LET YOUR BBC MICRO TEACH YOU TO PROGRAM

by Tim Hartnell

More than 40 programs are included in this book which leads you through programming on the BBC Microcomputer step by step from first principles.

LET YOUR BBC MICRO TEACH YOU TO PROGRAM

by Tim Hartnell

Introduction

Welcome to this book on the BBC Microcomputer. Even if you knew nothing about computers and programming when you begin, you should end up with quite a degree of knowledge by the time you finish working through it. And if you already know some BASIC, you should know even more by the time you turn the last page.

I've assumed that you don't know anything about programming, so some of the material in here may seem a little simple to you. I make no apology for this, as there is a time in all our programming lives when we know nothing. We have all had to start from square one at some stage or another.

I suggest you read through the material in this book in the order in which it is presented. The development from topic to topic is carefully graded, so working through the book, and — most importantly — entering the material into your computer as you read, in the sequence in which it is presented here should assist you in developing your programming skills. Special thanks to Jeremy Ruston who provided a substantial quantity of material for the latter part of the book, to Roger Munford who prepared extensive notes on many of the functions of the BBC Microcomputer, and to Graham Charlton for Microsoft versions of some of the programs.

This book has been a pleasure to write, because the BBC Microcomputer 'co-operates well' with a programmer, and the computer shows the result of careful design. Your BBC Microcomputer should be a companion for years to come. I hope this book will assist you in making the most of your new machine.

Tim Hartnell,
London W12, 1982

LET YOUR BBC MICRO TEACH YOU TO
PROGRAM

First published in Great Britain by:

INTERFACE PUBLICATIONS,
44 - 46 Earls Court Road,
LONDON W8 6EJ

(c) Hartnell, 1982

First printing - June, 1982
Second printing - December, 1982
Third printing - February, 1983

ISBN 0907563 14 7

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, except for the sole use by the purchaser of this volume, without the prior, written permission of the copyright holder. No warranty in respect of the contents of this volume, or their suitability for any purpose, is expressed or implied.

Any enquiries regarding this book should be directed by mail to the publisher.

INTERFACE PUBLICATIONS also publishes THE BBC MICRO REVEALED, by Jeremy Ruston, available from the above address for £9.95 (inc. p & p).

The PRINT Statement

PRINT is probably the most-used command in BASIC. It is the command which allows the computer to communicate with the user.

Type the following line into your computer, and then press RETURN:

```
PRINT 5
```

You'll see that the computer obediently prints the number five. You can use the PRINT command to make your computer act as a calculator. Enter the following and press RETURN:

```
PRINT 5 + 3
```

When you press RETURN, you'll see it prints up the correct result. This 'direct calculation mode' can work out problems as complex as you wish. Try the following (remembering to press ENTER to make the computer act on what you've typed in):

```
PRINT SQR (8 + 1)
```

This asks the computer to PRINT the square root (that's what SQR means) of the sum of the numbers in brackets, that is, the square root of nine. If your computer is functioning correctly, you should have got an answer of three. Don't worry about the SQR at this time. We will come to it later on.

So you can see that PRINT can be used in the direct mode to print out numbers, and the results of calculations. It can also print out words. Type in the following, then press

RETURN:

```
PRINT HI THERE
```

Instead of happily printing HI THERE, the computer comes up with what is called an error message. In this case, the error message reads *No such variable*. We'll look at variables in due course, but for now, keep in mind that if we want the computer to print out words, the words must be enclosed within quote marks. Enter and run (that is, press RETURN after typing it in) the following:

```
PRINT "HI THERE"
```

You"ll see the words HI THERE appear on the line below the one which you typed in.

To recap quickly: Simply used as a command, typing PRINT 2 + 3 will tell the computer to print out the result of that addition. Entering PRINT "WORDS" will get the computer to print out everything which is within the quote marks.

Computers use programs, and it is now time to write our first, simple program. Enter and run the following:

```
10 REM PROGRAM ONE
20 PRINT"THIS IS A DEMONSTRATION"
30 PRINT 1
40 PRINT 2
50 PRINT "THIS IS THE END"
```

When you RUN this (which you do by typing in the word RUN, then pressing RETURN), you should see:

```
>RUN
THIS IS A DEMONSTRATION
          1
          2
THIS IS THE END
```

While we have this program in the computer, let's learn a little more about programs. Type in LIST, then press RETURN. You'll see the program listing comes back. Notice that every line starts with a line number. The first line, in this case number 10, starts with the word REM. REM is computer talk for REMARK, and is used in a program when you want to explain what is going on in within a program, or what a program is (as in this case), so that when you return to it later, you'll know what is going on. The computer ignores REM statements when it comes to them.

A REM statement is made up of a line number, then the word REM, and some text. The message can be made up from almost anything you like — letters, numbers or punctuation marks — although it is best to keep the messages as clear and brief as you can. Although anything typed after the word REM is ignored by the computer when it is running a program, a REM line still uses up memory.

REM statements are often like the following:

10 REM THIS WORKS OUT THE SCORE

10 REM FIND THE ANGLE

There is no reason why there should be just one REM statement, but if the commentary you wish to add to a particular area of a program is one which may take up more than one line of text, it is important to place the word REM at the beginning of each new line. For example:

60 REM THE MULTIPLICATION ROUTINE IN WHICH

70 REM THE TWO VARIABLES A AND B

80 REM ARE MULTIPLIED TOGETHER

So long as each REMark line starts with the word REM, the computer will ignore the text that follows on that line (although the complete program listing, REMs and all, will be printed if a LIST is requested). If you forget to place REM at the beginning of what is intended to be a REM statement, the computer will assume the line is a statement it does not understand, and reward you with an error message. Type in the number 10, then press RETURN, then type in LIST, and press RETURN again. You'll see your program reappear as follows:

```
20 PRINT"THIS IS A DEMONSTRATION"
30 PRINT 1
40 PRINT 2
50 PRINT "THIS IS THE END"
```

Line 10 has disappeared. It is very easy to get rid of lines you don't want in a computer, just by typing in the relevant line number, then pressing RETURN. Try typing in line 10 again, as it was before, but leaving out the REM statement. Enter the following line, then run the program:

10 PROGRAM ONE

You'll get the error message 'Mistake at line 10', because the computer does not know what to do when it comes across the word PROGRAM. It has nothing within its *instruction set* which corresponds to the word PROGRAM.

Editing

We can put the word REM back in, and illustrate how the BBC Microcomputer's editing facilities work. Type in 1Ø REM, then go to the key with the arrow on it above the word RETURN. Press this once, and you'll see a solid square appear after the word REM and a flashing line (the cursor) move up a line. Keep pressing the arrowed key until you reach the line which reads 1Ø PROGRAM ONE. You will probably find that the flashing cursor is underneath the space after the number 1Ø. When you've done this, press the COPY key (under the RETURN key) and hold it down until the words PROGRAM ONE are 'copied' after the word REM in your new line 1Ø. Now press RETURN. This may sound confusing if you're not reading this book while seated at your computer, but should make sense if you are. Test that the new line is in place by typing in LIST, and then pressing RETURN. The program should reappear, with the word REM in the right position in line 1Ø. Check that it is correct by entering RUN, then pressing RETURN.

Just to recap, the REM statement has two uses. One is to act as an aid so you and your friends can untangle programming webs you've woven. The second use is to act as a separator between individual routines in a large program; to split it into visually separate blocks.

At the end of the book is a chapter called 'Improving your programs' which discusses the concept of 'structured programming'. The use of REMs in this second way is described in more detail at that point.

LIST and RENUMBER

LIST is the BASIC command which we use to get the computer to print out the whole of the program it is currently holding. You can simply enter L., instead of the whole word. Try it now. Type in L. and press RETURN. The program should list out in full. All lines in the program are LISTed in numerical order, rather than in the order in which they were entered into the computer. The computer automatically sorts its lines into order. Enter the following, press RETURN, and then LIST the program again:

15 PRINT "THIS IS A NEW LINE"

You'll see it stays at the bottom of the listing when you first press RETURN, but after typing in L. and press RETURN again, you'll see it snuggled into its proper position, as follows:

```
10 REM PROGRAM ONE
15 PRINT "THIS IS A NEW LINE"
20 PRINT"THIS IS A DEMONSTRATION"
30 PRINT 1
40 PRINT 2
50 PRINT "THIS IS THE END"
```

The BBC Microcomputer has a very useful facility to renumber lines. You can number lines with any interval you choose (like starting at one, then going up in ones), but this does not leave any space for additional lines to be inserted. If you type in RENUMBER, then press RETURN, then LIST, then RETURN again, you'll see the following:

```
10 REM PROGRAM ONE
20 PRINT "THIS IS A NEW LINE"
30 PRINT"THIS IS A DEMONSTRATION"
40 PRINT 1
50 PRINT 2
60 PRINT "THIS IS THE END"
```

As you can see, line 15 has been changed to line 2Ø, and all the subsequent lines have been renumbered. RENUMBER is a useful command to remember, and you can use it whenever you need additional space between lines, or before you SAVE a program on cassette, or print it out on a printer.

If you simply type LIST, the entire program will be listed by the computer. However, by varying the listing format, a number of useful

variations can be achieved. These allow you to examine individual portions of the program. Let's add a few more lines before we do so. Add the following:

```
 70 PRINT "HI"
 80 PRINT "THERE"
 90 REM HI THERE
100 PRINT 4
```

LIST 1Ø,5Ø will only allow the lines number 1Ø to 5Ø to be listed. Any lines numbered higher or lower than this will not be included.

LIST 7Ø will only list line 7Ø

LIST ,7Ø will print all the lines up to, and including, line 7Ø

LIST 7Ø, will print out all the lines above line 7Ø, including line 7Ø

You can halt a listing at any point by pressing ESCAPE.

RUN, STOP, END, NEW, OLD

As you've no doubt realised, the RUN command is used to start the computer operating on a program which you have entered into the computer, either by typing it in, or by loading a program in from casssette. The computer executes all the lines stored in its memory, starting from the lowest number, and working through in order. Various commands, as you shall see shortly, can make the program loop back on itself, but in essence, the computer works through a program in line number order, unless told to do otherwise.

If you want the program to STOP at a particular point, you can use, naturally enough, a command called STOP. Enter 35 STOP, press RETURN, then run the program. It will print out:

THIS IS A NEW LINE

THIS IS A DEMONSTRATION

Then, after a blank line, will be the message "STOP at line 35". BBC BASIC includes another word which does exactly the same thing, but

does not print out an error statement. Change line 35 to 35 END, press RETURN, then run the program again. You'll find it prints out the two lines as before, but does not print out a message like "END at line 35". You can use these commands wherever you wish a program to terminate. Other computers are not as tolerant as this. Some, like the Acorn Atom, demand that an END statement ends each program, and responds with a beep and an 'Error' message if the END is left off. You do not need to worry on the BBC Micro. The END is optional.

We'll return to look at PRINT In a little more detail in a moment, but there is one more (actually, two associated) command I'd like to discuss first, NEW and OLD.

The command NEW will erase any program from the computer's memory, and should always be used to remove anything from the memory before you start writing a new program. If you don't do this, and you use different line numbers for the second program, you'll find the lines may well be interwoven with the old program. The NEW command is fairly brutal on most computers, causing it to dramatically forget everything. Try it now on your computer.

Type in NEW, press RETURN, then enter LIST, and RETURN again. You'll find the 'greater than' symbol beside the flashing cursor appears immediately below the word LIST — but no listing. Try LIST 1Ø, then RETURN, and you'll get the same nothing result. However, and this is useful to remember, the BBC Micro has tucked the program away in another part of memory, just in case you change your mind. Type in OLD, then RETURN, then LIST and RETURN again and, as if by magic, the program has been restored. OLD is pretty easy to remember as the opposite of NEW.

PRINT formatting and TAB

To complete our exploration of the PRINT command, type in NEW again to get rid of the program. Now enter and run the following:

```
 10 REM **PRINT FORMATS**
 20 CLS
 30 PRINT
 40 PRINT
 50 PRINT
 60 PRINT "HI"60
 70 PRINT ''' "HI";70
 80 PRINT 1   2   3
 90 PRINT 1,2,3
100 PRINT 1;2;3
110 PRINT;1;2;3
```

Follow this explanation carefully, and you should learn a lot about the way the BBC Micro formats its print output, you can then use what you've learned to arrange output of your own programs as you wish. I'll go through the program line by line, a practice which will follow, from time to time, in the rest of the book.

1Ø	— Title REM statement
2Ø	— CLS is the command to clear the screen
3Ø-5Ø	— Each of these words PRINT, with nothing following, prints a blank line, moving the next print position down a line. This explains the gap at the top of the screen before anything is printed.
6Ø	— This prints the word HI and then, leaving a space, prints the number 6Ø. Note that the number can follow the word but will not necessarily be printed immediately after it, as does happen in the following line.
7Ø	— The three apostraphes (shift the 7 key), as you can see, have exactly the same effect as three blank PRINT lines, and are far more convenient to use. Note also that the 7Ø is printed hard up against the word HI. The semi colon (;) between the close quote marks and the number ensure they will be printed together in this way.
8Ø	— This allows the numbers 1, 2 and 3 to be printed in neat little rows. Note that there must be a space between them or the computer will print them as 123 or 12 and 3 or 1 23.

90 — This line uses commas between the numbers ensuring they will be printed in separate rows.

100 — The semicolons between the numbers ensures that, after the initial space between the left hand side of the screen and the numbers, the numbers will be printed hard up against each other.

110 — The semicolon *before* the first number ensures the printing starts at the extreme left hand edge of the screen.

You can use these within PRINT statements to create the effects you need. Clear the current program with NEW, then enter and run the next program, changing line 30 as shown, to produce the effects shown. Don't worry about what the rest of the program means at this stage. It will be explained shortly.

```
   10 REM **PRINT TWO**
   20 FOR J=1 TO 10
   30 PRINT J
   40 NEXT J
>RUN
          1
          2
          3
          4
          5
          6
          7
          8
          9
         10
```

```
   10 REM **PRINT TWO**
   20 FOR J=1 TO 10
   30 PRINT J;
   40 NEXT J
>RUN
          1         2         3         4
          9        10>
```

```
    10 REM **PRINT TWO**
    20 FOR J=1 TO 10
    30 PRINT ;J;
    40 NEXT J
>RUN
12345678910>
```

```
    10 REM **PRINT TWO**
    20 FOR J=1 TO 10
    30 PRINT ;J;"   ";
    40 NEXT J
>RUN
1   2   3   4   5   6   7   8   9   10   >
```

TAB (for tabulate) is a command which is very useful to combine with the word PRINT. It moves the PRINT position across the number of lines specified in the brackets following the number. Change line 3Ø so it reads as in the following program, and run it:

```
    10 REM **PRINT TWO**
    20 FOR J=1 TO 10
    30 PRINT TAB(J);J
    40 NEXT J
>RUN
 1
  2
   3
    4
     5
      6
       7
        8
         9
          10
```

And then like this:

```
    10 REM **PRINT TWO**
    20 FOR J=1 TO 10
    30 PRINT TAB(3*J);J
    40 NEXT J
>RUN
    1
       2
          3
             4
                5
                   6
                      7
                         8
                            9
                               10
```

We are now going to introduce a program which will be used in a few other places throughout this book to illustrate different aspects of the computer's actions, so I suggest you enter it, even though you may not know what all the lines mean, and then SAVE it on tape, so you can load it in again when we refer to it later (such as in the section on READ and DATA).

```
   10 REM TABULATOR ROCKET RANGE
   20 MODE7
   30 FOR J=10 TO 1 STEP -1
   40 PRINT '''J
   50 T=TIME
   60 REPEAT UNTIL TIME-T>50
   70 NEXT J
   80 REPEAT
   90 Q=RND(29)+1
  100 U=128+RND(5)
  110 FOR rocket=1 TO 7:READ ROCKET$
  120 PRINT TAB(0);CHR$(129);"(";TAB(Q)
      ;CHR$(U);ROCKET$;TAB(36);CHR$(129);")"
  130 NEXT rocket
  140 T=TIME
  150 REPEAT UNTIL TIME-T=2
```

```
160 SPACE =RND(10)
170 SOUND 16,-RND(5)-10,RND(3)-1,RND(20)
180 FOR print=1 TO SPACE
190 PRINT TAB(0);CHR$(129);"(";TAB(37);")"
200 NEXT print
210 RESTORE
220 T=TIME
230 REPEAT UNTIL TIME-T=2
240 UNTIL FALSE
250 DATA "  +  "," +++ "
260 DATA " +++ "," +++ "
270 DATA " +++ "," +++ "
280 DATA " <*> "
```

SAVE

You SAVE programs by typing in the program, connecting up your cassette recorder as shown in the manual, then typing in SAVE followed by the name of the program within quote marks. In this case, I suggest you use the name ROCKET, so you would type in SAVE "ROCKET", then follow the instructions given by the computer.

These mean you must turn your cassette recorder on, and start to record, and once you've done this, press the RETURN key. You'll see the word ROCKET appear, followed by a series of numbers as the computer saves the program in blocks. When it has all been saved, after a few seconds, the computer will beep to tell you it is finished, and the 'greater than' symbol will reappear, followed by the flashing cursor.

I suggest you make a habit of saving each program three times in a row, on a C–12 or C–15 (i.e. computer) cassette, and you only put one program on the side of each tape. Label the tape clearly with the load name (i.e. with ROCKET In this case). Although it may seem wasteful to use up the whole side of a cassette with just one program, recorded three times, the frustration you will save yourself by not having to search through tape after tape for a program you want will more than compensate for using more cassettes than is strictly necessary. The program is recorded three times just in case the tape

gets damaged at some point, or you accidentally erase part of the tape, or — as sometimes happens — one SAVEing of a program just refuses to load properly.

You should clean the recorder's heads frequently (using liquid, not a tape cleaner ribbon in a cassette) to ensure the clearest possible signal is put onto the tape.

Now let's return to our TABULATOR ROCKET RANGE.

TAB (X) and TAB (X,Y)

The important lines for this discussion are 12Ø and 19Ø, as these make use of TAB in printing. Line 12Ø behaves as follows:

TAB (Ø) — This is not strictly necessary, but ensures that the first PRINT is hard against the left hand edge of the screen
CHR$(129) — This ensures that the next thing printed will be red. The colour commands are described in detail later on in the book
"("— This prints a (red) left hand bracket
TAB(Q) — Q is a number between 2 and 3Ø (chosen in line 9Ø) which determines how many spaces across the PRINT position will move
CHR$(U) — This determines the colour of the rocket which will be printed
ROCKET$ — This is part of the rocket, selected by the READ statement in line 11Ø. Don't worry about this at the moment, as we will be using the ROCKET program again when discussing READ/DATA/RESTORE in due course. For now, it is enough to realise that ROCKET$ is part of the series of plus signs which you see in lines 25Ø to 28Ø.
TAB (36) — After the part of the rocket on that line has been printed, the PRINT position moves across to the 36th position on the line, where CHR$(129) turns the following ")" red to put a border down the right hand side of the screen.
Line 190Ø:
TAB(Ø) — As above
CHR$(129);"(" — As above
TAB (37);")" — This moves the PRINT position across to the 37th position, to print the bracket for the right hand border. Line 19Ø is used a random number of times (between one and ten times, chosen in line 16Ø) to put a space between the rockets.

So far we have only used TAB with a single number in brackets after the word. Remember, TAB(A) will move the start of the PRINT position A spaces across a line. You can have the word TAB followed by two numbers, such as TAB(6,10) which will move the PRINT position six spaces across, and 10 down. The top left hand corner of the screen is zero, zero, so TAB(0,0) indicates that the printing will begin in the top left hand corner of the screen. The left hand side of the screen in numbered 0, while the right hand side is 19, 39 or 79. We have been working in Mode 7 to date (the mode your computer is automatically in when you first turn it on), and in this mode, the screen is 40 characters wide, so the position furthest to the right is numbered 39.

The following program, SQUASH, uses TAB(X,Y) to position a ball (line 620) and a bat (line 530). You use the "Z" and "M" keys to move the slide (the bat) at the bottom of the screen right and left respectively. The program keeps track of how long you keep the ball in flight, and gives you a score at the end based on this time. Pressing any key at the end of the game will give you a new game.

You can make the game easier by changing the zero after INKEY$ in line 140 to a five or a 10, and add a delay loop between lines 170 and 180. A delay loop is simply something like 175 FORQ= 1 TO 300:NEXT Q. You may also wish to change the sound the ball makes when it bounces. Simply change the third figure after the word SOUND in lines 650 and 660. Try 120 + RND (130) or 240 + RND(14) for two different effects.

This listing may well look pretty horrifying at the moment. Once you've finished working through this book, you may wish to come back to programs like this, and try and work out what each section of the program is doing. You'll be surprised to see how much of it you can decipher.

```
 10 REM * SQUASH *
 20 REM Move bat using the "Z" and "M" keys
 30 MODE 7
 40 HIGHSCORE=0
 50 PROCset_up
 60 REM****************
 70 REPEAT
 80 PRINT TAB(0,23);CHR$(128+RND(5))
      ;"High score is ";HIGHSCORE
 90 PRINT TAB(0,5);CHR$(RND(5)+128);
      "You have kept the ball in play"
100 PRINT CHR$(RND(5)+128);
    "for ";(TIME DIV 10)/10;" seconds      "
```

```
110 A$=INKEY$(0)
120 *FX 15,1
130 IF A$="Z" OR A$="M" THEN PROCmove_bat
140 PROCmove_ball
150 PRINT TAB(B+11,19);BAT$
160 UNTIL end
170 IF TIME DIV 100>HIGHSCORE
      HIGHSCORE=TIME DIV 100
180 TI=TIME
190 REPEAT UNTIL TIME>TI+90
200 *FX 15,1
210 FOR T=0 TO 3
220 SOUND T+16,-15,RND(100)+100,255
230 NEXT T
240 TIME=0
250 REPEAT UNTIL TIME>150
260 *FX 15,0
270 *FX 12
280 PROCnewgame
290 REM****************
300 DEF PROCset_up
310 LOCAL
320 PRINT TAB(10,10);STRING$(20,CHR$(255)
330 FOR T=0 TO 10
340 PRINT TAB(10,T+10);CHR$(255);
       TAB(30,T+10);CHR$(255)
350 NEXT T
360 BAT$="___"
370 NOBAT$="   "
380 X=1:Y=1:L=1:M=1
390 TIME=0
400 B=10
410 PRINT TAB(11+B,19);BAT$
420 *FX 11,1
430 *FX 12,1
440 VDU 23;8202;0;0;0;
450 ENDPROC
460 REM****************
470 DEF PROCmove_bat
480 IF A$="M" AND B=16 THEN ENDPROC
490 IF A$="Z" AND B=0 THEN ENDPROC
500 PRINT TAB(11+B,19);NOBAT$
```

```
510 IF A$="M" THEN B=B+1
520 IF A$="Z" THEN B=B-1
530 PRINT TAB(11+B,19);BAT$
540 ENDPROC
550 REM***************
560 DEF PROCmove_ball
570 PRINT TAB(11+X,11+Y);" "
580 IF L+X>18 OR L+X<0 THEN L=-L:
        SOUND 1,-15,157,1
590 IF M+Y>8 OR M+Y<0 THEN M=-M:
        SOUND 1,-15,167,1
600 X=X+L
610 Y=Y+M
620 PRINT TAB(11+X,11+Y);"*"
630 IF Y=8 AND (X<B OR X>(B+2)) THEN
        end=TRUE ELSE end=FALSE
640 P=TIME
650 ENDPROC
660 REM***********
670 DEF PROCnewgame
680 PRINT'''''' CHR$(RND(5)+128);
      "Press any key for a new game"
690 A$=GET$
700 CLS
710 GOTO 50
720 ENDPROC
```

You can use TAB with either the PRINT statement or the INPUT statement. Experiment with TAB, and with the use of ; and , in PRINT statements, until you are confident you know what they are doing.

Random numbers

Random numbers are very useful for games playing, for creating designs and sounds, and for changing the colour of PRINT statements. We will look at their use in creating coloured PRINT output in the section on graphics, and at their use in making sounds in the SOUND part of the book. Right now we'll examine the production of random numbers, and use them in a few simple programs.

The BBC Microcomputer allows you to generate random numbers of two types — floating point between zero and one, and integers. Both these types can easily be set to be greater than, or less than, zero (i.e. to be positive or negative numbers).

Enter and run the following to see a range of numbers between zero and one:

```
10 PRINT RND (1)

20 GOTO 10
```

You'll get a list of numbers something like this:

```
0.918826409
0.266114519
0.102944293
0.295019549
3.66730662E-3
0.386568779
0.964225042
0.310764692
0.451954321
0.128487876
0.220589199
0.440212717
0.749411825
0.521860198
0.967153116
0.711896141
0.765480196
0.356654011
0.988525593
0.274666697
```

You'll find that random integers are often of far more use than are these numbers between zero and one. Most computers demand a quite complex statement (like INT(RND(1)*30) +1) to get random whole numbers, but a routine to do this already exists within the computer. Change the 1 in line 10 to a 10, so it reads PRINT RND(10), and run the program again. You're like to get a series of numbers such as the following:

```
 8
 7
 1
 8
 3
 9
 7
 9
 6
 7
 3
10
 8
 2
 8
 6
 7
 2
 2
 3
```

The computer takes the number in brackets (known as the *argument* of the function) and selects numbers at random between one and that number. To get negative random numbers, just put a minus sign in front of the word RND. Try that, and run it again, to get a result like this:

```
 -1
 -1
 -6
 -6
 -9
 -4
 -6
-10
 -5
```

```
-4
-10
-5
-5
-3
-10
-7
-9
-4
-2
-3
```

You can use the random number generator for any application where you need to emulate a random activity in the real world, like the distribution of weeds in a garden, the spread of clouds in the sky, or the result of rolling die. The next program emulates the role of a six-sided dice. Enter and run it a few times. Note that the single quote mark in line 3Ø is found on the 7 key.

```
10 REM *DICE ROLLER*
20 CLS
30 INPUT ''' "HOW MANY TIMES
          WILL I ROLL THE DIE?  "A
40 FOR B=1 TO A
50 PRINT RND(6)
60 NEXT B
```

```
>RUN
HOW MANY TIMES WILL I ROLL THE DIE?  7
         4
         2
         5
         5
         4
         2
         5
```

While the theoretical distribution of numbers between one and six with a six-sided die suggests each number has an equal chance of coming up in a long, long series of rolls, the totals produced when you use two dice, approaches the following distribution:

Total Showing	No. of ways it can be thrown	Probability	Percentage
2	1	1/36	2.77%
3	2	2/36, 1/18	5.55%
4	3	3/36, 1/12	8.33%
5	4	4/36, 1/9	11.11%
6	5	5/36	13.88%
7	6	6/36, 1/6	16.66%
8	5	5/36	13.88%
9	4	4/36, 1/9	11.11%
1Ø	3	3/36, 1/12	8.33%
11	2	2/36, 1/18	5.55%
12	1	1/36	2.77%

To test just how random the random number generator is, enter and run the following program, which rolls two dice as many times as you request.

```
 10 REM *TWO DICE*
 20 CLS
 30 INPUT '''"HOW MANY TIMES WILL I
              ROLL THE DICE? "D
 40 CLS
 50 DIM A(12)
 60 FOR C=1 TO D
 70 DIE1=RND(6)
 80 DIE2=RND(6)
 90 PRINT TAB(2,3),DIE1,DIE2
100 SUM=DIE1 + DIE2
110 A(SUM)=A(SUM)+1
120 FOR B=2 TO 12
130 PRINT TAB(5,B+3),B
    ,A(B),INT(A(B)/C*100);"%"
140 NEXT B
150 NEXT C
```

Here's the result of one run when I got the computer to roll the two dice over 700 times. The top two figures (3 and 4) are the result of the current roll. Next, from there down, the numbers 2 to 12 in the left hand column are the totals we are looking for, followed by the number of times that total has been rolled in the current run. The final column shows the approximate percentage distribution of each of the totals.

```
 3          4
 2         23          3%
 3         34          4%
 4         58          7%
 5         90         12%
 6        105         14%
 7        133         17%
 8         91         12%
 9         75         10%
10         73          9%
11         38          5%
12         26          3%
```

You can see that this run has approach the theoretical distribution fairly closely, which suggests the random number generator is performing its task properly. The numbers produced are not totally random, but are from a very long list of numbers, so long that no pattern can be discerned.

Acey Duecy

Here's a very simple game which shows the random number generator in action. You can see from the sample run how to play it. The game is not really much of a game, but entering and running it is well worth while. Once you've played a few rounds of the game return to this book for a discussion of the program. You should be pleasantly surprised at how much you have already learned.

```
 10 REM *ACEY DUECY*
 20 D=20
 30 REPEAT
 40 CLS
 50 A=RND(13)
 60 B=RND(13)
 70 IF ABS(B-A)<2 THEN 50
 80 C=RND(13)
 90 IF A=C OR B=C THEN 80
100 PRINT ''"MY FIRST NUMBER IS ";A
110 PRINT "MY SECOND IS ";B
120 PRINT 'CHR$(133);"YOU HAVE $";D
```

```
130 PRINT ''"HOW MUCH DO YOU BET MY NEXT"
140 PRINT "NUMBER LIES BETWEEN ";A;" AND ";B;"?"
150 INPUT E
160 IF E>D THEN 150
170 IF E<1 PRINT CHR$(129);"COWARD!!"
180 PRINT '"MY NUMBER IS ";C
190 IF E<1 THEN 270
200 IF NOT (C>A AND C<B OR C<A AND C>B) THEN 240
210 PRINT "WELL DONE, YOU WIN $";2*E
220 D=D+2*E
230 GOTO 270
240 PRINT "SORRY, YOU LOSE $";E
250 D=D-E
260 IF D<1 THEN 290
270 A$=GET$
280 UNTIL FALSE
290 PRINT "YOU ARE BROKE"
```

```
>
MY FIRST NUMBER IS 9
MY SECOND IS 11
YOU HAVE $20
HOW MUCH DO YOU BET MY NEXT
NUMBER LIES BETWEEN 9 AND 11?
?3
MY NUMBER IS 13
SORRY, YOU LOSE $3
MY FIRST NUMBER IS 8
MY SECOND IS 5
YOU HAVE $17
HOW MUCH DO YOU BET MY NEXT
NUMBER LIES BETWEEN 8 AND 5?
?2
MY NUMBER IS 10
SORRY, YOU LOSE $2
MY FIRST NUMBER IS 1
MY SECOND IS 3
YOU HAVE $15
HOW MUCH DO YOU BET MY NEXT
NUMBER LIES BETWEEN 1 AND 3?
?0
```

```
COWARD!!
MY NUMBER IS 10
MY FIRST NUMBER IS 10
MY SECOND IS 13
YOU HAVE $15
HOW MUCH DO YOU BET MY NEXT
NUMBER LIES BETWEEN 10 AND 13?
?15
MY NUMBER IS 3
SORRY, YOU LOSE $15
YOU ARE BROKE
```

Let us have a look at this program, line by line:

10	Title REM statement
20	Sets the variable D, which is the amount of money you have to equal $20
30	Starts a REPEAT/UNTIL loop which terminates in line 280
40	Clears the screen
50	Sets the first number (A) equal to a random number between 1 and 13
60	Does the same for B
70	Checks to see if the difference between the two numbers to select two new numbers
80	Chooses a third number between one and 13 (C)
90	Checks to see if C equals A or B and, if so, goes back to line 80 to choose a new number
100–110	Tells the player what the two numbers are, using the two apostraphes after the word PRINT to creat two blank lines
120	Prints out, in purple, the amount of money the player has. In mode 7, preceding a print line with CHR$(133) causes the following material to be printed in purple (magenta).
130–140	Asks the player to enter a bet regarding the likelihood of the third number lying between the first two
160	If the player attempts to bet more money than he or she has, goes back to accept another entry
170	If the player enters a bet less than one, prints the word COWARD in red (using CHR$(129)).
180	Prints out the third number, using an apostraphe to print a black line
190	If the bet was less than $1, goes to line 270 to await a new round of the game
200	Checks to see if the player has lost, and if so, sends action to line 240

210 Congratulates the winner
220 Adds the winning amount (2*E, where E is the amount bet) to the player's money (D)
230 Sends action to line 270 for the next round
240 Prints out the "SORRY, YOU LOSE" message
250 Deducts the loss (E) from the player's money (D)
260 Checks to see if the player has less than $1 and if so goes to the "YOU ARE BROKE" message
270 Waits for any key press
280 Goes back to the line after the word REPEAT, in this case line 40, to clear the screen for a new game

Reading through this explanation a couple of times, and looking carefully at the line or lines it refers to, should teach you quite a bit more about programming. There are a number of specific commands which we will look at in more detail, but you're probably starting to pick up quite a bit at this stage.

Variables

You will have noticed in the previous program that a series of letters were used to represent numbers. The letter A was assigned (in line 50) to a number between one and 13, B was assigned in the same way in line 60 and C was assigned in line 80. The amount of money you had was assigned to variable D in line 20. The letters A, B, C and D in this program are called variables.

There are three basic types of variables: integer numeric, numeric and string (alphanumeric).

Almost any combination of letters and numbers can be used as a variable in BBC BASIC, so long as it begins with a letter, and there are no punctuation marks or symbols within the name. So SMUDGEPOT and D17 are valid variable names, while 2 SMUDGE and 1D7 are not. Numeric variables, letters or combinations of letters and numbers beginning with a letter, are simple to use. You can assign a variable of this type to any number within the computer's numerical range.

As you probably know, the computer uses scientific notation to display numbers larger than nine digits long, with the number as a single digit and eight decimal places, followed by the letter E (for exponention) and the power of ten to which the number is to be multiplied. Enter and run the following demonstration which shows the variable A in use, being assigned to a number which is being multiplied repeatedly by 10, and then printed.

```
100 REM **SCIENTIFIC NOTATION**
110 A=1234
120 A=10*A
130 PRINT A
140 GOTO 120
```

```
     12340
    123400
   1234000
  12340000
 123400000
  1.234E9
 1.234E10
 1.234E11
 1.234E12
 1.234E13
 1.234E14
 1.234E15
 1.234E16
 1.234E17
 1.234E18
 1.234E19
```

You can see that after the number becomes nine digits long (123400000), it is printed as 1 something, followed by E and a power of 10. Looking at the listing tells us another couple of things about variables. The variable is assigned by just entering the name of the variable (in this case, A), followed by an equals sign, and the value which we want assigned to the variable. If we said A = 99, then following this with PRINT A would produce 99. Line 120 looks a little odd. The asterisk (*) stands for multiply in BASIC. Line 120 seems to be saying that A is equal to 10 times itself, which — in terms of standard arithmetic — is not true. Any line which assigns a value to a variable has, as an optional word, LET in it. Some BASICs insist on the word being used, so line 110 would have to read LET A = 1234, and line 120 would read LET A = 10*A. BBC BASIC regards LET as optional, but it is possibly easier to understand what is happening in an assignment statement if you either use the word LET, or mentally place it in position.

Try typing in things directly, such as LET B = 13, press RETURN, then enter PRINT B, then RETURN again. A 13 will appear. Follow this by

entering B = 16, RETURN, PRINT B, RETURN. A 16 will appear, showing that the LET is optional. Despite this, you may, as I suggested a moment ago, prefer to use LET for the time being, until you are familiar with assignment.

Integer variables can be used whenever whole numbers only (i.e. 8, 5Ø4, 33 or 2398) are used. These use up less memory, and are processed more quickly than are ordinary variables, which can accommodate floating point numbers. Integer variables are indicated by the presence of a percentage sign following the name of the variable.

Enter and run this simple routine to see integer variables in action.

```
10 REM VARIABLES - INTEGER NUMERIC
20 A%=4
30 PRINT A%
40 A%=4.2
50 PRINT A%
```

You should see the number 4 printed out twice. The first time (line 3Ø) this is what you would expect, as the 4 had been assigned to A% in line 2Ø. However, in line 4Ø, A% is assigned to 4.2 but the .2 is ignored when the variable is printed. Although I pointed out that integer variables used less memory, and were processed more quickly than were floating point variables, in most cases the additional speed is not significant. However you will find times when you want maximum speed (as in programs using moving graphics), and in these the speed difference can be quite significant.

String variables

String variables are a series of letters, followed by a dollar sign. Enter A$ = "HELLO", press RETURN, then PRINT A$, RETURN, will give you HELLO. ABECD$ = "THING A TRY A", RETURN, PRINT ABECD$, RETURN will give you THING A TRY A, complete with the spaces. You can put anything, including numbers, symbols, punctuation marks and letters within the quote marks, to be assigned to a string variable. A series of letters and whatever, within quote marks in this way, is known as a string.

The variable name does not have to be in upper case (capital) letters. You can get lower case letters by pressing the CAPS LOCK key, next to the CTRL key. Pressing the CAPS LOCK key again will return you to upper case.

Crickets

There is, strange to say, a correlation between the temperature and the number of times a cricket chirps each minute. The following program converts the number of chirps per minute into the temperature, in degrees Fahrenheit. Enter and run it a few times. Note that the variable chirp is set equal initially to 80 in line 20. this is converted into the variable temperature in line 30, and this latter variable is used in the PRINT statement in line 40. The variable chirp is incremented by a random number between one and seven in line 60, there is a short delay (line 70) and then the program returns to line 30 to go through the whole process again. It will run for a long, long time (until you exceed the highest possible number the BBC Microcomputer can cope with) if you do not interrupt its running with the ESCAPE key.

```
10 REM Chirp converter
20 chirp=80
30 temperature=INT((chirp/4)+40.5)
40 PRINT '"The temperature is ";temperature
50 PRINT "when there are ";chirp;" chirps."
60 chirp=chirp+RND(7)
70 FOR J=1 TO 800:NEXT J
80 GOTO 30
```

```
>RUN
The temperature is 60
when there are 80 chirps.

The temperature is 62
when there are 87 chirps.

The temperature is 63
when there are 91 chirps.

The temperature is 64
when there are 95 chirps.

The temperature is 65
when there are 100 chirps.

The temperature is 65
when there are 101 chirps.
```

Although it takes a little longer to type in long variable names, these have a clear advantage over use of names like A, B and C2. You know, without having to refer back, what each variable represents. Here is another program which uses two variable names to help make it clear what is going on. Enter and run this.

```
10 REM ***VARIABLES**
20 WORD$="THE NUMBER IS "
30 NUMBER=3
40 CLS
50 PRINT '''WORD$;NUMBER
60 PRINT '"THE SQUARE OF ";NUMBER
70 PRINT TAB(5);"IS ";NUMBER*NUMBER
80 PRINT ''"AND THE SQUARE ROOT"
90 PRINT "IS ";SQR(NUMBER)
```

To summarise:

- *Numeric variable — This can have any name, so long as it starts with a letter and does not contain punctuation or symbols*
- *Integer numeric variable — The name is as above, but with a percentage sign at the end. This takes less memory, and is processed more quickly than an ordinary numeric variable, but can only be assigned to a whole (i.e. non-floating point) number*
- *String variable — This is a letter or combination of letters and numbers, starting with a letter and ending with a dollar sign, which is assigned to anything within quote marks.*

Variable names of all three types may be of upper or lower case letters (or a combination of these), but they do not mean the same thing. That is, a$ is not the same as A$.

All variables can be assigned by use of a LET statement, which is optional, followed by the name of the variable, an equals sign, and then the value to be assigned to the variable.

INPUT

The INPUT statement is used to get information from a user while a program is actually running. The computer stops when it comes to an INPUT statement and waits for an entry of some kind from the keyboard before it will continue with the execution of the program.

Enter and run the following, which shows numeric inputs in action. The program will wait for you to enter one number, then press RETURN, then wait for another number. After you have pressed RETURN again, it will print the sum of the two numbers.

```
   10 REM **INPUT**
   20 INPUT X
   30 INPUT Y
   40 Z=X+Y
   50 PRINT Z
>RUN
?3
?6
          9
```

As you can see, the computer generates a question mark while waiting for your input in each case. That is OK so far as it goes, but you would not have known what to do when you ran the program unless you had read it in this book. There is a simple way to rectify this, by programming in user prompts. The preceding program can easily be rewritten so that the user has no doubt as to what he or she is meant to do.

```
    10 REM **INPUT**
    20 INPUT "GIVE ME A NUMBER",X
    30 INPUT "AND ANOTHER",Y
    40 Z=X+Y
    50 PRINT Z
>RUN
GIVE ME A NUMBER?3
AND ANOTHER?6
         9
```

Running this shows that the computer prints up the words within the quote marks, adds a question mark, then waits for the input. If you don't want a question mark, then leave the comma betwen the end of the material in quote marks and the name of the variable. You can combine the inputs into a single line, as follows:

```
10 REM **INPUT**
20 PRINT "GIVE ME TWO NUMBERS,"
30 INPUT "PRESSING RETURN BETWEEN THEM",X,Y
40 Z=X+Y
50 PRINT Z

>RUN
GIVE ME TWO NUMBERS,
PRESSING RETURN BETWEEN THEM?3
?66
         69
```

Run this again, this time entering the numbers in one lot, separated by a comma, before you press RETURN. That is, when it asks for the first number, enter it as something like 3, 5 and you'll see it will accept that for the two numbers. Try it now and see.

The comma between the two numbers informs the computer that two separate items of information have been entered. As it is looking for two pieces of information, it will continue processing from this point. You'll notice that the number you enter sits up hard against the INPUT statement when it is printed. To get around this, you can put the

question mark within the quote marks, then put a space or two, and leave off the comma to suppress the question mark.

Here's a program now which shows a number of inputs in action, some using the idea mentioned above to move the entered figure away from the input statement.

Bird Cage

The game is BIRD CAGE. In its first incarnation in England, this game had the somewhat improbable name of Sweat-Cloth, and when exported to the United States in the early years of the 19th century, it was first known as Sweat. Its name changed through the years to Chucker-luck, Chuck-Luck, Chuck-a-Luck or just plain Chuck. Nowadays, because of the equipment used in the non-computer version, the game is often called The Bird Cage.

The bird cage is an enclosed wire cage holding three dice. Players bet on the likelihood of a particular number coming up. If, for example, they place their money on six, and one of the three dice ends up with a six showing, they get their money back. If all three dice show six, then they get three times their money. It is a fairly simple game, but one which arouses passion among bird cage devotees.

After the program listing is a line by line explanation of the program.

```
 10 REM**Bird Cage**
 20 REM**Showing use of INPUT**
 30 MODE7
 40 M=30
 50 GOSUB290
 60 INPUT"Size of bet? "A
 70 IF A>M THEN 60
 80 PRINT 'A$;"You are betting $";A''
 90 M=M-A
100 INPUT"Which number will fall? "B
110 IF B<1 OR B>6 THEN 100
120 FOR C=1 TO 3
130 W=0
140 GOSUB 330
150 D=RND(6)
```

```
160 PRINT''A$;"Die: ";C;" fell ";D
170 IF D=B W=A:PRINT'A$;"And you win `";W
180 M=M+W
190 GOSUB 290
200 NEXT C
210 GOSUB 330
220 GOSUB 330
230 CLS
240 IF M>0 THEN 50
250 PRINT'CHR$(128+RND(6));CHR$(141);
    "The game is over, as you are broke!"
260 PRINT CHR$(128+RND(6));
CHR$(141);"The game is over, as you are broke!"
270 SOUND 1,-15,RND(20)+30,20
280 GOTO250
290 A$=CHR$(128+RND(6))
300 PRINTA$;CHR$(141);"Stake is now $"M
310 PRINTA$;CHR$(141);"Stake is now $"M'
320 *FX 15,0
330 TIME=0
340 REPEAT UNTIL TIME>50
350 SOUND 1,-15,RND(128)+127,5
360 SOUND 2,-15,RND(128)+127,5
370 RETURN
```

10–20	REM statements for title, purpose
30	Sets the Mode to 7, the normal mode when you turn your computer on
40	Sets the variable M, which is the amount of money you have, to 30
50	Sets action to the subroutine starting at line 290 (sub-routines are discussed a little later in the book)
60	This INPUT statement gets the size of the player's bet
70	If the player tries to bet more than he or she has, action goes back to line 60 for another input
80	This PRINT statement uses A$ as the colour control code. A$ is assigned in line 290. Assigning strings to control colour in Mode 7 is discussed in the section of the book on using the graphics
90	This line subtracts the size of the bet (A) from the player's stake (M)
100	This INPUT asks the player to predict which number between one and six, will show when the dice fall

110 — The player's prediction is checked, and if it is greater than six, or less than one, is rejected, and action goes back to line 100 for a new input

120–200 — This loop does a number of things. The variable W holds the win, if any, and this is set to zero for each of the three rolls. Line 140 sends action to the subroutine from line 330, which makes a few noises, and delays a little while. A delay like this is often used to enhance games. Too quick a reaction is sometimes undesirable. Line 150 actually rolls the die, and line 160 informs the player of the result of the roll, using A$ to determine the colour of the line. Line 170 checks to see if the number shown on the die is the same as the one predicted by the player, and if it is, prints out a win message, and assigns W to equal the size of the player's bet. Line 180 adds this to the money total. If the player has not won on that roll, W still equals zero (from line 130) so the player's total is not updated. Line 290 goes to the subroutine which prints out how much money the player now has.

210–220 — The delay/noise subroutine is called twice, to give a longer delay between rounds of the game.

230 — Clears the screen

240 — This checks to see if the player has any money left (i.e. if M is greater than zero) and, if so, sends action back to line 50 for a new round.

250–280 — If the player is broke, prints out an endless message to that effect, in doubleheight letters. CHR$ (141) puts the letters in double size.

290 — Selects a colour code, which is assigned to A$

300–310 — Prints out the money the player has left, using double height letters again

320 — Clears the input buffer.

330–370 — This is the sound/delay subroutine which is called throughout the program.

Compound Interest

Here is another program to show the INPUT statement in action. It also shows the use of explicit names for variables, which make it easier to understand what is going on. You may well want to save this program on cassette, as it has a degree of practical application.

```
 10 REM SIMPLE AND COMPOUND
 20 REM       INTEREST
 30 CLS
 40 INPUT '''"PRINCIPAL",PRINCIPAL
 50 INPUT"INTEREST",interest
 60 INPUT '"FOR HOW MANY YEARS",YEARS
 70 CLS
 80 PRINT'''"--------------------------------------"
 90 PRINT"YEAR    SIMPLE   COMPOUND  DIFFERENCE"
100 PRINT"--------------------------------------"
110 FOR MONEY=1 TO YEARS
120 SIMPLE=PRINCIPAL+MONEY*PRINCIPAL*(interest/100)
130 COMPOUND=INT(100*PRINCIPAL*(1+interest/100)^MONEY)/100
140 DIFFERENCE = INT(100*(COMPOUND-SIMPLE))/100
150 PRINT;MONEY;TAB(8);SIMPLE;TAB(17);COMPOUND;
160 PRINT TAB(27);DIFFERENCE
170 NEXT MONEY
```

```
>RUN
PRINCIPAL?100
INTEREST?8.25
FOR HOW MANY YEARS?12
--------------------------------------
YEAR    SIMPLE   COMPOUND  DIFFERENCE
--------------------------------------
1       108.25   108.25    0
2       116.5    117.18    0.68
3       124.75   126.84    2.09
4       133      137.31    4.31
5       141.25   148.64    7.38
6       149.5    160.9     11.39
7       157.75   174.17    16.42
8       166      188.54    22.54
9       174.25   204.1     29.85
10      182.5    220.94    38.44
11      190.75   239.17    48.42
12      199      258.9     59.89

--------------------------------------
```

This program works out compound and simple interest, for a principal and interest rate you determine, over the number of years you decide. The variable 'interest' is written in lower case letters to prevent the computer thinking it is the function INT followed by something incomprehensible. From time to time you'll notice that variable names will be rejected by the computer. This is because you will have tried to use a reserved word (i.e. a word from the vocabulary of BBC BASIC).

The final program in this section on INPUT is also a useful program. You use it to determine the arithmetic, or harmonic, mean from a list of numbers which you enter when the program is running. Again, you'll see that explicit variable names have been used.

Arithmetic mean

```
 10 REM       ARITHMETIC
 20 REM AND HARMONIC MEAN
 30 MODE 7
 40 PROCmenu
 50 REM**************
 60 REM ARITHMETIC MEAN
 70 PRINT TAB(12,3);"ARITHMETIC MEAN"
 80 PRINT'"ENTER THE NUMBERS YOU WISH ME"
 90 PRINT "TO USE TO DETERMINE
             THE ARITHMETIC MEAN"
100 PRINT "ENTER E TO END YOUR INPUT"
110 INPUT Q$
120 IF Q$="" THEN 110
130 IF Q$="E" THEN 170
140 SUM=SUM+VAL(Q$)
150 TALLY=TALLY+1
160 GOTO110
170 PRINT''"THE ARITHMETIC
             MEAN IS ";SUM/TALLY
180 PROCmenu
190 REM*************
200 REM HARMONIC MEAN
210 PRINT TAB(12,3);"HARMONIC MEAN"
220 PRINT'"ENTER THE NUMBERS YOU WISH ME"
```

```
230 PRINT "TO USE TO FIND THE HARMONIC MEAN"
240 PRINT "ENTER E TO END"
250 INPUT Q$
260 IF Q$="" THEN 250
270 IF Q$="E" THEN 310
280 SUM=SUM+(1/VAL(Q$))
290 TALLY=TALLY+1
300 GOTO250
310 PRINT''"THE HARMONIC MEAN IS ";
     1/(SUM/TALLY)
320 PROCmenu
330 REM**********
340 DEF PROCmenu
350 PRINT''"SELECT THE PROGRAM YOU WANT:"
360 PRINT'TAB(5);"1 - ARITHMETIC MEAN"
370 PRINT'TAB(5);"2 - HARMONIC MEAN"
380 PRINT'TAB(5);"3 - TO END THE PROGRAM"
390 Z=GET
400 Z=Z-48
410 IF Z=3 END
420 CLS
430 TALLY=0
440 SUM=0
450 ON Z GOTO 60,200
460 ENDPROC
```

```
>RUN
SELECT THE PROGRAM YOU WANT:
     1 - ARITHMETIC MEAN
     2 - HARMONIC MEAN
     3 - TO END THE PROGRAM
ARITHMETIC MEAN
ENTER THE NUMBERS YOU WISH ME
TO USE TO DETERMINE THE ARITHMETIC MEAN
ENTER E TO END YOUR INPUT
?100
?234
?54.678
?234
?-664.86
?234
?E
```

```
THE ARITHMETIC MEAN IS 31.9696667
SELECT THE PROGRAM YOU WANT:
      1 - ARITHMETIC MEAN
      2 - HARMONIC MEAN
      3 - TO END THE PROGRAM
HARMONIC MEAN
ENTER THE NUMBERS YOU WISH ME
TO USE TO FIND THE HARMONIC MEAN
ENTER E TO END
?100
?234
?54.678
?234
?-664.86
?234
?E
THE HARMONIC MEAN IS 151.494768
```

GOTO

One important ability in programming is to be able to branch to different parts of the program during execution. Without this, the program would always run from the lowest line number to the highest, and then stop. One statement which allows you to move around the program at will is GOTO. The GOTO statement consists of a line number followed by the word GOTO and another line number, or followed by a calculation (such as GOTO 2*X, or GOTO 200+340).

If the computer came across 140 GOTO 190, it would jump immediately from line number 140 to line 190. This is called an unconditional branch. That is, it is a jump that does not depend on the existence of any condition. Once at line 190, the program continues to execute in order, until it comes to the end, or comes to another line directing it somewhere else.

You can use GOTO to produce programs which run for ever. These can be quite effective, especially at the end of a game. Run the following to see this in action:

```
10 PRINT "YOU HAVE WON!!!!    ";
20 GOTO 10
```

IF.....THEN GOTO

The IF statement has a similar function to GOTO, but it will only reroute the program IF certain conditions are fulfilled. This creates a conditional branch. The IF statement is made up of a line number followed by the words IF and GOTO separated by a relationship which must be determined before leaving the line. There are six relation operators which can be used to compare two variables. These are:

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

These operators are used to connect the IF.....THEN statements to form the condition to be determined.

Here's an example:

70 IF Z > = 10 GOTO 100 or 70 IF Z > = 10 THEN 100

This will be read by the computer to mean IF the value of the variable Z is greater than, or equal to, 10 THEN the program will branch to line 100. If Z us less than 10, the program will continue normal execution, with line 80.

This gives the computer decision-making power, the real source of a computer's apparent ability to think. You can use IF...GOTO to terminate a 'win condition' message such as we used before after a certain number of cycles. Enter and run the following:

```
10 X=0
20 PRINT "YOU HAVE WON!!!!   ";
30 X=X+1
40 IF X<25 GOTO 20
```

This will ensure that YOU HAVE WON!!!! is printed out a limited number of times.

IF... is not just used to branch to new lines. NEW the program, and enter the following. You'll see it has a similar effect, although the IF.... is not just sending the program to a line number.

```
10 X=0
20 X=X+1
30 IF X<25 PRINT "YOU HAVE WON!!!!    ";
40 GOTO 20
```

This program is not as useful as the other one, as it will not terminate even when it has finished printing out YOU HAVE WON!!!! You can easily discover this by running it, then pressing ESCAPE, and then PRINT X, RETURN.

This does not stop the program from demonstrating that IF can be followed by a number of commands. You can chain the results of an IF statement. If the initial condition is false, the computer will forget that line, and move onto the next one. Try this example:

```
10 X=0
20 X=X+1
30 IF X<25 PRINT "YOU HAVE
   WON!!!!    ";:GOTO 20
40 END
```

This will only execute the jump back to line 2Ø if X is less than 25. The PRINT message and the GOTO 2Ø are both conditional on the result of the IF statement. If X is not less than 25, the computer will not process the rest of the line, but will move onto line 4Ø, to END.

The BBC Microcomputer is a little more tolerant of syntax in statements than are many other computers. The word THEN is implied in an IF statement (IF the cat is hungry THEN feed it), but it is not necessary to include it. IF X = 25 THEN GOTO 4Ø is accepted, as is IF X = 25 GOTO 4Ø and IF X = 25 THEN 4Ø.

It is perhaps worth mentioning that, in other areas, the BBC machine is a fairly dogmatic creature. If you specify that a program branch is to be made only if the value of Z, for example, is equal to 6, the program will continue in a never- ending loop if Z is not exactly equal to 6, no matter how close it is (like 5.999999). If you think the value might be fractionally different from the one you want as a condition for branching, make sure you specify that the relational operator should be, say, greater than 5.5, or greater than or equal to 5.9, rather than just equal to 6.

To show the flexibility of the IF statement, enter and run the following program, then return to the book for a discussion of the various parts

of the program. This game is based on one which was played in the film "Last Year at Marienband". There are a certain number of 'matches' at the start of the game, and you and the computer take it in turns to take one or more away. The maximum number you can take is shown at the top of the screen. The player who takes the last match loses. The computer is not infallible.

Matchsticks

```
 10 REM *MATCHSTICKS*
 20 E=0:Z=15+RND(9):CLS
 30 IF 2*(Z/2)=Z THEN Z=Z+1
 40 H=2+RND(2)
 50 PRINT ''"MAXIMUM TO TAKE IS ";H'
 60 IF E>0 PRINT "YOU TOOK ";E;
    TAB(20);"I TOOK "Q''
 70 FOR K=1 TO Z
 80 PRINT K;"  ";
 90 IF RND(4)=1 PRINT
100 NEXT K
110 INPUT'"HOW MANY WILL YOU TAKE",E
120 IF E>H OR E<1 THEN 110
130 CLS:Z=Z-E
140 IF Z=0 PRINT''''"I WIN":END
150 Q=Z-1-INT((Z-1)/(H+1))*(H+1)+RND(4)-1
160 IF Q>Z OR Q<1 OR Q>H THEN 150
170 Z=Z-Q
180 IF Z=0 PRINT ''"I TOOK ";
        Q;", SO YOU WIN":END
 190 GOTO 50
```

Here is a sample run of it:

```
MAXIMUM TO TAKE IS 3
         1          2          3          4          5
      8          9          10         11         12
  15         16         17         18         19
HOW MANY WILL YOU TAKE?3
MAXIMUM TO TAKE IS 3
```

```
YOU TOOK 3          I TOOK 3
        1           2
        3           4           5           6
        7           8
        9           10          11          12          13
HOW MANY WILL YOU TAKE?2
MAXIMUM TO TAKE IS 3
YOU TOOK 2          I TOOK 3
        1
        2           3           4           5           6
HOW MANY WILL YOU TAKE?1
MAXIMUM TO TAKE IS 3
YOU TOOK 1          I TOOK 2
        1           2           3           4           5
HOW MANY WILL YOU TAKE?1
MAXIMUM TO TAKE IS 3
YOU TOOK 1          I TOOK 3
        1
HOW MANY WILL YOU TAKE?1
I WIN
```

1Ø	Title
2Ø	Assigns the variables, E is the number of matches taken by the human player, Z is the starting number of matches
3Ø	This IF statement checks a condition, and if it is so (i.e. Z is an even number) adds one to it
4Ø	Assigns a value of 3 or 4 to H
5Ø	Prints out the value of H
6Ø	If the human has made a move (i.e. E is greater than Ø) then PRINTs
7Ø–1ØØ	Prints the numbers corresponding to matches left, using the IF in line 9Ø to randomly start a new line
11Ø	Accepts the player's input
12Ø	Uses the IF statement to check the validity of the player's move
13Ø	Clears the screen, deducts the player's number from the matches left
14Ø	Uses an IF statement to see if the game is over. The END is ignored if the IF condition is found to be false
15Ø	Works out the computer's response
16Ø	Uses an IF statement to check three conditions, any one of which (if true) would mean the computer had thought of an illegal move. If any of these three conditions is evaluated as true, the computer returns to line 15Ø for another move
17Ø	Subtracts the computer's move from the matches
18Ø	Uses an IF to see if the number of matches is zero, and if it is concedes defeat, and ENDs
19Ø	Sends action back to line 5Ø for another round

IF...THEN...ELSE

The BBC Microcomputer's dialect of BASIC contains a very useful variation of IF. It can be programmed to do something if the condition being tested for is found to be true, and something else, other than just go to the next line, if the condition is found to be false. Try the following, entering a series of numbers from one to nine. Line 1020 determines that any number except five triggers the message THAT WAS NOT FIVE, and entering FIVE triggers THAT WAS FIVE. Notice that the THEN is not needed in the line, although its presence is implied.

```
1000 REM IF THEN ELSE
1010 INPUT"ENTER A NUMBER FROM ONE TO NINE"A
1020 IF A=5 PRINT "THAT WAS FIVE" ELSE PRINT "THAT WAS NOT FIVE"
1030 GOTO 1010
```

Graphs

You can use IF...THEN...ELSE to produce some very interesting graphs. You simply enter the function you would like graphed, in terms of X, in line 100. This is not the most efficient method of graphing on the BBC Microcomputer, but it is useful at this point to illustrate IF...THEN...ELSE. Try the formulae given, then create a few of your own. It is probable that you will have to change the scaling for certain formulae.

```
 10 REM GRAPH
 20 MODE7
 30 REM Enter the function you would like
 40 REM graphed, in terms of X,
 50 REM and Y, in line 100
 60 PRINT
 70 FOR Y=10 TO - 10 STEP -1
 80 PRINTY;
 90 FOR X=-10 TO 10
100 IF Y-X*X<.5 PRINT "*"; ELSE PRINT ".";
110 NEXT X
120 PRINT
```

```
130 NEXT Y
140 PRINT TAB(10);"098765432101234567890"
```

```
>RUN
          10*******.......*******
           9********.....********
           8********.....********
           7********.....********
           6********.....********
           5********.....********
           4*********...*********
           3*********...*********
           2*********...*********
           1**********.**********
           0*********************
          -1*********************
          -2*********************
          -3*********************
          -4*********************
          -5*********************
          -6*********************
          -7*********************
          -8*********************
          -9*********************
         -10*********************
            098765432101234567890
```

```
 100 IF Y-X*X>.5 PRINT "*"; ELSE PRINT ".";
>RUN
          10.......*******.......
           9........*****........
           8........*****........
           7........*****........
           6........*****........
           5........*****........
           4.........***.........
           3.........***.........
           2.........***.........
           1..........*..........
           0.....................
          -1.....................
          -2.....................
          -3.....................
          -4.....................
          -5.....................
          -6.....................
          -7.....................
          -8.....................
          -9.....................
         -10.....................
            098765432101234567890
```

```
 100 IF ABS(Y)-X*X>.5 PRINT "*"; ELSE PRINT ".";
>RUN
          10.......*******.......
           9........*****........
           8........*****........
           7........*****........
           6........*****........
           5........*****........
           4.........***.........
           3.........***.........
           2.........***.........
           1..........*..........
           0.....................
          -1..........*..........
          -2.........***.........
```

```
100 IF SQR(ABS(Y*X*2))-X<.5 PRINT
          "*"; ELSE PRINT ".";
```

```
>RUN
         10..........*..........
          9..........*..........
          8..........*..........
          7..........*..........
          6..........*..........
          5..........*........**
          4..........*......****
          3..........*....******
          2..........*..********
          1..........***********
          0..........***********
         -1..........***********
         -2..........*..********
         -3..........*....******
         -4..........*......****
         -5..........*........**
         -6..........*..........
         -7..........*..........
         -8..........*..........
         -9..........*..........
        -10..........*..........
           098765432101234567890
```

```
100 IF ABS(Y*X)-X*X>.5 PRINT
    "*"; ELSE PRINT ".";

>RUN
      10.*********.*********.
       9..********.********..
       8...*******.*******...
       7....******.******....
       6.....*****.*****.....
       5......****.****......
       4.......***.***.......
       3........**.**........
       2.........*.*.........
       1.....................
       0.....................
      -1.....................
      -2.........*.*.........
      -3........**.**........
      -4.......***.***.......
      -5......****.****......
      -6.....*****.*****.....
      -7....******.******....
      -8...*******.*******...
      -9..********.********..
     -10.*********.*********.
        098765432101234567890

100 IF Y*X-X*X/1.1>0.25
     PRINT "*"; ELSE PRINT ".";

>RUN
      10...........**********
       9...........*********.
       8...........********..
       7...........*******...
       6...........******....
       5...........*****.....
       4...........****......
       3...........***.......
       2...........**........
       1.....................
```

```
  0.....................
 -1.....................
 -2........**...........
 -3.......***...........
 -4......****...........
 -5.....*****...........
 -6....******...........
 -7...*******...........
 -8..********...........
 -9.*********...........
-10**********...........
   098765432101234567890
```

FOR/NEXT loops

FOR/NEXT loops are additional useful parts of your BASIC working tools on the BBC Microcomputer. It makes sense to study them now, because the last series of programs relied heavily on two FOR/NEXT loops, the Y loop which started at line 7Ø and end at 13Ø, and the X loop which ran from line 9Ø to line 11Ø. Because these are slightly more complex than the simplest FOR/NEXT loops, we'll leave the discussion of those alone for the time being.

A FOR/NEXT loop is made up of two statements used to control a series of cycles of a part of a program. FOR begins the loop, specifying how many times the loop is to be executed, and the NEXT statement occurs at the end of the sequence, returning the program to the statement line following the one containing the FOR command.

FOR statements are made up of the line number, following by the FOR, a numeric variable (a single letter, or any combination of letters and numbers starting with a letter, but without punctuation marks or symbols), an equals sign, a numeric expression (a number, or a previously assigned numeric variable), the word TO and finally, another numeric expression (number of previously assigned numeric variable) which is different from the first one. That may sound incredibly complicated, but it is really quite simple.

The FOR line reads:

100 FOR J = 1 TO 100

or

100 FOR CAR = A TO B

The NEXT line, which terminates the loop, is of the form:

200 NEXT J

or

200 NEXT CAR

You can omit the control variable (the J or the word CAR) after the word NEXT, but it is best in the early stages of your programming to keep it there, just so you know what is going on.

The NEXT statement then, is made up from a line number, the word NEXT, and the variable set as the control in the FOR statement, earlier in the program. The NEXT sequence is used solely to tell the computer when the sequence of programming which is being repeated is to stop. When the value of the control variable (J or CAR) reaches the value set in the FOR statement (the second numeric variable set in the FOR statement), the program passes through the loop for the final time and then continues with the line following the one containing the word NEXT.

Enter and run these simple examples:

```
    10 FOR A=1 TO 10
    20 PRINT ,A,2*A,A*A
    30 NEXT A
>RUN
               1            2            1
               2            4            4
               3            6            9
               4            8           16
               5           10           25
               6           12           36
               7           14           49
               8           16           64
               9           18           81
              10           20          100
```

```
   10 A=5
   20 B=16
   30 FOR control=A TO B
   40 PRINT ,control,2*control,control^2
   50 NEXT control
>RUN
         5            10           25
         6            12           36
         7            14           49
         8            16           64
         9            18           81
        10            20          100
        11            22          121
        12            24          144
        13            26          169
        14            28          196
        15            30          225
        16            32          256
```

In the first example, the control variable is A, and line 2Ø prints out A itself, two times A and A squared. Note that the limits of the control loop are stated explicitly in line 1Ø (1 TO 1Ø). In the second example, the control variable is a word, 'control'. It performs exactly the same as A does in the first example, with line 4Ø printing out the value of control, or two times control, and of control square. Notice that the little up arrow (found two keys to the right of the zero key) means 'raising to the power', so using this arrow followed by a 2 is the same as multiplying something by itself (as at the end of line 2Ø in the first example).

Note that the limits of the FOR/NEXT loop are two variables, A and B, which have been previously defined. You will find there are many programs when you will want a limited FOR/NEXT loop, with the limits a result of things that have occurred elsewhere in the program.

Nested loops

As you've just seen, a FOR/NEXT loop allows us to alter the value of one variable (by a count of one in the cases we've studied), to repeat a programmed series of events a specified number of times. Now, suppose there were two or more variables to be operated upon. In this case, you would need to vary both values. This can be done quite simply by *nesting* loops, in which one loop, controlled by one set of FOR/NEXT statements, operates within another set.

Enter and run the following program, which *nests* a B loop within an A loop. It also contains a third loop (M), to slow things down, but the important loops for this demonstration are A and B.

```
10 REM FOR/NEXT - NESTED LOOPS
20 FOR A=1 TO 12
30 FOR B=1 TO 12
40 PRINT B;" times ";A;" is ";A*B
50 NEXT B
60 FOR M=1 TO 1000:NEXT M
70 PRINT'''
80 NEXT A
```

When you run this, you'll see it prints out the multiplication table, from 1 X 1, to 12 X 12, pausing slightly between each set of numbers. The M loop puts the pause between each set. Part of the run is:

```
 4 times 7 is 28
 5 times 7 is 35
 6 times 7 is 42
 7 times 7 is 49
 8 times 7 is 56
 9 times 7 is 63
10 times 7 is 70
11 times 7 is 77
12 times 7 is 84


 1 times 8 is 8
 2 times 8 is 16
 3 times 8 is 24
```

```
4 times 8 is 32
5 times 8 is 40
6 times 8 is 48
7 times 8 is 56
```

In this program, the control variable A stays at one, while the loop controlled by B runs from one to 12. After the pause (line 6Ø), the control variable A increases by one, and the B loop runs through again, this time with the A equal to two, and so on, until the B loop has run through with the A equal to 12. There is no reason why you should have only two nested loops.

It is vital that the control variables of nested loops be in the correct order, that is, the first loop begun is the last one to end. Try swapping lines 5Ø and 8Ø of this program,and see what happens. You should find it prints out the 'one times table', and then stops with the error message 'No FOR at line 8Ø'.

Remember that the control variable for the NEXT statement need not be stated when programming on the BBC Microcomputer. The clever machine automatically knows which NEXT corresponds with which FOR, thus removing the possibility of incorrectly nesting loops. Change lines 5Ø and 8Ø into just NEXT, and delete the M from the end of line 6Ø. You should find the program runs perfectly. Although you can omit the NEXT control variable, I repeat the suggestion given earlier to leave it in, just to make it clear in your mind what is happening.

STEP

For this next discussion, we need the program TABULATOR ROCKET RANGE which was introduced earlier. You'll recall I suggested that you should save it on tape so you could load it in again when needed. in case you did not do so, here is the listing again.

```
 10 REM TABULATOR ROCKET RANGE
 20 MODE7
 30 FOR J=10 TO 1 STEP -1
 40 PRINT '''J
 50 T=TIME
 60 REPEAT UNTIL TIME-T>50
 70 NEXT J
 80 REPEAT
 90 Q=RND(29)+1
100 U=128+RND(5)
110 FOR rocket=1 TO 7:READ ROCKET$
120 PRINT TAB(0);CHR$(129);"(";TAB(Q);CHR$(U);
            ROCKET$;TAB(36);CHR$(129);")"
130 NEXT rocket
140 T=TIME
150 REPEAT UNTIL TIME-T=2
160 SPACE =RND(10)
170 SOUND 16,-RND(5)-10,RND(3)-1,RND(20)
180 FOR print=1 TO SPACE
190 PRINT TAB(0);CHR$(129);"(";TAB(37);")"
200 NEXT print
210 RESTORE
220 T=TIME
230 REPEAT UNTIL TIME-T=2
240 UNTIL FALSE
250 DATA "  +  "," +++ "
260 DATA " +++ "," +++ "
270 DATA " +++ "," +++ "
280 DATA " <*> "
```

The important lines for our discussion at this point are 3Ø, 4Ø and 7Ø. You'll see when you run the program that this causes the numbers 1Ø down to 1 to appear on the screen. The word STEP (in line 3Ø) after the 1 controls this. Change the −1 following the word STEP to −2, and see what happens. If no STEP is specified, the computer assumes

you want a positive STEP of 1, which is what has been needed in the earlier examples in this section.

The STEP command, then, is used within a FOR/NEXT loop to allow the user to specify the value of the increment (or decrement) of the control variable. The STEP does not have to be a whole number, although you must ensure — if the number which follows the word TO in the initial FOR statement is lower than the number before the TO — that the STEP is negative. Try the following examples:

```
   10 FOR A=100 TO 1 STEP -12.5
   20 PRINT A
   30 NEXT A
>RUN
      100
     87.5
       75
     62.5
       50
     37.5
       25
     12.5
```

```
   10 FOR A=10 TO 1 STEP -0.719
   20 PRINT A
   30 NEXT A
>RUN
        10
     9.281
     8.562
     7.843
     7.124
     6.405
     5.686
     4.967
     4.248
     3.529
      2.81
     2.091
     1.372
```

REPEAT/UNTIL

Whereas a FOR/NEXT loop should always run its full course, another pair of statements — REPEAT/UNTIL — are available on the BBC Micro which will cycle through a loop only until a specified condition is satisfied. You'll see, if you look back at the listing of TABULATOR ROCKET RANGE, in lines 50 and 60, the REPEAT/UNTIL sets a variable T to equal the value of the internal clock (TIME), then holds the program with a REPEAT UNTIL until the difference between T and TIME is 50. Also within this program is a master REPEAT/UNTIL loop, which starts in line 80 and terminates (UNTIL FALSE) in line 240. A REPEAT loop which terminates with UNTIL FALSE will run for ever, or until BREAK is pressed.

```
10 MODE2
20 REPEAT
30 GCOL 0,RND(16)-1
40 MOVE RND(1280),RND(1024)
50 PLOT 85,RND(1280),RND(1024)
60 UNTIL FALSE
```

You'll find this simple program (which you should run in Mode 2 changing line 10 to Mode 5, if you have a Model A) produces a splendid demonstration of chaotic triangles. Note that the REPEAT (line 20) and UNTIL FALSE (line 60) makes the program run forever.

Now try the next program, which uses the UNTIL as a specified control. The program is written for a Model B machine, but will work adequately in Mode 5 on a Model A. It produces some startling results, as though a stone has been thrown at the centre of the TV screen.

```
 10 REM ***Broken glass***
 20 REPEAT
 30 MODE0
 40 REPEAT
 50 A=RND(8)-1
 60 B=RND(8)-1
 70 UNTIL A<>B
 80 VDU 19,0,B,0,0,0
 90 VDU 19,1,A,0,0,0
100 REPEAT
110 C=RND(1280)
120 D=RND(1024)
```

```
130 MOVE 640,500
140 DRAW C,D
150 UNTIL RND(50)=25
160 UNTIL FALSE
```

There are three REPEAT/UNTIL loops. The first one starts at line 2Ø and terminates at line 16Ø. This is the master loop, which continues for ever. The second loop starts in line 4Ø and ends in line 7Ø. A and B, the random numbers calculated in lines 5Ø and 6Ø, control the foreground and background colours for the 'broken glass', and this loop cycles until A is different from B (see line 7Ø). The third REPEAT/UNTIL operates from line 1ØØ to 15Ø, and cycles until the random number generated in line 15Ø is 25. You can work with this program to produce a number of variations. One variation would ensure that every line plotted was in a random colour, so that the background changed colour slowly, but the foreground colour changed for every line.

GOSUB and RETURN

A subroutine is a block of program within a larger program which performs one specific task. The main program is executed, line by line, until the subroutine is called, by the GOSUB command. The computer goes to the specified number, works through in line order from that point, until it hits the word RETURN. This is the signal for the computer to return to the main program, to the line *after* the one which sent it to the subroutine.

A subroutine is useful if a particular set of calculations has to be carried out a number of times within a program, and at different places within the program. For example, in a financial program, there may be a number of VAT calculations to be done at different points within the program. Whenever this need arises, the program is told to GOSUB, and it stays in this subroutine until it hits the word RETURN, when it returns to the line *after* the GOSUB command.

A subroutine is written exactly like the main program, except that it is a program within a program, and is bounded by two lines, one containing the GOSUB and the other is the RETURN line. The GOSUB command is made up from a line number, followed by the word GOSUB, and another line number. The line 4Ø GOSUB 1ØØ tells the computer to branch to line 1ØØ and continue executing the program in order, just as if line 4Ø had said GOTO 1ØØ. However, when the program reaches a line containing the word RETURN, the action

reverts to the main program, at the line number which follows the one containing the GOSUB statement (in this case, the first line number after 40).

A simple example, showing GOSUB and RETURN, is as follows. Enter and run it a few times, then come back to the book for a discussion on it.

```
 10 REM ***gosub/return demo***
 20 CLS
 30 REPEAT
 40 INPUT'''"Enter a number "A
 50 GOSUB 100
 60 UNTIL FALSE
 70 REM *********************
 80 REM  Subroutine follows
 90 REM *********************
100 PRINT '"Your number is ";A
110 PRINT ;A;" squared is ";A*A
120 PRINT "The square root of ";A
              ;" is ";SQR(A)
130 RETURN
```

```
>RUN
Enter a number 4573
Your number is 4573
4573 squared is 20912329
The square root of 4573 is 67.6239603
```

After line 20 clears the screen, lines 30 and 60 set up the master REPEAT/UNTIL loop. Line 40 asks you to enter a number, using INPUT, then line 50 transfers control to the subroutine starting at line 100. The required calculations are carried out, and the results of them printed, within the subroutine, then line 130 returns control to the line *after* the one which sent control to the subroutine that is line 60. As line 60 is the termination of the REPEAT/UNTIL, action goes back to line 40, where a new number is requested, and the whole merry dance begins again.

Enter and run the following program, which plays a kind of BLACKJACK, to see subroutines doing something a little more interesting than the demonstration we've just run.

The card game Blackjack is, as I suppose you know, pretty popular. In his book *Beat the Odds, Microcomputer Simulations of Casino Games,* (Hayden, 1980), Hans Sagan says it is "probably the most

popular and widely-played banking card game in the States. It is played in gambling houses, private clubs, political clubrooms, barracks, troops transports, back rooms of all kinds, and places you may never have heard of". With a recommendation like that, how could the program fail?

Mr Sagan's opinion is backed up by John Scarne, who points out in his authorative work *Scarne's Encyclopedia of Games:* "Blackjack is the most widely-played banking card game in the world". Sagan points out that Blackjack is "the one casino game where the player may have a chance". Part of that chance is based on the fact that the cards removed from a pack as a game is played change the odds of other cards appearing — and knowing the odds can be of some benefit in deciding what to do. The player's slight advantage is stymied in this version, because the BBC Micro has somehow acquired an infinite, constantly-replenished deck of cards.

Despite this, the computer plays reasonably well, and will certainly give you a run for your money. There is no direct betting in the game and this is a feature you may well want to add in due course. You may also, when you feel confident of your programming ability, change the program so that it goes through a pack of cards before there is a need to reshuffle. Enter and run the program, then return to the book for a discussion on it.

```
10 REM **Blackjack**
20 REM **(C) Hartnell 1982**
30 B$="The BBC Microcomputer has "
40 C$="The mere human has "
50 M=0:MODE7:GOTO230
60 CA=RND(11):IF M>0 PROCsketch
70 IF CA=11 AND D+CA>21 CA=1
80 D=D+CA
90 IF M>1 AND CA<>1 AND CA<11 PRINTCHR$(128+
    RND(6));CA;" has been dealt"
100 M=M+1:RETURN
110 CA=RND(11):PROCsketch:IF CA=11 AND B+CA>21 CA=1
120 TIME=0:REPEAT UNTIL TIME>100:B=B+CA
130 IF M>1 AND CA<>1 AND CA<11 PRINTCHR$(128+
    RND(6));CA;" has been dealt"
140 RETURN
150 Z=128+RND(6)
160 PRINT'CHR$(Z);"Enter A for another card"
170 PRINT TAB(3);CHR$(Z);"or S to stand"
180 G=0:G$=GET$:G=-(G$="A")
190 RETURN
200 PRINT '''CHR$(128+RND(6));"Do you
    want another game? (Y or N)"
210 A$=GET$:IF ASC(A$)<> ASC("N") RUN
220 PRINT ''''CHR$(128+RND(6));"OK, thanks for playing":
    SOUND RND(3),-15,RND(100),1:GOTO220
230 PRINT''':D=0:B=0:GOSUB60:H=CA
240 GOSUB60:A=CA:GOSUB110:E=CA
250 GOSUB110:F=CA
```

```
260 PRINT'CHR$(128+RND(6));B$;H
270 PRINT'CHR$(128+RND(6));C$;E;" and ";F
280 PRINTCHR$(128+RND(6));"totalling ";E+F
290 D=H+A:B=E+F
300 GOSUB150:IF G=1 THEN 380
310 IF D<17 THEN 430
320 IF D<>21 PRINTCHR$(128+RND(6));B$;D'CHR$(128+RND(6));C$;B
330 IF B=D AND B<>21 PRINTCHR$(128+RND(6));
    "so this round is a";CHR$(128+RND(6));"draw"
340 IF B>21 OR D>21 THEN 200
350 IF B>D PRINT'CHR$(128+RND(6));TAB(9);"You win!"
360 IF B<D PRINT'CHR$(128+RND(6));TAB(11);"I win!"
370 GOTO200
380 GOSUB110
390 PRINT CHR$(128+RND(6));C$;CA;", total is ";B
400 IF B>21 PRINT CHR$(129);"You've busted, so I win!!":GOTO200
410 GOTO300
420 TIME=0:REPEAT UNTIL TIME>150
430 PRINT 'CHR$(128+RND(6));B$;D
440 TIME=0:REPEAT UNTIL TIME>150
450 GOSUB60
460 PRINT CHR$(128+RND(6));"The total is now ";D
470 TIME=0:REPEAT UNTIL TIME>100
480 IF D>21 PRINT CHR$(129);"I've busted, so you win!!":GOTO200
490 IF D<17 THEN 450
500 GOTO 320
510 END
520 DEF PROCsketch
530 IF CA=1 PRINTCHR$(129);"ACE":ENDPROC
540 IF CA<11 ENDPROC
550 T=RND(3)
560 IF T=1 PRINT CHR$(130);"Jack"
570 IF T=2 PRINT CHR$(131);"King"
580 IF T=3 PRINT CHR$(132);"Queen"
590 ENDPROC
```

```
>RUN
8 has been dealt
3 has been dealt
The BBC Microcomputer has 2
The mere human has 8 and 3
totalling 11
Enter A for another card
   or S to stand
5 has been dealt
The mere human has 5, total is 16
Enter A for another card
   or S to stand
3 has been dealt
The mere human has 3, total is 19
Enter A for another card
   or S to stand
```

```
The BBC Microcomputer has 11
5 has been dealt
The total is now 16
8 has been dealt
The total is now 24
I've busted, so you win!!
Do you want another game? (Y or N)
6 has been dealt
4 has been dealt
The BBC Microcomputer has 6
The mere human has 6 and 4
totalling 10
Enter A for another card
   or S to stand
6 has been dealt
The mere human has 6, total is 16
Enter A for another card
   or S to stand
The BBC Microcomputer has 13
2 has been dealt
The total is now 15
ACE
The total is now 16
8 has been dealt
The total is now 24
I've busted, so you win!!
Do you want another game? (Y or N)
9 has been dealt
5 has been dealt
The BBC Microcomputer has 5
The mere human has 9 and 5
totalling 14
Enter A for another card
   or S to stand
3 has been dealt
The mere human has 3, total is 17
Enter A for another card
   or S to stand
The BBC Microcomputer has 8
6 has been dealt
The total is now 14
```

```
3 has been dealt
The total is now 17
The BBC Microcomputer has 17
The mere human has 17
so this round is adraw
Do you want another game? (Y or N)
OK, thanks for playing
OK, thanks for playing
OK, thanks for playing
OK, thanks for playing
OK, thanks for playing
```

Look at line 23Ø. It includes an instruction to GOSUB 6Ø. Referring to line 6Ø, we see CA = RND(11). CA is the card from one to 11, with one as an ACE, and eleven a picture card. Line 7Ø checks to see if the number dealt is an eleven, and if it is, checks to see if this would bring the total over 21. If it does, the eleven is changed to a one. Line 1ØØ includes the RETURN instruction, which sends action to the H = CA at the end of line 23Ø, the instruction after the one to GOSUB. The computer is well able to find a RETURN destination like this buried within a multistatement line, but the GOSUB destination must always be at the start of a line. Your programs will be much easier to read if they do not include any multistatement lines, but will take up more space than necessary. It may well be worth writing a program initially in single statement lines, then 'tightening it up' later, by joining lines together.

ON GOTO. . . .ON GOSUB

The word ON, preceding either GOTO or GOSUB, produces a special kind of branching, related to the IF/THEN result. An ON. . .GOTO or ON. . .GOSUB statement is made up from a line number followed by the word ON and the result of a previous calculation. This value is followed by the words GOTO or GOSUB and a list of numbers, separated by commas.

For example 15Ø ON X GOTO 2ØØ,3ØØ,4ØØ,55Ø will send action to line 2ØØ if X equals one, to line 3ØØ if X equals two, to 4ØØ if X equals 3 and to line 55Ø if X equals 4.

When a GOSUB is used instead of a GOTO command, the same thing happens as with an ON. . .GOTO, except that at the end of the subroutine, action reverts to the line after the ON. . .GOTO. Try the following simple example:

```
   10 REM**on gosub demo**
   20 REPEAT
   30 X=RND(3)
   40 FOR J=1 TO 500:NEXT J
   50 ON X GOSUB 70,90,110
   60 UNTIL FALSE
   70 PRINT "ONE"
   80 RETURN
   90 PRINT "TWO"
  100 RETURN
  110 PRINT "THREE"
  120 RETURN
>RUN
ONE
TWO
TWO
TWO
THREE
THREE
TWO
ONE
THREE
```

This routine converts the value of X generated randomly in line 3Ø into a word, using the ON...GOSUB in line 5Ø.

Finally in this section, here are two 'poetry' programs which use ON...GOSUB (see line 8Ø) to randomly join words together. As you can see from the sample runs, the 'poetry' produced is pretty awful, although some lines (such as SPIRIT IS NEAR HUMAN HIGHROAD) suggest a new era in the creation of old wise sayings.

```
 10 REM ***Poet***
 20 REM Showing ON GOSUB
 30 FOR J=1 TO RND(3)
 40 PRINT
 50 NEXT
 60 TIME=0:REPEAT UNTIL TIME>40
 70 D=RND(13)
 80 ON D GOSUB 110,120,130,140,150,160,
        170,180,190,200,210,220,230,230
 90 TIME=0:REPEAT UNTIL TIME>9
100 IF RND(3)=1 THEN 30 ELSE 70
```

```
110 PRINT"WATCHING ";:RETURN
120 PRINT"HUMAN ";:RETURN
130 PRINT"DWELLING ";:RETURN
140 PRINT"IS NEAR ";:RETURN
150 PRINT"FEARING ";:RETURN
160 PRINT"SPIRIT ";:RETURN
170 PRINT"ALIEN ";:RETURN
180 PRINT"SMOKY ";:RETURN
190 PRINT"HIGHROAD ";:RETURN
200 PRINT"SELF ";:RETURN
210 PRINT"DREAMER ";:RETURN
220 PRINT"COMES ";:RETURN
230 PRINT"WAITS THEN FOR ";:RETURN
```

```
>RUN
SELF FEARING SMOKY
SELF
DWELLING SMOKY
WAITS THEN FOR IS NEAR
SPIRIT IS NEAR HUMAN HIGHROAD
WATCHING
SELF SPIRIT HIGHROAD SPIRIT COMES
SELF DWELLING
SMOKY ALIEN SPIRIT
SELF SELF WAITS THEN FOR
IS NEAR HUMAN HIGHROAD
FEARING DREAMER
COMES
WATCHING DREAMER WAITS THEN FOR COMES SMOKY
DREAMER
SELF HUMAN
HUMAN HIGHROAD ALIEN SMOKY IS NEAR
HIGHROAD COMES
FEARING DREAMER SMOKY WATCHING
FEARING SPIRIT WAITS THEN FOR SMOKY
FEARING HUMAN IS NEAR
```

```
10 REM ***Poet***
20 REM Showing ON GOSUB
30 FOR J=1 TO RND(3)
40 PRINT
```

```
 50 NEXT
 60 TIME=0:REPEAT UNTIL TIME>40
 70 D=RND(13)
 80 ON D GOSUB 110,120,130,140,150,160,170,
          180,190,200,210,220,230,230
 90 TIME=0:REPEAT UNTIL TIME>9
100 IF RND(3)=1 THEN 30 ELSE 70
110 PRINT"VERDANT GLADES ";:RETURN
120 PRINT"WHISPERING HILLS ";:RETURN
130 PRINT"SOFT ";:RETURN
140 PRINT"HUSHED ";:RETURN
150 PRINT"SHADOWED O'ER ";:RETURN
160 PRINT"SILENT ";:RETURN
170 PRINT"PATHWAY ";:RETURN
180 PRINT"LEAVES ";:RETURN
190 PRINT"WAVING ";:RETURN
200 PRINT"FALLING ";:RETURN
210 PRINT"YEARNING ";:RETURN
220 PRINT"LOVERS ";:RETURN
230 PRINT"TREADING SOFTLY ";:RETURN
```

```
LOVERS


LOVERS
WHISPERING HILLS

TREADING SOFTLY VERDANT GLADES HUSHED

HUSHED FALLING TREADING SOFTLY
VERDANT GLADES PATHWAY VERDANT

VERDANT GLADES VERDANT GLADES
PATHWAY

LOVERS


WAVING LOVERS PATHWAY
```

```
WHISPERING HILLS
VERDANT GLADES YEARNING WHISPERING HILLS

HUSHED

LEAVES


LOVERS SILENT

VERDANT GLADES YEARNING WHISPERING HILLS

TREADING SOFTLY VERDANT GLADES

LEAVES FALLING TREADING SOFTLY
```

DIM and ARRAYS

The DIM statement is used to set up a *list* which you can easily access. You may find it necessary, in some programs, to refer to elements of a long list of numbers, such as if you INPUT a quantity of DATA, and you wish to use it in certain ways, such as PRINTing it in order or magnitude.

An ARRAY is a set of memory spaces reserved in the computer, and referred to by the name of the array, and by a subscript. To produce an array to hold three elements, you enter DIM A(2) which creates spaces for an array called A. To hold four elements, you enter DIM B(3). On the BBC Microcomputer, there is always one more element of an array than the number in brackets which follows the DIMensioning of the array.

Enter and run the following program which should make it a little easier to understand.

```
10 REM ** ARRAYS DEMO **
20 DIM B(3)
30 FOR A=0 TO 3
40 B(A)=RND(10)
50 NEXT A
60 FOR A=0 TO 3
```

```
70 PRINT "B(";A;") IS ";B(A)
80 NEXT A
```

```
B(0) IS 4
B(1) IS 2
B(2) IS 4
B(3) IS 10

B(0) IS 2
B(1) IS 9
B(2) IS 9
B(3) IS 3
```

As I pointed out, an array contains one more element than the number which is used to dimension it, so the array B in the sample program contains four elements. You may well find it easier to 'forget' that the subscripts start at zero, and dimension an array with the number of elements you need, ignoring array element subscripted zero. Note that elements of an array are called *subscripted variables.*

As you can see from line 2Ø of the program you've just run, the computer needs you to DIMension an array before you can use it, with a DIM statement. The DIM statement is made up of a line number followed by the word DIM, and the name of the array, with the size of the array enclosed in brackets.

You can dimension more than one array at a time by using a line as follows: 1ØØ DIM A(4),B(2),S(6Ø). Just separate each of the array names with a comma.

The arrays we've been talking about so far are one-dimensional arrays, suitable for such things as holding a list of numbers. However, you can have arrays of more than one dimension. These arrays are called, reasonably enough, multidimensional arrays, and are set up with a DIM command having more than one subscript. Enter and run the following program:

```
10 REM ** MULTI-DIMENSIONAL
20 REM          ARRAYS**
30 DIM A(3,3)
40 FOR B=0 TO 3
50 FOR C=0 TO 3
60 A(B,C)=RND(9)
70 PRINT "A(";B;",";C;") IS ";A(B,C)
80 NEXT C
```

```
 90 NEXT B
100 PRINT '"    0 1 2 3"'
110 FOR B=0 TO 3
120 PRINT;B;"   ";A(B,0);" ";
    A(B,1);" ";A(B,2);" ";A(B,3)
130 NEXT B
```

When you run it you'll see something like this:

```
A(0,0) IS 3
A(0,1) IS 4
A(0,2) IS 2
A(0,3) IS 3
A(1,0) IS 5
A(1,1) IS 7
A(1,2) IS 6
A(1,3) IS 3
A(2,0) IS 9
A(2,1) IS 6
A(2,2) IS 8
A(2,3) IS 4
A(3,0) IS 6
A(3,1) IS 8
A(3,2) IS 1
A(3,3) IS 7

    0 1 2 3

0   3 4 2 3
1   5 7 6 3
2   9 6 8 4
3   6 8 1 7
```

Firstly the elements of the array will be filled with numbers between one and nine, and these are printed out by line 7Ø so you can see what is held by each element of the array. The little table printed below them shows how the elements of the array are organised. Any element can be accessed by giving its co-ordinates within the array. If this is so, element 3, 3 should lie where the two threes intersect, i.e. on the number 7. You'll see from looking above in our sample run that, in fact, A(3,3) does equal 7.

DIMensioning an array consumes memory, so do not set up an array larger than you need. The number of elements in an array is the first

number within the brackets plus one, multiplied by the second number plus one. Therefore, the array A(3,3) has 16 (3 plus 1 times 3 plus 1) elements. You can see from our sample run that this is so.

There is no reason why you should not have arrays with more than two dimensions, except for the fact that they can quickly become quite difficult to handle, and the number of elements rockets quite alarmingly. Here is a program to DIMension and fill a three-dimensional array. Although the array is only A(2,2,2), you can see the number of elements is quite large(3*3*3)

```
   10 REM ** MULTI-DIMENSIONAL
   20 REM          ARRAYS**
   30 DIM A(2,2,2)
   40 FOR B=0 TO 2
   50 FOR C=0 TO 2
   60 FOR D=0 TO 2
   70 A(B,C,D)=RND(9)
   80 PRINT "A(";B;",";C;",";D;
   90 PRINT ") IS ";A(B,C,D)
  100 NEXT D
  110 NEXT C
  120 NEXT B
>RUN
A(0,0,0) IS 9
A(0,0,1) IS 2
A(0,0,2) IS 5
A(0,1,0) IS 3
A(0,1,1) IS 4
A(0,1,2) IS 7
A(0,2,0) IS 7
A(0,2,1) IS 5
A(0,2,2) IS 4
A(1,0,0) IS 6
A(1,0,1) IS 7
A(1,0,2) IS 2
A(1,1,0) IS 2
A(1,1,1) IS 9
A(1,1,2) IS 9
A(1,2,0) IS 7
A(1,2,1) IS 6
A(1,2,2) IS 8
A(2,0,0) IS 4
```

```
A(2,0,1) IS 3
A(2,0,2) IS 2
A(2,1,0) IS 4
A(2,1,1) IS 5
A(2,1,2) IS 1
A(2,2,0) IS 1
A(2,2,1) IS 9
A(2,2,2) IS 4
```

Increase the number of dimensions to five, as in our next example, and although it is only A(1,1,1,1,1), there are now 32 (2*2*2*2*2) elements.

```
  10 REM ** MULTI-DIMENSIONAL
  20 REM          ARRAYS**
  30 DIM A(1,1,1,1,1)
  40 FOR B=0 TO 1
  50 FOR C=0 TO 1
  60 FOR D=0 TO 1
  70 FOR E=0 TO 1
  80 FOR F=0 TO 1
  90 A(B,C,D,E,F)=RND(9)
 100 PRINT "A(";B;",";C;",";D;",";E;",";F;
 110 PRINT ") IS ";A(B,C,D,E,F)
 120 NEXT F
 130 NEXT E
 140 NEXT D
 150 NEXT C
 160 NEXT B
>RUN
A(0,0,0,0,0) IS 3
A(0,0,0,0,1) IS 9
A(0,0,0,1,0) IS 1
A(0,0,0,1,1) IS 7
A(0,0,1,0,0) IS 6
A(0,0,1,0,1) IS 1
A(0,0,1,1,0) IS 8
A(0,0,1,1,1) IS 9
A(0,1,0,0,0) IS 5
A(0,1,0,0,1) IS 5
A(0,1,0,1,0) IS 1
A(0,1,0,1,1) IS 8
```

```
A(0,1,1,0,0) IS 8
A(0,1,1,0,1) IS 1
A(0,1,1,1,0) IS 9
A(0,1,1,1,1) IS 3
A(1,0,0,0,0) IS 2
A(1,0,0,0,1) IS 1
A(1,0,0,1,0) IS 4
A(1,0,0,1,1) IS 9
A(1,0,1,0,0) IS 7
A(1,0,1,0,1) IS 3
A(1,0,1,1,0) IS 1
A(1,0,1,1,1) IS 2
A(1,1,0,0,0) IS 2
A(1,1,0,0,1) IS 8
A(1,1,0,1,0) IS 1
A(1,1,0,1,1) IS 9
A(1,1,1,0,0) IS 4
A(1,1,1,0,1) IS 8
A(1,1,1,1,0) IS 2
A(1,1,1,1,1) IS 6
```

Here is a version of the game MASTERMIND (the name is copyright Invicta Plastics) to show single-dimensional arrays in use. The game is simple to play. The computer 'thinks of' a four-digit number, and you have ten guesses to work it out. A correct digit in the wrong position within the code gives you a 'white', while a correct digit in the correct position gives you a 'black'.

```
 10 REM **MASTERMIND
 20  MODE7
 30 DIMC(4),G(4)
 40 CLS
 50 PRINT'''CHR$(133);
  "I am thinking of a four-digit number,"
 60 PRINTCHR$(133);"which you
        have 10 goes to discover"'
 70 PRINT''CHR$(129);"All four
         digits are different."'''
 80 PRINTCHR$(129);"Press any key to begin"
 90 A$=GET$
100 CLS:PRINT'''
110 C(1)=RND(9)
```

```
120 FOR Z=2TO4:C(Z)=RND(9)
130   FORJ=1TOZ-1:IFC(J)=C(Z)THEN110
140     NEXT:NEXT
150 FORG=1TO10:PRINTCHR$(133);
        "Enter guess number ";G
160   INPUTA:A1=A:PRINT
        CHR$(11);CHR$(11);CHR$(11)
170   FORZ=1TO4:G(Z)=A-10*INT(A/10)
180     A=INT(A/10):NEXT
190   B=0:W=0
200   FORZ=1TO4:IFC(Z)<>G(Z)THEN220
210     B=B+1:G(Z)=0
220     NEXT
230   FORZ=1TO4:IFG(Z)=0THEN270
240     FORJ=1TO4:IFC(Z)<>G(J)THEN260
250       W=W+1
260       NEXTJ
270     NEXTZ
280   PRINTA1;CHR$(132);"scored";
       CHR$(129);B;" black";:IFB<>1 PRINT"s";
290   PRINTCHR$(132);"and";CHR$(129);W;
       " white";:IFW<>1 PRINT"s"
300   IFW=1PRINT
310   IFB=4 PRINTCHR$(133);"You guessed it..
      .in just ";G;" guess";:IFG>1 PRINT "es"
320   IFB<>4NEXTG
330 PRINT CHR$(134);"The code was";
       CHR$(129);C(4);C(3);C(2);C(1)
```

The next program, to keep your personal finances in order, uses the array A to hold the relevant amounts of money.

It is a fairly simple, but very useful, personal accounts program. When you first RUN it, you'll see that the balance — naturally enough — is zero. You can work out a series of accounts by using GOTO 7Ø, instead of RUN, to keep the 'previous balance' (variable B) intact. Note that the Centronics printer has turned all the pound signs into a single apostrophe ('). Enter these as pound signs.

The program is set up to deal with six categories — cheques, credit cards, rates, mortgage, standing orders, and monies in — as well as a seventh, salary earned, but can easily be modified to handle as many categories as you like. Simple change the six in line 4Ø (M=6) to the

number of categories you need. As well as this, you'll have to add additional data on the ON F GOSUB line, line 18Ø, so the computer will have extra destinations for additional categories. Simply add the categories before the start of the initialisation procedure (line 4ØØ). Note that 'monies in' are recorded as negative numbers, and will be shown as such in the display, which is updated after each entry is made.

It would be very easy to modify this program to give you an option to dump the accounts onto a printer. The modification should be entered between lines 21Ø and 22Ø.

Note the use of the GET function in line 54Ø, to stop the program until any key is pressed.

```
 10 REM *Personal accounts*
 20 REM (C) Hartnell/Ron Jones
 30 MODE 7
 40 M=6
 50 DIM A(M)
 60 PROCinitialise
 70 GOSUB230
 80 INPUT'"Any changes (Y or N)"Z$
 90 IF Z$="N" THEN 200
100 SOUND 1,-7,RND(50)+75,3
110 INPUT'"Number"K:IF K>M OR K<1 THEN 110
120 SOUND 1,-7,RND(50)+75,1
130 INPUT'"New amount"E
140 IF K=6 E=-E
150 A(K)=E
160 GOTO 70
170 PRINT F;
180 ON F GOSUB 330,340,350,360,370,380
190 PRINTTAB(4);"`";A(F)
200 INPUT"Salary"S:GOSUB230:R=S-T+B
210 PRINT"Balance `";R:B=R
220 END
230 T=0:CLS:PRINT''TAB(12);CHR$(128+RND(5));
     "Personal accounts"
240 PRINT'TAB(4);"Previous balance `";B'
250 FOR F=1 TO M
260 SOUND INT(F/2),-RND(15),F*20,RND(F)
270 PRINT 'F;
```

```
280 ON F GOSUB 330,340,350,360,370,380
290 PRINT"  `";A(F)
300 T=T+A(F)
310 NEXT F
320 RETURN
330 PRINT CHR$(133);"Cheques out";:RETURN
340 PRINT CHR$(129);"Credit card(s)";:RETURN
350 PRINT CHR$(130);"Rates";:RETURN
360 PRINT CHR$(131);"Mortgage";:RETURN
370 PRINT CHR$(132);"Standing orders";:RETURN
380 PRINT CHR$(133);"Monies in";:RETURN
390 REM ***************
400 DEF PROCinitialise
410 B=0
420 SOUND 1,-5,100,7
430 PRINT'''"This is a personal accounts program"
440 PRINT'"To save the balance after an earlier"
450 PRINT "run, use GOTO 70 rather than RUN."
460 PRINT' "The program is set up at present"
470 PRINT "to cater for six items.  If you need"
480 PRINT "to have more, change the value of M"
490 PRINT TAB(6);"in line number 40."
500 PRINT "Note that";CHR$(129);"monies in"
510 PRINT "is shown as a negative number"
520 PRINT' "Press any key to start the program..."
530 Z=GET
540 ENDPROC
>
```

So far we've been looking at numeric arrays. Note that the name of the array does not have to be a simple variable like A, B or C. Here are some sample valid array names:

```
10 REM VALID ARRAY NAMES
20 REM *****************
30 DIM ARRAY(9)
40 DIM array(9)
50 DIM end_of_the_world(32,5)
60 DIM My_dog_is_black(30,44,2)
70 DIM A234(76)
```

Notice that the array called 'ARRAY' is quite different from the one called 'array'. The arrays 'end_of_the_world' and 'My_dog_is_black' contain the *underline* character (shift on the £ key), not the hyphen or minus sign. Line 70 shows another valid array name. So long as it starts with a letter, and does not contain symbols or punctuation (apart from the pound sign and the underline) it will be accepted as a valid array name.

String arrays

You can also have string arrays, which are very similar to numeric arrays. Enter and run the following program to see the string array in practice, entering five words (each followed by RETURN), when prompted.

```
   10 REM STRING ARRAYS
   20 DIM A$(4)
   30 FOR B=0 TO 4
   40 INPUT A$(B)
   50 NEXT B
   60 FOR B=0 TO 4
   70 PRINT A$(B)
   80 NEXT B
>RUN
?WATER
?REASON
?WASTE
?TERROR
?ANGLOPHILE
WATER
REASON
WASTE
TERROR
ANGLOPHILE
```

The BBC version of BASIC is far more tolerant of array names than most other BASICs. Here, for example, are some of the valid string array names:

```
10 REM VALID ARRAY NAMES
20 REM ****************
```

```
30 DIM ARRAY$(77)
40 DIM TESTING$(87,23)
50 DIM better_man_than_you$(12,2,2)
```

Note that the main difference between a string array and a numeric array is the dollar sign immediately following the name. This tells the computer the name refers to a string.

Here's a string sort program to show string arrays in use. As set up, and as demonstrated in the sample run, the program caters for five words. To adapt it for more, change the 5 in lines 2Ø, 3Ø and 4Ø to the number of words you need to sort. This introduces a slight redundancy as a string array (like a numeric array) starts at zero, rather than one, but it is simpler to pretend to forget about the zero element when analysing how a program of this type works, than it is to try and keep the zero element in mind.

```
 10 REM **STRING SORT**
 20 DIM W$(5)
 30 B=0:G=5
 40 FOR A=1 TO 5
 50 INPUT W$(A)
 60 NEXT A
 70 Z=1
 80 B=Z+1
 90 IF B>G THEN 160
100 IF W$(B)>W$(Z) THEN 120
110 Z=Z+1:GOTO80
120 Q$=W$(Z)
130 W$(Z)=W$(B)
140 W$(B)=Q$
150 GOTO110
160 PRINT W$(G)
170 G=G-1
180 IF G>0 THEN 70
```

```
>RUN
?LATCH
?BREATH
?BREAD
?DRAIN
?DRAGON

BREAD
BREATH
DRAGON
DRAIN
LATCH
```

There are, then, two primary uses of the DIM statement, to create arrays for (a) numeric arrays; and (b) string (alphanumeric) arrays. There is a third use of the DIM statement which is used to reserve bytes in memory. To reserve n bytes, you enter DIM A n-1. There must be a space between the A and n-1, and there are no brackets. Use of the DIM statement for this purpose is outside the scope of this book, but is mentioned for completeness.

String Handling

Our discussion of string arrays leads us neatly into strings, and string handling. As you've probably realised by now, a string is a collection of alphanumeric characters within quote marks (including symbols and spaces, if desired). It is assigned to a variable whose name ends with a dollar sign. The same names which were given as valid for string arrays are valid for string names. So A$, niggle$, WORD$ and HI_THERE_BOB$ are all valid string names. Strings are assigned in much the same way as are numeric variables, by a statement of the form A$ = "HI". The LET (as in LET A$ = "HI") is optional, but makes for greater sense in the earlier stages of programming.

There are a number of very useful string functions, which can be used for manipulation of strings, and for extracting parts of the strings. The functions are:

ASC(X$)	This gives the ASCII code of the first character in X$, so if X$ equalled BBCMICRO, ASC(X$) would give 66.
CHR$(66)	We can check to see if, in fact, 66 is the code of the first letter of X$ (i.e., if it is the code of B) by asking the computer to PRINT CHR$(66). This goes give a B. In effect, CHR$ is the opposite of ASC, and turns a code back into a character.
LEFT$(X$,n)	This gives a string containing the n leftmost characters of X$, so LEFT$(X$,3) will give "BBC".
LEN(X$)	This function gives the length of a string, so using our string, X$, of "BBCMICRO", we get LEN(X$) of 8.
MID$(X$,n,m)	This string function produces a string from X$ which is m characters long, starting from character number n. MID$(X$,3,4) gives "CMIC".
RIGHT$(X$,n)	This function is the opposite, as may be expected, of LEFT$, and gives the n rightmost characters in the string. RIGHT$(X$,5) gives "MICRO"

STR$(A) This turns a variable (A) into a string, so if the variable was 234, the string version would be "234". This may not seem to be much use, but allows certain manipulation of numbers when they are strings which would be extremely difficult in their numeric form.

VAL(X$) This is the 'opposite' of STR$(A) and takes the first numeric value found in the string and turns it into a number. Thus VAL(X$), where X$ equals "22 + 34" would return 22. In some BASICs, the VAL function evaluates the whole of the string, so VAL ("22 + 33") would give 55. On the BBC Micro however, it gives 22.

Here is a printout from the BBC Microcomputer showing the string functions in operation.

```
>X$="BBCMICRO"

>PRINT ASC(X$)
        66

>PRINT CHR$(66)
B

>PRINT LEFT$(X$,3)
BBC

>PRINT LEN(X$)
         8

>PRINT MID$(X$,3,4)
CMIC

>PRINT RIGHT$(X$,5)
MICRO

>X$="12+43"
>PRINT X$
12+43

>PRINT VAL(X$)
        12
>X=12+43
```

```
>PRINT X
        55
>X$=STR$(X)
>PRINT X$
55
>PRINT LEN(X$)
         2

>X$="22+54+43"
>P.X$
22+54+43
>X=VAL(X$)
>P.X
        22

>X$="22/22"
>X=VAL(X$)
>P.X
        22
```

Now we have a version of the program ALPHA, which shows ASC and CHR$ in use. The computer 'thinks of' a letter of the alphabet, and you have to try and guess it. This program runs in Mode 7, and has a 'highest score' feature. In this case, it is a lowest score feature, as you are trying to guess the computer's letter in the shortest number of goes.

Run the program, then return to the book for a discussion on it.

```
 10 REM ***Alpha***
 20 REM Showing the use of ASC and CHR$
 30 CLS
 40 D=0
 50 C=1:A=64+RND(26)
 60 SOUND 1,-15,120,20
 70 PRINT''CHR$(128+RND(6));"I am thinking of a letter..."
 80 PRINT''CHR$(128+RND(6));"Try to guess it..."
 90 PRINT'CHR$(128+RND(6));"Enter your guess number "C
100 INPUT A$
110 IF ASC(A$)=A THEN 180
120 PRINT'CHR$(128+RND(6));"No, it is not "A$
130 PRINT'CHR$(128+RND(6));"Try closer to the "
140 IF ASC(A$)<A PRINT CHR$(128+RND(6));"end";
       ELSE PRINT CHR$(128+RND(6));"start";
150  PRINTCHR$(128+RND(6));"of the alphabet"
160 C=C+1
170 GOTO 90
180 PRINT'CHR$(128+RND(6));"Yes, I was thinking of "A$
```

```
190 PRINT'CHR$(128+RND(6));"That took "C" guesses"
200 IF C<D OR D=0 THEN D=C
210 PRINT'CHR$(128+RND(6));"Your best
        score this game is "D'''''
220 TIME=0
230 REPEAT UNTIL TIME>300
240 GOTO50
```

The variable for the 'highest score' is D, and this is set to zero line 40. The number of guesses is controlled by the variable C, set in line 50. The balance of line 50 chooses a letter. What it is actually doing is choosing a number between 65 and 90, because 65 is the code of A (PRINT ASC "A" would give 65) and 90 is the code of Z. Line 60 produces a short burst of sound to indicate that the computer is ready, and lines 70 to 80 announce that it is 'thinking of a number'. Line 90 requests the guess, giving the number of the guess (C) as it does so. Line 100 accepts the player's guess as a string (A$). The code (ASC) of this guess is compared with the computer's number (A), and if they are the same, action is sent to line 180 where the 'congratulations' message is printed out.

Line 140 uses the IF/THEN....ELSE we learned about earlier to tell the player to try closer to the beginning or the end of the alphabet. Line 160 adds one to the guess count, and line 170 sends the computer back to line 90 for the next guess.

The next program — Music Maker — uses other string functions which allow you to enter your melody as a string, which is then interpreted, and the music played.

The principle of the program is very simple. The BBC Micro's sound command has four parameters, and is written in the form SOUND 1, −15, 128,4. The first number after the word sound chooses the channel (0 to 3), the second is the volume (−1 to −15, with −15 the loudest), the third number is the pitch (0 to 254) and the fourth is the duration (from one upwards).

The channel (parameter one) is fixed in this program to be channel one (the first number after the word SOUND). The volume varies randomly from −11 to −15, the pitch and duration are set by the melody which you enter as a string. (The SOUND command will be looked at in more detail a little later).

Lines 40 and 50 set the initial display to tell you to 'Enter your song', and — once you have done this — to determine the speed, from one (very fast) to nine (slow). The speed is accepted in lines 80 and 90.

The main REPEAT/UNTIL loop, which actually turns the elements of the string into 'music', runs from line 110 to 200. Line 120 calls up a procedure (PROCdisplay) to print the words 'Music Maker' in a random colour, on a randomly coloured background, on the screen.

How to enter a song:

The program works by accepting the notes you need as letters, running as follows: CDEFGABcdefgabx, Note that the highest 'c' is accepted as an 'x'. After each letter comes a number, which determines the duration on the note. A rest is shown by a P (for 'pause'), so a string which read "A3B4c5P2c3" would play the note A for a count of three, followed by B for a count of four, c for five, a rest of two then c again for three. The program will play the music over and over again until you press BREAK.

There are three sample songs, which you can enter by typing MODE 4, RETURN, then GOTO 270. The program will quickly stop with an error code. To play the first song, enter — as a direct command — A$ = Z$, then follow this by GOTO 80, when you will be asked how fast you want "Cielito Lindo" to be played. To get song two, enter A$ = M$, followed by GOTO 80. To get your BBC Microcomputer to play scales, enter SCALE$ = A$, then GOTO 80. You can easily store tunes you've worked out in strings in this way.

```
 10 REM *Music maker*
 20 MODE 4
 30 REM (C) HARTNELL 1982
 40 VDU 19,3,3,0,0,0
 50 VDU 19,0,4,0,0,0
 60 PRINT'''
 70 INPUT"Enter your song        "A$
 80 PRINT"How fast? 1 (very fast) to 9 (slow) "
 90 SPEED$=GET$:TEMPO=ASC(SPEED$)-48
100 REM***********
110 REPEAT
120 PROCdisplay
130 FOR J=1 TO LEN(A$)-1
140 B$=MID$(A$,J,1)
150 N=-53*(B$="C")-61*(B$="D")-69*(B$="E")
    -73*(B$="F")-81*(B$="G")-89*(B$="A")
    -97*(B$="B")-101*(B$="c")-109*(B$="d")
    -117*(B$="e")-121*(B$="f")-129*(B$="g")
    -137*(B$="a")-145*(B$="b")-149*(B$="x")
160 D=VAL(MID$(A$,J+1,1))
170 IF B$="P" THEN GOTO 220
180 SOUND 1,-(RND(5))-10,N,D*TEMPO
190 NEXT J
200 UNTIL FALSE
210 REM***************
220 FOR Z=0 TO D*TEMPO
230 SOUND 1,0,0,0
240 NEXT
```

```
250 GOTO 190
260 REM ****Sample songs follow****
270 Z$="c1c1A2B1G1c1C1A2B1G1c1c1A2B1G1F1D5
    B1B1B2A1G1F1F1D2E1F1G1G1G2F1E1D1C5e3d2c1A6d3
    d2c1e1c4G1A2G1A1A1G1f1f1d2B1G1A1A1G2F1G1E1D1C8":
    REM Cielito Lindo
280 REM**************
290 M$="G3E1G3E1G1A1G1F1E1G2G1C1C1C1D1E1E1E1E1
    D1D1D1E1D3P1G3E1G3E1G1A1G1F1E1G2G1C1C1C1D1E1E1
    E1C1D2C1B1C1P1c1B1A6c1A1G6C1B1C1C1C1D1E2E1C1D2C1B1
    C1P1c1B1A6c1A1G6 E1D1C1C1C1D1E2E1C1D2C1B1C4P4":
    REM SHE WORE A YELLOW RIBBON
300 REM**************
310 SCALE$="C1D1E1F1G1A1B1c4P4":SCALE
320 REM**************
330 DEF PROCdisplay
340 CLS
350 VDU 19,3,RND(3),0,0,0
360 VDU 19,0,RND(7),0,0,0
370 PRINT TAB(RND(20),RND(26))"## Music maker ##"
380 ENDPROC
```

GET, GET$, INKEY, INKEY$

These four commands are related, but they behave in slightly different ways, and expect different input from the keyboard. They share the characteristic that you do not need to press RETURN after pressing a key.

GET

This waits for a numeric input before continuing. It will wait forever for your key press. Try the following, waiting a different time before each key press. The reason for the 'different time' will become clear in due course. Enter a number from one to nine, by pressing the key of that number, and you'll see it print out YOU PRESSED 6, YOU PRESSED 1 and so on. Touch the zero key to end, when it will print out YOU PRESSED Ø and on the next line END OF DEMO.

```
10 REM **GET DEMO**
20 REPEAT
30 A=GET
40 PRINT "YOU PRESSED ";A-48
50 UNTIL A=48
60 PRINT "END OF DEMO"
```

The code of the "1" key is 49, so subtracting 48 from this code (which is what the keyboard reads), allows it to print out the number of the key you pressed. Try touching some of the alpha, or control keys. TAB, for example, will give −39 and RETURN will give −35. It is useful to build up a list of the code which each key returns for use in interactive games and other programs.

The next program — Prediction — uses the GET function. In this game, you have to try and anticipate the number (from one to nine) the computer will think of next. The computer's number is shown on the screen near the top, and the bottom number is the score. The lower the score at the end (when you manage to successfully predict the computer's number), the better. This is a fairly trivial game, but shows GET in action, and may well give you ideas for better games you can write. The screen will stay blank until you press a key.

```
 10 REM **Prediction**
 20 MODE7
 30 E=9
 40 W=RND(9)
 50 Q=0
 60 REPEAT
 70 Z=GET-48
 80 IF Z>0 AND Z<10 PRINTTAB(E,12);CHR$
      (128+RND(6));"Your number is ";Z
 90 Q=Q+1
100 IF RND(3)>1 THEN 120
110 W=RND(9)
120 PRINTTAB(5,8);CHR$(129+W/2)"My number is ";W
130 PRINTTAB(E,14);CHR$(128+RND(6));"The score is ";Q
140 IF W=Z THEN 130
150 UNTIL FALSE
```

GET$

GET$ is similar to GET, except that it waits for a string input (one character long). Here's a variation of our first program for GET, showing this.

```
    10 REM **GET$ DEMO**
    20 REPEAT
    30 A$=GET$
    40 PRINT "YOU PRESSED ";A$
    50 UNTIL A$="0"
    60 PRINT "END OF DEMO"
>RUN
YOU PRESSED H
YOU PRESSED G
YOU PRESSED R
YOU PRESSED 0
END OF DEMO
```

The 'Prediction' program can easily be modified to accept a GET$. Just change line 70. Notice how this uses one of the string functions mentioned a short time ago to convert the string into a number.

```
 10 REM **Prediction**
 20 MODE7
 30 E=9
 40 W=RND(9)
 50 Q=0
 60 REPEAT
 70 Z=ASC(GET$)-48
 80 IF Z>0 AND Z<10 PRINTTAB(E,12);
    CHR$(128+RND(6));"Your number is ";Z
 90 Q=Q+1
100 IF RND(3)>1 THEN 120
110 W=RND(9)
120 PRINTTAB(5,8);CHR$(129+W/2)"My number is ";W
130 PRINTTAB(E,14);CHR$(128+RND(6));"The score is ";Q
140 IF W=Z THEN 130
150 UNTIL FALSE
```

INKEY, INKEY$

The difference between the GETs and the INKEYs is that while the GET will wait forever for a key entry, INKEY sets a time limit. The time limit follows the word INKEY, or INKEY$, as follows: INKEY(40) or INKEY$(40). The number in brackets following the word is the time the program will wait, in hundredths of a second. The function will give a zero (INKEY) or a null string (INKEY$) if no key is pressed within the time limit.

The next program — Maze Maker — shows INKEY$ in action. Using the "A", "Z", "," and "." keys, you have to move the $ sign from the bottom left hand corner to the top right-hand one, without crossing any of the little white squares. Note that no path through is guaranteed, and there is no mechanism for checking that you don't cheat. At the end, the number of 'moves' it took you is printed on the board, using CHR$(128 + RND(5)) to make the colour change randomly. You may well wish to use this effect to end your own games.

```
 10 REM Maze Maker
 20 REM Use the A Z , . keys
 30 REM  to move the $ sign
 40 REM from the bottom left-hand
 50 REM corner to the top right-hand
 60 REM one.  No path through is
 70 REM    guaranteed!
 80 MODE7
 90 S=0
100 FOR item=1 TO 920
110 IF RND(3)=1 PRINT CHR$(255); ELSE PRINT " ";
120 NEXT item
130 PRINTTAB(0,0);"    "
140 PRINTTAB(0,1);"    "
150 X=39:Y=21
160 M=X:N=Y
170 PRINTTAB(X,Y);"$"
180 S=S+1
190 A$=INKEY$(0)
200 IF A$="" THEN 190
210 IF A$="A" AND Y>0 Y=Y-1
220 IF A$="Z" AND Y<22 Y=Y+1
230 IF A$="." AND X<39 X=X+1
```

```
240 IF A$="," AND X>0 X=X-1
250 IF X=0 AND Y=0 THEN 280
260 PRINTTAB(M,N);" "
270 GOTO 160
280 REPEAT
290 PRINTTAB(12,10);CHR$(128+RND(5);"You made it!!"
300 PRINTTAB(10,12);CHR$(128+RND(5);
                        "It took you ";S;" moves"
310 UNTIL FALSE
```

Note in line 190 that the delay after INKEY$ is set at zero. This ensures the program continues without a break, whether you are pressing a key or not. You'll find it easier to know which keys to press for right and left movement by noting that the 'greater than' and 'less than' symbols point in the respective directions.

The next program — Road Runner — shows INKEY$ in action again.

In this program you are attempting to drive a long line of letter V's down a twisting, turning track of red hash symbols (shown in this listing as £ signs). Your controls are "Z" and "M" which move you left and right respectively.

Line 140 moves the track randomly, making sure that it does not stray off the edge of the screen. Line 150 prints the V, which is scrolled up (as is the tract) by lines 220 and 240. The function readch(X,Y), which starts at line 290, checks the status of the position the V will next occupy, and if it finds anything other than a 32 there (32 is a space, see line 200) sends action to the procedure "end", which starts at line 370.

The one in brackets after INKEY$ in line 250 ensures that the computer waits one hundredth of a second before proceeding. Line 260 interprets the INKEY$, and by using the logic within each pair of brackets, ensures that the V is not allowed to go off either side of the screen. This program uses a REPEAT/UNTIL loop (see lines 130 and 270) to keep it running until the loop is exited by line 200. Note in line 270 it says UNTIL FALSE. This means that the loop will continue for ever, unless there is an exist condition within the loop which is satisfied, or the BREAK key is pressed.

If you'd like to slow the program down, change the one at the end of the SOUND statement in line 230 into a two or a three, and/or change the one in brackets into two or three in line 250.

```
10 REM *Road Runner*
20 REM (C) Hartnell 1982
```

```
 50 MODE7
 60 TIME=0
 70 A=10
 80 X=13
 90 Y=12
100 FORJ=1 TO 22
110 PRINT
120 NEXT J
130 REPEAT
140 A=A+RND(2)*(A>1)-RND(2)*(A<27)
150 PRINT TAB(X-1,Y);CHR$(132);"V"
160 LX=POS
170 LY=VPOS
180 Z=FNreadch(X,Y+1)
190 VDU 31,LX,LY
200 IF Z<>32 PROCend
210 PRINT TAB(A,22);CHR$(129);"£";TAB(A+7,22);
    CHR$(129);"£";CHR$(128+RND(5));(
    TIME DIV 10)/10;" miles"
220 PRINT
230 SOUND 0,-7-(RND(8)),RND(3),1
240 PRINT
250 C$=INKEY$(1)
260 X=X+(C$="Z" AND X>2)-(C$="M" AND X<38)
270 UNTIL FALSE
280 REM**************
290 DEF FNreadch(X,Y)
300 LOCAL A%,C
310 VDU 31,X,Y
320 A%=135
330 C=USR(&FFF4)
340 C=C AND &FFFF
350C=C DIV &100
360=C
370 DEF PROCend
380 M=(TIME DIV 10)/10 - 0.9
390 REPEAT
400 PRINT TAB(0,23);CHR$(128+RND(5));
  "End of race, you lasted for ";M;" miles"
410 SOUND 0,-15,RND(12),RND(5)
```

```
420 SOUND 1,-15,RND(12),RND(5)
430 SOUND 3,-15,RND(12),RND(5)
440 UNTIL FALSE
450 ENDPROC
```

SOUND

The SOUND statement on your computer is very versatile. It is easy to use, and if you avoid the use of ENVELOPE the first few times you use the SOUND command, you should find it fairly easy to master. Even without ENVELOPE, you'll find there is a wide variety of sounds you can make to incorporate into your programs.

As was pointed out a short time ago, the word SOUND is followed by four parameters. The first (Ø to 3), chooses the sound channel, the second (Ø to −15) sets the volume, with −15 being the loudest, the third chooses the pitch of the note (Ø to 255) and the fourth controls its duration (1 to 254). A simple, two-line program which puts the sound output through its paces indefinitely, producing a sort of weird electronic music, is:

```
10 SOUND RND(4)-1,-RND(15),RND(254),RND(20)
20 GOTO 10
```

We can demonstrate the statement in action more precisely with the following routine, which sets the volume at maximum (−15) and the duration at one. Enter and run this program to hear it in action. You can read the channel (the left hand column) and the pitch (the right hand one) as the program runs.

```
10 REM **SOUND DEMO**
20 VOLUME=-15
30 DURATION=1
40 FOR CHANNEL=0 TO 3
50 FOR PITCH=1 TO 255 STEP 3
60 SOUND CHANNEL,VOLUME,PITCH,DURATION
70 PRINT CHANNEL,PITCH
80 NEXT PITCH
90 NEXT CHANNEL
```

You'll see that the first run, when CHANNEL is set to zero, produces a strange sound, quite unlike the music produced when the CHANNEL

is one, two or three. This is because channel zero is for noise, and the other three allegedly for music. The rising tone produced on channels one, two and three is, I'm sure you'll agree, quite effective.

You can change this, by putting the volume in a loop, to produce a totally different sound. Try to predict what this routine will produce, before you run it. You're almost certain to be wrong. Modify the above program, by adding lines 55 and 75 below, and changing lines 50 and 70 as indicated.

```
10 REM **SOUND DEMO**
30 DURATION=1
40 FOR CHANNEL=0 TO 3
50 FOR PITCH=1 TO 100 STEP 3
55 FOR VOLUME=-15 TO -1
60 SOUND CHANNEL,VOLUME,PITCH,DURATION
70 PRINT CHANNEL,VOLUME,PITCH
75 NEXT VOLUME
80 NEXT PITCH
90 NEXT CHANNEL
```

If all went well, you should have heard a strange series of cymbal-like sounds, followed by notes played on a rather flat piano.

There is, of course, no reason why even random noise should not be musical. The following routine, "Bamboo flute", produces music of a sort, which — although somewhat atonal — is certainly better than the mishmash of noise produced by the two-line routine at the beginning of this section.

```
10 REM **Bamboo flutes**
20 REPEAT
30 VOLUME=-10 -RND(5)
40 CHANNEL=RND(3)
50 PITCH=RND(8)*30
60 DURATION=2*RND(10)
70 SOUND CHANNEL,VOLUME,PITCH,DURATION
80 PRINT CHANNEL,PITCH
90 UNTIL FALSE
```

The SOUND statement can also produce quite creditable sound effects, such as the following.

```
10 REM **Steam train**
20 K=-15
30 REPEAT
```

```
 40 SOUND 0,K,100,6
 50 SOUND 1,K,120,5
 60 SOUND 2,K,140,4
 70 SOUND 3,K,160,3
 80 T=TIME
 90 REPEAT UNTIL TIME-T>30
100 K=K+0.5
110 UNTIL K=0
```

Experimentation is the only way to discover how wide the range of this command is.

Let's leave noise alone for a moment, and try to produce some 'real music' from the computer.

The next routine turns the bottom row of keys (Z to the 'less than') into a one octave piano. It uses several things we've learned so far in the book. Enter and play a few melodies on it, then we'll have a closer look at the listing.

```
 10 REM ** PIANO **
 20 REPEAT
 30 *FX 15,0
 50 A=GET
 60 M=-53*(A=90)-61*(A=88)-69*(A=67)
-73*(A=86)-81*(A=66)-89*(A=78)-97*(A=77)-1
01*(A=44)
 70 IF M=0 THEN 50
 80 FOR J=-15 TO -5
 90 SOUND 1,J,M,1
100 SOUND 2,J,M,1
110 NEXT J
120 UNTIL FALSE
```

Lines 2Ø and 12Ø, of course, set up a master REPEAT/UNTIL loop to keep your piano playing. Lines 3Ø flushes the buffer so that if you linger on a key, it will not play endlessly. Once you've run the program a few times, delete line 3Ø and see what effect this has. *FX 15,Ø is a useful command to use in any programs which read the keyboard through GET, GET$, INKEY or INKEY$. M is the variable to be assigned to the pitch value in the sound command.

Line 5Ø read the keyboard, and the long line 6Ø changes the value read by the GET function into a pitch value corresponding to the note required.

Line 6Ø uses the way the BBC Microcomputer evaluates true and false. If the condition within the brackets is true, the computer returns −1. It returns zero if the condition within the brackets is false. The number in front of each bracket (like −53) is the pitch value. If A equals 9Ø (that is, the Z key is being pressed), the computer evaluates that as true, − 1, so M is set equal to 53 (that is, −53* −1). This is so for the rest of the line. If a key other than the eight on the bottom row has been pressed, all the conditions within brackets will be zero; so M will equal zero. In this case, the computer will go back to line 5Ø to read the keyboard again. Lines 8Ø to 11Ø 'play' the note, using the volume loop (lines 8Ø and 11Ø) to produce a sound somewhat like a piano. You can turn your computer into an 'autopiano' by making some changes and additions to the program. I have not renumbered it, so you can easily convert your 'piano' to play itself.

```
  10 REM ** AUTO PIANO **
  20 REPEAT
  30 *FX 15, 0
  50 A=RND(47)+43
  60 M=-53*(A=90)-61*(A=88)-69*(A=67)
-73*(A=86)-81*(A=66)-89*(A=78)-97*(A=77)-1
01*(A=44)
  70 IF M=0 THEN 50
  75 PRINT CHR$(128+RND(6)),M
  80 FOR J=-15 TO -1 STEP RND(3)
  90 SOUND 1,J,M,RND(3)
 100 SOUND 2,J,M,RND(5)
 110 NEXT J
 120 UNTIL RND(10)=1
 122 PRINT 'CHR$(128+RND(6)),"PAUSE"'
 125 FOR J=-15 TO -1
 130 SOUND 1,J,53,5
 140 NEXT J
 150 RUN
```

The possible permutations of this program are endless. Here is another version which produces a tone somewhat closer to an electronic organ trying to be a violin. Again, the program has not been renumbered, so you can easily modify your earlier program.

```
10 REM ** AUTO PIANO **
15 REM ** WITH VIBRATO **
20 REPEAT
30 *FX 15, 0
```

```
   50 A=RND(47)+43
   60 M=-53*(A=90)-61*(A=88)-69*(A=67)
-73*(A=86)-81*(A=66)-89*(A=78)-97*(A=77)-1
01*(A=44)
   70 IF M=0 THEN 50
   75 PRINT CHR$(128+RND(6)),M
   80 FOR J=1 TO RND(10)+5
   90 SOUND 1,-10-RND(5),M,RND(3)
  100 SOUND 2,-10-RND(5),M,RND(3)
  105 SOUND 3,-10-RND(5),M,RND(3)
  110 NEXT J
  120 UNTIL RND(14)=5
  122 PRINT 'CHR$(128+RND(6)),"PAUSE"'
  125 FOR J=1 TO RND(10)+10
  130 SOUND 1,-15,53,5
  135 SOUND 2,-6-RND(5),69,5
  136 SOUND 3,-6-RND(5),81,5
  140 NEXT J
  150 RUN
```

There is no reason why the 'piano' should be restricted to one note at a time. This final version play 'chords' (or, at least, two notes at once).

```
   10 REM ** AUTO PIANO **
   15 REM ** WITH VIBRATO **
   16 REM **AND 'CHORDS'**
   17 DIM E(2)
   20 REPEAT
   40 FOR B=1 TO 2
   50 A=RND(47)+43
   60 E(B)=-53*(A=90)-61*(A=88)-69*(A=67)
 -73*(A=86)-81*(A=66)-89*(A=78)-97*(A=77
)-101*(A=44)
   65 NEXT B
   70 IF E(1)=0 OR E(2)=0 THEN 40
   75 PRINT CHR$(128+RND(6)),E(1),E(2)
   80 FOR J=1 TO RND(10)+5
   90 SOUND 1,-10-RND(5),E(1),RND(10)
  100 SOUND 2,-10-RND(5),E(2),RND(10)
  105 SOUND 3,-10-RND(5),E(1),RND(3)
  110 NEXT J
  120 UNTIL RND(14)=5
```

```
122 PRINT 'CHR$(128+RND(6)),"PAUSE"'
125 FOR J=1 TO RND(10)+10
130 SOUND 1,-15,53,5
135 SOUND 2,-6-RND(5),69,5
136 SOUND 3,-6-RND(5),81,5
140 NEXT J
150 RUN
```

You should now be in a better position to understand the 'Music Maker' program, introduced a short while ago in the string handling section of this book.

The GET$ function is used in line 90 to get a number between one and nine to set the speed of the program. Line 150 should now make sense. It uses some of the same numbers outside the brackets as does 'Auto-piano', and — within the brackets — evaluates for portions of the string A$ which holds the melody. You can use the information within line 150 for which number corresponds to which note to write a program for a two octave piano program.

To recap our discussion to date on SOUND. The statement has four parameters, the first of which (voice, or channel number) can be from zero (noise) to three. The second parameter controls the volume, and can be set from 0 to −15, with −15 the loudest. The third figure after the word SOUND is the pitch, and this can be any number from zero to 255, while the fourth parameter — which controls the duration of the note — can have any value from one to 254.

You will find, from time to time, that sounds tend to continue after the line which generated them as long been passed. The computer, when necessary, stacks up notes and waits for the sound channel to be clear so it can play them. If you want a note to be played immediately, in effect to dismiss the notes stacked up waiting for their turn, you can use a slightly more elaborate form of the SOUND statement.

The computer can stack up to four notes at a time, and the full SOUND statement can tell it how many notes you wish to play. The statement, in its full version, is as follows:

SOUND &HSFC, volume, pitch, duration

The first parameter after the ampersand, H can have a value of 0 or 1. In general use, it will be set to equal 0. It has a specific task when used in conjunction with the ENVELOPE statement. For all intents and purposes, at this stage, we can assume H will always equal zero.

The second one, S, dictates the number of notes which are to be sounded at once. S can equal Ø, 1, 2 or 3, and when set to zero, allows the notes to be played one at a time, in turn. This is the way the SOUND statement operates in its simpler form.

The third parameter, F, can be set to equal Ø, when the SOUND statement operates as normal, accepting the notes in the order in which they are 'queued up'. If F is set to 1, all the waiting notes are suppressed, the note which is sounding is cut off, and the newest note is sounded.

C stands for Channel, the voice which we have used in the past. It is set to Ø 1, 2 or 3.

ENVELOPE

The SOUND statement is relatively simple, although it can — as you have seen — produce quite exotic effects. However, the computer's noise-making abilities really come into play when the ENVELOPE statement is introduced. The ENVELOPE statement is, however, somewhat bewildering to use. It is followed by 14 parameters. A proper discussion of ENVELOPE is outside the scope of a book like this, but here is some material to start you thinking about, and using it. You can generally define up to four ENVELOPEs within a program. The next routine will define ENVELOPEs at random, and shows the range of each of the 14 parameters.

```
 10 REM **ENVELOPE DEFINER**
 20 NUMBER=RND(4)
 30 length=RND(127)-128*(RND(2)=1)
 40 PITCH1=128-RND(256)
 50 PITCH2=128-RND(256)
 60 PITCH3=128-RND(256)
 70 STP1=RND(256)-1
 80 STP2=RND(256)-1
 90 STP3=RND(256)-1
100 AMPATT=127-RND(255)
110 AMPDEC=127-RND(255)
120 AMPSUS=-RND(128)+1
130 AMPREL=-RND(128)+1
140 TARGATT=RND(127)-1
150 TARGDEC=RND(127)-1
```

```
  160 ENVELOPE NUMBER,length,PITCH1,
PITCH2,PITCH3,STP1,STP2,STP3,AMPATT,AMPDEC,A
MPSUS,AMPREL,TARGATT,TARGDEC
  170 SOUND 17,2,100,255
  180 PRINT';NUMBER;" ";length;" ";PITCH1;
" ";PITCH2;" ";PITCH3;" ";STP1;" ";STP
 2;" ";STP3;" ";AMPATT;" ";AMPDEC;" ";AMPSUS,
 AMPREL;" ";TARGATT;" ";TARGDEC
  185 FOR J=1 TO 200:NEXT J
  190 RUN
```

Variable Name Parameter	Range	Function
Number	1–4	This is the identifying number
length	1–127 plus 128 for dont't repeat	Length of each step
PITCH 1	–128 to 127	All three govern changes
PITCH 2	–128 to 127	
PITCH 3	–128 to 127	in pitch
STP1	Ø to 255	Number of steps in each
STP2	Ø to 255	section
STP3	Ø to 255	
AMPATT	–127 to 127	Change of amplitude per step in attack phase
AMPDEC	–127 to 127	Change during decay phase
AMPSUS	–127 to 127	Change during sustain phase
AMPREL	–127 to 127	Change during release phase
TARGATT	Ø to 126	Target level at end of attack phase
TARGDEC	Ø to 126	Target level at end of decay phase

Experiment with the ENVELOPE statement. You're sure to discover some very interesting sounds.

READ/DATA/RESTORE

READ and DATA are very convenient ways of accessing information within a program. It is relatively simple to use. Enter and run the following program, which shows READ and DATA in action, and then return to the book for an explanation of how it works.

```
 10 REM READ/DATA
 20 REM ****************
 30 REM ***Read the DATA
 40 REM ****************
 50 DIM B(5)
 60 FOR A=1 TO 5
 70 READ B(A)
 80 NEXT A
 90 REM ****************
100 REM Print it back
110 REM ****************
120 FOR C=5 TO 1 STEP -1
130 PRINT B(C)
140 NEXT C
150 DATA 13,35241,88,2,1999999
```

```
>RUN
  1999999
        2
       88
    35241
       13
```

In line 70, the computer comes across the instruction READ.... Whenever it finds a READ instruction, it goes to the first item following the word DATA, and READs that, in this case, into an array. The DATA items can be anywhere in the program (although it is useful to keep them fairly close to the READ statement which refers to them).

Return to the program TABULATOR ROCKET RANGE which we have used a few times before in this book. Here's the listing again to refer to.

```
10 REM TABULATOR ROCKET RANGE
20 MODE7
30 FOR J=10 TO 1 STEP -1
40 PRINT '''J
50 T=TIME
60 REPEAT UNTIL TIME-T>50
70 NEXT J
80 REPEAT
90 Q=RND(29)+1
100 U=128+RND(5)
110 FOR rocket=1 TO 7:READ ROCKET$
120 PRINT TAB(0);CHR$(129);"(";TAB(Q);
    CHR$(U);ROCKET$;TAB(36);CHR$(129);")"
130 NEXT rocket
140 T=TIME
150 REPEAT UNTIL TIME-T=2
160 SPACE =RND(10)
170 SOUND 16,-RND(5)-10,RND(3)-1,RND(20)
180 FOR print=1 TO SPACE
190 PRINT TAB(0);CHR$(129);"(";TAB(37);")"
200 NEXT print
210 RESTORE
220 T=TIME
230 REPEAT UNTIL TIME-T=2
240 UNTIL FALSE
250 DATA "  +  "," +++ "
260 DATA " +++ "," +++ "
270 DATA " +++ "," +++ "
280 DATA " <*> "
```

In the first program in this section, the DATA is numbers, and these are assigned to numerical variables (the elements of the array). In TABULATOR ROCKET RANGE, the DATA is strings (lines 250 to 280) and these are assigned in turn to the string variable ROCKET$, and then printed out in line 120. The next item of DATA is then assigned to ROCKET$, as the program continues through the loop labelled 'rocket'. As you know, this program runs over and over again, but there are only seven items of DATA, only enough to go through the 'rocket' loop once. The program needs the third word which goes with READ and DATA. This third word is RESTORE, which you'll see as line 210. This tells the computer to go back to the start of the list of DATA and start READing from the first item again. Here is another sample program, showing DATA in the form of strings, and illustrating RESTORE in action.

```
10 REM READ/DATA/STRINGS
20 REM ****************
30 REM ***Read the DATA
40 REM ****************
50 DIM B$(21)
60 FOR A=1 TO 21
70 READ B$(A)
80 IF 3*INT(A/3)=A THEN RESTORE
```

```
 90 NEXT A
100 REM ****************
110 REM Print it back
120 REM ****************
130 FOR C=1 TO 21
140 PRINT B$(C)
150 NEXT C
160 DATA WET,KILL,DIE
```

In this program, there are only three items of DATA, so RESTORE must operate once the three have been read. Line 80 ensures that this occurs every time the three are read while running through the A loop from 1 to 21. Notice that the string DATA do not have to be enclosed within quote marks. Despite this, it is a good idea to always include items of string DATA within quotes, to separate them clearly from numeric DATA.

```
 10 REM READ/DATA/STRINGS
 20 REM ****************
 30 REM ***Read the DATA
 40 REM ****************
 45 WET=99:KILL=12:DIE=807
 50 DIM B$(21),Z(21)
 60 FOR A=1 TO 21
 70 READ B$(A),Z(A)
 80 IF 3*INT(A/3)=A THEN RESTORE
 90 NEXT A
100 REM ****************
110 REM Print it back
120 REM ****************
130 FOR C=1 TO 21
140 PRINT B$(C),Z(C)
150 NEXT C
160 DATA "WET",WET,"KILL",KILL,"DIE",DIE
```

The value of separating string from numeric DATA is shown clearly in the preceding program where the numeric variables are made up from the same letters which form the strings. Even in this case, the computer sorts it all out. You can adapt the string READ/DATA program to read both string and numeric items by adding line 45, and by modifying lines 50, 70, 140 and 160.

As you have seen, a READ command is used within a line to assign values to variables from a sequence of items contained within a DATA

statement. Each item of DATA is separated from others by a comma.A READ statement is made up of a line number, followed by the word READ, and the variable names which are to be assigned to the variables taken from the DATA line.

When a program comes to a READ statement, it will — as I pointed out — move to the first DATA statement, no matter where it is in the program. The first value of the DATA statement will be assigned to the first variable in the READ statement. Apart from reading a DATA statement, the computer takes no notice of it, and will treat it as a REM statement. Move line 160 up to line 25, and run the preceding program again. You'll see that (a) the computer ignores line 25, and (b) still READs it successfully.

Even if the DATA is scattered all over the program, the computer will seek it out, as the following program shows. This one is, of course, based on the preceding ones.

```
 10 REM READ/DATA/STRINGS
 20 REM ****************
 25 DATA "HI",7
 30 REM ***Read the DATA
 40 REM ****************
 50 DIM B$(21),Z(21)
 60 FOR A=1 TO 21
 70 READ B$(A),Z(A)
 80 IF 3*INT(A/3)=A THEN RESTORE
 84 DATA "GOSH"
 90 NEXT A
 95 DATA 56,"BOB"
100 REM ****************
110 REM Print it back
120 REM ****************
130 FOR C=1 TO 21
140 PRINT B$(C),Z(C)
145 DATA 22
150 NEXT C
```

It is important to ensure that you have enough DATA items for the number of times you tell the computer to READ. Delete line 145 in the above program, and run it again. You will get the error message 'Out of DATA at line 70' where the computer had read two items of numeric DATA, then was unable to find a third because RESTORE had not yet been evoked.

Remember that although it is not essential to have the DATA items near the READ lines which are looking for them, it will probably make your programs easier to understand if they are held in this manner. It also makes it easier to know which lines to alter if you are working on a program.

Mastering the Graphics

There are eight graphic modes on the BBC Microcomputer, and it can be quite bewildering — when you first get your machine — to try and work out how to use them.

We'll start our discussion with Mode 7, the teletext display mode, as this is the easiest to understand. As well as this, it can be used to produce splendid displays, despite some limitations. Your computer is automatically in Mode 7 when you turn it on, and it is the mode which uses the smallest amount of memory. In mode 7, the screen is 40 characters across, and 25 down. There are six basic colours (plus black and white) available in this mode, and they are selected by preceding the text you want printed — after the word PRINT — with a character control. Type in 'program one', and you'll see what I mean.

```
10 REM MODE 7 GRAPHICS PROGRAM ONE
20 PRINT CHR$(129);"This will print in red"
```

Program one shows that character 129 turns the text on that line (but not on the following lines, even if the PRINT statement 'wraps around' to the next line) red. Try program two, which indicates that CHR$ (130) turns text which follows it green.

```
10 REM MODE 7 GRAPHICS PROGRAM TWO
20 PRINT CHR$(130);"This will print in green"
```

As I said, there are six colours you can select in this way. The six are shown by program three.

```
10 REM MODE 7 GRAPHICS PROGRAM THREE
20 FOR colour= 1 TO 6
30 PRINT CHR$(128+colour);"TESTING "colour'
40 NEXT colour
```

The codes, and their colours, are:

129 — red
130 — green
131 — yellow
132 — blue
133 — magenta (purple)
134 — cyan (light blue)

This list suggests that there is a very simple way of generating randomly coloured PRINT statements which can be very useful for increasing the effectiveness of a program. Try program four to see this random colour change in action.

```
10 REM MODE 7 GRAPHICS PROGRAM FOUR
20 REPEAT
30 PRINT 'CHR$(128+RND(6));"Testing..."
40 TIME=0
50 REPEAT UNTIL TIME>60
60 UNTIL FALSE
```

The important line in this program is, of course, line 30, which generates a random number between one and six, and adds it to 128. Incidentally, this program also shows use of the REPEAT/UNTIL loop for two quite different purposes. The loop beginning at line 20 and ending at line 60 is a 'perpetual loop', as you have seen in several other programs in this book. This sort of REPEAT/UNTIL continues until BREAK is pressed. The use of an 'UNTIL FALSE' terminator in this way is more elegant programming than use of GOTO 30 for line 60.

The second REPEAT/UNTIL loop is in line 50. Line 40 sets a 'clock', which is always running while the computer is turned on, to zero. Line 50 holds the program in a REPEAT/UNTIL loop until the time had incremented to 61. Turn to your computer now, and type, in the direct mode, PRINT TIME. As you'll see, it has increased considerably from the 61 it read when you last ran program four. A REPEAT/UNTIL delay loop is a more flexible delay device than a 'dummy' FOR/NEXT loop such as was used in some early programs in this book.

Now, let's get back to the graphics.

The numbers higher than 134 also have interesting and useful effects if PRINTed following CHR$. Enter and run program five to see what these are.

```
10 REM MODE 7 GRAPHICS PROGRAM FIVE
20 FOR colour=129 TO 151
```

```
30 PRINT ;colour;CHR$(colour);" abcdefgh
     ijklmnopqrstuvwxyz"
40 NEXT
```

The first thing you'll notice is that control character 135 turns the text back to white, and 136 makes the text, which is white, flash off and on. The line proceded by character 141 looks very odd. If you look at it closely, you'll see it is the top half of the relevant letters. This shows us the way to get double-height characters. Enter the following line, in the direct mode, to see character 141 do its work:

```
>PRINT CHR$(141);"TEST":PRINT CHR$(141);"TEST"
```

As you can see, this writes the word TEST in double-height letters. You need to enter every line twice, following CHR$(141)'s, to get words to print in this manner. To prove this, change the T of the *second* word TEST to W and see what happens:

```
>PRINT CHR$(141);"TEST":PRINT CHR$(141);"TWST"
```

As you can see, it combines the top half of the first line with the bottom half of the second line. It is possible to combine colours, and double height characters, to produce large coloured PRINT output. So as not to disturb the program (program five) which you still have in your machine, enter the following program starting at line 100, and run it by entering GOTO 100.

```
100 REM DEMO SIX DOUBLE HEIGHT, COLOURED
110 REPEAT
120 PRINT CHR$(141);CHR$(128+RND(6));"This is a test!"
130 PRINT CHR$(141);CHR$(128+RND(6));"This is a test!"
140 TIME=0
150 REPEAT UNTIL TIME>60
160 UNTIL FALSE
```

You can see that mixed colours are allowed, and can be most effective. Add 50 END to the program (program five) which you still have in your computer, and RUN it again. You'll see that the output when the control characters lie between 145 and 151 are very interesting, producing a good selection of teletext 'chunky graphics'. You've probably seen how effectively these can be combined to produce pictures on Ceefax, Oracle or Prestel. You may like to create

your own table of these characters, so you can use them at will to create pictures of your choice. Change line 30 to the following to see more of the available graphics:

```
30 PRINT ;colour;CHR$(colour);"abcdefghijk
    lmnopqrstuvwxyz!$%&'()0=^";CHR$(col
    our);"{*+}<>?"
```

(Note that although the printer prints out the character as it appears on the keyboard, the character on the screen in the teletext mode is sometimes different. For example, the curly bracket after the quote marks following the third appearance of the word colour in line 30 appears on the screen as ¼.)

To see the graphic shapes more clearly, enter and run demonstration program seven.

```
10 REM DEMO SEVEN GRAPHIC CODES
20 FOR A=33 TO 254
30 PRINT ;A,CHR$(150);CHR$(A)
40 TIME=0
50 REPEAT UNTIL TIME>60
60 NEXT
```

Look at the program output closely, and see if you can determine whether or not the characters repeat, and — if they do — which pattern underlies the repetition.

To get colours, as well as graphics, you precede the lower case letters in the PRINT statement with the following numbers: 145, red; 146, green; 147, yellow; 148, blue; 149, magenta; 150, cyan; or 151, white. The following program demonstrates this:

```
10 REM DEMO EIGHT COLOURED GRAPHICS
20 FOR code=145 TO 151
30 PRINT 'code CHR$(code);"a  b  c  d  e"
40 NEXT code
```

You can mix text with graphics in the same colour, from the same PRINT line, so long as you are happy with the text being in upper case letters. Program nine should make this clear:

```
10 REM DEMO NINE GRAPHICS, LETTERS
20 FOR code=145 TO 151
30 PRINT 'code CHR$(code);"a A b B c C d D e E"
40 NEXT code
```

If you wish to change the background colour, you need to select the colour you want (using the code you discovered from running program three), and *follow this* with CHR$(157), which tells the computer you want the colour which preceded it to apply to the background. You follow the CHR$(157) with another character (again chosen from those demonstrated in program three) to select the colour of the text.

```
10 REM DEMO 10  COLOURED BACKGROUND
20 FOR background = 129 TO 135
30 FOR foreground = 129 TO 135
40 PRINT CHR$(background);CHR$(157);
     CHR$(foreground);"ABCDabcd"
50 NEXT foreground
60 TIME=0
70 REPEAT UNTIL TIME>100
80 NEXT background
```

You can see what a splendid effect this has. The delay loop (line 70) is to give you a chance to admire one set of foregrounds on a background before the next set appears. The foreground and background colours can be changed at random for some quite spectacular effects. Try program eleven. You may well be able to use a routine like this in one of your own programs.

```
 10 REM DEMO 11  RANDOM COLOURED
 20 REM COLOURS ON COLOURS
 30 MODE7
 40 PRINT''''"What is your full name";
 50 INPUT A$
 60 REPEAT
 70 PRINT CHR$(128+RND(7));CHR$(157);CHR$(128+RND(7));A$
 80 TIME=0
 90 REPEAT UNTIL TIME>60
100 UNTIL FALSE
```

As you can see from running this (which includes white as one of the colours, to increase variety), from time to time the foreground and background colours will be identical, so that nothing can be seen. It is very easy to write a routine to overcome this, using a REPEAT/UNTIL loop.

```
10 REM DEMO 12 RANDOM COLOURS
20 REM         ON COLOURS
30 MODE7
40 PRINT''''"What is your full name";
50 INPUT A$
60 REPEAT
```

```
 70 REPEAT
 80 A=128+RND(7):B=128+RND(7)
 90 UNTIL A<>B
100 PRINT CHR$(A);CHR$(157);CHR$(B);A$
110 TIME=0
120 REPEAT UNTIL TIME>60
130 UNTIL FALSE
```

You'll notice that parts of the program, like the PRINT line (100), are getting a little messy. Fortunately, because the BBC Micro allows concatenation (adding together) of strings, you can easily combine all the colour information, as well as the other required information, into a single string.

```
 10 REM DEMO 13 RANDOM COLOURS
 20 REM WITH CONCATENATION
 30 MODE7
 40 PRINT''''"What is your full name";
 50 INPUT A$
 60 REPEAT
 70 REPEAT
 80 A=128+RND(7):B=128+RND(7)
 90 UNTIL A<>B
100 B$=CHR$(A)+CHR$(157)+CHR$(B)+A$
110 PRINT B$
120 TIME=0
130 REPEAT UNTIL TIME>60
140 UNTIL FALSE
```

If you want to print everything in one foreground/background combination, say yellow on red, you can combine all the required information into one string, and then precede each PRINT statement with this string. A$ = CHR$(129) + CHR$(157) + CHR$(131) can be used before any string to print it in yellow letters on a red background, in the form PRINT A$; "Hello Bob".

You're sure to find particular colour combinations, such as yellow (or white) on red, particularly effective. Whenever you discover one which looks good on your television set (and, unfortunately, colour televisions seem to vary widely in their response to BBC Microcomputer colours), make a note of it so you can use it in a program.

If you want the letters to flash, precede the sequence with CHR$(136). The flash is turned off with CHR$(137). Change line 100 of program 13 so it reads as follows:

```
100 B$=CHR$(136)+CHR$(A)+CHR$(157)+CHR$(B)+A$
```

You'll see when you run this that your name will flash quite pleasantly.

If you want doubleheight characters on a new background, you can combine all the information into a single string:

```
10 REM DEMO 14
20 REM MORE CONCATENATION
30 A$=CHR$(141)+CHR$(129)+CHR$(157)+CHR$(132)+"HI THERE, BOB"
40 PRINT A$
50 PRINT A$
60 RUN
```

You can easily decide to have, for example, randomly coloured letters, which can be quite spectacular, as this example shows:

```
10 REM DEMO 14B
20 REM RANDOM LETTER COLOURS
30 A$=CHR$(141)+CHR$(129)+CHR$(157)+CHR$(129+RND(6))+"HI THERE, BOB"
40 PRINT A$
50 PRINT A$
60 RUN
```

Note that the colours chosen for the letters do not include red (129) to ensure that red on red does not appear.

Typing out CHR$ (whatever) can become a little tedious. It is possible to get around this by using VDU statements. The following program, which uses VDU statements, produces the same effect as program 14B.

```
10 REM DEMO 14C
20 REM USING VDU STATEMENTS
30 VDU 141,129,157,129+RND(6)
40 PRINT "HI THERE, BOB"
50 VDU 141,129,157,129+RND(6)
60 PRINT "HI THERE, BOB"
70 RUN
```

I'll now try and summarise some of the points from the preceding discussion, leading into another version of the program 'Mastermind'. There are many, many computer versions of the game. In most of these (and in this version), the computer selects a four-digit number, and the human player has to guess the number. A correct digit in a correct position in the four-digit code scores a 'black', and a correct digit in the wrong position scores a 'white'. Each digit in the code is different.

As we said a little earlier, preceding a PRINT statement with a character control code, from 129 to 135, changes the colour of the

PRINT output from that line. Enter the program, and then return to the book for a discussion on it.

```
 10 REM ** MASTERMIND **
 40 DIM C(4),G(4)
 50 MODE7
 60 PRINT '''''
 70 VDU 129,157,131
 80 PRINT "I am thinking of a four-digit number"
 90 VDU 129,157,131
100 PRINT"which you have 10 goes to discover."''
110 VDU 129,157,131
120 PRINT"All four digits are different..."'''
130 VDU 129,157,131
140 PRINT"Press any key to begin."
150 A$=GET$
160 CLS:PRINT'''
170 C(1)=RND(9)
180 FOR Z=2 TO 4:C(Z)=RND(9)
190 FOR J=1 TO Z-1:IF C(J)=C(Z) THEN 170
200 NEXT:NEXT
210 FOR G=1 TO 10: PRINT CHR$(136);CHR$(157);
          CHR$(133);"Enter guess number ";G
220 INPUT A:A1=A:PRINT CHR$(11);CHR$(11);CHR$(11)
230 FOR Z=1 TO 4:G(Z)=A-10*INT(A/10)
240 A=INT(A/10):NEXT
250 B=0:W=0
260 FOR Z=1 TO 4:IFC(Z)<>G(Z) THEN 280
270 B=B+1:G(Z)=0
280 NEXT
290 FOR Z=1 TO 4:IF G(Z)=0 THEN 330
300 FOR J=1 TO 4:IF C(Z)<>G(J) THEN 320
310 W=W+1
320 NEXT J
330 NEXT Z
340 PRINT A1;CHR$(132);"scored";CHR$(129);B
    ;" black";:IF B<>1 PRINT "s";
350 PRINT CHR$(132);"and";CHR$(129);W;
    " white";:IF W<>1 PRINT "s"
360 IF W=1 PRINT
```

```
  370 IF B=4 PRINT CHR$(129);CHR$(157);
CHR$(131);"You guessed it in just ";G;" g
uess";:IF G>1 PRINT "es" ELSE PRINT
  380 IF B<>4 NEXT G
  390 PRINT CHR$(134);"The code was";
      CHR$(129);C(4);C(3);C(2);C(1)
```

Look at line 390. This prints the words "The code was" in cyan, and C(4);C(3);C(2);C(1) in red. To change the background colour, you specify the colour (i.e. CHR$(129) for red) followed by CHR$(157) which tells the computer you want that colour on that line as a background. The CHR$(157) is followed by the control for another colour, which determines the colour of the letters printed. Line 370 prints a red (129) background with yellow (131) words.

The constant printing of CHR$'s can take time, so a VDU statement can be used instead. Look at line 70. This line — VDU 129,157,131 — takes the place of all that appears between the word PRINT and ;"You guessed it..." in line 370. We'll be looking at applications of VDU in more detail shortly, but first we must examine the other graphics modes.

As we said earlier, there are eight graphics modes, numbered from zero through to seven. The lower the number of the mode, the higher the resolution. The higher resolution modes need more memory than do the lower resolution modes. If you have a Model A machine you can only use modes 4, 5, 6 and 7.

Here is a summary of the modes:

Mode number	text/graphics	grid(across by down)	Memory needed
7	text, chunky graphics	340 X 25	1K
6	text, two colours	340 X 25	8K
5	text graphics four colours	20 X 32 160 X 256	10K
4	text graphics two colours	40 X 32 320 x 256	10K
3	text, two colours	80 x 25	16K
2	text graphics 16 colours	20 x 32 160 X 256	20K
1	text graphics four colours	40 X 32 320 x 256	20K
0	text graphics	80 X 32 640 X 256	20K

You set mode, foreground and background colours by numbers. MODE n sets the mode to n, and also clears the screen. COLOUR n (where n is less than 16), sets the foreground colour, and where n is greater than 128 (actually is n + 128), sets the background colour.

There are two colours in modes Ø, 3, 4 and 6. Although these are set initially to white (1) and black (Ø), they can be changed. There are four colours available in modes 1 and 5. As with modes Ø, 3, 4 and 6, you can change these colours, but initially they are white (3), yellow (2), red (1) and black (Ø). Mode 2 is the most generously supplied with colours, eight standard ones and eight which flash. Numbered from zero to 15, the initial colour numbers are black, red, green, yellow, blue, magenta, cyan, white, flashing black, flashing red, flashing green, flashing yellow, flashing blue, flashing magenta, flashing cyan and flashing white.

Let us look at how the colours are allotted with a simple program. Enter and run this, then return to the book for a discussion on it.

```
 10 REM MODE 6 DEMO
 20 MODE 6
 30 REPEAT
 40 COLOUR 0
 50 COLOUR 128 + 1
 60 CLS
 70 PRINT '''"DEMONSTRATION"
 80 PRINT "FOREGROUND 0, BACKGROUND 128 + 1"
 90 FOR J=1 TO 2000:NEXT
100 COLOUR 1
110 COLOUR 128 + 0
120 CLS
130 PRINT ''''''''"DEMONSTRATION"
140 PRINT "FOREGROUND 1, BACKGROUND 128 + 0"
150 FOR J=1 TO 2000:NEXT
160 UNTIL FALSE
```

As I said a few paragraphs ago, there are four colours available in modes 1 and 5. This next routine goes through all the combinations available in mode 5, and also shows the size of the text (where there are 2Ø characters across the screen).

```
10 REM MODE 5 DEMO
20 MODE 5
30 FOR N=0 TO 3
40 FOR M=3 TO 0 STEP -1
```

```
 50 COLOUR N
 60 COLOUR 128 + M
 70 CLS
 80 PRINT '''"DEMONSTRATION"
 90 PRINT '''"FOREGROUND ";N
100 PRINT '''"BACKGROUND ";M;" + 128"
110 FOR J=1 TO 3000:NEXT
120 NEXT M
130 NEXT N
```

Already, just from running these two demonstrations, you should have picked up a number of ideas regarding the use of the COLOUR command. Line 70 is needed to make the entire screen clear to the designated background colour. Take out line 70 and see what difference this makes.

If you have a model B machine, run the following program, which puts the computer through all its paces. This routine is based on the demonstration for mode 5, and you need only make a few simple changes to run it.

```
 10 REM MODE 2 DEMO
 20 MODE 2
 30 FOR N=0 TO 15
 40 FOR M=15 TO 0 STEP -1
 45 IF M=N THEN 120
 50 COLOUR N
 60 COLOUR 128 + M
 70 CLS:SOUND 1,-15,N+M*10,1
 80 PRINT '''"DEMONSTRATION"
 90 PRINT '''"FOREGROUND ";N
100 PRINT '''"BACKGROUND ";M;" + 128"
110 FOR J=1 TO 2000:NEXT
120 NEXT M
130 NEXT N
```

As you run this program (which takes quite a while), you'll see some colour combinations are particularly effective, while others make it impossible to read the text, or are just unattractive, or both. Keep a pen and paper near you when you run this, and take note of the more effective combinations.

GCOL/CLG

You use the GCOL statement to set up the screen for coloured graphics. The GCOL statement has two parameters, the first determines the nature of the manipulation of the plotted point, and the second determines the foreground (n less than 16) or background (n greater than 128) colours. CLG is the graphics version of CLS: it clears the background to the colour specified. The following routines show some of the results of using GCOL. They are in mode 5, and may be run on both model A and B machines.

```
10 REM GCOL DEMO ONE
20 MODE 5
30 FOR N=0 TO 3
40 GCOL 0,N
50 PLOT 1,RND(500),RND(500)
60 FOR J=1 TO 2000:NEXT
70 NEXT N
```

```
 10 REM GCOL DEMO TWO
 20 MODE 5
 30 FOR N=0 TO 3
 40 FOR M=3 TO 0 STEP -1
 50 GCOL 0,N
 60 GCOL 0,M+128
 70 CLG
 80 PLOT 1,RND(1000),RND(1000)
 90 FOR J=1 TO 2000:NEXT
100 NEXT M
110 NEXT N
```

GCOL determines the colour which will be used — and how it will be placed on the screen — for all graphics operations which follow it in a program. The first number after GCOL specifies the logical operation which will be performed at that plotted point on the screen, as is graphically illustrated in the following routine.

```
10 REM GCOL DEMO THREE
20 MODE 5
30 GCOL 0,1:GCOL 0,128+2:CLG
40 FOR A=200 TO 1000 STEP 40
50 FOR B=1 TO 1000 STEP 40
60 DRAW A,B:NEXT:NEXT
```

```
 70 FOR N=0 TO 4
 80 FOR M=0 TO 4
 90 MOVE 0,0:SOUND 1,-15,20*(M+N),2
100 GCOL M,N
110 FOR A=200 TO 1000 STEP 40+M+RND(4)
120 FOR B=1 TO 1000 STEP 39+N+RND(4)
130 DRAW A,B:NEXT:NEXT
140 NEXT:NEXT
150 SOUND1,-15,43,20
```

The numbers which come immediately after GCOL are 1 to 4, and the operations they perform are 'logical OR' (1), 'logical AND' (2), 'logical EOR' (3) or the colour can be inverted (4).

If you have a model B machine, you might like to run the following version of the preceding routine.

```
 10 REM GCOL DEMO THREEB
 20 MODE 2
 30 GCOL 0,1:GCOL 0,128+5:CLG
 40 FOR A=200 TO 1000 STEP 40
 50 FOR B=1 TO 1000 STEP 40
 60 DRAW A,B:NEXT:NEXT
 70 FOR N=0 TO 15
 80 FOR M=0 TO 4
 90 MOVE 0,0:SOUND 1,-15,20*(M+N),2
100 GCOL M,N
110 FOR A=200 TO 1000 STEP 40+M+4*RND(4)
120 FOR B=1 TO 1000 STEP 39+N+5*RND(4)
130 DRAW A,B:NEXT:NEXT
140 NEXT:NEXT
150 SOUND1,-15,43,20
```

This next program also shows GCOL in use. The first parameter (i.e. Ø, 1, 2 or 3) is shown in the top left hand corner, and a circle is plotted. After the program has been running for a while, and the circles start overlapping, you'll be able to clearly see the effect each of these numbers has. Run this until you are sure you understand what is going on.

```
10 REM GCOL CIRCLES
20 REM Based on program
30 REM By Jeremy Ruston
40 MODE2
```

```
 50 REM ********************
 60 REPEAT
 70 SOUND 1,-5,100,4
 80 Q=RND(4)-1
 90 PRINTTAB(3,3);Q
100 GCOL Q,RND(7)
110 PROCcircle(RND(640)+320,
      RND(512)+256,RND(300))
120 UNTIL FALSE
130 REM *******************
140 DEF PROCcircle(xcoord,ycoord,radius)
150 LOCAL angle,step
160 step=5+RND(7)
170 FOR angle=0 TO 360 STEP step
180 MOVE xcoord,ycoord
190 MOVE SIN(RAD(angle))*radius+xcoord,
     COS(RAD(angle))*radius+ycoord
200 PLOT 85,SIN(RAD(angle+step))*radius
      +xcoord,COS(RAD(angle+step))*
       radius+ycoord
210 NEXT angle
220 ENDPROC
```

If you do not have a Model B machine, run the program in mode 5. It still works, but is not as attractive.

PLOT

PLOT is a remarkably flexible statement in BBC BASIC, with over 4Ø ways to use it. The word PLOT is followed by three parameters. The first one determines the kind of PLOTting which will take place, while the last two give a location, either absolute (such as Ø,Ø being the bottom left hand corner of the screen) or relative (so 1ØØ,1ØØ is 1ØØ points up and 1ØØ points across from the last point plotted). The first parameter dictates whether the following two will be treated absolutely or relatively.

For the purposes of this book, the important first parameters after the word PLOT are the numbers zero to seven, and 8Ø to 87.

First Para meter	Effect:
Ø	This uses a PLOT statement of the form PLOT Ø,x,y to move relative to last point. The computer initialises a graphics mode (with a line like 1Ø Mode 5) by moving to the Ø,Ø point, the bottom left hand corner of the screen.
1	This draws a line from the present position, to the point specified by the two co-ordinates. The line is drawn in the current foreground colour, which is white unless another colour has been specified.
2	This draws a line as does 1, but in the logical inverse colour (with the logical inverse of Ø in a two-colour mode being 1, and with the following logical inverses in a four colour mode: Ø (3), 1 (2), 2 (1) and 3 (Ø)).
3	This performs like 1 and 2, but draws the line in the background colour.
4	This is a very useful plot command, which moves the plotting point (but without actually plotting anything) to the absolute position specified. PLOT 4,Ø,Ø will move the PLOT position to the bottom left hand corner. Because this is so useful, the word PLOT and the first parameter may be replaced by MOVE so to move to the bottom left hand corner, you need just to enter MOVE Ø,Ø.
5	This is another very useful command, which draws a line from the present plotting position to the absolute position specified (as opposed to PLOT 3 which draws the line counting from the present PLOT position). PLOT 5,x,y can be replaced with DRAW x,y.
6	This performs as does PLOT 5, but plots the line in the logical inverse colour (see the explanation of PLOT 2 for information on logical inverse colours).

7 This is a companion to PLOTs 5 and 6, drawing a line to an absolute point, but in the current background colour.

PLOTs 4, 5 and 7 can be seen at work in this little routine, which appears to make a line flash off and on, but is really plotting it alternatively in the foreground and background colours.

```
 10 REM Using PLOT 4, 5 and 7
 20 REM "Electric Spark"
 30 MODE 5
 40 REPEAT
 50 X=RND(1280)-1
 60 Y=RND(1024)-1
 70 PLOT 4,0,0
 80 PLOT 5,X,Y
 90 SOUND 15,-15,RND(4),1
100 PLOT 4,0,0
110 PLOT 7,X,Y
120 UNTIL FALSE
```

Line 30 sets the mode to 5, and lines 40 and 120 are the master REPEAT/UNTIL loop which keeps things running forever. Lines 50 and 60 choose a random co-ordinate for the end-point of the line. Line 70 moves the plotting position to 0,0 and from there line 80 plots a line to the X and Y co-ordinates chosen in lines 50 and 60. Line 90 makes an appropriate noise. This line does nothing to help the demonstration except add appropriate (?) sound effects. Line 100 moves the plot position back to 0,0 and then line 110 plots another line which is identical to the one plotted by line 80, except that it is plotted in the background colour, so the line vanishes.

This demonstration could look very effective in colour. See if you can apply what you learned in the section on GCOL to alter this routine slightly so that instead of plotting in white on a black background, it plots in yellow on a red background. When you have tried this (and only then), examine the next listing which shows how I did it. The lines between 30 and 40 are the ones I added.

```
10 REM Using PLOT 4, 5 and 7
20 REM "Electric Spark"
30 MODE 5
32 GCOL 0,2
33 GCOL 0,128+1
34 CLG
40 REPEAT
```

```
 50 X=RND(1280)-1
 60 Y=RND(1024)-1
 70 PLOT 4,0,0
 80 PLOT 5,X,Y
 90 SOUND 15,-15,RND(4),1
100 PLOT 4,0,0
110 PLOT 7,X,Y
120 UNTIL FALSE
```

Line 32 determines that the foreground colour will be yellow, line 33 that the background will be red, and line 34 clears to the graphics background.

PLOT numbers 80 to 87 behave like number 0 to 7, except that they plot and fill a triangle, using the last two points visited when filling triangles with colour.

The PLOT commands are very flexible, as the following programs demonstrate.

```
 10 REM PLOT demonstration
 20 REM Based on "Ice Cave"
 30 REM     by Norman Alm
 40 MODE4
 50 FOR N=1 TO 2
 60 K=345+RND(11)
 70 X=RND(1240):Y=RND(1024)
 80 FOR R=RND(70)+100 TO 1000 STEP RND(20)+12
 90 MOVE R,K
100 DRAW X,R
110 DRAW (X-R),Y
120 DRAW R,K
130 T=TIME:REPEAT UNTIL TIME-T>10
140 NEXT R
150 T=TIME:REPEAT UNTIL TIME-T>100
160 NEXT N
170 RUN
```

```
 10 REM *Ice Cave 2*
 20 REM Based on ATOM program
 30 REM by Norman Alm
 40 MODE5
 50 REPEAT
 60 CLG
```

```
 70 FOR T=1 TO 2
 80 A=RND(3)
 90 GCOL 0,A
100 X=RND(1280)-1
110 Y=RND(1024)-1
120 FOR R=600 TO 1300 STEP 15+RND(10)
130 MOVE R,600
140 DRAW X,R
150 DRAW (X-R),Y
160 DRAW 600,(Y-R)
170 DRAW R,600
180 NEXT R
190 NEXT T
200 T=TIME
210 REPEAT UNTIL TIME-T>60
220 UNTIL FALSE
```

```
 10 REM *Ice Cave 2*
 15 REM Adapted for Mode 2
 20 REM Based on ATOM program
 30 REM by Norman Alm
 40 MODE2
 50 REPEAT
 60 CLG
 70 FOR T=1 TO 2
 80 A=RND(7)
 90 GCOL 0,A
100 X=RND(1280)-1
110 Y=RND(1024)-1
120 FOR R=600 TO 1300 STEP 15+RND(10)
130 MOVE R,600
140 DRAW X,R
150 DRAW (X-R),Y
160 DRAW 600,(Y-R)
170 DRAW R,600
180 NEXT R
190 NEXT T
200 T=TIME
210 REPEAT UNTIL TIME-T>60
220 UNTIL FALSE
```

```
 10 REM ZEBRA TRIANGLES
 20 REM RUN IN MODE 2 ON MODEL B
 30 MODE5
 40 REPEAT
 50 GCOL RND(7),0
 60 PLOT 85,RND(1280)-1,RND(1024)-1
 70 T=TIME
 80 REPEAT UNTIL TIME-T>20
 90 IF RND(1)<.01 RUN
100 UNTIL FALSE
```

```
 10 REM ROTATING SQUARES
 20 REM BY JEREMY RUSTON
 30 REPEAT
 40 MODE1
 50 R=RND(7)+3
 60 FOR X=0 TO 999 STEP R
 70 GCOL RND(5)-1,((X/R) MOD 3)+1
 80 MOVE X,0
 90 DRAW 0,1000-X
100 DRAW 1000-X,1000
110 DRAW 1000,X
120 DRAW X,0
130 NEXT X
140 UNTIL FALSE
```

```
 10 REM SHRINKER
 20 REPEAT
 30 MODE5
 40 R=RND(17)+4
 50 FOR X=0 TO 600 STEP R
 60 MOVE X,0
 70 DRAW 0,1000-X
 80 DRAW 1000-X,1000-X
 90 DRAW 1000-X,X
100 DRAW X,0
110 SOUND 17,-10-RND(5),256-X/3,1
120 NEXT X
130 T=TIME
140 REPEAT UNTIL TIME-T>100
150 UNTIL FALSE
```

```
 10 REM MOIRE-LACE
 20 REM By Jeremy Ruston
 30 REPEAT
 40 TIME=0
 50 MODE4
 60 PROCchopper
 70 X=RND(1000)-1
 80 Y=RND(1000)-1
 90 FOR T=0 TO 999 STEP 4
100 IF TIME>500 THEN PROCchopper
110 MOVE T,0
120 PLOT 6,X,Y
130 PLOT 6,T,999
140 MOVE 0,T
150 PLOT 6,X,Y
160 PLOT 6,999,T
170 SOUND 1,-RND(7),230+RND(25),1
180 NEXT T
190 TIME=0
200 REPEAT UNTIL TIME>200
210 UNTIL FALSE
220 DEF PROCchopper
230 SOUND 1,-15,RND(30),10
240 SOUND 2,-15,RND(30)+30,10
250 SOUND 3,-15,RND(30)+60,10
260 LOCAL A,B
270 A=RND(8)-1
280 REPEAT
290 B=RND(8)-1
300 UNTIL B<>A
310 VDU 19,1,A,0,0,0,19,0,B,0,0,0
320 TIME=0
330 ENDPROC
```

```
10 REM SINE CURVE
20 REM NOTE HOW FOREGROUND
30 REM AND BACKGROUND COLOURS
40 REM ARE DEFINED USING VDU
50 REM SEE FOLLOWING SECTION ON
60 REM USE OF VDU
70 MODE 4
```

```
 80 INPUT'''"Background number  ",A
 90 INPUT'''"Foreground number  ",B
100 IF A=B OR A<0 OR A>15
            OR B<0 OR B>15 THEN 70
110 VDU 19,1,B,0,0,0
120 VDU 19,0,A,0,0,0
130 CLG
140 FOR X=0 TO PI*2.53 STEP 0.01
150 MOVE 160*X,SIN(X)*460+412
160 DRAW 160*X,SIN(X)*460+512
170 SOUND 17,-15,SIN(X)*20+100,3
180 SOUND 18,-15,SIN(X)*20+101,3
190 SOUND 19,-7,255-SIN(X)*20+101,3
200 NEXT X
```

```
 10 REM **Tunnel Tuner**
 20 REM RUN IN MODE 2 ON MODEL B
 30 MODE2
 40 Z=2
 50 A=RND(300)
 60 REPEAT
 70 IF RND(1)>0.8 Z=RND(4)
 80 MOVE0,0
 90 DRAW Z*A,A
100 SOUND 17,-15,A,3
110 PROCchange
120 DRAW Z*A,Z*A
130 SOUND 18,-15,A,3
140 PROCchange
150 DRAW A,Z*A
160 SOUND 19,-15,A,3
170 PROCchange
180 DRAW A,A
190 PROCchange
200 GCOL 0,RND(7)-1
210 IF RND(9)<3  A=RND(200)
220 UNTIL FALSE
230 DEF PROCchange
240 A=A+RND(9)-RND(9)
250 ENDPROC
```

```
 10 REM SINE RIBBON
 20 MODE4
 30 VDU 19,0,0,0,0,0
 40 VDU 19,0,4,0,0,0
 50 C=100
 60 REPEAT
 70 MOVE C-100,0
 80 FOR A=1 TO 60
 90 B=A*PI/20
100 DRAW A*20+C,SIN(B)*310+320
110 SOUND 17,-15,255-A,3
120 NEXT A
130 C=C+3
140 UNTILFALSE
```

```
 10 REM Orbital Sketcher
 20 MODE 4
 30 REPEAT
 40 VDU19,0,0,0,0,0,19,0,RND(6),0,0,0
 50 M=RND(400)
 60 N=RND(400)
 70 MOVE 600+M,500
 80 FOR A=1 TO 205 STEP 5
 90 B=PI*A/100
100 C=M*COS(B)+600
110 D=N*SIN(B)+500
120 DRAW C,D
130 NEXT
140 IF RND(10)=5 CLG
150 IF RND(10)>4 THEN 50
160 UNTIL FALSE
```

```
 10 REM ROLLER COASTER
 20 MODE4
 30 X=1
 40 VDU 19,1,0,0,0,0
 50 C=100
 60 REPEAT
 70 T=RND(2)
 80 Q=120
 90 P=368
```

```
100 VDU 19,0,X,0,0,0
110 MOVE C-100,0
120 FOR A=1 TO 60
130 B=A*PI/20
140 MOVE Q,P
150 Q=A*20+C
160 P=SIN(B)*310+320
170 IF T=1 PLOT 7,A*20+C-50,SIN(B)*310+320
180 DRAW A*20+C,SIN(B)*310+320
190 SOUND 18,-A,255*SIN(B),10
200 SOUND 17,-A,250*SIN(B),10
210 NEXT
220 C=C+2
230 X=RND(7)
240 UNTIL FALSE
```

```
 10 REM COSMIC TABLE MAT
 20 REM ADAPTED FROM 'Spiraliser'
 30 REM By Jim Walsh and Paul Holmes
 40 REM RUN IN MODE 5 ON MODEL A
 50 MODE 2
 60 MOVE 600,500
 70 REPEAT
 80 FOR N= 1 TO 500 STEP RND(5)/RND(5)
 90 A=INT (200/200*N)
100 GCOL 0,A MOD 8
110 DRAW 600-A*COS(N/6*PI),500+A*SIN(N/6*PI)
120 NEXT N
130 Q=RND(4)-1
140 FOR N=500 TO 1 STEP -RND(5)/RND(5)
150  A=INT (200/200*N)
160 GCOL Q,A MOD 8
170 DRAW 600-A*COS(N/6*PI),500+A*SIN(N/6*PI)
180 NEXT N
190 IF RND(5)=2 CLG:MOVE 600,500
200 UNTIL FALSE
```

```
 10 REM COSMIC TABLE MAT
 15 REM MARK 11 (more adventurous)
 20 REM ADAPTED FROM 'Spiraliser'
 30 REM By Jim Walsh and Paul Holmes
```

```
 40 REM RUN IN MODE 5 ON MODEL A
 50 MODE 2
 70 REPEAT
 72 G=8+RND(3)-RND(3)
 75 MOVE 600,500
 80 FOR N= 1 TO 540 STEP RND(5)+2
 90 A=INT (200/200*N)
100 GCOL 0,A MOD G
110 DRAW 600-A*COS(N/6*PI),500+A*SIN(N/6*PI)
120 NEXT N
130 Q=RND(4)-1
140 FOR N=540 TO 1 STEP -RND(3)*2
150  A=INT (200/200*N)
160 GCOL Q,A MOD G
170 DRAW 600-A*COS(N/6*PI),500+A*SIN(N/6*PI)
180 NEXT N
190 CLG
200 UNTIL FALSE
```

```
 10 REM COSMIC TABLE MAT
 15 REM MARK 111 (with sound!)
 20 REM ADAPTED FROM 'Spiraliser'
 30 REM By Jim Walsh and Paul Holmes
 40 REM RUN IN MODE 5 ON MODEL A
 50 MODE 2
 70 REPEAT
 72 G=8+RND(3)-RND(3)
 75 MOVE 600,500
 80 FOR N= 1 TO 540 STEP RND(5)+2
 90 A=INT (200/200*N)
 95 SOUND 18,-15,N/2,10
 96 SOUND 17,-7,1000 MOD (A+12/N),10
100 GCOL 0,A MOD G
110 DRAW 600-A*COS(N/6*PI),500+A*SIN(N/6*PI)
120 NEXT N
130 Q=RND(4)-1
140 FOR N=540 TO 1 STEP -RND(3)*2
150  A=INT (200/200*N)
155 SOUND 18,-15,1000 MOD (A+5/N),10
156 SOUND 17,-15,N/2,10
```

```
160 GCOL Q,A MOD G
170 DRAW 600-A*COS(N/6*PI),500+A*SIN(N/6*PI)
180 NEXT N
190 CLG
200 UNTIL FALSE
```

VDU drivers

The VDU driver is a piece of software through which all characters that are going to be printed are sent. The characters are sent as codes between 0 and 255. The effects of the codes are different, depending on their values. All codes of 32 or greater are printed on the screen, and all codes less than 32 are acted upon in different ways, doing things like moving the cursor around. In addition, code 127 is the same as the delete key, and so fits in the latter category.

The exact meanings of the codes 0 to 31 are described below:

(Note that using this table you can see what control key to press to gain the same result. For example, code 20 can be generated by using VDU 20, or by pressing control-T, or even by PRINT CHR$(20).)

```
 0----@
 1----A
 2----B
 3----C
 4----D
 5----E
 6----F
 7----G
 8----H
 9----I
10----J
11----K
12----L
13----M
14----N
15----O
16----P
17----Q
18----R
```

```
19----S
20----T
21----U
22----V
23----W
24----X
25----Y
26----Z
27----[
28----\
29----]
30----^
31----_
```

I have only described in detail those codes that are useful and not explained fully in the User Guide.

Code Use

2 Used to stop character being sent to the printer.

3 Starts sending characters to the printer. Thus, to list a program to the printer, you use control-B, then type 'LIST', then press 'return'. When the listing is complete, you just press control-c to turn off the printer, and you're done.

4 Turns off the effect of code 5. See code 5.

5 Joins the text and graphics cursors. This means that text is printed at the last graphics point visited, and that the graphics cursor can be moved about with codes 8 to 11. This program shows you how it works.

```
10 MODE 4
20 VDU 5
30 FOR T=1 TO 40
40 MOVE RND(1280),RND(1024)
50 PRINT "BBC Computer 32K"
60 NEXT T
70 VDU 4
```

The User Guide shows you how to use the VDU 5 feature to make an accent on a letter — this is an important use of the feature. One disadvantage with the VDU 5 mode is that text is printed very slowly, and scrolling does not normally take place. Another useful way of using VDU 5 is in the labelling of axes in

graph drawing. The biorhythm program in the 'welcome' program collection illustrates this use. While in VDU 5 mode, text printing only takes place in the current graphics window. Windows are explained in the discussion of codes 24 and 28.

7 This code just causes a short 'bleep' to be added to the sound queue. This program shows one use of it.

```
10 FOR T=1 TO 10
20 VDU 7
30 NEXT T
```

8–11 These codes move the cursor left, right, down and up respectively. For example, this section of code moves the cursor left, to correct the error in line 10:

```
10 A$="BBC Computer 32J"
20 PRINT A$;CHR$(8);"K"
```

Assuming you are conversant with the effect of *FX4, 1 on the cursor control keys, you may have noticed that the codes created by them in this state are 128 greater than the cursor movement codes. Thus this program allows you to type on the screen, shifting the cursor around with the cursor control keys.

```
10 *FX 4,1
20 REPEAT
30 A=GET
40 IF A>127 A=A-128
50 VDU A
60 UNTIL FALSE
```

13 Code generated by pressing 'return'. Returns the cursor to the start of the current line. BASIC usually puts a cursor down in after this character.

14 Turns page mode on. You will often use this mode to list programs too long to fit on the screen, so it can be done in sections. When page mode is active, the computer waits for you to press one of the two shift keys before printing each new page, to give you time to assimilate the text being printed. This program shows VDU 14 by printing 100 random numbers. The trouble is that they scroll off the screen too fast for you to read them. The second time, however, page mode is on, which gives you ample time to read the numbers.

```
10 FOR T=0 TO 100
20 PRINT RND
```

```
 30 NEXT T
 40 PRINT "PRESS ANY KEY ..."
 50 DUMMY=GET
 60 VDU 14
 70 FOR T=0 TO 100
 80 PRINT RND
 90 NEXT T
100 VDU 15
```

19 VDU 19.... enables you to change the colour of an area or spot of colour of the screen, without having to redraw the object.

To use this command effectively, you have to think about colour graphics in a different way. Rather than consider blobs of colour, think of the screen being divided into a number of squares, where each square, or pixel, can hold a number. This number is restricted to magnitude by the graphics mode you are using:
Mode Ø — Ø to 1
Mode 1 — Ø to 3
Mode 2 — Ø to 15
Mode 3 — Ø to 1
Mode 4 — Ø to 1
Mode 5 — Ø to 3
Mode 6 — Ø to 1

The machine is set up in such a way that each number corresponds to a specific colour. In modes Ø,3,4 and 6, a pixel holding Ø will appear black and one holding 1 will appear white. In modes 5 and 1 the relationships are:

Ø — black
1 — red
2 — yellow
3 — white

In mode 2, the relationships are:
Ø — black
1 — red
2 — green
3 — yellow
4 — blue
5 — magenta (purple)
6 — cyan (light blue)
7 — white
8 — flashing white/black
9 — flashing red/cyan
1Ø — flashing green/magenta
11 — flashing yellow/blue
12 — flashing blue/yellow

13 — flashing magenta/green
14 — flashing cyan/red
15 — flashing white/black

The intriguing possibilities soon become apparent once you realise that you can change these relationships by executing a command of the form VDU 19, colour_number, colour, Ø,Ø,Ø where 'colour_number' is a number in the range outlined in the first table. Colour is a variable in the range Ø to 15. After this statement has been executed, any pixels of holding the number 'colour_number' will assume the colour given by 'colour' according to the second table of this section.

There are many applications for this command. If you first set the pixels of a graphics screen to appear black, no matter what number they hold, and you then draw a complex shape or pattern before restoring the normal relationships with VDU 2Ø, it will appear as if the drawing was done instantaneously.

For example, this program draws a circle with the screen turned off, and then restores all the normal colours very quickly. The effect can be quite dramatic:

```
 10 MODE5
 20 FOR colour=1 TO 3
 30 VDU 19,colour,0,0,0,0
 40 NEXT colour
 50 MOVE 640,512+500
 60 FOR angle=0 TO 330 STEP 30
 70 SOUND 1,-7,angle/2,1
 80 GCOL 0,1
 90 MOVE 640,512
100 PLOT 85,SIN(RAD(angle+10))*500+640,
        COS(RAD(angle+10))*500+512
110 GCOL 0,2
120 MOVE 640,512
130 PLOT 85,SIN(RAD(angle+20))*500+640,
        COS(RAD(angle+20))*500+512
140 GCOL 0,3
150 MOVE 640,512
160 PLOT 85,SIN(RAD(angle+30))*500+640,
        COS(RAD(angle+30))*500+512
170 NEXT angle
180 VDU 20
```

As a logical step from this example, if you draw many similar figures, all set to black, and then in rotation set each to a specific colour and then back to black again, the illusion of movement is created. This example should make the technique clear:

```
 10 REM By Jeremy Ruston
 20 MODE5
 30 FOR X=0 TO 1279 STEP 20
 40 MOVE X,0
 50 GCOL 0,((X DIV 20) MOD 3)+1
 60 DRAW X,1023
 70 NEXT X
 80 REPEAT
 90 FOR colour=1 TO 3
100 SOUND 18,-15,75*colour,10
110 VDU 19,colour,4,0,0,0
120 Q=RND(25)
130 TIME=0
140 REPEAT UNTIL TIME=Q
150 VDU 19,colour,0,0,0,0
160 NEXT colour
170 UNTIL FALSE
```

The program works by drawing a series of vertical lines in each of the three colours in turn. Then each colour is selectively turned to blue — line 11Ø. A time consuming loop, lines 13Ø to 14Ø is set up before the colour is reset to black. This technique works well in mode 2, where you can use 15 colours. Without this command it would be impossible to have any colours other than black and white in modes Ø,4,6, and 3. Thus if you like the idea of purple text on a yellow background, execute this: VDU 19,Ø,3,Ø,Ø,Ø,19,1,5,Ø,Ø,Ø. Make the five be 23, and try this in mode 6, and see what happens.

24 This code enables you to restrict all graphics commands to operate in a rectangular section of the screen. This section usually consists of the whole screen. The rectangle is stated in terms of the coordinates of its bottom left hand corner and its top right corner. The four numbers have to be separated with semi-colons, and not commas, as you might expect. This program draws two sets of random lines — one with no screen window in effect, and one with a graphics window. The program expects a key press after the first session, to tell it to start the next lot of lines.

```
 10 MODE 4
 20 FOR T=1 TO 50
 30 MOVE RND(1280),RND(1024)
 40 DRAW RND(1280),RND(1024)
 50 NEXT T
 60 A=GET
 70 MODE 4
 80 VDU 24,200;200;700;700;
 90 FOR T=1 TO 500
100 MOVE RND(1280),RND(1024)
110 DRAW RND(1280),RND(1024)
120 NEXT T
```

28 This code is like the previous one, except that it defines a window within which text is restricted. The coordinates of the window are given in the same way as above, i.e. bottom left and then top right. Remember that the origin for text is the top left hand corner of the screen. Commas are used to separate the data. This program fills the screen with random letters and then defines a text window in the middle of the screen. The program ends without clearing the screen. When it has finished, type CLS, then press 'return'. Try listing the program to see what happens. Do not make the coordinates given in the command too large for the current screen mode.

```
10 MODE 7
20 FOR T=0 TO 799
30 VDU RND(26)+64
40 NEXT T
50 A=GET
60 VDU 28,5,6,34,3
```

26 Returns all the graphics and text windows to normal. Control Z is normally more convenient.

31 Used to move the cursor to the position X,Y. Used in this format: VDU 31,X,Y.

23 Redefines text characters. Use this program to redefine the character set by the variable CHAR. The pattern for the character should be set up in A$(). P should be set to TRUE for just printing the correct VDU command to redefine the character, and FALSE to actually do the redefinition. An example is given in the text of the program. Use 'MODE 5, PRINT CHR$(255)' to examine the character you defined, presuming it is character 255.

```
10 REM Character redefinition
20 REM Copyright (C) Jeremy Ruston
30 P=FALSE
40 DIM A$(8)
50 A$(1)="........"
60 A$(2)="*******."
70 A$(3)="*.....*."
80 A$(4)="*.***.*."
90 A$(5)="*.*.*.*."
100 A$(6)="*.***.*."
110 A$(7)="*.....*."
120 A$(8)="*******."
130 CHAR=230
140 IF P THEN PRINT "VDU 23,";CHAR;
                     ELSE VDU 23,CHAR
150 FOR ROW=0 TO 7
160 TEMP=0
170 FOR COL=0 TO 7
180 IF MID$(A$(ROW+1),COL+1,1)="*"
           THEN TEMP=TEMP+2^(7-COL)
190 NEXT COL
200 IF P THEN PRINT ;",";TEMP;
                     ELSE VDU TEMP
210 NEXT ROW
220 PRINT
230 END
```

Finally, here are two more programs which use a VDU statement for rather unusual results. Once you've run them, and seen what they do, try and work out how they achieve this. Notice that the second program must be exited via ESCAPE, then by typing in MODE 7, then pressing RETURN.

```
10 REM SIDEWAYS SCROLL
20 REM BY JEREMY RUSTON
30 REM MODEL B ONLY
40 MODE1
50 FOR X=0 TO 999 STEP 20
60 GCOL 0,((X/20) MOD 3)+1
70 MOVE X,0
80 DRAW 0,1000-X
90 DRAW 1000-X,1000
```

```
100 DRAW 1000,X
110 DRAW X,0
120 NEXT X
130 REPEAT
140 FOR T=0 TO 79
150 VDU 23,0,13,T,0;0;0;
160 NEXT T
170 FOR T=79 TO 0 STEP -1
180 VDU 23,0,13,T,0;0;0;
190 NEXT T
200 UNTIL FALSE
```

```
 10 REM CORRUPTION
 12 REM ENTER MODE 7 AFTER ESCAPE
 20 REM BY JEREMY RUSTON
 30 REM MODEL B ONLY
 40 MODE1
 50 FOR X=0 TO 999 STEP 200
 60 GCOL 0,((X/20) MOD 3)+1
 70 MOVE X,0
 80 DRAW 0,1000-X
 90 DRAW 1000-X,1000
100 DRAW 1000,X
110 DRAW X,0
120 NEXT X
130 REPEAT
140 FOR T=0 TO 79
150 VDU 23,0,12,T,0;0;0;
160 NEXT T
170 FOR T=79 TO 0 STEP -1
180 VDU 23,0,12,T,0;0;0;
190 NEXT T
200 UNTIL FALSE
```

Introduction to arithmetic on the computer

Have a quick glance at this section before you read it. You may well find it has no new information for you. If this is the case, feel free to turn to the next section.

The symbols for the various operations in BBC BASIC are probably well known to you by now. They are multiply (*), divide (/), subtract (−), add (+), raise to the power (^). The computer follows a strict heirachy of operations. Anything within brackets is worked out first, and the material within brackets follows the same heirachy as material outside the brackets. Brackets may be nested, with the calculations within the innermost pair being worked out first.

Raising to a power has the highest priority, followed by multiplication and division, with addition and subtraction last. A number is assumed to be positive unless it is preceded by a minus sign. Similarly, unless a decimal point appears within a number, the computer will assume it is an integer. Although you can use decimal points when working with the computer, commas are not allowed. The use of *scientific notation* for very large and very small numbers was explained in the section on variables. Refer back to that if you need a reminder on how this works.

The BASIC on the BBC Microcomputer works very, very quickly, as can be seen by running the following programs. The first program in this section works out arithmetic progressions. You must enter the first term, the common difference, and the number of terms, and the computer will produce the information for you instantly.

```
 10 REM **Arithmetic progression**
 20 MODE7
 30 Z=RND(6)+128
 40 PRINT '''CHR$(Z);"I'll now work out for you the"
 50 PRINT 'CHR$(Z);"arithmetic progression from the"
 60 PRINT 'CHR$(Z);"information you give me"
 70 INPUT"Enter the first term "FIRST
 80 INPUT"And now the common difference "DIFF
 90 INPUT"How many terms? "TERMS
100 TERMS=INT(TERMS+.5)
110 CLS
120 PRINT''CHR$(Z);"Arithmetic progression"
130 PRINT CHR$(Z);"***********************"
140 PRINT'CHR$(Z);"Term number     Value"
150 count=0
160 FOR loop= 0 TO TERMS-1
170 W=loop+1
180 Q=FIRST+(loop*DIFF)
```

```
190 count=count+Q
200 PRINT W,Q
210 NEXTloop
220 PRINT 'CHR$(Z);TAB(8);"The sum is ";count
```

```
>RUN
I'll now work out for you the
arithmetic progression from the
information you give me
Enter the first term 66
And now the common difference 3
How many terms? 12
Arithmetic progression
**********************
Term number     Value
         1         66
         2         69
         3         72
         4         75
         5         78
         6         81
         7         84
         8         87
         9         90
        10         93
        11         96
        12         99
       The sum is 990
```

As you can see, the program also works out the sum of the terms.

```
I'll now work out for you the
arithmetic progression from the
information you give me
Enter the first term .0034
And now the common difference .00012
How many terms? 13
Arithmetic progression
**********************
Term number     Value
         1     3.4E-3
         2    3.52E-3
         3    3.64E-3
```

```
 4    3.76E-3
 5    3.88E-3
 6       4E-3
 7    4.12E-3
 8    4.24E-3
 9    4.36E-3
10    4.48E-3
11     4.6E-3
12    4.72E-3
13    4.84E-3
The sum is 5.356E-2
```

The next program in this section determines the moment of inertia, polar moment of inertia and the area connected within a circular section. All you have to do is enter the radius.

```
 10 REM **Circular sections**
 20 MODE7:Z=RND(6)+128
 30 PRINT '''CHR$(Z);"This program will work out the moment"
 40 PRINT'CHR$(Z);"of inertia, polar moment of interia ,"
 50 PRINT'CHR$(Z);"and the area connected within"
 60 PRINT'CHR$(Z);TAB(5);"a circular section"
 70 PRINT''''CHR$(Z);"Please enter the radius"
 80 INPUT radius
 90 X=2*radius:M=PI
100 PRINT'CHR$(Z);"The moment of inertia is ";(M*(X^4))/64
110 PRINT'CHR$(Z);"The polar moment of interia is"'
     CHR$(Z),2*(M*(X^4))/64
120 PRINT'CHR$(Z);"The area of section is ";(M*(X*X))/4
```

```
This program will work out the moment
of inertia, polar movement of inertia,
and the area connected within
     a circular section
Please enter the radius
?35
The moment of inertia is 1178588.12
The polar moment of interia is
           2357176.24
The area of section is 3848.451

This program will work out the moment
of inertia, polar movement of inertia,
and the area connected within
     a circular section
Please enter the radius
?1
```

```
The moment of inertia is 0.785398163
The polar moment of interia is
          1.57079633
The area of section is 3.14159265
```

Prime numbers are very easy to determine.

```
 10 REM +Prime numbers+
 20 MODE7:Z=128+RND(6)
 30 PRINT ''''CHR$(Z);"Enter the value of the maximum"
 40 PRINT 'CHR$(Z);"prime number you want"
 50 INPUT A:IF A<1 THEN 50
 60 DIM PRIME(A):KL=A
 70 FOR J=1TOA:PRIME(J)=J:NEXT
 80 IF A<4 THEN PROCprint_out:END
 90 PRIME(4)=5
100 KL=4:IZ=5
110 IZ=IZ+2:IF IZ>A THEN PROCprint_out:END
120 JO=3
130 EX=IZ/PRIME(JO)
140 IF EX=INT(EX) THEN 110
150 IF EX<PRIME(JO+1) THEN 180
160 JO=JO+1
170 GOTO 130
180 KL=KL+1:PRIME(KL)=IZ:GOTO110
190 REM*********************
200 DEF PROCprint_out
210 CLS
220 PRINT '''CHR$(Z);"The prime numbers up to ";A;" are:"
230 PRINT 'CHR$(Z);TAB(17);"Prime #   Prime"
240 FOR count=1 TO KL:PRINT CHR$(Z),count,PRIME(count):NEXT
250 ENDPROC
```

```
Enter the value of the maximum
prime number you want
?87
The prime numbers up to 87 are:
                Prime #   Prime
                   1         1
                   2         2
                   3         3
                   4         5
                   5         7
                   6        11
                   7        13
                   8        17
                   9        19
                  10        23
                  11        29
```

```
12          31
13          37
14          41
15          43
16          47
17          53
18          59
19          61
20          67
21          71
22          73
23          79
24          83
```

The mathematical ability of the computer can also, of course, be turned to produce other kinds of information, such as the day of the week a specified date falls on.

```
 10 REM *Day of the week*
 20 A$=".,MONTUEWEDTHUFRISATSUN"
 30 INPUT "Day? "D
 40 IF D<1 OR D>31 THEN 30
 50 INPUT "Month? (as 7) "M"
 60 IF M<1 OR M>12 THEN 50
 70 INPUT "Year? (as 1983) "Y
 80 Q=Y-(M<3)
 90 K=Q/100
100 T=M+12*(M<3)
110 R=INT(13*(T+1)/5)+INT(5*Q/4)-INT(K)+INT(K/4)+D+5
120 R=R-(7*INT(R/7))+1
130 PRINT D;"/";M;"/";Y-1900;" - ";MID$(A$,R*3,3)
```

```
Day? 25
Month? (as 7) 12
Year? (as 1983) 1984
        25/12/84 - TUE

Day? 1
Month? (as 7) 1
Year? (as 1983) 1999
         1/1/99 - TUE
```

The final program in this section uses the computer to simulate the life cycles of two species, one of which preys upon the other, and to graph their relative populations. The relationship between the two species is controlled by a differential equation. You enter the starting populations, as numbers between one and nine. Fractions are acceptable, and it is fascinating to enter a very low population for one

of the animals, and a high one for the other, and watch the two evolve. When the program has run through a specificed number of generations, it will stop and display a question mark on the screen. This is so you can enter another starting population for the first species. Press RETURN and a second question mark will apear for the starting level of the second species. The development of this relationship will then be graphed, on top of the existing graph, so you can build up a number of graphs showing the effects of different starting populations for the predator and its prey.

```
 10 REM **SPECIES**
 20 MODE7
 30 INPUT''''"How many of species one",X
 40 INPUT''''"And how many of species two",Y
 50 MODE5
 60 GCOL 0,2
 70 REPEAT
 80 MOVE 50*(X+5),50*(12-Y)
 90 FOR Z=1 TO 12
100 FOR T=1 TO 7 STEP 0.25
110 PRINT TAB(1,1);INT(X*10000);"   "
120 PRINT TAB(1,2);INT(Y*10000);"   "
130 X=X+(4*X-2*X*Y)*0.01
140 Y=Y+(X*Y-3*Y)*0.01
150 PLOT 5,50*(X+5),50*(12-Y)
160 NEXTT:NEXTZ
170 INPUT X
180 INPUT Y
190 PRINT TAB(0,3);"     ";TAB(0,4);"     "
200 UNTIL FALSE
```

Functions

The BBC Microcomputer's dialect of BASIC, in common with other BASICs, contains a number of preprogrammed functions which you can use in a program, or in the direct mode. As well as the programmed functions, you can create your own, with the DEF FN (DeFine FunctioN) command. In this article, we will look at the functions which come with the BASIC, as well as discussing the use of DEF FN. The discussion includes a program which uses a defined function to draw a picture of a bat! General functions:

ABS — This function, ABSolute, gives the value of X, ignoring the sign, so that if X was — 1Ø, ABS(X) would be 1Ø. Similarly, if X was 1Ø, ABS(X) is still 1Ø.

INT — The INT functions gives the whole number, or INTeger part of a number, giving the largest number which is not greater than X. If X was 2.42, INT(X) would be 2.

RND — This is used to generate a RaNDom number. If X was 2Ø, RND(X) could be 13, 7, 4, 2Ø, or any whole number between one and 2Ø. RND(1) gives a random number between zero and one.

SGN — This function returns the SiGN of the variable in brackets, the SiGN of the *argument* as this variable is known. If X equals 2Ø, that is, X is a positive number, SGN(X) = 1. SGN(−2Ø) = −1. SGN(Ø) = Ø.

TAB — As pointed out earlier in the book, this is the TABulating function, which moves the PRINT position across the line the number of spaces indicated by the argument of the function. Thus, PRINT TAB(7);"£" will print the £ at the seventh position across from the left hand edge, while PRINT TAB(14);"£" will print it 14 spaces across. The direction down the screen can also be specified, by adding a second argument after a comma within the brackets. Thus, PRINT TAB(4,9);"£" will print a pound sign four spaces across, and nine down.

EXP — This function gives the value of e raised to the power of the argument, so PRINT EXP(5) will give 148.413159.

LOG — This calculates the common logarithm of a number to base 1Ø, so PRINT LOG(X) where X is five will yield Ø.6989700Ø4, whereas LN (X) yields the natural logarithm to base e, so PRINT LN(5) gives 1.6Ø943791.

SQR — This function yields the SQuare Root of a number, so when X is five, PRINT SQR(X) gives 2.236Ø6798

Trigonometrical functions:

SIN — This gives the sine of an angle in radians. SIN(5) yields −Ø.958924274.

COS — Yields the cosine of an angle in radians. PRINT COS(X) where X equals five gives Ø.283662185.

TAN — Produces the tangent of angle X in radians, so PRINT TAN(X) where X equals five produces 1.3734ØØ77.

It is likely that you won't be used to measuring angles in radians. PI radians equals 18Ø degrees. Fortunately, the BBC Microcomputer has another trigonometrical function, called RAD which converts from degrees to radians for you. The argument of the function must be in radians. The DEG (DEGree) function works the other way, converting angles expressed in radians into degrees.

```
10 REM RADIANS TO DEGREES
20 REPEAT
30 INPUT "ANGLE IN RADIANS",X
40 PRINT ;X" RADIANS IS "DEG(X)" DEGREES"
50 UNTIL FALSE
```

Defining functions

This feature allows you to DEFIne functions within a program, which you can then call whenever you need to while running the program. DEF FN can save space as well as time, as complex calculations can be defined with a short name, and called up at will by use of this name.

There are four things in the statement which defines the function:

- The word DEF
- The name of the function, which consists of the letters FN, followed by the name
- The argument of the function which follows the name, in brackets
- The formula, using the argument, for working out the function.

This sounds a lot more complicated than it is in practice. Look at this program.

```
10 REM DEFINE A FUNCTION
20 DEF FNA(Z)=Z*Z
30 REPEAT
40 INPUT Z
50 PRINT FNA(Z)
60 UNTIL FALSE
```

Line 2Ø defines a function A, with the argument Z as being Z squared. Then, whenever the program comes across FNA(Z), it will square the value assigned to the variable Z. You can see this in the demonstration.

The next program defines a function such that the argument (which you enter in line 4Ø) is multiplied by 2.178 and divided by the square root of the integer of itself. This function, as a moment's thought will show, will default if the argument is less than zero.

```
10 REM DEFINE A FUNCTION
20 REPEAT
30 DEF FNA(Z)=2.178*Z/SQR(INT(Z))
40 INPUT Z
50 PRINT FNA(Z)
60 UNTIL FALSE
```

Look to the next program — BAT — in which a function is defined in line 6Ø. The function bat(B) gets the square root of the difference between the squares of two variables, and in the routine 12Ø to 21Ø, uses the value H (see line 13Ø) to determine the printing positions of the dots which will draw up the bat. PROC delay (a procedure), defined from line 27Ø, is there simply to slow things down, and produce some bat-like sounds.

```
 10 REM "BAT"
 20 REM SHOWING DEF FN
 30 MODE7
 35 REM LINE 40 TURNS OFF
 36 REM  THE CURSOR
 40 VDU23;8202;0;0;0
 50 L=0:P=11:Q=17
 60 DEF FNbat(B)=SQR(L*L-B*B)
 70 PRINT CHR$(12)CHR$(30)
 80 PRINT TAB(Q,P);"O"
 90 REPEAT
100 PRINT TAB(16,9);"! !"
110 L=L+1
120 FOR B=0 TO L
130 H=FNbat(B)
140 PRINT TAB(Q+H,P+B);"."
150 PROCdelay
160 PRINT TAB(Q-H,P+B);"."
170 PROCdelay
```

```
180 PRINT TAB(Q-H,P-B);"."
190 PROCdelay
200 PRINT TAB(Q+H,P-B);"."
210 NEXTB
220 PROCdelay
230 UNTIL L=11
240 REPEAT
250 PROCdelay
260 UNTIL FALSE
270 DEF PROCdelay
280 W=TIME
290 SOUND 1,-15,RND(6)+249,3
300 REPEAT
310 UNTIL TIME-W>15
320 ENDPROC
```

LOCAL VARIABLES

It is important to ensure that all variables used within functions for temporary results, or passing the function's value to the calling statement are defined as 'local'. This ensures that any variables that you've used inside the function that are also used outside the function definition will be treated as different entities. Thus you can use variables such as X and Y in a function for their logical coordinate purposes, without them interfering with a possible X and Y elsewhere in the program.

Jeremy Ruston, author of THE BBC MICRO REVEALED, points out that it is good practice to make the first line after the function definition a blank LOCAL statement. Then, when the function has been written, you can fill in the required variables into the LOCAL statement. This serves as a useful memory aid to prevent the inadvertent omission of the LOCAL statement.

We'll be discussing the use of LOCAL variables again in a few pages times in the section on procedures.

There is a variation to the kind of user defined function we've just discussed. Often the value you wish the function to take on cannot be calculated in a single line, maybe because a loop is required. In this case, you can use functions which are multi-line. The spirograph program, which follows uses a multi-line function.

```
 10 REM ********************
 20 ON ERROR GOTO 20
 30 REPEAT
 40 MODE 4
 50 VDU 29,640;512;
 60 A=RND(200)+250
 70 B=A-RND(A-50)
 80 C=A-RND(A-10)
 90 PROCSPIRO(A,B,C)
100 TIME=0
110 REPEAT UNTIL TIME>300
120 UNTIL FALSE
130 END
140 REM ********************
150 DEF PROCSPIRO(A,B,D)
160 E=D
170 F=(A-B)
180 G=0
190 H=PI*0.02
200 K=A/B
210 N=B/FNHCF(A,B)
220 L=100*N
230 MOVE F+E,0
240 FOR I=1 TO L
250 G=G+H
260 P=G*K
270 X=F*COS(G)+E*COS(P)
280 Y=F*SIN(G)-E*SIN(P)
290 DRAW X,Y
300 NEXT I
310 ENDPROC
320 REM ********************
330 DEF FNHCF(I,J)
340 LOCAL M
350 REPEAT
360 M=I MOD J
370 I=J
380 J=M
390 UNTIL M=0
400=I
410 REM ********************
```

The differences between this function at line 330 and the normal functions we looked at include:

1. It has taken two arguments, rather than a single argument.
2. There are extra statements between the final equals sign (line 400) and the end of the function heading.

Instead of using something like the ENDPROC statement, functions are terminated with a single equals sign assignment, line 400. Then, when the function is called, in line 210 of the procedure, it finds the highest common factor of the two numbers and returns that facts via the variable 'I'.

Try to make all routines which return a result into a fraction, and make the functions fairly short. It is possible to use functions which do not take any arguments. However, if you ever write a function which gives a null, or irrelevant result, used purely as a dummy, you should be writing a procedure. This leads us neatly into a discussion of procedures.

Procedures

In essence, procedures are subroutines which you call by name, rather than by line number. They have the advantage that you can use local variables within them, and they can be located far more quickly by the computer than can subroutines. The computer knows where the procedure is, so it does not have to search through a whole program, line number by line number, looking for the start of a subroutine.

To call a procedure from within a program, you just include the line PROCname_of_procedure. Somewhere else in the program you must define the procedure, which you do by having an opening line DEF PROCname_of_procedure. After doing whatever you want done within the procedure, you end the definition section with a line reading ENDPROC. The following simple example should help make it clear.

```
10 REM USING PROCEDURES
20 REPEAT
30 PROCprint_name
40 PROCcount
50 PROCdelay
60 PROCcount
70 PROCdelay
```

```
 80 PROCdelay
 90 UNTIL FALSE
100 REM**********
110 DEF PROCdelay
120 TIME=0
130 REPEAT UNTIL TIME>100
140 ENDPROC
150 REM*********
160 DEF PROCprint_name
170 PRINT ''''"My name is Bob"
180 ENDPROC
190 REM***********
200 DEF PROCcount
210 FOR J= 1 TO 10
220 PRINT J
230 NEXT J
240 SOUND 1,-15,100,2
250 ENDPROC
```

As you can see, procedures are very much like functions, but are somewhat easier to use in many cases, and can be far more complex than functions.

In the sample program just given, the 'real' program — which is executed repeatedly while the program is running — lies within the master REPEAT/UNTIL loop (lines 20 to 90). The computer goes through this loop, calling up the procedures as it comes to them: PROCprint_name (note that the underline symbol can be used to link words in a procedure name), PROCcount, PROCdelay, PROCcount, PROCdelay, PROCdelay. It does not matter in which order the procedures are defined.

The statements between the DEF PROCname and ENDPROC are executed every time the computer comes across the line PROCname. One great advantage of procedures is that it leaves the 'real program' relatively short and each procedure can be written and debugged individually.

Here's the spirograph program again. Run it a few more times, then return to the book for a discussion on the use of the procedure PROCSPIRO(A,B,D).

```
10 REM ********************
20 ON ERROR GOTO 20
30 REPEAT
40 MODE 4
```

```
 50 VDU 29,640;512;
 60 A=RND(200)+250
 70 B=A-RND(A-50)
 80 C=A-RND(A-10)
 90 PROCSPIRO(A,B,C)
100 TIME=0
110 REPEAT UNTIL TIME>300
120 UNTIL FALSE
130 END
140 REM ********************
150 DEF PROCSPIRO(A,B,D)
160 E=D
170 F=(A-B)
180 G=0
190 H=PI*0.02
200 K=A/B
210 N=B/FNHCF(A,B)
220 L=100*N
230 MOVE F+E,0
240 FOR I=1 TO L
250 G=G+H
260 P=G*K
270 X=F*COS(G)+E*COS(P)
280 Y=F*SIN(G)-E*SIN(P)
290 DRAW X,Y
300 NEXT I
310 ENDPROC
320 REM ********************
330 DEF FNHCF(I,J)
340 LOCAL M
350 REPEAT
360 M=I MOD J
370 I=J
380 J=M
390 UNTIL M=0
400=I
410 REM ********************
```

PROCSPIRO as you know, draws a spirograph pattern. For the moment don't worry about the (A,B,D) at the end of line 15Ø. The main point to learn from it at the moment is that the procedure definition takes up much of the program. When you run it, an endless

variety of spirograph type patterns are displayed. If you get bored with any one pattern, just press ESCAPE to get a new one.

Now, about the (A,B,D). The procedure in this program needs three pieces of information: the radius of the outer circle of the spirograph; the radius of the inner circle; and the distance from the centre, of the inner circle of the pen. It would of course have been possible to set three variables to the values of these constrants before executing PROCSPIRO. However, the BBC Microcomputer has a simple mechanism for passing variable values to procedures. When you write a procedure, you choose the variables that will be passed to it, be they string variables or numeric variables. You place the names of these variables in brackets at the end of the procedure heading as in line 15Ø. When the procedure is called, the values contained in the brackets of the call will be copies into the variables you specified in the procedure heading. If you have a variable with the same name as one of the variables in the procedure heading, the variable outside the procedure and the variable within the procedure heading are treated as seprate variables. Changes in the value of the variables within the procedure do not affect the same variable outside the procedure. However, there are more variables used in this procedure than the two mentioned in the procedure heading. It would be good if they too were LOCAL to the procedure. All you do as we said earlier, is specify that certain variables be LOCAL.

In the spirograph program, a suitable line would be '155 LOCAL E,F,G,H,K,N,L,G,P,X,Y,I'. The variable names following the LOCAL statement are all those that appear on the left hand side of the assignment statements in the procedure.

Twenty One

In this program, you and the BBC Microcomputer take it in turns to throw a six-sided dice as many times as you like, trying to get a total of 21, or close to 21, without exceeding 21 ('busting'). This is a dice version of Blackjack, and the computer plays very well. You go first in every game, entering "1" to roll the dice again, or "2" to stand, that is, to stay with the total you have accumulated. There are five rounds to a game and the winner, of course, is the player who wins the most out of the five rounds.

Line 2Ø sets the mode, and line 3Ø sends action to a procedure called 'initialise'. This short procedure (lines 7Ø to 1ØØ) sets the two variables which hold the scores — SI and SM — to zero. From there, line 4Ø sends the computer's attention to the procedure called game_count. The 'game_count' procedure (lines 11Ø to 15Ø) set up a loop which calls the procedure 'game' five times. As you've probably noticed earlier in this book, the BBC Microcomputer supports loop names which are whole words (in this case the word, as you can see, is 'count'). The procedure, 'game', which is called from within the procedure game_count, runs from lines 16Ø to 33Ø.

```
 10  REM TWENTY ONE
 20  MODE 7
 30 PROCinitialise
 40 PROCgame_count
 50 PROCfinale
 60  END
 70 DEF PROCinitialise
 80 SI=0
 90 SM=0
100 ENDPROC
110 DEF PROCgame_count
120 FOR count=1 TO 5
130   PROCgame
140   NEXT count
150 ENDPROC
160 DEF PROCgame
170 E=0:F=0
180 PROCupdate
190 T=TIME:REPEAT UNTIL TIME-T>50
200   IF E>0 PRINTCHR$(129),;"TOTAL ";E
210 IF E=0 PRINTCHR$(131);"1 TO ROLL,
                    2 TO STAND"''
```

```
220 A=GET
230 IFA=49 E=E+RND(6): GOTO190
240 IF E>21 PRINT CHR$(129);"YOU'VE
     BUSTED!!": PROCend_of_game:ENDPROC
250 PRINT'CHR$(133);"OK, YOU STAND ON ";E''
260 G=RND(6)
270 F=F+G
280 T=TIME:REPEAT UNTIL TIME-T>150
290 PRINT CHR$(128+G);"I rolled a ";
         G;", so my total is ";F
300  IF F=E PROCend_of_game:ENDPROC
310  IF F>18 OR F>E  PROCend_of_game:
                         ENDPROC
320 IF F<17 OR F<E AND E<22 THEN 260
330 ENDPROC
340 DEF PROCend_of_game
350 IF E=F AND E<22 PRINT'CHR$(129);
     "DEAD HEAT":SI=SI+1:SM=SM+1
360  IF E<>F AND (F>E AND F<22) OR E>21
     PRINT'CHR$(129);"I WIN":SI=SI+1
370  IF E<>F AND F>21 OR(E>F AND E<22)
     PRINT'CHR$(133);"YOU WIN":SM=SM+1
380 ENDPROC
390 DEF PROCfinale
400  PRINT''CHR$(130);"FINAL SCORES:"
410   PRINT'CHR$(129);"YOU: ";SM,"ME: ";SI'''
420 T=TIME:REPEAT UNTIL TIME-T>150
430 IF SI>SM PRINT"I ";
440 IF SM>SI PRINT"YOU ";
450 IFSI=SM PRINT"WE BOTH ";
460 PRINT CHR$(129);"WIN!"
470 ENDPROC
480 DEF PROCupdate
490 PRINT'CHR$(132);"££££££££££££££££££
                ££££££££££££££££££££"'
500 T=TIME:REPEAT UNTIL TIME-T>150
510 PRINT'CHR$(130);"ROUND ";count
520 PRINT'CHR$(134);"YOU ";SM,"ME ";SI''
530 ENDPROC
```

The variables E and F store the human and computer scores respectively within a particular round of the game. The procedure 'update' (lines 480 to 530) is called to draw a coloured line across the screen, and report how the two protaganists are going. Line 190 is a delay loop, setting the variable T to the value held by the computer's internal clock, and then passing through a REPEAT/UNTIL loop until half a second (TIME increments by one every hundredth of a second) has passed. Line 220 waits for the player's input, and line 230 rolls the dice, and adds its value to the score (line 200) if the player has decided to keep rolling.

If not, the computer checks (line 240) to see if the player has exceeded 21 (and, if so, goes to the 'end_of_game' procedure). If the player has not busted, the computer acknowledges the total the human has decided to stick with (line 250) and proceeds to roll the dice itself. A short delay (line 280) comes before the computer announces the value it has rolled, and its total. Lines 300 to 320 check the computer's score, and compare it with the player's score, and from this comparison decides whether or not to roll again.

The procedure 'end_of_game' (lines 340 to 380) look at the two final totals, and decides which player has won. After five rounds have been played (counted, you'll recall, by the loop in the 'game_count' procedure), action goes to the 'finale' procedure (lines 390 to 470) to find the winner of the five games.

SEVEN-UP

Our next program — SEVEN-UP — shows the use of procedures quite clearly. The game is played within the master REPEAT/UNTIL loop from lines 60 to 110.

SEVEN-UP is somewhat like checkers or draughts, except that it is played on a 7 X 7 board, hence the name. The pieces move as draughts pieces — diagonally one square, jumping over an opponent for capture into an empty square beyond the opponent. The main differences from draughts, apart from the size of the board, are that pieces may move forward and backward at will, there are no kings (every piece can move as it is a king) and there are no multiple jumps.

The computer is the X's moving down the screen, as you can see from the sample run, and you are the O's. You move by entering the number of the square from which you are moving — entering the

number along the left-hand edge first, then the number across the top, then pressing RETURN; and the two number which refer to the square to which you are moving.

The computer keeps track of the score, tells you the move it makes, and terminates the game as soon as one player manages to capture five of the opponent's pieces.

The way procedures have been used in this program shows how well they work for a long program. Each procedure was written and debugged separately. In fact, the master REPEAT/UNTIL loop was worked out before any of the procedures were more than a name.

```
 10 REM Seven-Up
 20 REM showing use of procedures
 30 REM (C) Hartnell, 1982
 40 MODE7
 50 PROCinitialise
 60 REPEAT
 70 PROCcomputer_move
 80 PROCprint_board
 90 PROChuman_move
100 PROCprint_board
110 UNTIL FALSE
120 REM***********
130 DEF PROChuman_move
140 PRINT'CHR$(128+RND(5));"From";:INPUT M
150 PRINT CHR$(11);CHR$(128+RND(5));" ";M;" to";
160 INPUT N:PRINT CHR$(11);"                "
170 H(N)=79
180 IF ABS(M-N)=22 OR ABS(M-N)=18 H((M+N)/2)=46:
       ME=ME+1
190 H(M)=46
200 ENDPROC
210 REM***********
220 DEF PROCcomputer_move
230 A=76
240 IF H(A)<>88 THEN 310
250 B=1
260 IF A<28 AND B<3 THEN 300
270 IF A>60 AND B>2 THEN 310
280 Q=2*Z(B)
290 IF H(A+Z(B))=79 AND H(A+Q)=46 THEN 330
300 IF B<4 B=B+1:GOTO260
```

```
310 IF A>12 A=A-1:GOTO240
320 GOTO 350
330 H(A+Z(B))=46:H(A)=46:H(A+Q)=88:Y=A+Q
340 X=A:IT=IT+1:GOTO 460
350 Y=0
360 Y=Y+1
370 K=RND(66)+11
380 IF H(K)<>88 AND Y<100 THEN 360
390 IF H(K)<>88 THEN 470
400 T=1
410 IF H(K+Z(T))=46 THEN 450
420 IF T<4 T=T+1:GOTO410
430 IF Y<70 THEN 350
440 GOTO 470
450 H(K+Z(T))=88:H(K)=46:X=K:Y=K+Z(T)
460 ENDPROC
470 PRINT''"I concede..."
480 END
490 REM*************
500 DEFPROCprint_board
510 PRINTTAB(0,4);CHR$(128+RND(5));
    "SCORES:   You-";ME;"   Me-";IT
520 PRINTCHR$(130);TAB(11);"1234567"
530 PRINTCHR$(131);TAB(7);"***************"
540 FOR J=70 TO 10 STEP -10
550 SOUND RND(3),-15,RND(250),RND(3)
560 A=H(J+1):B=H(J+2):C=H(J+3):D=H(J+4)
570 E=H(J+5):F=H(J+6):G=H(J+7)
580 PRINTCHR$(130);TAB(7);J/10;CHR$(129)
  "*";CHR$(128+RND(5));CHR$(A);CHR$(B)
   ;CHR$(C);CHR$(D);CHR$(E);CHR$(F);
  CHR$(G);CHR$(129);"*";CHR$(130);J/10
590 NEXT J
600 PRINTCHR$(131);TAB(7);"***************"
610 PRINTCHR$(130);TAB(11);"1234567"
620 PRINTTAB(0,18);CHR$(128+RND(5));
     "I moved from ";X;" to ";Y
630 IF IT=5 OR ME=5 THEN 650
640 ENDPROC
650 IF IT=5 PRINT "I win"
660 IF ME=5 PRINT "You win"
670 END
```

```
680 REM*************
690 DEF PROCinitialise
700 DIM H(87),Z(4)
710 VDU23;8202;0;0;0
720 IT=0
730 ME=0
740 FOR A=1 TO 87
750 IF A>77 OR A=70 OR A=60 OR A=68
    OR A=69 OR A=50 OR A=59 OR A=58 OR
    A=40 OR A=49 OR A=48 THEN 800
760 IF A=30 OR A=38 OR A=39 OR A=20
    OR A=28 OR A=29 OR A=18 OR A=19
    OR A<11 THEN 800
770 H(A)=ASC(".")
780 IF A=72 OR A=74 OR A=76 OR A=61
OR A=63 OR A=65 OR A=67 THEN H(A)=ASC("X")
790 IF A=21 OR A=23 OR A=25 OR A=27
OR A=12 OR A=14 OR A=16 THEN H(A)=ASC("O")
800 NEXT A
810 FOR T=1TO4:READ Z(T):NEXT
820 PRINT'''''
830 ENDPROC
840 DATA -11,-9,11,9
```

```
SCORES:   You-0   Me-0
           1234567
       ***************
         7*.X.X.X.*7
         6*X...X.X*6
         5*.X.....*5
         4*.......*4
         3*.......*3
         2*O.O.O.O*2
         1*.O.O.O.*1
       ***************
           1234567
I moved from 63 to 52
From?25
 25 to?36
```

```
SCORES:    You-0    Me-0
        1234567
    ***************
      7*.X.X.X.*7
      6*X...X.X*6
      5*.X.....*5
      4*.......*4
      3*.....O.*3
      2*O.O...O*2
      1*.O.O.O.*1
    ***************
        1234567
I moved from 63 to 52


SCORES:    You-0    Me-0
        1234567
    ***************
      7*.X.X.X.*7
      6*X.....X*6
      5*.X.X...*5
      4*.......*4
      3*.....O.*3
      2*O.O...O*2
      1*.O.O.O.*1
    ***************
        1234567
I moved from 65 to 54
From?36
 36 to?45
```

```
SCORES:    You-0    Me-0
        1234567
    ***************
      7*.X.X.X.*7
      6*X.....X*6
      5*.X.X...*5
      4*....O..*4
      3*.......*3
      2*O.O...O*2
      1*.O.O.O.*1
    ***************
        1234567
I moved from 65 to 54


SCORES:    You-0    Me-1
        1234567
    ***************
      7*.X.X.X.*7
      6*X.....X*6
      5*.X.....*5
      4*.......*4
      3*.....X.*3
      2*O.O...O*2
      1*.O.O.O.*1
    ***************
        1234567
I moved from 54 to 36
```

We'll be looking again at writing games programs shortly, but before that there are two important topics to discuss, how to make the most of the user-definable function keys, and file handling.

User-definable Function Keys

When you are developing a program, you may have phrases or expressions which crop up many times. For example, the sequence 'TIME=Ø:REPEAT UNTIL TIME=1Ø' appears many times in my programs, to give a one tenth of a second delay. Your computer gives you a way of calling up these often-used expressions, at the touch of a key, by defining them to perform a specific function.

You can define any of the ten red keys at the top of the keyboard. Some, or all, of the keys can be programmed, but the total number of letters being stored under the keys must not be greater than 255.

The keys are, of course, numbered from Ø to 9. To define a key, enter something like: *KEY 4 "HELLO", where the text inside the quote marks is the text you want to have stored under the key. My first example is programmed: *KEY 2"TIME=Ø:REPEAT UNTIL TIME=1Ø". This sample program shows some more possible key definitions:

```
 10 REM ** User definable key demo **
 20 *KEY 0 "FOR T=32 TO 126:VDU T:NEXT T|M"
 30 *KEY 1 "OLD|MRUN|M"
 40 *KEY 2 "MODE 5|M VDU 19,3,4,0,0,0|M"
 50 *KEY 3 "MODE 7|M LIST|M"
 60 *KEY 4 "|!|A"
 70 *KEY 5 "|!|B"
 80 *KEY 6 "|!|C"
 90 *KEY 7 "|!|D"
100 *KEY 8 "|!|E"
110 *KEY 9 "|!|F"
120 *KEY 10 "OLD|M RUN|M"
```

To use this program, run it, then press any of the red keys, and note the result. The functions of some of them may not be immediately obvious. The VDU command, in this case, prints out the character with the ASCII code 'T'. The net effect is to print out all the characters with codes between 32 and 126, which is the entire ASCII set. At the end of the line there is curious line character, followed by an M. The line is obtained with 'shift reverse slash'. In mode 7, it appears as a 'double bar', like an equals sign on its end. The line means that the next character is a 'control character'. Control characters are explained more fully in the VDU drivers chapter. They are the characters

generated when you use the control key with the other keys. You may have used control-G, or control-O and N to turn page mode on and off. In fact, the 'return' key generates the same code as pressing control-M, as you can easily verify by entering control-M a few times. So the effect of the two end characters is to generate a single control character, control-M, or 'return'. This means that when you press the key, you do not need to press 'return' afterwards, to start the printing.

More often than not, you will find no use for any of the other control characters in the function keys, but control-M can often be used to great effect. You should be able to see that the next line, line 3Ø, defines a key which types OLD and then RUN, without any interference from you. This is the sort of key which can be very useful, if you are prone to accidently press 'BREAK'.

The next key places you in mode 5, and makes colour 3 appear as blue, rather than white by using the VDU 19... command.

The key in line 5Ø is also very useful. Whenever pressed, it takes you back into mode 7, and automatically lists the program.

For the moment, I'll miss out keys 4 to 9.

You are probably a little puzzled at the *KEY1Ø statement in line 12Ø. This statement is redefining the 'BREAK' key. Whenever the BREAK key is pressed after the execution of this line, the computer will be OLDed and RUNed. If you press the BREAK key twice in succession you will cancel the information under the BREAK key, and all the others.

To return to keys 4 to 9. You may have read the section about teletext graphics by now, and so you should be conversant with the idea of the codes from 128 onwards giving colour in mode 7. The bar, followed by the exclamation mark allow you to put these codes into the function keys. The bar with the exclamation mark adds 128 to the next control code so given that 'bar'A'' gives code 1, the effect of line 6Ø is to define a key which hold the code 129, or the code for red alphanumerics. Thus, just press this key, and anything else you type on the same line will appear red. In the same way, line 11Ø gives code of 128+6, which is 134, the code for light blue. By increasing the letters, you can use the graphics codes as well. It is also possible to mix normal text and the colour control codes in a single key. If you use the colour control codes outside quote marks in your programs, the computer will turn them into words like 'EOR', so beware. The codes will also do funny things to printers, so use them with care.

There are many uses for function keys. Often a program will ask a question which is usually answered in the same way — like 'Enter your name' — so it is wise to store the answer under a function key, and then most of the time you only need press the key, instead of typing

out the whole name. Apart from 'standard responses', you will probably find they are most useful when you are writing programs.

File handling

File handling is reading and writing information to and from some medium, such as cassettes. This discussion assumes you are using cassettes. First of all, enter this program, then I'll discuss it, then we'll run it.

```
 10 REM ** FILE HANDLING **
 20 *TAPE3
 30 FILE=OPENOUT("EXAMPLE")
 40 PRINT#FILE,"This is a series of
                    example pieces"
 50 PRINT#FILE,"Now is the time for
   all good men to come to the aid
   of the party."
 60 PRINT#FILE,"The quick brown rat
       jumped over the tiny elephant"
 70 CLOSE#0
 80 PRINT "Now rewind the tape, and
   press 'play', then the space bar"
 90 D=GET
100 PRINT "OK"
110 FILE=OPENIN("EXAMPLE")
120 INPUT#FILE,A$,B$,C$
130 CLOSE#FILE
140 PRINT A$'B$'C$
150 END
```

Line 2Ø — Lowers the speed at which information is set to the cassette recorder. You can use normal speed transfers if you replace this line with *TAPE. I use slow tape access for file handling small quantities of data, for reliability in data recall. Line 3Ø — Tells the computer that you want to send some information to a file called "EXAMPLE". The word in quotes is called the file name, and is equivalent to the name you give programs when you save them on cassette. You could use something like A$ instead, if the name of the file had not been decided at the time of writing the program. This word will appear on the screen when you

do a *CAT of the things of the tape you used for data storage. Having opened the file, and remembered its name, the computer assigns a unique number to it, called its channel number. This number has been copied into the variable 'FILE' in this example. The channel number is used for all future transactions between the computer and the file. That is the file name is not used again. When this line is executed, a message telling you to press PLAY and RECORD will be output by the computer.

The use of the name 'FILE' is completely arbitrary — it usually makes more sense to make the channel number variable the same as the file name, to prevent confusion when you are dealing with a large number of files.

Line 4Ø — this outputs to the file with the channel number 'FILE' the text in quotes.
Line 5Ø — Outputs more text in the same way.
Line 6Ø — and a little bit more.
Line 7Ø — Tells the computer that you have finished with all your file handling for the moment, and so it closes all the files. You can close specific files by quoting the channel number, as in line 13Ø. The process of closing a file is akin to tying up a parcel. The OPENOUT statement makes the box ready for the parcel, the PRINT hash statements put things in the box and the CLOSE statement ties up the parcel, and sends it off.

Line 11Ø — Opens the same file in the same way as in line 3Ø, except that it tells the computer that you are going to INput information from the file, and not OUTput information to it. The channel number is used in exactly the same way as before.

Line 12Ø — Inputs, from the file with channel number 'FILE', three string variables, A$, B$ and C$. These three strings will hold the data output to the file above.
Line 13Ø — Closes the file, as in line 7Ø. A closed file — when it has been used for input — is obviously not sealed, but it still tells the computer that it can stop worrying about the file.
Line 14Ø — Outputs to the user all the information which was read from the file.

The first thing you'll see when you run the program is a message telling you to start your recorder. After a pause, while the information is sent to the recorder, a message telling you to rewind the cassette and to press the space bar is printed. When you do this, there will be another short pause while the information is read in from the cassette recorder, then the information will be printed out.

There are several useful alterations you can make to this program. The first is to make the data stored in numeric, rather than string form. To do this,change lines 4Ø to 6Ø to something like 'PRINT FILE, 3.14159'.

Now the data will be written as numeric information. The second part of the program is still expecting string information, so you'll need to alter lines 12Ø and 14Ø to use the numeric variables A, B and C.

To show the commands described in action, here is a simple stock control program. I have assumed that the user is the owner of a small pet shop, and the program keeps track of the amount of parrot food(!) in stock at anyone time. It is also capable of telling when to re-order.

```
 10 REM ***************************
 20 REM *** Parrot shop program ***
 30 REM Copyright (C) Jeremy Ruston
 40 REM ***************************
 50 ON ERROR IF ERR<>17 THEN REPORT:PRINT " at line ";ERL:END
 60 *TAPE3
 70 UN=0
 80 REPEAT
 90 MODE 7
100 PRINT ''CHR$(129);"Select one of the following :"
110 PRINT ''CHR$(132);"1.Enter new stock"
120 PRINT 'CHR$(132);"2.Deduct stock"
130 PRINT 'CHR$(132);"3.Re-order"
140 PRINT 'CHR$(132);"4.Save to cassette"
150 PRINT 'CHR$(132);"5.Load from cassette"
160 PRINT 'CHR$(132);"6.Print stock"
170 PRINT 'CHR$(132);"7.Computer dating"
180 PRINT ''CHR$(131);"---> ";
190 REPEAT
200 A$=GET$
210 UNTIL A$>="1" AND A$<="7"
220 CLS
230 IF A$="1" THEN PROCNEW_STOCK
240 IF A$="2" THEN PROCDEDUCT
250 IF A$="3" THEN PROCRE_ORDER
260 IF A$="4" THEN PROCSAVE
270 IF A$="5" THEN PROCLOAD
280 IF A$="6" THEN PROCPRINT
290 IF A$="7" THEN PROCDATING
300 UNTIL FALSE
310 DEF PROCNEW_STOCK
320 PRINT ''CHR$(131);"The present stock level is ";
             UN'CHR$(131);"units of feed."
330 PRINT 'CHR$(131);"What should it be ?";
340 INPUT ""UN$
350 UN=VAL(UN$)
360 ENDPROC
370 DEF PROCDEDUCT
380 PRINT ''CHR$(131);"The present stock level
            is ";UN'CHR$(131);"units of feed."
390  PRINT 'CHR$(131);"Enter the deduction ";
400 INPUT ""UN$
410 UN=UN-VAL(UN$)
420 ENDPROC
430 DEF PROCRE_ORDER
440 IF UN>100 THEN PRINT ''CHR$(131);"No need
   to re-order." ELSE PRINT ''CHR$(131);"Re-order now"
450 D=GET
460 ENDPROC
470 DEF PROCSAVE
```

```
480 A=OPENOUT("FILE")
490 PRINT#A,UN
500 CLOSE#A
510 ENDPROC
520 DEF PROCLOAD
530 PRINT "Play tape"
540 A=OPENIN("FILE")
550 INPUT#A,UN
560 CLOSE#A
570 ENDPROC
580 DEF PROCPRINT
590 PRINT ''CHR$(131);"The present stock level
     is ";UN'CHR$(131);"units of feed."
600 A=GET
610 ENDPROC
620 DEF PROCDATING
630 PRINT '''''CHR$(134);"Come off it..."
640 A=GET
650 ENDPROC
```

The program operates as follows:

Line Comments...

50 Sets the error handler to such a way that if 'ESCAPE' is pressed, the program returns to the main menu
60 Selects the low cassette speed
70 Starts the stock level at zero
80 Starts the main program REPEAT loop
90 Places the machine in mode 7
100 Starts printing the menu
190 Starts the REPEAT loop which continues until a valid response is found
200 Gets a character from the user
210 Continues the loop until a valid response is entered
220 Clears the screen, ready for each of the routines
230 Starts the calling section
300 Returns to the menu after the routine has executed
310 PROCNEW_STOCK
320 Prints the current stock level...
330 then requests the new level
340 Gets the new value
350 Resets the stock level variable
370 PROCEDEDUCT
380 Prints the present stock level
390 Requests the deduction
400 Gets the deduction
410 Updates the stock level variable
430 PROCRE_ORDER
440 Decides whether re-ordering is needed, and prints an appropriate message
450 Gets a key press before...
460 returning

470 PROCSAVE..
480 OPENS the file for output
490 Outputs the present stock level
500 Closes the file
520 PROCLOAD
530 Asks the user to press play
540 Opens the file for input
550 Gets the stock value from tape
560 Closes the file
580 PROCPRINT
590 Prints the present stock level
600 Gets a keypress
610 Ends.

This program is somewhat trivial, but it shcws important features of business software, including the menu, and is virtually crashproof.

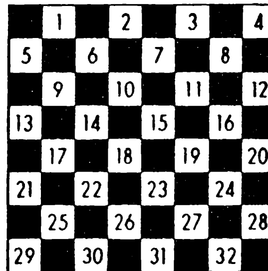
Note that, in file handling, unlike DATA statements, you cannot load a numeric value from tape into a string variable.

DRAUGHTS AND OTHER GAMES

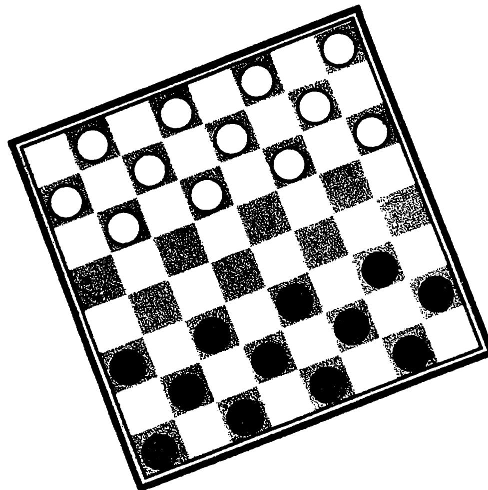
No matter why you bought your BBC Microcomputer, it is likely that you spend at least some time playing games with it. Writing, developing and adapting games programs is probably the least painful way which exists to improve programming skills. The first game we will look at in this section is draughts. One of the reasons for including it is so that I can explain a method of numbering boards for board games which make it easy for a computer to manipulate. A similar board-numbering system can be used as the core of a chess program, noughts and crosses, Nine Mens' Morris or whatever.

Books on how to play draughts make use of a numbering convention shown in the illustration. The white squares are counted off from one to 32. But this numbering method creates a problem when we try to define a move in terms of the mathematical relationship between two diagonally adjacent squares. The difference between the squares can be three, or it can be four in one direction; and in the other direction, the difference in the number on the square, in terms of a diagonal move, can be five or four. This creates problems for a computer. As well, there are no 'spare' numbers to indicate where the board ends.

In the 1960s, A L Samuels wrote an article for Scientific American (see Strachey, Christopher, "Systems Analysis and Programming," in 'Readings from Scientific American', W H Freeman and Co., San Francisco, 1971) in which he devised a clever numbering system in which the difference between the diagonally adjacent squares is always four and five (or minus four and minus five). His system also allowed for numbers to be given to squares which were 'off the board'.



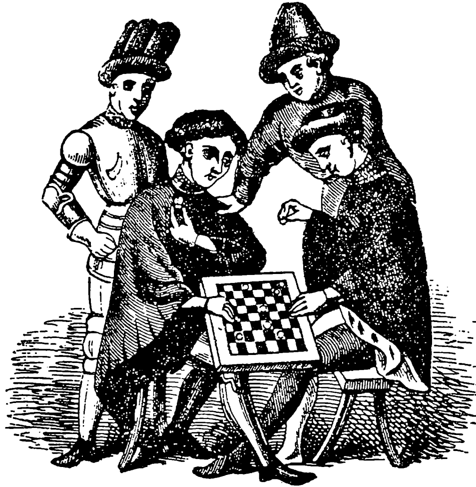
I've changed his numbering system a little to make it more convenient for a computer. In my system, the diffference between squares is always six or seven (or minus six and minus seven). My system, very simply, sets up an array and allots certain elements of the array to squares on the board. All other squares are understood by the computer to be off the board. The computer assigns a value of 99 to any square which is off the board, zero to any empty square, one to a computer's ordinary piece (two to a computer king) and minus one to a player's ordinary piece (with minus two for a player's king). This may sound a little complex, but it should become clear shortly.



Here is my numbered board. You can see that if you move from the top left hand corner (69) to the square diagonally below it (63), the difference between the two squares is minus six. Now, choose any other square on the board from which you can move down and to the left, and you'll see there is a difference of minus six between the square you started on and the square on which you finished. This sort of predictable result is relatively easy for a computer to handle.

	72		71		70		69
66		65		64		63	
	59		58		57		56
53		52		51		50	
	46		45		44		43
40		39		38		37	
	33		32		31		30
27		26		25		24	

Move in the other direction, that is downwards and to the right, and you'll see the difference between the two squares is minus seven. In the first version of draughts in this book — KIDDIE CHECKERS — you'll need this numbered board as a reference, to know how to enter your moves. The first version of the game has no strategy to speak of (except to make a capture when it can, and if no capture is available, make a legal move), but it is still quite a fascinating program to run. There are no multiple jumps, although there are kings. Kings are made automatically.



To briefly explain the program, I'll go through the main sections of it. The subroutine starting at line 780 sets up the starting conditions for the game, alloting values of 99, 0, 1 and 2 for the elements of the array A. The board is printed from the subroutine starting at line 500. Lines 510 to 570 change the number 0, 1, 2 and so on into the codes of the characters which will be used to print the board on the screen. Zero, for example, is changed to 32, the ASCII code for a space. Lines 600 to 690 print out the board, using a neat routine developed by Toni Baker.

The loop from lines 50 to 70 checks to see if any player's pieces should be crowned. The routine from there to line 170 checks to see if a capture can be made, and if it can, effects the capture with lines 200 to 220. If not, the routine from line 250 to line 350 chooses squares at random, and, if one of its own pieces is on that square, the computer checks to see if it can make a move. If it can, it does. If it has not got one of its pieces, or there is no possible move, and the computer has chosen less that 1000 random numbers, it goes back to look again. If, in 1000 random selections, the computer has not found a move it can make, it concedes defeat (line 360).

After printing the board again (with the subroutine called from line 400) the computer accepts the player's move, which is entered (line 420) as two numbers. The first is the number of the square the player is moving from, and the second is the square he or she is moving to. Line 460 'erases' a computer's piece if the player has captured it.

```
 10 REM *Kiddie Checkers*
 20 REM (C) Hartnell 1982
 30 GOSUB 780
 40 Q=0:Z=24
 50 FOR G=69 TO 72
 60 IF A(G)=-1 A(G)=-2
 70 NEXT G
 80 IF A(Z)=99 OR A(Z)<1 THEN 170
 90 IF Z<28 AND A(Z)=1 A(Z)=2
100 Y=1
110 IF A(Z+X(Y))<0 AND A(Z+2*(X(Y)))=0 Q=X(Y)
120 IF A(Z)=2 AND A(Z-X(Y))<0 AND
          A(Z-2*X(Y))=09 Q=-X(Y)
130 IF Q<>0 AND Z+2*Q>23 THEN 200
140 Q=0
150 IF Y=2 GOTO 170
160 Y=2:GOTO110
170 Z=Z+1
180 IF Z<73 GOTO 80
190 IF Q=0 THEN 250
200 A(Z+Q)=0
210 A(Z+2*Q)=A(Z)
220 A(Z)=0
230 GOSUB 500
240 GOTO410
250 U=0
260 Z=23+RND(49)
270 U=U+1
280 IF A(Z)=1 OR A(Z)=2 THEN 300
290 GOTO260
300 Y=1
310 IF A(Z+X(Y))=0 Q=X(Y)
320 IF A(Z)=2 AND A(Z-X(Y))=0 Q=-X(Y)
330 IF Q<>0 THEN 380
340 IF Y=1 Y=2:GOTO310
350 IF U<1000 THEN 260
```

```
360 PRINT "You win"
370 END
380 A(Z+Q)=A(Z)
390 A(Z)=0
400 GOSUB500
410 SOUND 1,-15,100,3
420 INPUT D,E
430 PRINT CHR$(11);"          "
440 A(E)=A(D)
450 A(D)=0
460 IF ABS(D-E)>7 A(D+(INT(E-D)/2))=0
470 GOSUB 500
480 GOTO40
490 IF Q<>0 THEN 290
500 PRINT'''
510 FOR M=24 TO 72
520 IF A(M)=1 A(M)=79
530 IF A(M)=2 A(M)=33
540 IF A(M)=0 A(M)=32
550 IF A(M)=-1 A(M)=88
560 IF A(M)=-2 A(M)=36
570 NEXT M
580 PRINT CHR$(30)''' C$;Z,Z+Q
590 PRINT
600 FOR K=0 TO 3
610 FOR J=0 TO 3
620 PRINT CHR$(255);CHR$(A(72-J-13*K));
630 NEXT J
640 PRINT
650 FOR J=0 TO 3
660 PRINT CHR$(A(66-J-13*K));CHR$(255);
670 NEXTJ
680 PRINT
690 NEXT K
700 FOR M=24 TO 72
710 IF A(M)=79 A(M)=1
720 IF A(M)=33 A(M)=2
730 IF A(M)=32 A(M)=0
740 IF A(M)=88 A(M)=-1
750 IF A(M)=36 A(M)=-2
760 NEXT M
```

```
770 RETURN
780 DIMA(99),X(2):Q=0
790 X(1)=-6:X(2)=-7
800 FOR Z=1 TO 99
810 A(Z)=99
820 IF Z<73 AND Z>55 AND NOT(Z=67 OR Z=68
          OR Z=60 OR Z=61 OR Z=62) A(Z)=1
830 IF Z<54 AND Z>42 AND NOT (Z=47 OR Z=48
          OR Z=49) A(Z)=0
840 IF Z<41 AND Z>23 AND NOT(Z=34 OR Z=35
       OR Z=36 OR Z=28 OR Z=29) A(Z)=-1
850 NEXT Z
860 C$="My move: "
870 CLS
880 GOSUB 500
890 RETURN
```

Next we have a 'proper' draughts game. That is, the next program is a development of the proceeding one. In this version, there are multiple jumps. The computer will make its multiple jumps automaticaly. After you make a capture, you'll be asked "Can you jump again?" Just press RETURN if you cannot, and the game will continue. If you can jump again, press any key, *before* you press return and you'll be offered another move. You enter your move by typing in the letter at the bottom of the square which is on the line on which, your piece is sitting, then the number along the right hand side, then a comma, then the letter and number co-ordinate of the square you're moving to. There are clear player prompts within the program, so you should have little trouble in playing it. You'll find this version of the game puts up a far stronger defence than the other program, and is most reluctant to move into danger. There are three ways the game can end. The computer will either concede defeat, even if there are still possible moves, if it judges the situation is hopeless; or it will claim victory on capturing all you pieces, or give you the game when you capture all of its pieces. although you should not have too much trouble beating this program (which plays like a better-than-average beginner), it is still a most entertaining program to run, and a study of how it decides which moves to make should stand you in good stead when you write your own games of strategy.

```
10 REM***********************
20 REM       *Draughts*
30 REM   (C) Hartnell, 1982
40 REM***********************
50 PROCinitialise:MODE7
```

```
 60 REM **Master REPEAT/UNTIL**
 70 REPEAT:Q=0:Z=24
 80 REM **Create human kings**
 90 FOR G=69 TO 72:IF A(G)=-1 A(G)=-2
100 NEXT G
110 PROCprint_board
120 REM **Computer looks for capture**
130 IF A(Z)=99 OR A(Z)<1 THEN 220
140 IF Z<28 AND A(Z)=1 A(Z)=2
150 Y=1
160 IF A(Z+X(Y))<0 AND A(Z+2*(X(Y)))=0 Q=X(Y)
170 IF A(Z)=2 AND A(Z-X(Y))<0 AND
                         A(Z-2*X(Y))=0 THEN Q=-X(Y)
180 IF Q<>0 AND Z+2*Q>23 THEN 250
190 Q=0
200 IF Y=2 THEN 220
210 Y=2:GOTO160
220 Z=Z+1
230 IF Z<73 THEN 130
240 IF Q=0 THEN 340
250 A(Z+Q)=0:A(Z+2*Q)=A(Z):A(Z)=0
260 Z=Z+2*Q:COM=COM+1:PROCprint_board
270 Q=0:Y=1
280 IF A(Z+X(Y))<0 AND A(Z+2*(X(Y)))=0 Q=X(Y)
290 IF A(Z)=2 AND A(Z-X(Y))<0
                 AND A(Z-2*X(Y))=0 THEN Q=-X(Y)
300 IF Q<>0 AND Z+2*Q>23 THEN 250
310 IF Y=1 Y=2:GOTO 280
320 GOTO 480
330 REM **Computer looks for non-capture
340 U=0
350 Z=23+RND(49):U=U+1
360 IF A(Z)=1 OR A(Z)=2 THEN 380
370 GOTO 350
380 Y=1
390 IF A(Z+X(Y))=0 AND (A(Z+2*X(Y))>-1
  AND A(Z+2*X(Y)+1)>-1 AND A(Z+2*X(Y)-1)>
  -1 OR U>600) Q=X(Y)
 400 IF A(Z)=2 AND A(Z-X(Y))=0 AND
(A(Z-2*X(Y))>-1 AND A(Z-2*X(Y)+1)>-1 AND
 A(Z-2*X(Y)-1)>-1 OR U>600) Q=-X(Y)
 410 IF Q<>0 THEN 450
```

```
420 IF Y=1 Y=2:GOTO 390
430 IF U<1000 THEN 350
440 PRINT "I concede the game!":END
450 A(Z+Q)=A(Z):A(Z)=0
470 PROCprint_board
480 PRINT'''TAB(5);CHR$(128+RND(5));
                    "Enter your move"
490 PRINT TAB(4);CHR$(128+RND(5));
               "as A9, B8 separated"
500 PRINTTAB(7);CHR$(128+RND(5));
                         "by a comma"
510 SOUND 1,-15,100,3
520 INPUT A$,B$:FOR W=1 TO 2:Z=0
530 IF W=1 C$=A$
540 IF W=2 C$=B$
550 Z=-24*(C$="G9")-25*(C$="E9")-
      26*(C$="C9")-27*(C$="A9")-30*
      (C$="H8")-31*(C$ ="F8")-32*(C$=
      "D8")-33*(C$="B8")-37*(C$="G7")
   -38*(C$="E7")-39*(C$="C7")-40*(C$="
A7")-43*(C$="H6")-44*(C$="F6")-45*(C$=
     "D6")-46*(C$="B6")-50*(C$="G4")
560 IF Z<>0 THEN 580
570 Z=-51*(C$="E4")-52*(C$="C4")-53
*(C$="A4")-56*(C$="H3")-57*(C$="F3")
-58*(C$ ="D3")-59*(C$="B3")-63*(C$=
"G2")-64*(C$="E2")-65*(C$="C2")-66*
 (C$="A2")-69*(C$=" H1")-70*(C$="F1")-
   71*(C$="D1")-72*(C$="B1")
580 IF W=1 D=Z
590 IF W=2 E=Z
600 NEXT W
610 PRINT CHR$(11);CHR$(11);
          CHR$(11);CHR$(11);CHR$(11)
620 FOR T=1TO 4
630 PRINT "                              "
640 NEXT
650 A(E)=A(D):A(D)=0
660 IF ABS(D-E)>7 A(D+(INT(E-D)/2))=0:
                           HUM=HUM+1
```

```
670 PROCprint_board
680 IF ABS(D-E)>7 PRINT''':INPUT
     "Can you jump again?"U$:
 PRINT CHR$(11);"                         ":
      IF U$<>"" THEN 520
690 UNTIL HUM=12 OR COM=12
700 IF HUM=12 PRINT"You win, human!":END
710 IF COM=12 PRINT"I have beaten you!":END
720 REM **Print board procedure**
730 DEF PROCprint_board
740 FOR M=24 TO 72
750 A(M)=-79*(A(M)=1)-33*(A(M)=2)-32*
(A(M)=0)-88*(A(M)=-1)-36*(A(M)=-2)-99*
          (A(M)=99)
760 NEXTM
770 PRINT CHR$(30)'''"
            "
780 IF Z<>Z+Q PRINT CHR$(11);CHR$
   (128+RND(5));"My score is ";COM;
    ", and yours is ";HUM
790 T=-2:FOR K=0 TO 3:FOR J=0 TO 3
800 PRINT CHR$(255);CHR$(A(72-J-13*K));
810 NEXTJ:T=T+1
820 PRINT;CHR$(128+RND(5));INT((J+K)/2)+T
830 FOR J=0 TO 3
840 PRINT CHR$(A(66-J-13*K));CHR$(255);
850 NEXTJ:T=T+1
860 PRINT;CHR$(128+RND(5));INT((J+K)/2)+T
870 NEXT K
880 PRINT "ABCDEFGH"
890 FOR M=24 TO 72
900 A(M)=-(A(M)=79)-2*(A(M)=33)-0*(A(M)
  =32)+(A(M)=88)+2*(A(M)=36)-99*(A(M)=99)
910 NEXTM
920 ENDPROC
930 REM **Initialisation procedure**
940 DEF PROCinitialise
950 CLS:DIM A(99),X(2):X(1)=-6:X(2)=-7
960 FOR Z=1 TO 99:A(Z)=99
970 IF Z<73 AND Z>55 AND NOT(Z=67 OR Z=68
     OR Z=60 OR Z=61 OR Z=62) A(Z)=1
```

```
980 IF Z<54 AND Z>42 AND NOT(Z=47
      OR Z=48 OR Z=49) A(Z)=0
990 IF Z<41 AND Z>23 AND NOT(Z=34 OR Z=35
     OR Z=36 OR Z=28 OR Z=29) A(Z)=-1
1000 NEXT Z
1010 VDU23;8202;0;0;0
1020 COM=0:HUM=0:ENDPROC
```

Whereas the computer plays only like a fairly good beginner at draughts, it manages a much stronger game in the following program, 'Othello'. Othello is a registered name, copyright Mine of Information, 1 Francis Avenue, St Albans. The game is a development of Reversi, which was invented in the 1880s, with a restriction in the opening moves.

Reversi was played on a standard draughts board, using pieces which were doublesided, black on one side, red on the other. R C Bell explains, in his book *Discovering Old Board Games* (Shire Publications Ltd., Aylesbury, 1980), that black begins the game by placing a piece black side up one of the four central squares on the empty board. Red replies by placing his or her first red side on another central square. "The four squares are covered in the first four turns of play and then the players continue alternately, placing their pieces on a square adjacent to one occupied by an enemy piece," says Bell. (Incidentally, this book, and others by Bell, are a great source of games ideas for computer programs.)

Any enemy pieces in a straight line between the latest piece placed and another one of the player's pieces is then flipped over to show the colour of the player. The winner is the player with the most pieces when the board is covered, or when neither player may move.

A computer programmer from Chichester, G J Suggett, has pointed out to me that most published Othello programs evaluate the best move for the computer to make "on the basis of maximising the number of captures made with a possible extra score given to certain positions, such as the corners. In fact, Mr Suggett said, "in the early stages of the game, positional play is far more important than making a large number of captures." In this version of Othello, great weight is placed upon positional play. The program was originally written in Microsoft by Graham Charlton of Romford, and adapted by me for the BBC Microcomputer.

```
10 CLS
20 X=ASC("X"):O=ASC("O")
30 DIM A(10,10)
40 R=0
50 FOR B=1TO10
60 FOR C=1TO10
```

```
 70 IF B<>1 AND C<>1 AND B<>10 AND C<>10 THEN A(B,C)=ASC(".")
 80 NEXT C
 90 NEXT B
100 A(5,5)=X
110 A(6,6)=X
120 A(6,5)=O
130 A(5,6)=O
140 P=0
150 PRINT
160 PRINT "DO YOU WANT TO GO"
170 PRINT"FIRST (1-YES,2-NO)"
180 INPUT W
190 CLS
200 GOSUB 720
210 IF W=1 THEN 600
220 PRINT"MY MOVE       "
230 S=O
240 T=X
250 H=0
260 FOR A=2 TO 9
270 FOR B=2 TO 9
280 IF A(A,B)<>46 THEN 540
290 Q=0
300 FOR C=-1 TO 1
310 FOR D=-1 TO 1
320 K=0
330 F=A
340 G=B
350 IF A(F+C,G+D)<>S THEN 400
360 K=K+1
370 F=F+C
380 G=G+D
390 GOTO 350
400 IF A(F+C,G+D)<>T THEN 420
410 Q=Q+K
420 NEXT D
430 NEXT C
440 IF A=2 OR A=9 THEN Q=Q*2
450 IF B=2 OR B=9 THEN Q=Q*2
460 IF A=3 OR A=8 THEN Q=Q/2
470 IF B=3 OR B=8 THEN Q=Q/2
480 IF (A=2 OR A=9) AND (B=3 OR B=8) THEN Q=Q/2
490 IF (A=3 OR A=8) AND (B=2 OR B=9) THEN Q=Q/2
500 IF Q<H OR Q=0 OR (RND(1)>0.3 AND Q=H) THEN 540
510 H=Q
520 M=A
530 N=B
540 NEXT B
550 NEXT A
560 IF H=0 AND R=0 THEN 1100
570 IF H=0 THEN 590
580 GOSUB 930
590 GOSUB 720
600 INPUT"YOUR GO ",R
610 S=X
```

```
620 T=O:REM LETTER O
630 REM 0 TO PASS
640 IF R=0 THEN 700
650 IF R<11 OR R>88 THEN 600
660 M=INT(R/10)+1
670 N=R-10*INT(R/10)+1
680 IF A(M,N)<>ASC(".") THEN GOTO 600
690 GOSUB 930
700 GOSUB 720
710 GOTO 220
720 PRINT TAB(0,0);
730 C=0
740 H=0
750 PRINT:PRINT TAB(10);"OTHELLO":PRINT
760 PRINT TAB(10);"12345678"
770 FOR B=2 TO 9
780 PRINT B-1;
790 FOR D=2 TO 9
800 PRINT CHR$(A(B,D));
810 IF A(B,D)=X THEN C=C+1
820 IF A(B,D)=O THEN H=H+1
830 NEXT D
840 PRINT STR$(B-1)
850 NEXT B
860 PRINT TAB(10);"12345678"
870 PRINT
880 PRINT"I HAVE ";C;"   "
890 PRINT
900 PRINT"YOU HAVE ";H;"   "
910 PRINT
920 RETURN
930 FOR C=-1 TO 1
940 FOR D=-1 TO 1
950 F=M
960 G=N
970 IF A(F+C,G+D)<>S THEN 1010
980 F=F+C
990 G=G+D
1000 GOTO 970
1010 IF A(F+C,G+D)<>T THEN 1070
1020 A(F,G)=T
1030 IF M=F AND N=G THEN 1070
1040 F=F-C
1050 G=G-D
1060 GOTO 1020
1070 NEXT D
1080 NEXT C
1090 RETURN
1100 GOSUB 720
1110 IF C>H THEN PRINT"I WON ";C;",";H
1120 IF C<H THEN PRINT"YOU WON ";C;",";H
1130 IF H=C THEN PRINT"IT'S A DRAW!"
1140 END
```

We finally have a number of programs which you may like to try on your BBC Microcomputer. There will be no 'commentary' with them, as they are primarily designed for use, rather than as teaching aids. However, you are sure to discover many new ideas for programs by reading through the listings, and working out how they do what they do. The first program is one of the old classics of computer games, a Lunar Lander. Listings of other programs will follow this one, without introduction.

```
 10 REM *Lunar Lander*
 20 M=0:T=0:S=0:H=4000+RND(1000)
 30 F=5000/RND(3):Q=-17:B=1
 40 CLS:VDU23;8202;0;0;0
 50 MODE7:GOTO210
 60 REM**************
 70 PRINT CHR$(128+RND(6));"(+ is towards Luna)";
 80 INPUT Z:IF Z<-30 OR Z>30 THEN 80
 90 PRINT CHR$(128+RND(6));"For how many seconds";
100 INPUT E:E=E+1
110 PRINT TAB(0,15);"                              "/"
"/"                              "
120 T=T+E
130 S=S+10+3*E*((Z+1)/B)
140 F=F-3*E*ABS(Z*RND(3))
150 IF F<500 PRINT TAB(20,6);CHR$(128+RND(6));"FUEL
    LOW":SOUND1,-15,250,255:SOUND2,-15,240,255
160 H=H-E*S
170 IF H<20 AND H>-10 AND S<12 THEN 430
180 IF H<-10 OR F<1 THEN 370
190 IF RND(10)=5 AND M<>2 THEN GOSUB 480
200 PROCdelay
210 PRINT TAB(0,8);CHR$(129+M);"HEIGHT ABOVE SURFACE: "INT(H)"    "
220 PROCdelay
230 IF Q<>-17 THEN Q=Q-RND(16)
240 IF Q<0 AND Q>-17 THEN 370
250 IF Q<>-17 PRINT CHR$(128+RND(6));"OXYGEN LEFT: "Q"    "
260 PROCdelay
270 PRINTCHR$(129+M);"VELOCITY: "INT(S)"    "
280 IF B<>1 PRINT CHR$(129+M);"WARNING - THRUST ERRATIC"
290 PROCdelay
300 PRINT CHR$(129+M);"FUEL LEFT: "INT(F)"     "
310 PROCdelay
320 PRINT CHR$(129+M);"FLIGHT TIME: "T
330 FOR A=1 TO 20:PRINT CHR$(128+RND(6));"*";:NEXT
340 PRINT TAB(0,15);CHR$(128+RND(6));"THRUST (-30 TO 30)"
350 GOTO 70
360 REM***********
370 REPEAT
380 PRINT TAB(RND(4));CHR$(128+RND(6));"**CRASH**
      YOU HIT THE SURFACE AT"
390 PRINT TAB(RND(7));CHR$(128+RND(6));ABS(S)"
          METRES PER SECOND"
400 SOUND 0,-15,RND(3),RND(10)
410 UNTIL FALSE
420 REM***********
430 REPEAT
440 PRINT CHR$(134);"SUCCESSFUL LANDING"'
450 PRINT CHR$(134);"FINAL VELOCITY: "ABS(S)
460 UNTIL FALSE
470 REM**************
```

```
480 CLS:M=M+1:U=RND(32000)
490 FOR V=1 TO 4
500 PRINT 'CHR$(128+RND(6));"HOUSTON, WE HAVE A PROBLEM!"
510 SOUND 0,-15,RND(4),RND(10)
520 PRINT TAB(RND(20));CHR$(129);"DANGER!!"
530 NEXT
540 PRINT ''CHR$(129);"MALFUNCTION.";CHR$(130);"USE
          COMPUTER ACCESS"
550 PRINT TAB(5);CHR$(130);"CODE "U" FOR DETAILS"
560 INPUT V
570 CLS
580 IF U<>V S=INT(S*H/5):GOTO370
590 P=RND(2)
600 ON P GOSUB 670,700
610 PRINT 'CHR$(128+RND(6));"PRESS ANY KEY TO RETURN TO FLIGHT"
620 A$=GET$
630 CLS
640 PROCdelay
650 RETURN
660 REM****************
670 Q=101+RND(19)
680 PRINT TAB(0,9);CHR$(130);"OXYGEN METER UNRELIABLE"
690 RETURN
700 B=B+RND(3)
710 PRINT TAB(0,9);CHR$(130);"THRUST CONTROL ERRATIC"
720 RETURN
730 REM****************
740 DEF PROCdelay
750 TIME=0:REPEAT UNTIL TIME=50
760 SOUND 3,-15,RND(254),2
770 ENDPROC
```

```
 10 REM MASTERTRIO
 20 REM 'Mastermind' WITH 3 DIGITS
 30 MODE 7:PRINT'''''
 40 DIM A(3),B(3)
 50 FOR Z=1 TO 3
 60 A(Z)=RND(9)
 70 NEXT Z
 80 IF A(1)=A(2) OR A(1)=A(3) OR A(2)=A(3) THEN 50
 90 D=100*A(1)+10*A(2)+A(3)
100 FOR C=1 TO 10
110 PRINT CHR$(129+RND(5));CHR$(157);CHR$(129);
120 PRINT "WHAT IS YOUR GUESS NO. ";C;
130 SOUND 1,-15,RND(10)+50,2
140 INPUT X
150 B(1)=INT(X/100)
160 B(2)=INT((X-100*B(1))/10)
170 B(3)=X-100*B(1)-10*B(2)
180 IF D=X THEN 430
190 W=0
200 N=0
210 FOR E=1 TO 3
220 IF A(E)<>B(E) THEN 250
230 N=N+1
240 A(E)=0
250 NEXT E
260 FOR F=1 TO 3
270 IF A(F)=0 THEN 320
280 FOR E=1 TO 3
290 IF B(F)<>A(E) THEN 310
300 W=W+1
```

```
310 NEXT E
320 NEXT F
330 A(1)=INT(D/100)
340 A(2)=INT((D-100*A(1))/10)
350 A(3)=D-100*A(1)-10*A(2)
360 SOUND 2,-15,RND(20)+235,2
370 PRINT CHR$(129);"YOU SCORED ";N;" BLACKS ";
380 PRINT "AND ";W;" WHITES"
390 NEXT C
400 PRINT CHR$(129);CHR$(157);CHR$(131);
410 PRINT "MY NUMBER WAS ";A(1);A(2);A(3)
420 END
430 PRINT CHR$(129);CHR$(157);CHR$(131);"CONGRATULATIONS"
440 END
```

```
 10 REM 'MATCHSTICKS'
 20 M=0:E=0
 30 Z=16+RND(7)
 40 IF 2*INT(Z/2)=Z THEN 30
 50 H=2+RND(2)
 60 REPEAT
 70 CLS
 80 PRINT ''CHR$(132);"MAXIMUM TO TAKE ";H
 90 GOSUB 320
100 IF E>0  PRINT 'CHR$(130);"YOU TOOK ";E
110 IF E>0 PRINT CHR$(129);"  I TOOK ";Q'
120 FOR K=1 TO Z
130 PRINT CHR$(129);CHR$(157);CHR$(129+RND(5));
140 PRINT K;
150 IF RND(1)>0.6  PRINT
160 NEXT K
170 GOSUB 320
180 PRINT "HOW MANY WILL YOU TAKE";
190 INPUT E
200 IF E>H OR E<1 THEN 190
210 Z=Z-E
220 GOSUB 320
230 IF Z<1 PRINT "YOU TOOK THE LAST ONE"
240 IF Z<1 PRINT "SO I WIN":END
250 Q=Z-1-INT((Z-1)/(H+1))*(H+1)-INT(RND(1)*2)+INT(RND(1)*2)
260 IF Q<1 OR Q>H THEN 250
270 GOSUB 320
280 Z=Z-Q
290 IF Z=0 THEN PRINT CHR$(129);"I TOOK ";Q
300 IF Z=0 THEN PRINT "SO YOU WIN":END
310 UNTIL FALSE
320 FOR T=1 TO 4
330 PRINT
340 NEXT T
350 RETURN
```

```
10 REM 'PERSONAL ACCOUNTS'
20 REM ADAPTED FROM ZX80 PROGRAM
30 REM WRITTEN BY RON JONES
40 DIM A(6)
50 B=0
60 GOSUB 290
70 INPUT "ANY CHANGES (Y OR N)",Z$
```

```
 80 IF Z$="N" THEN 150
 90 INPUT "NUMBER",K
100 IF K>6 OR K<1 THEN 90
110 INPUT "NEW AMOUNT"E
120 IF K=6 THEN E=-E
130 A(K)=E
140 GOTO 60
150 INPUT "ENTER SALARY",S
160 GOSUB 290
170 R=S-T+B
180 PRINT '"BALANCE $";R
190 B=R
200 PRINT
210 PRINT "ENTER 1 TO END, 2 TO SET BALANCE TO"
220 PRINT "ZERO, OR 3 TO RUN AGAIN, STARTING"
230 PRINT "WITH CURRENT BALANCE"
240 INPUT Q
250 IF Q<1 OR Q>3 THEN 240
260 IF Q=1 THEN END
270 IF Q=2 THEN 50
280 IF Q=3 THEN 60
290 T=0
300 CLS
310 PRINT '"PERSONAL ACCOUNTS"
320 PRINT ''"PREVIOUS BALANCE ";B
330 PRINT
340 FOR F=1 TO 6
350 PRINT F;" ";
360 ON F GOSUB 420,440,460,480,500,520
370 PRINT "  $";A(F)'
380 T=T+A(F)
390 NEXT F
400 PRINT
410 RETURN
420 PRINT "CHEQUES OUT";
430 RETURN
440 PRINT "CREDIT CARDS";
450 RETURN
460 PRINT "RATES";
470 RETURN
480 PRINT "MORTGAGE";
490 RETURN
```

```
500 PRINT "STANDING ORDERS";
510 RETURN
520 PRINT "MONEY IN";
530 RETURN
```

```
 10 REM DIGITAL CLOCK
 20 PRINT "ENTER HOURS"
 30 INPUT H
 40 PRINT H;" ENTER MINUTES"
 50 INPUT M
 60 PRINT H;" ENTER SECONDS"
 70 INPUT S
 80 MODE7
 90 REPEAT
100 PRINTTAB(0,10);CHR$(129);CHR$(157);
110 PRINT TAB(15);CHR$(131);H;" ";
120 IF M<10 THEN PRINT "0";
130 PRINT ;M;" ";
140 IF S<10 THEN PRINT "0";
150 PRINT ;S;" "
160 TIME=0
170 REPEAT UNTIL TIME=98
180 S=S+1
190 IF S=60 THEN M=M+1
200 IF S=60 THEN S=0
210 IF M=60 THEN H=H+1
220 IF M=60 THEN M=0
230 IF H=13 THEN H=1
240 UNTIL FALSE
```

```
 10 REM INTEREST ON A LOAN
 20 CLS
 30 INPUT ''''"PERCENTAGE INTEREST RATE"E
 40 INPUT ''"PRINCIPAL ($)"P
 50 INPUT '"LENGTH OF LOAN (YEARS)  "Y
 60 Y=365*Y
 70 INPUT  '"AND DAYS"D
 80 D=D+Y
 90 E=INT(P*D/36500*E+0.9)
100 PRINT ''''"INTEREST IS $";E
110 PRINT ''"TOTAL (PRINCIPAL PLUS"
120 PRINT "INTEREST) IS $";P+E
```

```
 10 REM TEMPERATURE CONVERSION
 20 INPUT '"ENTER HIGHEST TEMPERATURE IN F"A
 30 INPUT '"AND LOWEST"B
 40 INPUT''"ENTER INCREMENT"C
 50 PRINT CHR$(131);"F";TAB(10);"C";TAB(20);"K"
 60 FOR T=B TO A+C STEP C
 70 M=INT(5*(T-32)/9)
 80 K=M+273
 90 PRINT CHR$(129);INT(T);TAB(10);M;TAB(20);K
100 NEXT T
```

```
 10 REM STOCK RECORD
 20 MODE7
 30 INPUT ''"HOW MANY CATEGORIES",A
 40 DIM A$(A,10)
 50 DIM Z(A)
 60 FOR C=1 TO A
 70 PRINT "ENTER THE NAME OF CATEGORY ";C
 80 INPUT M$
 90 A$(C,10)=M$
100 NEXT C
110 REPEAT
120 CLS
130 PRINT '''
140 FOR C=1 TO A
150 PRINT C;" ";A$(C,10);" ";Z(C)
160 NEXT C
170 PRINT '"NUMBER OF CATEGORY?"
180 PRINT ,"-99 TO END"
190 INPUT D
200 IF D>A THEN 190
210 IF D=-99 THEN END
220 PRINT A$(D,10),"VALUE";
230 INPUT E
240 Z(D)=Z(D)+E
250 UNTIL FALSE
```

```
 10 REM SQUARE ROOTS BY ITERATION
 20 MODE7
 30 INPUT''"NUMBER TO FIND ROOT OF",B
 40 IF B<=0 THEN 30
 50 A=RND(B)
 60 X=B/A
 70 Y=(X+A)/2
 80 PRINT 'Y
 90 IF A=Y THEN 120
100 A=Y
110 GOTO 60
120 PRINT ''"THE SQUARE ROOT OF ";B;" IS ";Y
```

```
   10 REM **************************
   20 REM French bathrooms
   30 REM (C) 1982 Jeremy Ruston
   40 REM **************************
   50 DIM X(4),Y(4),XD(4),YD(4)
   60 MODE 4
   70 REM **************************
   80 FOR X=0 TO 1249 STEP 250
   90 FOR Y=0 TO 999 STEP 250
  100 XM=X DIV 250
  110 YM=Y DIV 250
  120 IF (XM MOD 2)=(YM MOD 2) THEN SMU=
0.1 ELSE SMU=0.9
  130 PROCTWIRL(X+250,Y+250,X+250,Y,X,Y,
X,Y+250,SMU)
  140 NEXT Y
  150 NEXT X
  160 END
  170 REM **************************
  180 DEF PROCTWIRL(X1,Y1,X2,Y2,X3,Y3,X4
,Y4,SMU)
  190 X(1)=X1:X(2)=X2:X(3)=X3:X(4)=X4
  200 Y(1)=Y1:Y(2)=Y2:Y(3)=Y3:Y(4)=Y4
  210 LOCAL RMU,I,J,NJ
  220 RMU=1-SMU
  230 FOR I=1 TO 21
  240 MOVE X(4),Y(4)
  250 FOR J=1 TO 4
  260 DRAW X(J),Y(J)
  270 NJ=(J MOD 4)+1
  280 XD(J)=RMU*X(J)+SMU*X(NJ)
  290 YD(J)=RMU*Y(J)+SMU*Y(NJ)
  300 NEXT J
  310 FOR J=1 TO 4
  320 X(J)=XD(J)
  330 Y(J)=YD(J)
  340 NEXT J
  350 NEXT I
  360 ENDPROC
  380 REM Change 230 to:
  390'FOR I=1 TO 21 STEP 2'
```

```
 10 REM CARD 21
 20 MODE7
 30 PRINT '''
 40 M=0
 50 GOTO 270
 60 CA=RND(11)
 70 IF CA=11 AND D+CA>21 CA=1
 80 D=D+CA
 90 IF M>1 PRINT "A ";CA;" HAS BEEN DEALT"
100 M=M+1
110 RETURN
120 CA=RND(11)
130 IF CA=11 AND B+CA>21 CA=1
140 B=B+CA
150 RETURN
160 PRINT "ANOTHER CARD (1) OR WILL"
170 PRINT "YOU STAND (0)";
180 INPUT G
190 *FX 15,0
200 RETURN
210 PRINT ''''"ANOTHER GAME, CARD-SHARP";
220 INPUT A$
230 M=0
240 IF A$<>"NO" OR A$<>"N" THEN 270
250 PRINT "OK, THANKS FOR PLAYING"
260 END
270 D=0
280 B=0
290 GOSUB 60
300 H=CA
310 GOSUB 60
320 A=CA
330 GOSUB 120
340 E=CA
350 GOSUB 120
360 F=CA
370 B$="THE COMPUTER HAS "
380 C$="THE HUMAN HAS "
390 PRINT B$;H
400 PRINT C$;E;" AND ";F
410 PRINT "    TOTALLING ";E+F
420 D=H+A
```

```
430 B=E+F
440 IF B=21 THEN 520
450 GOSUB 160
460 IF G=1 THEN 550
470 IF D<17 THEN 600
480 IF D<>21 THEN PRINT B$;D
490 IF B<>21 THEN PRINT C$;B
500 IF B=D AND B<>21 THEN PRINT
 "SO THIS ROUND IS A DRAW"
510 IF B>21 OR D>21 THEN 210
520 IF B>D THEN PRINT "   YOU WIN"
530 IF D>B THEN PRINT "      I WIN"
540 GOTO 210
550 GOSUB 120
560 PRINT C$;CA;" TOTAL: ";B
570 IF B>21 THEN PRINT ">>>>> BUSTED"
580 IF B>21 THEN 210
590 GOTO 450
600 PRINT B$;D
610 FOR Y=1 TO 900:NEXT Y
620 GOSUB 60
630 PRINT "TOTAL IS NOW ";D
640 FOR Y=1 TO 900:NEXT Y
650 IF D>21 THEN PRINT ">>>> BUSTED"
660 IF D>21 AND B<=21 PRINT "SO YOU WIN"
670 IF D<17 THEN 620
680 GOTO 480
```

```
 10 REM 9TH HOLE
 20 MODE 7
 30 C=0
 40 FOR Z=1 TO 9:SC=0
 50 J=RND(12)-1
 60 Q=RND(3)+2
 70 IF Q=5 Q$="FIVE"
 80 IF Q=4 Q$="FOUR"
 90 IF Q=3 Q$="THREE"
100 PRINT '''CHR$(128+RND(6));"###HOLE NUMBER ";Z;"###"
110 PRINT ''CHR$(129);CHR$(157);CHR$(151);
120 PRINT "HOLE DIFFICULTY IS ";Q$''''
130 GOSUB 370
140 PRINT CHR$(139);'"STROKE ";:INPUT A
150 IF J>26 A=-A
160 J=J+INT(A/RND(Q))
170 GOSUB 370
180 SC=SC+1
190 PRINT CHR$(131);CHR$(157);CHR$(129);"AFTER THAT ";
200 PRINT "STROKE YOUR SCORE IS ";SC'
```

```
210 PRINT
220 IF J<>26 THEN 140
230 GOSUB 450
240 C=C+SC
250 PRINT 'CHR$(128+RND(6));"SCORE FOR ";Z;
260 PRINT " HOLES IS ";C
270 FOR M=1 TO 1000:NEXT
280 FOR L=1 TO 20:FOR Y=1 TO L
290 PRINT " ";:SOUND 18,-15,200-L,1
300 SOUND 19,-15,200-Y,1
310 NEXT Y
320 PRINT CHR$(128+RND(6));CHR$(157);
330 PRINT CHR$(128+RND(6));"STAND BY"
340 NEXT L
350 NEXT Z
360 GOTO 550
370 IF J>30 THEN J=30
380 FOR K=1 TO J-2
390 PRINT " ";
400 NEXT K
410 PRINT CHR$(129);"o"
420 PRINT CHR$(130);"^^^^^^^^^^^^^^";
430 PRINT "^^^^^^^^^^ ^^^^^^^^^"
440 RETURN
450 FOR L=1 TO 30
460 PRINT CHR$(128+RND(6));
470 PRINT "YOU DID IT IN ";SC;" STROKES"
480 SOUND 17,-15,200-L,2
490 SOUND 18,-15,10*L,2
500 NEXT L
510 PRINT
520 PRINT CHR$(130);"^^^^^^^^^^^^^^^^^^^^^^^";
530 PRINT CHR$(129);"o";CHR$(130);"^^^^^^^"
540 RETURN
550 PRINT "END OF THAT ROUND"
560 PRINT
570 PRINT "YOU SCORED ";C
580 PRINT "YOUR AVERAGE WAS ";INT(C/9)
590 PRINT
600 PRINT "DO YOU WANT ANOTHER ROUND?"
610 INPUT T$
620 IF T$<>"NO" THEN RUN
630 PRINT "OK, THANKS FOR PLAYING, CHAMP"
```

```
 10 REM HIDE'N'SEEK
 20 REM This is just the raw bones of
 30 REM a program to 'hide' something
 40 REM on a 10 X 10 grid. You are
 50 REM sure to be able to improve it.
 60 MODE7
 70 A=RND(10)
 80 B=RND(10)
 90 FOR Z=1 TO 10
100 PRINT
110 PRINT '"YOU HAVE JUST ";11-Z;
```

```
120 PRINT " SECONDS LEFT"
130 PRINT "WHERE IS THE GOLD?"
140 INPUT C,D
150 PRINT " 1234567890"
160 IF A=C AND B=D THEN 400
170 FOR F=1 TO C-1
180 PRINT ;F
190 NEXT F
200 PRINT;F;
210 FOR F=1 TO D-1
220 PRINT " ";
230 NEXT F
240 PRINT "*"
250 FOR F=C-1 TO 10
260 PRINT ;F
270 NEXT F
280 PRINT "IT IS NOT AT ";C;" ";D
290 PRINT '"HERE IS A CLUE:-"
300 PRINT "TRY TO THE ";
310 IF A>C THEN PRINT "SOUTH";
320 IF A<C THEN PRINT "NORTH";
330 IF B<D THEN PRINT "WEST"
340 IF B>D THEN PRINT "EAST"
350 NEXT Z
360 PRINT
370 PRINT "TIME IS UP. THE GOLD ";
380 PRINT "WAS AT ";A;" ";B
390 END
400 PRINT "WELL DONE, YOU'VE FOUND ";
410 PRINT "$";1000*RND(20)
420 END
```

```
 10 REM *Piano dementia*
 20 MODE7
 30 FOR J=255 TO 0 STEP -1
 40 SOUND 17,-15,J,1
 50 PRINT CHR$(128+RND(6));CHR$(157);
 60 NEXT J
 70 FOR J=0 TO 255
 80 SOUND 17,-15,J,1
 90 NEXT J
100 REPEAT
110 A=RND(47)+43
120 M=-53*(A=90)-61*(A=88)-69*(A=67)-73*(A=86)
     -81*(A=66)-89*(A=78)-97*(A=77)-101*(A=44)
130 IF M=0 THEN 110
```

```
140 PRINT CHR$(128+RND(6));CHR$(157);CHR$(128+RND(6));M,M,M
150 FOR J=-15 TO -1 STEP RND(3)
160 SOUND 1,J,M,RND(3)
170 SOUND 2,J,M,RND(5)
180 NEXT J
190 UNTIL RND(10)=1
200 PRINT CHR$(128+RND(6));"#####"
210 FOR J=-15 TO -1
220 SOUND 1,J,53,5
230 NEXT J
240 FOR J=1 TO 255 STEP RND(10)
250 SOUND 17,-15,J,2
260 SOUND 18,-15,255-J,2
270 SOUND 19,-15,J/2,2
280 PRINT CHR$(128+RND(6));CHR$(157);
290 NEXT J
300 GOTO 30
```

Improving your programs

You've probably gone through several stages as you develop your programming skills. After the first, brief struggle with BASIC, you suddenly discovered you could, after a fashion, write programs which ran. They may have looked pretty convoluted when you looked at their listings, and friends may have needed a detailed explanation from you before they knew what to do when running the programs, but as least they worked.

There comes a stage when you decide you're going to have to do better than that. But while you may be vaguely dissatisfied with your programs, you may not have much idea of how to go about becoming a better programmer. Here are a few guidelines which may help.

First, have a look at a printout of your listing. Programs linked by REM statements look better, and are easier to understand when you return to them after a break. Of course, shortage of memory may preclude the luxury of REM statements, but if you have the memory, you should include them. REM statements filled just with a line of asterisks can prove quite useful in separating each major section of the program. Examine any unconditional GOTO critically. Too many GOTOs leapfrogging over other parts of the program show a lack of directed thinking, make programs run more slowly, and can make them almost impossible to decipher.

It is very good programming practice, as I have suggested earlier, to have each of the main sections of the program (like the one which

assigns the variables at the beginning of a run, the one which prints out the board, the one which works out who has won, and so on) in separate subroutines or procedures. The beginning of your program could well look like this:

```
 10  REM *NAME OF PROGRAM*
 20  REM ASSIGN VARIABLES
 30  GOSUB 9000
 40  REM PRINT BOARD
 50  GOSUB 8000
 60  REM HUMAN'S MOVE
 70  GOSUB 7000
 80  REM COMPUTER'S MOVE
 90  GOSUB 6000
100  REM CHECK IF GAME OVER
110  GOSUB 5000
120  GOTO 50
```

As you can see, this ensures that the program actually cycles through a continuous loop over and over again, until the program terminates with the "CHECK IF GAME OVER" subroutine. You can actually write a series of lines like these before you start writing anything else, and even before you know how you are going to actually perform some of the tasks within the subroutine.

Then you can write the program module by module, making sure that each module works before going onto the next. It is relatively easy to debug a program like this, and far simpler to keep an image of 'where everything is' when you do this, then when you just allow a program to, more or less, write itself.

The listing should be, then, as transparent as you can make it, both for your own present debugging, and for future understanding of what bit carries out what task. The output of the program should also look good. Again, if memory is not a problem, make sure the display is clear and uncluttered. Use blank PRINT lines to space it out, use rules and graphic symbols or whatever to break the screen up into logical sections and so on. Once you have a program working satisfactorily, it is worth spending extra time on the subroutine which controls the display. Here you'll appreciate again the advantage of having all the display handling in one subroutine, as it will be easy to know where to go to enhance the display.

Of course, as we live in a far from ideal world, it is unlikely that every single display command can be contained within one subroutine, but if you aim towards that end, it will make subsequent working upon the program much easier than it might be otherwise.

The 'structured' approach outlined also helps you realise another aim of a good program — to do what you expected it to, every time you

run it. You should write a program so that, even if you are not present when a friend decides to run it for the first time, it performs as expected. This means not only, of course, that it is properly debugged, but that the instructions (which can be contained within the ASSIGN VARIABLES subroutine or procedure) are clear and complete.

The user prompts should be clear, so the human operators know whether to enter a number, a series of numbers, a word, a date, a mixture of letters and numbers, and so on. The program has to assume that the operator is a complete idiot, and that no matter how clearly the instructions and/or user prompts are stated, he or she will attempt to do things the wrong way. A classic example of this is the entering of dates. 'Mug traps', as the routines to reject erroneous input from the operator are called, should be set up to reject a date being entered in a form which the computer cannot understand (such as the month before the day) or which is clearly wrong (such as entered the 32nd of February). You should ensure that, no matter what the operator does, the program does not crash or otherwise misbehave. This can happen if the program was expecting a numerical input, and the operator tried to enter a letter or a word, or hit RETURN without entering anything at all. You can get around this by always allowing a string input, going back for another input if the empty string is entered, and taking the ASC, VAL or CODE of the input to turn it into numerical form.

Documentation is an area of programming which is often neglected. It is virtually essential for a program which is intended for publication, and most advisable for long programs which you've written for yourself. At the least, the documentation should include a list of variables, an explanation of the program structure (which should be easy to do if you've followed the 'modular' approach advised earlier), and brief instructions, especially if the program itself does not contain instructions. A sample run showing the kind of inputs, and the nature and layout of the program outputs, is also useful.

Your program should run as quickly as possible. Every time there is a subroutine or GOTO call, the computer must search through the whole program, line by line, to find the specified line number so placing often used subroutines near the beginning of the program will speed them up fractionally. That is why the instructions are often placed right at the end. You do not want the computer to have to wade through the initialisation and instruction lines every time it has been told to GOTO or GOSUB looking for the destination, or return line number. Use procedures rather than subroutines if you can as these run more quickly then subroutines. Use variables instead of constants. The computer takes a little more time to change a constant into a number than it does for it to look up a value in an array, for example. Define often-used variables first, so they will occupy the early slots in the varibles store. The computer will search the store only until it finds the

variable it wants, so there is no point in getting it to look at more entries than absolutely necessary.

Finally, and this is by far the best way to test a program you've written, call in a friend and sit him or her in front of the TV, and tell them to press RUN, without you saying anything, and just sit back and watch. If there is any hesitation, or the program hiccups, you have more work to do.

In summary, then:

- Use REM statements
- Make program listing neat and logical
- Use structured programming techniques, controlling the program through a loop of subroutine calls, or procedures
- Examine unconditional GOTO commands critically, using REPEAT/UNTIL instead if it is appropriate
- Make output display attractive and clear
- Ensure all user prompts are clear
- Add 'mugtraps' on all user input
- Document your programs, even if you just make a list of variables
- Make your program run as quickly as possible
- Test programs by allowing someone unfamiliar with the program to run it.

Good programming.

CONTENTS:

Introduction -- 3
The PRINT statement ------------------------------ 5
Editing -- 8
LIST and RENUMBER -------------------------------- 9
RUN, STOP, END, NEW, OLD ----------------------- 10
PRINT formatting and TAB ------------------------ 12
TABULATOR ROCKET RANGE ---------------- 15
SAVE -- 16
TAB (X) and TAB (X,Y) --------------------------- 17
SQUASH -------------------------------- 18
Random numbers ---------------------------------- 21
DICE ROLLER -------------------------- 23
ACEY DUECY --------------------------- 25
Variables -- 28
String variables --------------------------------- 31
CRICKETS ----------------------------- 31
INPUT --- 33
BIRD CAGE ---------------------------- 35
COMPOUND INTEREST -------------------- 38
ARITHMETIC MEAN ---------------------- 39
GOTO -- 41
IF...THEN GOTO ---------------------------------- 42
MATCHSTICKS -------------------------- 44
IF...THEN...ELSE -------------------------------- 46
GRAPHS ------------------------------- 46
FOR/NEXT loops ---------------------------------- 51
Nested loops ------------------------------------ 54
STEP -- 56
REPEAT/UNTIL ------------------------------------ 58
GOSUB and RETURN -------------------------------- 59
BLACKJACK ---------------------------- 61
ON GOTO...ON GOSUB ------------------------------ 64
POETRY ------------------------------- 65
DIM and ARRAYS ---------------------------------- 68
MASTERMIND --------------------------- 73
PERSONAL ACCOUNTS -------------------- 74
String arrays ----------------------------------- 77
STRING SORT -------------------------- 78
String handling --------------------------------- 79
ALPHA -------------------------------- 81
MUSIC MAKER -------------------------- 82
GET, GET$, INKEY, INKEY$ ------------------------ 84
PREDICTION --------------------------- 85
MAZE MAKER --------------------------- 87
ROAD RUNNER -------------------------- 88
SOUND --- 90
BAMBOO FLUTES ------------------------ 91
STEAM TRAIN -------------------------- 91
PIANO -------------------------------- 92
AUTO PIANO --------------------------- 93
ENVELOPE --- 96
READ/DATA/RESTORE ------------------------------- 98

MASTERING THE GRAPHICS ------------------------- 102
DOUBLE HEIGHT ------------------------- 104
CHUNKY GRAPHICS ------------------------ 105
CONCATENATION -------------------------- 107
MASTERMIND 11 ------------------------- 109
MODES 0 to 7 ---------------------------- 110
GCOL/CLG ------------------------------ 113
PLOT --- 116
ELECTRIC SPARK ------------------------ 117
ICE CAVE -------------------------------- 118
ZEBRA TRIANGLES, ROTATING SQUARES, SHRINKER, MOIRE-LACE, SINE CURVE, TUNNEL TUNER, SINE RIBBON, ORBITAL SKETCHER, ROLLER COASTER, COSMIC TABLE MAT------------------ 120
VDU drivers -------------------------------------- 126
CHARACTER REDEFINITION ---------------- 133
SIDEWAYS SCROLL ----------------------- 133
CORRUPTION ---------------------------- 134
Introduction to arithmetic -------------------- 135
ARITHMETIC PROGRESSION ---------------- 135
CIRCULAR SECTIONS --------------------- 137
PRIME NUMBERS ------------------------- 138
DAY OF THE WEEK ----------------------- 139
SPECIES ------------------------------- 140
FUNCTIONS ------------------------------------- 141
DEFINING FUNCTIONS -------------------- 142
BAT ------------------------------------- 143
LOCAL VARIABLES ------------------------------ 144
SPIROGRAPH ---------------------------- 145
Procedures ------------------------------------- 146
TWENTY ONE ---------------------------- 150
SEVEN-UP ------------------------------ 152
User-definable function keys ------------------- 158
File handling ---------------------------------- 160
PARROT SHOP -------------------------- 162
Draughts and other games ---------------------- 164
KIDDIE CHECKERS ----------------------- 168
DRAUGHTS ------------------------------ 170
OTHELLO ------------------------------- 174
LUNAR LANDER ------------------------- 177
MASTERTRIO ---------------------------- 178
MATCHSTICKS 11 ------------------------ 179
PERSONAL ACCOUNTS 11 ------------------ 179
DIGITAL CLOCK ------------------------- 181
INTEREST ON A LOAN -------------------- 181
STOCK RECORD -------------------------- 182
SQUARE ROOTS -------------------------- 182
FRENCH BATHROOMS --------------------- 183
CARD 21 ------------------------------- 184
9th HOLE ------------------------------ 185
HIDE'N'SEEK --------------------------- 186
PIANO DEMENTIA ------------------------ 187
Improving your programs ----------------------- 188

LET YOUR BBC MICRO TEACH YOU TO PROGRAM

This book by best-selling author Tim Hartnell is the ideal companion for you if the BBC Microcomputer is your first computer. It takes you, step by simple step, through programming in BBC BASIC – with a number of worthwhile programs – and then goes further, giving you leads to develop and expand your programming skills in the coming months.

Once you've mastered the fundamentals of BASIC, this book will help you understand and apply such things as the use of the ENVELOPE command, how to master the graphics and use them to best effect, the use of VDU drivers, graphic windows, how to define your own characters, the use of functions and procedures, and ways of writing better programs. There are a number of utility and games programs in the book, ready to run.

Tim Hartnell is the author of a number of books, including "GETTING ACQUAINTED WITH YOUR ZX81" and "SYMPHONY FOR A MELANCHOLY COMPUTER". He is a frequent contributor to computer magazines, and edits the magazine "ZX COMPUTING".

Another great book from Interface Publications