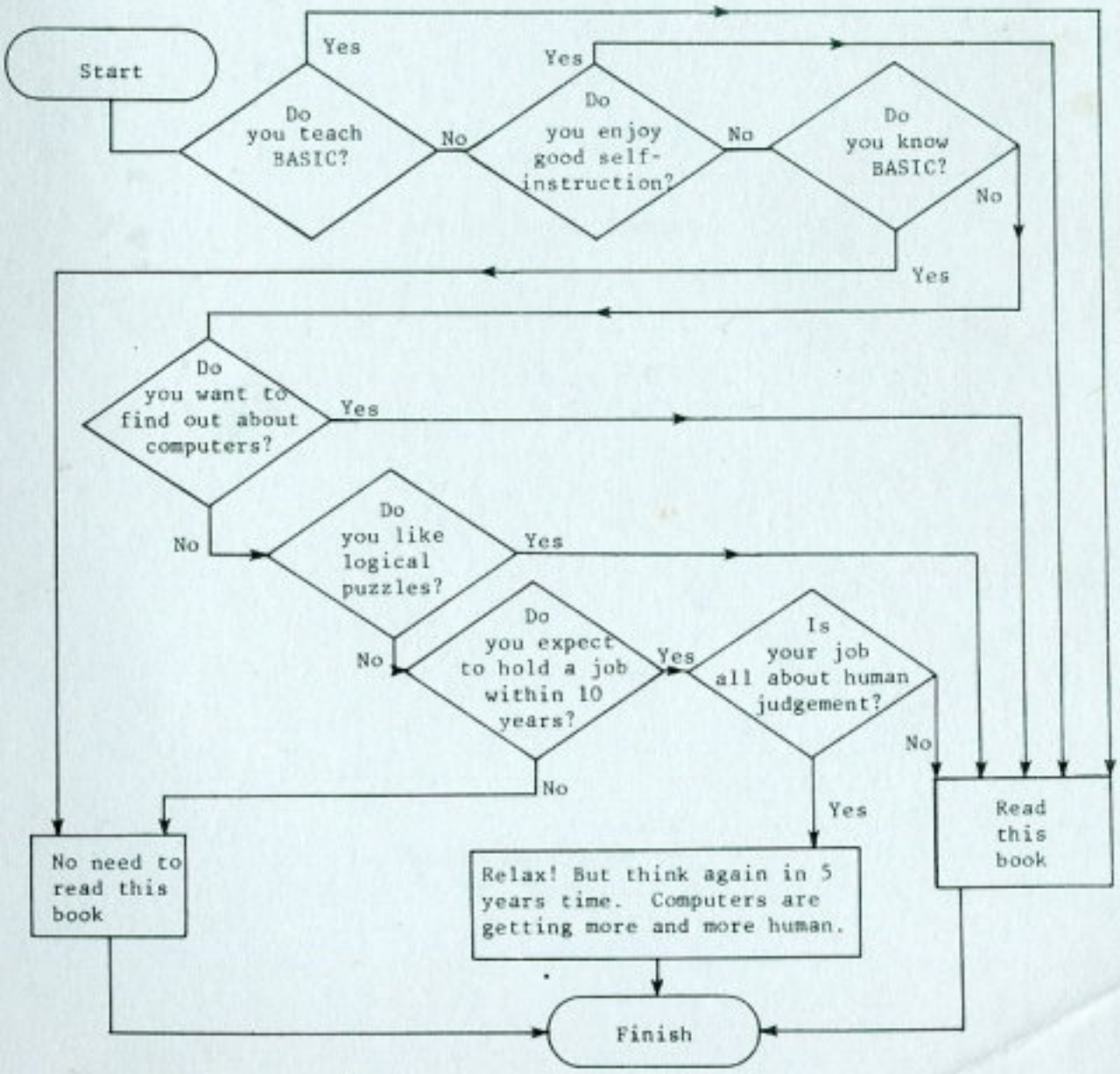
Cambridge Learning Limited

Computer Programming in BASIC

A SELF INSTRUCTION COURSE in 4 Volumes

PART 1 — BASIC Basics

By:
Ian Williamson Rodney Dale Tim Eiloart



COMPUTER PROGRAMMING in BASIC

PART 1 - BASIC BASICS

PART 2 - INTRODUCING BASIC

PART 3 - APPLYING BASIC

PART 4 - ADVANCED BASIC

Computer Programming in BASIC

A unique, teach-yourself course

Part 1: BASIC Basics

by

Ian Williamson, Rodney Dale and **Tim Eiloart**

First published 1979.
Reprinted 1980 (twice), 1981, 1982

ISBN 0 905946 05 7

published by
Cambridge Learning Limited
Rivermill Lodge
St. Ives
Huntingdon
Cambridgeshire
U.K.

printed in England by
Cambridge Learning Limited

CONTENTS with Summaries

PROGRAMMED LEARNING

This course, COMPUTER PROGRAMMING IN BASIC, consists of four self-instruction or programmed learning books. Learning by self-instruction has the advantage of being quicker and more thorough than classroom learning. The student works at his or her own speed and is regularly asked to respond by answering a question or solving a problem on new information before continuing the course.

The course is divided into 60 Lessons. Each lesson follows a standard format:

- a lesson begins with a very short summary of the contents.
- lessons which are optional start with a question, if you answer the question easily then you can skip the lesson.
- occasionally instructions at the beginning of a lesson may direct you to a supplementary lesson.
- words printed in italics thus - *debugging* - are in the glossary. We use italic text the first time you come across such words.
- during each lesson you will be expected to use your mind often. Whenever you see QUESTION or PROBLEM, try to answer the question or solve the problem before reading on. Sometimes we give a HINT separated by a dotted line from the question. If you cannot work out the answer, read the hint, and then try the question again. The answer is not a trick and may often seem obvious.

Below is a sample of a QUESTION, HINT and ANSWER.

QUESTION

Why is self-instruction more thorough than classroom learning?

HINT

How fast do you learn?

ANSWER

The student can work at his or her own speed and frequent questions and problems are used to check the rate of learning.

USING THIS TEXT

How this course is designed to be highly effective whether or not you have access to a computer.

There is no 'correct' way to learn computer programming - particularly computer languages like BASIC - just as there is no 'correct' way to learn a foreign language. Many people learn French in the classroom, and become fluent in the language without ever visiting a French-speaking place. If one is fortunate, one has the opportunity of learning it whilst living and working in France or Quebec. In this case one learns more quickly, but we cannot say that classroom French is any less correct than French learned from day-to-day contact with French speakers.

You can learn a lot about BASIC and other computer languages while sitting at a computer terminal - learning by your mistakes. However, whilst you will master the language of the computer, you may not learn much about effective computer programming. To learn computer programming, a lot of thought and patience will serve you well in the long run. As always there is no correct way to learn, readers must develop the styles best suited to themselves.

With computer programming, practical considerations will largely determine this style of learning. Some readers will have ready access to computers. These readers are fortunate; they will have the opportunity of experimenting with programs introduced early in the text. Other readers may have to pay for the computer time they use - and computing can be very expensive if one has to pay the full commercial rate. These readers would be best advised to wait until a full understanding of computer programming has been developed, before testing their skills on the computer itself.

This text has been designed for both classes of user, and all those in between. The reader with ready access to a computer is referred to supplementary lessons which should guide his or her experiments on the computer itself. If you are not going to use a computer regularly, don't worry, this is a programmed-learning course, and we will refer you back to these supplementary lessons as required; so you will achieve a complete understanding of computer programming and the BASIC language.

A word about the derivation of this course. The first draft of the text was written by IAN WILLIAMSON and RODNEY DALE to complement the very successful Design of Digital Systems also published by Cambridge Learning Limited. This first draft was used by TIM EILOART to teach BASIC PROGRAMMING to general studies students. This practical experience with the text led to modifications particularly of PART 1 of the text where too high a mathematical ability had been assumed. Here then is a course which makes no assumptions about your mathematical ability.

Additional lessons are provided to ensure that your appreciation of mathematical terms does not limit your understanding of computer programming. For many readers these lessons will be very simple - if so, then just skip the lesson. Instructions and a question at the start of each lesson will help you decide whether the lesson is relevant to you. Follow the instructions and you will find the text neither boring nor over-taxing in content.

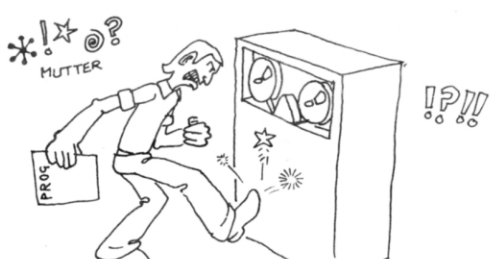
A text on computer programming should ideally concentrate on the development of a clear and effective approach to problem-solving. The pedantic detail of a computer programming language - grammar, punctuation and spelling - is less important. Even so some part of the text must be given over to the detail of the programming language. In this text, opportunities for problem-solving and detailed programming appear in most lessons. It is up to you the reader to make the most of these opportunities - the more you put into this course, the more you will get out. In this way, we hope that you will not only acquire a detailed knowledge of the BASIC language, but also develop an ordered and logical approach to problem-solving in computer programming.

A few further points to bear in mind when you read these books;

- The computer language, introduced in this book, is BASIC. BASIC stands for Beginners' All-purpose Symbolic Instruction Code.

- There are many dialects of BASIC. By this we mean that BASIC has never become a standard language and different computer systems will have versions of BASIC which differ in many small ways. In this text we have tried to keep to a near-standard BASIC often referred to as Dartmouth BASIC. (BASIC was developed at Dartmouth College in USA.) However, before attempting to execute any programs from this text on your computer system, you should check the BASIC manual of your computer for any variations from the 'standard' used in this text.

- In most languages the beginner can get by with small mistakes. Not so with computer programming - you have to be word perfect and punctuation perfect. The computer is very pedantic so you will find many of the questions a little pedantic - persevere! Fortunately the computer is very good at spotting small errors in spelling and punctuation.



LESSON 1

WHAT IS A COMPUTER?

Many terms are defined so that you will be able to understand the lessons that follow, and talk to people about computers.

A computer is a machine which takes in information, processes it and provides a set of results as an output. Contrary to what many people believe, computers cannot 'think' in the way humans can; they can only do what they are told. However, they can be told to do things which often give the appearance of thinking.

Its characteristics are:

1) It works very quickly.

2) It can carry out a complicated sequence of operations once it has been told what to do. Such operations include calculation, presenting tables etc.

3) It can *branch*, that means choose between alternative sequences of calculations.

4) It can *store* as much information as it has capacity for, and there is theoretically no limit to the amount of storage which can be made available. Information is stored in the *memory*.

5) It can usually present the results of its labours in one or more different ways - as *alphanumeric characters* (letters and figures) printed on paper, or displayed on a screen; graphically; on tape - punched paper or magnetic and so on.

As we have said, a computer will operate at high speed. Typically, a large computer (often called a *mainframe* computer) is capable of carrying out several million additions per second, so it should not be kept waiting by its human operator. For this reason, it follows a *program* which directs the course of its operations. The program is stored within the computer in a *language* which it can understand. This series of books will teach you the language called BASIC. The process of preparing the sequence of instructions which can be executed by a computer to perform some task is called *computer programming*.

In the early days of computing, as in the early days of motoring, the only person who could use the machine was the one who had designed and built it. This was due mainly to the difficulty of telling the machine what to do. As we shall see, modern computers are such that their programmers need not understand the inner workings, though a good programmer will be aware of the facilities offered by his computer and the type and capacity of its memory. He will understand the methods by which programs and information may be fed into his computer, and he will appreciate the range of *output devices* which may be used to present the results of the program.

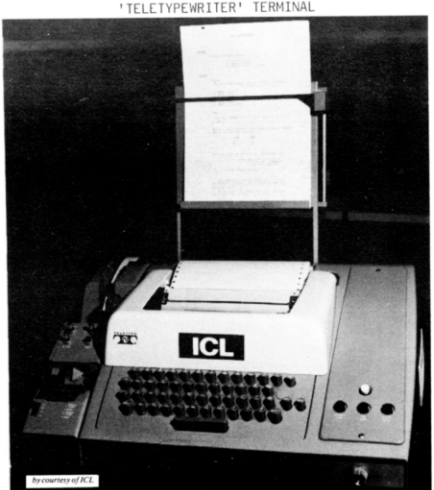
Finally, a good programmer will be aware of the relationship between a program, the amount of memory required to store the program, and the time the computer will take to execute it. The memory requirement and the execution-time largely determine the cost of using the computer for any particular problem.

COMPUTER SYSTEMS

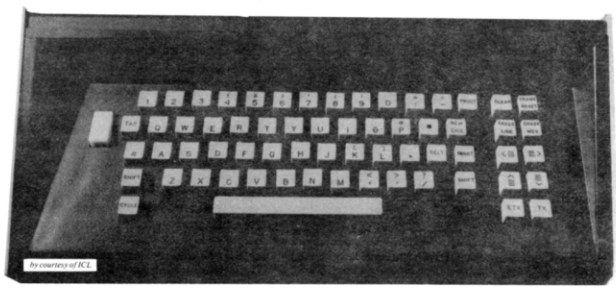
Your first contact with a computer may well be by way of a *time-sharing* system. In a time-sharing computer system, several users share the services of one computer at the same time, but it generally appears to each user that he has sole use of the computer. The user is equipped with a personal terminal, which is usually a *visual display unit*, a television-like display equipped with a keyboard. On older computer systems the terminal is often a typewriter-like device referred to as a *teletype*. (Teletype is a trademark of the Teletype Corporation which manufactures many of the terminals currently in use.) The terminal is often installed in the office, factory, or home of the user, remote from the computer installation. Communication with the computer takes place over a standard telephone line using an electronic *modem* (modulator-demodulator) which converts the data into a suitable form for transmission over a telephone circuit. At the computer installation, a special type of program called an *executive* controls the way in which the computer serves each user in sequence, thus giving the effect of all the users being served simultaneously.

VISUAL DISPLAY UNIT

'TELETYPEWRITER' TERMINAL



TYPICAL KEYBOARD LAYOUT



In a time-sharing system the user is said to be *on line* to the computer. In an on-line system, the user can *interact* with the computer as it executes the program. The computer can be made to pause during a program run - usually after presenting some intermediate results - and the user can then provide additional data, stop the program run, or instruct the computer to present more detailed results. Another advantage of an on-line computer system is that the user has immediate access to the storage, editing, and processing facilities of the computer, because the user terminal is connected directly to the *central processing unit* (CPU) of the computer.

Because they are interactive, on-line systems are particularly suitable for the development of programs, and for training inexperienced computer programmers. This is because the results of the program come back immediately, and, if the program is wrong, it can be corrected before its logical flow has been forgotten.

A time-sharing system is only one example of an on-line system. All computers can be operated on-line, that is with the user able to interact with the computer, but, because the human operator is very much slower than the slowest computer, this is a very inefficient way of utilising the sometimes expensive resources of a computer. In small computers (minicomputers), where the CPU and memory does not support a time-sharing *executive*, on-line access is available to one person at a time only. Programming minicomputers is more difficult than programming large ones, because of the limited memory-capacity of small machines.

A recent development is the *micro computer*, which is a very small computer system, constructed from cheap, large-scale integrated units (LSI). Many hobby-computers are now available, ranging from the bare minimum introductory kits to quite sophisticated machines with many of the facilities of larger computer systems. Popular micro computers such as the TRS-80 from Tandy, the PET from Commodore and the APPLE II from APPLE can all be programmed using the BASIC language.

An alternative to the on-line system is the *batch-processing* computer system. In a batch-processing system, the program is submitted to the computer on punch cards, paper tape, magnetic tape or some other transportable storage medium. The program then joins a queue, and is executed by the computer when it reaches the front of the queue. The print-outs are collected or delivered later - in many computer installations, processing is carried out over-night and the results are available the following day.

The program is prepared *off-line*, that is, on a terminal not connected to the CPU of the computer. A terminal with a paper-tape punch and reader, or a magnetic tape cassette unit is usually used for off-line preparation of programs. The programmer types out his program, and a record of the program is stored on paper-tape (or magnetic tape). Errors are corrected by copying tape-to-tape, stopping the tape-reader as necessary to correct typing errors. When error free, the program is read from paper-tape or magnetic tape into the computer for processing.

Batch-processing is ideal for large, established programs such as those for calculating a company payroll, or sorting and analysing questionnaires; it is the cheapest form of computer service because the computer is kept fully occupied processing the job queue. Batch-processing can be frustrating and inefficient for program development, since it can take up to 24 hours for a programmer to see the results of a program run. Fortunately, many computer bureaux offer a time-sharing service during the day, and batch-processing over-night - thus, the programmer can choose the more suitable computer service. In either case the cost is relatively low since the overheads of the computer installation are spread among hundreds of users.

In the world of computers, programs are referred to as *software* in contrast to *hardware*, the computer itself. Some programs, referred to as *system software*, are provided by the computer manufacturer. An example of a system program is the executive program which controls the computer in a time-sharing system. Programs which you will generate as a programmer for your own use are referred to as *user-programs*. However, increasingly important in assessing the usefulness of a computer system is the availability of applications-software. By applications-software, we mean standard programs for frequently encountered problems. Applications-software might be used to do such diverse tasks as critical path analysis, bought-ledger accounting, financial accounting, time-sheet analysis, curve-fitting and graph-plotting. The use of applications-software normally requires little or no experience of programming.

This course is designed to develop your skills as a programmer so that you can generate your own user-programs.

QUESTIONS

a) What is the difference between a time-sharing computer system, and a batch-processing computer system?

b) What are the advantages and disadvantages of each?

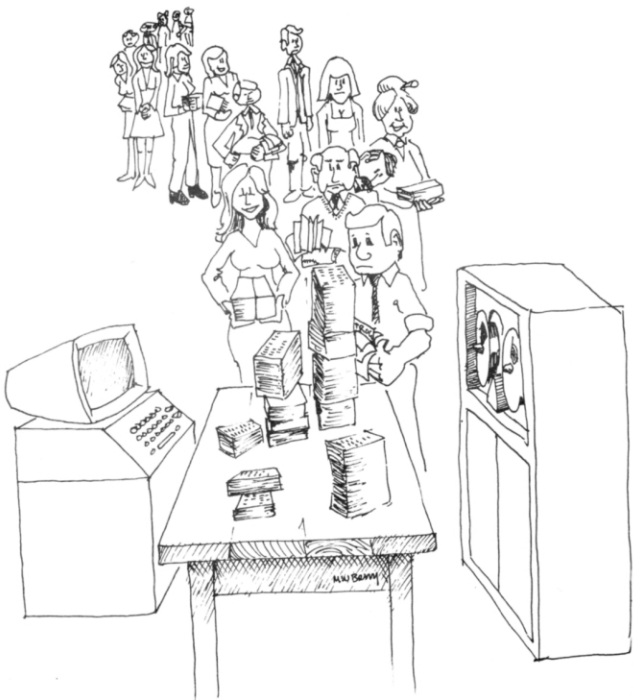
ANSWERS

a) Difference between time-sharing and batch-processing:

In a time-sharing system, many users have access to the computer simultaneously and are served in turn by a special program called an executive. The executive switches the computer resources to each user in turn, sufficiently rapidly for each user to appear to have sole use of the computer.
In a batch-processing system, the programs and data are held in a queue, and each job is processed completely when it reaches the front of the queue. Thus, the result of the job may not be available for perhaps 24 hours after the submission of the program.

b) Advantages and disadvantages of time-sharing and batch-processing.

A time-sharing system is excellent for a programmer who wishes to develop his program as he sits at the terminal; on the other hand, this can be very expensive. Batch-processing does not allow such development, but it makes good use of computer time - provided that there is always a queue of work.



LESSON 2

WHAT COMPUTERS ARE BEST AT DOING

If people agree on the right way to tackle a problem, it is likely that a computer can be programmed to solve it very quickly, with no mistakes. Where people do not agree, then a computer cannot easily help.

QUESTIONS

Decide which of the following pairs of tasks a computer might easily be made to do and which it might be made to do only with difficulty. These tasks it would do as requested but some people would probably say it had done them poorly. For each pair of questions, computers are better suited to one than the other.

1) Keep records of league scores for a football league.
2) Select the best team for a football match.

3) Choose the best price to sell old, unfashionable shoes.
4) Monitor the number of meals of each type being sold by a chain of restaurants.

5) Calculate the time of arrival of an aeroplane crossing the Atlantic.
6) Propose the most suitable pilot for promotion to fleet-captain.

7) Play chess.
8) Play noughts and crosses (tic-tac-toe).

ANSWERS

The computer is more easily made to do 1, 4, 5, and 8. If you had problems selecting the right answer then read the rest of this lesson.

A computer can do very simple things very quickly, such as addition, subtraction, and comparing names on a list in order to arrange them alphabetically. It will do this sort of work again and again, so fast, that it can appear to do quite complicated things. It can easily compile a telephone directory in alphabetical order of surnames. Then it can compile another according to the order of the phone numbers. Then it can write a directory, street-by-street.

Similarly, computers are very good at operating accounting systems, wages systems and so on, because these applications involve the accurate processing of huge amounts of data.

The power of a computer system can be approximately assessed from the *size* of its memory and the *speed* of the central processor unit (CPU). All the actions of the computer can be broken down to the transfer of a single package of data from memory to CPU, within the CPU, or from CPU to memory. The size of memory determines how much data can be processed, and the speed of the CPU determines how fast it can be processed.

Suppose we wish to use a computer to sort a list of names into alphabetical order. First the unordered list would be entered into the computer memory. Then, under control of a program, the CPU would examine each name in the list in turn looking for the name placed highest in the alphabet. This name would then be placed at the top of a new list in computer memory, and erased from the original list. The CPU would then scan through the original list once again looking for the next highest name in the alphabet. And so on, until the list is exhausted. A list of a thousand names might require one thousand thousand operations by the computer - perhaps as much as 2 seconds of CPU time in a powerful modern computer.

Remember a computer does not think. It faithfully regurgitates facts (which you originally provided) and it is only going to tell you things which you told it to look for. Computers are often described as number-crunchers. We hope you see why.

Here are the questions presented at the start of this Lesson. We have added a brief explanation of why computers can do one of the tasks more easily.

DISCUSSION OF PROBLEMS

1) Keep records of league scores for a football league.
   This is a numerical job that a computer can easily do.

2) Select the best team for a football match. This is a matter of judgement. The team coach should know about the many factors which will affect the teams performance, such as:

   - The players on the opposing team, their fitness and their strengths and weaknesses.
   - The reputation of players on both teams.
   - The form of the home team.

   There are many factors which will influence the best choice of team, and the computer can only evaluate these factors if it has been given all the information and has been painstakingly told how to make a decision on this information. There would probably be no benefit in using a computer for this task. No matter who chooses the team, the press and the public will say the wrong players were selected. It is because *people* cannot agree about how to choose football teams, that it is difficult to use a computer to do the job.

3) Choose the best price to sell old, unfashionable shoes.
Again this is a matter of judgement; how unfashionable? How quickly must the old stock be sold? and so on. The computer has little to contribute to this type of decision. Again any two experts are unlikely to agree with one another.

4) Monitor the number of meals of each type being sold by a chain of restaurants.
This is purely numerical, and requires efficient processing of a massive amount of data. Most companies find computers very valuable when monitoring demand in this way.

5) Calculate the time-of-arrival of an aeroplane crossing the Atlantic.
This is a complex calculation but, provided the computer has the necessary data on cruising speeds, wind direction and speed, and fuel consumption, it can easily do the arithmetical computation.

6) Propose the most suitable pilot for promotion to fleet-captain.
As in 2) above this is a matter of judgement. People will disagree about the correct choice. So it is difficult to use a computer to help.

7) Play chess.
Computers are not very good at chess. At present the best computer chess program playing on the most powerful computer in the world is not at grand-master standard in the chess world. Nonetheless computers can beat most normal players because computers never make silly mistakes. But computer programs are not able to match the behaviour of chess champions, or to use the gamesmanship and psychology so evident in major chess tournaments. The difficulty is that we cannot understand the thought-processes of chess champions - so we cannot teach computers to play the game so well.

8) Play noughts and crosses, (tic-tac-toe)
Computers are perfect at games such as noughts and crosses because they never make mistakes. In noughts and crosses you can always draw if you make the right moves. People too play this game perfectly. But at least computers here are as good as the most competent humans.

LESSON 3

SAYING IT WITH LETTERS

Sometimes you use initial letters as a type of shorthand, for example: "P.T.O." for "Please turn over". When you program a computer you use this approach all the time.

In Lesson 4 we introduce a very simple computer program. Computer languages usually use letters and names to represent the data being processed. There is nothing difficult about this *labelling* process, indeed it was designed to make your job as a computer programmer easier. If you studied algebra, or set-theory, or even geometry, at school or college, you will be quite at home with the use of letters to represent data in computer programs.

QUESTION

P = 3C+1.5M+2B

where P = contents of my pocket

C = coins

M = matches

B = ball-point pens

Write out in words exactly what I have in my pocket.

ANSWER

I have three coins, one-and-a-half matches, and two ball-point pens.

Even if you found that easy, you may wish to read the rest of this lesson for interest.

Try the exercise below.

QUESTION

A doctor prescribes aspirin as follows:

150 aspirin tablets for each of F cases of flu
50 aspirin tablets for each of H cases of headache
25 aspirin tablets for each of C cases of common cold.

What is the total number of aspirin tablets 'A' the doctor requires?

ANSWER

A = 150 x F + 50 x H + 25 x C

This can also be written 150F + 50H + 25C - where we omit the multiplication signs. We can, of course, apply the same principle when fractional quantities are required.

QUESTION

Each cow yields 2¼ gallons of milk a day, each goat yields only ½ gallon a day. If a farmer has C cows and G goats, how many gallons of milk, M, can he expect each day?

ANSWER

M = 2¼ x C + ½ x G *or* 2¼C + ½G (again, we can omit the multiplication sign).

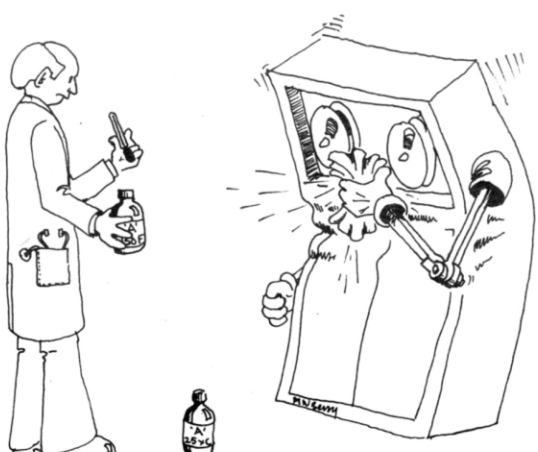
QUESTION

Now express this in decimal form .....?

ANSWER

M = 2.25 x C + Ø.5 x G or 2.25C + Ø.5G

These examples demonstrate how letters are used in computer programs to represent quantities in arithmetic calculations. Names are used to represent quantities which may vary. Thus, in the example above, the number of cows and goats may vary on each individual farm, but the yield of milk per animal does not.



LESSON 4

READ, DATA, PRINT, END for a simple program

Now you can start to work out simple problems using BASIC.

Here is a simple program - it solves a problem that you met at an early age.

```
1Ø READ A,B
2Ø PRINT A+B
3Ø DATA 2,2
4Ø END
```

The computer uses Ø for zero to avoid confusion with capital O.

QUESTION

You might like to guess what the program does and what the program will print?

HINT

The DATA 2,2 refers to A and B.

ANSWER

The program solves 2+2 = 4
the sum is printed as
4

Now let us look at each line in turn.

```
1Ø READ A,B     This means READ the values of A and B from a DATA statement
2Ø PRINT A+B    This means PRINT out the value of A+B
3Ø DATA 2,2     This means A=2 and B=2
4Ø END          This tells the computer that the job is finished
```

Notice that every line has a line number. The contents of the line form a single BASIC *statement* which is identified by the line number. Statements are executed in the order of the line numbers starting with the line that has the lowest number.

QUESTION

Now try to write a program yourself, following the same form, which will PRINT the volume of a rectangular brick whose dimensions are:

height H 3 units
length L 6 units
width W 1 unit

Volume is H x L x W but for multiplication use this sign *

ANSWER

```
1Ø READ H,L,W
2Ø PRINT H*L*W
3Ø DATA 3,6,1
4Ø END
```

You will already have noticed differences between the computer print-out and normal text. Firstly, many computer terminals print only in capital letters; so we have to distinguish between O and Ø, the latter representing, as we have said, the number zero. Secondly the printer has no character for multiplication, and therefore computer printers use asterisks * to represent the multiplication sign. You will spot some other differences as you become familiar with the computer terminal, but the meaning of these new symbols is usually fairly obvious.

If you had any trouble with the problem, then here is another. Otherwise skip this problem - it introduces no new facts.

QUESTION

Write a program to find the total thickness of six books.
They are, respectively, 12, 21, 32, 15, 14 and 24 mm thick.

ANSWER

```
1Ø READ A,B,C,D,E,F
2Ø PRINT A+B+C+D+E+F
3Ø DATA 12,21,32,15,14,24
4Ø END
```

LESSON 5

RUNNING BASIC PROGRAMS

When you work on-line you have to understand the things the computer says, as well as knowing how to write your program.

Most people, when first confronted with a computer, have difficulty understanding how programs are fed into the computer, and how they are executed and the results printed out. Learning to program from text-books can even add to this confusion. In this lesson we show how simple programs can be executed and how the computer responds to the commands of the programmer.

Suppose we have switched on the computer and identified ourselves (this is explained in detail later), the computer is now waiting for the first command. Below we show a typical session at the computer terminal. Everything printed by the computer is underlined, the rest is typed by the programmer.

```
RUN BASIC

NEW OR OLD---NEW
NEW PROGRAM NAME IW1Ø
READY

1Ø READ A,B,X
2Ø LET Y=A*X/4+B
3Ø PRINT "Y EQUALS",Y
4Ø DATA 4.2,Ø.5,2.7
5Ø END
RUN
IW1Ø              31-MAY-99          19:34:53

Y EQUALS       3.335
END AT 5Ø
READY

SAVE IW1Ø
READY

KILL
```

Let us look at each line of this print-out individually.

RUN BASIC

After we have turned the computer on, the machine is under the control of an internal program which is sometimes called the EXECUTIVE. Our program is in BASIC and we need to tell the computer that a BASIC program is about to be entered. We type the command RUN BASIC and in response to this command the executive reads from the computer memory all the special data and routines needed for the BASIC language. The computer is prepared to run the BASIC language programs. This is as though we had been offered a multi-lingual human interpreter. We might say "Right! French to English!" With computers we use only one language at any time. But we have to tell the computer which language we are going to use. If we are in a time-sharing system the computer might be running BASIC for us, and other languages for other people.

RUN BASIC is a command to the computer executive and is not actually part of the BASIC language.

NEW OR OLD--NEW

The computer responds to the RUN BASIC command with the message NEW OR OLD. The computer is asking whether a new program is to be entered or whether the program had already been entered and is stored in the memory of the computer.

We have a new program - so we type NEW.

NEW PROGRAM NAME IW1Ø

To store programs or data in its memory the computer needs a program name. Just like any filing system. We name our program IW1Ø. Program names are simply convenient tags by which we can recall the program from the computer memory. Program names are usually representative of the function of the program - for example CHESS for a chess program; or they identify the programmer - in this case IW1Ø may stand for I. Williamson program version 1Ø. Usually the program name can be any combination of up to 6 or 7 letters or numbers, but the FIRST CHARACTER MUST BE A LETTER. The program which we now enter is stored under the program name IW1Ø in a temporary *store within the computer. You will see later how to store the program more permanently.*

READY

The computer is now ready to accept the BASIC program.

```
1Ø READ A,B,X
2Ø LET Y=A*X/4+B
3Ø PRINT "Y EQUALS",Y
4Ø DATA 4.2,Ø.5,2.7
5Ø END
```

We type in the program, starting each line with a number. This line number tells the computer that a line of program is being entered.

This program is explained in detail in the next Lesson.

RUN		*We have now entered the complete program and want the computer to run it so that we can see the results. To do this we simply RUN. The computer takes the program from the temporary store, executes it, returns the program to the temporary store, and prints the results.*
		The computer executes the program and prints the results.
IW1Ø	31-MAY-99 19:34:53	*The computer prints the name of the program, the date, and the time of day. The time is 7:34 pm and 53 seconds. Computers are very pedantic about time; time costs money.*
Y EQUALS 3.335 STOP AT 5Ø READY		*This is the result of running the program. First the result according to the PRINT statement in line 3Ø of the program. Next a message from the computer telling you that it stopped running the program at line 5Ø, which is the END statement. Finally another READY message telling you that the computer is ready for a new command.*
SAVE IW1Ø		*The program works, we may want to use it again sometime, so we want to transfer it from the temporary store to a permanent store, usually a magnetic disc or magnetic tape. We do this by giving the command SAVE followed by the name of the program IW1Ø.*
READY		*Now the computer has completed the transfer to the permanent storage, and is waiting for a new command.*
KILL		*We have completed the computing session; our program has been entered, tested, and is now in permanent storage under the name IW1Ø. We can recall this program any time we wish, using the OLD command mentioned above. We now want to return to the executive and finish our computer session. We type the command KILL, this clears from the temporary memory the program named IW1Ø, and also erases the data and routines required to run BASIC programs.*

On the print-out shown above, some of the text is typed by the computer and some is typed by you - the programmer. When you sit at the computer terminal all this will become obvious, and who types what will be clear. However most of the examples in this book use actual terminal print-out as illustrations and therefore you should try to spot who is printing what from the pages of this book.

The main source of confusion comes from the use of the carriage-return and line-feed. On a typewriter carriage-return and line-feed are combined into a single lever or key. On a computer terminal, there are two actions. The carriage-return is controlled by the RETURN key, and the new line by the LINE-FEED key. When you are connected to the computer, and it is your turn to type, the computer waits for the RETURN key to be depressed before it takes further

action. Thus you type a command into the terminal, but the computer takes no action until you press the RETURN key to show that you have finished. The first action of the computer is to send a LINE-FEED to the terminal, so that the next line of text does not overprint the last. You then type your next command on the new line. You too may press the LINE-FEED key when you wish.

The computer will also send additional LINE-FEEDS as required, to ensure that the print-out is neatly spaced, as for example between the RUN command and the title of the program print-out.

QUESTION

Where would you have pressed the RETURN key in the print-out shown below?

```
RUN BASIC
NEW OR OLD--NEW
NEW PROGRAM NAME IW1Ø
READY

1Ø READ A,B,X
2Ø LET Y = A*X/4+B
3Ø PRINT "Y EQUALS",Y
4Ø DATA 4.2,Ø.5,2.7
5Ø END
RUN

IW1Ø          31-MAY-99              19:34:53

Y EQUALS        3.335
STOP AT 5Ø
READY

SAVE IW1Ø
READY

KILL
```

HINT

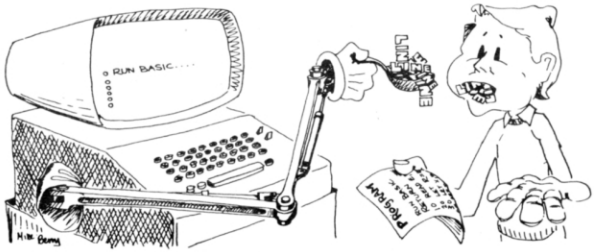
If you cannot work out which parts you type read the lesson again.
Here is the final segment of program marked with the RETURNS marked (RET).

```
READY

SAVE IW1Ø                  (RET)
READY

KILL                       (RET)
```

Now do the rest of the exercise yourself.

```
ANSWER

RUN BASIC                    (RET)
NEW OR OLD--NEW              (RET)
NEW PROGRAM NAME IW1Ø        (RET)
READY

1Ø READ A,B,X                (RET)
2Ø LET Y = A*X/4+B            (RET)
3Ø PRINT "Y EQUALS",Y         (RET)
4Ø DATA 4.2,Ø.5,2.7           (RET)
5Ø END                       (RET)
RUN                          (RET)

IW1Ø          31-MAY-99              19:34:53

Y EQUALS       3.335
STOP AT 5Ø
READY

SAVE IW1Ø                    (RET)
READY

KILL                         (RET)
```

Do you intend to use a computer terminal regularly during the first book of this programmed learning course? If so, then read Supplementary Lesson 1 before actually experimenting at the computer terminal. If you are not going to use a computer, go to Lesson 6; we will refer you later to the supplementary lesson, when it will help you to understand computer programming.



LESSON 6

ANOTHER BASIC PROGRAM EXAMPLE

More practice.

In Lesson 4 we showed another simple BASIC program to evaluate the algebraic expression

$$\frac{A*X}{4} + B$$

(Remember * means 'multiply').

Here is the program:

```
1Ø READ A,B,X
2Ø LET Y=A*X/4+B
3Ø PRINT "Y EQUALS",Y
4Ø DATA 4.2,Ø.5,2.7
5Ø END
```

This program is performing a task so straightforward that a computer would be unnecessary. It does illustrate the 'English language' nature of BASIC, and elements of quite detailed programs are contained in these five lines. Let us look at each line individually:

```
1Ø READ A,B,X
```

This READ statement causes the first three numbers in the DATA statement (line number 4Ø) to be read. In the example the variable A will be given the value of 4.2, the variable B the value Ø.5, and the variable X the value 2.7. (See line 4Ø)

```
2Ø LET Y=A*X/4+B
```

*This statement sets Y equal to the value of the expression A*X/4+B. The computer performs multiplication and division before addition. The division is indicated by the / character.*

```
3Ø PRINT "Y EQUALS",Y
```

The value of Y is printed on the terminal preceded by a label, "Y EQUALS". A "label" is a form of words which describes the meaning of a number. For example, here the words "Y EQUALS" tells you what the number means. If the program merely printed the value of Y, then you might soon forget what the value meant. So "Y EQUALS" is a label.

```
40 DATA 4.2,0.5,2.7
```

this list of numbers in the DATA statement does not cause any action by the computer. The DATA statement is an example of a non-executable *statement, since the computer does not execute any instruction. The READ, LET and PRINT statements, however, are* executable *because they cause the computer to do something.*

```
50 END
```

this tells the computer that the program is complete.

This example program contains the essential elements of any computer program, namely:

- a READ statement which brings data into the computer from the DATA statement
- a LET statement which performs the calculation
- a PRINT statement which instructs the computer to provide the output data.

Thus the program inputs data, processes it and outputs the results.

QUESTION

What is an executable statement?

ANSWER

A statement which instructs the computer to carry out some action during the execution of the program.

PROBLEM

Write a BASIC program to evaluate the equation:

$$P = \frac{3.14159}{180} \quad Q$$

where:

```
Q = (A-X)*(A-X)*(A-X)
A = 1.2
X = 0.032
```

NOTE: 3.14159 is approximately π or pi, the ratio of the circumference of a circle to its diameter. We could define π to more decimal places, but 3.14159 is sufficiently accurate for most calculations.

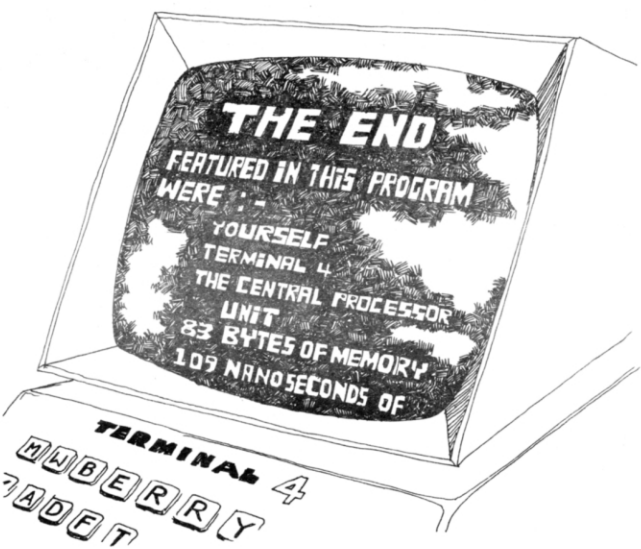
NOTE 2 - Lesson 8 tells you about the use of parentheses (brackets).

ANSWER

```
1Ø READ A,X
2Ø LET Q = (A-X)*(A-X)*(A-X)
3Ø LET P=3.14159*Q/18Ø
4Ø PRINT "P EQUALS",P
5Ø DATA 1.2,Ø.Ø32
6Ø END
```

In line 2Ø the quantity (A-X) is cubed or, in mathematical terms, raised to the power 3. In BASIC, raising to a power is achieved using a special symbol, '↑', an upwards arrow which is available on the computer terminal keyboard. Using this special symbol we can re-write line 2Ø as follows:

```
2Ø LET Q=(A-X)↑3
```



LESSON 7

RAISING TO A POWER

In which you learn to evaluate: 2^4, $36^{\emptyset.5}$, 3^3, $27^{1/3}$ *and* $9^{1.2} / 9^{\emptyset.7}$ *and similar problems.*

QUESTION

What is the value of these?

2^4, $36^{\emptyset.5}$, $256^{\emptyset.125}$, 3^3, $32^{2.1}/32^{1.9}$

ANSWER

16, 6, 2, 27, 2

If that was easy, skip the remainder of this lesson and go to Lesson 8 on page 34.

Consider a sum such as this gives rise to:

As I was going to St. Ives
I met a man with seven wives
And every wife had seven sacks
And every sack had seven cats
And every cat had seven kits
Kits, cats, sacks and wives,
How many were going to St. Ives?

QUESTION

What is the answer ?

ANSWER

The traditional answer is that the man and his wives, etc. were all walking away from St. Ives - (that's a small market town in East Anglia, England, where this course was originally edited).

However, many people say:

1 man	1
7 wives	7
7 x 7 sacks	49
7 x 7 x 7 cats	343
7 x 7 x 7 x 7 kits	24Ø1
TOTAL	28Ø1

A line like this:

$$7 \times 7 \times 7 \times 7 = 24\text{Ø}1$$

can be written like this:

$$7^4 = 24\text{Ø}1$$

which is read as "seven to the power of four equals 24Ø1".

To use this way of writing is simply a piece of mathematical shorthand. As long as it is agreed that $7^4 = 7 \times 7 \times 7 \times 7$, then there can be no confusion.

QUESTION
What is 3^3?
ANSWER
$27=(3 \times 3 \times 3)$
QUESTION
What is 2^5?
ANSWER
$32=(2 \times 2 \times 2 \times 2 \times 2)$

Another piece of mathematical shorthand can be used in this way.

QUESTION
Suppose we have a square of carpet whose *area* is 4 square metres. What is the length of the side?
ANSWER
Most people can work out that a 2 metres x 2 metres square of carpet has an area of 4 square metres. So, the length of a side is 2 metres. 2 is called the "square root" of 4.

Here is $2^2 = 4$ and also $\sqrt{4} = 2$, where $\sqrt{\ }$ indicates square root. But we can also write fractional powers thus:

$$2^2 = 4^1 \text{ therefore } 2^1 = 4^{1/2} = \sqrt{4}$$

Similarly $2^3 = 8^1$ therefore $2 = 8^{1/3} = \sqrt[3]{8}$

and so on.

QUESTION

What is $256^{\emptyset.125}$?

HINT

$\emptyset.125 = 1/8$

ANSWER

$2^8 = 256$, therefore $256^{1/8} = 2$.

The roots of some numbers, such as 2,3,5,7, etc. do not emerge as whole numbers, and like the number π we can never work them out fully. These roots are called *irrational* numbers. You may find yourself using such numbers in your computations - but the computer will only evaluate the first 10 or 12 figures in the root, which corresponds to the limit of its own internal accuracy.

QUESTION

What is $2^3 * 2^4$?

ANSWER

$2^3 * 2^4 = 2^7 = 128$

Perhaps you calculated the result by evaluating first $2^3 = 8$, followed by $2^4 = 16$ and completing the calculation with $8 * 16 = 128$. But adding the *indices* 3 and 4 gives the same result.

$$2^3 * 2^4 = 2^{3+4} = 2^7 = 128$$

The effect of adding indices can be seen from the simple example of square and cube roots: e.g.

$$3^{1/2} * 3^{1/2} = 3^{(1/2 + 1/2)} = 3^1 = 3$$

$$4^{1/3} * 4^{1/3} * 4^{1/3} = 4^{(1/3 + 1/3 + 1/3)} = 4^1 = 4$$

To multiply, the indices are added; to divide, the indices are subtracted: e.g.

$$2^3/2^2 = 2^{(3 - 2)} = 2$$

QUESTIONS

Here are a few examples for you to try

$4^{\frac{1}{2}} * 4$, $2^4 * 8$, $9^{\emptyset.8}$, $\frac{27^{\frac{1}{3}}}{3}$

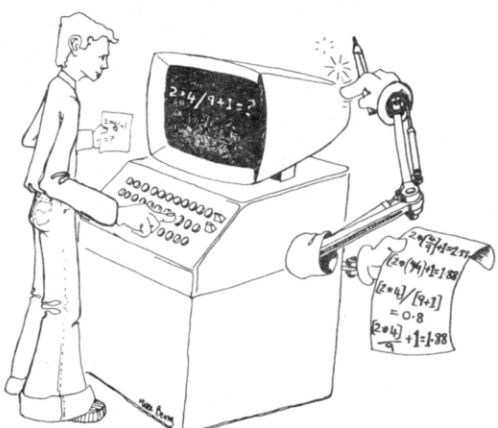
ANSWERS

$4^{1.5}$ or 8

2^7 or 128

$9^{\emptyset.5}$ or 3

27 or 3^3



LESSON 8

BRACKETS (or PARENTHESES)

In mathematical terms (2 + 3) x 4 is very different from 2 + (3 x 4).

If you fully understand the use of brackets in mathematical equations then you will have no problem with this calculation.

QUESTION

What is (((2 * 2/2) - 2)+2)/((2 + 2)/2)?

HINT

Remember that / is the division sign in BASIC so, for example, 1Ø/5 = 2.

ANSWER

2/2 = 1. If you find this a trivially easy problem, skip the remainder of this Lesson and go to Lesson 9.

In BASIC, mathematical equations (expressions as we shall soon call them) are written along a single line. This can lead to some confusion. For example, what is 2 * 3 + 4? Is it 1Ø or 14? It might mean 2 * 7 or it might mean 6 + 4.

Brackets, also called 'parentheses', are used to define the order in which the multiplications, divisions, additions and subtractions in a complex equation are carried out. The rule is simple. 'Always evaluate the expression within the brackets before continuing with the calculation.' Thus, if we had written 2 * (3 + 4) the answer would have been 14; if we had written (2 * 3) + 4 the answer would have been 1Ø. Here are some further examples:

(2 + 1) * (4 + 5) = 3 * 9 = 27

(4 + 2)/3 = 6/3 = 2

QUESTIONS

Try the following equations for yourself.

1) (3 + 2 + 1) / (1 + 1)

2) 2 / (1 + 3)

3) (6 + 8 - 2) * 2/6

ANSWER

1) 6/2 = 3

2) 2/4 = ½

3) 12 * 2/6 = 4

Using brackets within brackets is called *nesting*, and again the rule is simple: 'Always work from inner brackets through to outer brackets.'

Example: ((2 + 2) / 2) * 2

= (4/2) * 2 = 2 * 2 = 4

QUESTIONS

Here are three problems for you to try:

1) ((2 * 2 * (2 + 2)) / (2 + 2)) + 2

2) (4 + (3 - 2)) * (1 + 1)

3) (4 - (3 + 1)) * 1ØØ

ANSWERS

1) ((2 * 2 * 4) / 4) + 2 = (4 * 4/4) + 2 = (16/4) + 2 = 4 + 2 = 6

2) (4 + 1) * 2 = 5 * 2 = 1Ø

3) (4 - 4) * 1ØØ = Ø * 1ØØ = Ø

When preparing problems for yourself it is sometimes easier to use different shaped brackets to clarify the equation. For example, 1) looks better written as {[2 * 2 * (2 + 2)] / (2 + 2)} + 2

Unfortunately, in BASIC you are limited to the simple round brackets. and *be warned*, one of the most common errors in entering your program into the computer is forgetting to close brackets you have already opened!

There should be as many ((((as there are)))).

LESSON 9

EXPRESSIONS

In which you learn to write complicated mathematical problems, using brackets (parentheses), in BASIC.

An *expression* in mathematics is any set of mathematically meaningful symbols, letters and numbers. Equations and other formulae are expressions, and we use the word in this text to describe any set of mathematical symbols which you may encounter in BASIC statements.

Mathematical expressions in BASIC are similar in form to normal arithmetic notation, but since they are printed on typewriter-like terminals, they must occupy a single line. Arithmetic operations are performed using the five symbols tabulated below.

Operation	Symbol	Example	Meaning
Addition	+	X + Y	Add Y to X
Subtraction	-	X - Y	Subtract Y from X
Multiplication	*	X * Y	Multiply X by Y
Division	/	X / Y	Divide X by Y
Exponentiation	↑	X ↑ 5	Raise X to the fifth power (i.e. X^5)

Exponentiation (raising to the power) was introduced in Lesson 7, but, if you are not familiar with the concept, you will see the principle in the following examples.

In BASIC we can get the square of a number by typing, for example

```
PRINT 3*3
```

This is tedious with long numbers or large powers and impossible with fractional powers. An easier way to square long numbers is to use the exponentiation sign

```
PRINT 231.4567↑2
```

For a square root we can write

```
PRINT 4.321↑.5
```

However BASIC provides a special function for square root (because it is so common), so

```
PRINT       SQR(4.321)
```

has the same effect. (Note that 4.321 is in brackets.)

In fact, the computer will calculate 3*3 to a greater accuracy than 3+2. *(Note for mathematically-minded readers. The computer carries out exponentiation using the logarithm of 3, and calculation of the logarithm of 3 introduces some inaccuracy into the overall calculation. Similarly, because it is such a common calculation, the computer may use a special routine to calculate square roots, resulting in a more accurate answer when compared with exponentiation, and also a slightly faster execution time.)*

When expressions are written on single lines there are frequently ambiguities which can lead to mistakes. For example, the expression $\frac{X + Y}{2}$ may be written as X + Y/2 on a single line.
However, the computer will interpret this as X + ½Y and the result will be incorrect. The computer applies a fixed set of rules in interpreting expressions in BASIC. The programmer must therefore know and understand these rules in order to produce the correct results from a program.

Brackets - or parentheses are usually necessary to resolve ambiguities so that the computer actually calculates the sum the way you intended.

A good programmer uses parentheses liberally, even if they are not strictly necessary. This avoids errors introduced by 'clever' programming, and helps both programmer and other users to understand the program.

The rules which the computer uses to establish the method of calculation are:

- evaluate all expressions inside parentheses
- within parentheses, or in the absence of parentheses, the order of calculation is
    1) - exponentiation and functions (e.g. square roots)
    2) - multiplication or division (equal priority)
    3) - addition or subtraction (equal priority)
- for equal-priority operations, the order is simply from left to right as one would read the expressions.

Let's take an example. Here is a complicated expression:

```
PRINT  2↑3*4 - 3/2 + 3↑2 + 4*SQR(25)/5
```

Step 1 8*4 - 3/2 + 9 + 4*5/5 (↑ and SQR have been evaluated)

Step 2 32 - 1.5 + 9 + 4 (* and / have been evaluated)

Step 3 43.5 (+ and - have been evaluated)

(Note for purists: BASIC adopts a slightly different approach to arrive at the same answer.)

PROBLEMS

Try the following expressions for yourself

1) PRINT 2*3+4↑.5

2) PRINT 3+4↑2*3

3) PRINT 6/3/2-2

4) PRINT 2+4/2+1

ANSWERS

1) 8 or (2*3) + 2 = 6 + 2 = 8

2) 51 or 3 + (16 x 3) = 3 + 48 = 51

3) -1 or 2/2 - 2 = 1 - 2 = -1

4) 5 or 2 + 2 + 1 = 5

PROBLEMS

Now reverse the process and translate the formulae below into BASIC statements

1) $X = \frac{A}{B + C}$

2) $Y = \frac{AB}{CD}$

3) Z = X - (Y - D)

4) $C = AB^2$

Note

Use the LET statement

ANSWERS

1) LET X=A/(B+C)

2) LET Y=(A*B)/(C*D) or LET Y=A*B/C/D

3) LET Z=X-(Y-D) or LET Z=X-Y+D

4) LET C=A*B↑2

Here are some much harder problems.

PROBLEMS

Translate the following formulae into BASIC using the rules given above.

1) $A = \frac{(X+Y)^2}{Z}$

2) $X = \frac{A + B}{CD}$

3) $Y = \frac{\sqrt{-B+ B^2 - 4AC}}{2A}$

ANSWERS

1) LET A = (X+Y)↑2/Z or LET A = (X+Y)*(X+Y)/Z

2) LET X = (A+B)/(C*D) or LET X = (A+B)/C/D

3) LET Y = (-B+(B*B-4*A*C)↑Ø.5)/(2*A) or LET Y = (-B+(B↑2-4*A*C)↑Ø.5)/2/A
or SQR(B*B-4*A*C)/(2*A)

Problem (2) above demonstrates a frequent error in computer programming since it is very easy to write (A + B) / C * D which will be interpreted: $\frac{(A+B)D}{C}$ which is incorrect. Note that the use of brackets as in (A + B)/(C * D) helps to understand the formula.

Now you should be beginning to appreciate how pedantic the computer is; it has no judgement at all. Make a silly error, and it will either follow it blindly, or kick out the whole program and print out some cryptic error message, (more later about error messages).

So you have to learn the computer's rules and the next two lessons define the rules for handling numbers and assigning variable names in BASIC. Fortunately the rules are not too difficult to remember.

LESSON 10

LARGE AND SMALL NUMBERS

In which you learn about 2.46 x $1Ø^{9}$, and 3.2 x $1Ø^{-6}$, and so forth

QUESTIONS

Do you understand the following notation for very small and very large numbers?

$0.132*1Ø^{-6}$

$1.4657*1Ø^{9}$

Write out these numbers in full.

ANSWERS

Ø.ØØØØØØ132

14657ØØØØØ

If you find this type of problem trivial or easy you may skip the rest of this lesson, and go straight on to Lesson 11 on page 38.

The exponential notation, which is used to represent very large and very small numbers, relies on the mathematical principles which were introduced in Lesson 7 on 'raising to a power'.

Let's look at some powers of 1Ø.

1Ø*1Ø	=	1ØØ	= $1Ø^{2}$
1Ø*1Ø*1Ø	=	1ØØØ	= $1Ø^{3}$
1Ø*1Ø*1Ø*1Ø	=	1ØØØØ	= $1Ø^{4}$
1Ø*1Ø*1Ø*1Ø*1Ø	=	1ØØØØØ	= $1Ø^{5}$

QUESTION

What is $1Ø^{1}$?

ANSWER

$1Ø^{1}$ = 1Ø

That should be obvious from the example above. The power to which 1Ø is raised (sometimes called the index) is simply the number of Øs on the left hand side.

QUESTIONS

What are the results of the following calculations expressed in the $1\emptyset^n$ form?

$1\emptyset\emptyset\emptyset * 1\emptyset\emptyset\emptyset\emptyset = ?$

$1\emptyset * 1 * 1\emptyset\emptyset * 1\emptyset\emptyset = ?$

$1\emptyset / (1\emptyset\emptyset\emptyset * 1\emptyset) = ?$

ANSWERS

$1\emptyset^8$

$1\emptyset^5$

$1\emptyset / 1\emptyset^4 = 1\emptyset^{-3}$

The negative index means

$$1\emptyset^{-n} = \frac{1}{1\emptyset^n}$$

so we have a whole new series of numbers;

1/1Ø	= Ø.1	$= 1\emptyset^{-1}$
1/1ØØ	= Ø.Ø1	$= 1\emptyset^{-2}$
1/1ØØØ	= Ø.ØØ1	$= 1\emptyset^{-3}$
1/1ØØØØ	= Ø.ØØØ1	$= 1\emptyset^{-4}$

QUESTION

Now $1\emptyset^1 = 1\emptyset$, and $1\emptyset^{-1} = \emptyset.1$, so what is $1\emptyset^{\emptyset}$ .....?

HINT

The answer is not zero. We have already explained that

$1\emptyset^1 * 1\emptyset^2 = 1\emptyset^{1+2} = 1\emptyset^3 = 1\emptyset\emptyset$ You can use the same sort of rule for

$1\emptyset^2 / 1\emptyset^1 = 1\emptyset^{2-1} = 1\emptyset^1 = 1\emptyset$ You can also use it for $1\emptyset^1 / 1\emptyset^1 = 1\emptyset^{1-1} = 1\emptyset^{\emptyset}$

ANSWER

$1\emptyset^{\emptyset} = 1$

Any number positive or negative is equal to 1 when raised to the power zero. Using these powers of 10 we can write very large and very small numbers quite conveniently. Here are a few examples;

$$9876543.21 = 9.87654321 * 10^{6}$$

$$210000000 = 2.1 * 10^{9}$$

$$0.000000004 = 0.4 * 10^{-8}$$

We can also multiply and divide large numbers by adding and subtracting the indices.

$$3.4*10^{6} * 2.2*10^{3} = 3.4*2.2*10^{(6+3)} = 3.4*2.2*10^{9}$$

$$1.1*10^{5}/3.2*10^{4} = (1.1/3.2)*10^{(5-4)} = (1.1/3.2)*10^{1}$$

QUESTION

Here are a few large, small, or awkward numbers. Convert them to exponential notation.

0.0076	123.456	100000000
0.000021	4.565	1000000.12

HINT

Each number has an imaginary 10^{0} at the end. Each time you move the decimal point to the left, add one to the index. Move the decimal point to the right and you subtract one from the index. Keep moving the decimal point until it is in the right place.

e.g. $1000 * 10^{0}$

$100.0*10^{1}$

$10.00*10^{2}$

$1.000*10^{3} = 10^{3}$

ANSWERS

$7.6*10^{-4}$	$1.23456*10^{2}$	10^{8}
$2.1*10^{-7}$	4.565	$1.00000012*10^{6}$

Note that where there are a lot of significant digits, as in the last example, there is little economy in writing the number in exponential notation.

LESSON 11

NUMBERS IN BASIC

How to write numbers such as 2.46 x 1Ø^9 in BASIC.

BASIC uses a notation called the exponential notation or E notation. In BASIC the number 1ØØØØØ would be written as 1E5 which is BASIC's way of writing $1Ø^5$.

You may remember from Lesson 6 that $1Ø^5$ is also 1Ø↑5 in BASIC. So 1E5 = 1Ø↑5.
The number 1ØØØØØ can also be written as

```
1E5
1ØE4
1ØØE3
1ØØØE2
1ØØØØE1
```

but it cannot be written as E5, because the E must always be preceded by a number.

Similarly 1E-5 means

```
1 x 1Ø^-5
or 1/1ØØØØØ
or .ØØØØ1 (notice that there are four zeros, not five.)
```

Again E-5 is not sufficient - the E must be preceded by a number.

```
1E-5
.1E-4
.Ø1E-3
.ØØ1E-2
.ØØØ1E-1
or .ØØØØ1
```

So now we can see the rules for writing numbers in BASIC.

The following numbers are *legal:*

```
15    -Ø.2Ø3      +4592.333      .ØØØ123456
```

Numbers in BASIC may be positive or negative, and the + sign is optional. Very large or very small numbers are handled more conveniently in BASIC if they are scaled by powers of 1Ø. This is achieved by following a number with the letter E (for exponential) and an *integer* which is the appropriate power of 1Ø.

For example:

$1Ø^{6}$	can be written as	1E6 or 1ØE5
1ØØØØØØØ	can be written as	1E7 or 1E+7 *but not* E7 or 1ØE6 or 1ØØE5 etc.
Ø.ØØØ134	can be written as	1.34E-4 or 134E-6
$-2.17 \times 1Ø^{-15}$	can be written as	-2.17E-15

QUESTIONS

1) Which of the following numbers are illegal in the BASIC language and why?
Write the legal numbers in full, rather than in BASIC E notation.

a) 16.1 b) -23179 c) Ø.ØØØØ49 d) -Ø.279

e) 45.92E-6 f) -6129.452E1Ø g) E4 h) 1E4.2

2) Write the following numbers in BASIC exponential notation:

a) 1Ø452Ø.1 b) -Ø.ØØØØØ5214 c) 1ØØ.234

ANSWERS

1) Illegal numbers:

g) E4 E must be preceded by a number

h) 1E4.2 E must be followed by an integer

Legal numbers in full:

e) 45.92E-6 = Ø.ØØØ4592

f) -6129.452E1Ø = 61294520000000

2)

a) 1Ø452Ø.1 = 1.Ø452Ø1E5

b) -Ø.ØØØØØ5214 = 5214E-6 or Ø.5214E-5

c) 1ØØ.234 = 1.ØØ234E2

Variables, just like numbers and formulae, must be defined according to the rules of BASIC. These rules are introduced in Lesson 12.

LESSON 12

VARIABLE-NAMES IN BASIC

A variable-name might be C for the cash in my pocket, B1 the number of buttons in the sewing basket. But it has to follow the rules of BASIC.

Variable-names in BASIC, as in algebra, are symbols which represent numbers - usually those numbers whose values are unknown when the program is written, or whose values will change during the program execution.

A variable-name in BASIC is denoted by a *single letter*, or a *single letter followed by a single digit*. Legal variable-names, all of which could appear in a single program, are C, OØ, X2, Y9, K and so on. Examples of illegal variable-names are 3A, -K, 26, AB, C39.

Values are assigned to variable-names in certain BASIC statements, for example, by the LET statement, and the value assigned remains unchanged until a new value is assigned by another BASIC statement. In some computer systems, the values of all the variable-names are set to zero before the execution of the program - but it is good programming practice not to take the initial value of the variable-name for granted.

QUESTION

1) Which of the following variable-names are illegal in BASIC, and why?

   A, -3B, B3, -A2, X12,

   -Q, Y6, E4, 21

2) What is the maximum possible number of variable-names in a BASIC program?

ANSWERS

1) -3B Variable-names *cannot* be negative.

   -A2 Variable-names *cannot* be negative.

   X12 A single letter must be followed by a single digit.

   -Q A single letter must be followed by a single digit.

   21 A single letter must be followed by a single digit.

2) There are 286 variable-names in BASIC

   A - Z

   AØ - ZØ - people often forget the Ø is one of the *ten* digits.
   to
   A9 - Z9

LESSON 13

THE LET STATEMENT

When you write LET Y = Y + 1 you don't break the rules of BASIC, because in this context = means "becomes" rather than "equals" - a vital difference for all BASIC programs.

We introduced the LET statement in the example in Lesson 4.
The program is repeated below for reference.

```
1Ø READ A,B,X
2Ø LET Y=A*X/4+B
3Ø PRINT "Y EQUALS",Y
4Ø DATA 4.2,Ø.5,2.7
5Ø END
```

The LET statement is the principal means of processing data in a BASIC program. In this example it causes the computer to evaluate the expression $\frac{AX}{4} + B$ and to store the result in a location in the computer which is labelled Y. Note that 'LET' is a *verb* which represents a command to the computer. All executable *BASIC* statements begin with a verb which indicates the action by the computer.

The *equals* sign in the LET statement does not have the same meaning as an equals sign in an algebraic equation. For example the following LET statements are perfectly acceptable:

```
LET Y = Y+4

LET Z = B-Z
```

QUESTION

What do these two statements do?

ANSWER

The first statement is interpreted by the computer as: "replace the old value of Y with a new value equal to the old value plus 4"; similarly the second statement is interpreted as: "replace the old value of Z with a new value equal to the value of B minus the old value of Z", or "assign to variable-name Y the old value plus 4", and "assign to variable-name Z a value equal to the B minus the old value of Z".

For this reason the LET statement is often called the *assignment* statement.

QUESTION

How will the computer interpret the following BASIC statements?

```
LET P = (Q-R)/2

LET Z = A+Z*2
```

ANSWER

Assign to variable-name P a value given by the value of Q minus the value of R all divided by 2, or replace the value of P by the value of Q minus the value of R all divided by 2.

Assign to variable-name Z a value equal to the value of A plus twice the old value of Z or replace the old value of Z with the value of A plus twice the old value of Z.

Go straight on to Lesson 14 which tells you some more about the PRINT statement.

LESSON 14

THE PRINT STATEMENT

You have used PRINT already. Now you see how to PRINT words as well as numbers, and how to PRINT your results in five columns across the page.

We have already included the PRINT statement in program examples. We are now going to see what it means to the computer, and how it can be used to set out the results of calculations.

The simplest PRINT statement concerns only a single variable-name or result. For example, the statement

```
5Ø PRINT F
```

will cause the computer to print the *value* that has been assigned to the variable-name F on a teletypewriter or line printer, and will advance the carriage to a new line. PRINT will also cause the value of F to be displayed on a video display unit if that is the output device you are using.

The item to be printed may also be specified by a function or formula, e.g.:

```
6Ø PRINT (X+Y)↑4

7Ø PRINT SQR(Y)
```

An important requirement in using a computer is the facility for *labelling* the results of calculations clearly - in BASIC, the PRINT statement can also be used for text allowing labels, titles, headings and the like to be printed, which vastly improves the presentation of the computer output. Text is printed using a statement of the following form:

```
7Ø PRINT "COST IN POUNDS STERLING"
```

The text inside the quotation marks will be printed unmodified by the computer - and the printer will be advanced to a new line.

The PRINT statement may be used for multiple items or labels - by separating each item or label in the PRINT statement with a comma. Two examples of multiple item PRINT statements are given on the next page.

a)

```
11Ø READ A,B,C,D,E
12Ø PRINT A,B,C,D,E
13Ø DATA 123.4,-4,.ØØØØ232,1ØØØØØ,-2.456
```

b)

```
14Ø READ A,B
15Ø PRINT "A=",A,"B=",B
16Ø DATA 123.4,-4
```

To interpret a multiple-item PRINT statement, the computer divides the sweep of the teletypewriter carriage into five zones of 14 or 15 spaces each. (The precise width of a zone varies in different computer systems, in our examples the computer uses a 14 zone on the printed line.) The statements given above would generate a print-out of the following *format*:

```
a)
   this is A     this is B     this is C     this is D     this is E
 123.4           -4            .ØØØØ232      1ØØØØØ       -2.456

b)
 A =             123.4         B=            -4
    this is a label
```

Note that where a number is printed, the first space in the zone is reserved for the sign of the number. The zones are shown here by dotted lines down the page. These dotted lines will not be printed by any computer, but they help you to see the 5-zone lay-out.

There is no limit to the number of items in the PRINT statement; the excess items will be printed on new lines. This is shown in the following example.

```
14Ø READ A,B
15Ø PRINT "A =",A,"B =",B,"SUM =",A+B
16Ø DATA 12,42

A =             12            B =             42            SUM =
54
    this is the sum of A and B
```

(It is obviously necessary to remember this characteristic when arranging a format for one's results.)

LESSON 15

ERRORS IN BASIC

Your mistakes might be picked up by the computer and rejected. Or you may write a program that happens to be quite the wrong sum, because you forgot how the computer would understand your words. The computer then does what you tell *it to do, not what you* want *it to do.*

With all these rules and regulations, it is inevitable that you will make mistakes when you program in BASIC. All programmers make mistakes, but careful programmers make fewer mistakes than others.

The computer will almost always spot simple mistakes. For example you might forget to type a line-number, or commas, or punctuation marks. You might (and frequently will) mistype a character by hitting the wrong key, or type the wrong character by forgetting that 0 is not the same as Ø or that 1 is not I.

This type of mistake is called a *syntax error*. Syntax errors are usually trivial mistakes which often come about during the process of entering the program into the computer. As we have said the computer will usually spot such errors. Here are a few examples.

```
10 READ A,B
ERROR Ø2 - SYNTAX    (The computer says there is an error)
```

QUESTION
Can you see the error.....?
HINT
Zero is Ø
ANSWER
The correct line should be 1Ø READ A,B

When the computer spots a syntax error, it prints out a message to help you find it, and takes no action on the line typed in.

PROBLEMS

Here are a few more examples containing syntax errors - can you find them:

a)
```
1Ø READ Q,R;S
2Ø PRINT Q+R+S
3Ø DATA 9,7,5
4Ø END
```

b)
```
10 READ X,Y
20 PRINT X*Y
30 DATA 4,3
40 END
```

c)
```
1Ø REED F,J
2Ø PRINT F+J
3Ø DATA 3,4
4Ø END
```

d)
```
1Ø READ, A,B,C
2Ø PRINT A+B*C
3Ø DATA 3,6,21
4Ø END
```

e)
```
1Ø READ A,B,C
2Ø PRINT BC+A
3Ø DATA 4,3,3
4Ø END
```

f)
```
READ X,Y
PRINT X+Y
DATA 42,34
END
```

g)
```
1Ø READ A,Z
2Ø END
3Ø PRINT A+Z
4Ø DATA 2;4
```

h)
```
1Ø READ A,B
2Ø PRINT A+B
3Ø DATA 4,3
4Ø FINISH
```

HINT

a) Line 1Ø

b) All lines

c) Spelling mistake

d) Punctuation

e) Something is missing in line 2Ø

f) Many omissions

g) Is something in the wrong place?

h) New word?

ANSWERS

a) The semicolon should be a comma in line 1Ø

b) O should be Ø in each line number

c) REED should be READ in line 1Ø

d) Omit comma in line 1Ø

e) 2Ø should be B+C+A or B*C+A or B-C+A or B/C+A

f) Line numbers are needed

g) Line 4Ø has a semi-colon instead of a comma.
Line 2Ø says END but line 3Ø is needed.
(Line 4Ø is non-executable and would be accepted after END.)
However, it should read:
```
1Ø READ A,Z
2Ø PRINT A+Z
3Ø DATA 2,4
4Ø END
```

h) FINISH is not a BASIC statement, line 4Ø should be END.

Syntax errors are the least of our problems in computer programming - the computer can help us with these. Much more serious are mistakes which are called *run-time errors* which are not detected by the computer before execution of the program but lead to incorrect results.

QUESTION

Here is a program to compute the average of three numbers X,Y and Z. What is wrong with the program?

```
1Ø READ X,Y,Z
2Ø PRINT "AVERAGE EQUALS",X+Y+Z/3
3Ø DATA 4,8,12
4Ø END
```

HINT

Below we show the result of executing this program, look at the answer:

```
RUN

AVERG              31-MAY-99              19:34:53

AVERAGE EQUALS  16
END AT 4Ø
READY
```

ANSWER

Line 2Ø should be:

```
2Ø PRINT "AVERAGE EQUALS", (X+Y+Z)/3
```

Now if we execute the program the print-out would be:

```
AVERAGE EQUALS 8
```

which is the correct result.

So we must not assume that if the computer accepts the program and executes it, then the program is free of mistakes. It is in fact free of trivial syntax errors but may produce totally incorrect results.

Later in this course we will show you how to check your programs for the presence of run-time errors, but for the moment you should simply try to remember the distinction between syntax and run-time errors.

SYNTAX ERRORS - GRAMMAR AND PUNCTUATION

RUN-TIME ERRORS - NOT DETECTED BEFORE PROGRAM IS EXECUTED.

LESSON 16

USING WHAT YOU KNOW

You now know enough to use a computer in earnest, for calculating the estimates needed by a carpenter.

So far you have learnt a few simple BASIC statements, LET, READ, DATA, PRINT, END and you have learnt some rules which the computer uses to understand variables, numbers and formulae. You can use these to solve practical problems.

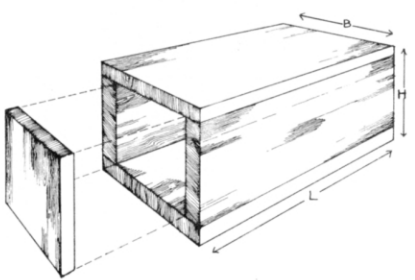
Suppose you are a carpenter who makes boxes, and you are often asked to quote your price for making many sorts of box.

The carpenter uses a standard procedure for estimating the price of a box:

- WOOD size the box and work out the area of wood in it. Wood costs 2.3Ø icu* per square metre.
- SAWING sawing takes 1Ø minutes/metre and the carpenter's time is charged at 3 icu per hour.
- NAILS nailing also takes about 1Ø minutes/metre with 1Ø nails used on each edge of the box. Nails cost 1 icu per 1ØØ.

This is a simple calculation and a simple computer program will do it.

Here is the box:



* This book is read worldwide. We therefore use the International Currency Unit (icu). There are 1ØØ pennies in an icu.

QUESTION

What are the expressions for:

- A, the area of wood,

- P, the price of wood,

- N, the cost of nails and nailing, and

- S, the cost of sawing?

ANSWER

A = 2 * (B * H + L * H + L * B) - *ignoring the thickness of the wood*

P = 2.3 * A

N = Ø.6 * 4 * (L + B + H) - *since nails and nailing cost Ø.6 icu per metre*

S = Ø.5 * 4 * (L + B + H) - *the length of the edges of the box is 4 times L+B+H, but each edge of the box consists of two sawn edges joined together. But when you saw, you cut between two usable pieces - so a 2m cut produces 4m of sawn edge.*

PROBLEM

Now you can write the computer program; try writing it yourself before examining the program below. Assume a box is 1 metre in each direction.

ANSWER

```
1Ø READ L,B,H
2Ø PRINT "LENGTH IN METRES",L
3Ø PRINT "BREADTH IN METRES",B
4Ø PRINT "HEIGHT IN METRES",H
5Ø LET A=2*(B*H+L*H+B*L)
6Ø PRINT "AREA OF WOOD",A
7Ø PRINT "PRICE OF WOOD",2.3*A
8Ø LET N=.6*4*(L+B+H)
9Ø PRINT "COST OF NAILING AND NAILS",N
1ØØ LET S=.5*4*(L+B+H)
11Ø PRINT "COST OF SAWING",S
12Ø PRINT "TOTAL COST OF BOX IS",N+S+2.3*A
13Ø DATA 1,1,1
14Ø END
```

Running the program produces the print-out shown below.

```
READY
RUN
LENGTH IN METRES                1
BREADTH IN METRES               1
HEIGHT IN METRES                1
AREA OF WOOD    6
PRICE OF WOOD   13.8
COST OF NAILS AND NAILING       7.2
COST OF SAWING                  6
TOTAL COST OF BOX IS            27

END AT 14Ø
READY
```

You might think that this program is too simple. It does not tell you how many nails to buy. It does not make any allowance for wasted wood. It also ignores the fact that the ends fit inside the top, bottom and sides, so that the true area of wood has to make allowance for the thickness of wood. All these modifications and improvements can be made to the program. Nevertheless the principle of the program will remain essentially the same, and the job it's designed for, estimating the cost of wooden boxes, will not change.

If you managed the simple program above, you will now be able to write programs for a wide range of practical problems. In the next part of this course we introduce many more BASIC statements which will widen the range of problems you can solve. Before you move on to the next part, you should try the problems set in the next lesson.

If you haven't read Supplementary Lessons 1,2 and 3 on using the computer terminal, then you will benefit now from these lessons even if you are not about to use the computer.

LESSON 17

ANOTHER PROBLEM

You can progress from the carpentry shop to the baker's oven. We hope you could start to deal with almost any repetitive estimates using simple costs of time and materials.

A baker wishes to know the cost of preparing various cakes and pastries. The main ingredients are always the same:

Eggs
Flour
Sugar
Milk
Margarine

Now the prices of these basic ingredients vary each week, but the recipes and the quantity of cakes and pastries baked remains constant. The baker needs a program which computes the total cost of ingredients and the cost per item, for the following daily bake.

SPONGE CAKE: 1 DOZEN

24 Eggs
3 lb Flour
3 lb Sugar
3 lb Margarine

SCONES: 4 DOZEN

2 lb Flour
½ lb Margarine
1 pt Milk

PASTRY: to make 5Ø tarts

2 lb Flour
1 lb Margarine
½ pt Milk

The baker buys flour in 56 lb bags, eggs by the dozen, sugar by the stone (14 lbs), and margarine by the 28 lb bag, and milk by the gallon (8 pints). On one week the prices were as follows:

Eggs Ø.5Ø icu per dozen
Flour 6.8Ø icu per 56 lbs.
Sugar 2.5Ø icu per stone
Milk Ø.8Ø icu per gallon
Margarine 8.ØØ icu per 28 lbs.

PROBLEM

Write a BASIC program to compute total cost of ingredients and cost per item.

```
SOLUTION

1Ø READ E,F,S,M1,M2
2Ø DATA .5,6.8,2.5,.8,8
3Ø PRINT "THIS WEEKS PRICES"
4Ø PRINT "EGGS PER DOZEN",E
5Ø PRINT "FLOUR PER 1/2 CWT",F
6Ø PRINT "SUGAR PER 14LB",S
7Ø PRINT "MILK PER GALLON",M1
8Ø PRINT "MARGE PER 1/4 CWT",M2
9Ø LET C=2*E+3*F/56+3*S/14+3*M2/28
1ØØ PRINT "SPONGE CAKES COST",C/12
11Ø LET S1=2*F/56+.5*M2/28+M1/8
12Ø PRINT "SCONES COST EACH",S1/48
13Ø LET P=2*F/56+M2/28+M1*.5/8
14Ø PRINT "PASTRY PER TART COSTS",P/5Ø
15Ø PRINT "TOTAL COST OF INGREDIENTS IS",C+S1+P
16Ø END

READY
RUN
THIS WEEKS PRICES
EGGS PER DOZEN                .5
FLOUR PER 1/2 CWT             6.8
SUGAR PER 14LB                2.5
MILK PER GALLON               .8
MARGE PER 1/4 CWT             8
SPONGE CAKES COST             .229762
SCONES COST EACH              .Ø1Ø119
PASTRY PER TART COSTS         .Ø11571
TOTAL COST OF INGREDIENTS IS                3.82143

END AT 16Ø
READY
```

Here are some notes on the program, to explain it:

LINE

1Ø READ all the costs
2Ø Feed in all the cost DATA
3Ø to 8Ø The computer prints out all the costs so we can check that our DATA was accurately entered.
9Ø C is the cost of a dozen sponge cakes. Note the way we calculate the cost of Eggs, Flour, Sugar and Margarine. For example Eggs cost E per dozen; so 2 dozen eggs cost 2*E, Flour costs F per 56 lbs, so 3 lbs of flour cost 3*F/56, and so forth.

1ØØ The cost of each sponge is C/12. This is PRINTed.
11Ø and 14Ø are similar to 9Ø, but for S1 (scones) and P (pastry used in tarts) respectively
12Ø and 14Ø are similar to 1ØØ, but PRINT is for the cost of scones and tart pastry
15Ø This PRINTs the total cost of 12 sponge cakes, 48 scones, and 5Ø tart-pastries.

SUPPLEMENTARY LESSON 1

GETTING STARTED

How you log-on to a computer, use a terminal keyboard, and delete mistakes.

There are many different types of computer. We can't tell you how to use all the different types of machine you may meet in your career as a computer programmer. Instead we will show you how to access a computer through a time-sharing bureau. This is rather similar to the logging-in procedure used by all on-line computers, but you will have to check in the operator's manual before trying for yourself.

First of all, make sure everything is plugged in and the power is switched on. With a time-sharing terminal, you will have in front of you a teletype or VDU, a modem, and a telephone. Make sure your teletype or terminal is switched to LINE or FULL DUPLEX. This allows the computer to control the printer independently of the keyboard. The teletype is used in LOCAL or HALF-DUPLEX for off-line preparation of programs in a batch-processing system.

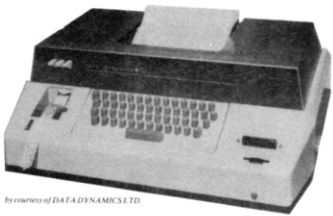
Now you must establish a telephone link to the computer along which the data is transmitted. The computer has its own telephone number - dial this and the computer will answer with a high-pitched continuous tone - quite unlike anything you would normally hear on the telephone. To establish a connection you press the button marked DATA on the top of the telephone and the terminal will hiccough to indicate a connection has been made. Don't put the handset back on its rest - this will clear down the line and you will have to redial.

The next step is to wake up the computer. This varies according to the computer. Sometimes pressing the RETURN key once or twice does the trick. On other machines you have to press the ESC key - 'Escape' which is located on the left-hand side of the keyboard.

The computer will type back a message identifying itself, telling you the date and time and then asking you for your USER IDentification, i.e. it prints: USER ID. Usually this is a numeric code which identifies you to the computer; your computer time will be charged against the account identified by this number, and all your programs will be referenced within the computer by this number. It's important that you get it right. Just to make sure, most commercial computer systems will also ask for a password, which is a jumble of 4 or 5 letters. To keep the password secret, it is not typed on the print-out, but you have to press the right keys anyway. Make a mistake and the computer will let you try again. Fail three times and your telephone line will be disconnected. This precaution is to prevent unauthorised users accessing your computer programs, and charging the computer time to you. Security measures like this are very important when confidential business data is being processed.

Now you are connected to the computer. The executive is in charge, and it is waiting for your first command. As we explained in Lesson 4 you now type RUN BASIC and the executive prepares the computer for your BASIC program - be it OLD or NEW.

OK - the computer is on and BASIC programs can be entered and executed. To enter your program you will have to become very familiar with the terminal keyboard. The picture below shows a typical computer terminal keyboard.



by courtesy of DATA DYNAMICS LTD.

This type of keyboard is called TRI-MODE, because each key has up to three meanings. Three? Yes, a normal typewriter keyboard has a SHIFT key which selects either upper or lower case characters. The computer terminal also has a SHIFT key which functions as on a normal typewriter, although the printer on the computer uses capital letters only so the shift key has no effect on the letters on the keyboard. But the computer terminal also has a CONTROL key which selects a third mode on the keyboard. The control-functions are not marked on the key-tops because there is no space; also the functions of the control keys vary from computer to computer. Thus pressing the CONTROL key and the letter U at the same time may delete the line of the text currently being entered. CONTROL and C may cause the computer to stop its current program and return control to the executive. All these functions vary, you will have to learn the set of codes used on your particular computer.

We mentioned the line-delete function in the discussion above. Let us look in more detail how this works. Remember that the control code may be different on your computer, but we will use CONTROL/U in this example.

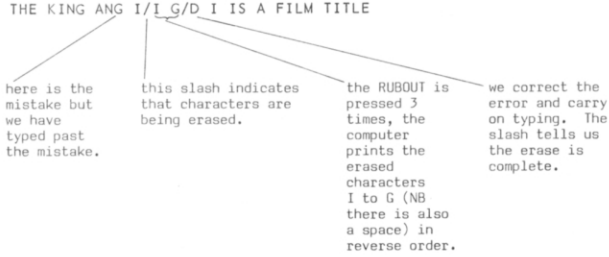
Suppose we are typing the program example used in Lesson 4. Here is the print-out, we have marked the carriage returns for clarity

```
10 READ A,B,X          (RET)
20 LET Y = A*X/4+B     (RET)
30 PRONT "Y EQU
```

Anybody can make a typing mistake. To delete the line we simply press the CONTROL key and holding the CONTROL down press the letter U. The computer will delete the line from its memory and returns the carriage of the printer for a new line. We can delete any line of program, data or even a command in this way provided we cancel the line before the RETURN key is pressed.

Please bear in mind that you may use a different key to delete a line. This example simply shows you how the control key is used on certain computers. Some computers use \, which is typed by pressing "shift" then pressing L, to erase a line. You will have to discover, by asking, or by reading the manual, how your computer erases a line.

We can also delete individual characters using the RUBOUT key. The RUBOUT key erases the last character typed, press the RUBOUT key twice and it will delete the last character and the last but one. And so on. The computer shows which characters are being erased by repeating them. Here is an example;



Some computers respond to the RUBOUT with a back-arrow to indicate the erased characters. Here is an example;

THE KING ANG I←←←D I IS A FILM TITLE

Here the computer does *not* repeat the letters which are being deleted. Again, this will depend on your computer.

The line-delete and character-delete functions have nothing to do with BASIC, they are provided by the executive which controls the computer. The executive takes the data from the keyboard, stores it until it has a complete line, and then decides what to do with it.

In the next supplementary lesson we describe some additional facilities provided by the BASIC language. Because these facilities are part of the BASIC language they should be available on almost all the computers you are likely to access, whereas the facilities described in this lesson, being functions of the machine executive, will vary from computer to computer.

Go straight on to SUPPLEMENTARY LESSON 2.

SUPPLEMENTARY LESSON 2

MORE BASIC COMMANDS

More facilities you will need when you are connected to a computer.

When you are connected on-line to the computer and BASIC is running, a number of additional facilities are available to you.

The most useful is the ability to replace whole lines of the BASIC program.
Here is an example:

```
10 READ A,B
20 PRINT A+B
30 DATA 2,2
40 END
```

We have entered this program into the computer and executed it. We now wish to change the data; the DATA statement is line number 30. We simply retype the line:

```
30 DATA 4,3
```

The computer exchanges the new line for the previous line 30.

OK, what is the new program? Simply type LIST and the computer will print the complete new program.

```
LIST

SUM          31-MAY-99          19:34:53      SUM is the program name

10 READ A,B
20 PRINT A+B
30 DATA 4,3
40 END
READY
```

When you type the RUN command, the computer tells you the program title, the date, and the time. It also tells you when it is ready for a new command by printing READY. (Even the word READY may not always be used. Some computers print * whenever they are ready, including a * at the start of each line of the program you are entering. It depends on your computer dialect.) Therefore, if we type a line number and a new line it should change the old line and keep the number.

We can delete lines by typing just the line number, and add new lines or replace old lines by typing in lines with new numbers. For example:

```
22 PRINT A-B
2Ø
LIST

SUM           31-MAY-99       19:34:53

1Ø READ A,B
22 PRINT A-B
3Ø DATA 4,2
4Ø END
READY
```

Line number 2Ø has been deleted and line number 22 has been added. In this case it would have been easier to type 2Ø PRINT A-B to replace line 2Ø with our new line. We always number lines in tens so that new lines can be inserted if necessary.

If you want to delete a complete program or a block of consecutive lines then the DELETE command is used. This command deletes all lines between the two line numbers which follow it, including the lines themselves. A bit complicated; here is an example:

```
DELETE 1Ø,3Ø
READY

LIST          You type LIST to check that lines Ø to 3Ø have been deleted

SUM           31-MAY-99       19:34:53

4Ø END
READY
```

Lines 1Ø, 22 and 3Ø have been deleted. Be very careful when using the DELETE command.

All these changes have been to the program stored in the temporary memory of the computer. If you have previously stored the program SUM then the OLD program is still intact in the permanent memory, and we can recall it if necessary using the OLD command. Let's do that.

```
OLD
OLD PROGRAM NAME--SUM
READY

LIST

SUM           31-MAY-99       19:34:53
1Ø READ A,B
2Ø PRINT A+B
3Ø DATA 2,2
4Ø END
READY
```

Now we have our original program back.

More often we need to put our new copy of the program into permanent store and to delete the old copy. Two BASIC commands can be used. The UNSAVE command deletes the permanent copy of a program. Once you have deleted with the UNSAVE command you will no longer be able to recall the program using the OLD command.

```
UNSAVE SUM
READY
```

All gone.

The other BASIC command which deletes the permanent copy is the REPLACE command. Using REPLACE you delete the old copy of a program but replace it in permanent memory with the new copy which is stored in the temporary memory. The format is the same as the UNSAVE command.

We have covered a lot of facts in this lesson. Here is a summary of the BASIC features we have described.

NAMING, STORING AND DELETING PROGRAMS

NEW - this command tells the computer that you wish to enter a new program. You can use it as a response to the question "NEW OR OLD--" or by itself as a BASIC command. The computer will respond with a question "NEW PROGRAM NAME--".

SAVE 'name' - this command will save the named program stored in temporary memory in the computer permanent memory. It can then be recalled using the OLD command.

UNSAVE 'name' - the opposite of SAVE, this command deletes the named program stored in the permanent memory.

REPLACE 'name' - this command replaces the old copy of a program with the new copy stored in the temporary memory. Thus you can recall a stored program using the OLD command, modify it using the BASIC editing capability, and then replace the old copy with the updated new copy.

EDITING PROGRAMS IN TEMPORARY STORE

LIST - this command causes the computer to print the contents of the temporary store on the terminal. The title, date and time of the program are also printed.

LIST n,m - the computer will list from line number n to line number m inclusive.

DELETE n,m - the computer will delete line numbers n to m inclusive.

To add a line - simply type the line number, followed by the line.

To delete a line - just type the line number. The computer will delete the line.

Remember not to use consecutive line numbers, so that you can later insert new sections of program as required.

RUNNING AND STOPPING

RUN - this command causes the computer to execute the program in temporary store.

IMPORTANT IMPORTANT IMPORTANT

How do you stop the computer once it has started to execute your program? All computers are different, so you must find out before you use the machine. The computer could run up a bill of hundreds of pounds before you realised what was wrong and how to stop the machine. Typically the machine is stopped by pressing the ESC key on the terminal (Escape) or sometimes the combination CONTROL and C, or pressing the RET, or typing STOP.
Find out, don't take a chance.

Continue with Supplementary Lesson 3.

SUPPLEMENTARY LESSON 3

COMPUTERS AS CALCULATORS

You may want to use a computer for simple arithmetic. It is a waste of the computer's power, but here is how to do it.

In Lesson 4 and Supplementary Lesson 2 we have defined the distinction between BASIC commands such as RUN, LIST, SAVE, UNSAVE etc., and BASIC statements such as LET, PRINT, READ, DATA, and END. BASIC statements are always preceded by a line number which tells the computer to store the statement in a temporary store until the complete program is acted upon by a BASIC command.

BASIC statements can also be used as BASIC commands. Here are two examples:

1)

```
PRINT 3*4
12
```

The computer has printed the result of 3*4

2)

```
PRINT "RESULT IS",4+5+6
RESULT IS       15
```

With no line number the computer will attempt to execute the BASIC statement immediately; this is called the *immediate mode* of BASIC. With it you can use the computer as a calculator for simple sums. Of course there is little point using READ, LET or END, in the immediate mode because the computer won't print anything in response to these statements.

Normal programs of this type introduced in Lesson 3 and 4 are executed in the *deferred mode*; this definition distinguishes the use of line numbers and the RUN command from the immediate execution of the IMMEDIATE mode.

The immediate mode is made more powerful by the colon. Here is an example:

```
LET A=3*3:LET B=A*2:PRINT "B =",B
B=              18
```

The colon allows you to write several BASIC statements on a single line. In the immediate mode this is useful, since the statements are executed the instant you press the RETURN key.

The immediate mode, and the use of the colon, vary with different computer systems and you should check in the system manual, before attempting to use these facilities on your computer.

If you came from Lesson 5, continue with Lesson 6. You may find it useful to refer to these Supplementary Lessons if you run into any difficulties using the computer terminal.

If you came from Lesson 16, go back to Lesson 17.

SUMMARY of BASIC LANGUAGE

Lesson numbers printed in italics

STATEMENTS

		Lessons
DATA	DATA 4,3.12,-24	*4,25*
DEF FN	DEF FNA(N)=(1+3.14159*SIN(N))	*52*
DIM	DIM A(2Ø),B(1ØØ),C(Y+2*Z)	*42*
END	END	*4*
FILES	FILES NAMES	*6Ø*
FOR...TO...STEP	FOR N=1 TO 1Ø STEP 2	*36*
GOSUB	GOSUB 1ØØØ or GOSUB (X+Y)	*5Ø*
GO TO	GO TO 8Ø or GO TO (X+Y)	*3Ø*
IF...THEN...	IF X>=Y THEN 9Ø	*3Ø*
IF END	IF END # 1 GO TO 9Ø	*6Ø*
INPUT	INPUT A,B,C	*28*
INPUT (of a file)	INPUT # 1,L,N$(I),X$(I)	*6Ø*
LET	LET A=B+1Ø	*6,13*
MAT	MAT X=Y+Z	*57*
NEXT	NEXT N	*36*
ON...GO TO...	ON N GO TO 1ØØ,12Ø,13Ø	*57*
PRINT	PRINT, "LABEL";N,A	*4,14,29*
PRINT USING	PRINT USING 4Ø,X,Y	*57*
RANDOMISE	RANDOMISE	*46*
READ	READ A,B,C	*4,25*
REM	REM THIS IS A REMARK	*27*
RESTORE	RESTORE	*39*
RETURN	RETURN	*5Ø*
STOP	STOP	*31*

NUMBERS

123	-456	7.981	-Ø.ØØ4	1E1Ø	-1.4E-4
(4)	*(4)*	*(4)*	*(4)*	*(11)*	*(11)*

VARIABLE NAMES

A	Z1	B(1)	X(12Ø)	Z(I+4*J)	T(4,4Ø)	C$	D1$	Y$(15)
(12)	*(12)*	*(41)*	*(42)*	*(42)*	*(42)*	*(54)*	*(54)*	*(54)*

FUNCTIONS

ABS	ATN	COS	EXP	INT	LOG	RND	SGN	SIN	SQR	TAB	TAN
(24)	*(23)*	*(23)*	*(24)*	*(24)*	*(24)*	*(46)*	*(24)*	*(23)*	*(9)*	*(48)*	*(23)*

OPERATORS

( )	↑	*	/	+	-	:	=	>	<	>=	<=	<>
(8)	*(6)*	*(4)*	*(5)*	*(4)*	*(6)*	*(SL3)*	*(4,54)*	*3Ø & 54*				

COMMANDS

DELETE	KILL	LIST	NEW	OLD	REPLACE	RUN	SAVE	UNSAVE	\	←
(SL2)	*(5)*	*(SL2)*	*(5)*	*(5)*	*(SL2)*	*(5)*	*(5)*	*(SL2)*	*(SL1)*	*(SL1)*

Cambridge Learning Limited,
Rivermill Lodge,
St. Ives,
Huntingdon,
Cambs PE17 4EP.

ISBN 0 905946 05 7