudo-language or a flowchart, is essential in the design stage as a way of keeping track of, and preserving, the program structure as a whole. You must try to stay away from the keyboard until you can honestly say that you know how to program every part of the problem This is called the top-down approach to programming, and the method can dramatically cut your de-bugging time.

Splitting problems into solvable tasks will lead you to write programs that are really collections of subroutines or procedures linked by a skeleton main program. This makes finding bugs easier, and it enables you to build a library of bug-free subroutines for use in later programs. The alternative is called 're-inventing the wheel': every time you write a program that sorts data, for example, you re-solve the problem of how to write a sort routine, and probably rewrite the same old bugs, as well! It is much easier to write and debug it once, save it, and recall it whenever you need it thereafter.

As far as BASIC allows, always try to use appropriate variable names, even if they have to be abbreviated. NET=GROSS−TAX, for example, explains itself; and NT=GR−TX isn't a bad substitute; but N=G−T is extremely ambiguous, and gives no clue as to what variables are involved. It's good practice to keep a variable table, which shows you all the variables used in the program and what they're for. This can lead you to standardise your use of variables (such as, always using certain single letter variables as loop counters), and stops you using the same variable for different purposes. Similarly, it's good practice to store constant values in

## Pest Control



```
100 GOTO 200:X$="THAT'S ALL FOLKS"
120 I=12:K=1984
140 FOR K=1 TO LT
160 PRINT"WHO NEEDS STRUCTURE ?;N$:NEXT
180 RESTORE
190 FOR L=1 TO I
200 INPUT"ENTER YOUR NAME":N$
220 INPUT"ENTER YOUR AGE";LT
240 GOSUB 100
260 PRINT IF YOU'RE";LT;"NOW"
280 PRINT"YOU WERE BORN IN";K-LT
300 YR$=K-LT
320 LY=INT(YR)/4*4
340 IF LY=YR THEN IF INT(LY/100)*100=LY THEN GOTO 370
380 PRINT YR'WAS A LEAP YEAR":GOTO 420
390 PRINT "YR WAS NOT A LEAP YEAR"
400 NEXT
420 PRINT X$
440 STEP
```

This statement will never be executed, as the GOTO command skips over it

These two lines are in the wrong order. Line 100 should have: GOTO 190

K is supposed to contain a constant, but this statement will eliminate it

Because the quotes are missing from here, the NEXT will not be executed

This should read: RETURN

Syntax Error: the colon ':' should be a semi-colon ';'

This will cause big trouble. It should probably read: GOSUB 140

The quotation marks are missing

This will result in some meaningless number, because K has been changed in value since line 120

Syntax Error: this should be YR=K-LT

The close bracket is misplaced, causing the calculation to fail. This should read: INT (YR/4)*4

There is no line 370!

This should be " GOTO 420 means jumping out of the FOR... NEXT loop

This may need the name of the loop variable: i.e. NEXT L

X$ has not been initialised, so this statement will do nothing

Syntax Error: this should be a STOP

variables at the start of the program, and refer back to these variables thereafter. This makes the program faster and neater, and it means that you can change these values without having to hunt through the program for every occurrence.

Even with the sort of formal approach that we have outlined here, it's difficult to eliminate bugs entirely, so it's important to adopt a disciplined method for finding and eradicating them. The commonest bugs are syntax errors, and you can usually correct them as soon as you encounter them. But this is not always the case. Consider:

```
10 PRINT"BIG BUGS HAVE LITTLE BUGS UPON"
20 PRINT"THEIR BACKS TO BITE THEM"
```

Such lines often cause an error message when executed if they're not keyed in as two separate lines. Line 10 contains 40 characters, so when you type it on a 40-column screen, the cursor finishes up at the start of the next screen line, which can cause you to forget to hit RETURN on line 10 before you start typing line 20. If so, then what look like two perfect lines in your program will actually be one line with a syntax error (the number 20) in the middle of it. One way of trapping these errors is to list suspect lines individually rather than as part of a piece of program.

Error messages, when they're not incomprehensible, can be misleading. Take for example:

```
25 DATA 10.2,34,56.9,0.008,15.6
30 FOR K=1 TO 5: READ N(K):NEXT K
```

This may fail to execute because of an alleged syntax error in line 30; whereas the error is actually in the data on line 25 (One of the zeros has been mis-keyed as the letter O).

Coding errors that don't result in syntax errors are the commonest bugs, and usually also the hardest to find. In this case, it is vital to be methodical. Begin by trying to find out roughly where the bug is in the program. This is reasonably easy with well-structured modular programs, and can be made easier by the TRACE utility, which causes the current program line number to be printed on the screen as it is executed. If your machine doesn't allow this, then you can create TRACE statements periodically throughout the program (PRINT "LINE 150" at the beginning of line 150, for example). Similarly, you can use the STOP command to halt program execution at significant places in the program so that you can examine the values of crucial variables. You can do this in direct mode using PRINT, or you can write a subroutine onto the end of your program:

```
11000 REM PRINT THE VARIABLES
11100 PRINT"SCORE,SIZE,FLAGS"
11200 PRINT SC;SZ;F1;F2
11300 PRINT"BOARD ARRAY"
11400 FOR K=1 TO 10:PRINT BD$(K):NEXT K
```

Consequently, when the program comes across a STOP command, you can type GOTO 11000, and have the current state of the variables displayed. You can even change them (by typing, say, SZ=17 and pressing RETURN), and then restart the program with the CONTinue command.

When you've found that the bug is lurking within certain lines, or in a particular variable, then you should be close to eliminating it, but tread carefully! Try one remedy at a time so that you can see what its exact effect on execution is. It's very easy to make several changes between runs, perhaps getting rid of one bug, but creating one or more new ones, and then forgetting exactly what it was you did!

Loops and branches, especially when they're nested, are particularly fertile ground for bugs, and require special care in both writing and de-bugging. Consider this piece of code:

```
460 IF SM<0 AND SC<>-1 THEN IF SC>0 OR
    SM=SC-F9 THEN LT=500
470 FOR C1=1 TO LT:FOR C2=LT TO C1 STEP-1
480 SC=SM+SC*C2
490 NEXT C2:SM=0:NEXT C1
```

What does this all mean? Even if you know what it's meant to do, would you know if it were succeeding or failing? Putting statements inside a loop when they should be outside is a sure way to encourage bugs. And so is failing to cover all possible conditions when writing IF . . . THEN statements. A special case of this occurs when you write multiple statements after IF . . . THEN. For example:

```
655 IF A$="" THEN GOTO 980:A$=B$
660 PRINT A$
```

The statement A$=B$ will never be executed because either A$="", in which case control passes to line 980, or A$<>"", in which case the rest of line 655 is ignored.

Experience is the best teacher of de-bugging, but a step-by-step approach and a disciplined method are invaluable aids. Take your time, and — above all — DON'T PANIC!

## Early Bug

To new programmers, bugs often seem to take on animate characteristics, such as hiding from the programmer and deliberately undermining all his efforts to find them. However, the first bug (at least the one from which the term is derived) really was animate. In trying to eliminate an error from a program she was developing on the Harvard Mrk II in 1945, Captain Grace Hopper discovered that a large moth had got caught up in the electromechanical working of the computer and was causing the fault. As a result of that incident, the term 'de-bugging' was coined

TONY LODGE

# Laser Show

## Optical (laser) disc technology opens up two major applications for home computers: interactive video and mass storage

Whenever one overhears a conversation about home computers, the first statistic quoted is invariably that of memory size. Certainly, the internal storage capacity of the computer is important, but the capacity of its mass storage system is likely to prove more critical in the long term. After a couple of months, the enthusiastic home computer user will have accumulated a considerable number of cassettes, or several boxes of disks. Yet most of these programs are never modified, and they would be better stored in ROM cartridges than on delicate magnetic media. What would be very useful is some form of digital storage system that was read-only like a cartridge, but had a much greater capacity.

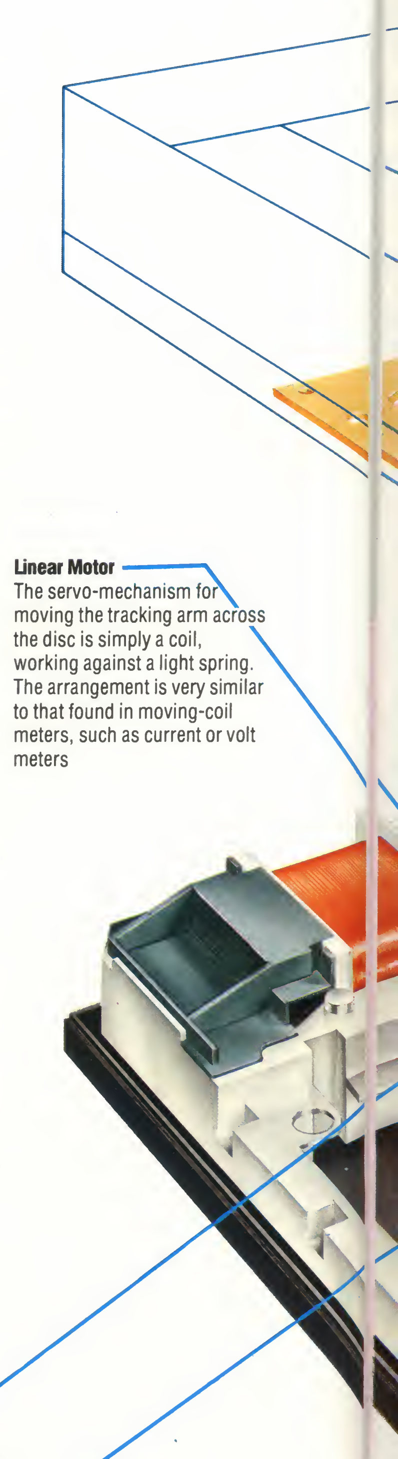
Such a system does exist— in the form of the optical laser disc. Currently, though, this system is used in the home only as an alternative to the video cassette recorder for showing pre-recorded material. Another use of the same technology is the compact audio disc, which is replacing the turntable and stylus format of hi-fi systems.

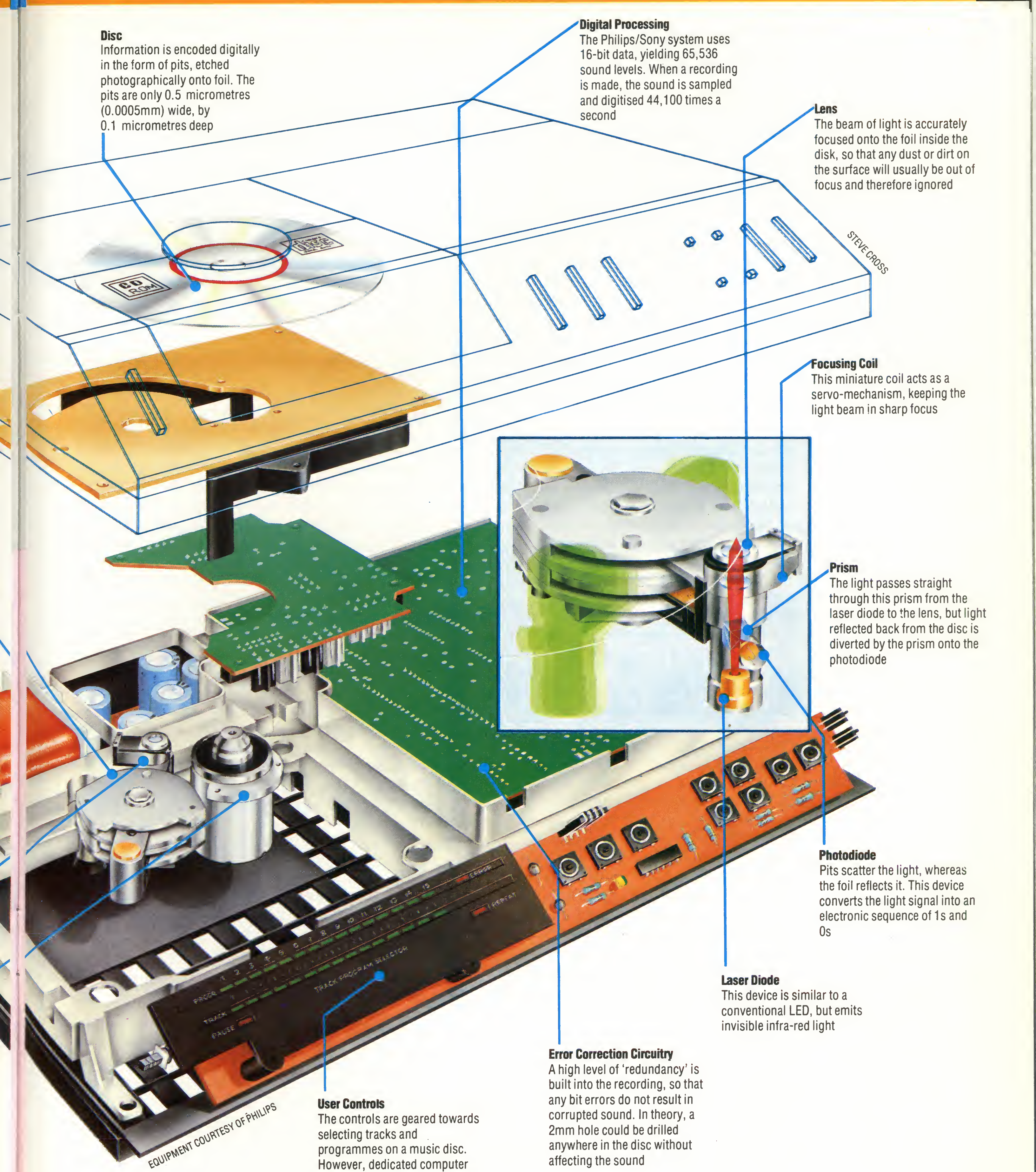
The difference between these two types of systems (apart from the diameters of their discs) is in their methods of operation. Whereas a video disc is an analogue system, a compact audio disc stores its information in digital form — i.e. as a sequence of ones and zeros. This information is turned back into the original audio signal by a digital-to-analogue convertor, which is the electronic opposite of the process that created the information in the first place. Because there are so many stray electric fields in the domestic environment, it is impractical to use magnetic media like floppy disks for video recording. In any case, the amount of information on an optical disc can run into millions of megabytes, and that is much more than even a Winchester disk can hold.

There are several optical laser disc systems available, but the most successful to date is that introduced by Philips. This system uses a 14 inch (35 cm) plastic disc, which is really only a protective envelope. The information itself is buried deep inside the plastic as a series of pits in a sheet of metal foil. As on a floppy disk, the stored information is catalogued on the video disc, so that, given the right sort of disc player, it is possible to move instantly to any single piece of information. Once the read head is in the desired location, the information is read back from the disc by the laser beam. The light passes through the plastic and falls on the surface of the metal foil. A light sensitive cell then reads the information as the light is reflected from the pits in the foil. The information is recorded on a single spiral track, with one frame of the video for every revolution. This gives a total of 54,000 frames on each side of the disc, or 36 minutes of playing time.

The main potential uses for optical discs in the field of computers fall into two areas. The first, and already available, development is that of 'interactive video'. A transmitted television programme is non-interactive — the viewer has no control over the order in which the scenes are presented. With interactive video, however, textual and visual information is stored on a video disc, which is connected to a computer. The disc can then be used as a reference library, with the displayed text superimposed over the video pictures on a conventional television screen. In response to prompts from the computer, the user can select specific 'tracks' or 'scenes' on the video disc to be played. Alternatively, the disc can be used as a training aid, with live action or stills being displayed on a television and the trainee's answers to relevant questions input to the computer, which can monitor and report on the user's performance. Interfaces between a domestic video disc and a home computer are still not widely available, though many enthusiasts have constructed their own. Philips do, however, market a professional model of their LaserVision, which can cope with interactive video on its own, or can interface with a computer by means of an IEEE488 or RS232 port.

The other area in which optical disc technology is likely to be exploited is the provision of computer software. Imagine, for example, the advantages of supplying a computer with all its systems software — word processor, database, spreadsheet, and several dozen games — on a single, incorruptible disk. This is likely to take the format of the compact audio disc, but as yet no compact disc player has been fitted with a computer interface. With such a huge market potential it is reasonable to expect domestic compact disc players with such interfaces within a very short time, as well as dedicated compact disc players for personal computers. Sony and Philips have already announced their intention to produce a dedicated disc player for computers, called CDROM.


**Linear Motor**
The servo-mechanism for moving the tracking arm across the disc is simply a coil, working against a light spring. The arrangement is very similar to that found in moving-coil meters, such as current or volt meters
**Tracking Arm**
The arm is pivoted centrally, and is both finely balanced and freely pivoted. The reading head consequently traces an arc across the disc
**Motor**
The rotation speed of the disc is very accurately controlled using feedback circuitry. As the arm moves from the inside to the outside of the disc, the speed will change from 500 to 200 rpm to keep the recording density constant



**Disc**
Information is encoded digitally in the form of pits, etched photographically onto foil. The pits are only 0.5 micrometres (0.0005mm) wide, by 0.1 micrometres deep

**Digital Processing**
The Philips/Sony system uses 16-bit data, yielding 65,536 sound levels. When a recording is made, the sound is sampled and digitised 44,100 times a second

**Lens**
The beam of light is accurately focused onto the foil inside the disk, so that any dust or dirt on the surface will usually be out of focus and therefore ignored

**Focusing Coil**
This miniature coil acts as a servo-mechanism, keeping the light beam in sharp focus

**Prism**
The light passes straight through this prism from the laser diode to the lens, but light reflected back from the disc is diverted by the prism onto the photodiode

**Photodiode**
Pits scatter the light, whereas the foil reflects it. This device converts the light signal into an electronic sequence of 1s and 0s

**Laser Diode**
This device is similar to a conventional LED, but emits invisible infra-red light

**Error Correction Circuitry**
A high level of 'redundancy' is built into the recording, so that any bit errors do not result in corrupted sound. In theory, a 2mm hole could be drilled anywhere in the disc without affecting the sound

**User Controls**
The controls are geared towards selecting tracks and programmes on a music disc. However, dedicated computer peripherals using CD (Compact Disc) technology will be available in the future

# Finishing Touches

**By removing the anomalies caused by stringing together the modules, and adding a few more facilities, our address book program is now complete**

In the last instalment of the course, readers were left with the problem of working out why running the address book program, then adding a record (using *ADDREC*), then locating a record (using *FINDREC*), and then exiting from the program (using *EXPROG*) would result in the added record not being saved. The problem arose through the use of the variable RMOD as a flag to indicate that a record had been modified (implying that the file might be out of order). The *SRTREC* subroutine would sort the file into alphabetical order, and then set RMOD to 0 on the assumption that the file is in order. Executing *EXPROG* checked to see if the file was in order (RMOD = 0) and didn't bother to save the file if it was in a sorted condition.

Adding a record (using *ADDREC*) would set RMOD to 1 (since a record had been modified, i.e. a new record had been added), but *SRTREC* would set RMOD to 0, indicating that the file had been sorted. What is really needed, however, irrespective of whether the file has been sorted or not, is a flag that signals that a record has been modified and a separate flag to show if the file is in a sorted condition or not. Then, subroutines that need to know that the file is sorted can check the 'sorted' flag, and subroutines that need to know if any record has been modified can check the 'modified' flag.

Suitable names for the two flags would be RMOD, to show if a record has been modified, and SRTD, to show if the file has been sorted.

When the program was presented on page 399, line 1230 contained the statement LET SVED = 0. The SVED variable has not been used so far, but when the line was included, it was realised that RMOD alone would not be enough. The variable name SVED was chosen with the idea that certain conditions would have to be true before a save (to tape or disk) would be necessary.

A more appropriate name for this flag would be SRTD (to indicate that the file is in a sorted condition). The original line 1230 has been changed to:

```
1230 LET SRTD = 1
```

There are now four possible states regarding the condition of the data file. These are:

| RMOD | SRTD | |
|---|---|---|
| 0 | 0 | Not modified, not sorted (illegal) |
| 1 | 0 | Modified, not sorted |
| 0 | 1 | Not modified, sorted |
| 1 | 1 | Modified, sorted |

RMOD=0 and SRTD=0 is illegal because the program ensures that the data file is always sorted before it is saved. When the program is run, RMOD is set to 0 (line 1220) to indicate that no modifications have taken place, and SRTD is set to 1 (line 1230) to indicate that the file is sorted.

Any operation that modifies a record (such as *ADDREC*, *DELREC* or *MODREC*) sets RMOD to 1 and this flag is not reset by any subsequent operation. SRTD, which is initially set to 1, is reset to 0 by any activity that might mean the data has become out of order (such as in *MODREC* if the name field is altered). Any activity that needs to assume the data is sorted (such as *FINDREC*) always checks SRTD and calls the sort routine if SRTD = 0. By using these two flags, instead of just RMOD, the program is able to terminate without saving the data file if no modifications have taken place during the current run of the program. It will not be 'tricked into' terminating without saving if a sort takes place after a record modification.

The other variable not used so far is CURR. This variable is used to save the 'current' position in the array of a record after one has been located by the search routine. CURR is not cleared after a value has been assigned to it; it is used to carry information about the target record to other routines in the program. The end of the *FINDREC* (search) routine has been modified in lines 3320 and 3330 to set the value of CURR: to 0 if the search failed to find the target record; and to MID if the search was successful.

Line 13340 branches to the *NOTREC* subroutine if CURR is 0. This displays a message saying that the record has not been found and displays the search key ,NAMFLD$(SIZE). *NOTREC* returns to the main menu after the space bar has been pressed. *NOTREC* could be modified quite easily to give the user the opportunity to:

```
PRESS RETURN TO TRY AGAIN OR
SPACE BAR TO CONTINUE
```

It might appear that the easiest way to achieve this would be to call *FINDREC* again if RETURN were pressed. However, calling a subroutine from within itself, whilst not illegal in BASIC, 'confuses' the return address and will cause the subroutine to be repeated again even when you don't want it to. There are ways of getting round this problem, but the programming starts to get a bit tricky!

An easier way would be to have used a flag (such as NREC for not record) and reset it in *NOTREC*, allow the subroutine to return in the normal way, and force a jump back to *EXECUT* in the main program, for example: 95 IF NREC=0 THEN 80. This approach was tried, and worked. But the coding started to look untidy. In accordance with our principle of avoiding GOTOs, we decided to keep things simple and just return to the main menu if a record is not found by *FNDREC*.

A small addition to the line 10490 in *MODNAM* should be noted. Numeric variable S should also be reset (LET S=0). Failure to do so can, under certain unusual circumstances, cause *MODNAM* to malfunction.

The other routine implemented in this final version of the program is *MODREC*. This routine first locates the record to be modified by calling *FNDREC* (line 14120). This line calls line 13030, not 13000, in order to suppress *FNDREC*'s clear screen statement. If the record cannot be located, the program will return to the main menu in the usual way (in line 14130). If the record is located, the target record is left displayed on the screen and users are instructed to:

```
MODIFY NAME?
PRESS RETURN TO ENTER NEW NAME
OR SPACE BAR FOR NEXT FIELD
```

The routine that finds out which of the two options is required can be found in lines 14190 to 14280.

Lines 14190 to 14220 constitute a simple loop that terminates only if either the space bar or RETURN is pressed. If A$ is NOT CHR$(13) (the ASCII value for a carriage return) AND NOT a space (you could also use CHR$(32) instead of " ") I will be reset and the loop will repeat. If the key pressed was RETURN (i.e. the name field is to be changed) the next few lines will fill the NAMFLD$(CURR) with the new name, set RMOD, reset SRTD, call *MODNAM* and fill MODFLD$(CURR) with the standardised name created by *MODNAM* and located in MODFLD$(SIZE).

The rest of *MODNAM* works in exactly the same way. Note, however, that modifying the other fields does set RMOD but does not reset SRTD (see line 14490, for example). The reason for this is that only changing the name field implies that the data file may be out of order, since the file is ordered by name. Changing any other field merely indicates that a record has been changed (RMOD = 1) and that the file must be saved when the program is terminated.

The other routine implemented is *DELREC* — to delete a record. This is very straightforward. First it clears the screen (line 15020) and displays a message explaining what's going on. It then calls *FINDREC* to locate the record to be deleted. A choice is then offered: to press RETURN to delete the record or the SPACE BAR to return to the main menu. A warning message is also displayed (line 15160). An even better approach might be to respond with an ARE YOU SURE? message if RETURN is pressed and then only delete the record if the Y key is pressed (i.e. IF INKEY$ = "Y" THEN ...).

*DELREC* does not reset the SRTD flag. Since the file is already in alphabetical order by name, deleting a complete record will not upset this order. It does, however, mean that the file has been modified and so RMOD is reset in line 15340 and SIZE is reduced by one in line 13550 to take account of the fact that the file now has one fewer valid records. All the records are moved 'down one' in lines 15260 to 15320.

You may also have noticed that *FNDREC* includes a conditional call to a subroutine called *LSTCUR* to print out the CURRent record located by *FNDREC*. If you don't have a printer, simply replace line 13540 with a REM for future implementation and omit lines 13600 to 13690.

This completes the address book program. We have carried out all the major options presented in the main menu: finding a record, adding a record, changing a record, deleting a record, and exiting from the program. The purpose of the computerised address book has been to illustrate how a programmer should set about specifying, designing and implementing a program. An essential modification by anyone who intends the program as a piece of application software will be to check for — and trap — the problem that would arise if SIZE were ever to equal 51. This would happen as soon as there were 50 records in the file.

In the next instalment of the Basic Programming course we will discuss programming style and cover a few of the more advanced aspects of the BASIC language.

## Basic Flavours

**LPRINT**

This command is not available on the Commodore 64, Vic-20, BBC Micro, or Dragon 32.

On the BBC Micro with a parallel printer insert the following lines:

```
13605 VDU 2
13680 VDU 3
```

These enable and disable the printer in turn. Substitute PRINT for LPRINT in lines 13610 to 13670. For more information see the user manual.

On the Commodores insert these lines:

```
13605 OPEN 4,4:CMD 4
13680 PRINT # 4: CLOSE 4
```

These enable and disable the printer in turn. Substitute PRINT for LPRINT in lines 13610 to 13670.

On the Dragon 32 insert these lines:

```
13605 OPEN"O", -2
13680 CLOSE -2
```

These enable and disable the printer in turn. Substitute PRINT -2, (the comma here is part of the command) for LPRINT in lines 13610 to 13670.

**ZX81 SPECTRUM**

The address book program will be published in full in the next instalment of the Basic Programming course.

## Address Book Program

```
10 REM  'MAINPG'
20 REM  *INITIL*
30 GOSUB 1000
40 REM  *GREETS*
50 GOSUB 3000
60 REM  *CHOOSE*
70 GOSUB 3500
80 REM  *EXECUT*
90 GOSUB 4000
100 IF CHOI <> 9 THEN 60
110 END
1000 REM  *INITIL* SUBROUTINE
1010 GOSUB 1100: REM  *CREARR* (CREATE ARRAYS) SUBROUTINE
1020 GOSUB 1400: REM  *RDINFL* (READ IN FILE) SUBROUTINE
1030 GOSUB 1600: REM  *SETFLG* (SET FLAGS) SUBROUTINE
1040 REM
1050 REM
1060 REM
1070 REM
1080 REM
1090 RETURN
1100 REM *CREARR* (CREATE ARRAYS) SUBROUTINE
1110 DIM NAMFLD$(50)
1120 DIM MODFLD$(50)
1130 DIM STRFLD$(50)
1140 DIM TWNFLD$(50)
1150 DIM CNTFLD$(50)
1160 DIM TELFLD$(50)
1170 DIM NDXFLD$(50)
1180 REM
1190 REM
1200 REM
1210 LET SIZE = 0
1220 LET RMOD = 0
1230 LET SRTD = 1
1240 LET CURR = 0
1250 REM
1260 REM
1270 REM
1280 REM
1290 REM
1300 RETURN
1400 REM *RDINFL* SUBROUTINE
1410 OPEN "I",#1,"ADBK.DAT"
1420 INPUT #1,TEST$
1430 IF TEST$ = "@FIRST" THEN GOTO 1540: REM  CLOSE AND RETURN
1440 LET NAMFLD$(1) = TEST$
1450 INPUT #1,MODFLD$(1),STRFLD$(1),TWNFLD$(1),CNTFLD$(1),TELFLD$(1)
1460 INPUT #1,NDXFLD$(1)
1470 LET SIZE = 2
1480 FOR L = 2 TO 50
1490 INPUT #1,NAMFLD$(L),MODFLD$(L),STRFLD$(L),TWNFLD$(L),CNTFLD$(L)
1500 INPUT #1,TELFLD$(L),NDXFLD$(L)
1510 LET SIZE = SIZE + 1
1520 IF EOF(1) = -1 THEN LET L = 50
1530 NEXT L
1540 CLOSE #1
1550 RETURN
1600 REM  *SETFLG* SUBROUTINE
1610 REM  SETS FLAGS AFTER *RDINFL*
1620 REM
1630 REM
1640 IF TEST$ = "@FIRST" THEN LET SIZE = 1
1650 REM
1660 REM
1670 REM
1680 REM
1690 RETURN
3000 REM  *GREETS* SUBROUTINE
3010 PRINT CHR$(12):REM  CLEAR SCREEN
3020 PRINT
3030 PRINT
3040 PRINT
3050 PRINT
3060 PRINT TAB(12);"*WELCOME TO THE*"
3070 PRINT TAB(9);"*HOME COMPUTER COURSE*"
3080 PRINT TAB(6);"*COMPUTERISED ADDRESS BOOK*"
3090 PRINT
3100 PRINT TAB(5);"(PRESS SPACE-BAR TO CONTINUE)"
3110 FOR L = 1 TO 1
3120 IF INKEY$ <> " " THEN L = 0
3130 NEXT L
3140 PRINT CHR$(12)
3150 RETURN
3500 REM  *CHOOSE* SUBROUTINE
3510 REM
3520 IF TEST$ = "@FIRST" THEN GOSUB 3860: REM  *FIRSTM* SUBROUTINE
3530 IF TEST$ = "@FIRST" THEN RETURN
3540 REM  'CHMENU'
3550 PRINT CHR$(12)
3560 PRINT "SELECT ONE OF THE FOLLOWING"
3570 PRINT
3580 PRINT
3590 PRINT
3600 PRINT "1. FIND RECORD (FROM NAME)"
3610 PRINT "2. FIND NAMES (FROM INCOMPLETE NAME)"
3620 PRINT "3. FIND RECORDS (FROM TOWN)"
3630 PRINT "4. FIND RECORD (FROM INITIAL)"
3640 PRINT "5. LIST ALL RECORDS"
3650 PRINT "6. ADD NEW RECORD"
3660 PRINT "7. CHANGE RECORD"
3670 PRINT "8. DELETE RECORD"
3680 PRINT "9. EXIT & SAVE"
3690 PRINT
3700 PRINT
3710 REM  'INCHOI'
3720 REM
3730 LET L = 0
3740 LET I = 0
3750 FOR L = 1 TO 1
3760 PRINT "ENTER CHOICE (1 - 9)"
3770 FOR I = 1 TO 1
3780 LET A$ = INKEY$
3790 IF A$ = "" THEN I = 0
3800 NEXT I
3810 LET CHOI = VAL(A$)
3820 IF CHOI <1 THEN L = 0
3830 IF CHOI >9 THEN L = 0
3840 NEXT L
3850 RETURN
3860 REM *FIRSTM* SUBROUTINE (DISPLAY MESSAGE)
3870 LET CHOI = 6
3880 PRINT CHR$(12): REM  CLEAR SCREEN
3890 PRINT
3900 PRINT TAB(8);"THERE ARE NO RECORDS IN"
3910 PRINT TAB(8);"THE FILE. YOU WILL HAVE"
3920 PRINT TAB(6);"TO START BY ADDING A RECORD"
3930 PRINT
3940 PRINT TAB(5);"(PRESS SPACE-BAR TO CONTINUE)"
3950 FOR B = 1 TO 1
3960 IF INKEY$ <> " " THEN B = 0
3970 NEXT B
3980 PRINT CHR$(12): REM CLEAR SCREEN
3990 RETURN
4000 REM  *EXECUT* SUBROUTINE
4010 REM
4020 REM
4030 REM
4040 IF CHOI = 1 THEN GOSUB 13000: REM  *FNDREC*
4050 REM  2 IS *FNDNMS*
4060 REM  3 IS *FNDTWN*
4070 REM  4 IS *FNDINT*
4080 REM  5 IS *LSTREC*
4090 IF CHOI = 6 THEN GOSUB 10000: REM  *ADDREC*
4100 IF CHOI = 7 THEN GOSUB 14000: REM  *MODREC*
4110 IF CHOI = 8 THEN GOSUB 15000: REM  *DELREC*
4120 IF CHOI = 9 THEN GOSUB 11000: REM  *EXPROG*
4130 REM
4140 RETURN
10000 REM *ADDREC* SUBROUTINE
10010 PRINT CHR$(12): REM  CLEAR SCREEN
10020 INPUT "ENTER NAME";NAMFLD$(SIZE)
10030 INPUT "ENTER STREET";STRFLD$(SIZE)
10040 INPUT "ENTER TOWN";TWNFLD$(SIZE)
10050 INPUT "ENTER COUNTY";CNTFLD$(SIZE)
10060 INPUT "ENTER TELEPHONE NUMBER";TELFLD$(SIZE)
10070 LET RMOD = 1: LET SRTD = 0: REM MODIFIED & NOT SORTED
10080 LET NDXFLD$(SIZE) = STR$(SIZE)
10090 LET TEST$ = ""
10100 GOSUB 10200: REM  *MODNAM*
10110 LET CHOI = 0
10120 LET SIZE = SIZE + 1
10130 REM
10140 REM
10150 RETURN
10200 REM  *MODNAM* ROUTINE
10210 REM  CONVERTS CONTENTS OF NAMFLD$ TO UPPER CASE,
10220 REM  REMOVES RUBBISH, AND STORES IN THE ORDER:
10230 REM  SURNAME+SPACE+FORENAME IN MODFLD$
10240 REM
10250 LET N$ = NAMFLD$(SIZE)
10260 FOR L = 1 TO LEN(N$)
10270 LET TEMP$ = MID$(N$,L,1)
10280 LET T = ASC(TEMP$)
10290 IF T >= 97 THEN T = T - 32
10300 LET TEMP$ = CHR$(T)
10310 LET P$ = P$ + TEMP$
10320 NEXT L
10330 LET N$ = P$
10340 REM  LOCATE LAST SPACE
10350 FOR L = 1 TO LEN(N$)
10360 IF MID$(N$,L,1) = " " THEN S = L
10370 NEXT L
10380 REM  REMOVE RUBBISH AND STORE FORENAME
10390 REM  IN CNAM$
10400 FOR L = 1 TO S - 1
10410 IF ASC(MID$(N$,L,1)) > 64 THEN CNAM$ = CNAM$ + MID$(N$,L,1)
10420 NEXT L
10430 REM  REMOVE RUBBISH AND STORE SURNAME
10440 REM  IN SNAM$
10450 FOR L = S + 1 TO LEN(N$)
10460 IF ASC(MID$(N$,L,1)) > 64 THEN SNAM$ = SNAM$ + MID$(N$,L,1)
10470 NEXT L
10480 LET MODFLD$(SIZE) = SNAM$ + " " + CNAM$
10490 LET P$ = "": LET N$ = "": LET SNAM$ = "": LET CNAM$ = "":
      LET S = 0
10500 RETURN
11000 REM  *EXPROG* SUBROUTINE
11010 REM  SORTS AND SAVES FILE
11020 REM  IF ANY RECORD HAS BEEN
11030 REM  MODIFIED (RMOD = 1)
11040 REM  OR NOT SORTED (SRTD = 0)
11050 REM  RMOD = 0 AND SRTD = 0 IS ILLEGAL
11060 REM
11070 IF RMOD = 0 AND SRTD = 1 THEN RETURN
11080 IF RMOD = 1 AND SRTD = 0 THEN GOSUB 11200: REM *SRTREC*
11090 GOSUB 12000: REM  *SAVREC*
11100 RETURN
11200 REM  *SRTREC* SUBROUTINE
11210 REM  SORTS ALL RECORDS BY MODFLD$ INTO
11220 REM  ALPHABETICAL ORDER AND UPDATES NDXFLD
11230 REM
11240 REM
11250 LET S = 0
11260 FOR L = 1 TO SIZE - 2
11270 IF MODFLD$(L) > MODFLD$(L + 1) THEN GOSUB 11350
11280 NEXT L
11290 IF S = 1 THEN 11250
11300 REM
11310 REM
11320 LET SRTD = 1: REM SETS 'FILE SORTED' FLAG
11330 REM
11340 RETURN
11350 REM  *SWPREC* SUBROUTINE
11360 LET TNAMFD$ = NAMFLD$(L)
11370 LET TMODFD$ = MODFLD$(L)
11380 LET TSTRFD$ = STRFLD$(L)
```

```
11390 LET TTWNFD$ = TWNFLD$(L)
11400 LET TCNTFD$ = CNTFLD$(L)
11410 LET TTELFD$ = TELFLD$(L)
11420 REM
11430 LET NAMFLD$(L) = NAMFLD$(L + 1)
11440 LET MODFLD$(L) = MODFLD$(L + 1)
11450 LET STRFLD$(L) = STRFLD$(L + 1)
11460 LET TWNFLD$(L) = TWNFLD$(L + 1)
11470 LET CNTFLD$(L) = CNTFLD$(L + 1)
11480 LET TELFLD$(L) = TELFLD$(L + 1)
11490 LET NDXFLD$(L) = STR$(L)
11500 REM
11510 LET NAMFLD$(L + 1) = TNAMFD$
11520 LET MODFLD$(L + 1) = TMODFD$
11530 LET STRFLD$(L + 1) = TSTRFD$
11540 LET TWNFLD$(L + 1) = TTWNFD$
11550 LET CNTFLD$(L + 1) = TCNTFD$
11560 LET TELFLD$(L + 1) = TTELFD$
11570 LET NDXFLD$(L + 1) = STR$(L + 1)
11580 LET S = 1
11590 REM
11600 RETURN
12000 REM  *SAVREC* SUBROUTINE
12010 REM
12020 REM
12030 OPEN "O",#1,"ADBK.DAT"
12040 REM
12050 FOR L = 1 TO SIZE - 1
12060 PRINT #1,NAMFLD$(L);",";MODFLD$(L);",";STRFLD$(L);",";TWNFLD$(L)
12070 PRINT #1,CNTFLD$(L);",";TELFLD$(L);",";NDXFLD$(L)
12080 NEXT L
12090 REM
12100 REM
12110 REM
12120 REM
12130 CLOSE #1
12140 REM
12150 RETURN
13000 REM  *FNDREC* (FIND RECORD) SUBROUTINE
13010 PRINT CHR$(12): REM  CLEAR SCREEN
13020 REM
13030 IF SRTD = 0 THEN GOSUB 11200: REM  *SRTREC*
13040 PRINT
13050 PRINT
13060 PRINT TAB(9);"SEARCHING FOR A RECORD"
13070 PRINT TAB(16);"BY NAME"
13080 PRINT
13090 PRINT TAB(9);"TYPE IN THE FULL NAME"
13100 PRINT TAB(7);"IN FIRSTNAME SURNAME ORDER"
13110 PRINT
13120 PRINT
13130 REM
13140 INPUT "NAME IS ";NAMFLD$(SIZE)
13150 GOSUB 10200: REM  *MODNAM* SUBROUTINE
13160 LET SCHKEY$ = MODFLD$(SIZE)
13170 REM
13180 REM
13190 REM
13200 REM
13210 REM
13220 LET BTM = 1
13230 LET TOP = SIZE - 1
13240 FOR L = 1 TO 1
13250 LET MID = INT((BTM + TOP)/2)
13260 IF MODFLD$(MID) <> SCHKEY$ THEN L = 0
13270 IF MODFLD$(MID) < SCHKEY$ THEN BTM = MID + 1
13280 IF MODFLD$(MID) > SCHKEY$ THEN TOP = MID - 1
13290 IF BTM > TOP THEN L = 1
13300 NEXT L
13310 REM
13320 IF BTM > TOP THEN LET CURR = 0
13330 IF BTM <= TOP THEN LET CURR = MID
13340 IF CURR = 0 THEN GOSUB 13700: REM  *NOTREC*
13350 IF CURR = 0 THEN RETURN
13360 REM
13370 REM
13380 PRINT CHR$(12)
13390 PRINT
13400 PRINT TAB(13);"*RECORD FOUND*"
13410 PRINT
13420 PRINT "NAME:",NAMFLD$(CURR)
13430 PRINT "STREET:",STRFLD$(CURR)
13440 PRINT "TOWN:",TWNFLD$(CURR)
13450 PRINT "COUNTY:",CNTFLD$(CURR)
13460 PRINT "PHONE:",TELFLD$(CURR)
13470 PRINT
13480 PRINT TAB(7);"PRESS ANY LETTER TO PRINT"
13490 PRINT TAB(7);"OR SPACE-BAR TO CONTINUE"
13500 FOR I = 1 TO 1
13510 LET A$ = INKEY$
13520 IF A$ = "" THEN I = 0
13530 NEXT I
13540 IF A$ <> " " THEN GOSUB 13600: REM  *LSTCUR*
13550 RETURN
13600 REM  *LSTCUR* (LIST CURRENT RECORD) SUBROUTINE
13610 LPRINT
13620 LPRINT "NAME:",NAMFLD$(CURR)
13630 LPRINT "STREET:",STRFLD$(CURR)
13640 LPRINT "TOWN:",TWNFLD$(CURR)
13650 LPRINT "COUNTY:",CNTFLD$(CURR)
13660 LPRINT "PHONE:",TELFLD$(CURR)
13670 LPRINT
13680 LPRINT
13690 RETURN
13700 REM  *NOTREC* (RECORD NOT FOUND) SUBROUTINE
13710 PRINT CHR$(12): REM CLEAR SCREEN
13720 PRINT TAB(11);"*RECORD NOT FOUND*"
13730 PRINT TAB(4);"*IN THE FORM: ";NAMFLD$(SIZE);" *"
13740 PRINT
13750 PRINT TAB(5);"(PRESS SPACE-BAR TO CONTINUE)"
13760 FOR I = 1 TO 1
13770 IF INKEY$ <> " " THEN I = 0
13780 NEXT I
13790 RETURN
14000 REM  *MODREC* (MODIFY RECORD) SUBROUTINE
```

```
14010 REM
14020 PRINT CHR$(12): REM  CLEAR SCREEN
14030 PRINT
14040 PRINT
14050 PRINT
14060 PRINT
14070 PRINT TAB(10);"*TO MODIFY A RECORD*"
14080 PRINT TAB(3);"*FIRST LOCATE THE DESIRED RECORD*"
14090 REM
14100 REM
14110 REM
14120 GOSUB 13030: REM *FNDREC* SUBROUTINE WITHOUT CLS
14130 IF CURR = 0 THEN RETURN: REM  RECORD NOT FOUND
14140 PRINT
14150 PRINT TAB(14);"MODIFY NAME?"
14160 PRINT
14170 PRINT TAB(5);"PRESS RETURN TO ENTER NEW NAME"
14180 PRINT TAB(6);"OR SPACE-BAR FOR NEXT FIELD"
14190 FOR I = 1 TO 1
14200 LET A$ = INKEY$
14210 IF A$ <> CHR$(13) AND A$ <> " " THEN I = 0
14220 NEXT I
14230 IF A$ = CHR$(13) THEN INPUT "NEW NAME";NAMFLD$(CURR)
14240 IF A$ = CHR$(13) THEN RMOD = 1
14250 IF A$ = CHR$(13) THEN SRTD = 0
14260 IF A$ = CHR$(13) THEN NAMFLD$(SIZE) = NAMFLD$(CURR)
14270 IF A$ = CHR$(13) THEN GOSUB 10200: REM  *MODNAM* SUBROUTINE
14280 IF A$ = CHR$(13) THEN LET MODFLD$(CURR) = MODFLD$(SIZE)
14290 PRINT
14300 PRINT TAB(13);"MODIFY STREET?"
14310 PRINT
14320 PRINT TAB(5);"PRESS RETURN TO ENTER NEW STREET"
14330 PRINT TAB(6);"OR SPACE-BAR FOR NEXT FIELD"
14340 FOR I = 1 TO 1
14350 LET A$ = INKEY$
14360 IF A$ <> CHR$(13) AND A$ <> " " THEN I = 0
14370 NEXT I
14380 IF A$ = CHR$(13) THEN RMOD = 1
14390 IF A$ = CHR$(13) THEN INPUT "NEW STREET";STRFLD$(CURR)
14400 PRINT
14410 PRINT TAB(13);"MODIFY TOWN?"
14420 PRINT
14430 PRINT TAB(5);"PRESS RETURN TO ENTER NEW TOWN"
14440 PRINT TAB(6);"OR SPACE-BAR FOR NEXT FIELD"
14450 FOR I = 1 TO 1
14460 LET A$ = INKEY$
14470 IF A$ <> CHR$(13) AND A$ <> " " THEN I = 0
14480 NEXT I
14490 IF A$ = CHR$(13) THEN RMOD = 1
14500 IF A$ = CHR$(13) THEN INPUT "NEW TOWN";TWNFLD$(CURR)
14510 PRINT
14520 PRINT TAB(12);"MODIFY COUNTY?"
14530 PRINT
14540 PRINT TAB(4);"PRESS RETURN TO ENTER NEW COUNTY"
14550 PRINT TAB(6);"OR SPACE-BAR FOR NEXT FIELD"
14560 FOR I = 1 TO 1
14570 LET A$ = INKEY$
14580 IF A$ <> CHR$(13) AND A$ <> " " THEN I = 0
14590 NEXT I
14600 IF A$ = CHR$(13) THEN RMOD = 1
14610 IF A$ = CHR$(13) THEN INPUT "NEW COUNTY";CNTFLD$(CURR)
14620 PRINT
14630 PRINT TAB(8);"MODIFY TELEPHONE NUMBER?"
14640 PRINT
14650 PRINT "PRESS RETURN TO ENTER NEW TELEPHONE NUMBER"
14660 PRINT TAB(8);"OR SPACE-BAR TO CONTINUE"
14670 FOR I = 1 TO 1
14680 LET A$ = INKEY$
14690 IF A$ <> CHR$(13) AND A$ <> " " THEN I = 0
14700 NEXT I
14710 IF A$ = CHR$(13) THEN RMOD = 1
14720 IF A$ = CHR$(13) THEN INPUT "NEW NUMBER";TELFLD$(CURR)
14730 REM
14740 REM
14750 RETURN
15000 REM  *DELREC* (DELETE RECORD) SUBROUTINE
15010 REM
15020 PRINT CHR$(12): REM  CLEAR SCREEN
15030 PRINT
15040 PRINT
15050 PRINT
15060 PRINT
15070 PRINT TAB(10);"*TO DELETE A RECORD*"
15080 PRINT TAB(3);"*FIRST LOCATE THE DESIRED RECORD*"
15090 REM
15100 REM
15110 REM
15120 GOSUB 13030: REM  *FNDREC* SUBROUTINE WITHOUT CLS
15130 IF CURR = 0 THEN RETURN: REM  RECORD NOT FOUND
15140 PRINT
15150 PRINT TAB(3);"DO YOU WANT TO DELETE THIS RECORD?"
15160 PRINT TAB(5);"*WARNING* -- NO SECOND CHANCES"
15170 PRINT
15180 PRINT TAB(9);"PRESS RETURN TO DELETE"
15190 PRINT TAB(8);"OR SPACE-BAR TO CONTINUE"
15200 FOR I = 1 TO 1
15210 LET A$ = INKEY$
15220 IF A$ <> CHR$(13) AND A$ <> " " THEN I = 0
15230 NEXT I
15240 IF A$ = " " THEN RETURN
15250 FOR L = CURR TO SIZE - 2
15260 LET NAMFLD$(L) = NAMFLD$(L + 1)
15270 LET MODFLD$(L) = MODFLD$(L + 1)
15280 LET STRFLD$(L) = STRFLD$(L + 1)
15290 LET TWNFLD$(L) = TWNFLD$(L + 1)
15300 LET CNTFLD$(L) = CNTFLD$(L + 1)
15310 LET TELFLD$(L) = TELFLD$(L + 1)
15320 LET NDXFLD$(L) = STR$(L)
15330 NEXT L
15340 LET RMOD = 1
15350 LET SIZE = SIZE - 1
15360 REM
15370 REM
15380 REM
15390 RETURN
```

# Grace Hopper

**Grace Hopper was largely responsible for the development of high level languages, and identifying the first bug!**

COURTESY OF SPERRY LTD

Computer science is generally regarded as a strictly male preserve. But, increasingly, women are taking their place alongside men, as equals, in the development and application of computers. A woman pioneer of computing was Grace Hopper, whose most significant contributions were in the field of software — she created the first compiler and helped invent the language COBOL. But she was also the first person to isolate a 'bug' in a computer, and successfully 'de-bug' it.

After doing postgraduate work at Yale, Grace Hopper returned to her original university, Vassar, as a member of the mathematics faculty. Here she remained until the age of 39, when she was called up for war service with the Naval Ordinance Computation Project. In 1945, she was ordered to go to Harvard University to assist a physicist, Howard Aiken, in the building of a computer. Aiken had approached IBM in 1937 with the idea of constructing a computer using adapted tabulating equipment. His first computer, although mechanical in design, was successful enough to encourage IBM to invest in an improved model that would use electromechanical relays. The machine that was subsequently developed was known as the Harvard Mark II.

In these early days, machines had to be programmed by rewiring them for each new task. Thus, in the hot summer of 1945, Grace Hopper found herself literally enmeshed in the wiring of the computer. Ballistic computing facilities were urgently needed for the war effort, and Commander Aiken would often come into the workshop and demand: 'Why aren't you making numbers, Hopper?' After one troublesome breakdown of the computer, when the fault was eventually found to be a moth that had flown in through the open windows and been hammered to death in a relay switch, Grace tersely replied: 'We are debugging the machine!' This first recorded 'bug' was carefully removed from the relay with a pair of tweezers and is preserved at the Naval Museum in Virginia in the log book of the Harvard Mark II. It is glued beside the entry for 15.45 on 9 September 1945.

In the same year another computer, ENIAC (see page 46), was being built by two engineers, John Mauchly and Presper Eckert. After the war, the two men set up their own business to manufacture a commercial version of the machine, and invited Grace to join their team. Her main contribution to the development of this computer, called UNIVAC (UNIVersal ACcounting machine), was in creating software for it. And it was during her attempts to construct programs for business use on UNIVAC that Grace first sought out ways to short-cut the need for rewriting certain subroutines that recurred over and over again. By employing what was then considered the remarkable idea that a computer could write its *own* programs, Grace created the first programming language, together with the compiler needed to translate it into machine code. This was given the name 'A–O'. When this compiler was first presented it caused incredulity amongst computer professionals who thought their machines could only perform arithmetic or manipulate symbols. They were amazed to see a computer jump to a subroutine in its library store on encountering an imperative verb at the beginning of what was almost a normal English sentence.

In May 1959, Captain Hopper was invited by the Pentagon to join a working committee that was to attempt to create and standardise a single language for computers in commercial use. In less than a year the committee produced the first version of the COmmon Business Oriented Language (COBOL). Grace contributed a great deal to the committee's attempt to distil the best aspects of each of the existing languages and thus create a language acceptable to the industry through its sheer quality. It is a measure of the success of the committee's work that COBOL is still one of the most widely used languages today.

**COBOL**

COBOL was one of the first programming languages to be written with the intention of making it easily accessible to non-mathematicians. The language encourages the use of generalised procedures written in a narrative style of English, rather than coded routines peculiar to a particular problem.

A program in COBOL is built out of four units. The name of the program, its author, and other reference information comprise the Identification division. Although COBOL programs are intended to be portable (capable of being used on many machines), any details describing the particular computer for which the program was originally written are noted in the Environment division. Since the same data may be used in many parts of the same program, COBOL has a separate Data division. Finally, the procedures that are to operate on the data are listed in the Procedure division

# THE HOME COMPUTER COURSE BINDER

Now that your collection of Home Computer Course is growing, it makes sound sense to take advantage of this opportunity to order the two specially designed Home Computer Course binders.

The binders have been commissioned to store all the issues in this 24 part series.

At the end of the course the two volume binder set will prove invaluable in converting your copies of this unique series into a permanent work of reference.

**Buy two together and save £1.00**

✻ Buy volumes 1 and 2 together for £6.90 (including P&P). Simply fill in the order form and these will be forwarded to you with our invoice.

✻ If you prefer to buy the binders separately please send us your cheque/postal order for £3.95 (including P&P). We will send you volume 1 only. Then you may order volume 2 in the same way – when it suits you!

**Overseas readers**: This binder offer applies to readers in the UK, Eire and Australia only. Readers in Australia should complete the special loose insert in Issue 1 and see additional binder information on the inside front cover. Readers in New Zealand and South Africa and some other countries can obtain their binders **now**. For details please see inside the front cover.

Binders may be subject to import duty and/or local tax.

**NEXT TO YOUR COMPUTER...YOUR COURSE MANUALS**

# Mentathlete

Home computers. Do they send your brain to sleep – or keep your mind on its toes?

At Sinclair, we're in no doubt. To us, a home computer is a mental gym, as important an aid to mental fitness as a set of weights to a body-builder.

Provided, of course, it offers a whole battery of genuine mental challenges.

The Spectrum does just that.

Its education programs turn boring chores into absorbing contests – not learning to spell 'acquiescent', but rescuing a princess from a sorcerer in colour, sound, and movement!

The arcade games would test an all-night arcade freak – they're very fast, very complex, very stimulating.

And the mind-stretchers are truly fiendish. Adventure games that very few people in the world have cracked. Chess to grand master standards. Flight simulation with a cockpit full of instruments operating independently. Genuine 3D computer design.

No other home computer in the world can match the Spectrum challenge – because no other computer has so much software of such outstanding quality to run.

For the Mentathletes of today and tomorrow, the Sinclair Spectrum is gym, apparatus and training schedule, in one neat package. And you can buy one for under £100.

sinclair