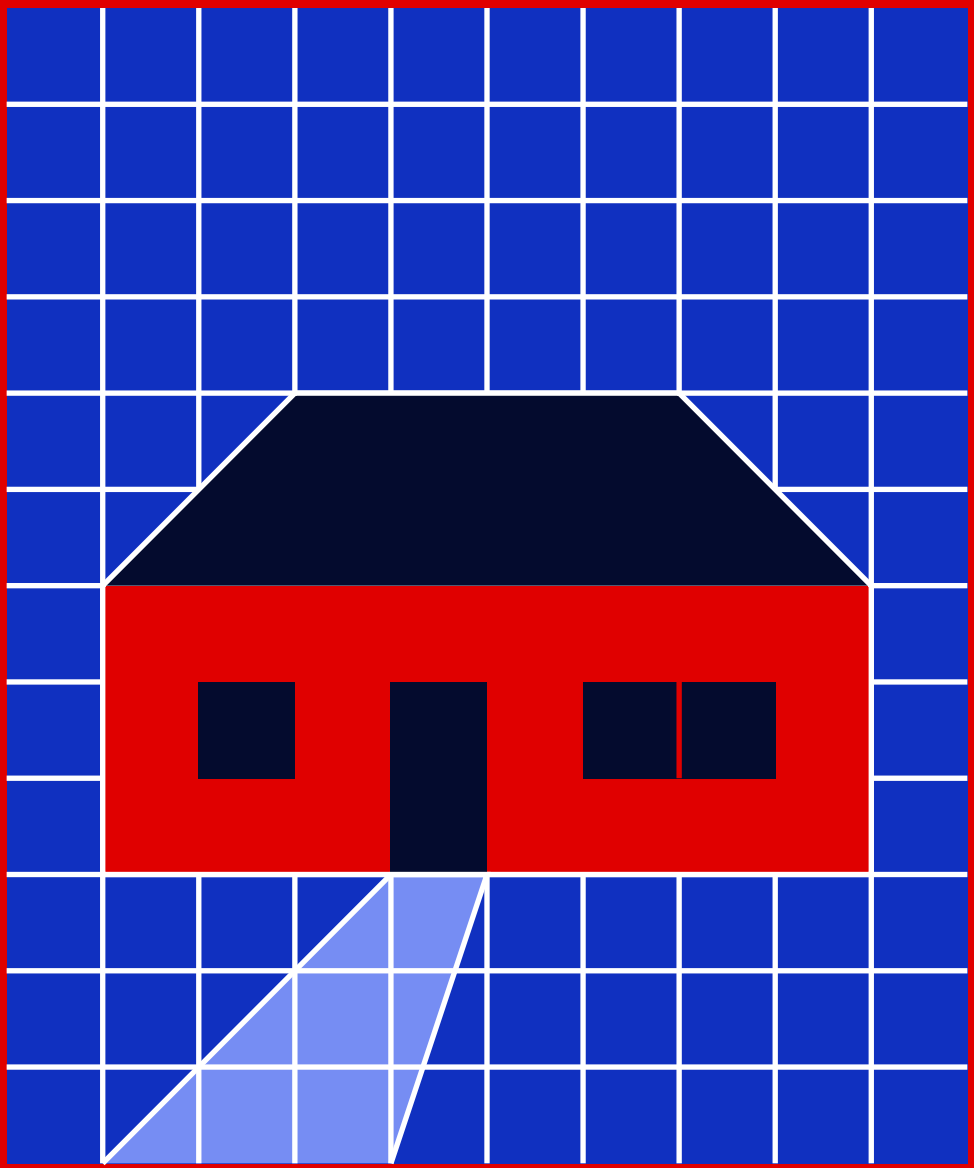
Introducing BBC BASIC

R. B. Coats



Introducing BBC BASIC

R.B. Coats

Principal Lecturer in Computer Studies
Leicester Polytechnic

Edward Arnold

First published in Great Britain 1984 by
Edward Arnold (Publishers) Ltd, 41 Bedford Square, London WC1B 3DQ

Edward Arnold, 300 North Charles Street, Baltimore, Maryland 21201, U.S.A.

Edward Arnold (Australia) Pty Ltd, 80 Waverley Road, Caulfield East, Victoria 3145, Australia

ISBN 0 7131 3520 4

Printed in Great Britain by Butler & Tanner Ltd, Frome and London

Preface

As the title suggests, the aim of this book is to introduce you to BASIC on the BBC computer. Because of its size, it cannot cover ALL features of BBC BASIC. However, by the end of the book, you will have met the important features of BASIC, and you should be able to use these features in simple programs. Indeed, if you complete all the exercises contained in the book, then you will have written some twenty-four programs.

'Introducing BBC BASIC' is written in such a way that young people (aged about 12 and upwards) will be able to use it. Simple examples are used throughout. These examples are easy to understand, and require no mathematics other than simple numeracy. Many of the examples are, in fact, drawn from a school environment, so the book is particularly relevant to schools and colleges.

This book is suitable for use as a classroom text. It is divided into relatively short units. Each unit focusses on a particular feature of BASIC, and contains:

explanatory text;
test questions;
practical exercises.

The majority of units are short enough for the explanatory text and the test questions to be covered in one lesson. The practical exercises may be undertaken individually by pupils on a computer if sufficient access to computers is available. Alternatively, the teacher may demonstrate these exercises on a single computer in front of the class. Those practical exercises which involve writing programs should be tackled by each pupil first writing his or her answer on paper. These answers can then by tried on a computer. Again, if sufficient access to computers is available, each pupil can type his/her own program into a computer and run it. Otherwise, the teacher can select one or two of the pupils' answers, and run those on a single computer in front of the whole class. Constructive criticism of programs written by others is a valuable part of learning to program.

There is a section at the end of the book containing additional questions, over and above those provided at the end of a unit. These questions cover fundamental ideas of BASIC. The additional questions on a particular unit may be tacked any time after completing the unit, either immediately to reinforce the ideas, or later as revision. If you can answer ALL these questions correctly, you will have gained a good grasp of the fundamentals of BASIC.

This book is also suitable as a self-study text, provided you have a BBC computer of your own, or have reasonable access to one. The practical approach encourages you to learn by trying things out directly on the computer, and the carefully sectioned text allows you to progress at your own pace.

How does 'Introducing BBC BASIC' (128 pages) relate to the book 'BBC BASIC' (256 pages) written by the same author and published by Edward Arnold? Both books introduce you to the important features of BASIC, and teach you how to use these features in simple programs. This book leaves you at this stage, however, whereas 'BBC BASIC' progresses to describe more advanced features of BASIC, and also describe how to design, construct, test and implement larger and more complex programs. Both books place strong emphasis on 'good' programming - sound programming techniques are taught throughout. In summary, 'Introducing BBC BASIC' will provide you with a good grasp of programming fundamentals, and a firm foundation should you wish to take programming further.

I am grateful to Peter Messer for checking the final draft of this book. I also wish to thank Leicester Polytechnic for the use of their computer facilities to prepare the book.

Finally, I am very grateful to my wife and children for their encouragement and help, without which this book could not possibly have been written. In particular, I would like to thank David (aged 13) and Martin (aged 10) for working through the units to check them.

February 1984 R.B.Coats

Contents

1 Introduction

1.1 Using this book

BASIC is a language which was designed to be easy for beginners to learn. Because of its simplicity, it has become widely available, and nearly all computers provide it. The aim of this book is to teach you how to program the BBC computer in BASIC.

This book is organised into Units. Each unit concentrates on a particular aspect of BASIC, and assumes that you understand the previous units. Therefore, the units must be tackled in order. In each unit there are three components:

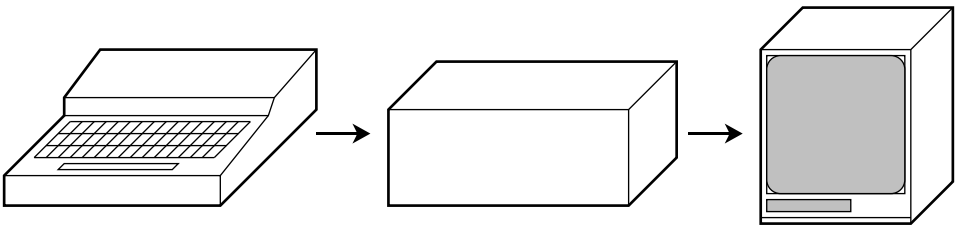
The written text	describing the particular aspect of BASIC covered in the unit.
Practical exercises	to be carried out on your computer. Learning to program is similar to learning to ride a bicycle - you learn by trying to ride, and not by reading about it. Hence, these exercises are very important, because they reinforce your understanding of the unit.
Questions	at the end of the unit, to challenge you to apply the knowledge that you have just acquired. Answers are provided in Appendix A to SOME of these questions; those with an answer are indicated by an * immediately following the question number.

All the practical exercises and the questions should be completed before going on to the next unit.

Appendix C contains Additional Questions for selected units of the book. These questions cover the fundamentals of BASIC; if you can answer them all correctly you will have a good grasp of these fundamentals. The questions on a particular unit may be tackled any time after completing that unit, either immediately to reinforce the ideas, or later as revision. Answers to ALL the Additional Questions are provided in Appendix D.

1.2 Computers

There are three main parts to a computer.



1. The KEYBOARD, which is used to enter information into the computer. The information that is entered into a computer is called the input.

2. The COMPUTER itself, which manipulates information. For example, it might add two numbers together. Or it might sort a list of names into alphabetical order.

3. The TELEVISION (or MONITOR), which is used to display the results produced by the computer. For example, it might show the answer obtained from adding the two numbers. Or it might display the names in alphabetical order. The information that is sent out by the computer to the screen is called the output.

1.3 The PRINT command

You can get the computer to do something by giving it a command. A command is typed on the keyboard. When you have finished typing the command, you must always press the RETURN key, to let the computer know that you are ready for it to obey the command. The computer will now obey the command, and then wait for you to enter another command.

PRINT is an example of a command. It tells the computer to display something on the screen.

Type

```
PRINT 3+1
```

and then press RETURN.

The computer should display 4 on the screen.

Normally, the word 'execute' is used to describe what a computer does with a command, rather than the word 'obey' ('execute' means 'carry out'). Hence, a computer is said to execute a command. You can tell when the computer is waiting for a command because it displays a 'prompt' on the screen. The prompt on the BBC computer is the > character. Pressing the RETURN key causes a new line to be taken. If you forget to press RETURN, it will be obvious since you will have two commands on the same line.

- -

Type

```
PRINT 5+2

PRINT 6+3

PRINT 7-4

PRINT 10-5
```

Did you remember to press the RETURN key each time? Note that '-' is on the same key as '=', and means subtract (take). Check that the computer's answers are correct.

- -

Type

```
PINT 3+1
```

Note that PRINT has been spelt incorrectly: you missed out the R. Typing mistakes are common, especially for the beginner. Fortunately, there is no harm done. The computer only recognises certain commands, and PINT is not one of them. So it tells you that you have made a mistake.

If you realise that you have made a mistake BEFORE you press the RETURN key, then you can correct it. If you press the DELETE key once, the last character you typed is deleted. If you press it again, another character is deleted. And so on.

- -

Type

```
PINT                    but don't press the RETURN key.

Press DELETE            and the T should disappear.

Press DELETE            and the N should disappear.

Press DELETE            and the I should disappear.
```

Now type

```
RINT 3+1                and press RETURN.
```

The command should be executed correctly. From now on, you won't be reminded to press RETURN after each command. This is something you will have to remember for yourself.

- -

Type

```
CALCULATE 3+1

DISPLAY 3+1
```

Although the English looks correct, the computer does not know the meaning of the words CALCULATE and DISPLAY, and so thinks you have made a mistake. The computer will only accept words it knows, and the one for displaying something on the screen is PRINT.

- -

The computer can also perform multiplication (times) and division (divides).

Type

```
PRINT 2*4

PRINT 5*5

PRINT 8/2

PRINT 9/3
```

The asterisk (*) tells the computer to do multiplication; the slash (/) tells the computer to do division. Check that the computer's answers are correct.

- -

This exercise shows the effect of inserting spaces in a command.

Type

```
PRINT2*3

PRINT 2*3

PRINT 2       *       3

PRIN T 2*3
```

All these are valid commands except the last one. Spaces can be inserted between words and numbers, but it is wrong to insert spaces between the letters making up a word: PRIN T is incorrect because of the space between the N and the T.

You should always put a space after a command word. For example:

```
PRINT 6/2     is clearer than

PRINT6/2
```

It is even more important when dealing with more complicated commands, which we meet in later units:

```
IF AGE < 11 THEN PRINT "CHILD"   is clearer than

IFAGE<11THENPRINT"CHILD"
```

The PRINT command tells the computer to display something on the screen

Questions

1. What is meant by the word input?
2. How do you provide 'input' to your computer?
3. What is meant by the word output?
4. Where does the 'output' from your computer appear?
5. What is a command?
6. You have met one command in this unit. What is it?
7. How do you give a command to the computer?
8. What is meant by executing a command?

9. What happens when you press the RETURN key after typing in a command?

10. What happens if you give the computer a command that is spelt incorrectly?

11. Work out what the computer will do with these commands. Write your answers in the 'your answer' column.

command	your answer	computer
PRINT 5+3		
PRINT 5-3		
PRINT 2*3		
PRINT 15/5		
PING 2+3		
PRINT 5 + 3		
CALCULATE 5+3		
PRI NT 5+3		
PRINT 4+3+2		
PRINT 4+3-2		

Now type the commands into the computer, and see if you were right.

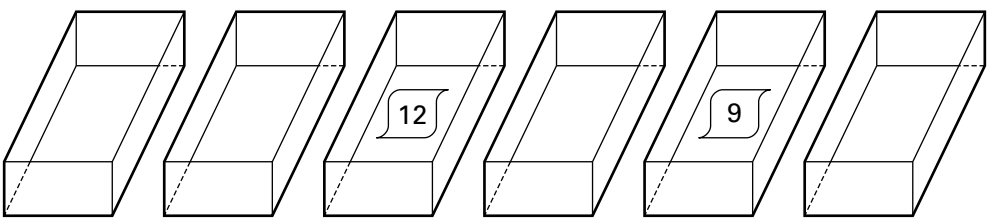
12. How do you correct typing mistakes? Make some deliberate mistakes, and then practise correcting them.

2 Variables

2.1 Computer memory

A computer has a memory. It stores information in its memory just as you store information in your memory. Information can be both recalled (got back from memory) and remembered (saved in memory for future recall). This unit describes how you can use BASIC to save information in, and recall information from, your computer's memory.

You can think of a computer's memory as being made up of a large number of boxes.



Each box can hold one piece of information. At this stage we will only consider numbers being stored in the boxes, but in later units we will see that other information (this book, for example) can be stored as well. In the diagram above, the third box from the left contains the number 12, and the fifth box from the left contains the number 9.

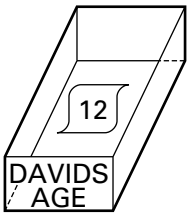
2.2 The LET command

Suppose you want to tell the computer about the age of a boy called DAVID. This can be done by typing the command

```
LET DAVIDSAGE=12
```

The computer will respond to the command as follows:

a) it selects ONE of its unused memory boxes;
b) it attaches the name DAVIDSAGE to this box;
c) finally it stores the number 12 in the box.



The boxes making up a computer's memory are 'electronic boxes' which cannot be seen with your eyes. So how can you be sure that the computer has created a box called DAVIDSAGE? Remembering that the PRINT command causes something to be displayed on the screen, try typing

```
LET DAVIDSAGE=12
PRINT DAVIDSAGE
```

and see what happens. Success! The computer will display the contents of the named box.

A name such as DAVIDSAGE is called a variable. It is said to be 'variable' because the contents of its box can be altered. For example, after David's next birthday, you might type

```
LET DAVIDSAGE=13
```

The box for DAVIDSAGE already exists, and it currently holds the number 12. This command simply stores a new number (13) in the box, and the previous number (12) is forgotten.

> The LET command puts a value into one of the boxes making up the computer's memory.

Type

```
LET MARTINSAGE=9
PRINT MARTINSAGE
```

Does the computer do as you expect?

- -

Type

```
LET MARTINSAGE=10
PRINT MARTINSAGE
```

The MARTINSAGE memory box should now contain the number 10.

- -

Type

```
PRINT ALISONSAGE
```

The computer should tell you that there is 'no such variable'. A memory box called ALISONSAGE can only be created by a LET command such as LET ALISONSAGE=5. Since you haven't typed such a command, the computer doesn't have a box called ALISONSAGE, and therefore it cannot display its contents.

- -

Type

```
PRINT DAVIDSAGE
NEW
PRINT DAVIDSAGE
```

The NEW command tells the computer to forget about all the names that have so far been attached to the boxes, and to start again. You can think of the computer rubbing off the names. Hence, following the NEW command, it doesn't know about DAVIDSAGE and MARTINSAGE, and when you try to use one of these names, the computer tells you that there is 'no such variable'.

> The NEW command rubs off any names that you have attached to the boxes making up the computer's memory.

Suppose David is paid pocket money by his parents. Each week he receives five pence for each year of his age. If he is 10, he gets 50 pence per week; if he is 11, he gets 55 pence; and so on.

Type

```
LET DAVIDSAGE=10
PRINT DAVIDSAGE
```

The LET command creates a box called DAVIDSAGE, and stores the number 10 in it.

Type

```
LET PAYFORDAVID=5*DAVIDSAGE
```

This command creates a box called PAYFORDAVID. The value assigned to PAYFORDAVID is calculated by multiplying 5 by the number in the DAVIDSAGE box. This number is 10. 5 times 10 makes 50, so the box called PAYFORDAVID should contain 50.

Type

```
PRINT PAYFORDAVID
```

and check that it contains 50.

Type

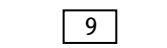
```
PRINT DAVIDSAGE
```

and check that it still contains 10. The value of a variable is not changed if it appears on the right-hand side of the = in a LET command (for example LET ... = 5*DAVIDSAGE). The computer simply finds out what value is in the box called DAVIDSAGE, and then uses that value in its calculations.

- -

Type

```
LET MARTINSAGE=9
PRINT MARTINSAGE
```



Type

```
LET MARTINSAGE=MARTINSAGE+1
```

Although this command may look rather odd at first, the computer processes it in a similar way to the last exercise. Firstly it gets the value held in MARTINSAGE, which is 9. Then it adds 1, giving 10. Finally it assigns the value 10 to MARTINSAGE.

Type

```
PRINT MARTINSAGE
```

and check that it is correct. Note that the previous value (9) has been lost.

- -

The numbers in the boxes below show the values that will be stored in the computer's memory after the following LET commands have been executed.

```
NEW
LET A=2          [ 2 ] A
LET B=3                  [ 3 ] B
LET C=A+B                        [ 5 ] C
LET C=C-1                        [ 4 ] C
```

Type these commands into the computer, and check (using PRINT commands) that the values in the computer's memory boxes agree with the values above.

- -

Again, the numbers in the boxes below show the values that will be stored in the computer's memory after the following LET commands have been executed. However, a deliberate mistake has been made. Can you spot it?

```
                 [ 2 ] A  [ 3 ] B  [ 4 ] C
LET A=A+4        [ 6 ] A
LET B=A-C                [ 2 ] B
LET C=B*B                        [ 9 ] C
LET A=A/2        [ 3 ] A
```

Type these commands into the computer, and see (using PRINT commands) whether the computer finds the same mistake as you did.

- -

2.3 Variable names

To finish this unit we look more closely at variable names.

```
Variable names can be any length.

They can contain: capital letters (A, B,... Z);
                  small letters (a, b,... Z);
                  numbers (0, 1,... 9);
                  the 'underline' character.

No other characters may be used. Because there is a
danger of confusing the 'underline' character with the
'minus', we will not use the 'underline' until much
later in this book.

A name must start with a letter.

Spaces are not allowed in the middle of a name.
```

The following are examples of correct names:

```
GALLONS
AD1984
AGE
```

The following are examples of names that are incorrect:

```
55BC                because it doesn't start with a letter.
PETER*SMITH         an asterisk is not allowed in a name.
GOALS IN MONTH      spaces are not allowed in a name.
```

You have to be careful on the BBC computer to make sure that variable names do not begin with BASIC "keywords" such as LET and PRINT. If they do, the computer gets confused.

Type

```
LET LETTER=9
LET PRINTER=4
```

LETTER is invalid because it starts with the letters LET. PRINTER is wrong because it starts with the letters PRINT. If you encounter a 'syntax error' such as this, and there is nothing obviously wrong with the line you have typed, then suspect the

variable name you have used, and try a different one.

One way round this problem is to use only small letters for variable names. This requires greater skill at typing, however, because you are continually using the SHIFT key. Until you become good at typing, you will find it easier to use capital letters for variable names, and avoid using names which begin with BASIC keywords. With a bit of practice this becomes straightforward.

Finally, you should always choose names which remind you what the variables are being used for (DAVIDSAGE, for example).

Questions

(See Page 108 for Additional Questions on this unit)

1. What is the name of the part of a computer that stores information?

2. What is a variable in BASIC?

3. Which are the variables in the following command?

```
LET CENTIMETRES = METRES * 100
```

4. Write a command which will set a variable called DAY to hold a value of 7.

5. What happens if you try to use a variable which has not yet been given a value?

6. What is the purpose of the NEW command?

7. Work out what values will be in the memory boxes after the computer has processed each of the following commands. Write your answers in the boxes provided.

	your answers	computer
LET A=6	☐ A	☐ A
LET B=A/2	☐ B	☐ B
LET C=A-B	☐ C	☐ C
LET C=3-C	☐ C	☐ C

Now type the commands into the computer, and check (using PRINT commands) that your answers are correct.

3 What is a program?

3.1 A simple program

Up to now, whenever you have typed a command at the keyboard the computer has executed it immediately. If you want the computer to do the same thing again you must re-enter the command. This can be boring. It would be nice if there was a way of saving commands somewhere, and then telling the computer to execute the commands you have saved. This unit describes how to do this.

A program is a sequence of commands. Each line of a program consists of a line-number followed by a command. For example:

```
1 LET AGE=10
2 PRINT AGE
3 LET PAY=5*AGE
4 PRINT PAY
5 END
```

The program is stored in the computer's memory. You can tell the computer to execute the commands of the program by typing RUN. In response, the computer will fetch from memory the command contained in Line 1 of the program, and execute it. Then the computer will fetch the command contained in Line 2, and execute that. This continues until it reaches the end of the program. As you can see, the line-numbers tell the computer the order in which the commands are to be executed.

How do we get a program into the computer's memory in the first place? Easy. Simply type the line-number, followed by the command, and repeat for each line of the program.

- -

Remembering the line-numbers, type

```
1 LET AGE=10
2 PRINT AGE
```

Notice that the computer simply accepts what you type, and then displays its prompt, waiting for another command. It does not execute the commands; if it did, you would see the number 10 appear on the screen, resulting from the PRINT AGE command.

```
Type
     3 LET PAY=5*AGE
     4 PRINT PAY
     5 END
```

The program should now be in the computer's memory.

- -

```
Type
     LIST
```

The LIST command causes the program stored in memory to be displayed on the screen. Check each line and make sure that the program is correct. If any line is wrong, then just re-type that line (not forgetting the line-number).

- -

```
Type
     RUN
```

The numbers 10 and 50 should be displayed on the screen. The 10 results from Line 2 (PRINT AGE) being executed. The 50 results from Line 4 (PRINT PAY) being executed.

A program is a sequence of commands.

The RUN command tells the computer to execute the program stored in its memory. This is called running the program.

The LIST command causes the program stored in the computer's memory to be displayed on the screen.

3.2 The TRACE command

The TRACE command tells the computer to display the line-number of a command on the screen just before it executes that command. Hence, you can see which lines the computer has executed.

- -

```
Type
     TRACE ON
     RUN
```

The computer should display:

```
<1>  <2>          10
<3>  <4>          50
<5>
```

The numbers between the < > pairs are the line-numbers. Line 1 is displayed first because it was executed first; then comes Line 2, followed by the display of 10 resulting from the PRINT AGE command; Line 3 is next; then Line 4, followed by the display of 50 resulting from the PRINT PAY command; finally Line 5 is executed.

- -

Type

```
TRACE OFF
```

This tells the computer to stop tracing. Try the RUN command again, and confirm that the computer is no longer printing line-numbers.

> The TRACE command tells the computer to print out the line-number of a command before it executes that command. Hence, you can see which lines the computer has executed.

3.3 The AUTO command

Type

```
NEW
LIST
```

In Unit 2 we saw that the NEW command tells the computer to forget about all variable names. In addition, it tells the computer to forget about the program in its memory. In fact, the NEW command clears memory completely. Hence, the LIST command has no effect, because there is now no program in memory.

Type

```
5 END
2 PRINT AGE
1 LET AGE=10
```

and list the program. Even though we typed the program in reverse order, the computer stores the program in its memory in line-number order.

- -

Type

```
     3 LET PAY=5*AGE
     4 PRINT PAY
     LIST
```

Again, the computer maintains the program in line-number order. Line 3 is inserted after Line 2, and Line 4 is inserted after Line 3. Check that the program runs as before.

In the last exercise we were able to insert two lines in the middle of the program, between Line 2 and Line 5, by giving them line-numbers of 3 and 4. We couldn't insert another line, however, because there aren't any more unused line-numbers. Instead of using line-numbers of 1, 2, 3,... etc., it is common to make the first line of a program line-number 10, the second line of the program line-number 20, and so on. In this way, there are nine unused line-numbers between any two lines, and so up to nine lines can be inserted, if necessary.

BBC BASIC provides the AUTO command to generate line-numbers automatically in steps of 10 when you are typing in a program. This very useful command makes entering programs simpler, because you need type only the commands.

- -

Type

```
     NEW
     AUTO

     10 should appear on the screen. Type  LET AGE=10
     20 should appear on the screen. Type  PRINT AGE
     30 should appear on the screen. Type  LET PAY=5*AGE
     40 should appear on the screen. Type  PRINT PAY
     50 should appear on the screen. Type  END
     60 should appear on the screen. Press ESCAPE
```

To terminate automatic line-numbering you must press the ESCAPE key.

- -

Type

```
     LIST
```

and check that your program is correct. If any line is incorrect, then simply re-type that line (not forgetting the line-number).

Type

```
TRACE ON
RUN
TRACE OFF
```

Notice that the computer executes the line with the smallest line-number (10) first, then the line with the next smallest line-number (20), and so on.

> The AUTO command generates line-numbers automatically when you are typing in a program.

Questions

(See Page 109 for Additional Questions on this unit)

1. What is a program?

2. How do you tell the computer to run a program?

3. Where is the program stored while it is being run?

4. What happens to the program in the computer's memory when you type NEW?

5. What is the purpose of a line-number? Why do you normally use line-numbers of 10, 20, 30... ?

6. What is the purpose of the TRACE command?

7. What command do you use to display a program on the screen?

8. When you are typing a program into your computer, what command will cause the computer to generate line-numbers automatically?

9* The distance that a car travels in a certain time can be calculated by multiplying its speed by the time. Using variable names of SPEED, HOURS and MILES, write a program to calculate the distance travelled in 4 hours when travelling at 60 miles per hour. The program should display the speed, the time and the distance on the screen, in this order. Type the program into the computer, list it and check that you have typed it correctly. Now run the program, and check that it produces the correct answer.

4 Input and output

4.1 More about the PRINT command

```
10 LET AGE=10
20 PRINT AGE
30 LET PAY=5*AGE
40 PRINT PAY
50 END
```

When it is run, this pocket money program displays two numbers on the screen. These numbers are 10 and 50. From our knowledge of the program, we know that 10 means '10 years of age', and 50 means '50 pence per week'. However, someone not familiar with the program is unlikely to know this. This section describes how we can make output easier to understand.

So far we have used the PRINT command in two ways:

```
PRINT 2+3    : the computer works out 2 add 3, and then displays
               the answer on the screen.
PRINT AGE    : the computer displays on the screen the number
               stored in the memory box called AGE.
```

Type

```
PRINT "HELLO"
```

The computer displays HELLO on the screen. We can get the computer to display any message on the screen by using a PRINT command of this form. The message is written between two quotation marks (".....").

What will be displayed by the command PRINT "AGE"?

Type

```
PRINT "AGE"
```

Were you correct? Now type

```
LET AGE=10
PRINT AGE
```

The command PRINT "AGE" tells the computer to display the word AGE. The command PRINT AGE tells the computer to display the contents of the box called AGE (10 in this case). It is important that you understand the difference between these two cases.

Using AUTO, type the following program into the computer:

```
10 PRINT "A"
20 PRINT "B";
30 PRINT "C"
40 PRINT "D"
50 END
```

Did you notice the semi-colon in Line 20? If not, retype the line. Now run the program. You should see the following output:

```
A
BC
D
```

The cursor is the name given to the marker that the computer displays on the screen to show you where the next character will appear. Cursors vary from computer to computer - on the BBC computer it is a flashing underline character.

PRINT "A"	tells the computer to display the letter A at the cursor position, and then to move the cursor to the beginning of the next line.
PRINT "B";	tells the computer to display the letter B at the cursor position. The ; at the end of the command tells the computer to leave the cursor on the same line, immediately following the B. The cursor is NOT moved to the beginning of the next line.
PRINT "C"	tells the computer to display the letter C at the cursor position. Hence, the C is displayed immediately after the B. The cursor is then moved to the beginning of the next line.
PRINT "D"	is the same as for A and C.

If there is no semi-colon at the end of a PRINT command, the cursor is moved to the beginning of the next line.

If there is a semi-colon at the end of a PRINT command, the cursor is left on the same line as the output.

- -
What output would you expect this program to produce?

```
10 PRINT "123";
20 PRINT "45"
30 PRINT "6";
40 PRINT "78"
50 END
```

Try it on your computer, and see if you were right.

- -
Enter into the computer the pocket money program shown at the start of this unit (don't forget to type NEW before you start).

Type

```
15 PRINT "AGE = ";
35 PRINT "POCKET MONEY = ";
LIST
```

You have inserted two extra lines into the program. Now run the program. The output should be:

```
AGE = 10
POCKET MONEY = 50
```

Is the output as you would expect? Notice that both Line 15 and Line 35 have a semi-colon at the end of the line.

- -
Type

```
15 PRINT "AGE = "
35 PRINT "POCKET MONEY = "
LIST
RUN
```

Note the effect of missing off the semi-colons.

- -
Up to now we have only used the PRINT command with one item, for example PRINT AGE. BASIC allows us to include a number of items in one PRINT command; the items are separated by semi-colons. When the command is executed by the computer, all the items will be displayed on one line.

Type

```
PRINT "ABC" ; "123" ; "DE" ; "45"
```

You should see ABC123DE45 displayed on the screen. The spaces around the semi-colons are not necessary - they have been included simply to make the command easier to read.

- -

Type

```
LET AGE=10
PRINT "AGE = ";AGE
```

and you should see AGE = 10 displayed on the screen. Notice that there are two items in the PRINT command ("AGE = " and AGE).

- -

Our pocket money program now becomes:

```
10 LET AGE=10
20 PRINT "AGE = ";AGE
30 LET PAY=5*AGE
40 PRINT "POCKET MONEY = ";PAY
50 END
```

Type this program into the computer, and see if it works correctly.

4.2 The INPUT command

Our pocket money program works well for a boy or girl who is aged 10. Suppose, however, that we want to work out the pocket money for David, who is 12 years old. As it stands, the program will not work. We must change Line 10 to LET AGE=12, and then run the program.

Type

```
10 LET AGE=12
LIST
RUN
```

and see if the results are correct.

If we now want to work out the pocket money for Martin, who is 9 years old, we will have to change the program yet again. This is becoming very boring. This difficulty can be overcome by using the INPUT command.

Type

```
10 INPUT AGE
LIST
RUN
```

When the computer executes the INPUT AGE command, it displays a question mark on the screen, and waits for you to type a value on the keyboard. This value is then stored in the memory box called AGE.

Type
```
     10        in response to the ?, and press the RETURN key.
```

The output should be the same as we obtained previously for a 10 year old.

Type
```
     RUN
and  5         in response to the ?, and press the RETURN key.
```

Has the computer produced the correct answer? We can now run the program as often as we like, and obtain the pocket money for whatever age we enter.

> The INPUT X command accepts a number typed at the keyboard and stores this number in the memory box called X.

When the computer executes the INPUT command in the pocket money program, and it displays the ? on the screen, we know that the computer is asking us to enter the child's age because we are familiar with the program. However, someone not familiar with the program is unlikely to know this. The problem is the same as we met when dealing with output (the numbers 10 and 50), and it can be overcome in exactly the same way.

Type
```
     5 PRINT "ENTER AGE ";
     LIST
     RUN
```

When the computer executes Line 5, it will display the message ENTER AGE on the screen. The semi-colon at the end causes the cursor to remain on the same line. Line 10 displays the ?, and then we can type a number (say 10). The program then continues exactly as before.

The message can be included in the INPUT command itself. For example

```
10 INPUT "ENTER AGE ",AGE
```

has the same effect as the two lines

```
 5 PRINT "ENTER AGE";
10 INPUT AGE
```

Making this change, our pocket money program now becomes:

```
10 INPUT "ENTER AGE ",AGE
20 LET PAY=5*AGE
30 PRINT "POCKET MONEY = ";PAY
40 END
```

Notice that the PRINT "AGE = ";AGE command is no longer needed, because we can see the age from the message produced by Line 10.

Questions

(See Page 109 for Additional Questions on this unit)

1. What new commands have you met in this unit?

2. What output will the following program produce?

```
10 PRINT "A"
20 PRINT "BB";
30 PRINT "CCC"
40 PRINT "DDDD"
50 END
```

3. Type in the program, run it and see if you were correct.

4. What output will the following program produce?

```
10 PRINT "A";"B";
20 PRINT "C";"D"
30 PRINT "E";"F"
40 END
```

5. Type in the program, run it and see if you were correct.

6. Write an INPUT command which has the same effect as:

```
PRINT "ENTER WEEKLY AMOUNT ";
INPUT WEEKLY
```

7. How can you stop the cursor moving to the next line after a PRINT command is executed?

8* In the present version of the pocket money program, the weekly amount for each year of the child's life is fixed at 5 pence per week. Modify the program so that this amount can be entered from the keyboard.

Hints:

a) you will need another INPUT command between Line 10 and Line 20, asking for the weekly amount. Use a variable called WEEKLY.
b) you will need to replace Line 20 by a line that calculates PAY by multiplying WEEKLY by AGE, rather than 5 by AGE.

Type the program into the computer, and then run it. When the program has run, the screen should look something like:

```
ENTER AGE ?10
ENTER WEEKLY AMOUNT ?6
POCKET MONEY = 60
```

9* Modify the program you produced for question 8 so that the words PENCE PER WEEK are added to the final line of output. An example line of output is:

```
POCKET MONEY = 60 PENCE PER WEEK
```

Hint : you will need to modify Line 30.

5 Looking after your programs

5.1 Backing store

A program is a sequence of commands. When the computer is given the RUN command, it will execute the program stored in its memory. Type the following program into your computer.

```
10 PRINT "I AM A CLEVER COMPUTER"
20 PRINT "GOOD BYE"
30 END
```

Type

```
LIST
RUN
```

Does the program work correctly? Now switch off your computer, and then switch it back on again.

Type

```
LIST
```

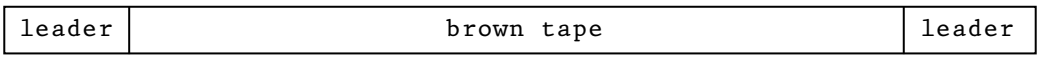
You should find that the program has disappeared. If you want to run this program again, you will have to re-type it all.

> A computer's memory is cleared when the computer is switched off, and any program it contains is lost.

It is boring to have to type in a program every time you want to use it. Therefore, nearly all computers provide a memory in which programs can be saved even when the computer is switched off. This memory is called a backing store. Cassette tapes are the most common form of 'backing store' on small computers. Larger, and more expensive computers, may use 'floppy disks'. This book deals only with cassette tapes.

5.2 Cassette tapes

Recording a program on tape is just like recording music on tape; you can play the cassette back later, and the music will still be there. If you were to unwind a cassette tape, you would find it similar to the following diagram:



Most of the tape is brown, and it is on this part of the cassette that programs are recorded. Both sides of the tape can be used, simply by turning the cassette over. The length of the tape depends on the capacity of the cassette. For example:

a C15 cassette provides 15 minutes recording (seven-and-a-half minutes on each side);

a C30 cassette provides 30 minutes recording (15 minutes on each side);

Fixed to either end of the brown tape there is a short plastic 'leader'. These leaders attach the tape to the two spools inside the cassette.

IT IS IMPORTANT TO REMEMBER THAT PROGRAMS CANNOT BE RECORDED ON THESE LEADERS.

- -

Many of the units in this book require you to save programs on a cassette. One C15 cassette will hold all the programs contained in this book, so get a C15 cassette.

- -

Every cassette should be given a name which reflects the purpose of the programs on that cassette. A suitable name for our cassette might be UNIT PROGRAMS, reflecting the fact that the programs on it are the programs associated with the units of this book. If your cassette has a label already, then write UNIT PROGRAMS on this label. If not, then get a sticky label, write UNIT PROGRAMS on it, and stick the label to the cassette.

- -

Insert the cassette into your recorder, with Side 1 uppermost. Rewind the tape, and set the tape counter to 000 (if your recorder has one). Press the PLAY button on the recorder, and let

the tape wind forward until the tape counter reads 003. Take the cassette out of the recorder. The 'leader' should have been wound onto the right-hand spool, and you should see just brown tape. If the leader is still visible, you will have to wind it a little further. Put the cassette back into the recorder. It is now ready for recording programs.

5.3 Program titles

You will need to give a name to each program that you save on tape. A maximum of 10 characters is allowed in the name if you are saving programs on cassette, and a maximum of 7 characters if you are saving programs on disk. This book uses names of 7 characters or less, so that the programs may be saved on either cassette or disk without having to alter the names used.

Choose sensible names such as:

"ALIENS" for an 'aliens' program.

"POCKET" for our 'pocket money' program. Note that the name has been shortened because there are 11 letters in the words 'pocket money', which exceeds the maximum we are using (7).

People modify programs after they have been written, and often keep several versions of the same program. If you use 6 characters or less for the program name, then there will be a spare character which can be used for the version number. For example, "POCKET1" for version 1 of the pocket money program, "POCKET2" for version 2, and so on.

It is good practice to start every program with a title. With its title, the pocket money program becomes:

```
10 REM  POCKET1
20
30 INPUT "ENTER AGE ",AGE
40 LET PAY=5*AGE
50 PRINT "POCKET MONEY = ";PAY
60 END
```

Line 10 is the 'title'. REM is short for REMARK. We use remarks in BASIC when we want to include some lines to make the purpose of the program clearer to ourselves. When the computer executes a REM command, it ignores everything else on that line. Hence, a REMark is purely for the reader's benefit. The title (POCKET1) explains the

purpose of the program, and is the name under which it is saved on cassette.

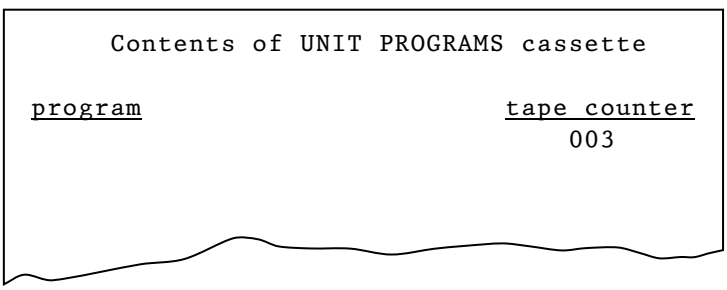
Line 20 is a blank line to separate the header from the rest of the program. This line is not essential, but it does make the program more readable. To insert a blank line into a program:

1. press the space bar once;
2. then press the RETURN key.

Pressing the RETURN key by itself will not produce a blank line.

5.4 Cassette contents sheet

You cannot see the programs recorded on a cassette. However, you can use a cassette contents sheet to help you to remember the programs that are stored on the cassette. Every time you save a program on the cassette, add its name, and the final tape counter reading, to the Contents Sheet.



Make a 'cassette contents' sheet for your UNIT PROGRAMS cassette, as shown above. This sheet should be kept with the cassette.

5.5 The SAVE command

The SAVE command saves on cassette the program that is currently stored in your computer's memory.

Type the pocket money program shown on Page 28 into the computer. Run it to make sure that it is working correctly.

1. Type SAVE "POCKET1"

   The message 'RECORD then RETURN' will be displayed on the screen.

2. Press the RECORD button on your recorder. The tape should start moving.

3. Pause for 2-3 seconds, to produce a short gap on the tape.

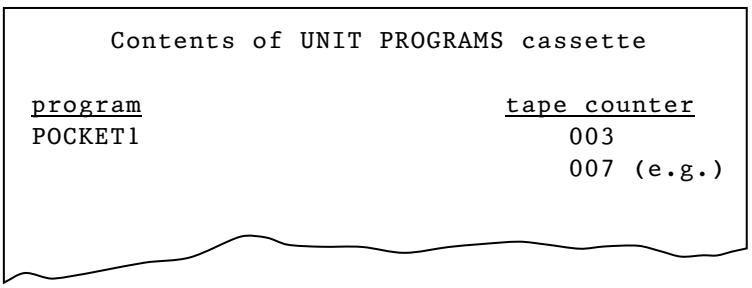
4. Press the RETURN key on the keyboard.

   The computer will record the program on the tape. When the computer has finished, the > prompt will appear on the screen. If your cassette has automatic motor control, then the tape will stop automatically. If it hasn't, you will have to stop the recorder manually.

5. Release the RECORD button on your recorder.

- -

Add the name of the program (POCKET1), and the final reading of the tape counter, to the 'cassette contents' sheet, as below:



003 is the position of the start of POCKET1
007 is the position of the end of POCKET1. It also becomes the position of the start of the next program that you save on the cassette.

- -

Type the following program into your computer:

```
10 REM  CLEVER1
20
30 PRINT "I AM A CLEVER COMPUTER"
40 PRINT "GOOD BYE"
50 END
```

Make sure that the program is working correctly. Now save it on the tape immediately after POCKET1, calling it CLEVER1. The steps are similar to those you used to save POCKET1, except that the SAVE command is SAVE "CLEVER1".

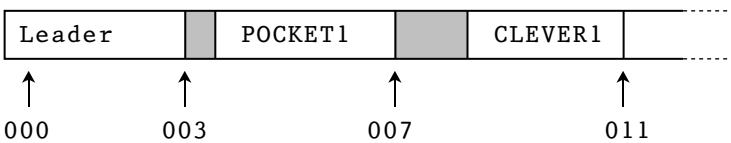
- -

Add the name of the program (CLEVER1), and the final reading of the tape counter, to the 'cassette contents' sheet, as below:

Contents of UNIT PROGRAMS cassette

program	tape counter
POCKET1	003
CLEVER1	007
	011 (e.g.)

The layout of the tape is now:



The shaded areas represent short gaps between the programs. These gaps are caused by the pause between pressing the RECORD button, and pressing the RETURN key, when saving programs.

5.6 The LOAD command

The LOAD command loads a program into the computer's memory from cassette.

1. Insert the UNIT PROGRAMS cassette into the recorder.

2. Type
   ```
   LOAD "POCKET1"
   ```
   The message SEARCHING will be displayed.

3. Rewind the tape.

4. Start the cassette playing by pressing the PLAY button on the recorder. Whenever the computer finds a program on the tape, it will display its name on the screen. If that program happens to be the one it is looking for, it will also display the message LOADING. When the loading is complete, the > prompt will reappear. If your cassette has automatic motor control, the tape will stop automatically; otherwise, you will have to stop the recorder manually.

5. Release the PLAY button on your cassette.

- -

The program is now in the computer's memory. List it, and check that it is correct. Now run the program. Does it perform correctly?

- -

With the UNIT PROGRAMS cassette as it was at the end of the last exercise, type

```
LOAD ""
```

and press the PLAY button on your recorder. The two quotes are typed one after the other with no space between. If you omit the program name in the LOAD command, and just type LOAD "", then the computer will load the next program on the tape, no matter what it is called. In this case, CLEVER1 should be loaded.

- -

List the program, and check that it is correct. Now run the program. Does it perform correctly?

5.7 Cataloguing a cassette

To find out what programs are saved on a cassette, rewind it to the beginning, type

```
*CAT                    (CAT stands for CATalogue)
```

and press the PLAY button on the recorder. The name of each program is displayed on the screen as it is encountered on the tape. No program is loaded into memory.

5.8 Backup

A cassette occasionally develops a fault which prevents one of its programs from being loaded into memory. When this occurs, the program is lost. Sometimes, several programs from one cassette may be lost. The solution to this problem is to keep a duplicate copy on a separate cassette. If your cassette becomes corrupted, you should still be able to retrieve the program from the second cassette. This process is called backup; the duplicate cassette is called the backup cassette. Each of your cassettes should have its own backup cassette. These should be kept in a safe place separate from the main cassettes, so that any mishap occurring to the main ones is unlikely to affect the backup cassettes as well.

You make a backup copy of a program by first loading it into memory from your main cassette (using the LOAD command), then saving it on your backup cassette (using the SAVE command).

5.9 The OLD command

The NEW command clears the computer's memory in readiness for you typing in your next program. What happens if you accidently type NEW before you have saved your present program? With many computers, the program will be lost, and you will have to retype it all. The BBC computer provide a safety net. If you type OLD before you have entered any lines of the new program, then the previous program will be restored to memory.

The REM command is ignored by the computer. Use it when you want to include explanatory text in a program.

The *CAT command displays on screen a catalogue of the programs saved on a cassette.

The OLD command restores to memory a program previously lost by typing NEW (or by pressing the BREAK key), so long as you haven't started entering a new program.

Questions

1. What new commands have you met in this unit?

2. What happens to the program in the computer's memory when the computer is switched off?

3. What is a backing store?

4. What is the purpose of the program title?

5. What is the purpose of the SAVE command?

6. If a program is saved on cassette as GUESS3, what does the 3 mean?

7. Why should you keep a cassette contents sheet?

8. What will happen to your program if you try to record it on the plastic 'leader' at the start of a cassette?

9. What is the purpose of the LOAD command?

10. What is the effect of the command LOAD ""?

11. How can you obtain a catalogue of the programs on a cassette?

12. What is backup? Why is it necessary to backup your cassettes?

13. What is the effect of the OLD command?

6 Editing

As we have already seen, programs are almost always altered after they have been written. The process of altering a program is called editing. This unit describes how you edit programs.

6.1 Modifying whole lines

- -

Load the POCKET1 program from your cassette. List it and run, and make sure that it is working correctly. Suppose we now want to make the computer appear a little more polite.

Type

```
     30 INPUT "PLEASE ENTER YOUR AGE ",AGE
```

List and run the program. The previous Line 30 has been replaced by the line you have just typed in. ANY line can be replaced by typing a new line with the same line-number.

- -

Type

```
     55 PRINT
     56 PRINT "GOOD BYE"
     57 PRINT
```

List and run the program. A new line can be inserted into a program by choosing its line-number to be BETWEEN the line-numbers of the lines above and below the new line. Hence, Lines 55, 56 and 57 are inserted between Line 50 and Line 60. Notice that a PRINT command by itself (as in Line 55) simply produces a blank line on the screen when it is executed by the computer.

- -

Type

```
     RENUMBER
```

List and run the program. The program should run exactly as before. Note, however, that the line-numbers have been altered so that each line-number is 10 more than the previous one. You should always renumber the line-numbers after inserting lines,

because programs whose line-numbers increase in regular steps are easier to understand.

- -

Type

```
     RENUMBER 1,1
     LIST
     RENUMBER 100,100
     LIST
     RENUMBER 100,10
     LIST
```

Look at each listing carefully. You should be able to work out that the RENUMBER n,m command alters the line-numbers so that the first line-number is n, and each subsequent line-number is m more than the previous one.

- -

Type

```
     110
     LIST
```

Notice that Line 110 has been deleted. The simplest way to delete a single line is to type its line-number, and then press the RETURN key. Contrast this with inserting a blank line, where we type the line-number, a SPACE, and then press the RETURN key.

- -

Type

```
     DELETE 150,170
     LIST
```

A group of lines may be deleted by the DELETE command. Line 150, Line 170, and all the lines between 150 and 170 (in this example, only Line 160) are deleted.

- -

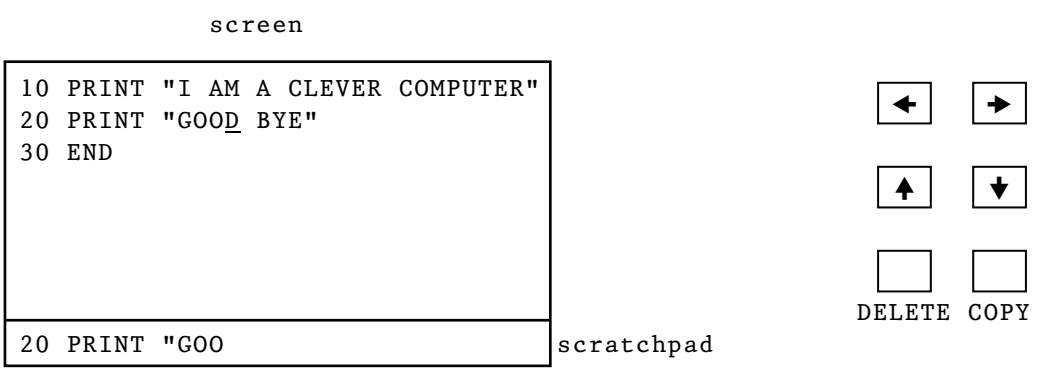
RENUMBER the program. It should now be:

```
     10 REM  POCKET1
     20 INPUT "PLEASE ENTER YOUR AGE ",AGE
     30 LET PAY=5*AGE
     40 PRINT "POCKET MONEY = ";PAY
     50 END
```

To replace a line:	type the new line with the same line-number as the line to be replaced.
To insert a line:	type the new line with a line-number BETWEEN the line-numbers of the lines above and below the new line.
To delete a line:	type the line-number by itself. To delete a group of lines, use DELETE x,y where x is the smallest line-number of the group, and y is the largest.
The RENUMBER command	causes the lines of a program to be renumbered. Steps of 10 are usual.

6.2 Modifying part of a line

When you find an error in a line of your program, you can correct it by retyping the whole line. The new line will replace the previous line if it is given the same line-number. The BBC computer provides a means by which you can correct PART of a line without having to retype the WHOLE line.



On the right-hand side of the keyboard of the BBC computer are six special keys. The keys with the arrows on them allow you to move the cursor to any point on the screen. As soon as you press one of these arrow keys, the computer enters 'edit mode', and a scratchpad appears on the screen. You construct your new line in the scratchpad. Any characters typed at the keyboard are

displayed in the scratchpad. In addition, you can move the cursor to any line on the screen, and copy characters from that line into the scratchpad by pressing the COPY key. The underline of the D in Line 20 indicates the cursor. Characters are deleted from the scratchpad in the normal way using the DELETE key. Pressing the RETURN key takes the computer out of 'edit mode', and the line in the scratchpad becomes the new program line.

- -

Use 'edit mode' on your computer to change the following lines of the pocket money program:

```
20 INPUT "PLEASE TYPE YOUR AGE ",AGE
40 PRINT "YOUR POCKET MONEY IS ";PAY;" PENCE PER WEEK"
```

List and run the program.

Questions

1. What is meant by editing?

2. How do you replace a line in a program?

3. How do you insert a new line into a program?

4. How do you delete a line from a program?

5. How do you delete a group of lines from a program?

6. What is the effect of the RENUMBER command?

7 REPEAT loops

A program is a sequence of commands. In all the programs we have met so far, the computer executes the first command, then it executes each of the following commands in turn, until it reaches the END of the program. If we want to use the pocket money program to find the pocket money of several children, then we have to RUN the program several times, and enter a different age on each run. What we would like to be able to do is to run the program ONCE, and during that run get the computer to execute a group of commands over and over again, until we tell it to stop. This can be written as:

```
repeat
     a group of commands
until told to stop
```

This can be translated into BASIC very easily on the BBC computer. The pocket money program becomes:

```
10 REPEAT
20   INPUT "ENTER AGE ",AGE
30   PAY=5*AGE
40   PRINT "POCKET MONEY = ";PAY
50 UNTIL told to stop
60 END
```

The REPEAT command in Line 10 tells the computer to execute over and over again all the commands between Line 10 and the UNTIL in Line 50. Hence, Lines 20, 30 and 40 form the group of commands to be repeated. How can we tell the computer to finish its repetition? As we are not interested in the pocket money of a child whose age is zero, we could use a zero value of age to tell the computer to finish repeating the group of commands. Hence, Line 50 becomes:

```
50 UNTIL AGE=0
```

Notice that we have typed two spaces at the start of Lines 20, 30 and 40. This is known as indentation. We indent the group of commands between the REPEAT and the UNTIL so as to make the program easier to read. if there are several REPEAT commands in a

program, and you are not using indentation, then it isn't always easy to see which UNTIL belongs to a particular REPEAT. With indentation it is obvious.

- -

Type

```
     10 REPEAT
     20   INPUT "ENTER AGE ",AGE
     30   PAY=5*AGE
     40   PRINT "POCKET MONEY = ";PAY
     50 UNTIL AGE=0
     60 END
```

List and run the program. Try it with values of 5, 12, 7 and 0. You should be asked to ENTER AGE four times, and then the program should stop running. Does it work correctly?

- -

Type

```
     45 ..PRINT   (replacing the two dots by two spaces).
```

List and run the program. Try it again with values of 5, 12, 7 and 0. It should produce the same results as before, but they should be displayed more clearly on the screen because of the blank line between each set of results (produced by the computer executing Line 45). You should always try to make your results as clear as possible.

- -

Type

```
     TRACE ON
     RUN
```

The computer should display:

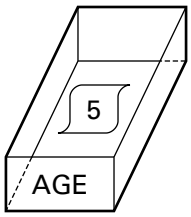
```
<10>  <20>  ENTER AGE ?
```

The computer executes Line 10, and then waits at Line 20 for you to enter an age.

Type

```
     5
```

The computer stores this value in the memory box called AGE.



The computer should now display:

```
<30>  <40>  POCKET MONEY = 25
<45>
<50>  <20>  ENTER AGE ?
```

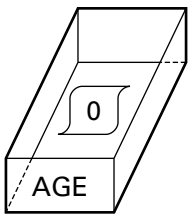
The computer executes Line 30, followed by Line 40, which causes POCKET MONEY = 25 to be displayed on the screen. Then it executes Line 45, producing a blank line on the screen.

Now the computer comes to Line 50 (UNTIL AGE=0). The computer gets the value contained in the memory box called AGE (5 in this example) and says 'is this equal to 0?'. Obviously it is not, so the computer goes back to the first line of the group of commands (Line 20) and asks you to enter the age again.

Type

```
     0
```

The computer stores this value of 0 in the memory box called AGE.



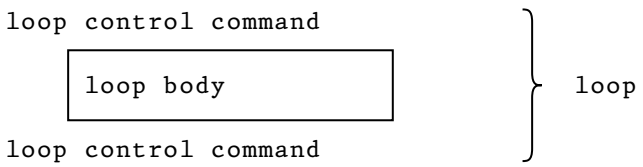
It now continues as it did above, displaying:

```
<30>  <40>  POCKET MONEY = 0
<45>
<50>  <60>
```

When the computer comes to Line 50, it gets the value contained in the AGE memory box (0 this time) and says 'is this equal to 0?'. It is. The computer has repeated the commands UNTIL AGE=0, and it is now time to stop the repetition. So the computer goes on to Line 60, which ends the program.

```
First  time through: it executes Lines   10 20 30 40 45 50
Second time through: it executes Lines      20 30 40 45 50
Other times through: it executes Lines      20 30 40 45 50
                     until it is told to stop.
```

The computer keeps returning to Line 20 and executing all the commands through to Line 50. This sort of structure is called a loop. The name 'loop' is used because the idea is similar to a loop in a piece of string; if you start at the top of a loop, and run your hand along the string, it will come back to the top of the loop again. Likewise, the path that the computer takes when executing the commands making up a program loop eventually leads back to the top of the loop. The group of commands being executed again and again is called the body of the loop. The commands controlling the number of times a loop is executed are known as loop control commands. Hence, a loop can be thought of as:



Notice that the loop 'body' is indented a few spaces so as to make it stand out. All loops have this general form. The loops described in this unit are called 'REPEAT loops' because they are controlled by the REPEAT command.

- -

Type

```
TRACE OFF
```

- -

Modify the pocket money program to become:

```
10 REM  POCKET2
20
30 REPEAT
40   INPUT "ENTER AGE ",AGE
50   PAY=5*AGE
60   PRINT "POCKET MONEY = ";PAY
70   PRINT
80 UNTIL AGE=0
90 END
```

List and run the program. Make sure that it is working correctly. Now save the program on your UNIT PROGRAMS cassette as POCKET2. Don't forget to fill in the 'cassette contents' sheet.

The REPEAT command marks the start of a group of commands that are to be executed over and over again.

The UNTIL command is paired with a REPEAT command, and marks the end of the group of commands being repeatedly executed. The commands are repeated until the condition specified immediately after the UNTIL is satisfied.

Questions

(See Page 110 for Additional Questions on this unit)

1. What is a loop? Why is it called a 'loop'?

2. What is the body of a loop?

3. What are loop control commands?

4. What is meant by the word indented? Why should the loop body be indented?

5* Suppose you have been shopping and have bought a number of items. The shop gives you a bill showing you how much you paid for each item, and how much you paid altogether. Write a program to check that the bill is correct. Written in ordinary English, the program should be something like:

```
set bill to 0
repeat
  input item
  add item to bill
until item=0
print bill
end
```

Translate this into a BASIC program, and use variable names ITEM and BILL. List the program to check that it is correct, and then run it with the following data:

```
the first  item cost  5 pence;
the second item cost 15 pence;
the third  item cost 11 pence.
```

Did you remember a value of 0 to finish the list?

When the program is working correctly, save it on your cassette as SHOP1. Don't forget the 'cassette contents' sheet.

8 Strings

In previous units we have used the PRINT command to display messages on the screen. For example

```
PRINT "POCKET MONEY = "
```

will display the message 'POCKET MONEY = ' on the screen. The set of characters between the quotes is called a string. A character is any of the symbols on the keys of the keyboard. This includes:

the capital letters	A, B, C,... Z
the small letters	a, b, c,... z
the numbers	1, 2, 3,... 9
a space	
the special characters	<, >, ?, $, %,...

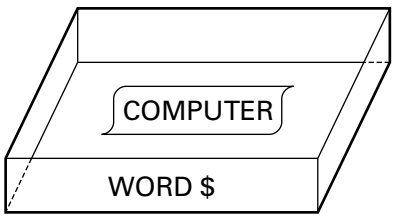
Note that a space is counted as a character. Hence, there are 15 characters in the string "POCKET MONEY = " (11 letters, 3 spaces and an =). Because the quote character(") is used to mark the beginning and the end of a string, there are difficulties in using it as a character within the string; at this stage you are advised to avoid doing so.

The command LET AGE=10 defines a memory box called AGE, and stores the number 10 in it. This memory box is really a 'number' memory box because it is used to hold numbers. Strings can be stored in memory just as we can store numbers in memory.

The command

```
LET WORD$="COMPUTER"
```

defines a 'string' memory box called WORD$, and stores the string "COMPUTER" in it.



The $ at the end of the variable name tells the computer that this is a string variable, and distinguishes it from the number

variable of the same name (i.e. WORD). The rules for naming string variables are the same as those for naming number variables; the name is chosen, and a $ is then added at the end.

- -

Type

```
10 INPUT "PLEASE TYPE YOUR NAME ",NAME$
20 PRINT "HELLO ";NAME$
30 PRINT
40 PRINT "I AM YOUR CLEVER BBC COMPUTER"
50 END
```

and run the program. Type your name when invited to do so. The characters you type are stored in the string variable called NAME$. The contents of NAME$ are then displayed on the screen by Line 20.

- -

Type

```
LET A$="12"
PRINT A$
```

A string can consist of just numbers (e.g. 1, 2, 3...) between quotes. Here we have stored the string "12" in A$. The string "12" is NOT the same as the number 12.

- -

Type

```
LET B="12"+3
```

The computer will report a 'type mismatch' error, because it cannot add numbers to strings. In fact, it cannot perform any arithmetic (adds, takes, times or divides) on strings.

- -

Type

```
LET A$=3
```

Again, the computer will report a 'type mismatch' error, because you are not allowed to assign a number to a string variable. Only strings can be assigned to string variables.

- -

You are now in a position where you can type English words into the computer, and it can display English words on the screen. Type the following program into your computer.

```
 10 REM  JOKES1
 20
 30 INPUT "PLEASE TYPE YOUR NAME ",NAME$
 40 CLS
 50 PRINT "HELLO ";NAME$
 60 PRINT
 70 PRINT "I AM YOUR CLEVER BBC COMPUTER"
 80 PRINT
 90 PRINT "*******************************"
100 PRINT "WHAT DID THE SEA SAY TO THE SAND";
110 INPUT PAUSE$
120 PRINT
130 PRINT
140 PRINT "NOTHING - IT JUST WAVED"
150 PRINT
160 PRINT
170 PRINT "BET YOU DIDN'T GET THAT ONE, ";NAME$
180 PRINT
190 PRINT
200 PRINT "*******************************"
210 PRINT "WHAT WOBBLES AND FLIES";
220 INPUT PAUSE$
230 PRINT
240 PRINT
250 PRINT "A JELLYCOPTER. HA HA"
260 PRINT
270 PRINT
280 PRINT "*******************************"
290 PRINT
300 PRINT "BYE FOR NOW"
310 END
```

- -

List the program and check that you have typed it correctly. Now run the program. Does it perform correctly?

`CLS` in Line 40 is a command which clears the screen (it stands for CLear Screen).

`INPUT PAUSE$` causes the computer to pause waiting for you to press any key. When you do so, it continues to the next command.

- -

When the program is working correctly, save it on your UNIT PROGRAMS cassette as JOKES1. Don't forget to fill in the 'cassette contents' sheet. You can obviously extend this program by putting in your own jokes.

- -
Type

    LIST

As the program contains more lines than can be displayed on one screen, the early lines of the program are lost off the top of the screen before you have time to read them. The computer is said to be in scroll mode.

- -
You can set the computer into paging mode so that it stops at the bottom of each page.

Type

    CTRL N   (hold down the CTRL key, then press the letter N)
    LIST

You now have chance to read the text on the screen.

- -
Press

    SHIFT

and the next page will be displayed.

- -
To set the computer back to 'scroll' mode:

Type

    CTRL O   (hold down the CTRL key, then press the letter O)
    LIST

and check that the program scrolls off the top of the page, as it did previously.

- -
Type

    LIST 100,200   to list all lines between 100 and 200

    LIST ,100   to list all lines up to and including 100.

    LIST 200,   to list all lines beyond 200

- -

Questions

(See Page 111 for Additional Questions on this unit)

1. What is a string?

2. How many characters are there in the string "XRY 382 T"?

3. How do you tell the computer that a variable is actually a string variable?

4. Which of the following are illegal as string variables? Give reasons.

   a) NUMBER b) NUMBER$ c) AD1984$
   c) 1984AD$ d) FIRST NAME$ e) FIRST-NAME$

5. Which of the following are incorrect BASIC commands?

   a) LET A$="10"
   b) LET A$=10
   c) LET A=10
   d) LET A="10"

6. What will the following program do when it is executed?

```
10 REPEAT
20   INPUT NAME$
30   PRINT "I LIKE ";NAME$
40   PRINT
50 UNTIL NAME$="STOP"
60 END
```

7. What command clears the screen when it is executed?

8. What command can you use to make the computer pause waiting for you to press a key?

9. What is meant by paging mode? How do you set the computer into paging mode?

10. What is meant by scroll mode? How do you set the computer into scroll mode?

11. What command would you use to:

    a) list all lines of a program up to Line 60;
    b) list all lines of a program after Line 100;
    c) list all lines of a program between Lines 50 and 90.

9 Graphics

9.1 Introduction

Many of the computers now cheaply available in shops are able to draw pictures on their screens. These pictures often involve several colours. The ability to draw pictures is called graphics. The BBC computer provides very many facilities for graphics, but it is beyond the scope of this book to deal with them all in detail. Indeed, whole books have been written just on graphics. The aim of this unit, therefore, is to provide you with a basic understanding of graphics, from which you can build a deeper knowledge.

Firstly, we should distinguish between 'text' and 'graphics'.

Text : Up to now, we have displayed only characters on the screen. These characters have been the same as those which appear on the keyboard. Characters such as these, appearing on the screen, are called text.

Graphics: There are special BASIC commands which instruct the computer to draw lines and shapes on the screen, in a variety of colours. These lines and shapes are referred to as graphics.

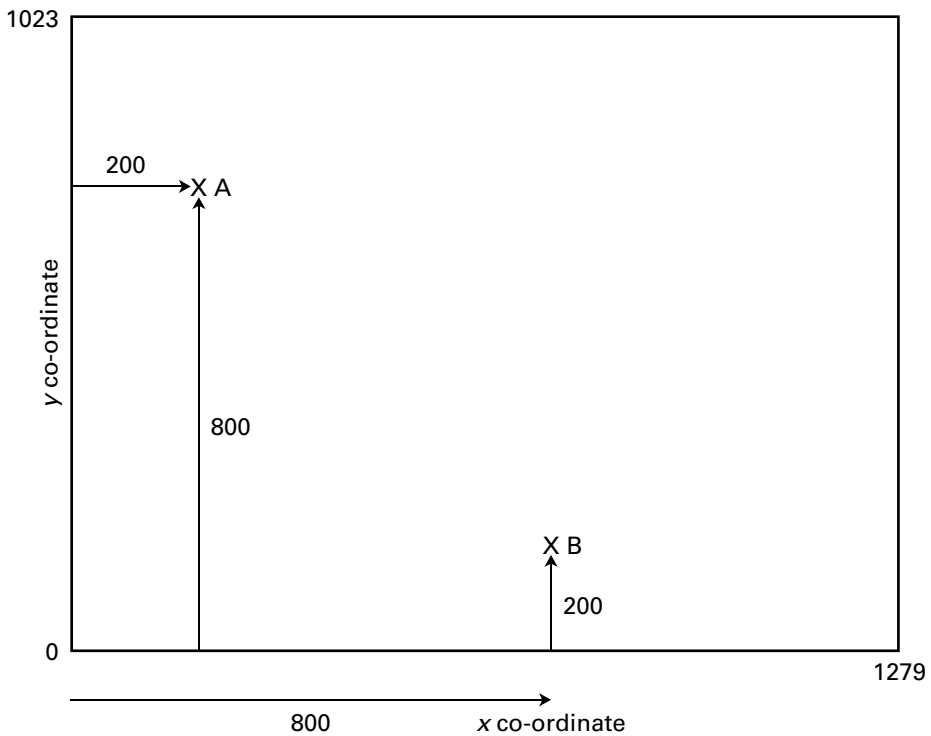
The BBC computer has EIGHT modes in which it can operate. These modes are numbered from 0 to 7, and are called Mode 0, Mode 1,... Mode 7. Only one mode can be used at a time. Each mode has different capabilities regarding 'text' and 'graphics'. The differences between the modes are explained in the User Guide which came with your BBC computer.

We will use Mode 1 for both the graphics in this unit and the colour in the next unit. Mode 1 is available on a Model B BBC computer, or a Model A with 32K of memory. If your computer is a standard Model A, then use Mode 5 instead (i.e. whenever you come across the command MODE 1, replace it by MODE 5).

9.2 Drawing lines

You can think of a computer display screen as consisting of a large number of dots. Each dot can be made to emit light. Hence, by carefully choosing which dots to light up, you can draw pictures on the screen.

BBC BASIC caters for 1280 dots across the screen, and 1024 dots up the screen. The horizontal dots are numbered 0, 1,..1279, and the vertical dots are numbered 0, 1,.. 1023. Numbering starts from the bottom left-hand corner of the screen, as shown in the following diagram:



The position of a particular dot on the screen is specified in terms of:

1. how far across the screen the dot is. For example, Dot A is at position 200 across the screen. This is often called the X co-ordinate of the point.

2. how far up the screen the dot is. For example, Dot A is at position 800 up the screen. This is often called the Y co-ordinate of the point.

Hence, Dot A has co-ordinates (200,800). Notice that the X coordinate is written before the Y co-ordinate.

What are the co-ordinates of Dot B in the diagram?

- -

Type

```
10 MODE 1           (MODE 5 on a Model A BBC computer)
20 MOVE 200,800
30 DRAW 900,100
40 END
```

and run the program. You should see a line on the screen, drawn from the top left-hand corner towards the bottom right-hand corner of the screen.

The MODE command switches the computer into the mode specified, Mode 1 in this case. When the computer is first switched on, it is in Mode 7.

The MOVE command moves the cursor to the position specified by the co-ordinates, in this case (200,800). This is, of course, our Dot A.

The DRAW command draws a line from the point specified, in this case Dot B at (900,100), to the last point visited (which is Dot A at (200,800) : we went there with the MOVE command).

- -

Type

```
35 DRAW 900,800
```

and run the program. You should see an extra line on the screen, drawn vertically upwards from point (900,100).

- -

Type

```
36 DRAW 200,800
```

and run the program. You should see an extra line on the screen, drawn horizontally from point (900,800) and completing the triangle.

- -

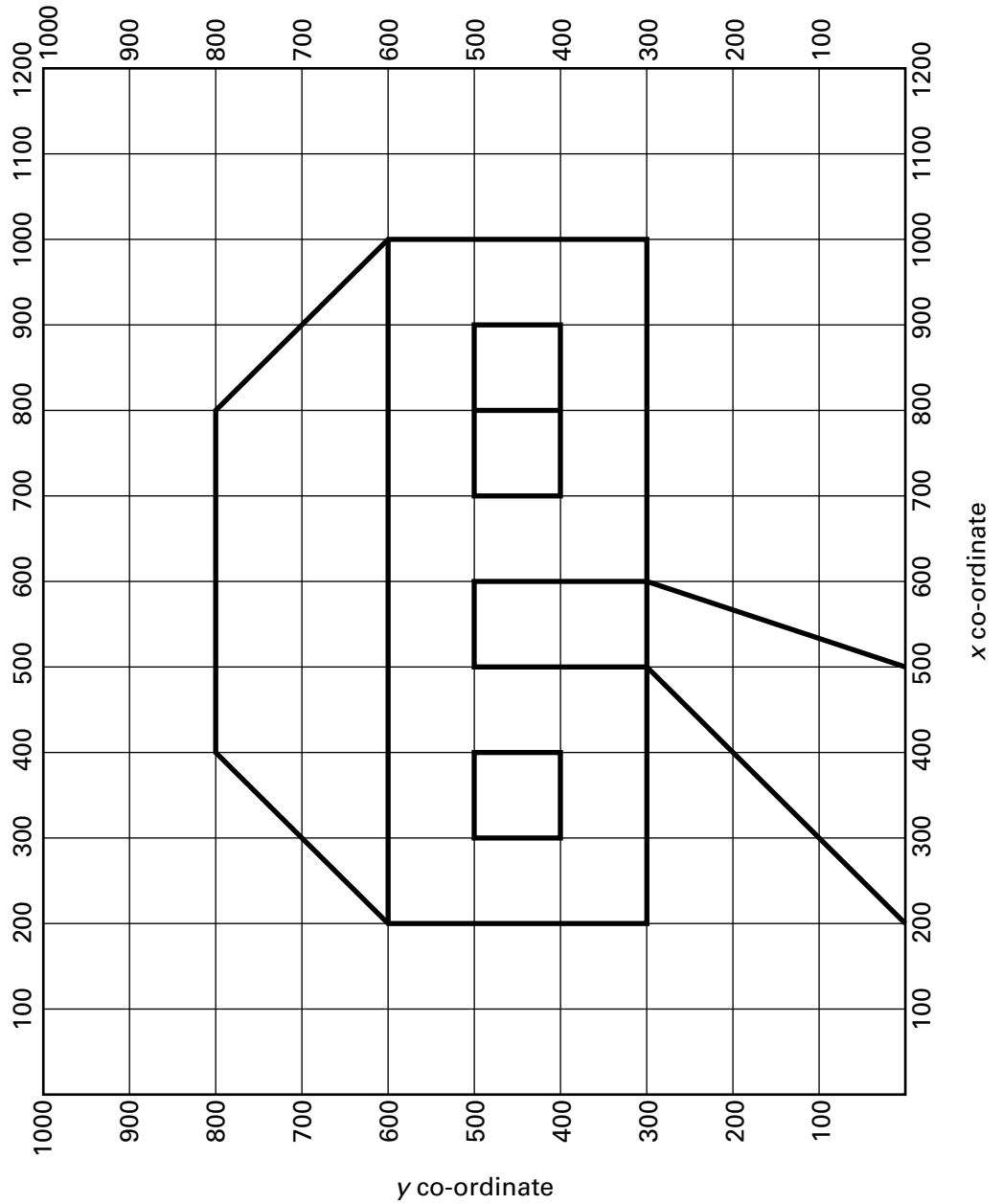


Figure 9.1 A grid representing the screen of the BBC computer.

9.3 Drawing shapes

So far we have seen how the computer can draw a single line using the MOVE command and the DRAW command together; more complex shapes can be drawn by adding further DRAW commands. Let us now get the computer to draw the house shown in Figure 9.1 opposite.

- -

Type

```
10 REM  HOUSE1
20 MODE 1
30
40 MOVE 200,300
50 DRAW 200,600
60 DRAW 1000,600
70 DRAW 1000,300
80 DRAW 200,300
```

and run the program. You should see on the screen a rectangle representing the front wall of the house. Look at the commands closely, and make sure that you understand how they work.

- -

Type

```
 90
100 MOVE 200,600
110 DRAW 400,800
120 DRAW 800,800
130 DRAW 1000,600
```

and run the program. The roof should now be added to the house. Notice that we have left a blank line between the group of commands which draws the front wall, and the group of commands which draws the roof. The blank line makes the program easier to follow when you make a mistake and have to go back to correct it.

- -

When the program is working correctly, save it on your UNIT PROGRAMS cassette as HOUSE1. Don't forget to fill in the 'cassette contents' sheet.

- -

Questions

1. What new commands have you met in this unit?

2. What is the difference between text and graphics?

3. How many modes does the BBC computer have?

4. What are co-ordinates?

5. Write a program to draw a line between point (300,100) and point (700,200). Test it on your computer. Don't forget the MODE command.

6. Write a program to draw a triangle with corners at points (300,100), (900,200) and (500,800). Test it on your computer.

7. Draw a grid similar to that shown in Figure 9.1 on a piece of squared paper. On this paper draw a rectangle with opposite corners at points (300,100) and (700,800). The edges of the rectangle are to be parallel to the X and Y axes.

8. Write a program to draw the rectangle you produced in answer to Question 7. Test it on your computer.

9* Load HOUSE1 from your UNIT PROGRAMS cassette. Extend the program to draw the remainder of the house, and the pathway, as shown in Figure 9.1. Leave a blank line between each group of commands. Test the program on your computer.

10. When the program is working correctly, change its title to HOUSE2, then save the program on your cassette as HOUSE2. Don't forget to fill in the 'cassette contents' sheet.

10 Colour

10.1 Text colour

Mode 1 (or Mode 5) has four colours - black, red, yellow and white. When you first switch to Mode 1, characters are printed in white on a black background. If the colours are displayed on a black-and-white screen, they will appear as shades of grey. Different colours are selected using the COLOUR command.

```
COLOUR 0  sets the foreground colour to BLACK.
COLOUR 1  sets the foreground colour to RED.
COLOUR 2  sets the foreground colour to YELLOW.
COLOUR 3  sets the foreground colour to WHITE.
```

- -

Load the HOUSE2 program from your UNIT PROGRAMS cassette. Run the program and check that it is working correctly.

- -

Type

```
MODE 1
COLOUR 1
LIST          red characters on a black background.

COLOUR 2
LIST          yellow characters on a black background.

COLOUR 3
LIST          white characters on a black background.
```

- -

Type

```
COLOUR 0
LIST
```

You should see nothing. COLOUR 0 selects a black foreground. As the background is also black, no characters will be visible.

```
Type
     COLOUR 129
     LIST
```

The characters will be black (selected in the previous exercise) on a red background. If the number in a COLOUR command is 128 or greater, then the COLOUR command sets the background colour.

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Type
     COLOUR 130
     LIST           black on a yellow background.

     COLOUR 131
     LIST           black on a white background.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Type
     CLS
```

The screen will be cleared, and left in the background colour.

```
COLOUR 128 sets the background colour to BLACK  (128=128+0)
COLOUR 129 sets the background colour to RED    (129=128+1)
COLOUR 130 sets the background colour to YELLOW (130=128+2)
COLOUR 131 sets the background colour to WHITE  (131=128+3)

Notice that the number used to select a background colour
is simply 128 more than the number used to select the same
colour in foreground.

CLS       clears the screen and leaves it in the current
          background colour. CLS stands for CLear Screen
```

10.2 Graphics colour

The COLOUR command changes the colour of the text foreground and background. It cannot be used for Graphics. There is a similar command for graphics, however, and this is the GCOL command. GCOL stands for Graphics COLour.

Type
```
     35 GCOL 0,1
```

and run the program. The lines drawn on the screen should be red.

- -

Type
```
     35 GCOL 0,2
```

and run the program. The lines drawn on the screen should be yellow. GCOL is followed by two numbers. The first is normally 0 (it is beyond the scope of this book to explain the purpose of the non-zero values of this first number). The second determines the graphics colour, using the same values as before (1=red, 2=yellow, etc).

- -

Type
```
     36 GCOL 0,129
     37 CLG
```

and run the program. A background colour can be set by the GCOL command in much the same way as with the COLOUR command. 129 in the GCOL command sets the background to red. Hence, in this example, you should see yellow lines on a red background. The command in Line 37 clears the screen, and leaves it in the background colour. CLG stands for CLear Graphics.

- -

Type
```
     CLG
```

The screen should be cleared, and left in the background colour.

GCOL 0,N	sets the Graphics COLour. N takes values of 0, 1, 2 and 3 for foreground colours, and values of 128, 129, 130 and 131 for background colours.
CLG	clears the screen of graphics and leaves it in the background colour. CLG stands for CLear Graphics

Questions

1. What new commands have you met in this unit?

2. What commands do you need to enter to list the HOUSE2 program in black on a yellow background? Try them on your computer (don't forget that Mode 1 is needed).

3* Modify your HOUSE2 program so that:

the background is white;
(use CLG after setting the background colour);

the house and roof are drawn in red;

the door and windows are drawn in yellow;

the garden path is drawn in black.

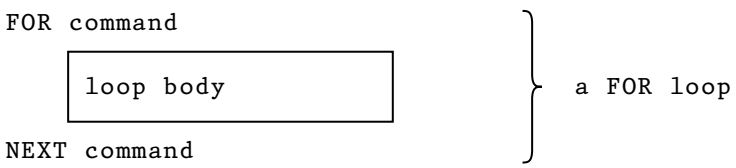
4. When your house program is working correctly, change its title to HOUSE3, then save it on your UNIT PROGRAMS cassette as HOUSE3. Don't forget to fill in the 'cassette contents' sheet.

11 FOR loops

11.1 The FOR and NEXT commands

We saw in Unit 7 that REPEAT loops are used when you want to execute a group of commands repeatedly until some condition is satisfied. For example, you can enter as many ages into the pocket money program as you like - it makes no difference whether it is one age, or ten ages, or fifty-seven ages. The computer keeps on executing the group of commands within the loop until you enter an age of zero, at which point the program finishes.

There are many instances when you want to execute a group of commands a fixed number of times. FOR loops are used in this situation. The structure of a FOR loop is similar to the structure of a REPEAT loop.



The commands between the FOR command and the NEXT command make up the body of the loop. Notice that the loop body is indented just as it is in a REPEAT loop.

- -

Type

```
10 FOR J=1 TO 5
20   PRINT "HELLO"
30 NEXT J
40 END
```

The FOR command in Line 10 marks the start of the loop; the NEXT command in Line 30 marks the end of the loop. The loop body in this example consists solely of Line 20, which is indented by two spaces.

- -

Now run the program. The word HELLO should be displayed on the screen 5 times.

The FOR J=1 TO 5 command tells the computer to:

```
set J to 1, and execute the body of the loop;
then set J to 2, and execute the body of the loop;
then set J to 3, and execute the body of the loop;
then set J to 4, and execute the body of the loop;
and finally, set J to 5, and execute the body of the loop.
```

Hence, the loop is executed five times, with J taking values of 1, 2, 3, 4 and 5.

- -

Modify Lines 10 and 30 to become:

```
10 FOR K=1 TO 5
30 NEXT K
```

and run the program again. The results should be the same.

The variable used in the FOR command controls how many times the loop is executed, and so is called the loop-control-variable. We can use any name for the loop-control-variable, so long as it is a valid variable name. Many programmers use a single letter for the name (such as J or K), and we will follow this convention in the rest of the book.

- -

Modify Line 20 to become:

```
20 ..PRINT "K = ";K
```

replacing the two dots by two spaces to preserve the correct indentation. Now run the program. The output should be:

```
K = 1
K = 2
K = 3
K = 4
K = 5
```

You can see that K takes successive values of 1, 2, 3, 4 and 5 during the five executions of the loop.

- -

Modify Line 10 to become:

```
10 FOR K=3 TO 7
```

and run the program. The starting value of K can be any value: it

does not have to be 1. In this example, the loop will be executed five times, with K values of 3, 4, 5, 6, and 7.

The NEXT command marks the end of a loop. It tells the computer to go back to the FOR command, and execute the loop again with the 'next' value of K. The loop finishes when it has been executed with K set to the value specified after the TO, which is 7 in this example.

- -

Modify your program to become:

```
10 FOR K=3 TO 7
20   PRINT "K = ";K
40 END
```

and run the program. If you omit the NEXT command, as we have done here, then the K will still be set to 3 by the FOR command, and the loop body will still be executed. However, as there is no NEXT command to send the computer back to Line 10, the program will finish at Line 40, and the loop will have been executed only once.

- -

Type

```
NEW
20   PRINT "HELLO"
30 NEXT K
40 END
```

and run the program. Here, we have missed out the FOR command. As there is no FOR command to match the NEXT command at Line 30, the computer doesn't know which line to go to, and so it tells you that you have made a mistake.

FOR and NEXT commands are always used in pairs.

The FOR command marks the start of a FOR loop.
The NEXT command marks the end of a FOR loop.

The FOR command takes the form FOR K=2 TO 5. This tells the computer to execute the loop with K values of 2, 3, 4 and 5. Hence, the loop will be executed five times.

Type

```
10 KSTART=3
20 FOR K=KSTART TO 7
30   PRINT "K = ";K
40 NEXT K
50 END
```

and run the program. A variable can be used to give the start-value; we have used KSTART here.

- -

Modify your program to become:

```
10 KSTART=3
15 KFINISH=7
20 FOR K=KSTART TO KFINISH
30   PRINT "K = ";K
40 NEXT K
50 END
```

and run the program. A variable can also be used to give the finish-value; we have used KFINISH here.

- -

Type

```
40 NEXT KSTART
```

and run the program. The computer will display the error

```
Can't Match FOR at Line 40.
```

There is no loop with KSTART as its loop-control-variable, and so the computer can't find a FOR command to match the NEXT command.

- -

Modify your program to become:

```
10 FOR K=5 TO 3
20   PRINT "K = ";K
30 NEXT K
40 END
```

and run the program. The loop will be executed once, with K set to 5. Even though the start-value is greater than the final-value when the computer first encounters the FOR command, the loop will still be executed. It is only when the computer gets to the NEXT command that it compares the K. value with the final-value, and is able to terminate the loop.

A FOR loop is ALWAYS executed at least once.

11.2 Using a STEP in the FOR command

In all the examples we have used so far, the computer increases the value of the loop-control-variable by 1 each time it goes round the loop. We can extend the FOR command to include a STEP-value, so that each time the computer goes round the loop, it increases the loop-control-variable by this step-value.

- -
Modify your program to become:

```
10 FOR K=1 TO 5 STEP 2
20   PRINT "K = ";K
30 NEXT K
40 END
```

and run the program. The loop will be executed with K values of 1, 3, and 5. In other words, K increases in steps of 2.

- -
Change Line 10 to:

```
10 FOR K=1 TO 5 STEP 1
```

and run the program. The loop will be executed with K values of 1, 2, 3, 4, and 5. This is exactly the same as a FOR command of FOR K=1 TO 5. Hence, if you omit STEP from the FOR command, a step of 1 is assumed.

- -
Change Line 10 to:

```
10 FOR K=9 TO 5 STEP -2
```

and run the program. Notice that minus STEP-values can be used. In this example, the loop will be executed three times, with K values of 9, 7, and 5.

- -

11.3 A 'tables' program

Suppose you want the computer to work out and display 'tables'. The output for the 5-times table might be:

```
 1 TIMES 5 = 5
 2 TIMES 5 = 10
 3 TIMES 5 = 15
     ....
10 TIMES 5 = 50
```

You want to be able to select which particular table is displayed (5-times, 3-times, and so on), but for any table you want ten lines of output.

Type

```
10 REM  TABLES1
20
30 INPUT "WHICH TABLE ",TABLE
40 FOR K=1 TO 10
50   LET ANSWER=K*TABLE
60   PRINT K;" TIMES ";TABLE;" = ";ANSWER
70 NEXT K
80 END
```

List and check the program, and then run it.

Type

```
5          when invited to enter a table.
```

The output should be:

```
 1 TIMES 5 = 5
 2 TIMES 5 = 10
 3 TIMES 5 = 15
     ....
10 TIMES 5 = 50
```

The loop is executed ten times, because it is controlled by the command FOR K=1 TO 10, and K takes values of 1, 2,.. 10. A line of output is displayed on the screen each time Line 60 is executed, and so there are ten lines of output.

- -

When the program is working correctly, save it on your UNIT PROGRAMS cassette as TABLES1. Don't forget to fill in the 'cassette contents' sheet.

Questions

(See Page 112 for Additional Questions on this unit)

1. What new commands have you met in this unit?

2. What output will the following program produce?

```
10 NUMBER=3
20 FOR K=1 TO NUMBER
30   PRINT K
40 NEXT K
50 END
```

Type in the program and then run it. Is the output as you expect?

3. What output will the following program produce?

```
10 KSTART=5
20 KFINISH=9
30 FOR K=KSTART TO KFINISH STEP 2
40   PRINT K
50 NEXT K
60 END
```

Type in the program and then run it. Is the output as you expect?

4. For each of the FOR commands below, write down in the boxes the value that K holds each time the loop is executed. The first has been done for you, to show you what to do:

FOR K=2 TO 3 the 1st time round the loop, K has a value of 2;
the 2nd time round the loop, K has a value of 3.

	value of K this time round the loop				
	1st	2nd	3rd	4th	5th
FOR K=2 TO 3	2	3			
FOR K=1 TO 7 STEP 2					
FOR K=1 TO 11 STEP 3					
FOR K=4 TO 2 STEP -1					
FOR K=8 TO 3 STEP -2					
FOR K=4 TO 2					

Check your answers on the computer.

5. Here is an alternative solution to the 'shopping' program described in Question 5 of Unit 7. Rewrite the program using proper indentation.

```
 10 REM  SHOP2
 20
 30 LET BILL=0
 40 INPUT "ENTER NUMBER OF ITEMS ",NUMBER
 50 PRINT
 60 FOR K=1 TO NUMBER
 70 PRINT "COST OF ITEM ";K;" = ";
 80 INPUT ITEM
 90 LET BILL=BILL+ITEM
100 NEXT K
110 PRINT
120 PRINT "TOTAL BILL = ";BILL
130 END
```

6. Type your indented version into the computer, and run it with the following data:

```
5

7
3
2
4
1
```

When the program is working correctly, save it on your UNIT PROGRAMS cassette as SHOP2. Don't forget to fill in the 'cassette contents' sheet.

12 READ and DATA

Up to now, we have entered data at the keyboard in response to an INPUT command, when the program is being executed. Another way of getting data into a program is to store it in DATA commands within the program itself, and to READ the items of data from these DATA commands.

Type

```
 10 READ A$
 20 PRINT "A$ CONTAINS ";A$
 30 READ B
 40 PRINT "B  CONTAINS ";B
 50 READ C$
 60 PRINT "C$ CONTAINS ";C$
 70 DATA "ALPHA"
 80 DATA 10
 90 DATA "BRAVO"
100 END
```

The items of data contained in the DATA commands are formed into a data list by the computer, in the same order as they appear in the program:

"ALPHA"	10	"BRAVO"

Every time the computer comes to a READ command, it takes the next item of data from this data list, and assigns it to the variable specified in the READ command. Run this program, and check that it works correctly (A$ should contain "ALPHA", B should contain 10, and C$ should contain "BRAVO").

- -

Type

```
 70 DATA "ALPHA", 10, "BRAVO"
```

and delete Line 80 and Line 90. Several data items may be placed in a single DATA command, so long as they are separated by commas. Run the program and check that it works.

- -

Type

```
     70 DATA "ALPHA", "BRAVO", 10
```

and run the program. The string "ALPHA" is assigned to A$, but when the computer tries to execute the READ B command in Line 30, it finds that the next item in the data list is the string "BRAVO". A string cannot be assigned to a 'number' memory box, so the computer will tell you that there is a 'type mismatch'.

- -

Type

```
     70 DATA "ALPHA", 10
```

and run the program. The computer should tell you that it is 'out of DATA' at Line 50, because you haven't provided sufficient data items (two data items for three READ commands).

- -

Type

```
     10 READ A$,B,C$
     70 DATA "ALPHA", 10, "BRAVO"
```

and delete Line 30 and Line 50. Several variable names, separated by commas, can be included in a single READ command. Run the program and check that it performs correctly.

- -

The following program asks you to enter the number of a month (1 is January, 2 is February,... 12 is December). The computer then tells you the name of that month. The program makes use of DATA commands for storing the names of the months.

Type

```
      NEW
      10 REM  MONTHS1
      20
      30 INPUT "MONTH NUMBER = ",M
      40 FOR K=1 TO M
      50   READ MONTH$
      60 NEXT K
      70 PRINT "MONTH ";M;" = ";MONTH$
      80
      90 DATA "JAN", "FEB", "MAR"
     100 DATA "APR", "MAY", "JUN"
     110 DATA "JUL", "AUG", "SEP"
     120 DATA "OCT", "NOV", "DEC"
     130 END
```

Check that you have typed the program correctly. Now run the program.

Type

```
1    when asked for the month number.
```

The computer should reply that

```
MONTH 1 = JAN
```

- -

Run the program again, and enter 12 when asked for the month number. Does the computer give the correct answer?

- -

Run the program again, and enter 13 when asked for the month number. The computer will attempt to execute the FOR loop thirteen times. As there are only 12 data items in the DATA list, the computer will report an 'out of DATA' error when it tries to execute Line 50 for the thirteenth time.

- -

When the program is working correctly, save it on your UNIT PROGRAMS cassette as MONTHS1. Don't forget to fill in the 'cassette contents' sheet.

Any number of DATA commands can be included in a program. They can be placed anywhere. It is good practice, however, to group them at the end of the program.

An item in the data list must be the same type (string or number) as the variable into which it is being read.

There must be sufficient items in the data list to match the number of READ commands executed in the program.

DATA commands are normally used when the data required by the program is the same each time the program is run. In the previous example it is unlikely that the names of the months will change!

Questions

(See Page 112 for Additional Questions on this unit)

1. What new commands have you met in this unit?

2. From where does the READ command get its data?

3. What errors can arise with READ and DATA commands?

4* Modify MONTHS1 so that it displays:

    the name of the requested month;
    the number of days in the requested month.

The number of days in the requested month is to be read from DATA commands into an extra variable called DAYS.

When the program is working correctly, save it on your UNIT PROGRAMS cassette as MONTHS2. Don't forget to fill in the 'cassette contents' sheet.

13 Numbers

13.1 Simple arithmetic

In Unit 1 we saw that the computer can perform simple arithmetic - adding, subtracting, multiplying and dividing.

Type

```
PRINT 3+1
PRINT 5-3
PRINT 2*5
PRINT 15/3
```

and check that each answer is correct.

- -

Type

```
PRINT 1+4*3
```

Arithmetic operations can be combined. In this example, the answer is 13. The computer performs the multiply before the addition. Hence

```
1 + 4 * 3     becomes
    ^^^^^
1 +  12       which becomes
^^^^^^^^^
   13
```

- -

Type

```
PRINT (1+4)*3
```

The answer is 15. By putting brackets around 1+4, we tell the computer to work this out first, giving an answer of 5. This 5 is then multiplied by the 3 to give an answer of 15:

```
(1 + 4) * 3    becomes
 ^^^^^
   5    * 3    which becomes
^^^^^^^^^^^
     15
```

- -

Type

```
PRINT 5+12/4
```

The answer is 8. The computer performs the division before the addition (12/4 = 3, then 5+3 = 8).

BASIC performs arithmetic in a certain priority order:

1. It works out whatever is contained in brackets. If there are brackets within brackets, it will do the innermost first.

2. It works out multiplication and division.

3. It works out addition and subtraction.

   For example:

```
2 * (1 + (6 - 2) * 2)    becomes
2 * (1 +    4    * 2)    which becomes
2 * (1 +        8)       which becomes
2 *        9             which becomes
      18
```

Type

```
PRINT 1+2*3
```

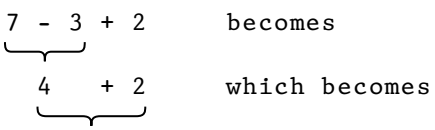
The answer is 7. Multiplication is 2nd in the priority order, whereas addition is 3rd, so the multiplication is done before the addition.

- -

Type

```
PRINT 7-3+2
```

The answer should be 6. Both addition and subtraction are 3rd in the priority order. When operations of the same priority occur in the same line, as here, the computer deals with them from left to right. Hence,



13.2 Decimal numbers

Up to now, we have only used whole numbers in our programs. The computer can also hold decimal numbers (i.e. numbers with a decimal point and a fractional part).

Type

```
PRINT 4/2
PRINT 5/2
PRINT 5/4
PRINT 5/3
```

The first answer is a whole number. All the other answers are decimal numbers.

- -

Type

```
LET X=4/2
PRINT X
LET X=4/3
PRINT X
```

A 'number' box in the computer's memory can hold both whole numbers and decimal numbers.

13.3 DIV and MOD

Type

```
PRINT 12345/100
```

The answer should be 123.45, which has a whole-number part and a decimal part.

Type

```
PRINT 12345 DIV 100
```

DIV performs a divide just as '/' does, but the answer that it produces is a whole-number. In this example the answer is 123, because 12345 divided by 100 goes 123 times, with a remainder of 45. This remainder can be calculated on the computer using MOD.

Type

```
PRINT 12345 MOD 100
```

The answer is 45, the remainder after dividing 12345 by 100.

13.4 Random numbers

Many games are based on chance (sometimes called luck). This 'chance' is often provided by throwing a dice. For example, the game of 'Snakes and Ladders' involves the players moving counters on a board according to the number of spots shown on the face of a dice. These games often require you to throw a 'six' before you can start. Sometimes you can throw a 'six' almost immediately, yet at other times your luck seems to be out, and you have to wait a long time for a 'six'. Many games available on computers also involve 'chance'. For example, the computer may produce enemy space ships for you to shoot down, and these space ships seem to appear at random, with no apparent pattern. In contrast, all the programs we have written so far produce the same answers on every run of the program, so long as we provide the same data. This section describes how we can tell the computer to produce what appears to be random behaviour.

- -

Type

```
10 REM  DICE1
20
30 FOR K=1 TO 10
40   LET X=RND(6)
50   PRINT X
60 NEXT K
```

and run the program. You should see ten numbers displayed on the screen, just like the numbers you might get by throwing a dice ten times. Line 40 generates a whole number at random, and stores it in memory box X. The 6 in Line 40 tells the computer that the number must be in the range 1-6 (that is, 1 or 2 or 3 or 4 or 5 or 6). To generate random numbers between 1 and 10 say, replace RND(6) by RND(10).

- -

Run the program again. The ten numbers now displayed will be different from the previous numbers. In fact, each time you run the program you will get a random set of numbers - you should not be able to PREDICT what numbers will be displayed. In this sense, the numbers are said to be random numbers.

- -

When the program is working correctly, save it on your UNIT PROGRAMS cassette as DICE1. Don't forget to fill in the 'cassette contents' sheet.

DIV	gives the whole number part of the result of a divide. For example, 7 DIV 2 is equal to 3.
MOD	gives the remainder after a divide. For example, 7 MOD 2 is equal to 1.
RND(K)	gives a whole number chosen at random from the numbers 1, 2, 3,... K.

Questions

(See Page 113 for Additional Questions on this unit)

1. What answers will the computer produce when it executes the following commands? Write down your answers below.

	Your answer	computer
PRINT 3+3*3		
PRINT 7-4+2		
PRINT 3*4/2		
PRINT (1+3)*3-4/2		
PRINT 9 DIV 4		
PRINT 9 MOD 4		

Try them on the computer, and check whether you were right.

2. What are random numbers?

3* We want to make the computer look as if it is tossing a coin twenty times. We can do this by writing a program which displays twenty 1's and 2's chosen at random, and assume that 1 means 'heads' and 2 means 'tails'. Write this program.

When the program is working correctly, save it on your cassette as COIN1. Don't forget the 'cassette contents' sheet.

14 Sound

14.1 The SOUND command

The BBC computer can generate sounds through its internal loudspeaker. There are two commands which control the sounds produced - these are SOUND and ENVELOPE. We will only deal with the SOUND command in this book. It has the form:

```
SOUND 1, A, P, D
```

A is the Amplitude (or loudness). This parameter controls the loudness of the sound, and can be varied between 0 (off) and -15 (loudest).

D is the Duration of the note (the length of time that the note is emitted). This is in twentieths of a second. Hence, if D is 10, the sound will last 10 twentieths of a second, that is, half a second.

P is the Pitch of the sound. This is a number which relates to musical notes as shown below:

P value :	53	61	69	73	81	89	97	101
Note :	Middle C	D	E	F	G	A	B	C

- -

Type

```
SOUND 1,-15, 53, 10
SOUND 1, -7, 53, 10
SOUND 1, -1, 53, 10
```

and compare the sounds produced for different amplitudes.

- -

Type

```
SOUND 1,-15, 53, 5
SOUND 1,-15, 53, 10
SOUND 1,-15, 53, 20
```

and compare the sounds produced for different durations.

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Type
     10 FOR K=1 TO 8
     20   READ P
     30   SOUND 1,-15, P, 10
     40 NEXT K
     50 DATA 53, 61, 69, 73, 81, 89, 97, 101
     60 END
```

and run the program. The musical scale will sound.

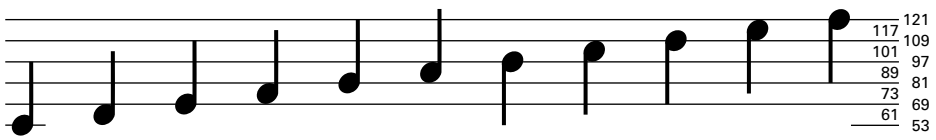
```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Type
     35 ..SOUND 1,0,0,3
```

replacing the two dots by two spaces. The musical scale will again sound, but there will be a distinct pause between each note, produced by Line 35 sounding a note of zero loudness and zero pitch for a very short duration.

14.2 Playing music on your BBC computer

Now that we can sound individual musical notes, we should be able to program the BBC computer to play music.

Music is writers on a special set of lines known as a stave.



note:	C	D	E	F	G	A	B	C	D	E	F
pitch:	53	61	69	73	81	89	97	101	109	117	121

The pitch of a note is indicated by its vertical position on the stave. The first note on the stave shown above is 'middle C'; its position is on the lowermost line. Successively higher notes (D, E,...) occupy lines higher up the stave. Notice that the names of notes go as far as G, and then start again at A. Two notes with the same name (C, for example) sound the same, but the second is at a higher pitch than the first; in music they are said to be an octave apart. The pitch values to produce each note on the BBC computer are written at the end of the stave, to make it easier for you to relate a note to its pitch value.

The duration of a note is expressed as a multiple of some

basic time unit (perhaps half a second), and is indicated by the symbol drawn for that note.


means the note lasts for 2 time units;


means the note lasts for 1 time unit;


means the note lasts for .5 time units;

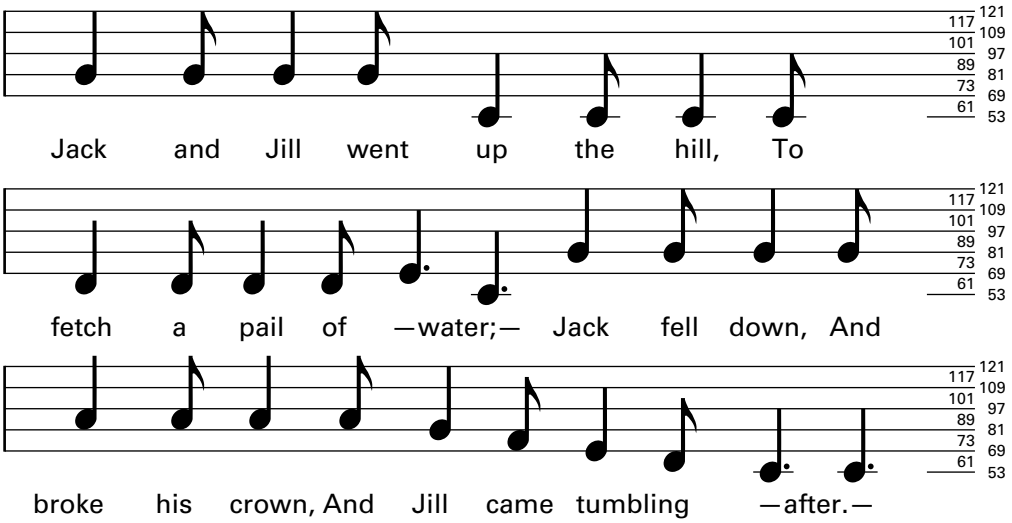

means the note lasts for .25 time units.

A dot following a note increases its duration by half its normal duration. Hence


lasts for 1.5 time units (1 + .5)


lasts for .375 time units (.25 + .125)

An example should help to clarify all this. The music for the Nursery Rhyme 'Jack and Jill' is:



The program shown opposite will play this tune. Each note of the tune has a corresponding DATA statement specifying the pitch and duration of that note.

The first note of the music is G. You can read off the pitch value for this note at the end of the stave. It is 81. The duration of the note is 1 time unit. This information is coded in Line 100 of the program as DATA 81,1.

The following program will play the Nursery Rhyme 'Jack and Jill' on the BBC computer.

```
 10 REM  MUSIC1
 20
 30 LET TIMEUNIT=10
 40 REPEAT
 50   READ PITCH,DURATION
 60   SOUND 1, -15, PITCH, DURATION*TIMEUNIT
 70   SOUND 1, 0, 0, 3
 80 UNTIL DURATION=0
 90
100 DATA 81,1
110 DATA 81,.5
120 DATA 81,1
130 DATA 81,.5
140 DATA 53,1
150 DATA 53,.5
160 DATA 53,1
170 DATA 53,.5
180
190 DATA 61,1
200 DATA 61,.5
210 DATA 61,1
220 DATA 61,.5
230 DATA 69,1.5
240 DATA 53,1.5
250 DATA 81,1
260 DATA 81,.5
270 DATA 81,1
280 DATA 81,.5
290
300 DATA 89,1
310 DATA 89,.5
320 DATA 89,1
330 DATA 89,.5
340 DATA 81,1
350 DATA 73,.5
360 DATA 69,1
370 DATA 61,.5
380 DATA 53,1.5
390 DATA 53,1.5
400
410 DATA 0,0
420 END
```

The second note of the music is also G, so its pitch value is 81 too. Its duration, however, is .5 time units. This is coded in Line 110 of the program as DATA 81,.5.

This process is repeated for all 28 notes of the music. A final DATA command of DATA 0,0 is included at Line 410 to tell the program that it has read all the data.

- -

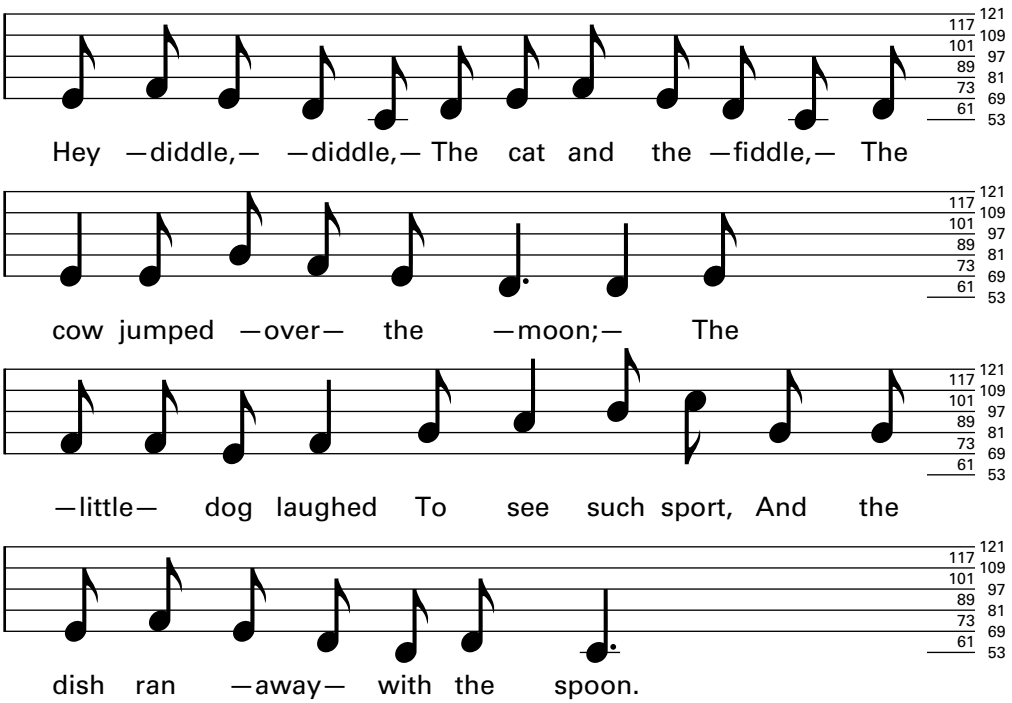
Enter the program into the computer, and check that you have typed it correctly. Run the program. Does it perform correctly?

- -

When the program is working correctly, save it on your UNIT PROGRAMS cassette as MUSIC1. Don't forget to fill in the 'cassette contents' sheet.

Question

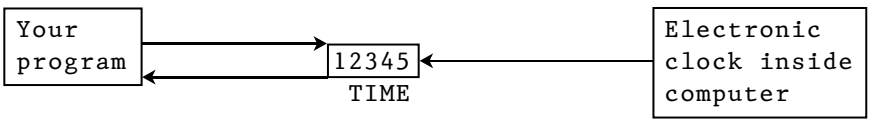
1. Write a program for the Nursery Rhyme 'Hey Diddle Diddle'.



When the program is working correctly, save it on your UNIT PROGRAMS cassette as MUSIC2. Don't forget to fill in the 'cassette contents' sheet.

15 Timing

15.1 Using the computer's clock



There is an electronic clock inside the BBC computer which your programs can use for timing. Every one-hundredth of a second, this clock will add 1 to a variable called TIME, and it continues to do so no matter what your program is doing. Hence, the TIME variable can be used to keep track of how much time has passed since some particular event occurred.

- -

Type

```
PRINT TIME
```

The number displayed on the screen shows the amount of time (in hundredths of a second) that has passed since the computer was first switched on.

- -

Type

```
LET T=TIME
```

The variable T now holds the value that was in TIME when this command was executed. This is the amount of time that has passed since the computer was first switched on.

Type

```
PRINT T
```

This amount of time is displayed on the screen, in hundredths of a second.

Type

```
PRINT T DIV 100, T MOD 100
```

The amount of time is now displayed in seconds and hundredths of

a second.

Type
```
PRINT TIME DIV 100, TIME MOD 100
```

Notice that the TIME variable has been increased by the electronic clock while you have been doing these exercises. The difference between T and TIME is the time that it has taken you to do the exercises.

- -

Type
```
10 TIME=0
20 REPEAT
30   LET T=TIME
40   PRINT T DIV 100, T MOD 100
50 UNTIL FALSE
60 END
```

and run the program. The value of the TIME variable (in seconds and hundredths of a second) is repeatedly displayed on the screen. The UNTIL FALSE means the loop executes for ever (or until you interrupt it). Let the program run for 10-20 seconds, and then press the ESCAPE key. Study the program, and try to understand how it works. Notice that you can give a value to the TIME variable: we set it to zero in Line 10.

Now look carefully at the output displayed on the screen, and notice that the second number (the hundredths) increases in steps of 2 (and occasionally 3). Why does it not increase in steps of 1, since the electronic clock adds 1 to TIME every one-hundredth of a second? The answer is that our program takes more than two hundredths of a second to execute Lines 40 and 50. Hence, when the computer gets back to Line 30, and samples the value in TIME again, it has increased by two or three hundredths of a second.

- -

If you have a stop-watch, then try the following experiment to see how accurate the clock within the computer is.

1. Type RUN but do not press the RETURN key.
2. Press the RETURN key, and, at the same time, start your watch.
3. Do not look at the screen, but keep a careful eye on your watch. When it gets to 50 seconds, press the ESCAPE key.
4. Look at the last time displayed on the screen, and see how close it is to 50 seconds.

15.2 Causing delays on the computer

The following program causes the computer to pause for a specified length of time.

```
10 LET DELAY=200
20 TIME=0
30 LET NOW=TIME
40 REPEAT UNTIL TIME>=NOW+DELAY
50 PRINT TIME
60 END
```

Line 10 sets the delay to 200 hundredths of a second, that is, 2 seconds.

Line 30 sets the variable NOW to the current value in TIME.

Line 40 causes the computer to wait until 2 seconds have passed. Notice that there are no commands in the REPEAT loop.

Enter the program, and then run it. The program should pause for two seconds before printing the current value in TIME. You will find that the value is actually 201, the extra one-hundredth of a second being used to execute the PRINT command itself.

- -

Type

```
10 LET DELAY=500
```

and run the program. Does it perform correctly?

- -

Type

```
10 FOR DELAY=1500 TO 7500 STEP 1500
20   TIME=0
30   FOR K=1 TO DELAY : NEXT K
40   PRINT DELAY; "   "; TIME
50 NEXT DELAY
60 END
```

and run the program. You can use a FOR loop to get the computer to pause for a period of time, as we have done in Line 30. Notice in Line 30 that we have put the FOR and the NEXT on the same line, separated by a colon; normally you should put them on separate lines, but in this particular example it makes the program clearer to have them on the same line. The length of the pause can be varied by changing the final value in the FOR loop.

The output should show that a value of 1500 in the delay loop causes a delay of about one second. If you wanted a delay of, say, 4 seconds, you would need a value of 6000.

15.2 Reaction-time program

This program uses the TIME variable to measure the time that it takes a person to react to some event.

```
 10 REM  REACT1
 20
 30 CLS
 40 LET ADDUP=0
 50
 60 FOR K=1 TO 10
 70   LET DELAY=200+RND(500)
 80   LET NOW=TIME
 90   REPEAT UNTIL TIME>NOW+DELAY
100   SOUND 1, -15, 53, 5
110
120   TIME=0
130   INPUT "" X$
140   LET T=TIME
150
160   LET ADDUP=ADDUP+T
170   PRINT T
180 NEXT K
190 PRINT
200 PRINT "AVERAGE = ";ADDUP/10;" HUNDREDTHS"
210 END
```

Type the program into the computer, and check that that it is correct. Now run the program, and make a note of your average reaction time. The following notes may help you to understand how this program works.

Line 30 clears the screen.

Lines 70-90 cause the computer to pause. The length of the pause is random between 200 and 700 hundredths of a second (i.e. between 2 and 7 seconds).

Line 100 sounds a note. As soon as you hear the noise, press the RETURN key as quickly as you can.

Lines 120-140 measure how long it takes you to press the key. This time is displayed by Line 170.

- -
Save the program on your UNIT PROGRAMS cassette as REACT1. Don't forget to fill in the 'cassette contents' sheet.

- -
Type

```
100 ..PRINT "PRESS";
```

replacing the two dots by two spaces. Instead of the bell being sounded, the word PRESS is displayed on the screen. As soon as you see the word on the screen, then press the RETURN key as quickly as you can. Run the program, and make a note of your average reaction time. You will probably find that this average is larger than the previous average - indicating that you react more quickly to signals from your ears than you do to signals from your eyes.

- -
Save the program on your UNIT PROGRAMS cassette as REACT2. Don't forget to fill in the 'cassette contents' sheet.

TIME	is used to read or set the computer's clock. This clock can be used by your program for timing.
:	You can put several commands on one line (as we did in Section 15.2) so long as the commands are separated by colons. Normally, you should put ONE command per line; occasionally, however, it makes your program clearer to put several on a line.

Questions

1. What does the electronic clock inside your computer do?

2. What will the command 'PRINT TIME' display on the screen?

3. What will the command 'TIME=0' do?

4. What command should you use to display the time in seconds and hundredths of a second?

16 The IF command

16.1 The IF command

Decisions are part of everyday life. For example:

```
     IF  you are hungry THEN  eat some food.

     IF  it is raining  THEN  put on a coat before going outside.

     IF  you are tired  THEN  go to bed.
```

These examples all have the same form:

```
     IF  some condition is true  THEN  take some action.
```

Taking the first example, the condition is 'you are hungry' and the action is 'eat some food'. If the condition is true, then you take the action.

A computer can also take decisions. Indeed, it is this capability which makes it into such a powerful and versatile piece of equipment. A computer can take different actions depending on the data that it is given. The command which tells the computer to do this is the IF command.

- -

Type

```
     10 FOR AGE=8 TO 11
     20   PRINT AGE;
     30   PRINT
     40 NEXT AGE
     50 END
```

Did you notice the semi-colon at the end of Line 20? Now run the program. The output should be:

```
      8
      9
     10
     11
```

- -

The local football club runs a junior team called the under-10's. Only boys who are under 10 years of age are eligible to play. Let's get the computer to tell us the ages that can play for this team.

Modify the program to become:

```
10 FOR AGE=8 TO 11
20   PRINT AGE;
25   IF AGE<10 THEN PRINT " IS OK";
30   PRINT
40 NEXT AGE
50 END
```

and run the program. The output should be:

```
 8 IS OK
 9 IS OK
10
11
```

The '<' in Line 25 means is less than. Hence, Line 25 really means

```
IF  AGE is less than 10  THEN print 'IS OK' alongside the
                              age printed out by Line 20.
```

When AGE has a value of 8, then the condition AGE<10 is true (8 is less than 10), and so the message 'IS OK' is printed.

When AGE has a value of 9, then the condition AGE<10 is also true (9 is less than 10), and so the message 'IS OK' is printed.

When AGE has a value of 10, then the condition AGE<10 is false (10 is not less than 10, it is actually equal to 10), and so the message 'IS OK' is not printed.

When AGE has a value of 11, then the condition AGE<10 is again false (11 is not less than 10, it is actually greater than 10), and so the message 'IS OK' is not printed.

- -

The local school is organising a class outing. However, only children who are over 8 years of age can go, because it would be too tiring for younger children. Let's get the computer to tell us the ages that can go on this outing.

Modify Line 25 to become:

```
25 ..IF AGE>8 THEN PRINT " IS OK";
```

replacing the two dots by two spaces. Now run the program. The output should be:

```
 8
 9 IS OK
10 IS OK
11 IS OK
```

The '>' in Line 25 means is greater than. Hence, Line 25 really means:

```
IF  AGE is greater than 8  THEN print 'IS OK' alongside the
                                age printed out by Line 20.
```

Obviously, ages of 9, 10, and 11 satisfy this condition, because they are greater than 8, but an age of 8 does not.

16.2 What conditions can be tested?

Up to now, we have encountered TWO conditions:

```
<   meaning   'is less than'       e.g. IF AGE<10
>   meaning   'is greater than'    e.g. IF AGE>8
```

There are FOUR other conditions that are commonly used in BASIC:

```
=   meaning   'is equal to'
<>  meaning   'is not equal to'

<=  meaning   'is less than or equal to'
>=  meaning   'is greater than or equal to'
```

- -

A children's competition has been organised by the local library. However, only 10-year olds can enter. Let's get the computer to tell us the ages that can enter this competition.

Modify Line 25 to become:

```
25 ..IF AGE=10 THEN PRINT " IS OK";
```

and run the program. The '=' in Line 25 means is equal to. Is the

output what you would expect? The message 'IS OK' should only be printed alongside an age of 10.

- -

For those children who were not eligible to enter the competition the library has organised a film show.

Can you work out what ages (from 8, 9, 10, and 11-year olds) can go to the film show?

Modify Line 25 to become:

```
25 ..IF AGE<>10 THEN PRINT " IS OK";
```

and run the program. The '<>' in Line 25 means is not equal to. If 10 is the only age that can enter the competition, then those children who are NOT 10 (i.e. 8, 9, and 11-year olds) can go to the film.

- -

A swimming club is entering a team for a swimming gala. One of the age groups is '9 and under'. This means that only children who are 9 or under 9 can be entered. Let's get the computer to tell us the ages that can be entered in this age group.

Modify Line 25 to become:

```
25 ..IF AGE<=9 THEN PRINT " IS OK";
```

and run the program. The '<=' in Line 25 means is less than or equal to. Hence, Line 25 selects all children who are '9 or under 9'.

- -

Only children who are '10 or over' are allowed to take the YOURTOWN cycling proficiency test. Let's get the computer to tell us the ages that can take this test.

Modify Line 25 to become:

```
25 ..IF AGE>=10 THEN PRINT " IS OK";
```

and run the program. The '>=' in Line 25 means is greater than or equal to. Hence, Line 25 selects all children who are '10 or over 10'.

```
<   means   'is less than'
>   means   'is greater than'

=   means   'is equal to'
<>  means   'is not equal to'

<=  means   'is less than or equal to'
>=  means   'is greater than or equal to'
```

16.3 A program to play 'guess the number'

The following program plays a game called GUESS-THE-NUMBER. The computer thinks of a number between 1 and 100, and you have to guess it. The computer gives you clues as to whether your guess is too large or too small.

```
 10 REM  GUESS1
 20
 30 CLS
 40 PRINT "THE COMPUTER WILL THINK OF"
 50 PRINT "A NUMBER BETWEEN 1 AND 100"
 60 PRINT
 70 PRINT "TRY TO GUESS IT"
 80 LET NUMBER=RND(100)
 90
100 REPEAT
110   PRINT
120   INPUT "ENTER GUESS ",GUESS
130   IF GUESS>NUMBER THEN PRINT "TOO LARGE"
140   IF GUESS<NUMBER THEN PRINT "TOO SMALL"
150 UNTIL GUESS=NUMBER
160 SOUND 1, -15, 53, 5
170 PRINT "WELL DONE"
180 END
```

The program is basically just a REPEAT loop (Lines 100-150). This loop is terminated when your GUESS is equal to the computer's NUMBER. The RND command in Line 80 generates a random number in the range 1..100. The SOUND command in Line 160 sounds a note when you have guessed the number correctly.

- -

Type in the program. List it, and then check that you have typed it correctly. Now run the program. Does it work correctly? If not, go back and re-check, correcting any mistakes.

- -

When the program is correct, save it on your cassette as GUESS1. Don't forget to fill in the 'cassette contents' sheet.

16.4 Modifying the pocket money program

You may remember that the pocket money program of Unit 7 prints the pocket money when an AGE of 0 is entered to finish the program. This is a bit untidy. Let us see how we can use the IF command to overcome this problem.

- -

Load POCKET2 from your UNIT PROGRAMS cassette. Run the program, and make sure that it is working correctly.

- -

Modify the program to become:

```
10 REM  POCKET3
20
30 REPEAT
40   INPUT "ENTER AGE ",AGE
45   IF AGE=0 THEN 80
50   LET PAY=5*AGE
60   PRINT "POCKET MONEY = ";PAY
70   PRINT
80 UNTIL AGE=0
90 END
```

We have inserted line 45, which shows the IF command being used differently. This version of the IF command has the form:

```
IF  some condition is true  THEN  transfer to a line-number
```

Hence, the IF command in Line 45 tells the computer to transfer to Line 80 if AGE is equal to 0. If it is not, then the computer continues to the next line, which is Line 50. Consequently, Lines 50, 60, and 70 will not be executed when we terminate the program by entering 0 for AGE, and so we have solved our problem.

- -

When the program is correct, save it on your cassette as POCKET3. Don't forget to fill in the 'cassette contents' sheet.

We have used two forms of the IF command:

```
IF  condition  THEN  command

IF  condition  THEN  line-number
```

Questions

(See Page 114 for Additional Questions on this unit)

1. What new commands have you met in this unit?

2. What character or characters are used in BASIC to mean:

   is less than?
   is greater than?

   is equal to?
   is not equal to?

   is less than or equal to?
   is greater than or equal to?

3. Which condition would you use to test for ages:

   under 10?

   equal to 8?

   9 or under?

   over 10?

   10 or over?

   not 9?

4. For each of the conditions shown below, put a tick against each value of AGE which makes the condition true. The first has been done to show you what to do.

condition	AGE				
AGE<7	✓ 5	✓ 6	☐ 7	☐ 8	☐ 9
AGE>8	☐ 5	☐ 6	☐ 7	☐ 8	☐ 9
AGE<>7	☐ 5	☐ 6	☐ 7	☐ 8	☐ 9
AGE>=7	☐ 5	☐ 6	☐ 7	☐ 8	☐ 9
AGE=8	☐ 5	☐ 6	☐ 7	☐ 8	☐ 9
AGE<=8	☐ 5	☐ 6	☐ 7	☐ 8	☐ 9
AGE>3	☐ 5	☐ 6	☐ 7	☐ 8	☐ 9
AGE<5	☐ 5	☐ 6	☐ 7	☐ 8	☐ 9

5* In the following fragment of a program, will Line 40 or Line 70 be executed after Line 30?

```
10 LET A=17
20 LET B=8
30 IF A+B>25 THEN 70
40 ...
```

6* In a match between two football teams, the team that plays on its own ground is called the HOME team, and the visitors are called the AWAY team.

If the HOME team wins the match, the result is a 'HOME win'.
If the AWAY team wins the match, the result is a 'AWAY win'.
If the scores are equal, the result is a 'DRAW'.

Write a program which will accept as input

a. the number of goals scored by the HOME team
b. the number of goals scored by the AWAY team

and then print the result of the match (HOME, AWAY or DRAW).

When the program is working correctly, save it on your cassette as FTBALL1. Don't forget the 'cassette contents' sheet.

17 Procedures

17.1 Introduction

In the reaction-time program developed in Unit 15, we included commands to cause the computer to delay for an amount of time chosen at random between two and seven seconds. The actual commands were:

```
LET DELAY=200+RND(500)
LET NOW=TIME
REPEAT UNTIL TIME>=NOW+DELAY
```

Delays could be required in a number of places in one program; they could also be needed in other programs. BBC BASIC allows you to put commands together into a group, and to attach a name to that group. Such a group is called a procedure.

The beginning of a procedure is marked by the DEFine PROCedure command (DEF PROCxxxx); xxxx is the name that we are attaching to this group of commands. You should choose a name which indicates the purpose of the commands. PAUSE might be a suitable name for a delay procedure, for example.

The end of a procedure is marked by the END PROCedure command (ENDPROC).

Hence, our delay procedure could be:

```
DEF PROC_PAUSE
LET DELAY=200+RND(500)
LET NOW=TIME
REPEAT UNTIL TIME>=NOW+DELAY
ENDPROC
```

To make the program easier to read, we have used the 'underline' character (which is on the same key as the 'pound') to separate the word PROC from the actual procedure name (PAUSE). In Mode 7, the 'underline' character looks like a 'minus' when displayed on the screen, so you have to be careful to use the right character.

17.2 Calling a procedure

Type

```
 10 TIME=0
 20 PROC_PAUSE                          Main program
 30 PRINT TIME
 40 END

 50

 60 DEF PROC_PAUSE
 70 LET DELAY=200+RND(500)
 80 LET NOW=TIME                        Procedure
 90 REPEAT UNTIL TIME>=NOW+DELAY
100 ENDPROC
```

and list the program. Check that you have typed it correctly.

Line 20 contains the procedure name by itself. This tells the computer to start executing the procedure, and is known as calling the procedure.

Lines 70-90 The computer transfers to Line 70, which is the first line of the procedure, and starts executing. It continues executing commands until it meets the ENDPROC command in Line 100.

Line 100 At this point it returns to the line immediately after the line which called the procedure, Line 30 in this case.

- -

Type

```
TRACE ON
RUN
```

and check that the commands are executed in the order that you expect. Line 60 (the DEF PROC... line) does not appear in the trace - the first line of the procedure is actually Line 70. Notice the pause in the trace at Line 90 while the computer executes the REPEAT loop.

- -

```
Type
     35 PROC_PAUSE
     36 PRINT TIME
```

and run the program again. Notice that the procedure is now called twice. Each time, the computer begins at Line 70, and then executes Line 80, 90, and 100. However, when the procedure is finished, the computer returns to Line 30 after the first call, and to Line 36 after the second call. Hence, the computer remembers the line-number of the line calling the procedure, so that it can return correctly.

17.3 Parameters

We have used RND on a number of occasions in previous units to generate random numbers. For example, RND(6) generates random numbers between 1 and 6; RND(10) generates random numbers between 1 and 10. The number within the brackets is called a parameter. It tells RND the range of numbers from which to select a random number. We can specify a different range each time we call the procedure, simply by changing the value of the parameter.

We can also use parameters with procedures. Modify the program to become:

```
10 TIME=0
20 PROC_PAUSE(200)
30 PRINT TIME
40 END
50
60 DEF PROC_PAUSE(DELAY)
70 NOW=TIME
80 REPEAT UNTIL TIME>=NOW+DELAY
90 ENDPROC
```

and run the program. The procedure should cause a delay of about 2 seconds. Lines 60-90 define a procedure called PROC_PAUSE. Notice that we have now written the name of a variable (DELAY) in brackets immediately after the name of the procedure.

The procedure is called in Line 20, with a value of 200 written in brackets following the procedure name. This number of 200 is the parameter; it is the value we want the procedure to work with on this occasion. Before executing the first command of the procedure, the variable (DELAY) is set to 200. The computer then executes the commands in the procedure, with DELAY having a value of 200.

- -

Modify Line 20 to become:

```
     20 PROC_PAUSE(500)
```

and run the program again. This time, the procedure is called with a parameter of 500. Before executing the first command of the procedure, the variable DELAY in the procedure is set to 500. The computer then executes the commands in the commands in the procedure, with DELAY having a value of 500. Hence, there should be a delay of about 5 seconds.

- -

Type

```
     20 PROC_PAUSE(200 + RND(500))
```

and run the program again. Notice that the parameter may involve arithmetic (we have used + and RND here).

- -

Type

```
     15 LET WAIT=500
     20 PROC_PAUSE(WAIT)
```

and run the program again. Notice that the parameter may be a variable (we have used WAIT here).

- -

The general form of a procedure is:

```
     DEF PROCprocedure-name (parameters)
     ...
     ...       the commands making up the procedure.
     ...
     ENDPROC
```

Values are passed to a procedure by means of parameters. A procedure may use more than one parameter.

A procedure is called by including its name in a line of the program.

17.4 Some 'graphics' procedures

One of the advantages with procedures is that you can use procedures written by someone else without having to understand the details of the commands within the procedures. So long as a procedure does what you are wanting, and you know its name and the parameters it expects, then you can use it.

In units 9 and 10 we developed a program to draw a house on the screen. This was a simple line drawing. In this unit we will use procedures to draw shapes such as rectangles, triangles and circles on the screen; these shapes will be completely filled in with a colour of our choosing. Hence, instead of representing the wall of the house as red lines on the screen (as we did in Unit 10), we will be able to colour the entire-wall red.

We can think of the screen as a sheet of PAPER, and of us drawing the shapes in INK. Hence, we need to be able to change the colour of both the PAPER and the INK.

Let us now look at the procedures provided. These procedures use features not described in this book, and so you won't be able to understand them in detail. That won't prevent you from using them, however.

`PROC_PAPER(colour)`	allows you to change the colour of the paper. You can use BLACK, RED, YELLOW or WHITE.
`PROC_INK(colour)`	allows you to change the colour of the ink. You can use BLACK, RED, YELLOW or WHITE.
`PROC_LINE(X1,Y1, X2,Y2)`	draws a line between (X1,Y1) and (X2,Y2) in the INK colour.
`PROC_RECT(X1,Y1, X2,Y2)`	draws a filled-in rectangle in the INK colour whose opposite corners are at (X1,Y1) and (X2,Y2). The sides of the rectangle are parallel to the X and Y axes.
`PROC_TRI(X1,Y1, X2,Y2, X3,Y3)`	draws a filled-in triangle in the INK colour with corners at (X1,Y1), (X2,Y2) and (X3,Y3).
`PROC_CIRC(XC,YC, radius)`	draws a filled-in circle in the INK colour, with centre at (XC,YC), and radius of 'radius'.

```
 10 REM  HOUSE4
 20
 30 MODE 1
 40 LET BLACK=0
 50 LET RED=1
 60 LET YELLOW=2
 70 LET WHITE=3
 80 GOTO 500
 90
100 DEF PROC_PAPER(colour)
110 GCOL 0,(128+colour)
120 CLG
130 ENDPROC
140
150 DEF PROC_INK(colour)
160 GCOL 0,colour
170 ENDPROC
180
190 DEF PROC_LINE(X1,Y1, X2,Y2)
200 MOVE X1,Y1
210 DRAW X2,Y2
220 ENDPROC
230
240 DEF PROC_RECT(X1,Y1, X2,Y2)
250 MOVE X1,Y1
260 MOVE X2,Y1
270 PLOT85,X1,Y2
280 PLOT85,X2,Y2
290 ENDPROC
300
310 DEF PROC_TRI(X1,Y1, X2,Y2, X3,Y3)
320 MOVE X1,Y1
330 MOVE X2,Y2
340 PLOT85,X3,Y3
350 ENDPROC
360
370 DEF PROC_CIRC(XC,YC, radius)
380 LOCAL X,Y,K,ANGLE
390 LET X=XC : LET Y=YC+radius
400 FOR K=5 TO 360 STEP 5
410 MOVE XC,YC
420 MOVE X,Y
430 LET ANGLE=K*PI/180
440 LET X=XC+radius*SIN(ANGLE)
450 LET Y=YC+radius*COS(ANGLE)
460 PLOT85,X,Y
470 NEXT K
480 ENDPROC
```

```
490
500 PROC_PAPER(WHITE)
510
520 PROC_INK(RED)
530 PROC_RECT(200,300, 1000,600)
540 PROC_TRI(200,600, 400,600, 400,800)
550 PROC_RECT(400,600, 800,800)
560 PROC_TRI(800,600, 800,800, 1000,600)
570
580 PROC_INK(YELLOW)
590 PROC_RECT(500,300, 600,500)
600
610 PROC_INK(YELLOW)
620 PROC_RECT(300,400, 400,500)
630 PROC_RECT(700,400, 900,500)
640
650 PROC_INK(BLACK)
660 PROC_TRI(200,0, 500,0, 500,300)
670 PROC_TRI(600,300, 500,0, 500,300)
680
690 PROC_INK(WHITE)
700 PROC_LINE(200,600, 1000,600)
```

- -

Type the HOUSE4 program into your computer. List it and check that you have typed it correctly. Now run the program. Is the picture drawn on the screen the same as Figure 9.1? The wall and roof should be red, the door and windows yellow, the path black, and the background white.

- -

When the program is working correctly, save it on your UNIT PROGRAMS cassette as HOUSE4. Don't forget to fill in the 'cassette contents' sheet.

- -

You can now experiment with this program. For instance, you could draw the sun in the top right-hand corner, using PROC_CIRC.

- -

Lines 10-510 of this program form the basis of other picture-drawing programs; you simply change lines from 520 onwards, using any combination of the procedure calls to draw the picture you want.

Delete Lines 520-700. Change the title in Line 10 to PROCS1. Now save the program on your UNIT PROGRAMS cassette as PROCS1. Don't forget to fill in the 'cassette contents' sheet.

- -

Questions

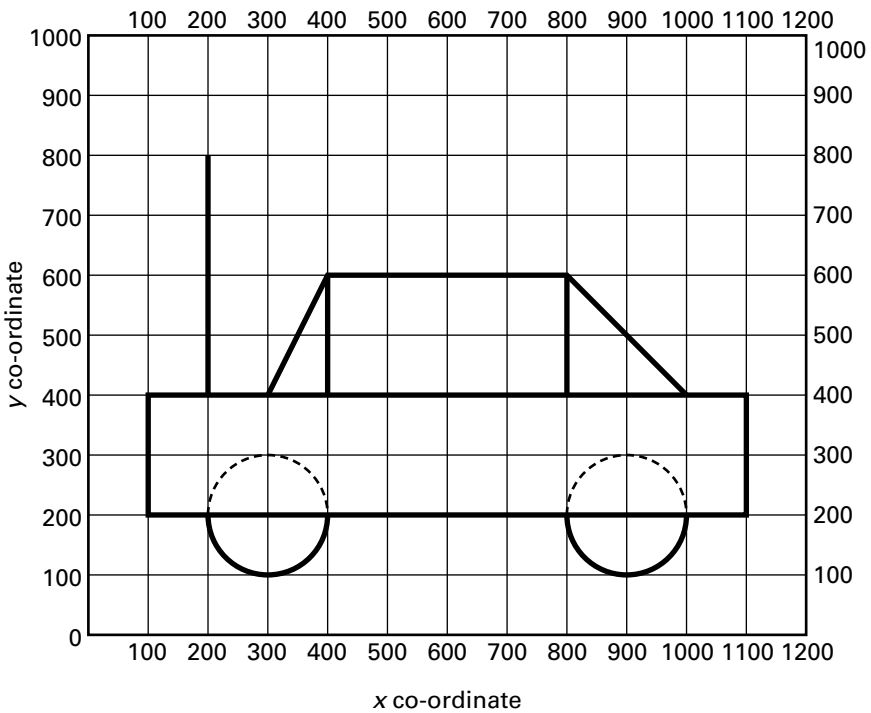
(See Page 115 for Additional Questions on this unit)

1. How do you define a procedure?

2. How do you call a procedure?

3. How are values passed to a procedure?

4. Load PROCS1 from your UNIT PROGRAMS cassette. Change the title in Line 10 to CAR1. Now extend the program from Line 520 onwards to draw the picture shown below. The car body is to be red, the windows yellow, the wheels and radio aerial black, with a white background.



5. When the program is working correctly, save it on your UNIT PROGRAMS cassette as CAR1. Don't forget to fill in the 'cassette contents' sheet.

Appendix A — Answers

Unit 3: Question 9

```
10 LET SPEED=60
20 PRINT SPEED
30 LET HOURS=4
40 PRINT HOURS
50 LET MILES=SPEED*HOURS
60 PRINT MILES
70 END
```

Unit 4: Questions 8 and 9

```
10 INPUT "ENTER AGE ",AGE
15 INPUT "ENTER WEEKLY AMOUNT ",WEEKLY
20 LET PAY=WEEKLY*AGE
30 PRINT "POCKET MONEY = ";PAY;" PENCE PER WEEK"
40 END
```

Unit 7: Question 5

```
 10 REM  SHOP1
 20
 30 LET BILL=0
 40 REPEAT
 50   INPUT "ENTER ITEM ",ITEM
 60   LET BILL=BILL+ITEM
 70 UNTIL ITEM=0
 80 PRINT
 90 PRINT "TOTAL BILL = ";BILL
100 END
```

Unit 9: Question 9

Add the following lines to HOUSE1:

```
140
150 MOVE 500,300
160 DRAW 500,500
170 DRAW 600,500
180 DRAW 600,300
190 DRAW 500,300
```

```
200
210 MOVE 300,400
220 DRAW 300,500
230 DRAW 400,500
240 DRAW 400,400
250 DRAW 300,400
260
270 MOVE 700,400
280 DRAW 700,500
290 DRAW 900,500
300 DRAW 900,400
310 DRAW 700,400
320
330 MOVE 500,300
340 DRAW 200,0
350
360 MOVE 600,300
370 DRAW 500,0
380 END
```

Unit 10: Question 3

Add the following lines to HOUSE2:

```
 35 GCOL 0,131
 36 CLG
 37 GCOL 0,1
145 GCOL 0,2
325 GCOL 0,0
```

Unit 12: Question 4

```
 10 REM  MONTHS2
 20
 30 INPUT "MONTH NUMBER = ",M
 40 FOR K=1 TO M
 50   READ MONTH$,DAYS
 60 NEXT K
 70 PRINT "MONTH ";M;" = ";MONTH$
 75 PRINT "IT HAS ";DAYS;" DAYS"
 80
 90 DATA "JAN", 31
100 DATA "FEB", 28
110 DATA "MAR", 31
... ....
200 DATA "DEC", 31
210 END
```

Unit 13: Question 3

```
10 REM  COIN1
20
30 FOR K=1 TO 20
40   LET X=RND(2)
50   PRINT X
60 NEXT K
70 END
```

Unit 16: Question 5

Line 40, because A+B is NOT greater than 25 - it is actually equal to 25. Hence, the condition is FALSE.

Unit 16: Question 6

```
 10 REM  FTBALL1
 20
 30 REPEAT
 40   INPUT "HOME TEAM GOALS = ",HOME
 50   IF HOME<0 THEN STOP
 60   INPUT "VISITORS GOALS = ",VISITORS
 70
 80   IF HOME>VISITORS THEN PRINT "HOME WIN"
 90   IF HOME<VISITORS THEN PRINT "AWAY WIN"
100   IF HOME=VISITORS THEN PRINT "DRAW"
110   PRINT
120 UNTIL FALSE
130 END
```

Notice the STOP command in Line 50. It has the same effect as the END command in that it stops the execution of the program. In addition, however, the STOP command displays a message on the screen showing the line-number at which the program stopped.

Appendix B — Summary of BASIC

AUTO — automatic. The computer automatically generates line-numbers when you are typing in a program. Pressing the ESCAPE key terminates the automatic numbering.

CLG — clear the graphics screen. The graphics area of the screen is cleared, and left in the current graphics background colour.

CLS — clear the text screen. The text area of the screen is cleared, and left in the current text background colour.

COLOUR — selects the foreground and background colour in which the computer will print text.

DATA — stores numeric and string data items within the program itself, for use by the READ command.

DEF — define. This command is used to define a procedure. For example, DEF PROC_PAPER.

DELETE — this command is used to delete a group of lines.

DIV — division of whole numbers. This gives the whole number part of the result of a division. For example, 7 DIV 2 is equal to 3.

DRAW — this command draws lines on the screen. It can be used in graphics Modes 0, 1, 2, 4, and 5.

END — tells the computer that it has reached the end of the program.

ENDPROC — marks the end of a procedure.

FOR — marks the start of a FOR... NEXT loop. FOR K=3 TO 6, for example, will execute a loop four times, with K values of 3, 4, 5 and 6.

GCOL graphics colour. This command is used to select the foreground and background colours in which the computer will perform graphics.

IF is part of the 'IF condition THEN...' command. It creates a test condition which controls what the computer will do next.

INPUT accepts information typed at the keyboard, and stores it in the computer's memory, for use by the program.

LET The LET command puts a value into a memory box, for example LET AGE=9.

LIST causes the program stored in the computer's memory to be displayed on the screen.

LOAD causes a program to be loaded from cassette into the computer's memory.

MOD modulus. This gives the remainder after division. For example, 7 MOD 2 is equal to 1.

MODE K sets the graphics mode to K, which may take values of 0, 1, 2, 3, 4, 5, 6 or 7.

MOVE this command moves the graphics cursor to some specified point on the screen.

NEW this command removes the program currently in the computer's memory.

NEXT marks the end of a FOR... NEXT loop.

OLD this command recovers a program after NEW has been entered, or the BREAK key pressed.

PLOT is a multi-purpose command for drawing points, lines or triangles on the screen.

PRINT displays strings and numbers on the screen.

PROC procedure. PROC attached to a variable name indicates a procedure. For example, PROC_INK(YELLOW).

READ this command tells the computer to read values from DATA commands, and assign these values to variables.

REM REMark. This command causes the computer to ignore the rest of a line. The REM command enables you to put comments into a program to explain how it works.

RENUMBER this command causes the lines of a program to be renumbered. Normally, line-numbering begins at 10, and increases in steps of 10.

REPEAT marks the start of a REPEAT... UNTIL loop.

RND(K) random. This function returns a number generated at random from the whole numbers 1, 2, 3,... K.

RUN causes the computer to execute the program in memory.

SAVE this command causes the program currently in memory to be saved on cassette.

SOUND causes the computer to emit sounds.

STEP is the part of the FOR command which specifies the size of the step to be added to the loop-control-variable each time the loop is executed.

STOP stops the execution of the program. It has the same effect as END, except that, in addition, a message is displayed on the screen showing the line-number at which the program stopped.

THEN is part of the 'IF condition THEN... command'. The commands on the THEN branch are executed if the condition is TRUE.

TIME is used to read or set the computer's electronic clock.

TO is the part of the FOR command which specifies the upper limit of the loop-control-variable. The loop will terminate when the loop-control-variable reaches or exceeds this upper limit.

TRACE makes the computer display the line-number of each line of the program just before it is executed.

UNTIL marks the end of a REPEAT... UNTIL loop

Appendix C — Additional Questions

This section contains additional questions for selected units of the book. These questions cover the fundamental ideas of BASIC and are, on the whole, not just applicable to the BBC computer. If you can answer all these questions correctly, you will have a good grasp of the fundamentals of BASIC, and should be able to apply these ideas to OTHER computers.

The question number tells you to which unit the question relates. For example, Questions 2A, 2B, 2C, 2D and 2E are five questions relating to Unit 2. Do not attempt to answer these questions until you have completed Unit 2. Likewise for the other questions.

2A Which of the following are invalid variable names in BBC BASIC?

(a) WEIGHT	(b) A2	(c) 2A
(d) R2D2	(e) Weight in grams	(f) w
(g) HOLE-IN-ONE	(h) W	(i) abcd
(j) 1234	(k) LETTER	

2B Which are the variables in each of the following commands?

(a) LET AGE=5	(b) LET A=B+1	(c) PRINT HEIGHT
(d) PRINT 5+2	(e) PRINT X+3	(f) PRINT 5*PRICE

2C Write a LET command:

(a) to put the number 5 into a memory box called A.
(b) to add 4 to 3, and store the result in memory box B.
(c) to subtract 2 from the number contained in memory box A, and store the result in memory box C.
(d) to add 1 to the contents of memory box C.
(e) to add the contents of memory box C to the contents of memory box A, and store the result in memory box D.

2D Write down the number that will be contained in each memory box after the LET commands of Question 2C have been executed.

A	B	C	D

2E Work out what values will be in the memory boxes after the computer has processed each of the following commands. Write your answers in the boxes provided.

(a)
```
LET A=3*3          A
LET B=A-3          B
LET C=B/2          C
LET D=A+1          D
```

(b)
```
LET M=2            M
LET M=M+3          M
LET N=2*M          N
LET M=N-2          M
```

3A What output will the following program produce?

```
10 LET GOALS=30
20 PRINT GOALS
30 LET GAMES=10
40 PRINT GAMES
50 PRINT GOALS/GAMES
60 END
```

3B The following program works out the cost of 10 apples, when each apple costs 5 pence. Complete the program by filling in the gaps.

```
10 LET NUMBER = .....
20 LET ..... = 5
30 LET BILL = NUMBER ..... PRICE
40 PRINT .....
50 END
```

3C The following program works out how many centimetres there are in 2 metres. Unfortunately, the commands have got mixed up. Put the commands into their correct order.

```
PRINT CENTIMETRES
LET CENTIMETRES=METRES*100
PRINT METRES
END
LET METRES=2
```

3D Correct the mistakes in the following program:

```
10 LET 5=AGE
20 PRINT AGE
30 PRINT PAY=10 x AGE
30 PRINT PAT
50 END
```

3E Write a program to work out how many grams there are in 3 kilograms, and then print the answer. Use variable names of GRAMS and KILOGRAMS. 1 kilogram = 1000 grams.

4A What output will the following commands produce?

```
(a) PRINT 2          (b) PRINT 2*3          (c) PRINT 4/2
(d) PRINT "A"        (e) PRINT "A";"B"      (f) PRINT "ANSWER = "
(g) PRINT "ANSWER = ";4*3          (h) PRINT "ANSWER = 4*3"
(i) PRINT "A";"B";"C"              (j) PRINT "A;B;C"
```

4B What output will the following programs produce?

(a)
```
10 PRINT "X"
20 PRINT "Y";
30 PRINT "Z"
40 END
```

(b)
```
10 PRINT "A";"B";
20 PRINT "C";
30 PRINT "DE"
40 END
```

4C The following program asks for two numbers to be entered via the keyboard, then it displays their sum. Fill in the gaps.

```
10 INPUT .....
20 ..... SECOND
30 LET SUM = ..... ..... SECOND
40 PRINT .....
50 END
```

4D What output will the following program produce?

```
10 LET SPEED = 50
20 LET HOURS = 2
30 PRINT "SPEED OF CAR = "; SPEED; " MILES PER HOUR"
40 PRINT "TRAVELLING TIME = "; HOURS; " HOURS"
50 PRINT "DISTANCE TRAVELLED = "; SPEED*HOURS; " MILES"
60 END
```

4E Modify the program shown in 4D to accept values of 'speed' and 'time' via the keyboard, as shown below:

```
ENTER SPEED ?50
ENTER HOURS ?2
```

The 50 and the 2 are typed by the user in response to the questions displayed on the screen by the computer.

4F When it is executed, the command PRINT "COST=";10;"POUNDS" will display the message COST=10POUNDS on the screen. Modify the command so that the message includes three extra spaces, and appears as COST = 10 POUNDS.

7A Rewrite the following program using proper indentation:

```
10 REPEAT
20 PRINT "STILL IN THE LOOP"
30 PRINT "ENTER 0 TO EXIT"
40 INPUT "ENTER A NUMBER ",NUMBER
50 UNTIL NUMBER=0
60 END
```

7B If values of 1, 2, and 3 are entered via the keyboard when the following program is run, what output will it produce?

```
10 PRINT "START NOW"
20 REPEAT
30   INPUT "ENTER NUMBER ",X
40   PRINT 2*X
50 UNTIL X=3
60 END
```

7C If you were to TRACE the execution of the program in 7B, what line-numbers would be displayed on the screen?

7D What output will the following program produce?

```
10 LET X=1
20 REPEAT
30   INPUT "ENTER NUMBER ",Y
40 UNTIL X=0
50 END
```

7E Write a program which will add together the ages of a group of children, entered via the keyboard. Use a REPEAT loop, and exit the loop when an age of 0 is entered. At the end of the program, display the total age of all the children.

8A How many characters are there in the following strings?

(a) "xyz" (b) "12345" (c) "Henry Smith"

8B Which of the following are illegal as string variables?

(a) A$ (b) R2D2 (c) 876$ (d) ADDRESS$

8C Which of the following are incorrect BASIC commands?

(a) LET X=10 (b) LET X$=10 (c) LET X="10"
(d) LET X$="10" (e) LET B=A$+3 (f) LET A$=B+3

8D Write a command which will accept a name typed at the keyboard, and store it in a variable called NAME$.

8E The following program is intended to display JOHN SMITH at the top of the screen. Fill in the gaps to complete the program.

```
10 CLS
20 FIRST$="JOHN"
30 .....="SMITH"
40 PRINT ..... ; .....; LAST$
```

8F Write a program that displays your name and address on the screen

11A Write a FOR command which will cause a loop to be executed with K values of:

(a) 1 2 3 4 5
(b) 3 4 5 6
(c) 10 20 30 40 50
(d) 7 6 5 4
(e) the odd numbers from 1 to 9
(f) the even numbers from 12 to 18
(g) the numbers starting at 10 and counting in 5's until 40

11B The following FOR loops are all incorrect. Can you find the mistake in each?

(a)
```
FOR K=1 TO 5
  PRINT K
NEXT J
```

(b)
```
LET K=1 TO 5
  PRINT K
NEXT K
```

(c)
```
FOR K=1,5
  PRINT K
NEXT J
```

(d)
```
FOR K$=1 TO 5
  PRINT K$
NEXT K$
```

11C What output will the following program produce if it is executed and a value of 3 is entered in reply to ENTER A NUMBER?

```
10 INPUT "ENTER A NUMBER ",NUMBER
20 ADDUP=0
30 FOR K=1 TO NUMBER
40   ADDUP=ADDUP+K
50 NEXT K
60 PRINT ADDUP
70 END
```

11D Write a program which uses a FOR loop to produce the following output:

```
1 x 1 = 1
2 x 2 = 4
3 x 3 = 9
4 x 4 = 16
```

12A What values will the variables contain after the following commands have been executed?

(a)
```
READ A, B, C$, D
DATA 10,20,"A",30
```

(b)
```
READ NAME$,TELEPHONE
DATA "JONES",143729
```

12B Can you find the errors in the following commands?

(a)
```
READ SCORE,TEAM$
DATA "TOTTENHAM",5
```

(b)
```
READ A$,B,C$
DATA "WXYZ",123
```

12C What output will the following program produce?

```
10 READ NUMBER                60 DATA 3
20 FOR K=1 TO NUMBER          70 DATA "DRILL",30
30   READ ITEM$,PRICE         80 DATA "SCREWDRIVER",5
40   PRINT ITEM$              90 DATA "HAMMER",7
50 NEXT K                    100 END
```

12D Make a list of names and ages of FOUR people, in the form:

```
DATA "MARTIN", 10
....
```

Write a program which uses a FOR loop to read this information into the computer, and display it on the screen.

13A How would you write the following in BASIC?

(a) 1+2 (b) 7-3 (c) 8÷2 (d) four times three
(e) 3+4+5 (f) 5x2-3 (g) 20÷2+4 (h) 2x3x4

13B 17 marbles are to be shared equally between 5 children. Using DIV and MOD, write PRINT commands which will display:

(a) how many marbles each child receives;
(b) how many marbles are left over.

13C Work out the values of the following:

(a) 3+3 (b) 5-2 (c) 4*2 (d) 8/4
(e) 1+2*3 (f) 9-6/3 (g) 7-2+3 (h) 3*4/2
(i) (1+2)*3 (j) (9-6)/3 (k) (7-2)+3 (l) 3*(4/2)
(m) 5 DIV 2 (n) 5 MOD 2 (o) 5/2

13D Fill in the boxes to show the values of the variables AFTER each line of the program has been executed.

```
10 READ A,B,C
20 LET D=A+B*C
30 LET B=(A+C)/B
40 DATA 4,3,2
50 END
```

A	B	C	D

13E Write a command which will set X to:

(a) a value selected at random between 1 and 5.
(b) a value selected at random between 6 and 10.
(c) a value selected at random from 10, 20, 30, 40 and 50.

13F What output will the following program produce?

```
 10 READ NUMBER
 20 RANDOM=RND(NUMBER)
 30 FOR K=1 TO RANDOM
 40   READ NAME$
 50 NEXT K
 60 PRINT "I HAVE CHOSEN ";NAME$
 70 DATA 7
 80 DATA "JOHN","ALAN","SUSAN","CHRISTINE"
 90 DATA "ANDREW","JONATHAN","ALISON"
100 END
```

16A Answer YES or NO to the following:

```
(a) Is 7 >  8 ?          (b) Is 7 <  8 ?
(c) Is 7 >= 8 ?          (d) Is 7 <= 8 ?
(e) Is 7 =  8 ?          (f) Is 7 <> 8 ?
(g) Is 4 <= 4 ?          (h) Is 4 >= 4 ?
(i) Is 4 =  4 ?          (j) Is 4 <> 4 ?
(k) Is 4 >  4 ?          (l) Is 4 <  4 ?
```

16B Using only < and >, complete the following statements:

```
(a) Is 4 .. 6 ? Yes      (b) Is 5 .. 8 ? No
(c) Is 8 .. 6 ? Yes      (d) Is 8 .. 3 ? No
(e) Is 5 .. 5 ? No
```

16C Using only <= and >=, complete the following statements:

```
(a) Is 4 .. 6 ? Yes      (b) Is 5 .. 8 ? No
(c) Is 8 .. 6 ? Yes      (d) Is 8 .. 3 ? No
(e) Is 5 .. 5 ? Yes
```

16D What output will the following program produce?

```
10 FOR K=1 TO 4
20   READ X
30   PRINT X;
40   IF X<10 THEN PRINT " is less than 10"
50   IF X=10 THEN PRINT " is equal to 10"
60   IF X>10 THEN PRINT " is greater than 10"
70 NEXT K
80 DATA 8,9,10,11
90 END
```

17A Using the procedures described on Page 98, write commands to:

(a) Set the paper colour to white.

(b) Set the ink colour to red.

(c) Draw a rectangle with opposite corners at (400,300) and (800,600).

(d) Set the ink colour to yellow.

(e) Draw a circle with centre at (600,700) and radius 100.

(f) Set the ink colour to black.

(g) Draw a triangle with corners at (200,400), (400,500) and 400,600).

(h) Draw a triangle with corners at (1000,400), (800,500) and (800,600).

(i) Draw a triangle with corners at (400,300), (500,100) and 600,300).

(j) Draw a triangle with corners at (600,300), (700,100) and (800,300).

(k) Draw a line from (500,400) to (700,400).

(l) Draw a line from (500,500) to (700,500).

Appendix D — Answers to Additional Questions

2A (c) doesn't start with a letter (e) contains spaces
(g) contains minus signs (j) doesn't start with a letter
(k) starts with a BASIC keyword (LET)

2B (a) AGE (b) A and B (c) HEIGHT (d) none (e) X (f) PRICE

2C (a) LET A=5 (b) LET B=4+3 (c) LET C=A-2 (d) LET C=C+1
(e) LET D=A+C

2D A=5 B=7 C=4 D=9

2E (a) A=9 B=6 C=3 D=10 (b) M=8 N=10

3A 30 10 3 on separate lines

3B Fill in the gaps with: 10 PRICE * BILL

3C
```
LET METRES=2
PRINT METRES
LET CENTIMETRES=METRES*100
PRINT CENTIMETRES
END
```

3D
```
Line 10       should be  10 LET AGE=5
Line 30       should be  30 LET PAY=10 * AGE
30 PRINT PAT  should be  40 PRINT PAY
```

3E
```
10 LET KILOGRAMS=3
20 PRINT KILOGRAMS
30 LET GRAMS=KILOGRAMS*1000
40 PRINT GRAMS
50 END
```

4A (a) 2 (b) 6 (c) 2 (d) A (e) AB (f) ANSWER =
(g) ANSWER = 12 (h) ANSWER = 4*3 (i) ABC (j) A;B;C

4B (a) X (b) ABCDE
YZ

4C Fill in the gaps with: FIRST INPUT FIRST + SUM

4D
```
SPEED OF CAR = 50 MILES PER HOUR
TRAVELLING TIME = 2 HOURS
DISTANCE TRAVELLED = 100 MILES
```

4E
```
10 INPUT "ENTER SPEED ",SPEED
20 INPUT "ENTER HOURS ",HOURS
```

4F
```
PRINT "COST = ";10;" POUNDS"
```

7A Lines 20, 30 and 40 should be indented.

7B
```
START NOW
ENTER NUMBER ?1
2
ENTER NUMBER ?2
4
ENTER NUMBER ?3
6
```

7C
```
10 20   30 40 50   30 40 50   30 40 50   60
```

7D It will keep printing ENTER NUMBER until you press the ESCAPE key or the BREAK key.

7E
```
10 LET ADDUP=0
20 REPEAT
30   INPUT "ENTER AGE ",AGE
40   ADDUP=ADDUP+AGE
50 UNTIL AGE=0
60 PRINT "TOTAL AGE = ";ADDUP
70 END
```

8A (a) 3 (b) 5 (c) 11

8B (b) $ missed out (c) doesn't start with a letter

8C
(b) putting a number into a string variable
(c) putting a string into a number variable
(e) adding a number to a string
(f) putting a number into a string variable

8D
```
INPUT NAME$
```

8E Fill in the gaps with: LAST$ FIRST$ " "

8F
```
10 PRINT "A. N. OTHER"
20 PRINT "12 ANDOVER STREET"
30 PRINT "YOURTOWN"
40 PRINT "ENGLAND"
50 END
```

11A
(a) FOR K=1 TO 5
(b) FOR K=3 TO 6
(c) FOR K=10 TO 50 STEP 10
(d) FOR K=7 TO 4 STEP -1
(e) FOR K=1 TO 9 STEP 2
(f) FOR K=12 TO 18 STEP 2
(g) FOR K=10 TO 40 STEP 5

11B (a) NEXT J should be NEXT K
(b) LET K=1 TO 5 should be FOR K=1 TO 5
(c) FOR K=1,5 should be FOR K=1 TO 5
(d) K$ should be K in all three lines

11C 6

11D
```
10 FOR K=1 TO 4
20   PRINT K; " x "; K; " = "; K*K
30 NEXT K
40 END
```

12A (a) A=10 B=20 C$="A" D=30
(b) NAME$="JONES" TELEPHONE=143729

12B (a) READ SCORE,TEAM$ should be READ TEAM$,SCORE
(b) No data item for C$

12C
```
DRILL
SCREWDRIVER
HAMMER
```

12D
```
10 FOR K=1 TO 4
20   READ NAME$, AGE
30   PRINT NAME$; " is aged "; AGE
40 NEXT K
50 DATA "MARTIN",10
60 ....
70 ....
80 ....
90 END
```

13A (a) 1+2 (b) 7-3 (c) 8/2 (d) 4*3
(e) 3+4+5 (f) 5*2-3 (g) 20/2+4 (h) 2*3*4

13B (a) PRINT 17 DIV 5 (b) PRINT 17 MOD 5

13C (a) 6 (b) 3 (c) 8 (d) 2
(e) 7 (f) 7 (g) 8 (h) 6
(i) 9 (j) 1 (k) 8 (l) 6
(m) 2 (n) 1 (o) 2.5

13D

	A	B	C	D
READ A,B,C	4	3	2	
LET D=A+B*C				10
LET B=(A+C)/B		2		

13E (a) LET X=RND(5) (b) LET X=5+RND(5) (c) LET X=10*RND(5)

13F It will choose a name at random from the 7 names given.

16A (a) NO (b) YES (c) NO (d) YES
(e) NO (f) YES (g) YES (h) YES
(i) YES (j) NO (k) NO (l) NO

16B (a) < (b) > (c) > (d) < (e) < or >

16C (a) <= (b) >= (c) >= (d) <= (e) <= or >=

16D 8 is less than 10
9 is less than 10
10 is equal to 10
11 is greater than 10

17A (a) PROC_PAPER(WHITE)

(b) PROC_INK(RED)

(c) PROC_RECT(400,300, 800,600)

(d) PROC_INK(YELLOW)

(e) PROC_CIRC(600,700, 100)

(f) PROC_INK(BLACK)

(g) PROC_TRI(200,400, 400,500, 400,600)

(h) PROC_TRI(1000,400, 800,500, 800,600)

(i) PROC_TRI(400,300, 500,100, 600,300)

(j) PROC_TRI(600,300, 700,100, 800,300)

(k) PROC_LINE(500,400, 700,400)

(l) PROC_LINE(500,500, 700,500)

You might draw these shapes on a grid similar to that shown in Figure 9.1, and colour each shape in its correct colour.

Now load the PROCS1 program from your cassette, extend the program by adding the commands listed above, and run the program. Is the drawing on the screen the same as on your grid?

Index

All the important features of BASIC are covered in this short introduction and the reader is shown how to use these in simple programs. The book is divided into short units. Each unit focuses on a particular feature of programming on the BBC microcomputer and consists of explanatory text, test questions and practical exercises. The practical exercises can be undertaken individually or demonstrated by a teacher on a single computer in a classroom.

For readers who have reasonably free access to a BBC microcomputer this book is an ideal first self-study text. Those wishing to progress to the more advanced aspects of BASIC and to learn how to design, construct, implement and test larger, complex programs are recommended to *BBC BASIC* by the same author.

Also published by Edward Arnold

BBC BASIC
R. B. Coats
Spectrum BASIC
R. B. Coats
Applesoft BASIC
B. M. Peake
Microcomputing with the PET
J. Arotsky, J. Taylor and D. W. Glassbrook
Microcomputing in BASIC on the RML 380Z/480Z
W. R. McDonough
Computer Keyboard Mastery
Stan Harcourt
Basic BASIC
Donald M. Monro
Basic BASIC on the BBC MICRO *(in preparation)*
Donald M. Monro

Edward Arnold

ISBN 0 7131 3520 4