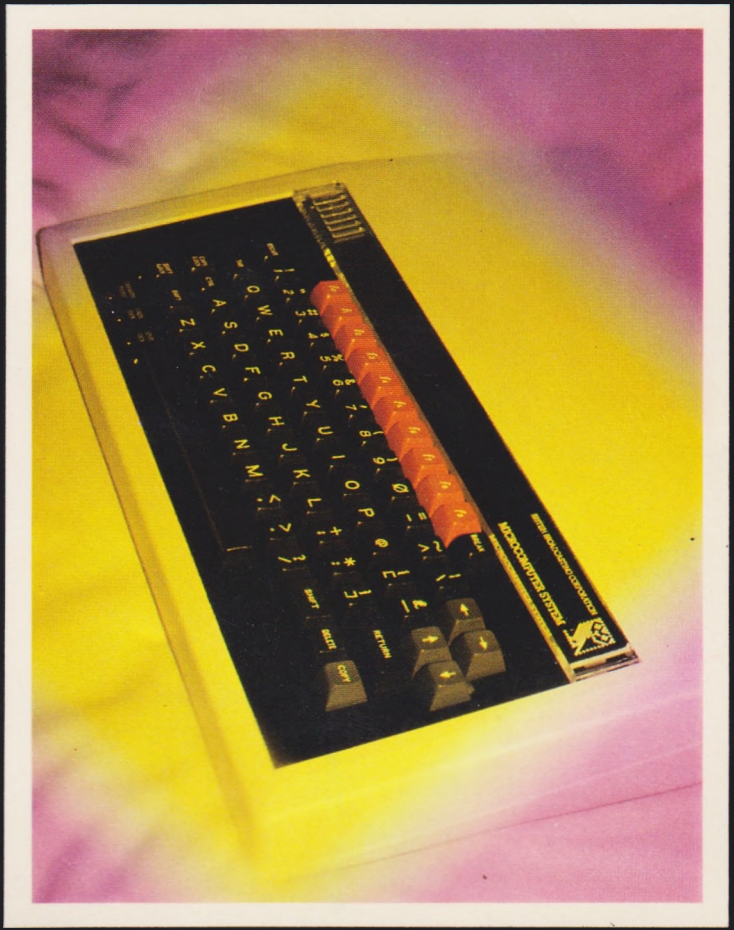
USING THE BBC MICRO IN EDUCATION



Don Thorpe

USING THE BBC MICRO IN EDUCATION

Don Thorpe

Interface Publications, London and Melbourne

First published in the UK by:
Interface Publications,
9-11 Kensington High Street,
London W8 5NP

ISBN 0 907563 87 2

Books published by Interface Publications are distributed in the UK by WHS Distributors, St. John's House, East Street, Leicester LE1 6NE (0533 551196) and in Australia and New Zealand by PITMAN PUBLISHING. Any queries regarding the contents of this volume should be directed by mail to Interface Publications, 9-11 Kensington High Street, London W8 5NP.

1st Printing - May 1984

Printed by J. W. Dunn (Printers) Ltd., Sutton, Surrey.

CONTENTS

INTRODUCTION

This book will help you in four ways, whatever your experience with computers in schools.

You can develop richer, more innovative and versatile educational programs.

You will be better able to help students face the exciting challenges of computing.

You will be able to help other teachers appreciate the power and importance of computers for their subject areas.

You will have more skills to apply to administration in school and classroom.

The book is divided into four clear sections which consider the following aspects of computing in schools.

What are the strengths and weaknesses of computers? What can be expected of educational computer programs? Why is the creative use of computers in education so important?

BBC BASIC is the "Rolls Royce" of the BASICs available to students. This book outlines how a solid foundation can be built even when there are wide variations in student experience. How can you teach students the facts about Keywords whilst capturing their imagination? Be aware of and know how to prevent commonly experienced difficulties. Helpful, innovative tables, charts and projects are provided.

Consider the factors of program design which will help you achieve educationally sound results. How do these vary for

group and individual learning situations? How can the very able students be extended and the strugglers assisted?

How do you cooperate with colleagues who know nothing about programming, to produce imaginative and effective programs? How can you produce programs to alleviate the boredom of routine school tasks?

I have written over sixty instructive programs especially for the BBC computer. Each listing has been reproduced directly from a working program. I have included four substantial programs, two for administration and two for the classroom. I have written material to which all students can relate with interest.

The world of computing in education is new and expanding rapidly. Amazing things are happening. We are fortunate to have the opportunity to join in.

Don Thorpe

SECTION I –

FIRST STEPS IN A NEW LAND

Teachers already have many aids to use in the classroom. Still the most important are the teacher's personality and voice. Next comes the traditional blackboard, a marvellously versatile and economic aid. After that, we have charts, audio tapes, slides, the overhead projector, film, videotape, and the physical paraphernalia peculiar to particular subjects.

What does the microcomputer have to offer the teacher? To some, a computer is a powerful de-humanising agent which is devilishly quick with figures. For most however, there is a recognition that a very powerful teaching aid has arrived.

A drawback with the traditional blackboard is that its contents cannot be saved until next week or next year. The effort must be duplicated each time. Many a teacher has been proud of a piece of blackboard work and has reluctantly erased it to make way for the oncoming tide of new material. A computerised "electronic blackboard" can be saved, stored away compactly, updated easily and reproduced any number of times. At present however, the screen is far smaller than a blackboard.

Younger children in particular find simple, bright and colourful animation very engrossing. This is reflected by the ratings success of the generally worthless offerings of cartoon television.

A teacher receives a film or videotape as a completed product. A computer program however can be cheaply adapted and improved practically without limit.

A computer program can offer the brightness and colour of the overhead or slide projector, although not, as yet, the image size. But the computer can offer sophisticated animation and sound effects.

There are ways in which the computer can make a unique contribution to teaching. Firstly, a major educational use of the computer is simulations.

SIMULATIONS

A computer can produce with great realism and accuracy, a simulation of a very complex situation. Of course it would be silly to simulate something which could be done simply, quickly and cheaply in reality. There are many occasions where an activity can't be carried out except in simulation.

Sometimes the situation is too dangerous. For example, students may wish to investigate how the control and fuel rods work in a simple nuclear reactor.

Sometimes the exercise would be too costly to carry out in reality. One group of students was working on the buoyancy characteristics of various designs of concrete boat. They quickly found out by simulation what NOT to build!

Sometimes the computer can portray happenings which will always be invisible, although the laws governing them are reasonably well understood. For example, the movement of electrons through a gas. Or the way a volcano proceeds towards an eruption.

Another type of simulation is the "what if.." type. Here, the user is able to choose various parameters in a situation and see what the effects are. Computer simulations can produce un-dreamt of predictions. Bypassing human pre-conceptions, a program can prompt our intelligence and imagination.

A computer can handle a large number of repetitions quickly. For example, in a probability lesson we may want to discuss outcomes with dice. The computer can "throw" thousands of dice during a lesson, and produce results for discussion. It's not that a person could not do it, but the time is not available.

Using a computer simulation, it is often possible to change the time-scale of events. This is similar to projecting a film at a faster or slower rate than that at which it was taken. Much can be learnt in this way. The "evolution" of an animal species under various conditions can be simulated so that hundreds of thousands of years pass in moments. Conversely, an incident which is over in a flash, can be stretched out to last minutes in simulation.

Randomness can be built into a simulation, to mirror the chance-events we find in life. Sometimes even the programmer is amazed at what turns up. This is particularly true with historical simulations. For example, if a military or political figure is killed, or perhaps a crucial supply ship hits a reef, the whole course and outcome can change. It is fascinating to construct these "alternate histories" – we see how frail the paths of our lives really are.

The complexity of computer operations enables an "holistic" experience to be produced. Thus a pilot can "experience" flying a 747, with hundreds of interrelated factors operating in simulation at the same time. This includes rare events

which the pilot needs to be prepared for. Of course, if this was the only type of training a pilot received, we would be worried, because a computer simulation is usually an approximation.

Two recent movies illustrate this. In "The China Syndrome", computer simulation of the possible hazards in a nuclear plant proved inadequate. Insufficient allowance had been made for human greed. The film "Capricorn 1" gives an entertaining portrayal of the ultimate simulation of our era. but in the end, being an approximation it had to fail.

Simulations in Science teaching.

One aspect of simulations must be watched carefully. We must stress that we are not watching "nature". I saw a group of students studying printouts of simulated collisions between objects. They were impressed, and clearly regarded the figures as "proof" of the physics laws involved. However, the programmer based his work on those laws, so the success of the simulation merely "proved" his competence as a programmer. He could have "proved" $F=ma^2$ if he had wished. Simulations can illustrate aspects of the physical world in very interesting ways. But we must not let students think they are "proving" anything.

Rene Descartes, a contemporary of Galileo, was revered as the greatest scientist of the age. However, he wrote that a person need not interact with the physical world to study it. Most of his philosophical speculations about the real world were later proved preposterous when tested against nature. In his "ivory tower", Descartes attempted to discover truths by mathematical analysis, which can only be discovered by experiment.

We must not allow the computer to become our electronic "ivory tower". We need to bear in mind what Professor Karl Popper says about "scientific" conclusions. They must not only be verifiable by repeated experiments, but they must be falsifiable. A computer simulation, by itself, is not falsifiable in this sense. If we program a computer to handle facts in a certain way, the validity of our conclusions will reflect our wisdom in programming. Teachers must always stress the need for accurate observation and encourage students to be critical and curious, and want to test propositions in the laboratory.

In fact, a computer is more suited to disproving a law, since it can search for the necessary single exception, without tiring or losing patience.

A computer can be used to construct a model of a situation. This model can be refined and improved repeatedly, by continual referral back to nature. The computer can even suggest what the next direction might be. Thus the computer is an indefatigable assistant to the scientist, not his replacement.

Interaction

Another unique feature of the computer as a teaching aid its interactive ability. The user can be offered choices, help, direction, even criticism. Hence a computer program is much more adaptable to the needs of individual learners, than any other type of teaching aid. In a foreign language course for example, a student may be able to choose the level of difficulty, the type of work (vocabulary, grammar etc) which is appropriate. The program can be designed to correct errors and offer suggestions in keeping with the student's progress. A textbook cannot tell how well the

reader is doing. The rate and direction of progress through the language course can change continually, just as a good coach would do for a student. The flexibility is there to allow the bright student to skip ahead. The slow learner has in a computer, a "teacher" with endless patience and no pre-conceived ideas or prejudices. This "teacher" can also be programmed to always encourage and never abuse.

STORING DATA

The computer's ability to store vast amounts of information in a readily accessible form, can also be useful to the teacher. This applies not only to records, marks and reports, but to actual teaching. For example, subjects in which there is a lot of classification could make much use of a database. In biology and geology particularly, students could access sets of information of their own choosing, very quickly. Commercially available file-management programs would allow teachers with almost no computer experience to design extremely useful databases to suit themselves. Thus the computer can act as a marvellous subject resource, particularly if a disc system is available. Also it is very cheap and easy to add to, delete from, or alter records. Such a resource could be shared with other teachers around the country or across the world, with ease. Databases created by local people everywhere could be exchanged to everyone's benefit.

Interfacing

The BBC computer is ideal for interfacing to other equipment. This has opened up many possibilities, particularly in the scientific areas of education.

Handling numbers

The microcomputer can be useful to the teacher in many ways. We haven't yet mentioned that a computer knows a lot about "Maths" too. A secondary school student thumbing through the BBC User's Guide see many functions known from Maths courses. Logarithms, trigonometry, exponents, all these things one finds on a calculator. Yet a computer is far more powerful than a nomal calculator. A computer can be very useful to the Mathematics teacher.

And of course the computer can be programmed to help teach computing.

Although there are already so many uses for computers in Education, the future promises many more wonderful things.

When is the use of a computer program inappropriate?

If a lot of text needs to be presented, the bright video screen is not the place for it. Even with low glare green or amber screens, students generally resist reading a lot of material. At present, there is a certain psychological urgency associated with the screen. Also, students using computers are often on restricted time and feel under pressure. Perhaps the next generation of students, growing up in ever closer relationship to video screens will not have this problem.

Sometimes a computer program needs a longer time frame than the student can realistically spend at the computer. The considerable thought provoked by some programs makes them unsuited for use in a classroom situation. There are

ways around this problem, like the production of a hard copy. Another technique is that used in "Adventure" games, where data can be saved on cassette or disc, and the program resumed at a later date from the same point.

A computer program is unsuitable for teaching when a simpler, easier aid does a better job. Related to this is the use of a computer to present a boring treatment of a topic. In other words, when the computer is only used for novelty value. This does harm to the use of computers in education.

Efficiency and Readability.

The more one explores the habits of the BBC microcomputer, the more short cuts one finds. In syntax for example, quotes and brackets may sometimes be omitted. VDU statements can be strung together. THEN can often be omitted from IF-THEN structures. Commas can be used in place of a series of NEXT statements. There are many other abbreviated methods, some very pretty and sophisticated. Many lead to a greater execution speed, and offer a valuable saving in memory usage. Sometimes a call to a resident machine language routine produces far, far better results than BASIC does.

As teachers we must resist the temptation to appear "clever". To the student it may simply be obscure. When a program is displayed to students learning BASIC, readability is of the greatest importance. Efficiency comes next.

Chauvinism.

Let's imagine that all existing knowledge about computing is displayed in a meaningful sequence on a notice board one

metre high and many kilometres long. If we started at one end and tried to assimilate everything as we moved along, most of us would find our minds were "leaking". To maintain sanity, a person's mind tends to reject information it regards as less relevant to survival. For example, our eyes receive vast amounts of information each day. If we were aware of it all, we would quickly be swamped.

If we imagine the computing community standing at this notice-board, what would we find? Some people will be very familiar indeed with certain areas of it. This may reflect a particular interest, or vocational need. Or it may simply mean they have been reading the board for a long time. There is a tendency for specialists in a small area to look askance at those who are not. Some Pascal exponents regard BASIC users with disdain. For some, machine code programming is the only "real" computing. These people have tunnel vision. They are rooted to a spot in front of one section of the "notice board". I am not suggesting that we should have a shallow encyclopaedic knowledge and master nothing. But we must all be respectful of one another's knowledge, and have enough humility to learn from each other. There is no place for derisive factions within the educational computing community. One teacher who was proficient with the Apple computer, waved away dismissively, an enthusiastic group of hobbyists because the Sinclair ZX81's they had at home were "just toys". Had he listened to them he could have at least gained from their exuberance. Instead, he taught them how to be chauvinist.

The computer room.

The characteristics and disposition of a school computer room are topical concerns.

The room should be painted in an unobtrusive colour, with diffuse lighting to avoid reflections in the video screens.

For safety, there must be no trailing power cords.

To help control dust, the floor should be hard, rather than carpeted and chairs preferably not covered in material. If drapes cannot be avoided, they need to be washed regularly. Chalk should be replaced by water based felt pens used on an overhead projector or whiteboard. There should be no tap in the room and no facilities for making tea and coffee, since liquids are so hazardous to computers.

Concerning furniture, the benches should either be robust, or secured to the floor so that equipment is not easily dislodged. A display board is needed for notices, along with plenty of lockable cupboards for storing equipment. Consider particularly the storage of and accountability for discs, tapes and other small items which are attractive to the petty pilferer. The chairs used by the students should be height adjustable to allow a proper keyboard position for each student.

The computer room needs a lockable phone with a direct outside line. The computer can then be conveniently used with a modem to communicate with the outside world. This is a very exciting aspect of school computing.

For security, the room should preferably be on an upper floor, and have secure windows and doors. The room should be protected against burglary by a movement detector or other suitable alarm. Don't forget to tell the cleaners about it.

SECTION II

RUNNING A FIRST COURSE IN BBC BASIC

Management of a Computing class

Scene:

A class is about to arrive at the computer room for the first lesson of your course. Many will be very keen. Others will be apprehensive: "Here's something else to fail at. I'll look stupid again."

There may be the usual few who are ill-suited to school. Their life-problems will surface in this subject, as they do in all the others. However, some so-called "disruptive influences" have the first real scholastic success of their lives with computers and undergo a transformation.

The computing background of the students may vary enormously. Expect a grading through from the student who knows nothing, to the one who is contemptuous of your "simple course". (Sometimes the latter turns out to be like the former).

What will you say to these people? How can you communicate the excitement of computing? How can you allow them maximum individuality and help them best, while keeping good group control?

Choosing groups and places

If groups are to be used, I prefer to let students choose their own partners. However, emphasize that their choice should be from those they can work well with, not necessarily their buddies. They should understand that if there is undue inattention or misbehaviour you will not hesitate to reallocate them.

Nevertheless it may be necessary to exercise your discretion and choose some combinations yourself. For example the different ones often play safe by choosing partners like themselves. Because of social conditioning, girls in a mixed group tend to take the passive roles. They often make notes and record values while the boys do the creative things and make the decisions. This must be circumvented, even if it means forming some all-girl groups.

Organisation of equipment

Make each student (or small group) responsible for the same set of equipment throughout the course. Each work-station should have a number which is indelibly written on each item. The names of those using a work station in each session should be kept in a book (not a wall chart).

Every group's first task each session is to see that all the equipment is present and in good condition. Any discrepancies, damage or malfunction must be reported immediately. Make it plain that otherwise they will be held responsible. Follow up any problems with the previous users of that work-station, and take the necessary corrective measures. Do not allow faults with a particular workstation to drag on.

The licence course

This course is detailed a little later. It sets out not only a base of computer knowledge but also ground-rules for behaviour. All students should acknowledge a set of standards for the little community which uses the computer room.

Infractions of rules

Enforcement of rules is particularly necessary in a computer room because of the fragility and expensiveness of the equipment. One common infringement occurs at lunchtimes, when students, particularly hangers-on, are tempted to bring food and drink into the room. They will say
"But I'm just watching." or
"The rules only apply to lessons, not to lunchtimes."

Point out that the rules are those of the Computing Room Community, and that if they are not observed, you will have to lock the room. Use of the room outside of lesson times is a privilege.

Should a charge be made for damage?

Clearly this depends on the school, and the affluence of the students. But it is at least a worthwhile bluff which makes a good point. It does seem to reduce the hammering of keyboards and careless treatment of cassettes and discs.

Helping stragglers, occupying the precocious

One handicap with the present state of computing as a subject, is that only the lucky few will have access to a computer after hours. The release of the Electron, and generally falling prices should help. Certainly those with a BBC computer at home are unlikely to need help anyway. The teacher will need to devise written material to help the slow learners who are without computers. This is largely unbroken ground. The language used in this material will need to be free of unnecessary jargon and straightforward in style. The quick student can be given access to books or magazines, to research a particular area (eg. the uses of GCOL1 to GCOL4.)

There is a list of student projects at the end of this section, which could be used. They might learn how to use the printer or disc drive. There are filing, word processing and accounting packages which can be purchased at reasonable prices. A good student could be let loose on one of these. There are speech synthesisers and word recognition devices. There are other languages like Forth and Assembly language which can be studied. Fortunately there are many fascinating things for the able student interested in computers.

Equipment failures

Symbiosis should allow the school to have a good relationship with a dealer. Preferably, the peripheral products should also come from the same supplier. Then if a puzzling breakdown should occur in the school's computer system,

the dealer cannot shuffle off responsibility by blaming a component which he did not supply. The dealer should be able and willing to give occasional advice over the phone. He should be situated close to the school and be willing to lend equipment in place of that which he is repairing.

If possible, have one working spare for each of the workstation items. In case one or more of the work stations is disabled, have some computer-related activities which people can go on with.

What about playing games?

The use of microcomputers as vehicles for "Arcade" games like Pacman has muddied the waters for computing teachers. A common impression held by non-computing students and teachers alike, is that a computer is something one "plays with". Indeed, it is even common parlance among computing teachers. This arises because computing can be a tiring and frustrating struggle, but for most of us it is also very stimulating and enjoyable. Hence to begin with, even a serious course in Basic may have a frivolous image.

Students should not be playing prepared game software in a computing course. It may be acceptable as a leisure-activity but it should not be termed a "Computer Club". One group I know of, meets after school to play "Adventure" games. They are known as the "Adventurers", and they have a marvellous time. But not at any stage are they doing any computing. The same applies to users of dedicated word processors, laser discs etc. Let's keep the concept of "computing" separate from the multifarious uses of microprocessors generally.

Building BBC BASIC on a firm foundation

When beginning a course, the teacher needs to know how much the students already understand.

I expect all students to first pass a "licence" test. This test is in three parts: Behaviour, Practical and Theory.

The **Behaviour** part of the test involves a knowledge of and a commitment to the following rules. (The Oxford dictionary defines a commitment as an engagement that restricts freedom of action).

a) No food, drink or chalk in the computer room.
   (Explain the damage which liquids and small particles can cause to tape drives, discs and keyboards).
b) No unruly behaviour: jostling, loud noises etc.
c) Generally show a sensible concern for the people sharing the room, and for the equipment.

The **Practical** part of the test involves the following:

a) Know how to tun the equipment on and off.
b) Know how to correctly handle, insert and store cassettes and/or discs.
c) Treat the keyboard gently.
   (Many people's only keyboard experience is with a manual typewriter, where each character is impacted onto the paper through a ribbon. The force of the fingers is transmitted via levers. Point out to students and teachers, that a computer keyboard is an array of sensitive switches. Also, avoid the word "Punch" as in the "Punch in the information". Use of this word dates back to the days when holes were punched in cards or paper tape to input information. But "punch" has the wrong connotation for

a modern keyboard).

d) Have a fair knowledge of where the keys are. Keyboard proficiency is not demanded, but some students scan the whole keyboard systematically every time. It should not take 15 seconds to find N and then 15 seconds to find M.
e) Know how to make simple adjustments to the video equipment.
f) Some teachers would include elementary printer operations too.

The **theory** material for the test is outlined below. It consists of fundamental material common to almost all micro-computers, for which there are many good teaching books available.

A computer program is a numbered or named set of logical instructions. These are performed in a sequence determined by the program. Unless instructed otherwise, the computer follows the line numbers in ascending order, unconcerned by gaps. The order in which the lines were typed is irrelevant.

The commands LIST, RUN, NEW, LOAD and SAVE are introduced, together with the statements RETURN, LET, PRINT, GOTO, FOR-NEXT-STEP, IF-THEN, INPUT, DIM, DATA, READ, RESTORE and REM. An introduction to numeric and string variables is given. The four simple arithmetic operators are taught, along with AND.

As well as things peculiar to the BBC micro, other material is not presumed at this stage. This includes TAB, ON, STOP, GET, INKEY, ELSE, REPEAT-UNTIL, exponentiation, AND, NOT , RND all the other numerical and and string functions, and GOSUB. I hope they don't hear about GOSUB until I have taught PROCEDURES.

A good way of teaching the licence material is to use a system of carefully graded cards which are used individually on a hands-on basis by half the class. The other half of the class receives direct instruction The two halves alternate between the card system and the class group until the course is finished. This will only work if the cards are carefully written so that the group using them is self-sufficient. A set of extension cards on a topic like Animation should be included for those who finish the course early. The advantage of this system is that the expected large variation in the knowledge of the students is to some extent allowed for. Some will have a computer at home and be keen hobbyists. Others may have an emotional resistance to learning anything about computers at all. The card system allows some degree of individual progression. At the end of the Licence course an on-lesson exam, partly written and partly practical is an excellent idea. Once we have a class of graduates, we can assume a certain body of knoweldge on which to build our BASIC course.

Here are some typical Licence Test questions:

1. Write a program using a FOR-NEXT loop which prints the famous visual oddity:
PARIS IN THE
THE SPRING
six times on the screen, with a blank line between each. Save it and have it ready to LOAD and RUN for the Tester.
2. Have in the computer's memory ready to RUN, a program which prints the letter Q when that key is pressed, and ignores any other key (except ESCAPE and BREAK). Be able to LIST it. Include REM statements explaining what the program does.
3. Without using the computer, design a program on paper which prompts the user to supply two numbers, adds them and prints their average.

4. Why are gaps left in program line numbers?
5. What happens if NEW is not used before typing a new program into the computer?
6. Why do we avoid using the letters I O U and V as variable names?
7. Study the following program. What value of N is needed? What does the program do?

```
10 FOR J=1 TO N
20 READ NAME$
30 READ AGE
40 PRINT NAME$,AGE
50 NEXTJ
60 DATA FRANK,15,JILL,16,TOM,14,SUE,15,
   RICARDO,16
```

What would happen if we added this line, and ran the program again?
70 GOTO 10
8. Why shouldn't we GOTO a line which is a REM statement? 200

TEACHING BBC BASIC

You might like to start by providing your students with this User-defined-keys program called "R" (for Red keys). It's justifiable to use "R" (or something similar) without any explanation at this stage, because more of value can be achieved in the limited time available.

For your convenience, the program is reproduced below. The methods used to define the red keys are discussed in Section III.

```
 10 REM RED KEYS DEFINED FOR STUDENT USE
 20*KEY0 |V7LIST|M
 30*KEY1 RUN|M
 33*KEY2 SAVE"
 37*KEY3 PRINT
 40*KEY4 INPUT
 45*KEY5 PROC
 50*KEY6 DEFPROC
 60*KEY7 ENDPROC
 65*KEY8 LOAD"
 70*KEY9 CHAIN"
 80 *FX138,0,78
 90 *FX138,0,69
100 *FX138,0,87
110 *FX138,0,13
```

A paper strip bearing the key labels can be inserted under the plastic plate. Some teachers prefer to put Scotch "Post it" or a similar product over the plate, because label strips can then be changed easily.

f0	f1	f2	f3	f4	f5	f6	f7	f8	f9
LIST	RUN	SAVE"	PRINT	INPUT	PROC	DEFPROC	ENDPROC	LOAD"	CHAIN"

There are five command keys, two of which include RETURN. Key 0 is the most appreciated by students. It clears the screen, changes to MODE 7 and LISTs the program which the student has in the memory.

Keys 2, 8 and 9 are SAVE", LOAD" and CHAIN". The program name, second " and RETURN are provided by the student. Students would usually use CHAIN rather than LOAD, but there are occasions when the teacher wishes to discuss a program before it is RUN. Indeed, some programs cannot be LISTed after they are RUN. (Like this R program for instance.) Keys 3 to 7 print statements on the screen to allow more time for thinking rather than typing. The first thing your students could do each session is to CHAIN "R". It deletes itself from memory but the keys stay defined even if BREAK is used. (CONTROL & BREAK will remove the key definitions). Nothing in this section relies on the USER KEYS, so the choice is yours.

PRINTing

First let's deal with the neat formatting of text. Whatever follows the keyword PRINT in a program, is called the "print list". The first character in the print list is important for formatting, even if it happens to be a space.

To prevent text wrapping from one displayed line to the next, students should enter the words using the MODE in which the text will be displayed. It is then obvious by vertical alignment with the first character, when a line to be displayed cannot take more text. Also, no space is left between words if it happens beneath the first character.

Here is an example which illustrates these points.

If a student wants to display some text in MODE5, which has 20 columns, it should be entered using MODE5 noting the column where the "first character" is. In this example, it is in column 10, the columns being labelled from 0 to 19.

```
210 PRINT"Napoleon t
hen saw  that the a
pproachingwinter was
 a danger!".
```

Two extra spaces are left after "saw" so that the second displayed line starts below the "first character". Also, no space is left before the word "winter", since it begins the third line of the display. Note the significance of the space before "a" in the last line. Some teachers like to distribute "text planning" grids to help students format text.

If two separate PRINT statements are being used, and the first completely fills a display line, the BBC will leave a blank line before printing further. To avoid this a ; character is used to tell the computer to follow on immediately with the next word.

Here is an example, using MODE5. The print list in line 3000 has exactly twenty characters.

```
3000 PRINT"THE CAMPAIGN IS OVER"

3010 IF D = -1 THEN 2160

3020 PRINT"AND YOUR ARMY HAS    BEEN
     DEFEATED"
```

To prevent the blank line, a ; is inserted at the end of line 3000.

Sometimes the opposite effect is required: we may want to force a new line. This is done with the apostrophe. Here is an example.

```
10 PRINT "Two blank lines will now be
   left"''
20 PRINT "before continuing."
```

On occasions, you will need to leave a space at the end so that words will not be run together. Example:

```
10 INPUT"What is your favourite colour",
   COLOUR$
20 CLS
30 PRINT'"I see that you really like
   ";COLOUR$;"!".
```

Note the space left after the word "like".

When formatting numbers, BBC BASIC divides the screen into zones 10 characters wide. Zone 1 runs from screen column 0 to column 9. A comma can be used to tell the computer to print any numbers which follow, in the next zone and as far to the right in that zone as possible. So PRINT 457 will cause 457 to be printed in columns 7 8 and 9. This is useful for aligning the units when printing numbers in columns on the screen – it won't matter how many digits in the numbers.

```
10 FOR item=1 TO 7
20 READ number
30 PRINT number
40 NEXT
50 DATA 23,78,7,8,143,44,5567
```

If column 0 is the desired start, a semicolon must precede the number. eg PRINT ;1073

Students need to realise also, that BBC BASIC provides a minus sign for a negative number but omits the + sign when printing a positive number.

When a printing text, the comma causes the computer to print what follows, at the beginning of the next zone.

```
10 REM COMMAS IN TEXT PRINTING
20 MODE7:REM This clears the screen too
40 FOR field = 1 TO 4
50   FOR fieldcol = 0 TO 9
60     PRINT;fieldcol;
70   NEXT fieldcol
80 NEXT field
90 PRINT"THIS STARTS AT COLUMN 0"
100 PRINT"&","THIS STARTS AT COLUMN 10"
```

The TAB function

The actual screen position will depend on the mode. In MODE5 with 20 columns and 32 rows, PRINT TAB(9.15)"*"

will print in the middle of the screen. In MODE6 with 40 columns but only 25 rows, the star will appear nearer the lower left corner. The following program shows this:

```
10 MODE5
20 PRINTTAB(9,15)"*"
30 FOR T=1 TO 5000:NEXT
40 MODE6
50 PRINTTAB(9,15)"*"
60 PRINT''"Same coordinates. Different
   place."
```

Later when learning about graphics, students discover that a more precise coordinate system can be used with the PRINT statement, via VDU5.

TAB can also operate with only one number in the bracket, referring to the column at which printing is to start. TAB(0) is at the left edge of the screen (or "text window" – see Section III). Point out that there is no specification of row position when using the simple one number TAB and so scrolling will occur as normal. Here is a program showing these effects.

```
10 MODE7
20 FOR column=0 TO 40
30 PRINTTAB(column)"STEPS"
40 NEXT
```

ENTERING INFORMATION FROM THE KEYBOARD

The keyboard buffer

There are usually places in a program where a key (or a series of keys) need pressing to enter information or instructions. The BBC computer has a keyboard "buffer" which stores up to ten characters. This is a problem if an earlier keypress, not relevant now, is next in the queue. The keyboard buffer can be flushed clear by entering *FX21,0. This calls a machine language routine, which is stored permanently in the computer. If the students type in and RUN the following program, you can show them that characters entered during the delay loop are taken as INPUT when line 20 is revisited, before the question-mark prompt even appears. Then delete the word REM from line 10 so that *FX21,0 will take effect when the program is RUN again. Now only entries typed after the prompt, are accepted.

```
10 REM *FX21,0
20 INPUTA$
30 PRINTA$
40 FOR delay=1 TO 5000:NEXT
50 GOTO10
```

The ESCAPE and BREAK keys

Another difficulty arises if the user presses the ESCAPE key wrongly, stopping the program. Entering *FX229,1 will disable the escape key. When the input process is over, *FX229,0 will enable it again. Here is a program to demon-

strate the point. (Some details of this program are new but need not be explained at this point).

```
 10 REM NOT AVAILABLE BEFORE ISSUE 1.0
 20 MODE7
 30 VDU23;8202;0;0;0;:REM Erase Cursor
 40 *FX229,1
 50 FOR count=1 TO 300
 60 PRINTTAB(0,5)"*FX229,1 DISABLES THE
    ESCAPE KEY."
 70 PRINTTAB(0,10)"TRY IT!"
 80 PRINT TAB(0,15)"WHEN 300 IS REACHED,
    IT WILL BE ENABLED"
 90 PRINT TAB(0,20)count
100 NEXT count
110 PRINT'"ESCAPE KEY NOW ENABLED"
120 *FX229;0
```

For issue 0.1 ?&226=&01 can be used to disable the BREAK key. It can be reenabled with ?&226=&00.

The BREAK key cannot be disabled but it can be redefined. Entering *KEY10 OLD ;M RUN :M does it. The ; can be explained as "CONTROL". CONTROL M when used in a program, enables the computer to press its own RETURN key. Point out that the program will be stopped and immediately re-RUN. So if a student is entering (say) a long list of numbers and accidentally presses BREAK, bad luck! If the user wants to override this redefinition of the BREAK key, entering *KEY10 will restore its function. It is important that these disabling mechanisms be reversed within the program, once they are no longer useful. In the demonstration program which follows, this has deliberately not been done.

```
10 MODE7
20 ON ERROR GOTO 40
30 *KEY10 OLD !M RUN !M
40 PRINT'"YOU CAN'T STOP ME UNLESS YOU
   TURN THE COMPUTER OFF."
50 FOR delay=1 TO 10000:NEXT
60 PRINT'"SEE? YOU ARE QUITE HELPLESS."
70 FOR delay=1 TO 10000:NEXT
80 PRINT'"AS I TOLD YOU......":GOTO40
```

Holding the CONTROL key down while pressing BREAK will undefine all the user-keys. Before telling the students that they can use CONTROL and BREAK to get free, warn them to then type OLD and press RETURN, if they want to keep the program. You could also point out that line 20 shows another way to disable the ESCAPE key, though it fails in some circumstances (eg if delay=INKEY(5000)) is used as a delay method. Also, using ON ERROR like this may have unforseen effects, and sometimes real errors are trapped. So *FX229 is better.

Using a "prompt" with the INPUT statement

Students will know the simple use of INPUT from their "Licence" course. Here we first consider the extension of the INPUT statement by the inclusion of a "prompt". This, together with the use of "Menus" form the basis of a "user-friendly" or "humanised" program.

After the word INPUT, the prompt is typed within quotes. This will be printed on the screen. It is followed by the name of a numeric or string variable.

If the "prompt" is in the form of a statement, like "ENTER A NUMBER FROM 0 to 100 AND PRESS RETURN", no question mark is required. If the "prompt" is in the form of a query, like "How old are you" a comma can be inserted before the variable. Then the computer will insert two spaces and the question mark. Of course, the user could insert spaces and the question mark within the quote marks instead.

If the INPUT statement expects a string and you enter a number, the computer accepts it as a string consisting of a number. You can use the normal string operations on it, but not do any maths with it. What if the INPUT statement expects a number and you supply a string? It interprets that by entering no number, you have entered zero. Many computers instead reject the string and stop the program with a "TYPE MISMATCH" error message. Teachers of computing argue about this. It is true that stopping the program can be unnecessarily brutal. But should we give students the idea that "nothing" is the same as "zero"? After all, an empty set is different from the set which contains the number zero as its only member.

Single key INPUT

The INPUT statement takes no notice of the keyboard until the RETURN key is pressed. Students need to realise that the computer works on a completely foreign time-scale to ours. It performs at least a million operations a second. The time it takes a human to press a key, is a very long time in the computer's time-frame. This is well presented in the film "TRON". In some situations, particularly real-time games, a single-key input is needed so that the effect will be immediate. The BBC computer provides two single-keypress entry methods, each with a numeric and a string variation.

First we consider GET and GET$

These statements cause the computer to wait indefinitely for you to press a key. Each statement needs a variable, for example
200 A = GET

Just before the GET statement is reached, the keyboard queue should be flushed using *FX21,0. Then the keyboard buffer will be empty and the computer will be scanning the keyboard waiting for you to enter a number. It gives to the variable A, the ASCII code value of the key pressed. Here is a program which generates ASCII values, together with a printout of the results.

```
10 MODE6
20 FOR asciinumber = 32 TO 126
25 PRINT' asciinumber;"   ";
30 VDU asciinumber
40 NEXT
50 PRINT
```

At this stage, the VDU statement in line 30 can be interpreted to students as an instruction which sends the symbol whose ASCII code number is n, to the Video Display Unit ie the screen. For students familiar with other Basics, this use of VDU can be described as a BBC BASIC abbreviation of PRINT CHR$(n).

ASCII CODES FOR MODES 0 to 6.

(There are minor differences for MODE 7. See page 486 of the 1982 User Guide).

32	space	57	9	82	R	107	k
33	!	58	:	83	S	108	l
34	"	59	;	84	T	109	m
35	#	60	<	85	U	110	n
36	$	61	=	86	V	111	o
37	%	62	>	87	W	112	p
38	&	63	?	88	X	113	q
39	'	64	@	89	Y	114	r
40	(	65	A	90	Z	115	s
41	)	66	B	91	[	116	t
42	*	67	C	92	\	117	u
43	+	68	D	93	]	118	v
44	,	69	E	94	^	119	w
45	-	70	F	95	_	120	x
46	.	71	G	96	`	121	y
47	/	72	H	97	a	122	z
48	0	73	I	98	b	123	{
49	1	74	J	99	c	124	|
50	2	75	K	100	d	125	}
51	3	76	L	101	e	126	~
52	4	77	M	102	f		
53	5	78	N	103	g		
54	6	79	O	104	h		
55	7	80	P	105	i		
56	8	81	Q	106	j		

GET$ needs a string variable assigned to it. For example,
500 A$ = GET$

If the keyboard buffer is empty, the computer scans the keyboard waiting for you to enter something.

As soon as you press a key, the computer takes this as the variable value, and the program proceeds.

With both GET and GET$, the program waits indefinitely for a keypress. (This is not the case with some other BASICs.)

INKEY

The other single-press entry is the INKEY statement. It too has string and numeric forms but differs from GET in that it waits for a time interval, which you specify. If the keyboard buffer is empty and no key is pressed within that time, the program proceeds without it. The time is entered in brackets, in hundredths of a second. Here are examples.

A=INKEY(200)

The program waits up to 2 seconds for you to type a number. If you type a letter, it interprets that you pressed the zero key.

It can be used as a delay mechanism.
eg delay=INKEY(100)
makes the program pause for 1 second. The keyboard queue should be cleared by *FX21,0 first.
A$=INKEY$(50)

If the buffer is empty, the program waits up to half a second for you to enter something.

Other spurious inputs.

We have discussed spurious inputs from the ESCAPE and BREAK keys. Let's now consider what else could go wrong.

Suppose an INPUT routine is to be used which prompts for an integer from 0 to 99 inclusive.

eg 100 INPUT"ENTER A WHOLE NUMBER FROM 0 TO 99"num0to99

Here, num0to99 is the name of the numeric variable.

What if the user enters 112? Or -3? Or enters a non-integer eg 65.2. Or enters a letter eg a W instead of a 2. The following program is useful to illustrate the trapping of such errors.

```
 10 MODE7
100 INPUT'"ENTER A WHOLE NUMBER FROM 0
    TO 99"'num0to99
110 IF num0to99 <0 OR num0to99 >99 THEN
    PRINT'"NO, PLEASE":GOTO100
120 IF num0to99 <> INT(num0to99) THEN
    PRINT'"NO, AN INTEGER":GOTO100
200 PRINTnum0to99
210 GOTO100
```

In line 110, the range of the variable is restricted. Line 120 rejects a non-integer INPUT by means of INT. This function produces the integer which is just less than or equal to the number entered. If instead we wanted the nearest whole number to be used, INT(num0to99+.5) would provide it, as discussed later.

Here is a program which illustrates the rejection of unsuitable input when GET and GET$ are used. The symbol <> can be interpreted to students as "is different from".

```
10 REM REJECTING SPURIOUS INPUT
20 CLS
30 PRINT'"Press C to continue."
40 key$=GET$
50 IF key$<>"C" AND key$<>"c" THEN 40
60 PRINT'"Press one of the keys 0 to 9"
70 key=GET
80 IF key<48 OR key>57 THEN 70
90 PRINT"You pressed ";key-48
100 GOTO 30
```

Point out that both upper and lower case are tested because the CAPS LOCK key may have been inadvertently pressed. Other ways around this problem are discussed in Section III. Other input protection routines are available which use more structured statements like REPEAT-UNTIL. These are discussed later. The discussion of menus in Section III is relevant to this work, particularly the use of ON-GOTO-ELSE.

Negative INKEY

Another means of communication with the BBC computer via the keyboard is "negative INKEY".

When the computer meets a negative INKEY statement in a program, it scans the keyboard to see if any keys are being held down at that time.

Negative INKEY differs from GET and positive INKEY in many ways. Since it is concerned only with the present not the past, emptying the keyboard queue is not needed. It also differs in that negative INKEY can detect multiple keypresses. This means that a combination of keys can be pressed to send a complicated message to the computer while it is running.

The negative INKEY statement number refers not to a wait time but to a unique number allotted to a key. For example INKEY(-1) detects either of the SHIFT keys. Here is a list. Note that some keys (eg £, * <>) are not mentioned. As you can see, the numbers used are quite different from the ASCII numbers.

Key	Number	Key	Number
f0	−33	1	−49
f1	−114	2	−50
f2	−115	3	−18
f3	−116	4	−19
f4	−21	5	−20
f5	−117	6	−53
f6	−118	7	−37
f7	−23	8	−22
f8	−119	9	−39
f9	−120	0	−40
A	−66	-	−24
B	−101	^	−25
C	−83	\	−121
D	−51	@	−72
E	−35	[	−57
F	−68	_	−41
G	−84	;	−88
H	−85	:	−73

I	−38	]	−89
J	−70	'	−103
K	−71	.	−104
L	−87	/	−105
M	−102	ESCAPE	−113
N	−86	TAB	−97
O	−55	CAPSLOCK	−65
P	−56	CTRL	−2
Q	−17	SHIFTLOCK	−81
R	−52	SHIFT	−1
S	−82	SPACEBAR	−99
T	−36	DELETE	−90
U	−54	COPY	−106
V	−100	RETURN	−74
W	−34	↑	−58
X	−67	↓	−42
Y	−69	←	−26
Z	−98	→	−122

Unlike the other statements which check for keyboard input, negative INKEY works even for those without ASCII numbers. Keys like CAPS LOCK, CONTROL, DELETE and f0 to f9 can thus be detected. The following program detects any combination of three keys, none of which have ASCII numbers.

```
10 REM NEGATIVE INKEY DEMO
20 MODE7
30 VDU23;8202;0;0;0;:REM Erase the cu
   rsor so the display looks better
40 IF INKEY(-65) THEN PRINT TAB(0,5)"
CAPS LOCK IS BEING PRESSED       " ELSE PR
INT TAB(0,5)"CAPS LOCK IS NOT BEING PRES
SED"
```

```
50 IF INKEY(-1) THEN PRINT TAB(0,10)"
SHIFT IS BEING PRESSED      " ELSE PRINT
TAB(0,10)"SHIFT IS NOT BEING PRESSED"
60 IF INKEY(-118) THEN PRINT TAB(0,15
)"f6 IS BEING PRESSED      " ELSE PRINT T
AB(0,15)"f6 IS NOT BEING PRESSED"
70 GOTO40
```

INPUT LINE

Finally in our discussion of inputs from the keyboard, we come to the INPUT LINE statement.

The comma has a special meaning and cannot be used in a normal INPUT string. INPUT LINE allows the entering of a string up to 255 characters long, which can contain anything, including leading spaces and commas. Here is a program to illustrate these points.

```
10 CLS
20 PRINT''"INPUT A STRING CONSISTING
OF 5 LEADING  SPACES, A WORD, A COMMA TH
EN ANOTHER    WORD"''
30 INPUTstring$
40 PRINTstring$
50 PRINT''"SEE HOW THE LEADING SPACES
 WERE OMITTED AND NOTHING AFTER THE COMM
A WAS TAKEN"''
60 PRINT''"TRY THE SAME STRING AGAIN"''
70 INPUTLINEstring$
80 PRINTstring$
```

Procedures

A procedure is a self-contained section of a program which is invoked by name. The procedure can be fed certain values which you want it to use.

The section of a program which defines a procedure is never run into but always called from afar. Hence procedures are commonly placed at the end of a program after an END statement. However, from a teaching point of view it is sometimes better to display and explain the procedures of a program at the start. In this case, a GOTO can be used, as shown in the following program. We will return to examine this program shortly.

```
  5 REM PROCEDURES DEMONSTRATION PROGRAM
 10 GOTO1000 :REM JUMP OVER PROCEDURES

 20 DEF PROCprintnumbers(A,B,C)
 30  LOCAL J,K,L
 40  FOR J = 1 TO A
 50  PRINTA : NEXT
 55  PRINT"Local variable J =";J
 60  FOR K = 1 TO B
 70  PRINTB : NEXT
 80  FOR L = 1 TO C
 90  PRINTC : NEXT
100 ENDPROC
200 REM******START HERE*********
1000 A=3:B=6:C=2:J=17
1010 PROCprintnumbers(A,B,C)
1020 PRINT"Original J =";J
```

Students need to note that certain special-use characters (eg the operators * / + and -) are not allowed in procedure names. The use of lower case letters is often advised for naming procedures to avoid conflict with certain BBC BASIC keywords. To the operating system, upper and lower case are quite different letters. For example, the BBC micro regards RUN and run as quite distinct things. However, some teachers prefer students to keep the CAPS LOCK on, at least to begin with. Its so easy to forget to return to capitals, and the clumsiness of the BBC editor makes correction tedious, once RETURN has been pressed.

Spaces are also not allowed in PROCEDURE names. Students can use the underline character (on the £ key) to separate words. The name of a procedure should be concise but informative.

If we have a program with a procedure in it, we call it when we want to as in the following example.
1010 PROCPrintnumbers (3,5,2)
The Procedure then takes the particular variable values (parameters) we provide, deals with them as we want, then returns to the next line of the program, which could be another procedure.

Now let's look at the demonstration program more closely. The section defining the procedure must start with DEF PROC (with or without the space) to tell the computer we are defining a procedure. It must end with ENDPROC, typed without spaces since both END and PROC are keywords. The use of LOCAL is not obligatory in every case, but is a highly desirable practice. It declares that the variables J K and L, no matter how they have been used already in the program, are "born again". At the end of the procedure they have their original values (if any) restored to them. eg line 1020 will show J=17. In the meantime, these variables

are just used locally within the procedure. It's common for a student to accidentally use a variable name more than once. This can foul up a program in a very puzzling way. Its so helpful to declare variables as LOCAL. Some teachers insist that students type a line containing the words LOCAL DUMMY immediately after every DEF PROC statement, even if the names of the variables have not been finalised. This will remind students to declare them.

The advantages of using PROCEDURES.

Some programs one sees consist of many lines of code, with no procedures or subroutines. Often they have no demystifying REM statements either. Such a program may work very well, so why use procedures, which take time to set up? Without them, long programs are usually very difficult to read and understand. If there is a problem, it can be almost impossible to fix.

The use of procedures can be justified by their "abilities."

READABILITY

A helpful name can reveal the intention of a procedure. Also, a sequence of procedures renders the actions of a program more obvious.

UNDERSTANDABILITY

A long programming task can be broken into palatable pieces. These can even be shared among members of a "task force". The "dreaded GOTO" can be more easily avoided.

WORKABILITY

It is easier to check, refine and debug individual procedures.

COMPACTABILITY

It is often necessary to conserve memory, especially with the greedier BBC modes. The repetition of bunches of code can often be replaced by repeated calls to a procedure. If tape is being used, loading time is usefully reduced by compacting a program.

PORTABILITY

A library of useful procedures can be built up and used in different programs, saving a great deal of time and effort "re-inventing the wheel". There are books of useful subroutines published, but procedures are even better because variables can be localised. They won't clash with global variables in the host program which happen to have the same name.

The disadvantages of PROCEDURES.

It's not possible to change MODE with a procedure, because of a stack problem. Compared with GOSUB and GOTO, a PROC takes more memory, more listing space, is "fiddly" to set up (PROC, DEFPROC, ENDPROC) and is usually a little slower. Also, the use of procedures can be overdone (the new toy syndrome). Some programmers

become obsessed with "structure" to the point where they would never use a GOTO.

GOTO and GOSUB certainly have a place, particularly in conjunction with the ON statement. In fact, the last disadvantage of PROC is that it can't be used directly after ON..GOTO and ON..GOSUB.

Finally it should be pointed out that the BBC Basic RENUMBER disguises one of the main evils of GOSUB and GOTO and makes procedures look fussy.

SUBROUTINES

A subroutine is an earlier, more primitive form of procedure, and is also usually parked near the beginning or end of a program. It is called using its line number, not by name as a procedure is. Unless REM statements are used, a subroutine is not as readable.

Another less sophisticated aspect of subroutines is that variables cannot be declared LOCAL. Indeed, from a BBC programming point of view, subroutines should not be emphasized. But students will meet GOSUB in the computing literature, and may want to convert a program written for the BBC machine, to a computer with a less sophisticated BASIC (eg Commodore 64). Most popularly-priced computers do not at present have procedures.

A subroutine put near the start of a program, might look like this:

reaches the RETURN at line 50, it resumes at line 510 by printing "WHAT NOW?".

It should also be pointed out that parameters cannot be passed to a subroutine.

The BBC computer can perform "computed" GOSUBs and GOTOs. That is, GOSUB can be followed by an expression which is evaluated to yield the destination line number. Generally however, students should not use these since they cannot be renumbered easily. The following program has curiosity value only. It illustrates the lack of portability of a computed GOSUB. If the BBC RENUMBER statement is used (say RENUMBER 10,10) the expression following GOSUB is left stranded, and the program falls over.

```
 0 X=1 : CLS :GOTO30
 1 PRINT TAB(5,5)"ONE   ":RETURN
 4 PRINT TAB(5,5)"TWO   ":RETURN
 9 PRINT TAB(5,5)"THREE":RETURN
16 PRINT TAB(5,5)"FOUR  ":RETURN
25 PRINT TAB(5,5)"FIVE  ":RETURN
30 GOSUB X*X
40 X = RND(5)
50 delay=INKEY(20)
60 GOTO30
100 REM @@@@@@@ COMPUTED GOSUB DEMO,
    TROUBLE WITH RENUMBER
```

```
10 GOTO1000:REM JUMP OVER SUBROUTINES

19 REM **SUBROUTINE TO REPORT ON
        PRESENT SITUATION**
20 CLS: PRINT''"YOU ARE NOW AT
   POSITION ";P$
30 PRINT;L" LITRES OF FUEL LEFT."
40 PRINT;T" MINUTES REMAIN BEFORE
   DOOMSDAY."
50 RETURN
999 REM****START HERE*****
1000 P$="(2,4)":L=1.6:T=17
2000 T=17:GOSUB20:PRINT''"WHAT NOW?"'''
```

Note that a subroutine simply begins: there is no equivalent of DEF PROC. A REM statement explaining the purpose of the subroutine is often placed at a line number one less than the line which begins the subroutine. For an example of REM used like this, see the NAMES program at the end of Section IV. It is bad practice to use a REM as the first line of a subroutine, as people entering listings often remove REMS from programs to conserve time and memory. Indeed, programs called "REM strippers" are sold, which do just this. The GOSUB, unless altered, will then have nowhere to go. The RENUMBER statement of BBC BASIC will automatically correct GOSUB (and GOTO) statements to suit the new numbers, but cannot insert missing lines.

RETURN is the equivalent of ENDPROC. The program goes back to the next statement after the GOSUB call, even within a multistatement line.

Point out that in line 2000, after setting T=17, the program performs the subroutine beginning at line 20. When it

FUNCTIONS

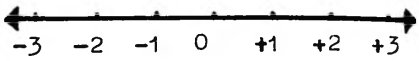
BBC BASIC has so many functions available that for this course we must choose those of most general interest. Certain functions are left for "Computing in Maths" courses. These include the trig and log functions and their inverses, and ABS DIV and MOD.

We suggest teaching INT TIME RND and VAL, and its inverse STR$. Of the many functions used to manipulate strings, LEN, LEFT$ RIGHT$ and MID$ are included, but not INSTR or STRING$. The priorities and interests of a particular group may extend a course in various directions. I have included the most necessary functions here.

INT

The INT function was mentioned earlier as a method of rejecting input which was not an integer. Let's consider it further.

Most students are familiar with the real-number line.



Every dot on the line has a "1-1 correspondence" with a real number.

In BBC BASIC, INT (number) gives the integer which is equal to the number, or just left of it on the line.
So INT (-1.1)=-2
INT(6.9)=6
INT(18)=18
INT(-7)=-7
That is, the INT function "rounds down".

To round to the nearest integer, INT(number + .5) can be used, as shown in these examples:

5.6	INT(5.6 + .5)= INT(6.1)=6
4.3	INT(4.3 + .5)=INT(4.8)=4
-11.6	INT(-11.6 + .5)= INT(-11.1)= -12
-11.2	INT(-11.2 + .5)= INT(-10.8)= -11

TIME

Internally, a computer is a slave to time. All processes are strictly in sequence and governed always by cycles of an internal clock, regulated by the naturally constant periods of vibrating atoms. TIME is a computer-controlled function, updated every hundredth of a second. Here are three common uses which students find interesting.
1. As a delay mechanism.

```
10 MODE7: then=TIME
20 PRINT'"Time flies like an arrow."
30 PRINT'"Fruit flies";
40 now=TIME
50 IF now-then=500 THEN PRINT" like a
   banana.":END
60 GOTO40
```

2. To measure how long a process takes. In this example we compare the time for 1000 procedure calls with the time for 1000 GOSUB calls.

```
10 MODE7:VDU23;8202;0;0;0;:GOTO70
20 DEF PROCtest
30    PRINT TAB(0,5)loopnumber
40 ENDPROC
50 PRINTTAB(0,15)loopnumber:RETURN
60 REM ****START HERE****
70 PRINT''"Here are 1000 PROCEDURE
   calls."
80 TIME=0
90  FORloopnumber = 1 TO 1000
100    PROCtest
110  NEXT
120 PROCresult
130 PRINT'''"NOW LETS USE 1000 GOSUBs
    INSTEAD"
140 TIME=0
150  FORloopnumber = 1 TO 1000
160    GOSUB50
170  NEXT
180 PROCresult:END
190 DEF PROCresult
200 time=TIME: PRINT''"TIME ELAPSED IS
    ";time;" CENTISECONDS"
210 ENDPROC
```

3. To set a time limit on an activity.

```
10 MODE7:VDU23;8202;0;0;0;
20 PRINT'"There are 5 levels of diffi
   culty. Press a number from 1 to 5."
30 PRINT'"1 is easy and 5 is almost
   impossible."
40 *FX21,0
50 choice=GET:level=choice-48:IF level
   <1 OR level>5 THEN 50
60 limit=10000-level*1000
70 topofrange=20*level-1
80 secret=RND(topofrange)
90 PRINT''"GUESS THE COMBINATION WHICH
   WILL DEFUSE THE GALACTIC BOMB."
100 PRINT'"IT IS A NUMBER FROM 1 TO ";
    topofrange
110 TIME=0
120 PRINT'"Enter your guess and press
    RETURN"'
130 PRINT'"You have ";INT(limit/100);"
    seconds."
140 INPUTguess
150 IF guess=secret THEN 180
160 IF TIME<limit THEN VDU11:PRINT"
 ":VDU11,11,11:PRINT;guess;" is WRONG
 ":PRINT"You have ";INT(limit/100)-INT
  (TIME/100);" seconds left  .":GOTO140
170 CLS:PRINT''"THE GALAXY IS DESTROYED.
    ":END
180 CLS:PRINT'''"BOMB DEFUSED. THANKS
    TO YOU THE GALAXY  IS SAFE."
```

RND

On the BBC computer, the RND function has two forms useful in schools.

If RND is followed by a 1 in brackets, the function produces a "random" decimal number from 0 up (but not including) 1. This produces a load of untidy looking decimal places, and is unpopular with students. The INT function is necessary to reduce them to a useful number.

If the brackets contain a larger integer N eg RND(12), a random positive integer from 1 to N inclusive is produced. This is simpler than the RND function of many other computers, but note that no space must be left between RND and the bracket.

A truly "random" number has a value which is completely unpredictable. It is produced by no rule and is unrelated to any previous numbers. Since everything that happens within a computer is determined by natural laws, it is argued that a computer can never produce truly random numbers. Indeed, most computers produce "pseudo-random" numbers. These numbers are on a very long list, often generated by squaring the previous number and re-arranging the digits. This list is eventually repeated. Unless steps are taken, each time a program is run the next numbers on the list will appear. Suppose I switch on my BBC computer and run the following program.

```
20 FOR J=1 TO 6
30 PRINT RND(6)
40 NEXT
```

These are the numbers produced by the first three runs:
4 4 2 1 3 3, 4 2 4 6 1 6, 2 4 3 1 2 4.
If I switch off, re-load the program and repeat, the same numbers appear again.
But if I add the following line:
10 seed = RND(-TIME)
the numbers will be taken from different parts of the list. Some programs would otherwise be spoilt. Suppose a student runs a program designed to help with simple arithmetic. If the student wishes to try again tomorrow it's probably not very valuable if the program serves up the identical questions again.

How long is a piece of string?

LEN is a function which gives the length of a string.
eg LEN("whale")=5 LEN("whale")=6
A$="120" LEN(A$)=3
LEN is often used to search for a string, or to pad a string out to a standard length, for formatting.

In English, the word "add" has several meanings. It can mean "append" (join onto) as happens when "ing" is added to "sing" to form "singing".

This can happen with digits too. If you lived in a suburb with a phone prefix 235 and were allotted the 716th phone, your number would be formed by "adding" to give 235 0716 as your phone number. Here, we are adding two strings. This is called "concatenation". In BBC BASIC, two strings can be concatenated to form a new string.
eg result$= string1$ + string2$
(Note that the plus sign is used, although the & sign is closer in meaning to the word concatenation). "Add" can mean

arithmetic addition too, as in 235 add 0716 equals 951. So sometimes we wish to handle a sequence of digits as a string and sometimes as a numerical value.

VAL

The common use of the VAL function is to change a string consisting of digits, into a number with those digits.
eg VAL("10.7") = 10.7
Students usually want to know why a number was not used to begin with. The point is that there are powerful functions available to manipulate strings. Strings can be "added" (concatenated) and printed in a way in which numbers can't. The disadvantage is that the arithmetic operators do not work with strings. Here is an example:

Suppose a student is compiling a list of noteworthy people who were born between 800 and 1800.
eg Aphra Benn (1640-1689), English Playwright.

It is convenient to enter all these pieces of information as strings (eg name$, birth$, death$, comment$) because of the power of the string functions. Information in string form can be easily sorted, supplemented and extracted.

VAL can be used to extract the number value of a string when required. eg VAL(birth$)= 1640. Then arithmetical facts like life span and how long ago they were born, can be calculated. This next program uses VAL in this way but for simplicity, comment$ has been omitted.

```
10 DIMentry$(100)
20 DIMname$(100)
30 DIMbirth$(100)
40 DIMdeath$(100)
50 DIMspan(100)
60 MODE7
70 PRINT"PLEASE ENTER THE CURRENT YEAR
   eg 1984"
80 INPUTnow:IFnow<1983 OR now>1993 THEN
   PRINT"Don't you know what year this
   is?":GOTO80
 90 CLS
100 FOR K = 1 TO 100
110 PRINT"_____________________________
_____________
    "
120 PRINT"ENTER NAME ";K;" , SURNAME
    FIRST"
130 PRINT'"When the whole name is
    finished,press   RETURN.
    Enter @ instead if  you have
    finished all the names."'
140 INPUTname$(K)
150 IF name$(K) = ""THEN 140
160 IF name$(K) = "@" THEN K=K-1: GOTO
    250
170 INPUT'"ENTER YEAR OF BIRTH
    "birth$(K)
180 IF birth$(K) = "" OR VAL(birth$(K)
    )<800 OR VAL(birth$(K))>1800THEN
   PRINT"BETWEEN 800 AND 1800":GOTO 170
190 INPUT'"FINALLY, ENTER YEAR OF DEATH
    "death$(K)
200 span(K)= VAL(death$(K)) - VAL(birth
    $(K))
```

```
210 IF span(K)<0 OR span(K)>110 THEN
    PRINT"PARDON??" :GOTO190
220 IF death$(K) = "" THEN190
230 entry$(K) = name$(K) + "   " + birth
    $(K) + " to " + death$(K)
240 NEXT
250 CLS
260 FOR J = 1 TO K
270 PRINTentry$(J)
280 PRINT"BORN ";now-VAL(birth$(J));"
    YEARS AGO."
290 PRINT"LIVED FOR ";span(J);" YEARS"

300 PRINT"_____________________________
__________"
310 NEXT
320 INPUT'"WANT TO ENTER MORE NAMES
    (Y or N)",ANS$
330 IF ANS$= "Y" OR ANS$="y" THEN K=K+1
    :GOTO110
```

This program could be expanded to include a "sort" routine to enable the printing of entries in alphabetical order, chronological order or life-span. See the "NAMES" program at the end of section IV for an example of a fast sorting routine.

STR

The STR function in BBC BASIC is the opposite of VAL. It converts a sequence of digits from being a number into being a string.
eg STR$(5.078) = "5.078".

We can't use our newborn string in any calculations, because it is not a number in the maths sense. But we can now use the powerful string functions of BBC BASIC on it.

Cutting a string.

The most versatile function which removes a piece from a string is MID$(string$, start, length)
eg If we have a string
mouthful$ = "antidisestablishmentarianism"
then MID$(mouthful$,9,4) would be "stab" and
MID$(mouthful$,20,3) would be "tar".

However, this may not always be the most convenient string extractor, nor the fastest.

LEFT$(string$, length) assumes you are beginning with the first character of the string.

RIGHT$(string$,length) assumes that you are removing the last part of the string.

In the popular Psycho-analysis game "Eliza", a routine is used to remove apostrophes from words which the "patient" utters. This helps the program in its search for certain words to which it can respond. The following program does this, and illustrates the string function concepts we have been discussing.

```
10 REM REMOVE APOSTROPHES
20 INPUT word$
30 FOR L = 1 TO LEN(word$)
40 IF MID$(word$,L,1) = "'" THEN word
   $ = LEFT$(word$,L-1) + RIGHT$
```

```
   (word$ , LEN(word$)-L):GOTO40
60 NEXTL
70 PRINTword$
80 GOTO20
```

USER FUNCTIONS.

If a lengthy function is needed at several places in a program, (or in a number of different programs) both typing time and memory can be saved by defining the function. This can be done anywhere, even after an END statement. That is, the computer does not need to come across the definition before it can be used.

On the BBC computer a function can involve more than one variable. Variables can be string or numeric, and the definition of a function can extend over several program lines if required. Many versions of basic only allow a single numeric variable to be used, defined on a single line. These are indeed powerful aspects of the BBC machine.

Teacher user-defined functions should begin simply, with a single numeric variable. Most students will be familiar with simple functions of one variable.

As an example, let's define a function which gives the area of a circle in terms of the radius.

```
10 MODE7
20 INPUT"ENTER A VALUE FOR THE RADIUS
   AND PRESS  RETURN "radius
30 DEF FNA(radius) = PI*radius*radius
40 CLS
```

```
50 PRINT'"THE AREA OF A CIRCLE OF
   RADIUS ";radius;'"IS ";FNA(radius)
60 PRINT'"A SPHERE OF THIS RADIUS HAS
   A SURFACE   AREA OF ";4*FNA(radius)
70 PRINT'"THE VOLUME OF A CYLINDER OF
   THIS RADIUS AND ONE UNIT HIGH IS ";
   FNA(radius);''
80 PRINT''"**************************
**********"
90 GOTO20
```

There is no need to use the same symbol in the INPUT statement as in the definition. We could rewrite line 40:

DEF FNA(custard)= custard*custard*PI

If there are many variables, we have to be careful to feed our function the right number and type. For example, don't give it a string if it expects a number. Here is an example of a multiple-variable function.

Suppose a professional photographer wishes to give quotes, do accounts and forward planning for the sale of photographic prints. There are four variables involved in the scheme.

basic	The charge in dollars or pounds or whatever (eg 1.75) made for one standard size print.
number	The number of photos involved.
size	Enlargements from 2X to 10X of the standard format are offered. Since print cost relates to the area of the paper used, these numbers must be squared.
col bw	Colour costs a certain factor (eg 2.5) times as much as black and white. This factor can vary.

Also, a discount is offered which increases gradually to 70% for large orders. A function might be defined as follows:

DEF FNcharge(number,col bw, basic,size)= number*(.3 + .7/n)*col bw*basic*(size12)

As a project, students could write a program using this function. You will need to describe how a user-function is called from within a normal BASIC statement.

eg PRINT FNprofit(cost, sell,tax)

Some further statements

ON

These are three forms of the ON statement. ON ERROR is used to investigate errors (the dreaded "bugs") which occur in programs, and also has a few special uses.

The other two uses of ON are ON GOTO and ON GOSUB. Each of these involves a variable whose value is decided by the program. To be useful, the values of the variable have to be positive whole numbers. Here is an example.

```
 10 REM ONGOTO DEMO
100 CLS
110 INPUT"ENTER 1, 2, 3 or 4 AND PRESS
```

```
    RETURN "number123or4
120 ON number123or4 GOTO 500,200,700,
    600
200 PRINT"TWO":GOTO110
500 PRINT"ONE":GOTO110
600 PRINT"FOUR":GOTO110
700 PRINT"THREE":GOTO110
```

Students should note that the line numbers after GOTO can appear in any order. The full range of numbers from 0 to 32767 may be used. ON GOSUB works similarly, and returns to the statement following the ON GOSUB call.

Unfortunately, ON cannot lead directly to a procedure, but GOTO can be used to direct the program to the line number where the procedure is.

If the variable gets a value which is not one of the program's line numbers, an error will result. Students can try values like -1, 4.5, 20 or 32768 and see that they get the error: ON RANGE at line 120.

ELSE

To avoid the program stopping with an error, the ELSE statement is sometimes appropriate. Suppose we add to line 120: ELSE 100

Now an unsuitable input will be rejected.

ELSE can be used with ON GOSUB in a similar way.

The third use of ELSE is with IF-THEN. It is a clear and neat way of handling program branching, envied by users of many other BASICs. However in BBC BASIC, IF-THEN-

ELSE can only be used within a single program line. Here is an example of the use of IF-THEN-ELSE. Mathematics teachers in particular may find this program useful because it tests numbers to see if they are "prime".

```
10 MODE7
20 PRINT'"This program tests to see
   whether or notthe whole number you
   enter is PRIME."
30 PRINT'"A PRIME number is a whole
   number which  has exactly two
   different factors:  itself and 1."
40 PRINT'"The number 1 is therefore not
   prime."
50 PRINT'"All other whole numbers have
   other       factors as well. If you
   enter one of     these, the program
   will tell you its smallest factor."
60 GOTO110
70 IF number>0 AND number<4 THEN PRINT'
 "****     ";number;" is PRIME.":GOTO110
80 I=0:divisor=2
90 IF number MOD divisor =0 THEN PRINT'
   '"The smallest factor of ";number;"
   is ";divisor:GOTO110 ELSE I=I+1:
   L=divisor *divisor
100 IF L>number THEN PRINT'"****    ";
    number;" is PRIME.":GOTO110 ELSE
    divisor= 2*I+1:GOTO90
110 PRINT'"-------------------------
----------"
120 PRINT"Enter a number and press
    RETURN. If you wish to stop, enter
    zero."'
```

```
130 INPUT number
140 IF number=0 THEN END ELSE 70
```

REPEAT-UNTIL

REPEAT (a set of operations) UNTIL (a condition becomes true).

Students find REPEAT-UNTIL easy to understand. Teachers often use it on them, as in "Carry on with the exercises (until I tell you to stop)". This is frequently coupled with an IF-THEN-ELSE. "If you behave yourself then I will be satisfied with you, ELSE you will be in trouble!"

(Repetition conjoined with threats of punishment seems a rather sour summary of teaching. Yet students seem to be so familiar with these concepts).

Let's begin with the simplest use of REPEAT-UNTIL, to produce an endless loop. The same task performed by a GOTO is presented for comparison.

```
10 REM USING REPEAT UNTIL
20 N=1
30 REPEAT
40   PRINT"NONGNONG ";N
50   N=N+1
60   delay=INKEY(50)
70 UNTIL FALSE
```

```
10 REM USING GOTO
20 N=1
30 PRINT"NONGNONG ";N
40 N=N+1
50 delay=INKEY(50)
60 GOTO30
```

The structured form is easier to follow, especially if the lines to be repeated are inset. The bunch of operations between REPEAT and UNTIL FALSE is performed over and over indefinitely, because FALSE is never true. Only the ESCAPE or BREAK key will stop the loop.

(Although UNTIL 0 is equivalent to UNTIL FALSE, it is not as comprehensible to students).

This next program illustrates that REPEAT-UNTIL terminates on the achievement of a specific condition, compared with FOR-NEXT, which specifies a certain number of loops. —>

```
10 MODE7
19 REM DISABLE ESCAPE KEY
20 *FX229,1
29 REM REDEFINE BREAK KEY TO RUN AGAIN
30 *KEY10 OLD|M RUN|M
40 REPEAT
50   PRINT"I WILL KEEP RUNNING UNTIL
     YOU TELL ME   OTHERWISE"
60   INPUTmessage$

70 UNTIL message$="OTHERWISE" OR
   message$="otherwise"
89 REM RE-ENABLE ESCAPE KEY
90 *FX229,0
```

```
 99 REM RE-ENABLE BREAK KEY
100 *KEY10
110 PRINT"PHEW!!"
```

BBC BASIC allows a maximum of 20 REPEATs without reaching an UNTIL. This is plenty of course, except when UNTIL is skipped a lot. The following program will show students what happens then.

```
10 MODE6
20 CLS:num=0
30 *FX21,0
40 PRINT'"If you press THE SPACE bar 20
   times you will see the error occur."
50 REPEAT
60 key=GET
70 IF key=32 THEN 90
80 UNTIL FALSE
90 num=num+1:PRINT TAB(2,10)"You pressed
   SPACE ";num;" times."
100 GOTO50
```

Here is a program which many students find interesting. This sort of program structure can easily go wrong and stop with the TOO MANY REPEATS error, so care is needed. All it takes is 20 jumps out of the loop to occur.

```
10 MODE4
20 CLS:right=0:guesses=0
30 PRINT'"THIS PROGRAM TESTS TO SEE IF
   YOU HAVE   PSYCHIC POWERS. TRY TO
   GUESS THE NUMBER FROM 1 TO 9 WHICH
   I AM THINKING OF."
```

```
40 PRINT''"If only chance is operating,
   your        'performance factor' will
   be close to 11"
50 PRINT''"If you have made over 100
   guesses and   your factor is
   consistently over 20,     you are
   truly psychic!"
60 REPEAT
70 guesses=guesses+1
80 compnum=RND(9)
90 PRINT''"**************************
***********"
100 PRINT'"GUESS NUMBER ";guesses
110 PRINT'"PRESS ONE OF THE KEYS 1 TO 9"
120 yournum=GET-48
130 IF yournum<1 OR yournum>9 THEN 120

140 PRINT'"I see you picked ";yournum
150 IF compnum<>yournum THEN PRINT'"No
   ,  I thought of ";compnum ELSE PRINT'
    "You picked it!":right=right+1
160 PRINT"That's ";right;" out of ";
    guesses;" guesses."
170 PRINT'"Your performance factor is
    ";INT(100*right/guesses)
180 UNTIL FALSE
```

VDU codes: An overview for students

This is a good time to introduce the most versatile statement·

in BBC BASIC. The BBC micro controls the Video Display Unit and the printer by means of VDU followed by a code number from 1 to 255. Many of the codes are followed by yet more numbers. These extra numbers are separated by either commas or semicolons depending on whether the numbers are part of a 0-255 set, or a 0-65535 set. (That is, whether the number can be sent quickly as one byte or two bytes are necessary.)

The VDU code system evolved a little haphazardly into its present form, and so the numbering system is not completely logical. I have grouped the codes as far as possible according to function.

Control of the Printer

1. Only the printer is to accept the next character. VDU1 does not let the BBC micro operating system see the next number, because its code numbers and the printer's code numbers signify different things.
2. The printer is now "listening".
3. The printer is not "listening". This is sometimes necessary. eg we may want to clear the screen with CLS or VDU12. To the printer however, this could be an instruction to "form-feed" ie roll the paper through one page length.

The Screen display

4. Print text at the text cursor.
5. Print text at the graphics cursor.
14. List the program a screen-full at a time. There is a slight overlap between screens. The SHIFT key is used to move on. ESCAPE is pressed to enable editing to happen, and the screen scrolls a little. A few dummy REMs are needed at the start of a listing to gain access

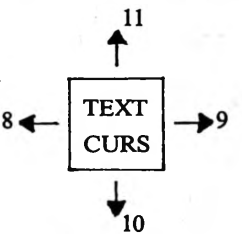
to the first few lines of a program.

15. Undoes VDU14. This, or CONTROLO is needed when using the printer. Otherwise if in page mode, the printer too will stop every page.

Enabling the computer to "press its own keys"

8
9
10.
11.

The text cursor can be controlled from within a program.



13. Moves the cursor to the left end of the text line.
30. Moves the cursor to the top left of the text area.
31. Acts like PRINT TAB, being followed in the same way by the coordinates of the desired print position.
127. Backspaces like VDU8 but deletes as well. ie like the DELETE key. This is very useful in some GCOL operations with VDU5.

32-136. Represents all the letters and numbers on the keyboard. See the ASCII chart earlier in this section for a full list.

Codes equivalent to text and graphics keywords

12	16	17	18	22	25
CLS	CLG	COLOUR	GCOL	MODE	PLOT

Some BASIC programmers prefer to use these VDU code equivalents because they can be strung together compactly. This saves typing time and memory. However, some people have trouble with the commas and semicolons involved.

Emitting a beep

7. Used to alert the user that a particular stage in the program has been completed.

Changing the palette

19. Choosing new subsets of colours from the 16 in the BBC micro repertoire.

20 Returns to the "default" colours. ie those provided when the machine is switched on.

The remaining codes are not part of this first course in BASIC for students, but are included here for completeness.

User-defined characters

23. Used in conjunction with 244-255 (and others)

Windows

Rectangular subsets of the text and graphics screens can be defined. At first, both text and graphics screens occupy the whole of the available screen area.

24. Set graphics windows. Semicolons are used because the numbers which have to be sent are part of a two-byte system. ie many graphics coordinates exceed 255.
28. Set text windows. Only one byte is needed so commas are used.

26. Reset the graphics and text windows to whole-of-screen.

Moving the graphics origin

29. Semicolons are used because the numbers are part of a two-byte system.

Teaching the use of colour.

So much could be said on this topic. Some restriction is needed to retain proportion in the BBC BASIC course. In many schools, only Model A machines are available. In others, monochrome video is used. So when considering the scope of this BBC BASIC course for students, it was decided to deal only with MODE5 in detail. MODE5 is available on the unexpanded model, and colours can be chosen which can also be distinguished in monochrome. The flashing effects are worthwhile too.

Here are the eight colours and eight flashing effects in the repertoire of the BBC computer. They are referred to by the numbers shown.

The list of sixteen

0	black	8	flashing black-white
1	red	9	flashing red-cyan
2	green	10	flashing green-magenta
3	yellow	11	flashing yellow-blue
4	blue	12	flashing blue-yellow
5	magenta (blue-red)	13	flashing magenta-green
6	cyan (blue-green)	14	flashing cyan-red
7	white	15	flashing white-black

MODE 5

This mode has 32 rows and 20 columns of text. A maximum of four different colours can be used at one time. (These can be mixed together. See the QUILT Program in the Little Library, section III). Medium resolution is possible on the graphics screen, there being 160 Pixels (lightable areas) across the screen and 256 down. User defined characters are also available in this mode. When the computer is switched on and put into MODE5 the text and graphics "screens" overlap completely. At this stage the four colours in use are:
0 black, used as background
1 red
2 yellow
3 white, used as the foreground colour for both text and graphics screens.

Any subset of four can be chosen from the list of sixteen using VDU19 as discussed shortly.

Altering foreground and background colours on the text screen

The COLOUR statement is used to choose new text foreground or background colours from the four presently available. COLOUR is followed by one of the numbers 0,1,2,3, for the foreground and 128, 129, 130 and 131 for the background. The CLS statement clears the text screen to whichever background colour is the current choice. Here is a program which students can use to investigate the points made so far.

```
10 MODE5
20 COLOUR1:COLOUR130
```

```
30 CLS
40 PRINT'"The MODE statement  in line 10
   makes thescreen BLACK."
50 PRINT'"COLOUR1 makes the   foreground
   RED.That is, the textis printed
   in RED."
60 PRINT''"COLOUR130 gives eachtext
   character       (even the spaces) a
   yellow background."
70 PRINT'"However, the screen will still
   be BLACK unless CLS is used  to make
   it YELLOW   all over."
80 PRINT''"Remove line 30 to   see this
   effect."
```

Changing the set of four colours.

The default set of four is changed by VDU19 followed by five numbers separated by commas. The last three are always zero, and allow for possible expansion of the system by Acorn. Even though they are always zero, they must be included. Beginners often omit them.

The first number after the 19 is 0, 1, 2 or 3. These are called the "logical" colour numbers. The second number is always taken from the list of 16 which we have just seen. Whichever number we choose from the list replaces one of the "gang of four" presently in use.

For example VDU19,0,4,0,0,0 changes the background colour from black(0) to blue(4). The "gang of four" logical colours is now blue, red, yellow and white. This is the new "palette".

Let's summarise this with a diagram:

A note on other modes

This table shows the logical colour numbers which are used as foreground and background in each mode.

MODE	Background	Foreground	Others available
0,3,4,6	0	1	nil
5,1	0	3	1,2
2	0	7	1,3,4,5,6

We have found however that students should stick to MODE5 until they understand the ideas. This is particularly true when learning about the graphics colours.

Here is a program to illustrate the changing of the palette in MODE5.

```
10 MODE5
20 PRINT'"MODE 5 IS A FOUR     COLOUR
   MODE AND NORMALLY THE COLOURS ARE:"
```

```
30 PRINT'"0 BLACK","1 RED"
40 PRINT'"2 YELLOW","3 WHITE"
50 PRINT'"Always, COLOUR0 is the
   background colour. At present it is
   black but we are about to change it
   to yellow."
60 PRINT'"Always, COLOUR3 is the text
   colour. At the moment it is white
   but we are about to change it to
   green."
70 PROCanykey
80 PROCchangepalette
90 CLS:PRINT'"The palette is now:"
100 PRINT'"0 YELLOW","1 RED"
110 PRINT'"2 BLUE ","3 GREEN"
120 PRINT'"NOTE:Always, COLOUR 0 IS THE
    BACKGROUND COLOUR"
130 PRINT'"NOTE:Always, COLOUR 3 IS THE
    TEXT COLOUR."
140 PRINT'"Now let's change the text
    colour to blue and the background
    to cyan using this palette."
150 PRINT'"Look at the listing to see
    how this is done."
160 PROCanykey
170 COLOUR1
180 COLOUR130
190 REM
200 PRINT'"Now the writing is in red
    on a blue background."
210 PRINT'"The screen colour is still
    yellow at this stage."
220 COLOUR0:COLOUR129
230 PRINT'"Now it's yellow on a red
    background."
```

```
240 PRINT'"Change line 190 to  CLS and
    re-run the  program. What will
    happen now?"
250 END
260 DEF PROCchangepalette
270 VDU19,0,3,0,0,0
280 REM** WE ARE LEAVING COLOUR 1 AS RED
290 VDU19,2,4,0,0,0
300 VDU19,3,2,0,0,0
310 ENDPROC
320 DEF PROCanykey
330 PRINT'"PRESS ANY KEY TO SEE THIS
    EFFECT"
340 *FX15,1
350 key$=GET$
360 ENDPROC
```

GCOL and PLOT

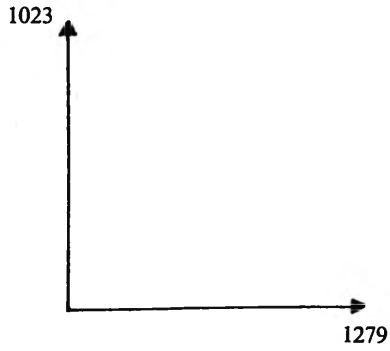
Producing graphics with PLOT

The PLOT statement can be used in MODES 0, 1 2, 4 and 5 to draw points, lines and shaded figures.

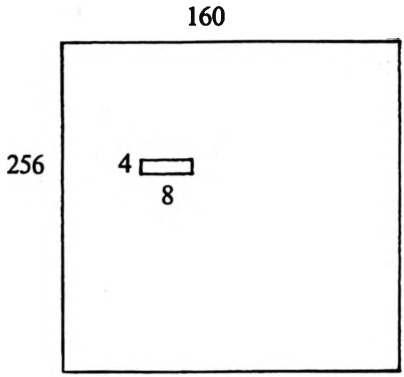
The full graphics screen is slightly wider than it is high, and divided into imaginary tiles or "pixels" which are all alike. That is, the graphics screen is sliced into zones by horizontal and vertical lines. But ulike the graph paper which students are used to, the horizontal line spacing need not be the same as for the vertical lines. On the graphics screen a zone is rectangular rather than square, and varies in proportion from MODE to MODE. Also of course the lines on the screen are not visible like the ones on graph paper. For each

graphics mode we can choose the colour of each pixel, from the colours in the palette being used. Consistent with earlier work, we will discuss only Mode 5 in detail.

There are 40960 Pixels in mode 5: 256 rows of 160 little rectangles, as shown below. The number of Pixels used to cover the graphics screen varies with the mode used. Hence the PLOT statement uses a system of coordinates which allows graphics designs to transfer between modes (although proportions do alter). The graphics origin is normally at the lower left, but can be changed. (See the Spiral program in the Little Library, Section III.)



The graphics axes



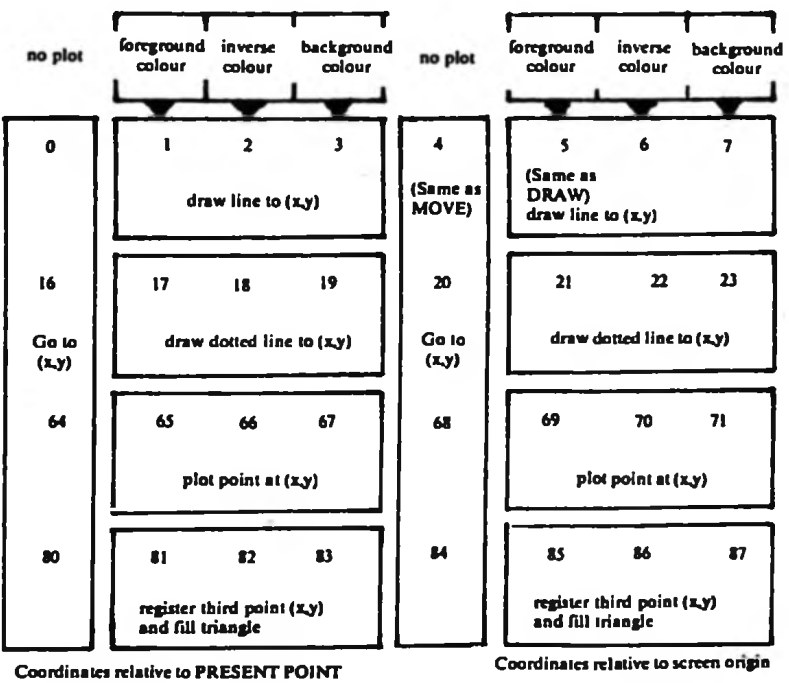
Mode 5 pixels

In MODE5, each little pixel is thus four points high and eight points wide. A MODE5 pixel will respond if any of its 32 points are addressed. Here is a program which illustrates these ideas:

```
10 MODE5
20 PRINT'"The moral of this   story is
   that any   Point in a pixel can act
   as a switch."
30 PRINT'''"The flashing pixel  is the
   fifth across from the origin and the
   sixth one up."
40 PRINT''"Try changing the 69 in line
   50 to a 65, and the 71 in line  70 to
   a 67. What do you expect to see?"
50 PLOT69,31+RND(8),19+RND(4):REM Turn
   me on.
```

```
60 FOR delay=1 TO 3000:NEXT:REM Gimme
   a break willya?
70 PLOT71,31+RND(8),19+RND(4):REM Turn
   me of.
80 FOR delay=1 TO 3000:NEXT:REM Gimme
   a break willya?
90 GOTO50
```

Your students will more easily follow the meanings of the PLOT numbers in this section if they have a copy of this PLOT CHART.



Having plotted points, let's move on to the drawing of lines. We start with PLOT4 (or VDU25,4 or MOVE) which simply registers the first pair of coordinates in memory. Then PLOT5 is used to provide the other pair of coordinates and draw the line in the current foreground colour.

Point out to students that the right hand side of the PLOT chart deals with coordinates relative to the screen origin. We recommend they stick to these at first.

Here is a program which shows two equivalent methods of drawing a line. If using VDU25, students must remember to use semicolons after numbers which belong to a two-byte system ie in which a number greater than 255 may be involved. Graphics coordinates are in this category.

```
10 REM LINES
20 MODE5 : REM or VDU22,5
30 PLOT4,1000,1000 :REM or MOVE1000,
   1000
40 PLOT5,100,100
50 PROCdelay
60 CLG
70 PROCdelay
75 REM THIS NEXT LINE DOES THE SAME JOB
80 VDU25,4,1000;1000;25,5,100;100;
90 PROCdelay
100 VDU16 : REM EQUIVALENT TO CLG
110 PROCdelay
120 GOTO30
130 DEF PROCdelay
140 FOR delay=1 TO 3000:NEXT:ENDPROC
```

Drawing lines is much faster with PLOT5 than using a loop to light up individual pixels. Similarly, PLOT85 very quickly fills in an area, as the next program illustrates.

```
10 MODE5
20 limit=1000
30 PLOT4,400,10:PLOT4,10,200:PLOT85,
   10,400
40 FOR delay= 1 TO limit :NEXT:REM
   See what happens
50 PLOT4,400,600:PLOT85,800,400
60 FOR delay= 1 TO limit :NEXT:REM
   See what happens
70 PLOT4,800,200:PLOT85,400,10
80 PRINT'"Try changing the    limit to
   0 instead  of 1000."
90 PRINT'"How could you fill  the
   centre in?"
```

Students are usually very interested in experimenting with graphics. They will enjoy the novelties to be discovered in the PLOT CHART.

Altering foreground and background colours on the graphics screen.

The statement used here is GCOL0,n where n is 0 to 3 for foreground colours and 128 more than these for background colours. This is similar to the COLOUR statement used for text.

Alternatively, some teachers prefer VDU18,0,n (or VDU18;n). This is equivalent to GCOL0,n but is more compact, since VDU statements can be strung together.

CLG is the statement used to clear the graphics screen. VDU16 is the equivalent of CLG.

Some teachers prefer their students to use VDU for everything. However, commas and semicolons and missing zeros can cause problems. It could also make the program less readable to others.

Here is a program which illustrates the use of GCOL0 and VDU19 (to change palettes). It is most effective on a large colour TV in a darkened room. Some people have uncomfortable reactions to strobe effects. Ascertain if there are problems before using any program which has flashing lights.

```
10 MODE7
20 PRINT''"Warning. Do not expose any
   one sensitive to strobe effects to
   this program."
30 PRINT'''"To adjust the flash rate
   of the DISCO   CUBE to suit your
   music, press F for    faster or
   S for slower."
40 *FX21,0
50 PRINT''"Press any key to start.":
   anykey$=GET$
60 limit=500
70 MODE5
80 K=1.6:REM ALTER THIS VALUE TO CHANGE
   CUBE SIZE
```

```
90 VDU23;8202;0;0;0;:REM SUPPRESSES
   CURSOR
100 REM PAINT THE LEFT SIDE IN LOGICAL
    COLOUR 3 AS SET BY LINE 140
110 MOVEK*250,K*100:MOVEK*100,K*285:PLOT
    85,K*100,K*550:MOVEK*250,K*400:
    PLOT85,K*250,K*100
120 REM PAINT THE RIGHT SIDE IN LOGICAL
    COLOUR 1
130 GCOL0,1: PLOT85,K*500,K*250:MOVEK*
    500,K*510:PLOT85,K*250,K*400
140 REM PAINT THE TOP SIDE IN LOGICAL
    COLOUR 2
150 GCOL0,2: PLOT85,K*100,K*550:MOVEK*
    340,K*620:PLOT85,K*500,K*510
160 REM CHOOSE A NEW PALETTE AT RANDOM
170 VDU19,1,RND(8)-1,19,2,RND(8)-1,19,
    3,RND(8)-1:FORdelay=1 TO limit:NEXT
180 IF INKEY(-68) THEN limit=limit-10:
    IF limit<0 THEN limit=0
190 IF INKEY(-82) THEN limit=limit+10:
    IF limit>10000 THEN limit=10000
200 GOTO170
```

The other forms of GCOL are beyond the scope of a first course for students.

The Sound of Music?

Can the BBC micro be used to teach music? The ROM software dealing with sound is very comprehensive. However, the hardware is not good enough to take full advantage of it. The tiny speaker is inadequate in clarity and frequency response. The maximum loudness available is insufficient, and cannot be manually controlled. The computer emits a lot of electronic "noise". For the computer to be useful in the teaching of Music as a subject the sound output needs to be interfaced to a hi-fi system which should include headphone sets. Even then, many Music teachers would be dissatisfied with the performance of the BBC computer sound chips.

The rather random experiments with the SOUND and ENVELOPE statements which computing students usually do, have curiosity value only. Concepts like "attack" "sustain" etc are best taught using sound and animated colour graphics together. Professional software is available which plays and graphically illustrates the more sophisticated aspects of the SOUND and ENVELOPE statements. For these reasons our BBC BASIC course omits ENVELOPE and discusses the SOUND statement only in its "simple" form. The synchronisation of notes to form chords is demonstrated by the "Organ" program in the Little Library, Section III.

The SOUND statement is followed by four numbers. The first (called C) specifies one of the four available sound "channels".

Channels 1,2 and 3 are identical in function, each producing single notes.

The second parameter (A) controls the loundess. When the SOUND statement is used in its simple form, A ranges from -15 (loudest) to 0 (silent).

The next parameter (P) is the "pitch", or position on the musical scale. It ranges from 0 to 255. Every increase of 48 raises the note one octave. A pitch of 53 is "middle C". The next program allows variable control over both pitch and loudness while displaying the values being used.

```
10 MODE7
20 VDU23;8202;0;0;0;
30 PRINT'"Hold down P to vary pitch
   with the UP & DOWN cursor controls."
40 PRINT'"Hold down L to vary loudness
   with the   UP& DOWN cursor controls."
50 PRINTTAB(2,10)"The pitch value is now"
60 PRINTTAB(2,15)"The loudness value
   is now"
70 loud=-6:pitch=150
80 IF INKEY(-58) AND INKEY(-56) THEN
   pitch=pitch+1:IF pitch>255 THEN
   pitch=255
90 IF INKEY(-42) AND INKEY(-56) THEN
   pitch=pitch-1:IF pitch<0  THEN
   pitch=0
100 IF INKEY(-58) AND INKEY(-87) THEN
    loud=loud-.5:IF loud<-15  THEN loud
    =-15
110 IF INKEY(-42) AND INKEY(-87) THEN
    loud=loud+.5:IF loud>0 THEN loud=0
120 SOUND1,loud,pitch,1
130 PRINT TAB(24,10)pitch
140 PRINTTAB(24,15)loud
150 GOTO 80
```

The last parameter (D) is the duration or time interval of the sound. D=20 means the note lasts for one second. Students should experiment with these parameters for themselves. The most frequent mistake is to use a positive.

Channel 0 is used for special effects, which are decidedly unmusical. For many students, this is the "best" channel, and they are keen to investigate its properties.

The Parameters A and D are the same as for the other channels. But P is quite different and takes the values 0 to 7. To keep the course in proportion, the values 3 and 7, used in conjunction with Channel 1, have been omitted.

The effects of the other P values are described in the table below, and illustrated by the following program.

```
10 CLS : READ P
20 IF P = -1 THEN PRINT TAB(5,10)"END
   OF TEST" : END
30 PRINT TAB(5,5)"THE VALUE OF P IS";P
40 SOUND 0,-15,P,50
50 X = INKEY(500)
60 GOTO10
70 DATA 0,1,2,4,5,6,-1
```

Table of P values Channel 0

	Low	Medium	High pitch
Buzzer effect	0	1	2
Static effect	4	5	6

As an exercise, students could use P=4 and vary the loudness using a FOR-NEXT loop to create the sound of breaking glass.

AND, OR, EOR and NOT as logical operators

In this student course, only the "logical" sense of these operators needs to be taught. Their "bit-wise" use is necessary for GCOL3 etc which can be seen in the "Train" program, Section III.

AND

Here is an example for students.

Suppose Freddie is waiting for a train to arrive. The train may be carrying his friend Janie, who may have a pet frog with her. In order for Freddie to see the frog, Janie must be on the train AND she must have the frog with her. Both must be true. Now if the train arrives but Janie is not on it, we do not need to know if she has the frog. We know Freddie does not see it. But BBC BASIC tests both conditions anyway. The following program illustrates the situation. It makes use of FALSE=0 and TRUE=-1.

```
10 MODE7
20 seed=RND(-TIME):A=RND(10):B=RND(10)
30 PRINT''"For success, the Janie factor
   must be   less than 5 AND the frog
   factor must    be less than 8. On
   this occasion:"
40 PRINT'"The Janie factor= ";A;" The
   frog factor= ";B
50 IF A<5 THEN Janieontrain=-1 ELSE
   Janieontrain=0
```

```
60 IF B<8 THEN hasfrog=-1 ELSE hasfrog
   =0
70 IF Janieontrain AND hasfrog THEN
   PRINT''"Freddie sees the frog." ELSE
   PRINT''"Bad luck Freddie."
80 IF A>4 THEN PRINT"Janie was not on
   the train."
90 IF B>7 THEN PRINT"Janie did not have
   the frog with her."
100 *FX21,0
110 PRINT''"To see what happens next
    week, press any key.":key$=GET$:
    GOTO10
```

A series of conditions can be used, all of which must be true for the whole chain to be true.

eg IF cost<20 AND funcs>=cost AND colour=green AND wheels=2 THEN Purchase=-1.

OR

This logical operator means that if X is true or Y is true, or both are true then (X OR Y) is true. It is known as an "inclusive" use of the word "or". As an example suppose Alice chooses a number from 1 to 20 from a hat. If the number is odd or greater than 15 she wins a prize. The number 17 satisfies both conditions but only one is necessary, so 3 and 16 are winners too.

EOR

The word "or" can also be used in an "exclusive" sense. (A exclusive or B) is true when A is true or B is true but not both.

Suppose a restaurant offers a set-price menu which offers for main course, a meat dish, or fish, or poultry. We would assume that the word "or" here is meant "exclusively". EOR is used much less than OR, and then usually to manipulate bits (see Section III for its use in GCOL3). Most BASICs do not even provide EOR. Here is a program which illustrates OR and EOR.

```
10 CLS
20 PRINT TAB(0,2)"Two numbers are chosen
   independently    from the set 1,2,3."
30 PRINT''"The conditions being tested
   are:"
40 PRINT'"The first number is less than
   the second"
50 PRINT"The first number = 2"
60 PRINT"--------------------------
---------"
70 firstnum=RND(3):PRINT''"First number
   = ";firstnum
80 secnum=RND(3):PRINT"Second number
   = ";secnum;'
90 IF firstnum<secnum OR firstnum=2
   THEN PRINT"The OR  condition is
   satisfied "
100 IF firstnum<secnum EOR firstnum=2
    THEN PRINT"The EOR condition is
    satisfied."
```

```
110 IF firstnum=2 AND secnum=3 THEN
    PRINT'"Note that OR accepts the
    number pair (2,3) but EOR does not."
120 PRINT''''"PRESS ANY KEY FOR MORE."
130 *FX21,0
140 key$=GET$:GOTO10
```

Sometimes the word "or" is ambiguous. For example, a parent may say to a student who is learning for important exams:

"Until the exams are over, you may go out on Friday or Saturday night only."

The parent might well mean "one night a week" but it could interpreted as two. Because the English word "or" is subject to interpretation, BBC BASIC uses the invented word EOR.

NOT

This is also rarely used. The ELSE statement and <> are more common ways of negating a simple condition. The structure
...IF A<>5 THEN...
or ...IF A=5 THEN (event 1 happens) ELSE (event 2 happens) .. is more usual than
...IF NOT(A=5) THEN...

Sometimes however, the testable condition is quite complex, and NOT is needed to negate the condition. For example, suppose a certain product is being safety tested by measuring various parameters. Then a complicated function called

FNsafe is defined. It may be necessary to use a line like IF NOT FNsafe THEN PROCremedy

Precedence among operators

A full list is given on page 144 of the User Guide (1982 Ed). NOT is stronger than AND. OR and EOR are the weakest of all the operators. Whichever of these comes first in a program line, gains precedence. To change this pre-ordained order of supremacy, brackets can be used.

Example

People are assembled for a club meeting, but Sue (President), Ann (Secretary) and Tim (Treasurer) have not yet arrived. Suppose the Club Constitution requires the following: IF Sue arrives OR Ann arrives AND Tim arrives THEN the meeting can start. The next program sets out this situation and the students can choose various combinations of arrivals. In lines 80 to 100, -1 can be omitted. Also THEN can be omitted wherever it appears. These things allow the OR and EOR logic to be more easily seen.

```
10 REM OPERATOR PRECEDENCE
20 CLS : *FX21,0
30 Suearrives=0:Annarrives=0:Timarrives
   =0 : Sue$="" : Ann$="" : Tim$=""
40 PRINT'"Do you want Sue to arrive?
   (Y or N)": Sue$=GET$ : IF Sue$="Y" OR
   Sue$="y" THEN Suearrives=-1
50 PRINT'"Do you want Ann to arrive?
   (Y or N)": Ann$=GET$ : IF Ann$="Y"
```

```
   OR Ann$="y" THEN Annarrives=-1
60 PRINT'"Do you want Tim to arrive?
   (Y or N)": Tim$=GET$ : IF Tim$="Y"
   OR Tim$="y" THEN Timarrives=-1
70 CLS : IF (Suearrives+Annarrives+Tim
   arrives)=0 THEN PRINT'"None of them
   arrive. What a fizzer!"
80 IF Suearrives=-1 THEN PRINT'"Sue
   arrives."
90 IF Annarrives=-1 THEN PRINT'"Ann
   arrives."
100 IF Timarrives=-1 THEN PRINT'"Tim
    arrives."
110  REM The next line is the point of
     the exercise.
120 IF Suearrives OR Annarrives AND Tim
    arrives THEN PRINT'"The meeting
    starts" :END
130 PRINT'"The meeting cannot start."
```

The AND is more powerful than the OR. Hence if only Sue arrives the meeting can start. If only the other two arrive it can start. All three arriving is O.K.

But suppose brackets are inserted thus:

IF (Sue arrives OR Ann arrives) AND Tim arrives THEN... Ask the students to change line 120 and RUN the program again. Now they will find that Tim must be present. So must at least one of Ann or Sue, before the meeting can begin.

PROGRAM FAILURES

Six ways in which a computer program can fail to do as you want.

1 – Typographical errors

When typing program lines, the user must be exact. Here are some common typos.

The letter O instead of the number 0. Beginners often type things like 3O instead of 30 for a line number. In BBC MODE 7 the zero does not have a slash. Discourage students from using this mode for typing-in, if they have trouble discriminating between the appearance of O and 0. (0 is more diamond-shaped.)

I and 1 are often confused when a learner copies in a program without any understanding of meanings. Discourage the use of I as a variable name. It's not helpful to use single letters for variables anyway. O is out too, as are U and V because they are too similar.

Learners sometimes do not appreciate the crucial programming difference between the : and ; symbols.

Improper spelling of a keyword. The computer has no idea what a PRIMT is.

2 – Not pressing RETURN

Failure to press RETURN at the end of the each program

line causes lines to run together. This will produce a syntax error, or may deprive the program of a line it needs to GOTO or GOSUB.

3 - The RENUMBER statement

RENUMBER is a very useful aid to programming. However, students should avoid "computed" GOTOs and GOSUBs as RENUMBER cannot adjust them.

4 - Programming errors

Sometimes the operations you prescribe are impossible because of some unforseen circumstance. For example, you may be expecting the computer to divide x by y, even though y may sometimes be zero. Range errors, overlapping loops and type mismatch fall into this category.

5 - Syntax errors

These usually occur when a wrong combination of letters and symbols is used in a BASIC statement. To us, what seems a trivial infraction of the rules of spelling or punctuation is totally unacceptable to the BASIC Interpreter of the BBC computer. Spaces can cause syntax problems. For example, if a space is left before the bracket in PRINT TAB() a "no such variable" error will appear.

```
10 CLS
20 PRINT TAB(2,2)"LEAVE NO SPACE BETWEEN
   TAB AND ("
30 PRINT TAB (2,4)"THIS WILL NOT BE
   PRINTED"
```

Typing keywords in lower case causes syntax errors. Most Beeb users have typed in run instead of RUN and been greeted by "MISTAKE". It would be possible to design for the computer a "Basic Interpreter" which was not as fussy, but it would be slower. The BBC microcomputer is renowned for its processing speed.

Another common student error is the omission of the final quote mark in a PRINT statement.

6 – Conceptual errors

The program runs without crashing, but it is not doing what the student intended. Back to the drawing-board (and the Manual). It could even be that another bug has been unearthed in BBC BASIC.

Give encouragement.

Sometimes a student works with great dedication writing a program. It crashes, producing clouds of disappointment and frustration. Students sometimes criticise themselves very harshly. Be sympathetic. Programming can be a very lonely activity. The student needs to be aware that everyone makes mistakes and that progress is rarely possible without them. The fascinating thing about programming is that at any level of expertise, one rarely leaves a session at the keyboard without learning something new. And designing a successful program can be extremely satisfying.

The computer reacts to a faulty program in three ways.

Firstly, the computer can "crash".

This happens when the program has interfered with the operating system. Emphasize to students that nothing they type can possibly damage the computer. A program which crashes does not damage the hardware. After a crash, possibly only the BREAK key will work. Advise the student to then type OLD and press RETURN. It may still be possible to list the program. If not, simply LOAD the program again. (A student who does not SAVE a finished program before trying to RUN it, will soon learn). When examining the listing for the error(s), look for misuse of operating system codes. Here is a sample of a program which will "crash".

```
10 PRINT"I WON'T CRASH, YOU CAN'T MAKE
   ME"
20 VDU21,22
30 PRINT"SEE, I DIDN'T CRASH!"
```

Secondly, the computer may stop and produce an error message

The computer states the number of the last complete line it was able to execute. You might even get a helpful message, like
MISSING " or
Type mismatch.

See page 474 of the User Guide (1982) for a full list of error messages. While the computer is stopped in mid-stream, there are two things the student can do. Firstly, list the lines in the vicinity of the stopping point. The problem is sometimes quite remote from this point however, and a hard copy of the listing is often helpful.

Then use the direct mode to print the values the variables had when the crash came. Look for strange values. You may find that a variable has a negative or fractional value, or a very large value, which it is not supposed to have. Or a string variable may be unexpectedly null because an input routine somewhere was wrongly skipped. Since the BBC computer has no DUMP routine which prints out the values of all variables, the user has to print them all out individually. Hence a list of variables and their expected ranges should always be kept as a student develops a program.

The third way a computer reacts to a problem is by "hanging". It is off in never-never land, gathering daisies. There can be no complaint of course, since the computer is following the programmer's instructions. Usually the students have unwittingly asked the computer to perform a task it can never complete, or at least not in their lifetimes. The main culprits when student programs hang; are IF-THEN, REPEAT-UNTIL, FOR-NEXT-STEP, and overlapping loops. As an example, see "Spiral" in the Little Library, Section III.

Programs like this are faulty from the user's point of view, but do not grind to a halt. The computer may not respond to the normal keys. Sometimes the screen is blank. It will seem as if the computer has gone out to lunch for an indefinite period. The student should stop proceedings using ESCAPE. The line number just completed will appear, and should be written down. There will of course be no error message

because there was no error. Examine the listing and print the variables as before. They can be altered by using LET, if one wishes to experiment. The program can be resumed with these variable values by using a GOTO to the line which the program is to execute next, not necessarily the next number in the listing. (Some computers have a CONTINUE statement, which is easier to use.) If RUN is used instead of GOTO XX, the variable values will be cleared. BBC BASIC does not have the RUN XX ability so, RUN must always start at the first line of a program.

Handling conceptual errors with STOP and TRACE

STOP is a BASIC statement often described as an "introduced bug". STOPs can be inserted at strategic places in a program. The user can discover what the computer is actually doing, which can be quite different from what they had in mind. When the computer comes across a STOP statement it ceases running and prints the line number it was executing. Then the processes of LISTing, and PRINTing or altering variables can be done before using GOTO (line number) to resume the program. Students sometimes forget to remove all the STOPS, and unfortunately BBC BASIC lacks a FIND statement. So students are advised to write down the locations of all the STOPS they insert. The TRACE statement can be useful to students but only to embrace a few lines at a time. TRACE ON can be inserted, and a few lines later, TRACE OFF. Students usually find the general application of TRACE bewildering. Everything happens so fast, and on the BBC computer the printing of line numbers is not "windowed". This means that the effects being examined (moving graphics, say) are obliterated by the tool being used.

How much help should a teacher give?

Suppose a student has designed a program and is having problems. Giving help can be very time-consuming, and sensible limits need to be set for the teacher's sake. The teacher will be more disposed to help, and more able to help if the student has documented the program properly.

Encourage the following:

1. A list of the variables used. These should have meaningful names, like mass not M, and child$ not C$.
2. Procedures rather than GOSUBs, and usefully named.
3. REM statements where useful, sometimes simply followed by a line of asterisks to visually break the listing up into blocks.
4. A hard copy of the program if possible.

The usual criterion used before giving help to students applies in computing too. They must have made reasonable efforts to help themselves. A student who has not bothered to properly learn the meaning of the simple BASIC statements, should first be directed toward that task. The teacher in a computer room can be very easily overwhelmed by demanding students.

Student projects

A deeper understanding of BASIC programs is gained by writing them. I use two types of projects, single-concept and Multi-level.

Single-concept projects.

The student is required to research and develop a program around one of the "reserved words" of BBC BASIC. The words not dealt with in the BASIC course (like "ENVELOPE") can be included but should only be done when the others haave been satisfactorily handled. Each teacher will devise projects best suited to their class, but here are some examples.

1. Use the RND function to simulate two dice being thrown together. Calculate the sum of the numbers showing. Do this for 10000 rolls. How many times does a sum of seven occur? It should happen about 10000/6 times.

2. Use the SOUND statement and a FOR-NEXT loop to produce effects for the "Train" program. Make the sound muffled while the train is in tunnels.

3. Use a single-key input statement in a simple game for two players. Allot each player a key to use. Arrange for something to appear on the screen. The first player to react correctly wins a point. (A variation my students have designed is for a word to appear. Only if the word is a noun should the player press their key. If it is a verb as well, like "plan" or "track" or "dance" the key is not to be pressed.)

4. Look at the DISCO CUBE program. Draw a tetrahedron (triangular based pyramid) and make each visible side a different colour.

5. Research the use of "windows". Write a program which creates text and graphics window (they may overlap). Write text in both windows.

6. Make the letter 0 move around the screen, bouncing off all the edges.

Multi-level projects

Each project begins simply and becomes more demanding by stages. A student can stop at any level past the first, as interest and ability dictate. Some students produce very clever programs, of considerable sophistication.

1. Use the graphics screen to design the floor plan of a house to scale. The total length of the outside walls is 80 metres, and all passages are 1 metre wide.

   Find the shape which will give the maximum area for the 80m perimeter. Make the area of passageway as small as possible.

   Design furniture using special characters and put the pieces in place.

   Produce copies of your designs on the printer. (The teacher can provide a screen-dump routine).

   Show all the lines for electricity, water etc into the building.

2. Illustrate in the mode of your choice, the intersection of a pair of two-lane roads as seen from a point high above. Represent symbolically, traffic lights which alternate between red and green (or equivalent shades of grey) at regular intervals.

   Use a letter or special character to represent a car and animate it on your road system. It need not turn

corners, but it must obey the traffic lights.

Develop a two car system, without traffic lights. One car is to move north-south or south-north at random. The other moves at right angles. Include a give-way routine so that the cars obey the traffic rules.

Extend the system to multiple vehicles, without allowing collisions to occur.

Include the ability of the vehicles to turn at the intersection.

Design a program with the computer in charge of one car and a person in charge of the other. Make the program teach the traffic rules which apply at intersections. Use graphics to illustrate collisions, and use SOUND to provide the student's car with a horn. Use of the horn must affect the computer's car when appropriate.

Students sometimes propose projects of their own. This is fine, but check that it is not too ambitious, and is reasonably worthwhile. A project can also be used in a competition, with the best entry attracting an "official certificate" and a small prize eg a blank cassette.

SECTION III –

DESIGNING AN EDUCATIONAL PROGRAM

General Considerations

As a programming aid, the red keys can be defined and labelled as follows: LIST, RUN, *CAT, CHAIN, PRINTER, PROC, DEPROC, PAGE MODES, SAVE", PAGE MODE7.

Here is the program.

```
10 REM  RED KEYS DEFINED FOR TEACHER
   USE IN PROGRAMMING
20*KEY0 |V7LIST|M
30*KEY1 RUN|M
33*KEY2 |V7*.|M
37*KEY3 CHAIN"
40*KEY4 WIDTH70 |M VDU2,1,15,1,27,1,4
   8 |M LIST |M |C WIDTH0 |M
45*KEY5 PROC
50*KEY6 DEFPROC
60*KEY7 |V5|N LIST|M
65*KEY8 SAVE"
70*KEY9 |V7|N LIST|M
80 *FX138,0,78
90 *FX138,0,69
100 *FX138,0,87
110 *FX138,0,13
```

Explanation of the program.

The shifted back-slash key produces 11 in mode 7 and 1 in other modes. It stands for CONTROL. M is the code for the RETURN key.

Let's now go through each of the keys in turn.

KEY0 Produces MODE 7 (clearing the screen0, LIST and RETURN.

KEY1 RUN and RETURN

KEY2 Displays the disc catalogue in MODE 7.

KEY3 Prints CHAIN and the first quote mark on the screen, to be completed by the user.

KEY4 This causes a condensed listing 70 columns wide to be produced on the EPSON MX100 printer. (*FX6,0 may also be needed if the simple modification has not been made to arrange for carriage return.) This will need to be modified to suit other printers.

KEYS 5 and 6 print words on the screen to save typing time.

KEY7 As for key 9 but in MODE5 for formatting.

KEY8 prints SAVE" to be completed by the User.

KEY9 displays the listing on the screen piece by piece, with a slight overlap. It is most important that SHIFT must be used to move to the next part. Otherwise people will repeatedly press KEY 9. N is being used to set "page mode". The last four lines insert the letters N E and W and RETURN into the keyboard buffer so that the program will delete itself from memory.

A program like this should be on every disc used for programming. Then you type CHAIN U (for User keys) before doing anything else. If you are using cassettes instead, SAVE the program many times on a cassette and CHAIN it first every time you turn on the computer. The keys will stay defined even though the BREAK key is used. However, using CONTROL & BREAK removes them. Some teachers find other statements and commands more useful. For example, RENUMBER10, 10 LOAD" *DRIVE *WIPE*.

Also, it is possible to specially define more of the keys. *FX4,2 redefines the COPY and cursor keys. (See program 2 in the Little Library later in this section.) *FX219,12 redefines the TAB key to clear the screen. (Reset, with 219,9) *FX220,0 redefines the shifted @ as a less vulnerable ESCAPE key. Reset with 220,27.

Some find that the screen display on their TV always needs an initial downward adjustment. The U and R programs can begin with *TV255 to accomplish this. The versatility of the BBC computer is one of its most attractive features.

Abbreviation of keywords.

Although abbreviations for keywords have been avoided in the previous section, the programmer can save much time with them. Some are not worth the trouble, but most of the single letter ones are useful. I tell student programmers to remember 'UNCLE DIG':

U. for UNTIL
N. for NEXT
C. for COLOUR
L. for LIST
E. for ENDPROC (a big saving.)

D. for DATA
I. for INPUT
G. for GOTO

Here is an example of a good "first-screen" for an educational program.

****** SETTLEMENT ******
by Ann Newfield 1983
CHALKIESOFT

Imagine you are in charge of setting up a new community in an area which has never been settled.
I wonder what things you would want to take with you? (Axes, matches. . .?)
What sort of people would you require. (Meek and obedient? Brave and resourceful?. . .)
What assortment of skills would you want them to have? (Carpentry?. . Cooking?. .)
What features would you consider when siting your village? (OK permanent fresh water, but what about floods?. . .)
Will you and your party survive the first year?
Press any key to begin.

Note the five essential ingredients of a good first screen.

1. Title.
2. Author and year.
3. Company name or contact address.
4. Brief, readable description of what the program is about, or will do.
5. What to do next.

Note that "PRESS" or "TYPE" implies that the RETURN key is not needed. The program is probably using a version of GET or INKEY. If the word "ENTER" is used, eg "ENTER YOUR NAME", you will need to add "AND PRESS RETURN". Otherwise many people will sit and stare at the screen wondering why nothing is happening.

Remember that some of the people you are writing for may never have used a keyboard before.

For a program with many instructions, it's helpful to offer the choice of whether to see them or not. This allows a person who knows what is required, to skip ahead. The computer could deal with the response as shown in the following program.

```
 10 MODE7
500 PRINT"Do you want instructions (Y
    or N).":*FX21,0
510 REPEAT
520     key$=GET$
530 UNTIL key$="Y" OR key$="y" OR key$
    ="N" OR key$="n"
540 IF key$="Y" OR key$="y" THEN 600
    ELSE 1000
600 PRINT'"Instructions here."
    1000 PRINT'"Program here."
```

Note that allowance is made for the CAPS LOCK key to be on or off. An alternative is to use *FX202,32 to turn CAPS LOCK on from within the program. Then only upper case INPUT need be checked. For example, turn CAPS LOCK off and then RUN this program.

```
10 REPEAT
15 *FX202,32
20 key$=GET$
30 UNTIL key$="A"
40 PRINT"O.K"
```

Or, lower case can be converted as follows.

```
10 REM Change to UPPER CASE
20 PRINT'"PRESS Y OR N"
30 *FX21,0
40 key=GET AND &DF
50 IF key=78 PRINT"You pressed N or n."
60 IF key=89 PRINT"You pressed Y or y
   ."ELSE 20
70 GOTO20
```

Menu design

You can lead the user into your program comfortably by offering a set of explicit choices at every stage. Here is an example:

PRESS THE NUMBER OF YOUR CHOICE
1. FIRST LEVEL
2. CHALLENGING
3. EXPERT
4. SHOW ME THE INSTRUCTIONS AGAIN

Let's look at one way a program could deal with the response to a set of choices.

```
100 CLS
110 PRINT'"WILL YOU TYPE  1,2,3 OR 4
    PLEASE?"
120 REPEAT
130   ascii = GET
140 UNTIL (ascii<53 AND ascii>48)
150 PRINT"TAR!"
```

Here ascii (lower case) is the numeric variable used. The number 1 has an ASCII code of 49, through to 52 for the number 4. In practice, line 200 might be something like: ON

ascii GOTO 200,300,400,500
If an alphabetical key is required, GET$ could be used as shown.

```
100 CLS
110 PRINT''"Will you type Y or y for me?"
120 REPEAT
130    key$ = GET$
140 UNTIL ( key$ = "Y" OR key$ = "y")

150 PRINT"Thanks for typing Y or y"
```

Alternatively, the AND & DF method is good too.

These methods of dealing with Menus, discard irrelevant input efficiently. But they do not protect against the User pressing ESCAPE or BREAK inadvertently. Hence it may be desirable to disable the ESCAPE key using *FX229,1 early in the program, and re-enable it with *FX229,0 as part of a "quit the program" procedure. It is possible to use *FX200 instead of *FX229. In passing I would mention *FX200,2. This disables the OLD statement, producing "Bad program". Some students are fascinated by this because with *FX229,1 and *FX200,2 they can run a "secret" program which nobody can stop and list.

Suppressing the flashing cursor

On some occasions the sight of the cursor flashing on the screen during a program can be too distracting. In OS>0.1 it is erased by VDU23;8202;0;0;0; and restored by a MODE statement. However this clears the screen, which could be

disastrous. In version 1, VDU23,1,0;0;0;0; erases the cursor and VDU23,1,1,;0;0;0; replaces it. The commas and semicolons must be exactly right, which can cause problems for students. Warn students using the 1982 User Guide that the final semicolon was accidentally omitted in the 8202 version, on page 77. Because of the awkwardness of the process, a cursor-off-on procedure could be defined whenever a program needs a lot of cursor switching. Then it only has to be correctly typed once. See the program called "SPIRAL" for another method of cursor suppression or use this: !&FE00=&10200A and !&FE00=&10670A to turn it on again.

It is sometimes feasible to use VDU5 and print that way. The flashing text cursor is automatically suppressed.

A HELP! key.

A HELP key is often valuable to the user of a complicated program. A further key is used to recommence from the point of interruption.

The HELP routine can be installed within the main working loop (or loops) of the program. The GET$ statement is not suitable because the program will only stop if required. However, *FX21,0 may be needed to regularly flush the keyboard buffer. The best alternative is "negative INKEY", since no keyboard flushing is required. The user simply holds the key down until help arrives. Negative INKEY also has the advantage that even keys without ASCII values can be detected.

Here is a sample HELP key routine

A message needs to appear on the MENU screen like:
Hold down the CTRL key if you need help.
In a suitable place (or places), put the line:
IF INKEY(-2) THEN PROChelp
Here is a possible form for the HELP PROCEDURE.

```
2000  DEF PROChelp

2010

  details....

2100  PRINT"Press the SPACE bar to
      continue.":  *FX21,0
2110  cont$=GET$ :  IF cont$ <> " "
      THEN 2110
2120  CLS : cont$= "" : REM resets the
      cont$ flag to null.
2130 ENDPROC
```

A problem arises when a screen display (eg graphics) would be irreparably spoilt by the help messages. In these circumstances, a text window may be created to hold any messages. Alternatively, the advice could be sent to the printer if one is available, leaving the screen display intact, by means of VDU1.

Using the Red Keys.

The use of *KEY is illustrated by the R and U programs discussed earlier. The Red keys can also be useful in that they can be detected by the negative INKEY statement. The Red keys are invaluable, when practically all of the others are in use, eg in a word-processing program.
Except in operating system 0.1 the copy and cursor keys can also be redefined by *FX4,2 and restored to normal with *FX4,0 (See program 2 in the Little Library).

Using graphics and text "windows".

As mentioned in Section II, VDU24 sets the parameters defining a graphics window. VDU28 is used to create a text window. VDU26 brings both back to occupy the whole screen. For examples of windowing, see the "MARKS" and "Sheep Station" programs at the end of this section.

COLOUR AND SOUND

Colour is wonderful as an attention-getter, unless overused. Also one should bear in mind that not all schools use colour video equipment. If you write a program for general use, it is best to stick to colours which give reasonable visibility and contrast in black-and-white or green and white too. Generally, green and blue give faint effects on monochrome equipment. The following program shows the colours available in MODE 7 for test purposes.

```
10 MODE7:code=129
20 !&FE00=&10200A
30 REPEAT
40 PRINTTAB(0,2)"WHITE"
50 READcolour$
60 FOR K = 1 TO 100
70    PRINT CHR$(code)colour$;
80 NEXT
90 wait=INKEY(200)
100 code=code+1:IF code=135 code=129:
    RESTORE
110 UNTILO
120 DATARED      ,GREEN   ,YELLOW ,BLUE
    ,MAGENTA,CYAN     ,LAST
```

The main disadvantages of MODE7 are that you cannot define your own characters and you cannot use MOVE or DRAW or graphics windows. It is possible to define rather crude shapes (see page 154 of the 1982 User Guide) but fine graphic work is impossible in MODE7.

Sound is good for emphasis, although the tiny speaker fitted to the BBC does not allows any dramatic effects. The envelope commands are so time-consuming to master that most teacher programmers will probably rely on the simple 4-parameter SOUND statement. However, notes can be easily sychronised to form chords, as the following program illustrates. It provides control over the pitch and loudness of channels 1 2 and 3 independently, by means of "negative-INKEY". Moreover, these values are displayed on the screen and the sound can be turned on or off without disturbing them. When an interesting chord is found, the values can be jotted down for future reference.
Pressing S, D or F raises the pitch of channels 1, 2 and 3. In conjunction with A, these same keys lower the pitch.
Similarly, J, K and L can be used in conjunction with H to control the loudness.

```
10 REM ORGAN5
20 MODE7
30 !&FE00=&10200A:REM erase cursor
40 REM initialise
50 P1=101:P2=129:P3=153:A1=-10:A2=-10
   :A3=-10:incP=0:incA=0
60 PROCdisplay
70 REPEAT
80 key$=INKEY$(0):IF key$="" THEN 80
90 *FX21,0
100 IF INKEY(-66) incP=-1 ELSE incP=1
110 IF INKEY(-82) P1=P1+incP
120 IF INKEY(-51) P2=P2+incP
130 IF INKEY(-68) P3=P3+incP
140 PROCcheckP
150 IF INKEY(-85) incA=1 ELSE incA=-1:
 REM Decreasing A makes the note louder.
160 IF INKEY(-70) A1=A1+incA
170 IF INKEY(-71)  A2=A2+incA
180 IF INKEY(-87) A3=A3+incA
190 PROCcheckA
200 PROCchord(A1,A2,A3,P1,P2,P3)
210 PROCupdate
220 IF INKEY(-17) THEN *FX210,1
230 IF INKEY(-34) THEN *FX210,0
240 UNTILO
250 DEFPROCchord(A1,A2,A3,P1,P2,P3)
260 SOUND&0211,A1,P1,1000
270 SOUND&0212,A2,P2,1000
280 SOUND&0213,A3,P3,1000
290 ENDPROC
300 DEFPROCcheckP
310 IF P1<1 P1=1
320 IF P2<1 P2=1
330 IF P3<1 P3=1
```

```
340 IF P1>255 P1=255
350 IF P2>255 P2=255
360 IF P3>255 P3=255
370 ENDPROC
380 DEFPROCcheckA
390 IF A1<-15 A1=-15
400 IF A2<-15 A2=-15
410 IF A3<-15 A3=-15
420 IF A1>0 A1=0
430 IF A2>0 A2=0
440 IF A3>0 A3=0
450 ENDPROC
460 DEFPROCdisplay
470 CLS
480 PRINT'"CHANNEL ONE"
490 PRINT'TAB(10)"Pitch     ="
500 PRINT'TAB(10)"Amplitude= "
510 PRINT''"CHANNEL TWO"
520 PRINT'TAB(10)"Pitch     ="
530 PRINT'TAB(10)"Amplitude= "
540 PRINT''"CHANNEL THREE"
550 PRINT'TAB(10)"Pitch     ="
560 PRINT'TAB(10)"Amplitude= "
570 PRINT'"    Q:Sound OFF    W:Sound ON"
580 PRINT"A     S     D     F    *  H     J
  K     L"
590 PRINT"down      pitch          down
loudness"
600 ENDPROC
610 DEFPROCupdate
620 PRINTTAB(20,3)P1;"    "
630 PRINTTAB(20,5)A1;"    "
640 PRINTTAB(20,10)P2;"    "
650 PRINTTAB(20,12)A2;"    "
660 PRINTTAB(20,17)P3;"    "
```

```
670 PRINTTAB(20,19)A3;"     "
680 ENDPROC
```

While on the subject of sound, here is a discovery by Ian Nicholls about the production of lower notes than those discussed in the manual.

```
5 REM LOW NOTES Ian Nicholls

10 SOUND2,-15,1,100
20 FOR delay=1 TO 5000:NEXT
30 P=188
40 SOUND0,-15,3,180
50 SOUND1,0,P,10
60 P=P-2
70 FOR delay=1 TO 100:NEXT
80 GOTO40
```

A little library of special effects.

These can add interest and highlight the points being made. They should not be allowed to obscure the teaching message however.

1. Use of GCOL3 in animation.

This program illustrates the use of GCOL3 to make one figure appear to pass behind another. An animated train enters and emerges from tunnels.

```
10 REM TRAIN
20 REM Define special characters:train
   1, train2 and tunnel.
30 VDU23,240,0,2,0,226,106,255,255,115
40 VDU23,241,2,0,0,226,106,255,255,230
50 VDU23,242,255,255,255,255,255,255,
      255,255
60 MODE5:X=0:I=0
70 REM Change palette: 0=cyan 1=red
   2&3=white
80 VDU19,0,6;0;19,1,1;0;19,2,7;0;19,3
   ,7;0;
90 REM Draw tunnels in white and print
   at graphics cursor. Then draw tracks.
100 GCOL0,2:VDU5
110 MOVE300,500:VDU242,242,242,32,32,32,
    242,242:MOVE0,470:PLOT5,1279,470:
    MOVE0,500
120 REM Set up EOR.
130 GCOL3,1
140 MOVEX,500
150 REM Print train
160 VDU240+I,8
170 FOR pause=1 TO 40:NEXT
180 REM Erase train by backspacing and
    EOR printing again.
190 VDU240+I
200 REM Move on a bit.
210 X=X+8
```

```
220 REM Swap to other image of train.
230 IF I=1 I=0 ELSE I=1
240 IF X>1200 X=0
250 GOTO140
```

2. Using the cursor keys to move a figure around the screen

This program illustrates the redefinition of the cursor and COPY keys.

```
10 MODE7
20PRINT'"Control the X with the cursor
   keys. Make it vanish with the copy
   key. Any other key ends the program."
30PRINT''"Press any key to continue."
40 *FX21,0
50 key$=GET$:CLS:REM erase cursor
60 !&FE00=&10200A
70 P=20:Q=12:A=20:B=12
80 PROCPLACE
90 *FX4,1
100 *FX15,1
110 *FX229,1
120 X = GET
130 *FX229,0
140 ONERROR GOTO230
150 N = 140-X
160 IF N<1 OR N>5 170
170 ON N GOTO 220,210,200,190,180
180 CLS:GOTO100
```

```
190 P=P-1:PROCPLACE:GOTO100
200 P=P+1:PROCPLACE:GOTO100
210 Q=Q+1:PROCPLACE:GOTO100
220 Q=Q-1:PROCPLACE:GOTO100
230 PRINT "A CURSOR KEY OR THE COPY KEY
    WAS NOT PRESSED: END OF DEMO"
240 *FX4,0
250 FOR delay=1 TO 5000:NEXT
260 MODE7:END
270 DEF PROCPLACE
280 IFP<1 P=1
290 IFP>39 P=39
300 IFQ>22 Q=22
310 IFQ<1 Q=1
320 PRINT TAB(A,B)" "
330 PRINT TAB(P,Q)"X"
340 A=P:B=Q
350 ENDPROC
```

3. Three dimensional display

This program shows the use of GCOL1 to produce a three dimensional "shadow" effect. It could be useful for titles, for example.

```
10 REM 3-D LETTERING
20 MODE5
30 VDU19,0,3,0,0,0,19,1,0,0,0,0
40 PRINTTAB(0,5)"ENTER A MESSAGE"
50 INPUTstring$
60 CLS
70 VDU5: MOVE58,504
80 PRINTstring$
90 GCOL1,1
```

```
100 MOVE50,500
110 PRINTstring$
```

4. Sideways scrolling

This program scrolls the screen left and right. This could be used to introduce text in a novel way. There are many other possible applications.

```
10 REM SCROLL TO AND FRO
20 FOR A = 1 TO 100
30 PROCscroll
40 NEXT
50 FOR A = 100 TO 1 STEP -1
60 PROCscroll
70 NEXT
80 END
90 DEF PROCscroll
100 TIME = 0 : REPEAT UNTIL TIME = 20
110 PRINTTAB(5,20)A
120 VDU23;13,A;0;0;0;
130 ENDPROC
```

5. All the spectral colours in a four-colour mode

This program and the next will interest teachers of Art and Science.

```
 5 REM SPECT5
10 VDU23,240,170,85,170,85,170,85,170,85
```

```
20 VDU23,241,85,170,85,170,85,170,85,170
30 VDU23,242,255,255,255,255,255,255,
   255,255
40 MODE5
50 VDU19,0,4;0;19,3,2;0;16
60 VDU5
70 MOVE70,990:PRINT"Mode 5 is a four
colour mode but      here you see all
the colours of       the spectrum."
80 PROCred
90 PROCorange
100 PROCyellow
110 PROCgreen
120 PROCviolet
130 END
140 DEFPROCred
150 MOVE0,780
160 FOR X=1 TO 80
170 VDU18;1,242
180 NEXTX
190 ENDPROC
200 DEFPROCorange
210 MOVE0,650
220 FOR X=1 TO 80
230 VDU18;2,240,8,18;1,241,8
240 VDU18;2,241,8,18;1,240
250 NEXTX
260 ENDPROC
270 DEFPROCyellow
280 MOVE0,520
290 FOR X=1 TO 80
300 VDU18;2,242
310 NEXTX
320 ENDPROC
330 DEFPROCgreen
```

```
340 MOVE0,390
350 FOR X=1 TO 80
360 VDU18;3,242
370 NEXTX
380 ENDPROC
390 DEFPROCviolet
400 MOVE0,130
410 FOR X=1 TO 80
420 VDU18;0,240,8,18;1,241,8
430 VDU18;0,241,8,18;1,240
440 NEXTX
450 ENDPROC
```

6. A multi-coloured quilt based on the mixing of four colours

This MODE5 program also illustrates colour mixing, and the changing of the set of four colours from which the mixing is being done. Some very striking patterns result.

```
10 REM QUILT
20 VDU23,240,170,85,170,85,170,85,170
   ,85
30 VDU23,241,85,170,85,170,85,170,85,170
40 VDU23,242,255,255,255,255,255,255,
   255,255
50 MODE5
60 VDU19,2,4;0;19,0,3;0;19,3,2;0;16
70 VDU5
80 MOVE10,1023
90 PRINT"MODE 5 has 4 colours"
```

```
100 MOVE0,970
110 FOR square=1 TO 520
120 PROCcolour
130 NEXT
140 MOVE130,120:VDU18;2:PRINT"NOW let's
    try  different   palettes."
150 VDU19,2,RND(8)-1;0;19,0,RND(8)-1;0
    ;19,3,RND(8)-1;0;19,1,RND(8)-1;0;
160 delay= INKEY(500)
170 GOTO 150
180 END
190 DEFPROCcolour
200 C=RND(4)-1:D=RND(4)-1
210 VDU18;C,240,8,18;D,241,8
220 VDU18;C,241,8,18;D,240
230 ENDPROC
```

7. Changing the graphics origin

This program draws an outward curling spiral. It is referred to under "hanging programs" in Section II.

```
 5 REM SPIRAL
10 angle=0:arm=10
20 MODE4
30 REM erase cursor
40 ?&FE00=10:?&FE01=32
50 REM change graphics origin
60 VDU29,500;500;
70 REPEAT
80   PLOT69,arm*COSangle,arm*SINangle
90   arm=arm+1.6:angle=angle+.1
```

```
100 REM program "hangs" if condition is:
    angle=10*PI
110 UNTILangle>10*PI
120 MOVE510,-5
130 DRAW10,0
```

Class demonstration programs

Scene:

A teacher has a class-group of students and is using the computer to demonstrate a concept (eg in Geography). There is a verbal introduction, and perhaps a commentary while the program runs. However, the teacher is present as a subject specialist, not a computing specialist. This is similar to a teacher showing a film-making. The teacher discusses the subject of the program and answers questions, asks questions, provides homework and follows up where appropriate. This section deals with the approach and skills needed when designing programs of this type.

When a computer program might not be very suitable for group-dems.

When a program involves a lot of thoughtful interaction, many of the less interested or less capable group members become bored. Some of the others who are interested too often choose a passive role in the group. There may be other sociological reasons why the teacher may decide against showing the program to a group rather than an individual or pair basis. It is a matter of matching the program to the class.

The group-dem situation is not suitable for remedial work, where the learner must be able to proceed at a suitable pace, return to an earlier level or seek help without embarrassment or pressure.

During some programs, values need to be chosen for certain parameters. Much of the interest generated by such a program consequences. The learning experience is often of this trial-and-error nature. For example, in a program on Emergency First Aid, the student may choose certain actions in a simulated situation then see what happens to the "patient". It may be a biological or perhaps an historical simulation where there are many variables to manipulate which affect the outcome. Will the plant thrive? Will Napoleon now win?

However, group interaction can make a learning experience much more valuable. The sharing of insights, ideas and opinions can be very worthwhile. Demonstration to a group can certainly save time, and requires a minimum of equipment. Some schools use monochrome VDU's in the computer room. When a program comes along which makes important use of colour, a colour TV will suffice for a group showing.

VISUAL REQUIREMENTS

There should be an absolute minimum of text in a group-dem program. Tests of attention span within groups show that concentration can quickly diminish after the first twenty or so words, and less if unfamiliar terms are used. A recovery cycle usually occurs if a diagram or other non-text break can be arranged. When text is used, it must be large enough for all to read. MODE5 on the BBC provides a screen only 20 columns wide, with 160 x 256 graphics spots. Even on

a small screen, the letters are large enough for people at the back of a group to read. The medium resolution produces dots small enough to be useful but large enough to be visible from a distance. In the next program the text and effects can be seen by all members of a group. Try altering the MODE to MODE4.

```
10 MODE5
20 PRINT''" PROJECTILE  MOTION"
30 PRINT;"--------------------"
40 PRINT''"A projectile is let fall from
   rest. Thenothers are fired sideways."
50 PRINT''"The sideways speed  of each
   projectile  is constant as it
   crosses the screen."
60 PRINT''"Horizontal lines aredrawn
   to show that  the vertical motion is
   the same nomatterwhat the sideways
   speed is."
70 PRINT'''"Press a key to begin":*FX
   21,0
80 key$=GET$:CLS
90 VDU 23;8202;0;0;0;:REM erases cursor
100 elapsedtime=0:horizspeed=0
110 PRINTTAB(0,30)"Horizontal speed=0"
120 REPEAT
130 delay=INKEY(20-horizspeed):PROCplot
    point(elapsedtime)
140 elapsedtime=elapsedtime+7:IF horiz
    pos>1278 OR drop>880 THEN horizspeed
    =horizspeed+1:elapsedtime=0:PRINT
    TAB(0,30)"Horizontal speed=";
    horizspeed
150 UNTIL horizspeed=99
```

```
160 END
170 DEF PROCplotpoint(T)
180 horizpos=horizspeed*elapsedtime:drop
    =.05*elapsedtime*elapsedtime
190 PLOT 69,horizpos,1023-drop
200 MOVE 0,1023-drop
210 DRAW horizpos-15,1023-drop
220 ENDPROC
```

Another mode which has possibilities for group-dem work is MODE7 using double-height characters. The next short program shows this effect, coupled with flashing characters.

```
10 REM DOUBLE HEIGHT & FLASHING
20 !&FE00=&10200A:CLS
30 FOR I = 0 TO 1: PRINTTAB(2,10+I)CHR$
   136CHR$133CHR$141" HELLO POOR HUMAN!
   ": NEXT
40 FOR I = 0 TO 1: PRINTTAB(2,14+I)CHR$
   136CHR$131CHR$141" DON'T YOU WISH":
   NEXT
50 FOR I = 0 TO 1: PRINTTAB(2,18+I)CHR$
   136CHR$130CHR$141" YOU WERE AS SMART
   ": NEXT
60 FOR I = 0 TO 1: PRINTTAB(2,22+I)CHR$
   136CHR$135CHR$141" AS MEE ???":NEXT
```

Note that two identical PRINT lines are used and code 141 is invoked. VDU140 turns off the effect. The code to initiate flashing (136), only effects the rest of that line, use VDU137. However, control characters occupy one screen space, and we have to overcome this. The main point about these MODE

7 control codes is that they never carry on from one screen line to the next. For more details see the 1982 Use Guide, pages 152, 153.

MODE7 has useful features for group work. For example, flashing graphics, up to 7 colours, and 64 different graphics shapes. Also only 1K of memory is used, which can be very important when writing a long program that is to run on a Model A. The double height characters can look rather skinny, but if you want these other advantages, you may prefer MODE7 to MODE5.

Programs appear in magazines from time to time which produce enlarged characters for any mode.

SOUND

The sound output from the internal speaker is not usually loud enough for group demonstrations. If a proper interface to an amplified sound system is not available, a small microphone can be taped over the BBC micro speaker. The signal from this can be amplified using a record player or tape recorder. The results are usually reasonable, and manual control over loudness becomes available.

DOCUMENTATION

What kind of documentation is needed for group-dem material? Our fundamental assumption is that the teacher using the program has no knowledge of computers or programming. The documentation should be in three parts.

Installing and running the program

Explain how to turn on the equipment beginning with the wall switch. Then describe how to insert the cassette or disc. Now which keys and buttons to press, not forgetting the ubiquitous "and press RETURN" which is second nature to us. If the instruction to CHAIN "name of program" is used, it should be explained, including holding down the SHIFT key while pressing the 2 key to get the quote marks. Once again add, PRESS RETURN. Instruction may be needed on how to quit the program, especially if the ESCAPE and BREAK keys have been subverted. Ask also that the keys be tapped gently. Teachers with a background in manual typewriters can unwittingly set a poor example to students.

Some of this advice appears to insult the intelligence of the user. But bear in mind that many people find computers intimidating. Let's be clear without being condescending.

How to use the program

Explain without any computer jargon the features and method of using the program. For example, if a joystick or lightpen is involved, its use must be described. Here we are talking about the actual mechanics of getting the most out of the program – what it can do and what it can't.

Educational Goals.

These goals need to be clearly stated in such a way that their achievement can be tested. Too often a programmer becomes engrossed in mastering a difficult technical objective and ends up with a very cunning and perhaps innovative piece of programming which has practically no educational value. A writer can spend hundreds of hours producing a listing which from a programming point of view is tight, compact and logical, only to find a teacher being derisive or even contemptuous of it (and rightly) from the education viewpoint. This applies particularly to what has become known as "bells and whistles". An effect which is cute when seen for the first time becomes very wearing when repeated often. The sounds and colours and flashes must serve the educational ends, not get in their way.

If possible within available memory, it is a good idea to include in the initial program menu, an option to produce a hard-copy of the instructions. Sometimes printed material supplied with a program is lost. Other people are deterred by a large slab of text on a video screen. They feel more at home with a paper copy which they can peruse without the insistency and urgency which they attach to the bright screen. There is a difficulty here in that printers vary in the codes needed. The program may need to be modified by a programmer in the school, to adapt it to the needs of the resident printer, in conjunction with the printer manual. Every printer presents some problems. For example, with the EPSON printer, a "form-feed" occurs if it comes across a CLS statement (VDU12) in the wrong place.

Writing programs to be used by individuals or small groups.

Scene:

A teacher has a class-group sitting singly or in small groups at BBC computers into which programs (not necessarily all the same) have been loaded. The students may have been given printed documentation or other preparation. They will be working on the program at their own pace and should need minimal advice on the actual running of the program. The teacher's main input will be to discuss teaching points arising from the program. The teacher does not necessarily know anything about computing. This section deals with the approach and skills required when designing programs of this type.

Identify a need.

There are many programs which drill maths tables, and plenty of versions of "Hangman". Some programming ideas have become cliches. Let's try to develop a creative attitude amongst educational programmers, unlike the notoriously derivative approach in the "games" field. Another less desirable type of educational program is the "shoot-em-down" type, where aggression and violence are used as motivators. Calling the targets "aliens" or "droids" is a feeble ploy. Surely we don't aim to teach this six-gun approach as a mature way of dealing with problems.

As teachers we need to look at our student community. What are their needs and aspirations? What problems do they have which a computer program could best solve?

A computer is a marvellously adaptable teaching aid. We are able to cater for students differing abilities and interest levels. If forced to form small groups rather than teach individuals, make the groups as homogeneous as possible. Every member of the class should be able to work at their own pace, and find this both stimulating and rewarding.

The time factor.

At the outset we must consider how long our "typical student" will be able to, or wish to spend with the program. This will very strongly affect our aims. Commonly a remedial type of program will be designed to run for about half a lesson. This can then be followed by something quite different. At the other end of the scale, a program may be so engrossing that several lessons may be required to exhaust its possibilities. A program of this type may need a mechanism which saves the variables so that it can be resumed at the same point in a later lesson. This facility is bound to become very common. In this book there are two examples of programs which can be used with individuals. One is a remedial type program called "Directed Numbers" and the other is an open-ended holistic simulation called "Sheep Station".

Whatever the length of the program we are designing, there must be a few clearly defined aims. These need to be expressed in ways which allow their achievement to be judged reasonably objectively.

Reward and correction.

When writing an interactive program for an individual student there may be moments when encouragement or correction is called for. This happens particularly with remedial programs. Here we must distinguish between recognition of a good standard, and praise. Most commonly, what happens in remedial computer programs is hypocrisy, and recognised as such by students. The program usually asks for the student's name, stores it in a string (say names). This string (bona fide or note) is then used in such statements as "That is fantastic"; names; "you got"; score; "out of "; maximum;"!"

Of course the program has no way of knowing if the score (say) 8 out of 10 is really "good" for that student or not. For one student 8/10 is a poor effort. For another it may represent a magnificent achievement. (Or perhaps, with some programs, magnificent luck.) The point is that a program, being 100% objective, should only print out objective remarks. Indeed, this is one of the great strengths of a computer as a "teacher". It does not prejudge the student nor react subconsciously as people do. This does not mean that an encouraging remark is not possible. For example, if successive tries by "name$" reveals an improving score, the program can point out something like:
In the last five tries, your score has improved from53% to 78%, Jo.

Corrections should be objective and constructive. "Well you did it again, Dummy!!!" and "Only a real TURKEY does that !!!" are typical of the negative superior remarks which have no place in education. We have already seen the tragic result when a depressed boy blundered during a game of "Dungeons

and Dragons" program and could not cope with the consequences to his "Game character". The first line of our brief as teachers, has to be "Don't do harm."

Most writers on the subject of correction, rightly point out that a learning failure should not be "rewarded" by a stunning sound and/or graphics display. One mathematics program I saw, portrayed the student in a small boat. If the student scored less than 50%, a colourful battleship labelled "GODFATHER" sailed onto the screen and firing a series of noisy salvoes, sank it. Whenever this was happening, students came from all over the room to see it. The lad was initially highly embarrassed, but was soon "failing" deliberately to gain attention. By contrast, a successful student reached the far shore and was rewarded by a silent line of print:

Well done Veronica, you have been promoted!

Remedial and Extension programs.

Due to the size and non-homogeneous nature of most classes, teachers are usually forced to teach to the middle-ground of ability. The weakest students often do not grasp all the ideas, or only partially understand them. these people are not necessarily weak intellectually. For some, the traditional classroom is not a suitable learning environment. Others are simply not "ready" to learn a particular concept or skill. For others there are psychological barriers preventing progress. Another group may have missed some schooling through illness or a crisis in the family. There are many other possible reasons for offering remedial work. This includes, it must be said, the ineptitude of some teachers.

When writing a remedial program, decide whether you are writing for an intellectually slow student, or one who fits into another category. It is easy to insult the intelligence of the person using the "remedial" program. The student may be brighter than you are, but need help due to absence from lessons because (say) the family was relocated in another country. We need to widen our concept of "remedial".

It is usually essential in a remedial program, to give control of the rate of progress to the student. We should be seeing instructions like
"Press C to continue."
"Do you want to try more examples of the type, or to move on?"
"Do you want to skip the next section?"

In a remedial program, the privacy and dignity of the student needs to be preserved. No "bell or whistle" should announce to others in the room that the student is not coping too well. We should make the usage of the program straight-forward, and not an intelligence or perserverance test in itself. Otherwise many slower students will "turn off".

The remedial program should afford the learner a feeling of accomplishment, of progress toward an easily recognised goal. Without that feeling of success there is little enjoyment, and motivation drops away.

At the other end of the success scale are those students equally ill-served by the "middle-ground" approach. These people are often bored by their lessons, particularly if they have to sit through repetitious explanations until the whole class-group has "caught-on." Often these students are a behaviour problem. They are seeking a challenge out of sheer desperation In any case, there is a great waste of

human potential. The "individual" computer learning program should be yet another resource to which the gifted student can turn. Earlier we discussed giving control of the pace of a remedial program to the learner. To extend the gifted student it may be apropriate for the program to set the pace. Pace can be a challenge, as seen in some of the world's most popular arcade games. However the use of pace and acceleration should be developing acuity and not simply agility.

User testing and the handling of suggestions.

When you have decided to stop refining or enhancing your program, and you think it is free of bugs, it should be "user tested". This is the best way to find out if those input routines you wrote are as robust as you think. You will also find out just how "helpful" your instructions really are.

When choosing your "guinea pig" user, pick someone without computer experience. Your computing colleagues make poor guinea pigs because they will tend to unconsciously draw on their computing background to make your program work, over-riding any flaws. Also, the user should be a typical "target student". Otherwise they may not appreciate the points you are trying to make. You will not be able to see if the program is educationally effective.

When you have given the user the program, watch from a distance. Do not offer any advice which you will not be giving routinely every time the program is run. It is one thing to manage a single user, but soon you may have a class full of them. So ideally the documentation or instructions should enable the program to be used without any intervention at all.

A common fault with interactive programs is the use of a mixture of INPUT statements and single-key inputs like GET. Often the user will not know whether to press RETURN or not. If you do need to help, make careful note of the areas which may need clarifying or correcting. If the program crashes, note the line number and perhaps print out the variables.

The most common cause of a crash during user testing is a leaky input routine. When programming, one knows the type of values necessary. The user may enter a number outside the permitted range, or type a letter instead. A novice at the keyboard is also more likely to accidentally drop into lower case or touch ESCAPE instead of 1 or BREAK instead of f9.
Remedies for these were discussed earlier. Ask for criticisms and suggestions in writing if possible, but don't expect an essay. You should be looking for frank but explicit comments. "The program was stupid." (or "terrific") is not much help. The most common request is for more versatility, more user choices.

You need to be prepared to find that the program should be scrapped. Keep your ego out of it, despite the many hours you may have spent. On the other hand, it is wonderful to write a program which students are excited about, and grateful for. That will soon be happening far more often than not.

```
10 REM Don Thorpe 1983
20MODE5
30VDU23,240,126,0,0,0,0,0,0,0
40VDU23,241,0,16,0,124,0,16,0,0
50PROCname
60 DEF PROCname
70 CLS:PRINTTAB(2,5);"DIRECTED NUMBERS"
80PRINT''''"PLEASE ENTER YOUR    NAME
   IF YOU WANT TO,";
90PRINT"AND PRESS RETURN,    OR JUST
   PRESS RETURN"
100N$=""
110INPUTN$
120N$=N$+"!":CLS
130PRINTTAB(0,5);"HELLO ";N$
140 PRINT TAB(2,10)"PRESS A S M OR D
TO CHOOSE FROM :"
150PRINT''"ADDITION"
160PRINT'"SUBTRACTION"
170PRINT'"MULTIPLICATION"
180PRINT'"DIVISION"
190B$=INKEY$(0):IFB$=""THENR=RND(1):GOTO
   190
200 IFB$="A"ORB$="a"THENPROCaddition
210 IF B$="S" OR B$="s" THENPROC
    subtraction
220 IFB$="M"ORB$="m"THENPROC
    multiplication
230 IF B$="D"ORB$="d"THENPROCdivision
    ELSE 140
240ENDPROC
250 REM******************
260DEF PROCaddition
270R=0:N=0
280 X = 10-RND(20) : IFX=0 THEN 280
```

```
290 Z = 30-RND(60)
300  IF RND(1) <.5 THEN A=X : B=Z : GO
     TO 320
310 B=X : A=Z
320 U$=STR$(ABS(A)) : V$=STR$(ABS(B))
330IFA<0THENA$=CHR$(240)+U$:GOTO350

340A$=STR$(A)
350IFB<0THENB$=CHR$(240)+V$:GOTO370
360B$=STR$(B)
370Q$=A$+" + "+B$
380 PROCenter
390 Z=A+B:PROCcomment
400GOTO280
410ENDPROC
420 REM************
430DEF PROCsubtraction
440R=0:N=0
450 X = 10-RND(20) : IFX=0 THEN 450
460 Z = 30-RND(60)
470  IF RND(1) <.5 THEN A=X : B=Z : GO
     TO 490
480 B=X : A=Z
490 U$=STR$(ABS(A)) : V$=STR$(ABS(B))
500IFA<0THENA$=CHR$(240)+U$:GOTO520
510A$=STR$(A)
520IFB<0THENB$=CHR$(240)+V$:GOTO540
530B$=STR$(B)
540Q$=A$+" - "+B$
550 PROCenter
560 Z=A-B:PROCcomment
570GOTO450
580ENDPROC
590 REM***************
600DEF PROCmultiplication
610R=0:N=0
```

```
620 X = 10-RND(20)
630 Y = 10-RND(20)
640  IF RND(1) <.5 THEN A=X : B=Y : GO
     TO 660
650 B=X : A=Y
660 U$=STR$(ABS(A)) : V$=STR$(ABS(B))

670IFA<0THENA$=CHR$(240)+U$:GOTO690
680A$=STR$(A)
690IFB<0THENB$=CHR$(240)+V$:GOTO710
700B$=STR$(B)
710Q$=A$+" X "+B$
720 PROCenter
730 Z=A*B:PROCcomment
740GOTO620
750ENDPROC
760 REM***************
770DEF PROCdivision
780R=0:N=0
790 A = 100-RND(200)
800 B =10-RND(20):IF B=0 THEN 800
810 IF A MOD B <> 0 THEN A= A +1:GOTO810
820 U$=STR$(ABS(A)) : V$=STR$(ABS(B))
830IFA<0THENA$=CHR$(240)+U$:GOTO850
840A$=STR$(A)
850IFB<0THENB$=CHR$(240)+V$:GOTO870
860B$=STR$(B)
870Q$=A$+" "+CHR$(241)+" "+B$
880 PROCenter
890 Z=A/B:PROCcomment
900GOTO790
910ENDPROC
920 DEF PROCenter
930N=N+1:IFN=11THENPROCmorestop
940CLS:PRINTTAB(4,5)"NUMBER ";N
950PRINT''Q$,"="
```

```
960PRINT''"ENTER THE ANSWER ANDPRESS
   RETURN"'
970INPUTC$
980IFC$=""THEN940
990C=VAL(C$):C$=""
1000 ENDPROC
1010 DEF PROCcomment
1020 IF C <> Z THEN 1060
1030PRINT
1040SOUND1,-15,23,6:SOUND1,-15,71,4
1050R=R+1:PRINT"CORRECT!"':GOTO1090
1060PRINT'"INCORRECT THIS TIME."
1070 PRINT'"THE ANSWER IS ";Z
1080 IF ABS(C)=ABS(Z) THEN PRINT'"ONLY
     THE SIGN IS     WRONG."
1090 PRINT'"SCORE ";R;" OUT OF ";N
1100IFN>9THENPROCmorestop
1110PRINT''"PRESS SPACE TO GO ON"
1120L$=GET$:IFL$<>" "THEN1120
1130ENDPROC
1140 DEF PROCmorestop
1150 PRINT'"FINISHED ";N$
1160 PRINT'"FOR ANOTHER GO       PRESS Y"
1170 J$=GET$
1180IFJ$="Y"ORJ$="y" THEN 50
1190CLS:PRINTTAB(0,4)"GOODBYE ";N$
1200PRINT'"I HOPE YOU HAVE MORE"
1210PRINT"SUCCESS WITH"
1220PRINT'"DIRECTED NUMBERS"
1230PRINT'"NOW!!!"'''
1240 END
1250ENDPROC
```

```
10 MODE7
20 PRINT'"********   SHEEP   STATION
   ********"
30 PRINT'TAB(8,8)"By Don Thorpe"
40 PRINT'''"Do you want instructions?
   (Y or N)"
50 *FX21,0
60 ins$=GET$: IF ins$="N" OR ins$="n"
   THEN 200 ELSE PROCinstru
70 GOTO200
80 PRINT" JANUARY ":RETURN
90 PRINT" FEBRUARY ":RETURN
100 PRINT" MARCH ":RETURN
110 PRINT" APRIL ":RETURN
120 PRINT" MAY ":RETURN
130 PRINT" JUNE ":RETURN
140 PRINT" JULY ":RETURN
150 PRINT" AUGUST ":RETURN
160 PRINT" SEPTEMBER ":RETURN
170 PRINT" OCTOBER ":RETURN
180 PRINT" NOVEMBER ":RETURN
190 PRINT" DECEMBER ":RETURN
200 MODE5
210 REM Set up windows, redefine palet
te, suppress cursor, define special char
acters
220 REM 224 fox, 225 sheep, 226 rabbit,
227 deads, 228 deadf, 229 deadr
230 REM colours  0 green, 1 red, 2 yel
    low, 3 white
240 VDU23,224,0,64,132,136,140,254,72,
72,23,225,0,0,0,60,255,127,125,37,23,226
,0,8,8,8,28,48,112,56,23,227,0,0,0,36,36
,124,255,63
```

```
250 VDU23,228,0,0,0,0,40,40,102,253,23
,229,0,0,0,0,4,2,59,255,19,0,2,0,0,0,23;
8202;0;0;0;17,2
260 VDU28,0,26,19,0,12,24,0;0;1279;170
    ;18,0,130,16
270 REM INITIALISE
280 year=1:month=0:sheep=14000+RND(900
):begin=sheep:foxes=200+RND(50):rabbits=
8000+RND(999):pasture=sheep+RND(999)+150
0:weather=0:capital=100000:store=2000-RN
D(999):priceb=20:feed=0:reward=0:balesf=
0
290 REM START OF MAIN LOOP**********
    **********
300 IF sheep>pasture THEN sheep=pasture
    +feed
310 feed=0
320 weather=weather-2+RND(3):IF weather
    <-3THENweather=-3
330 IF weather>3 THEN weather=3
340 IF month<9 THEN season=month+4
350 IF month>8 THEN season=month-8
360 pricesheep=4+RND(6)+season*2
370 interest=INT(capital*.01):capital=
    INT(capital*1.01)
380 pasture= INT(20000*(1+weather/30)*
(1-rabbits/80000)):IF pasture<6000 THEN
pasture=6000+RND(500)
390 IF pasture>40000 THEN pasture=4000
    0:weather=-2
400 nfer= foxes*rabbits/(rabbits+sheep
): rabbits=INT((rabbits-10*nfer)*1.12):s
heep=INT(sheep-foxes+nfer):IF rabbits<2
THEN rabbits=2+RND(20)
```

```
410 IF sheep<50 THEN PROCend
420 month=month+1 : IF month=13 THEN m
onth=1 : year=year+1: IF year=6 THEN PRO
Cend
430 PROCdisplay:PROCwindow
440 *FX21,0
450 REM Disable escape key with *FX229
    ,1 and re-enable with *FX229,0
460 option=GET : IF option<48 OR option
    >57 THEN 460
470 REM killr,killf,buys,sells,blank,
    buyf,NOaction,blank,blank,resign
480 ON option-48 GOTO 490,500,510,520,
    530,540,300,460,460ELSE550
490 PROCkillr:PROCdisplay:GOTO440
500 PROCkillf:PROCdisplay:GOTO440
510 PROCbuysheep:PROCdisplay:GOTO440
520 PROCsellsheep:PROCdisplay:GOTO440
530 PROCfeeds:PROCdisplay:GOTO440
540 PROCbuyf:PROCdisplay:GOTO440
550 COLOUR1: PRINT TAB(0,22)"Do you re
    ally want  to resign? (Y or N)" ·
560 resign$=GET$:IF resign$="Y" OR res
ign$="y" THEN CLS:CLG:END ELSE COLOUR3:
VDU31,0,22:PRINT"
                ":VDU11,11:COLOUR2:
    GOTO440
570 REM END OF LOOP********************
    *********
580 DEF PROCwindow
590 VDU5,25,4,0;157;18;1:PRINT"1 ";:VD
U18;0:PRINT"Kill ";CHR$226;" ";:VDU18;1:
PRINT" 6 ";:VDU18;0:PRINT"Buy feed";:VDU
18;1:PRINT"2 ";:VDU18;0:PRINT"Kill ";CHR
```

```
$224;" ";:VDU18;1:PRINT" 7 ";:VDU18;0:PR
INT"Inaction";
600 VDU18;1:PRINT"3 ";:VDU18;0:PRINT"B
uy  ";CHR$225;" ";:VDU18;1:PRINT" 8 ";:V
DU18;0:PRINT"         ";:VDU18;1:PRINT"4
";:VDU18;0:PRINT"Sell ";CHR$225;" ";:VDU
18;1:PRINT" 9 ";:VDU18;0:PRINT"         "
;
610 VDU18;1:PRINT"5 ";:VDU18;0:PRINT"F
eed ";CHR$225;" ";:VDU18;1:PRINT" 0 ";:V
DU18;0:PRINT"Resign"
620 VDU4
630 ENDPROC
640 DEF PROCend
650 CLS
660 IF sheep<50 THEN VDU7: PRINT"The s
heep populationhas vanished and so shoul
d you!!"
670 IF sheep<2000 THEN PRINT"What grudge
    have    you against sheep?"
680 IF capital<-100000 THEN P .'"You
    don't handle    money too well."
690 IF sheep>begin THEN reward=INT((sh
    eep-begin)*10 + capital/15)
700 PRINT'"Your payment for    five ye
    ars of work ="
710 PRINT"$";reward
720 END
730 ENDPROC
740 DEF PROCdisplay
750 VDU12:PRINT" YEAR ";year;"  ";
760 ON month GOSUB 80,90,100,110,120,
    130,140,150,160,170,180,190
```

```
770 PRINT"----------------------"
780 IF month=9 THEN VDU31,11,6:PRINT"I
ncluding":VDU31,12,7:PRINT"Foxcubs":VDU3
1,12,8:PRINT"& Lambs":foxes=INT(foxes*1.
4):sheep=INT(sheep*1.75):GOTO830
790 fate=RND(60):IF fate>3 THEN 830 EL
SE VDU31,10,6:COLOUR1:PRINT"A disease":V
DU31,10,7:PRINT"broke out":VDU31,10,8:PR
INT"resulting":VDU31,10,9:PRINT"in ";
800 IF fate=1 THEN COLOUR2:PRINTCHR$22
9:PRINT;:rabbits=RND(10)+INT(rabbits*(2+
RND(7))/10)
810 IF fate=2 THEN PRINTCHR$228:PRINT;
  :foxes=RND(6)+INT(foxes*(4+RND(5))/10)
820 IF fate=3 THEN COLOUR3:PRINTCHR$22
    7:sheep=INT(sheep*(7+RND(2))/10)
830 COLOUR2: PRINT TAB(0,3)" There are
 now:":COLOUR3: PRINTTAB(1,5)CHR$(225);"
 ";:COLOUR2:PRINT;sheep:COLOUR1:PRINTTAB
(1,7)CHR$(224);" ";:COLOUR2:PRINT;foxes:
COLOUR2:PRINTTAB(1,9)CHR$(226);" ";rabbi
ts
840 PRINT'"Pasture now supports";
850 PRINT;pasture;:COLOUR3:PRINT" "CHR
$(225);:COLOUR2:PRINT" So ";:IF sheep>pa
sture THEN PRINT;sheep-pasture ELSE PRIN
T"none"
860 PRINT"will starve without extra fe
    ed."'
870 PRINT"Interest payment    this mon
    th= $";interest
880 COLOUR3: PRINT"Your present balanc
    eis $";capital:COLOUR2
```

```
890 PRINT"Maximum allowable   debt is
    $200,000"
900 ENDPROC
910 DEF PROCkillr
920 COLOUR1:PRINT TAB(0,22)"Hunters ch
arge $500 visiting fee plus $1bounty per
   "CHR$229
930 COLOUR3: INPUT"How many dead rabbits
    do you need",deadr
940 IF deadr>rabbits OR deadr<0 THEN 9
    30
950 rabbits=rabbits-deadr:capital=capi
tal-deadr-500:IF rabbits<2 THEN rabbits=
5+RND(14)
960 ENDPROC
970 DEF PROCkillf
980 COLOUR1: PRINT TAB(0,22)"Hunters c
harge $500 visiting fee plus   $20 bount
y per "CHR$228
990 COLOUR3: INPUT"How many dead foxes
do you need",deadf
1000 IF deadf>foxes OR deadf<0 THEN 990
1010 foxes=foxes-deadf:capital=capital-
deadf-500:IF foxes<2 THEN foxes=1+RND(4)
1020 ENDPROC
1030 DEF PROCbuysheep
1040 COLOUR3: PRINTTAB(0,22)"At present
sheep areselling for $";pricesheep
1050 INPUT"How many do you wantto buy?"
incsheep:capital=capital-incsheep*prices
heep:IF capital<-200000 THEN PROCend
1060 sheep=sheep+incsheep
```

```
1070 ENDPROC
1080 DEF PROCsellsheep
1090 COLOUR3: PRINTTAB(0,22)"At present
sheep areselling for $";pricesheep
1100 PRINT"How many are you    selling?"
1110 INPUTdecsheep:IF decsheep>sheep OR
decsheep<1 THEN 1110
1120 sheep=sheep-decsheep:capital=capit
     al+decsheep*pricesheep
1130 ENDPROC
1140 DEF PROCfeeds
1150 COLOUR1:PRINT TAB(0,22)"At present
number ofbales stored=";store
1160 PRINT"How many will you   feed your
     sheep?"
1170 INPUTfeed
1180 IF feed>sheep-pasture THEN feed=sh
     eep-pasture
1190 ENDPROC
1200 DEF PROCbuyf
1210 COLOUR1:PRINT TAB(0,22)"One bale 1
asts 1 ";:COLOUR3:PRINTCHR$225;:COLOUR1:
PRINT" 1";
1220 PRINT"month, cost=$";priceb
1230 PRINT"You are storing";store
1240 INPUT"How many wanted?"balesf
1250 COLOUR2
1260 capital=capital-balesf*priceb:store
     =store+balesf
1270 ENDPROC
1280 DEF PROCinstru
1290 CLS:PRINT'"You are managing a shee
p station for a  five year period. Your
aim is to build  up the wealth of the st
ation. You have  $100,000 in the bank, a
```

```
nd may overdraw  the account up to $200,
000."
1300 PRINT'"In time of severe weather o
r rabbit      plague, you will need to bu
y extra feed.This can be stored too."
1310 PRINT'"Sheep can be bought and sol
d. Market     factors determine prices, w
hich usually rise as September (lambing)
 approaches."
1320 PRINT'"You will receive monthly re
ports. If thefarm survives five years yo
u will be     paid an amount which depend
s on the sizeof the flock, and the balan
ce  in the    bank.  GOOD  LUCK!"
1330 *FX21,0
1340 PRINT''"Press any key to continue."
1350 FOR delay=1 TO 9000:NEXT:key$=GET$

1360 CLS:PRINT'"The animals"
1370 PRINT"___________"'
1380 PRINT"The sheep breed once a year.
They          compete with the rabbits for
the           available pasture, which var
ies with       the weather. Both species sh
are a          common enemy, the fox."
1390 PRINT'"Foxes have one litter a yea
r. However,   the rabbits breed the whole
 year round."
1400 PRINT'"Diseases can break out whic
h affect one or more of the species.Shee
p and rabbitscan both be affected by a p
lant disease.Rabbits and foxes can  be
```

```
affected by   diseases against which the
  sheep have  been innoculated."
1410 *FX21,0
1420 PRINT''"Press any key to continue."
1430 FOR delay=1 TO 9000:NEXT:key$=GET$

1440 CLS: PRINT'"Strategy"
1450 PRINT"________"'
1460 PRINT"There are various ways to ga
in wealth.  Some prefer caution. Others
opt for big flocks and matching overdraf
ts. Some    buy and sell sheep a lot."
1470 PRINT'"It is impossible to wipe ou
t the rabbitsand foxes completely. Some
are wily.     Some develop immunities. Al
so, rabbits  and foxes will enter from s
urrounding   properties."
1480 PRINT'"If you remove too many rabb
its because  they are eating the sheep's
 pasture, thefoxes will eat more sheep.
If you  kill too many foxes because they
 eat  your   sheep, the rabbit populatio
n will  boom."
1490 *FX21,0
1500 PRINT''"Press any key to continue.
     ":key$=GET$
1510 ENDPROC
```

SECTION IV –

Designing a custom program for a non-computing colleague.

Scene:

A non-computing colleague with specialist knowledge of subjects different from yours, asks for help to produce a program on a certain topic. It might be for example "wave-action" in Geology. This section examines ways of co-operating to produce a program which is both educationally sound and efficiently designed. Also, a need often arises outside the areas of particular subjects: the handling of lists, marks records etc. How should this be dealt with?

Establishing realistic expectations

People unfamiliar with computers often have quite unreal expectations of a school computer. Here are some that I have come across.

A student wants to turn the computer on and type in a request for information about a certain Test Cricket match in 1980.

A teacher wants it to tell him the most likely winner in a horse race.
A parent wants her child to have access to the computer at lunch time to catch up on school work missed due to illness.
A teacher has morbid fears that the students will use the BBC computer to find out the credit ratings of the teachers.
A music teacher who has never touched a computer sends a child around to borrow one so that she can demonstrate computer music to the class she is taking.

A gentle re-education

How can one explain to non-computerist people what a computer is and is not? What is a "program"? There are many excellent books written in a language suitable for beginners which describe how a computer works. Here are a few points useful for discussion with a non-computerist colleague.

A computer is not an "electronic brain". It has no intellect.

Typically, a microcomputer has two types of memory. One is ROM, the "book of rules" set down by the designers, by which the machine operates. It cannot be altered by the user. The other type of memory is RAM, which is available to the user. Information can enter RAM in many ways, including the keyboard, tape, disc and modem. RAM is cleared when the machine is turned off.

Hence a BBC computer will not become "wiser" or more knowledgable with use. It is like the human brain however in that it can only provide information which is a combination of that which is in ROM (the "sub-conscious) and currently in RAM ("conscious memory")

How does one describe what "BASIC" is?

The computer is in part a number arrangement of locations which have either a "high" voltage level or not. This is a "binary" situation like yes/no or on/off. Mathematically such a system only needs two digits, 0 and 1. The computer operates at this level, but for us to communicate with it using zeros and ones, is slow and tedious. It is usually only done by ROM designers, and only when they can't avoid it. A language like English is too complex and irregular to be a computer language. It is necessary to invent a highly restricted version of English (or Russian or whatever) with an utterly unambiguous syntax and a vocabulary of only a few hundred words, known as "keywords". Such a language is called a "high-level" computer language, other examples being forth and Pascal. BBC BASIC is quite "high" as computer languages go, but still very much based in favour of the machine. It is a meeting place for the human and the machine.

Many non-computing people believe that programming in BASIC is very mathematical. This frightens those who struggled with mathematics as students. However there is usually only a little simple arithmetic involved in BBC BASIC (calculating a screen position for example). Of course if a program is written on a mathematical topic, there is a wide range of mathematical abilities in the machine, and then one needs mathematical skills. Hence learning BASIC is at first more like learning a tourist version of a foreign language than embarking on a new branch of mathematics. The main requirement when communicating with a computer in BASIC, is to tell it exactly what you want done, in language it understands.

Producing a program, in whatever language, is not like cooking scrambled eggs, where mistakes cannot be undone. A program can be repaired, improved and extended until it is excellent. This makes it a very appealing and rewarding activity for many people.

Crystallize the aims of the program.

When conferring with a non-computing colleague over the production of a program, the following can be used as a guide.

1. What are the educational characteristics of a typical "target student"? How long do you envisage them spending with the program? What do you expect the student to gain?
2. Is the program to be multi-level, or have no choice of difficulty?
3. Do you want the program to present identically for every run, or to be different each time?
4. Is it the type of program which the student is intended to use once only, or repeatedly?
5. Is the program to be demonstrated by a teacher or used by small groups or individuals?
6. Will you expect the program to "stand alone" or be supported by a supervising teacher, printed matter or other assistance?
7. Is the nature of the work to be covered remedial, course work, or extension work?
8. Do you want the name of the student entered?
9. Do you want the program to keep some kind of score?
10. Will you need the program to interact with a printer?

Your colleague's program may need some special characters.

They may be meteorological symbols, or furnishing symbols for a house design, little symbols for statistical graphs, or perhaps dice which roll across the screen.

These characters can be designed by your colleague on an 8 x 8 grid, or combination of such grids. You can then encode them and use VDU23, as shown in the "Sheep Station" program.

There may be more involved than just designing the special characters. For example, the characters of a foreign alphabet may need to be printed on the screen vertically, beginning at the bottom right.

A graphic presentation of scientific apparatus can be built up from special characters and then animated. For example, a flask might be represented, with its contents gradually boiling away. A bunsen flame beneath it, could be represented and graphically adjusted from inside the program, with a corresponding change in the boiling action of the liquid.

Involve your colleague as much as possible at every stage and see if you can "demystify" what you are doing as you go along.

Testing the effectiveness of the program.

Although this topic was discussed in the previous section, there is a distinction to be made in the present context. Since

we are assuming that the subject area of the program is outside your province, we need to divide effectiveness testing into two parts:

1. The effectiveness of the program in translating your colleague's ideas into a good teaching aid.
2. The educational soundess of those ideas. Evaluation of the second part is in the hands of your colleague. You will need their frank opinions on the first part.

The overall worth of the program will be decided by its effects on the students and teachers who you hope will use it. It may or may not have clever graphics, sound etc. It may or may not make a significant educational statement. It must not be boring.

Aids to classroom and school management.

Computerist teachers will be asked more and more to write programs useful in the area of general administration. Heads and staff in schools are discovering the speed and efficiency of programs which print names in alphabetical order, or keep library, sporting or academic records, or handle timetable and roster chores. People can be spared many routine tasks and their creative energies used much more beneficially in the school.

The documentation needed with programs of this type was discussed in section III. It could be summed up as follows. Take the risk of insulting the intelligence of the user, by explaining every detail simply. When people first learn to drive or solder or thread a needle or cook a meal, unfamiliarity

can make them feel awkward, confused even harassed. yet quite dim and clumsy individuals often become proficient at all these things. It may be appropriate to produce a "Beginner's Guide" to accompany your disc or tape.

Sometimes a program is available cheaply from commercial sources. A filing system is a good example. It is poor economics for a teacher to spend a hundred hours 're-inventing the wheel". The result often compares unfavourably with the professional product. It can even cause ill feeling on a staff. In one instance a faulty filing program designed by a teacher, destroyed a list of names and addresses of old scholars. Foolishly, no other copy had been kept.

In any case, your computing time as a teacher is much more fruitfully spent in educational areas.

Here are two programs to help with the handling of marks and names.

```
10 REM  MARKS
20 REM  Don Thorpe 1983
30 REM
40 REM INITIALIZE
50 DIMentry(50) : sum = 0 :delay = 40
  : marknum = 1 : loud = -6 : onpaper=0
60 PROCinsl
70 MODE 4
80 VDU23;8202;0;0;0;:REM Turn cursor off
90 REM Disable ESCAPE key.
100 REM Disable BREAK key.
110  PROCenter
120 PROCprintem
130 PROCanykey
200 DEF PROCinsl
```

```
210 CLS
220 PRINTTAB(0,1)"********** TEACHER'S
    FRIEND *********"
230 PRINT'"          by Don Thorpe 1983"

240 PRINT"-----------------------------
-------"
250 PRINT"  This program adds and aver
ages marks, and arranges them in descend
ing order."
260 PRINT'"  Marks from 0 to 100 can be
entered,   including half marks."
270 PRINT'"  The RETURN key is only us
ed to correctentries. When entering a tw
o digit       number, do not pause unduly
 between      the digits. Half marks are
added by      typing a full-stop."
280 PRINT'"  For a relaxed rate of ent
ering marks, press R. For rapid entry, p
ress any      other key."
290 *FX21,0
300 relax$ = GET$ : IF relax$ = "R" THEN
    delay = 200
310 ENDPROC
320 DEF PROCenter
330 VDU28,0,31,39,7,17,1,17,128,24,0;8
00;1279;1023;18,0,0,18,0,129,16,5,18;0,2
5;4,50;1010;
340 PRINT"Press I if entry was incorre
    ct."
350 MOVE50,970 : PRINT"Press F when fi
    nished."
360 MOVE50,930 : PRINT"Press . to add
```

```
.5 to the last entry."
370 MOVE50,890 : PRINT"Press ^ to turn
    sound on or off."
380 MOVE50,840 : PRINT"***  Or type in
    the next mark.  ***"
390 VDU4
400 PRINT'"Please enter mark number ";
    marknum
410 *FX21,0
420 digit1$ = GET$
430 IF digit1$ >= "0" AND digit1$ <= "
    9" THEN 490
440 IF digit1$ = "I" OR digit1$ = "i"
THEN marknum = marknum -1 : VDU11,11,7 :
 PRINT'" **  Let's try again. **   "': G
OTO 400
450 IF digit1$ = "F" OR digit1$ = "f"
    THEN marknum = marknum - 1 : ENDPROC
460 IF digit1$ = "." THEN marknum= mar
knum-1 : entry(marknum)=entry(marknum)+.
5 : VDU11 : VDU11 :GOTO 530
  470 IF digit1$ = "^" THEN loud = - lou
d : PRINT'"Sound status changed as reque
sted."'
480 GOTO420
490 digit2$ = INKEY$(delay)  : IF digi
    t2$ = "" THEN 510
500 IF digit2$ < "0" OR digit2$ > "9"
    THEN 490
510 entry$ = digit1$ + digit2$ : entry
    (marknum) = VAL(entry$)
520 SOUND&0201,loud,40,5 : SOUND&0202,
    loud,68,5 : SOUND&203,loud,93,5
```

```
530 PRINT';entry(marknum);" was entere
d for mark number ";marknum : marknum =
marknum + 1
  540 PRINT'"------------------------
------"
550 GOTO 400
560 ENDPROC
600 DEF PROCprintem
610 VDU26,12: IFonpaper THEN onpaper=0
    : VDU2
620 PRINT"Here are the marks:"
630 col=0 : row = 2 : sum = 0
660 FOR mark = 1 TO marknum
670   IF mark=26 THEN col = 20 : row =2
680   VDU31,col,row : row = row + 1 :
    PRINTmark;,entry(mark)
690   sum= sum + entry(mark)
700 NEXT
710 PRINT TAB(0,27)"Those ";marknum;"
    marks add up to ";sum
720 PRINT"Their average is ";0.1*INT(1
    0*sum/marknum)
730 VDU3
740 ENDPROC
780 DEF PROCmenu
790 CLS : PRINTTAB(0,1)"Press the number
    of your choice."
800PRINT''"1  ENTER EXTRA NUMBERS"
810 PRINT''"2  CHANGE ONE OF THE ENTRI
    ES"
820 PRINT''"3  DISPLAY THOSE MARKS AGAIN
     on the screen"
830 PRINT''"4  DISPLAY THOSE MARKS AGAIN
    and print on paper too"
```

```
840 PRINT''"5  ARRANGE THE MARKS IN DE
SCENDING ORDER   and print on the screen"
850 PRINT''"6  ARRANGE THE MARKS IN DE
SCENDING ORDER   and print on paper too"
860 PRINT''"7  ENTER A NEW SET OF MARKS"
870 PRINT''"8  QUIT THE PROGRAM"
880 choice = GET
890 IF choice < 49 OR choice > 56 THEN
    880
900 ON choice-48 GOTO 910,920,120,1020
    ,980,1020,1000,1270
910 marknum = marknum+1 : CLS: GOTO110
920 PROCprintem : VDU31,0,29
930 INPUT"ENTER THE NUMBER OF THE ENTR
Y YOU WANT  TO CHANGE AND PRESS RETURN "
changenum
940 INPUT"Enter the changed mark and p
    ress RETURN "newmark
950 entry(changenum)=newmark
960 PROCprintem
970 PROCanykey
980 PROCarrangem
990 PROCanykey
1000 sum = 0 : marknum = 1 : CLS : GOTO
    110
1010 REM
1020 PROCarrangem: onpaper=-1 : GOTO 120
1030 REM
1040DEF PROCarrangem
1050 N = marknum : S=1
1060 max=entry(S) : min = max : IM = S
     : IX = S
```

```
1070 REM look through remaining entries
and pick smallest and largest ******
1080 FOR I = S TO N
1090 IF entry(I) < min THEN min = entry
     (I) : IX = I
1100 IF entry(I)>max THEN max = entry(I)
     : IM = I
1110 NEXT
1120 IF IM = N THEN IM = IX
1130 AA = entry(N) : entry(N) = entry(I
     X) : entry(IX) = AA : N = N-1
1140 AA = entry(S) : entry(S) = entry(I
      M): entry(IM)= AA : S = S+1
1150 IF N>S THEN 1060
1160 PROCprintem
1170 ENDPROC
1210 DEF PROCanykey
1220 VDU31,0,30 :  PRINT"Press any key
     when ready."
1230 *FX21,0
1240 reply$ = GET$
1250 PROCmenu
1260 ENDPROC
1270 REM Re-enable ESCAPE key.
1280 REM Re-enable BREAK key.
1290 CLS : PRINT''''"      Goodbye."
```

```
10 REM NAMES
20 MODE7
30 DIMname$(200)
40 start%=0:num=-1:onpaper=0
50 PRINT''"        **** NAME ARRANGER
   ****"
60 REPEAT
70 *FX202,32
80 PRINT''"Enter surname first. Enter
@ to finish. Press RETURN after each en
   try."
90 num=num+1: PRINT''"Name number ";n
   um+1
100 INPUT'name$(num)
110 IF name$(num)="" THEN 100
120 CLS
130 UNTIL name$(num)="@"
140 num=num-1
150 finish%=num
160 PROCsort(start%,finish%)
170 CLS:PRINT'"HOLD SHIFT KEY DOWN TO
    SCROLL NAMES."'
180 FOR num=0 TO finish%
190 PRINTname$(num)
200 IF onpaper THEN 230
210 IF INKEY(-1)=0 THEN 210
220 delay=INKEY(5)
230 NEXT
240 onpaper=0: VDU3
250 *FX202,32
260 PRINT'"        **** MENU ****"
270 PRINT'"  Press A to add a name.
    Press C to Correct a name.
    Press D to Delete a name."
280 PRINT"  Press H for a Hard copy.
```

```
Press S to see the list again."
290 key$=GET$
300 IF key$="H"THEN VDU2:onpaper=-1:GO
    TO180
310 IF key$="C" THEN 490
320 IF key$="D"  THEN 550
330 IF key$="S" THEN 170
340 IF key$="A" THEN num=num-1:CLS:GOTO
    60
350 GOTO290
360 END
370 DEF PROCsort(start%,finish%)
380 LOCAL M$,N$,A%,B%
390 A%=start%: B%=finish%
400 M$=name$((A%+B%)DIV2)
410 REPEAT
420 IF name$(A%)<M$ A%=A%+1:GOTO420
430 IF M$<name$(B%) B%=B%-1:GOTO430
440 IF A%<=B%  N$=name$(A%):name$(A%)=
  name$(B%):name$(B%)=N$:A%=A%+1:B%=B%-1
450 UNTIL A%>B%
460 IF start%<B% THEN PROCsort(start%,
    B%)
470 IF A%<finish% THEN PROCsort(A%,fin
    ish%)
480 ENDPROC
490 CLS:INPUT'"Enter the number of the
    entry you wish  to correct. "cornum
500 IF cornum<1 OR cornum>finish%+1 TH
    EN490
510 INPUT"Now type in the correct entr
    y and        press RETURN."'corname$
520 name$(cornum-1)=corname$
530 PROCsort(start%,finish%)
```

```
540 GOTO 170
550 CLS:INPUT'"Enter the number of the
   entry you want  to delete.  "delnum
560 delnum=delnum-1
570 FOR k=delnum TO finish%
580 name$(k)=name$(k+1)
590 NEXT
600 finish%=finish%-1
610 GOTO170
```

WORDS

Here is another version of the NAMES program, which allows the list to be stored. File handling is beyond the scope of this book, so DATA statements have been used. Different blocks of line numbers can be allocated to groups of related words: eg student names in year groups. These can be amended or deleted as required.

```
10 REM WORDS
20 MODE7:DIMname$(800)
30 start%=0:num=-1:onpaper=0
40 PRINT''"        **** WORD ARRANGER
   ****"
50 PRINT''"When sorting, numbers are
placed first, then UPPER CASE then lower
 case.";
60 PRINT'"Insert the words in DATA st
atements      between lines 490 and 30000
. The presentline 500 is a sample only ."
70 PRINT'"After the chosen line number,
```

```
   type DATA and then the entries separa
ted by       commas."
80 PRINT'"These can be changed by lis
ting the     program and adding, deletin
g or         correcting entries as desir
ed."
90 REPEAT
100    num=num+1:READ name$(num)
110 UNTIL name$(num)="@"
120 num=num-1:finish%=num
130 PROCsort(start%,finish%)
140 PRINT'"HOLD SHIFT KEY DOWN TO SCROLL
    ENTRIES."'
150 FOR num=0 TO finish%
160 PRINTname$(num)
170 IF onpaper THEN 200
180 IF INKEY(-1)=0 THEN 180
190 delay=INKEY(5)
200 NEXT
210 onpaper=0: VDU3
220 *FX202,32
230 PRINT'"       **** MENU ****"
240 PRINT'"Press H for a Hard copy.
    Press S for a screen display.
    Press Q to Quit the program."
250 key$=GET$
260 IF key$="H" THEN VDU2:onpaper=-1:GO
    TO150
270 IF key$="S" THEN CLS:GOTO 140
280 IF key$="Q" THEN CLS:END
290 GOTO250
300 DEF PROCsort(start%,finish%)
310 LOCAL M$,N$,A%,B%
320 A%=start%: B%=finish%
```

```
330 M$=name$((A%+B%)DIV2)
340 REPEAT
350 IF name$(A%)<M$ A%=A%+1:GOTO350
360 IF M$<name$(B%) B%=B%-1:GOTO360
370 IF A%<=B%  N$=name$(A%):name$(A%)=
 name$(B%):name$(B%)=N$:A%=A%+1:B%=B%-1
380 UNTIL A%>B%
390 IF start%<B% THEN PROCsort(start%,B%)
400 IF A%<finish% THEN PROCsort(A%,fin.
    ish%)
410 ENDPROC
420 REM *** PUT LINES OF DATA STATEMEN
TS AFTER THIS *** LINE 500 IS A SAMPLE O
NLY
500 DATA FISH,CHIPS,fish,chips,45,19,s
paces can be left
30000 DATA @,FINISH
```

The BBC Micro is an immensely powerful computer. And its potential for assisting in education is enormous. But the means of tapping that potential are not immediately obvious.

In this detailed work, experienced teacher Don Thorpe discusses the role the BBC Microcomputer could occupy in your teaching activitives. He compares its possibilties with those of other, more traditional, teaching aids, and outlines specific areas of application in which the computer has no rival.

You'll be shown - in step-by-step detail - how to run a first course in computing, the simplest way to get complicated subjects across to students, and how to maintain the students' interest once it has been aroused.

Designing an educational program is a demanding task, but this task is made much simpler for you with the explicit rules and suggestions outlined in the third part of this book. And this is no empty theory. Don Thorpe has devised these ideas based on solid experience in classrooms just like yours. See, for example, his advice on controlling unruly students. This is no armchair theorising.

Finally this book will help you design a custom program for a non-computing colleague. You'll be shown how to lead your fellow-teachers to an understanding of the possibilities of computers (so they do not have unrealistic expectations); how to crystallize their objectives, and how to implement them.

If your school has a BBC Micro in use, or is contemplating acquiring one, you need this book.

Another great book from

INTERFACE PUBLICATIONS

ISBN 0-907563-87-2

9 780907 563877

£5.25