A sinclair COMPUTERGUIDE

EDUCATIONAL USES OF THE ZX SPECTRUM
A guide-book for teachers and parents
By Tim Hartnell, Christine Johnson and David Valentine

Have fun *and* learn something. That is the message of this book. It is addressed to parents as well as teachers – in fact to all those who would like to do something more than thread mazes, play adventure games or zap aliens.

Tim Hartnell and his collaborators have produced a book packed with ideas and programs: simple and more advanced mathematics, graphics, languages, spelling, reading, are some of the subjects which are made interesting for the learner by using the computer's possibilities for interaction and moving display.

Christine Johnson contributes a detailed account, with illustrations, of how she introduced a computer into an infant school.

The numerous appendices include suggestions for further reading, a list of suppliers of Spectrum educational software, introductions to LOGO and PROLOG, and a glossary.

Published by Sinclair Browne Ltd., and marketed and distributed by John Wiley & Sons Ltd.

Cover Design by Jim Reader
Illustration by Martin Salisbury

£6.95 net

ISBN 0 946195 14 5 (paper)

EDUCATIONAL USES OF THE ZX SPECTRUM Tim Hartnell, Christine Johnson, and David Valentine

Educational uses of the ZX Spectrum

TIM HARTNELL, CHRISTINE JOHNSON, DAVID VALENTINE

A sinclair COMPUTERGUIDE

A sinclair COMPUTERGUIDE

Educational uses of the ZX Spectrum

A guide-book for teachers and parents

By
Tim Hartnell, Christine Johnson
and David Valentine

Sinclair Browne: London

First published by Sinclair Browne Ltd
10 Archway Close, London N19 3TD

British Library Cataloguing in Publication Data
Educational uses of the ZX Spectrum
1. Sinclair ZX Spectrum (computer)
I. Hartnell, Tim
001.64'04 QA76.8.S625

ISBN 0-946195-14-5

Book designed by Jim Reader
Production Services by Book Production Consultants, Cambridge
Printed and bound by The Alden Press, Osneymead, Oxford
Typeset by Cambridge Photosetting Services, Cambridge

Contents

Acknowledgements

A number of people have helped us write this book and we would like to thank them for their assistance.

Firstly, and most importantly, we must acknowledge the work of Mr Crispin Hill, Headmaster, Aldro School, Shackleford, who was program consultant for this project. Many of the programs in this volume are based on those in Mr Hill's booklet *'Introductory Notes and Simple Programs for the ZX81'*.

Others who contributed programs were:

Jeremy Ruston – NEWTON'S METHOD FOR SOLVING EQUATIONS
– BINARY/DECIMAL CONVERTER
David Perry – FACTOR FINDER
– TEXT MANIPULATION:
WRITING UPSIDE-DOWN
WRITING SIDEWAYS
FRAMING WORDS
LARGE LETTERS
CHARACTER GENERATOR
Jim Walsh – DECIMALS
Gwyn Dewey – MATHS
SUPERSKETCH
Paul Toland – CORETS
Derek Cook – ANAGRAMS
– THE KIMSPOT GAME
Gordon Armitt – FASTER READING

To avoid the pontifical 'we', we've stuck to 'I' throughout most of this book. In nearly all cases we have referred to 'he and she' rather than just one or the other, but any reference to one is intended to refer to both.

We hope this book will help you make effective use of

your Spectrum in education.
Tim Hartnell, London
Christine Johnson, Nottingham
David Valentine, Newthorpe

CHAPTER ONE

Computers and education

In this book, we'll be looking at a number of ideas for using the ZX Spectrum in education. Whether you are a teacher interested in using the computer to teach about computers and computer programming, or whether you wish to use the machine to assist with the teaching of another subject, you'll find material which should be of interest and benefit to you.

Perhaps you're a parent, and you bought your child a Spectrum in the hope that he or she would use it to help with their school work. You may, however, have been somewhat dismayed to find that the computer's main application seems to be to demolish 'aliens'. If this is the case, you'll find information in this volume to ensure that – at least some of the time – the Spectrum is used for the reason you bought it. Although I will be addressing teachers from this point on, much of what I say will apply to your situation. Please read the material with this in mind, adapting it to your own needs.

Microcomputers are now widespread within the education system. A wide variety of machines have prevented, to some extent, the creation of a universal library of programs (often called 'software'). The material in this book will go some of the way towards rectifying that lack for the Spectrum. Many programs included in this book are for educational use of the Spectrum. They are here mainly to give information on how such programs can be written and to give sample programs which can be tailored to your needs.

The demands of specific subjects are clearly defined. It is unrealistic to expect that a program written for one subject will apply to another one, or that a program created to assist students at a particular level in one subject will be of anything other than limited use for students working on the same subject at a different level. Nevertheless, if you use this book as a guideline, as a source of ideas, you'll find it

should save you a lot of time when preparing material for your own students or children.

It is a somewhat disturbing fact – but one which must be faced from the outset – that many of the children you'll be dealing with know far more about computers and computer programming than you. Being brought up with the micro has already produced a generation of whom many do not feel even vaguely threatened by the computer. The exotic jargon – bits, bytes, RAM and ROM – is a sea in which many young people appear to swim without undue problems.

There are more 'micros per head' in Britain than in any other country. Thanks to government programs which have promoted the acquistion of computers, and the range of locally-built, cheap microcomputers which is available, the penetration of computers into schools is approaching the point where there is at least one computer which the students can use, in each school. But this abundance of machines has brought its own problems. The number of teachers who can make effective use of this exciting resource is limited. It is this problem I seek to address in this volume.

I've included a section designed to teach the rudiments of programming in BASIC (the language the computer uses, an acronym for Beginners All-purpose Symbolic Instruction Code, developed at Dartmouth College in America in 1974). After you've taught yourself to program with this section, you'll probably use this as a framework for teaching BASIC within the classroom.

This book must make certain assumptions about teaching and your methods of teaching. If you do not agree with all I say, it is possible to adapt the material to your own needs. I am addressing a wide audience with this volume, and it is inevitable that some sections will be more relevant to your needs than others.

You will probably also disagree quite strongly with some of my assertions and assumptions. By all means discard anything which seems incorrect to you, inappropriate for your teaching methods, or inapplicable in your school environment. As I said a little earlier, look on the book as a source of ideas. No one can really say 'This is the only way to use the Spectrum in your teaching situation'. Please do not assume that I am suggesting this, although from time to time the book may appear, in fact, to be advocating such an inflexible position. Rather than spend the entire volume

qualifying every suggestion with 'if it seems appropriate to you', 'if it fits into your philosophy of education' or 'if it appears valuable for your needs at the moment', I will assume that you will reject what you don't want.

There is a wide range of materials in this book. Not all of it will be directly applicable to your needs, though I hope all of it will be of interest. Most of it should help you work out ideas to use the Spectrum for your own needs. Use the book however you like. The sequence of material presented here is only one of many possibilities. Dip into the book at whichever point seems relevant to your needs.

Note that there is a glossary of terms at the back of the book. After all, whole dictionaries of computers have been published. I've tried to select the most common computer terms, but this is by no means exhaustive. You'll find that any technical term whose meaning is not clear from the context in which it appears, will be further explained in the glossary.

There are many ways you can use the Spectrum in education. In this section, I'll discuss just a few of them. This list is by no means exhaustive, and it should start you thinking about other ways you can use the computer for your own needs.

The Spectrum can be used to help teach a subject by arranging the facts in a way which is interesting, clear and – possibly – interactive. That is, a program which displayed, for example, the table of the elements could be set up in such a way that colour was used to highlight the 12 which do not occur naturally (possibly with FLASH used to indicate unstable ones), or the program could be written so that a group with particular properties could be easily isolated. When you start thinking about this simple example, you can probably see that any body of material which can be presented in tabular form can be presented in a way which is interesting, and invites interaction from students. There is no way a simple printed table in a book can *involve* students in the same way that a well-presented computer-generated table can do.

There are a large number of programs which are generally classed as 'spread-sheet calculators' which invite users to handle data in a 'what if' environment. These programs allow the user to enter information on such things as costs and sales, to produce a balance sheet, and then enter such

questions as 'What would happen if we doubled output?', 'What would happen if sales rose 3½% per month for the next three months?', or 'What will happen to our profitability picture over the next financial year if we lose the Jones contract?'. The program then takes over, changing all figures affected by the hypothesis you have advanced. There are ways in which this process can be used to enrich the experience of students working with material in tabular form as outlined in the previous paragraph. For example, a simulation program could allow students to mix chemicals in varying proportions – even to the point of explosion.

The Spectrum can be used in mathematics, and several applications in that area are outlined in detail in this book. Although the computer can be used, more or less, just as an electronic calculator, using it in this way rather misses the point (and certainly makes minimal use of the machine's potential). As you'll see in the section on maths, there are a great many ways of using the computer effectively in this area.

I mentioned using the computer to produce tables of information which encouraged interaction with students. There are many things, in addition to tables, which can be presented by the computer to encourage student involvement, such as maps for geography and pie- and bar-charts for a variety of subjects. Some areas of knowledge, such as the ideas of Malthus regarding the interaction between available resources and population growth, could be modelled extremely effectively on the computer. In subject areas like this, it is hard to imagine a more effective way of getting the message across than by computer simulation.

Returning to the subject of maps, there is much that can be done again using on-screen modelling, to make ideas come alive. Animated maps, for example, could show such things as the effects of land height on rain distribution and the way the population of an area changed during the Industrial Revolution. A number of complete pictures can be stored in the Spectrum as strings, and these can be printed one at a time over each other.

One of the most common uses of the Spectrum in education is to present multiple choice questions. A multiple choice quiz framework which you can use to create tests for any subject of your choice is introduced a little later in the

book. We'll also be looking at a simpler program which does not give a series of choices, but simply looks for a correct answer.

The fact that the majority of computers in use in homes appear to be used at least part of the time for game-playing, shows the immense fascination game-playing with computers can exercise. Rather than decry this, it seems better to me to capitalise on this fascination, and ensure, whenever possible, that some aspects of computer game-playing (such as dramatic rewards for achieving a certain score) are incorporated into 'straight' programs.

Carrying this idea further, I've included a chapter of 'brain-stretching games' which, while they do not teach a specific subject, may well assist in the development of number and reading skills. From this chapter, you can see how many computer games can be modified to include sufficient educational 'return' to make them worth including in your school's computer activities. At the very least, having such material on tap can be useful in maintaining interest in the computer, and act as a break from straight learning activities. Such programs are also useful icebreakers for starting students off with computers.

CHAPTER TWO

Basic ideas: What is a computer? How do computers work?

Although it is no more necesssary to know how computers work in order to make use of them, than you need to understand the complexities of the internal combustion engine to drive a car, it is inevitable that you'll one day be asked 'How does a computer think?' or some such question. It is useful, anyway, to have at least some understanding of this.

The American National Standards Institute (ANSI) has defined a computer as a 'device capable of performing systematic sequences of operation upon data, including numerous arithmetic and logic procedures, without intervention by a human operator during the run'. This is not a definition which produces illumination easily. Let's take it section by section.

The first important phrase is 'systematic sequences of operation upon data'. The Spectrum is not like a radio. You cannot just plug it in and expect that it will do very much. If you connect your Spectrum to a television, and tune it in correctly, the computer will at least put an identifying message on the screen. But that's about it. Unless you *program* the machine to do something else, it will just sit there.

A computer needs a program in it before it can do anything. This book contains many programs, and a host of others are available in magazines. You can also buy programs on cassette tapes, or on microdrives, which only need to be loaded into the Spectrum in order to run.

As you'll learn shortly in the 'programming primer' part of this book, each statement (or line) within a computer program starts with a *line number*. The computer, unless told to do otherwise, works through a program line by line, in line number order. In other words, it carries out the instructions of each statement in a 'systematic sequence'.

'Data' in the ANSI definition, refers to any information – be it numbers, symbols or characters, or a combination of these – which the computer can process.

That covers the first part of the definition. The 'numerous arithmetic and logic procedures' which the computer can perform are fairly easy to understand. The adjective 'arithmetic' covers all the standard number manipulations – such as adding, subtracting, raising to a power, determining the square root, etc. – while the 'logic procedures' which the computer can perform include comparisons (of size, length or some other characteristic, such as position within a series) and the group of activities governed by the laws of 'Boolean algebra'* which can make decisions based on AND and OR (if this is true AND this is true, then do this . . . or if A is true and B is not true OR C is true, then carry out sequence D). The numerical manipulations, and the decision-making based on AND/OR conclusions, governs all the activities of the computer.

The final part of the ANSI definition, 'without intervention by a human operator during the run', is crucial. A computer is not the same as an electronic calculator. A calculator demands a series of responses to prompts, while it is operating. A computer can draw the material it needs from other places (such as from a cassette, from a disk, or from elsewhere within the program).

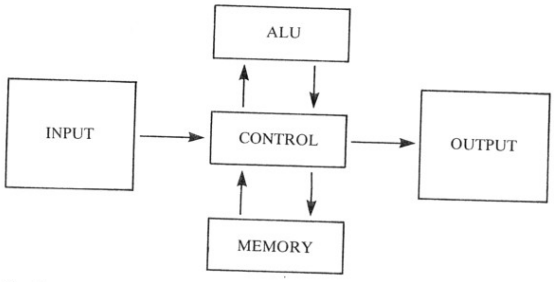
The Spectrum follows the instructions you give it, carrying out precisely the task you've ordained for it. Give it a program to generate multiple choice questions, and the Spectrum will do so (assuming the program has been correctly written). Unless you have programmed it to so do, halfway through the run the Spectrum will not start playing

*George Boole, an English logician and mathematician (1815–1864) was the first man to formulate laws which placed logical decisions within a mathematical framework. 'Boolean algebra', which he developed, allowed *operations* (such as the AND and OR we mentioned, along with the rules governing the conclusions of such comparisons – IF this THEN do this) to be carried out in accord with strict mathematical laws. Prior to the development of Boole's ideas, it had been assumed that logic was a branch of philosophy. With his books *Mathematical Analysis of Logic* (1847) and *Investigation of the Laws of Thought* (1854), Boole showed clearly that logic was a branch of mathematics.

an Irish folk song, or fill the screen with a wave of invaders hell-bent on the destruction of earth. The Spectrum will carry out your instructions methodically. Any apparent errors in judgement it makes will be yours.

There is no need, however, to be intimidated by this. Thanks to the unambiguous nature of the Spectrum's BASIC language, it is extremely easy to tell the machine to do exactly what you want it to do. As you'll see in the programming primer section, much of BASIC is very close to English. The programming word PRINT, for example, means just that. It tells the computer to PRINT something on the television screen. Other words you'll meet which are exactly (or very close to) what they mean in ordinary English, are AND, OR and IF.

There are five basic parts of every computer, as you can see in this diagram.



Each computer must have a way of getting information into it from the outside world. This is indicated by the section marked INPUT on the left-hand side of the diagram. The input, in the case of the Spectrum, is usually the keyboard, although devices are available which allow some limited information to be fed into the machine via a microphone. In addition you can call up some information from an external memory device (the cassette recorder, or a microdrive).

The information which has been entered via the INPUT section then goes to the three thinking parts of the machine. One, which we've called CONTROL, acts as a traffic policeman and timekeeper on the whole system, making sure that the activities of the computer occur in the correct sequence and at the right time. The ALU (arithmetic and

logical unit) makes the decisions and does the sums, and the MEMORY (which we'll look at in a little more detail in a moment) holds not only the intermediate results of whatever the computer happens to be working on at the time, but also contains the information the computer needs in order to be able to do such things as add numbers together, and to make decisions.

Finally, once the computer has reached a conclusion, it is sent to the outside world (that is, to you) via the section we've called OUTPUT. The output is either to the television screen, the printer, or the microdrive (or to any combination of these).

Although our diagram suggests that all the parts of the computer are distinct, it is not – as you probably realise – as clear-cut as that. In the case of the Spectrum, a custom-made chip combines the majority of the activities covered by the headings CONTROL, ALU and MEMORY. However, it is convenient to think of the parts of the computer in separate sections defined by their activities.

I said we'd look at memory in a little more detail. And here we come up against the first 'frightening' bit of jargon. The acronyms ROM and RAM are scattered throughout computer books and magazines. Despite the fact that they are words which do not appear outside the world of computing, they embody concepts which are not too difficult to grasp.

There are two types of memory in a computer (although these may be *stored* in different ways). The first type is ROM, which stands for 'Read-Only' Memory. Although the Spectrum needs a program in order to tell it what to do, it contains a great deal of knowledge when you buy it. Locked into a chip inside the case is the information the Spectrum needs to be able to run the television screen, to understand what you are typing in on the keyboard, to add numbers together, to compare different quantities and so on. You cannot change this information. It is etched indelibly into the chip. You can only *read* this information when you need to. Therefore, the memory which holds the basic intelligence and raw working information of the Spectrum is called Read-Only Memory (ROM).

In contrast to this, the computer needs a less permanent area of memory to store the program you have currently entered, and to store the intermediate results of its

calculations. You can jump about within this memory at random, moving to the start of it if you like, or to anywhere within it. Because of this random access feature, the impermanent memory is called Random Access Memory (RAM).

The only things you need to remember in order to use the terms ROM and RAM accurately is that ROM is fixed memory that tells the Spectrum how to do things like add numbers together, and RAM is the impermanent (computer people often say 'volatile') memory which holds the current program, and the results of processing. You can't alter ROM, but you can, and do, modify RAM.

CHAPTER THREE

Programming the Spectrum in BASIC

In this section of the book, we'll be looking at computer programming on the Spectrum in BASIC. The material here is intended for your own use, so that you can learn to program. Once you've done this (and you'll be pleased to discover it is a surprisingly easy task), you can use the information here as the basis for teaching your own students to program.

You will find, however, that you'll have to expand this material to some extent. Whole books have been written on how to program the Spectrum, and we can hardly hope – in just part of one book – to cover the ground in the same depth as some other books.

The Keyboard

The first thing you need to learn, unfortunately, is your own way around the keyboard. I say unfortunately because the keyboard can seem bewildering and intimidating to first-time users. However, as it would be impossible (obviously) to program without having some degree of mastery of the keyboard, it is here that we must start. (By the way, I'll assume from now on that you have a Spectrum turned on as you read, and that you'll enter the material I describe as you come to it.)

Once you've got the computer plugged in, following the instructions in the manual, and your TV switched on, you'll see the copyright message at the bottom of the screen. Now press the ENTER key. The message will disappear, and a flashing 'K' will appear. This is the *cursor*.The cursor will follow you along a line as you type it, will indicate errors in program lines, and the actual letter it is made up from (such as the K in this case), tells you which *mode* the computer is in. As you can see, it does quite a bit.

The three most important keys on the computer are those marked CAPS SHIFT, SYMBOL SHIFT and ENTER. Find them now. The ENTER key is used *after* you enter any instruction, program line or response to a question from the machine. It tells the Spectrum to act on whatever you have just typed in.

CAPS SHIFT works just like the shift key on a typewriter. Normally the computer works in lower case letters. However, when you hold down shift, the letters come out in upper case. Hold down the CAPS SHIFT and then press the CAPS LOCK key (the key marked with the number 2) and all letters from then on will be in upper case.

If you look closely at the keys, you'll see most of them have a word or symbol (such as ?, * or +) in red on them. You get these by holding down the SYMBOL SHIFT key (whose legend is also in red) and then pressing down the relevant key.

You 'clean the computer out' at any time by pressing the A key, which has the word NEW written on it in white. NEW is a drastic command. There is no way to get back any program material which was in the computer after NEW has been pressed, so regard the A key with a little awe. Press the A key now, so that the NEW appears on the screen, and then press ENTER. The screen will go black, and then clear to show the copyright notice again.

Now that the computer memory is empty, we can start our exploration of the keyboard in earnest. Many of the keys have words written on them in white. These are called *key-words*. One of the great features of the Spectrum is its 'one-touch key-word entry system'. This means that, instead of typing in a computer programming word such as PRINT letter by letter as you need to do with most other makes of computer, you just touch the P key (which has the word PRINT written in white on it) and the word PRINT appears at the bottom of the screen.

You'll recall that the cursor was a flashing K. The K stands for key-word. Whenever you press a key which has a key-word written on it in white, and the cursor is a K, you'll automatically get the key-word appearing on the screen.

However, once the key-word has appeared, as you can verify if you look closely, the K cursor changes into an 'L'. This stands for *letter*. Press one of the alphabet keys now, and you'll see the relevant letter appearing. The Spectrum

works more or less like a typewriter when in the 'letter mode'.

This is the time to try out the CAPS SHIFT and CAPS LOCK keys, to change the lower case letters appearing on the screen into upper case ones. (You'll see that once you do this, the L cursor will change into a 'C' to remind you that CAPS LOCK has been engaged.)

The bottom of the screen may now be getting pretty crowded with the letters and so on you've been typing. Pull the power plug out, wait a few seconds, and then replace it.

So far, we've looked at the key-word and the letter modes. You know, now, how to get all the things which appear actually *on* the keys: key-words, letters and the material printed in red (which, as you'll recall, you get by holding down the SYMBOL SHIFT key and pressing the relevant key at the same time.

However, there is more to the keyboard than the material we've discussed so far. You can see there are words in green above the keys, and words in red below them. You get the words *above* the keys by pressing down both shift keys at once (the CAPS SHIFT and the SYMBOL SHIFT) then letting them go. Do this now, and then have a look at the cursor. It has changed into an 'E'. This stands for 'extended mode'. In this mode, the computer accepts the functions which appear above the keys. Press, for example, the T key, and you'll get RND (which is the function for random numbers). The cursor will immediately change back into an L after a function has been selected.

Underneath the keys are legends in red. You get these by pressing both shift keys at the same time as when going into extended mode. However, while you release the left-hand one (CAPS SHIFT) you do not release the right-hand one (SYMBOL SHIFT). Still holding down SYMBOL SHIFT, press any key with a word or symbol *underneath* it, and the relevant word will appear. Pressing the Z key, for example, while in the extended mode and holding down the SYMBOL SHIFT will get you the word BEEP.

We have just about covered our tour of the keyboard, and its modes and cursors. There is one more we need to know about, the graphics mode. Hold down the CAPS SHIFT key, and then press the 9 key, which has the word GRAPHICS above it. The cursor will change into a 'G'. Now, press any of the keys on the top row of the keyboard,

and you'll see the little patterns printed on the keys appear on the screen. Hold down CAPS SHIFT and press the same number keys again, and you'll get the inverse of those patterns. You get out of the graphics mode by holding down CAPS SHIFT, and pressing the 9 again.

As I said at the outset, it is unfortunate that this tour of the keyboard and its cursors needs to be carried out before you can proceed. However, you've now covered most of the necessary ground. The only one thing you might need to note is DELETE which is written above the zero key. DELETE rubs out words working from the end of a word towards its start. You use it by holding down the CAPS SHIFT key, then pressing (or holding down, waiting for the auto-repeat to start) the zero key. I suggest you stop at this point, and go back and read through the material again. Once you're familiar with the keyboard you can continue.

Building Blocks

All BASIC programs are made up from a number of standard words, the building blocks of a program. These 'blocks' include words which are close to English (such as PRINT, THEN and PAUSE) and which are understood by the computer to mean more or less what they mean in English. Other words (like INKEY$, ATTR and SCREEN$) are foreign territory.

The most common word in most programs is the word PRINT. You use this, as I guess you'd imagine, to put material on the screen. Try this now. Clear out the computer by disconnecting the power for a few seconds, or by using DELETE and NEW.

Press the P key, to get the word PRINT, and then type a number after it, as follows:

```
PRINT 7
```

Now press the ENTER key (remember, you need to press the ENTER to get the computer to act on the material you've just typed in) and a 7 should appear at the top of the screen.

To use the Spectrum as a calculator, follow the word PRINT with a sum, such as the following:

```
PRINT 7 + 4 - 1
```


(I'll assume from now on you'll press ENTER after entering each line, so I will not keep reminding you to do so.) The result of the sum, 10, should appear on the top of the screen.

You can make the arithmetic you want the computer to solve as complex as you like. However, you'll see that the keyboard does not have a divide symbol. And, as you can easily verify for yourself, using the X to indicate multiplication, will simply confuse the computer. The asterisk (*) is used to indicate multiplication, so PRINT 7*3 will give 21. The slash (/, on the V key) indicates division. PRINT 21/7 will give an answer of 3.

The other symbol you'll need is the little arrow on the H key (which means raise to the power) so 3↑2 is the Spectrum's way of indicating three squared.

Test Program

We'll start with a simple program which will be able to teach a great deal. Enter the following program into your Spectrum, pressing ENTER at the end of each line. If you've entered the line correctly, you'll see it move to the top of the screen when you press ENTER. If you've made a mistake, you'll find the line will not be accepted by the computer. The cursor will move to the part of the line where it thinks the problem exists, which will help you track down typing errors.

Anyway, type in the following, and then press the R key (to get the keyword RUN) which tells the Spectrum to execute your program:

```
10 REM test program
20 PRINT "Enter a number"
30 INPUT a
40 PRINT a;" and another"
50 INPUT b
60 CLS
70 PRINT "Your numbers were ";
a;" and ";b
80 PRINT
90 PRINT "They add up to ";a+b
```

When you run this, you'll see the following on your screen:

```
Enter a number
123
123 and another
 88
```

```
Your numbers were 123 and 88
They add up to 211
```

We will go through this program line by line. It is quite amazing how much can be learnt from it. Firstly, as you can see, each line begins with a *line number*. These line numbers can be any numbers at all between 1 and 9999. Generally, the computer will execute the lines in order, from lowest to highest (although, as you'll see shortly, this is not an inflexible rule).

The first line of the program starts with the word REM. This stands for remark. Any REM line in the program is ignored by the computer. REMs are included simply to convey information to someone reading the program. A REM statement will often be used at the start of a program, as in this case, to say what the program is meant to do. REMs are also used *within* programs to point out what the following program line, or program section, is doing. REM statements make it much easier, when you return to a program after some time, to work out (a) what the program is supposed to do, and (b) how it does it.

In line 2Ø, the command PRINT is used to get the words "Enter a number" on the screen. You'll see that these words are enclosed within quote marks. You have to enclose letters within quote marks if you want the computer to handle them as words. Any material within quote marks, such as in the program line, is called a *string*.

Line 3Ø allows you to enter a number of your choice. INPUT means that the computer waits for the user to enter something, such as a number as in this case. The letter 'a' after the word INPUT is a *numeric variable*. A numeric variable is a letter (or any combination of letters and numbers, so long as it starts with a letter) to which the computer can assign a number. From then on, the computer treats the numeric variable as if it was the number. You can see this, further down the program, where the computer uses 'a' (and 'b', whose value is entered in line 5Ø) in lines 7Ø and 9Ø to print out firstly the numbers you've entered, and then to work out the value of their sum.

The word CLS in line 6Ø means 'clear the screen'. You probably noticed that the screen cleared after you entered your second number, before printing up the final two statements.

You can see that there is a great deal we have learned from this very brief program. Note as well (and this is important) that you need to use a semi-colon to join the 'a' outside the quote marks in line 4Ø to the rest of the material to be printed.

Spectrum Number

Get rid of that program from your computer by using NEW, then enter the following one.

There are a number of new elements in this program which could be tricky to enter, so I suggest you start typing the program in, and if you have any problems, read the notes referring to that line.

```
  10 REM Spectrum number
  20 LET a=INT (RND*50)+1
  30 LET guess=0
  40 INPUT "What is your first n
ame? ";b$
  50 PRINT "Hello, ";b$
  60 PRINT '"I am thinking of a
number"
  70 PRINT "between one and 50 w
hich you"
  80 PRINT "have to try and gues
s."
  90 LET guess = guess + 1
 100 PRINT '''"This is guess num
ber ";guess
 110 INPUT "Enter your guess ";c
 112 CLS
 115 PRINT "Your guess, ";b$;",
was ";c
 120 IF c=a THEN PRINT "Well don
e, you got it right!": PRINT "It
 took you ";guess;" guesses": ST
OP
 130 IF c>a THEN PRINT "That num
ber is too high"
 140 IF c<a THEN PRINT "That num
ber is too low"
 150 GO TO 90
```

This is what it looks like when it is running:

```
What is your first name? #Tim


Hello, Tim
I am thinking of a number
between one and 5Ø which you
have to try and guess.


This is guess number 4
```

```
Enter your guess, #7

Your guess, Tim, was 7
That number is too low

.
This is guess number 5

Enter your guess, #9

Your guess, Tim, was 9
That number is too high

.
This is guess number 6

Enter your guess, #8

Your guess, Tim, was 8
Well done, you got it right!
It took you 6 guesses
```

1Ø – This REM statement is much the same as the first line in the first program.

2Ø – This line uses the word LET (the key-word on the L key) to assign a value to the variable 'a'. You'll see that after the equals sign in this line, the word INT (which reduces a number with a fractional part to the next *lowest* whole number). You get INT from the R key, pressing both shift keys down first, then releasing them, then pressing the R key. The open and close brackets come from the 8 and 9 keys respectively, and you get these by holding down the SYMBOL SHIFT KEY then pressing the keys required. The asterisk (multiplication sign) is on the B key. Again the SYMBOL SHIFT is needed to get this. This line uses the random number facility of the Spectrum to generate a whole number between one and 5Ø. Use a statement in the same form as this line whenever you need a random number. The number within the brackets is the highest one in the range you want.

3Ø – This gives a variable called 'guess' a starting value of zero. Remember, a numeric variable can be a combination of letters and/or numbers, so long as it starts with a letter.

4Ø – This INPUT statement is somewhat different from the first one we looked at. As you can see, there is a statement within quote marks before we get to the 'b$' at the end. When you run the program, the words in quote marks after an INPUT line appear at the bottom of the screen. The *input prompt*, as it is known, is a very useful feature, and

allows you to tell the user exactly what information the computer is expecting. At the end of that line is a variable followed by a dollar sign ('b$'). This represents a *string variable* (as opposed to numeric variables, which we have been using to date). A string variable can be any letter of the alphabet, followed by a dollar sign. When you enter your name, in response to the input prompt, it is assigned to 'b$'.

5Ø – This is a fairly standard PRINT line, which prints up "Hello," then follow this with your name.

6Ø – This is a standard PRINT statement . . .

7Ø – . . . as is this . . .

8Ø – . . . and this.

9Ø – The value of the numeric variable 'guess' is increased by one. It is good practice to use explicit names for variables, such as this use of 'guess'. Although it is fairly easy to keep track of what each variable in a program represents when the program is short, you'll find it becomes increasingly difficult to do so when your programs get longer. In such cases, you'll see how valuable explicit variable names can be. (For example, in the very long chess program in my book *Dynamic Games for the ZX Spectrum* Sinclair Browne, 1983) the variable P stands for the white pawn, K for the white king, and so on, with PB for the black pawn, and KB for the black king. This was of great help when the program was being written, and makes it simpler for others to follow through.)

1ØØ – This looks like a standard PRINT statement, except for the three apostrophes (''') at the start of the line. (The apostrophe is on the 7 key.) When the Spectrum comes across an apostrophe at the *start* of a PRINT statement like this one, it moves one line down the screen. So, the three apostrophes here move the PRINT position down three lines, as you'll see when you run the program.

11Ø – This uses another INPUT prompt, this time ending with the variable 'c', to accept your guess.

112 – CLS clears the screen after you've pressed ENTER following entering your guess.

115 – This line is printed, to remind you of your guess. Note that it uses both your name ('b$') and your guess ('c'), linking them to the other material to be PRINTed with semi-colons.

12Ø, 13Ø, 14Ø – I'll consider these three lines together as they perform similar (very important) functions. Note that

they all start with the word IF. When the computer comes across an IF statement, it looks at the condition which follows the word IF and if it is true THEN carries out the instruction which follows the word THEN. IF and THEN always appear together in the same line. IF the weather is cold THEN turn on the heater, IF the television is broken THEN call a repairman, and so on. Line 12Ø compares your guess ('c') with the computer's number ('a') and IF it finds they are equal, THEN goes on to the rest of the line. You'll see that line 12Ø contains three complete program statements. You can put more than one statement in a single line number, if you put colons between them. The first part of line 12Ø, after checking that the numbers are the same, prints up the congratulation message, and the second PRINT statement tells you how many guesses it took. Finally, the program stops at the STOP statement. Note that if the condition being tested by IF is found to be false, then *none* of the other statements following THEN on that line number will be executed. Instead, the program will immediately go to the next line. You'll see that lines 13Ø and 14Ø test to see how the computer's number compares with your guess, and print appropriate messages. The THEN is on the G key.

15Ø – You'll recall I pointed out a little earlier in this section that a computer generally follows through a program line by line. Two of the commands which can redirect its action are GO TO and GO SUB. We'll be looking at GO SUB a little later. GO TO (it is two words, but is considered as a single word, and comes as a single key-word, from the G key), as you have probably guessed, simply redirects the program back to line 9Ø, where the whole cycle begins again.

Enter the program, and run it several times, then return and reread these notes. Make sure you understand the material they contain before continuing.

I realise that the material in this section may seem heavy going. After all, we are covering – in just a few pages – topics that other books spend several chapters explaining. However, if you take it slowly, it should all make sense.

You'll find this is an ideal program to use as an ice breaker for students who have not had previous experience with a computer. It uses a student's name (which never fails to impress a first-time computer user), and is sufficiently easy

to play to ensure that any student can 'win' it in a short time, thus ensuring the first contact with the computer is an enjoyable one.

Save the program on cassette once you're sure you understand how it works (full instructions on the use of the SAVE and VERIFY commands are given in your manual). We'll be coming back to this program in due course when we learn about colour, so you'll need it on hand. I suggest you save each program three times on a single side of a C-12 cassette, and put nothing else at all on that side of the cassette. Although I realise you can save a bit on cassettes by putting a number of programs on a C-60 or C-90 cassette, the 'frustration cost' of trying to locate the program you want is so great it is not worth the trouble. If one copy of your program fails to LOAD, all you need to do is try to load the next one. Also, if one copy of the program is accidentally wiped, or the tape is damaged, you have backup copies.

High Roll

We'll be covering a lot of ground in our next program as well. In this program – HIGH ROLL – you and the computer take it in turns to roll a pair of dice. The player with the highest total wins that round. (By the way, I believe that game-playing, and game-writing, is the easiest and most pleasant way to learn computer programming. That is why the programs at the early part of this section on how to program, are all games.)

In this program, we'll be covering the following important additions to your BASIC vocabulary:

– PAUSE
– BRIGHT
– The use of subroutines (with GO SUB and RETURN commands)
– FOR/NEXT loops
– BEEP

Here's what it looks like when you get it up and running:

```
Press any key to roll the dice

Die one fell 6
Die two fell 4
```

```
So your score is 10

Stand by for my roll

Here we go...

Die one fell 6
Die two fell 2

So my score is 8
and you're the winner!
```

Enter the program, run it a few times, then return to the book so we can go through it, line by line.

```
  10 REM High Roll
  20 PRINT "Press any key to rol
l the dice"
  30 PAUSE 0
  40 GO SUB 1000
  70 LET human=a+b
  80 PRINT '"So your score is ";
human
  90 PAUSE 50
 100 PRINT BRIGHT 1;'"Stand by f
or my roll"
 110 PAUSE 50
 120 PRINT '"Here we go..."
 130 PAUSE 50: GO SUB 1000
 140 LET computer=a+b
 150 PRINT "So my score is ";com
puter
 160 IF computer=human THEN PRIN
T "and it's a draw!"
 170 IF computer>human THEN PRIN
T "and I'm the winner!"
 180 IF computer<human THEN PRIN
T "and you're the winner!"
 190 PAUSE 300
 200 RUN
1000 REM This is subroutine
1010 LET a=INT (RND*6)+1
1020 LET b=INT (RND*6)+1
1030 FOR g=1 TO 30
1032 BEEP .03,g: BEEP .03,50-g
1036 NEXT g
1040 PRINT '"Die one fell ";a
1045 PAUSE 50
1050 PRINT "Die two fell ";b
1055 PAUSE 50
1060 PRINT
1070 RETURN
```

1Ø – This REM statement simply identifies the program

2Ø – This is a standard PRINT statement such as we've encountered before

3Ø – The PAUSE command (on the M key) holds the

display for a length of time related to the number which follows PAUSE. If PAUSE is followed by 5Ø, (as in PAUSE 5Ø), or 6Ø in the USA, the computer will stop execution for one second. Touching any key during PAUSE terminates it. PAUSE Ø will wait for ever so the effect of this line is to hold program execution until you obey the instructions in line 2Ø to "Press any key to roll the dice".

4Ø – When the computer comes to a GO SUB (for GO to SUBroutine) command, it goes to the line which follows the words GO SUB (like GO TO, although they are a pair of words, they always appear together and are treated as a single word; it is on the H key). So far, it has behaved in a similar way to a GO TO command. However, the difference occurs when the computer comes to the word RETURN in the subroutine. In this program, the subroutine starts at line 1ØØØ. The Spectrum goes to line 1ØØØ, and then follows through each line in order from that point. When it gets to line 1Ø7Ø it comes to the word RETURN mentioned above. At this point, the program returns to the *line after* the one which sent it to the subroutine. In this case, it returns to line 7Ø (the first line which appears after the 4Ø, which sent it to the subroutine). Subroutines are used when there is a segment of the program which we wish to use in different parts of the overall program. Rather than include the whole thing each time it is needed, we can use a subroutine. You'll see from the program printout that the computer rolls two dice for you and then two for itself. The dice-rolling all takes place within the subroutine so the part of the program which actually 'rolls the dice' can be used twice.

7Ø – This is the line after the GO SUB call, so this is the line to which the computer returns. Within the subroutine (see lines 1Ø1Ø and 1Ø2Ø) numbers chosen at random between 1 and 6 are assigned to the variables 'a' and 'b'. On returning from the subroutine, these are added together and a variable called 'human' is set equal to their total.

8Ø – This prints up your score.

9Ø – The PAUSE command makes the program wait for one second.

1ØØ – A message is printed up, using BRIGHT. You'll find BRIGHT at the very bottom of the keyboard, underneath the B key. There are several ways to modify print output on the screen, including the use of INVERSE (which prints white letters on a black background) and

FLASH which turns the printout from black on white to white on black over and over again. Here, as you'll see when you run the program, BRIGHT has brightened the little strip of screen upon which the words "Stand by for my go" are printed. If a word like BRIGHT is followed by a 1 (as in BRIGHT 1), it means 'turn that command on'. If it is followed by a Ø (BRIGHT Ø) it turns that command off. BRIGHT and its companion words, as will be made clear in the next program, can be used 'globally' or 'locally'. If BRIGHT 1 appears in a program line all by itself, then it alters all print output for the rest of the program. In this case, it is said to be acting globally. If, as in this program, the BRIGHT appears as part of a PRINT statement, it only affects the output of that particular line. It is, as you can see in this program, being used locally.

11Ø – Another wait for one second. Using PAUSE in this way to pace a program's output is very effective, especially if you want to give the impression that the computer is thinking while trying to solve a puzzle. Often, of course, PAUSE is used just to make sure people using a program have time to read it before the message is cleared from the screen.

12Ø – The message "Here we go . . ." is printed on the screen, with the apostrophe moving the print output down a line.

13Ø – This is a multi-statement line, with the two statements being separated, as you can see, by a colon. After the one second wait, the subroutine is visited, for the second time during the program's run.

14Ø – After running through the subroutine, the computer returns to the line *after* the one which sent it to the subroutine. In this case, line 14Ø. Here the variable *computer* is set equal to 'a' and 'b', the numbers generated in lines 1Ø1Ø and 1Ø2Ø.

15Ø – The result of the computer's dice throw is shown on the screen.

16Ø, 17Ø, 18Ø – These three IF/THEN lines compare the values of the two variables, *computer* and *human*, and determine which of you have won the round.

19Ø – The program waits for six seconds.

2ØØ – A new run is initiated. Note that you can use RUN, just as you can use just about every other key-word, within a program.

1ØØØ – This REM statement introduces the subroutine.

1Ø1Ø – The variable 'a' is set equal to a number chosen at random between 1 and 6.

1Ø2Ø – The same happens to variable 'b'.

1Ø3Ø, 1Ø32, 1Ø36 – These three lines form a FOR/NEXT loop. Just as GO SUB and RETURN always appear together, so do FOR and NEXT. A FOR/NEXT loop is used for repeating the part of the program which lies between the FOR and the NEXT lines, the number of times specified by the *difference* between the two numbers in the FOR statement. In line 1Ø3Ø, which starts the loop, we read:

FOR g = 1 TO 3Ø

This ensures that the loop will be cycled through thirty times. You can see in line 1Ø36 ('Next g') the end of the loop. Note that the letter used in the loop ('g', in this case) must be the same in both the FOR and the NEXT statements. When the computer executes a FOR/NEXT loop, the controlling variable (the 'g') has a value of 1 the first time through the loop, 2 the second time through, and so on. Therefore, when 'g' is used to control the output of the BEEP command in line 1Ø32, it means that 'g' has a different value each time through the loop, which creates the rising and falling tones. The BEEP command (the word BEEP is under the Z key) is always followed by two numbers. The first controls the duration of the note (with .007 the shortest practical note) and the second number controls the pitch of the note (which must be in the range −60 to 60). The tone varies as it does because 'g', the pitch variable, is changing as the loop is cycled through.

1Ø4Ø – After this musical display, the result of the first die roll is printed on the screen.

1Ø45 – There is a delay of a second.

1Ø5Ø – The result of the second die roll is printed.

1Ø55 – There is another one second delay.

1Ø6Ø – The computer prints a blank line.

1Ø7Ø – This is the RETURN statement, which terminates the subroutine and sends the computer back to the line after the one which called up the subroutine.

Additional Commands

Our final program for this section is based on the second program, SPECTRUM NUMBER. If you have it on

cassette, you can just LOAD it back into your computer, and make the adaptations.

In this program we'll be examining the following new words:

- FLASH
- INK
- PAPER
- BORDER
- INVERSE

Modify the program so that it looks like the following, then return to the book for a discussion on the new material:

```
  10 REM Spectrum number
  20 LET a=INT (RND*50)+1
  30 LET guess=0
  40 INPUT FLASH 1; INK 2;"What
is your first name? ";b$
  45 INK 2: PAPER 6: BORDER 6: C
LS
  50 PRINT INK 1; PAPER 7;"Hello
 ";b$
  60 PRINT '"I am thinking of a
number"
  70 PRINT "between one and 50 w
hich you"
  80 PRINT "have to try and gues
s."
  90 LET guess = guess + 1
 100 PRINT '''"This is guess num
ber ";guess
 110 INPUT "Enter your guess ";c
 112 CLS
 115 PRINT "Your guess, ";b$;",
was ";c
 120 IF c=a THEN PRINT FLASH 1;"
Well done, you got it right!": P
RINT FLASH 1; INVERSE 1;"It took
 you ";guess;" guesses": STOP
 130 IF c>a THEN PRINT "That num
ber is too high"
 140 IF c<a THEN PRINT "That num
ber is too low"
 150 GO TO 90
```

```
Your guess, tim, was 39
Well done, you got it right!
It took you 7 guesses
```

As you can see, this is a much more exciting program than it was originally. It is amazing what a difference colour can make to the output of a program. It is worth keeping this in

mind when creating your own programs.

As you've no doubt realised, FLASH and INVERSE work in a similar way to the BRIGHT we came across in HIGH ROLL: follow the word with 1 and you turn it on (as in FLASH 1); follow it with a zero and you turn it off (INVERSE 0).

I mentioned these words could be used locally or globally. The same is true for the colour commands INK (which controls the colour in which you are printing), PAPER (the background on which you are writing) and BORDER (the frame around the screen). In line 40, the INK is used locally, so it only affects the INPUT prompt. In the next line, 45, the INK is set to red (colour number 2), the PAPER to yellow (colour 6) and the BORDER also to yellow. The CLS at the end of the line is needed to make the PAPER colour cover the whole screen. Without this (as you'll discover if you leave the CLS out) the computer will simply put a yellow strip under whatever it happens to be printing.

You might like to add some sound to the program, perhaps related to the difference between your number and the computer's number.

I'll now look briefly at several of the other words you'll use in programming your Spectrum in BASIC. These will not be treated in as much detail as the earlier ones were. However, from what you now know, and from what you should pick up looking at the other programs in this book (and deducing how the commands work from the context in which they are used, and the results they produce) you should have little trouble continuing your self education in this field.

EDIT – This is used to change the program lines quickly. Put the line marker [>] next to the line you wish to change, and then holding down CAPS SHIFT, press the 1 key, so the line comes to the bottom of the screen. You can then move the cursor along the line, using CAPS SHIFT and the arrowed keys (5 and 8) to get it where you want it, to add more material (simply by typing it) or remove material with DELETE.

STEP – This is used in conjunction with FOR/NEXT when you don't want to progress through the loop in steps of one (as happened in the HIGH ROLL program). The step can be a positive number (or a fraction) or a negative number if you wish to count down. The form is: FOR a = b

TO c STEP d.

LIST – Gets the listing of a program on the screen, starting from the lowest line number. You can also enter LIST n, where the computer will list with line n at the top of the screen.

TAB – Moves the start of print output across the screen. It is used in the form: PRINT TAB n;"string".

PRINT AT – Specifies the starting point on the screen of PRINT output. It is used as PRINT AT x,y;"string" where x is one less than the number of lines down the screen, and y is one less than the number of character spaces across the line (so PRINT AT Ø,Ø is at the top left-hand corner of the screen).

PLOT – Places a dot in the position on the high resolution screen designated. The form is PLOT x,y where PLOT Ø,Ø puts a point in the bottom left hand corner of the screen.

INKEY$ – Reads the keyboard for a single character input, and does not need the ENTER key to be used afterwards. It is used in the form IF INKEY$ = "N" THEN . . . or LET A$ = INKEY$:IF A$ = "N" THEN . . .

DEF FN – For defining functions.

SCREEN$ – Reports the start of a designated character cell; the parameters are the same as the PRINT AT ones.

VAL – Converts the contents of a string to its numerical equivalent so VAL A$, where A$ equals "12 + 3*4" will return 24.

STR$ – This is the opposite of VAL, turning a number into a string.

LLIST, LPRINT, COPY – These commands control the printer, with LLIST and LPRINT working the same as LIST and PRINT, except they address the printer rather than the TV screen. COPY is used when you want a copy of the contents of the screen dumped to the printer. COPY was used to get the 'sample runs' of the programs in this book.

CHAPTER FOUR

String handling and its use in Maths

As you probably know, the BASIC used on the Spectrum is not the same in all respects to the BASICs used on other computers. The different 'dialects' of BASIC are sufficiently close to ensure that once you've learned to program on one machine you can turn to another one and program it reasonably well almost immediately. And after an hour or so of studying the manual, and working out what the differences are from your 'home BASIC', you'll be programming without problems.

Many aspects of the BASIC on your Spectrum (often called 'Sinclair BASIC') are pretty standard (despite the fact that, paradoxically, there is as yet no such thing as a recognised standard for BASIC). The codes used for numerical and alphabetical characters, as well as several others, are the ordinary ASCII (American Standard Codes for Information Interchange) codes (in contrast to the ZX80 and ZX81, which used an original Sinclair system). The majority of the programming techniques you'll master on the Spectrum can be easily transferred to other computers at your school.

However, there is one part of Sinclair BASIC which is unique – its string handling. You'll recall from the previous chapter that any material held within quote marks – such as "Hello" – is called a *string* in computer jargon. Now, as I said, the string handling on your Spectrum is, so far as non-Sinclair computers are concerned, unique. Rather than manipulate the eccentricities of other BASIC's LEFT$, MID$ and RIGHT$, Sinclair BASIC reduces nearly all string handling to use of brackets and the word TO. Once you understand it (which can take all of three minutes) you'll be able to do just about anything you want with strings.

You'll recall that a string variable is a letter (A to Z) with a dollar sign suffix (as A$ or a$). The words associated with

string handling include LEN (as in LEN A$) which returns the *length* of the string (that is, the number of characters, symbols and spaces it contains) as a number; CODE which gives the *character code* of the first element of the string (PRINT CODE A$, when A$ = "A" gives 65, as does PRINT CODE "A"); and TO, which we will discuss in a moment.

Other words we will look at in a moment are VAL and STR$, but to introduce them at this point would only serve to confuse you.

CHR$

CHR$ (spoken aloud as 'char-dollar' or 'character string') is used to change the CODE (mentioned above) back into a character. For example, if you typed in (and you can try this on your Spectrum now) PRINT CHR$ 65, the computer would print the letter A. If you then typed in PRINT CODE "A" you'd get, as we pointed out before, 65.

Enter this next program into your computer, and run it a number of times until you are sure you understand how CHR$ and CODE work:

```
  10 REM USE OF CHR$ AND CODE
  20 INPUT "ENTER A NUMBER BETWE
N 1 AND 255 ";A
  30 PRINT '"CHR$ ";A;" = ";CHR$
 A
  40 INPUT "NOW ENTER A STRING "
;A$
  50 PRINT '"CODE ";A$;"" = ";C
ODE A$
  60 GO TO 20
```

Here is a short run from it:

```
CHR$ 76 = L
CODE "test" = 116
CHR$ 107 = k
CODE "HTES" = 72
CHR$ 99 = c
CODE "TIM" = 84
CHR$ 254 = RETURN
CODE "tim" = 116
```



```
CHR$ 34 = "
CODE "fffffff" = 102
```

The use of 'TO'

The most valuable string-manipulation tool you have on the Spectrum is called 'slicing'. With this, you can extract any section of a string that you want, add different parts of the same string together in any order you like, or add sections of different strings. And all of this can be done with the simple word TO.

Study the following, and see if you can work out how TO works:

LET A$ = "ABCDEF"

Then, PRINT A$(1) will give A
PRINT A$(1 TO 3) will give ABC
PRINT A$(2 TO 4) will give BCD
PRINT A$(4) will give D

In other words, a single number after the string in brackets will return just that *element number* of the string, as you can see by looking at A$(1) and A$(5) above. There are two variations on this. Whereas PRINT A$(1 TO 5) will produce ABCDE, as you'd expect, if you want *all* of the string from a specific point, you do not have to mention the final number (5 in this case). In other words, PRINT A$ (1 TO) will produce exactly the same result as PRINT A$(1 TO 5). Further, if you want all of the string *from the beginning to a specific point*, all you need to do is mention the final number. Therefore, PRINT A$(TO 3) will return ABC. (The TO, by the way, comes from the F key; it is *not* typed in full.)

LEN

The next program will select elements at random from a string you select. Notice, in line 4Ø, the reference to LEN, the length of the string. Enter the program, and run it for a while, putting in any string you like, such as your name:

```
10 REM STRING HANDLING
20 INPUT "ENTER A STRING ";A$
```

```
  30 PRINT "YOUR STRING IS ";A$
  40 PRINT '"LEN A$ = ";LEN A$
  50 PRINT '"A$(1) = ";A$(1)
  60 PRINT '"A$(2 TO ) = ";A$(2
TO )
  80 LET B=INT (RND*(LEN A$)+1)
  90 LET C=INT (RND*(LEN A$)+1)
 100 IF C>=B THEN GO TO 80
 110 PRINT "A$(";C;" TO ";B;") =
";A$(C TO B)
 120 GO TO 80
```

Here are parts of two runs of the program:

```
YOUR STRING IS OXFORD*CIRCUS
LEN A$ = 13

A$(1) = O

A$(2 TO ) = XFORD*CIRCUS
A$(2 TO 3) = XF
A$(6 TO 10) = D*CIR
A$(11 TO 12) = CU
A$(8 TO 12) = CIRCU
A$(2 TO 12) = XFORD*CIRCU
A$(5 TO 8) = RD*C
A$(9 TO 12) = IRCU
A$(1 TO 6) = OXFORD
A$(1 TO 6) = OXFORD
A$(7 TO 11) = *CIRC
A$(10 TO 11) = RC
A$(9 TO 13) = IRCUS
A$(2 TO 12) = XFORD*CIRCU
A$(5 TO 11) = RD*CIRC
A$(4 TO 5) = OR


YOUR STRING IS CLIVE*SINCLAIR
LEN A$ = 14

A$(1) = C

A$(2 TO ) = LIVE*SINCLAIR
A$(3 TO 12) = IVE*SINCLA
A$(3 TO 8) = IVE*SI
A$(7 TO 14) = SINCLAIR
A$(9 TO 14) = NCLAIR
A$(6 TO 9) = *SIN
A$(5 TO 14) = E*SINCLAIR
A$(2 TO 4) = LIV
A$(3 TO 11) = IVE*SINCL
A$(4 TO 14) = VE*SINCLAIR
A$(4 TO 8) = VE*SI
A$(4 TO 7) = VE*S
A$(8 TO 14) = INCLAIR
A$(5 TO 9) = E*SIN
A$(1 TO 12) = CLIVE*SINCLA
A$(8 TO 13) = INCLAI
```


A string can be chopped up and added to other parts of the same string, or to parts of other strings. Adding strings is known as *concatenation*. (You cannot, by the way, subtract strings from each other.)

The next program selects parts of a string you enter and concatenates them. The little section in lines 45, 46 and 47, gives another example of how easily strings can be manipulated. Here, the string is printed back to front, as you can see in the sample run which follows the program listing. To stop this program you will need to press the BREAK key.

```
  10 REM STRING MANIPULATION
  20 INPUT "ENTER YOUR STRING ";
A$
  30 PRINT "YOUR STRING IS ";A$
  40 LET L=LEN A$
  45 FOR G=L TO 1 STEP -1
  46 PRINT A$(G);
  47 NEXT G
  50 LET B=INT (RND*L)+1
  60 LET C=INT (RND*L)+1
  70 IF B=C THEN GO TO 60
  80 PRINT 'B;" ";C
  90 IF B>C THEN PRINT A$( TO C)
+A$(B TO )
 100 IF B<C THEN PRINT A$( TO B)
+A$(C TO )
 110 GO TO 50

YOUR STRING IS CLIVE<>SINCLAIR
RIALCNIS<>EVILC
14 3
CLIR

8 13
CLIVE<>SIIR

9 10
CLIVE<>SINCLAIR

3 7
CLISINCLAIR

11 9
CLIVE<>SINLAIR

11 4
CLIVLAIR

6 1
C<>SINCLAIR
```

VAL

Another very useful string-handling word in BASIC is VAL.

This returns the *numerical equivalent* of the string. This is how it works:

LET A$ = "23 + 3"
PRINT A$ will give you, as you'd expect, 23 + 3
PRINT VAL A$ returns 26

You can use VAL to produce a simple calculator on the Spectrum, which will work out just about anything to tell it to do:

```
10 REM STRINGS AS NUMBERS
20 INPUT "ENTER YOUR CALCULATI
ON ";A$
30 PRINT 'A$;" = ";VAL A$
40 GO TO 20
```

Here's the program in action:

```
33+6-19 = 20

2↑5-7 = 25

LN 7-SQR 12 = -1.5181915

(79-4↑2)-COS 3 = 63.989992

(45/7+SGN 7)↑34.789 = 1.9857502E
+30
```

The opposite of VAL is STR$, which turns a number into a string. That is, telling the computer to set A$ equal to STR$ 123 will turn A$ into "123".

The whole point of our discussions to date in this chapter is to lead up to the extraordinary usefulness of string manipulation for number work. Because *parts* of strings can be easily extracted in a way in which parts of numbers cannot be, and because numbers and strings can be interchanged using STR$ and VAL, we have a powerful tool with which to handle numbers.

To introduce this, run the following program:

```
10 REM NUMBERS AS STRING
20 INPUT "ENTER YOUR NUMBER ";
A
25 PRINT "YOUR NUMBER IS ";A
30 LET A$=STR$ A
40 LET L=LEN A$
50 LET B=INT (RND*L)+1
60 LET C=INT (RND*L)+1
70 IF B>=C THEN GO TO 50
80 PRINT '"A$ (";B;" TO ";C;")
= ";A$(B TO C)
90 GO TO 50
```


As you can tell from this sample run, much of the output is fairly useless:

```
YOUR NUMBER IS 12.345
A$ (2 TO 4) = 2.3
A$ (2 TO 6) = 2.345
A$ (3 TO 4) = .3
A$ (2 TO 5) = 2.34
A$ (3 TO 4) = .3
A$ (5 TO 6) = 45
A$ (1 TO 2) = 12
A$ (1 TO 5) = 12.34
A$ (1 TO 3) = 12.
A$ (4 TO 5) = 34
```

However, look down the output till you come to A$(1 TO 2). You'll see here this has returned 12, the number stripped of the figures which follow the decimal point. Just below it is A$(1 TO 5). Here, 12.34 has been returned, the number stripped of the third digit after the decimal point. If we could control the stripping which takes place, we will indeed have a very useful device for manipulating numbers.

Correction to N Significant Figures

The rest of this chapter will seek to prove the accuracy of that claim. Our first program, as you can see, corrects it to the number of significant figures you specify:

```
10 REM TO CORRECT ANY POSITIVE
    NUMBER < 10E7
   TO N SIGNIFICANT FIGURES
20 INPUT "ENTER YOUR NUMBER ";
X$
25 INPUT "AND HOW MANY SIGNIFI
CANT","FIGURES DO YOU WANT ";(X$
),"CORRECTED TO? ";N
30 LET Y=VAL X$*(10↑(LEN X$-1)
)
40 PRINT 'X$;" to ";N;" signif
icant","figures is ";INT (VAL ST
```

```
R$ Y/10↑(LEN STR$ Y-N)+.5)*(10↑(
LEN STR$ Y-N))/(10↑(LEN X$-1))
  50 GO TO 20
```

```
54637 to 2 significant
figures is 55000

.054672 to 3 significant
figures is .0547

1/674 to 3 significant
figures is 0

674/45 to 3 significant
figures is 10

674/45 to 7 significant
figures is 14.978
```

The next version of the program is broken down to show what is happening. (For negative numbers, by the way, enter the digits only.) Line 3Ø turns the number into a whole number, in case the original number is a decimal. Line 7Ø gives the actual significant digits. Line 9Ø produces the corrected whole number, and line 11Ø gives the answer required:

```
   5 REM Correct to N sig.
         figures
  10 INPUT "Please enter your nu
mber ";X$
  20 PRINT '"Your number is ";X$
  30 LET Y=VAL X$*(10↑(LEN X$-1)
)
  40 PRINT '"Y = ";Y
  50 INPUT "To how many signific
ant","figures?  ";N
  60 PRINT '"I will correct it t
o ";N,"significant figures"
  70 LET Z=INT (VAL STR$ Y/10↑(L
EN STR$ Y-N)+.5)
  80 PRINT TAB 6;"Z = ";Z
  90 LET W=Z*(10↑(LEN STR$ Y-N))
 100 PRINT TAB 10;"Y = ";Y
 110 PRINT 'X$;" = ";W*VAL X$/Y;
" to ";N,"significant figures"
 120 GO TO 10
```

```
Your number is .054672

Y = 54672

I will correct it to 3
significant figures
        Z = 547
             Y = 54672
```



```
.054672 = .0547 to 3
significant figures

Your number is 674/45

Y = 1497777.8

I will correct it to 7
significant figures
        Z = 14978
             Y = 1497777.8

674/45 = 14.978 to 7
significant figures
```

Conversion to Standard Form

Finally, here is a program to convert numbers to standard form:

```
   5 REM CONVERSION TO
     STANDARD FORM
  10 INPUT "Enter your number ";
X$
  15 IF X$="S" OR X$="s" THEN ST
OP
  20 PRINT 'X$;" = ";
  30 LET Y=VAL X$*10↑(LEN X$-1)
  40 IF VAL X$)=10 THEN PRINT VA
L X$/10↑(LEN X$-1);"E+";LEN X$-1
  50 IF VAL X$<10 THEN PRINT Y/1
0↑(LEN STR$ Y-1);"E";LEN STR$ Y-
LEN X$
  60 GO TO 10
```

```
.00000345 = 3.45E-6

.0345 = 3.45E-2

.345 = 3.45E-1

345 = 3.45E+2

3450000000000000 = 3.45E+15
```

CHAPTER FIVE

Using the Spectrum for More Advanced Mathematics

The Spectrum seems almost purpose-built for maths work. Its wide range of mathematical functions, and the ease with which it can be programmed, invite students (and teachers) to create programs to solve particular problems, or demonstrate certain mathematical processes.

The Spectrum is ideal for situations where the basic working of a problem remains the same over a period of time, but the data which must be manipulated within that working procedures change. You should also find the computer excellent for producing visual demonstrations of maths in action.

In this chapter of the book, I'll look at a number of programs which I've included primarily to give you an idea of the range of mathematical processes which can be demonstrated on the Spectrum. Some of them may also be directly applicable to a course which you are taking.

From entering and running these programs, you should get an idea of just how flexible the Spectrum can be in this sort of work. In addition, you'll gain a number of ideas concerning effective visual displays which you can incorporate into the programs you write for maths and other subjects.

Number Series

The link between successive numbers in a series is often simple to state, but to calculate the series can be quite a time-consuming job, even if the mathematics involved in generating each number in the series is not very demanding. The Spectrum laps up work of this type, as you'll see from the following five programs.

The first one prints the series 1, 1*2, 2*3, 6*7 . . . and quickly approaches the Spectrum's upper numerical limit

(+ 7 * 10 ^ 38).

Here is the listing:

```
10 REM NUMBER SERIES
 (1, 1*2,2*3,6*7 etc)
20 LET A=1
30 PRINT A
40 LET A=A*(A+1)
50 GO TO 30
```

And this is what it looks like when running:

```
1
2
6
42
1806
3263442
1.06500657E+13
1.1342371E+26
```

Mystery Series

I'll leave you to work out the rule used for our second series. Here is the output of the program:

```
1
2
6
21
88
445
2676
18739
149920
1349289
13492900
1.4842191E+8
1.7810629E+9
2.3153818E+10
3.2415346E+11
4.8623018E+12
7.7796829E+13
1.3225461E+15
2.380583E+16
4.5231077E+17
9.0462153E+18
```

How did you get on? Here is the program listing:

```
20 LET A=1: PRINT A
30 FOR N=1 TO 20
40 LET A=N*(A+1)
50 PRINT A
60 NEXT N
```

Fibonnaci Numbers

Fibonnaci is the patron saint of rabbits. His name graces the number series produced by this brief program:

```
10 PRINT "FIBONNACI NUMBERS"
20 LET M=0
30 LET N=1
40 LET P=M+N
50 PRINT P
60 LET M=N
70 LET N=P
80 GO TO 40
```

The start of the Fibonnaci series:

```
FIBONNACI NUMBERS
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
10946
17711
28657
46368
75025
121393
196418
317811
514229
832040
1346269
2178309
3524578
5702887
9227465
14930352
24157817
39088169
63245986
1.02334416E+8
1.65580014E+8
2.679143E+8
4.3349444E+8
7.0140873E+8
```


Sum of Series

We now look at two programs to produce the sum of a series. The first one produces the sum of the series 1/X + 1/X↑2 + 1/X↑3 . . . .1/X↑N:

```
  10 REM SUM OF SERIES
1/X + 1/X↑2 + 1/X↑3....1/X↑N
  15 INPUT "Please enter value f
or X ";X
  20 LET S=0
  30 FOR N=1 TO 40
  40 LET S=S+1/(X↑N)
  50 PRINT S,
  60 NEXT N
```

Here is the series for X = 1.7:

```
0.58823529          0.93425606
1.1377977           1.257528
1.3279577           1.3693869
1.393757            1.4080923
1.4165249           1.4214852
1.4244031           1.4261195
1.4271291           1.427723
1.4280724           1.4282779
1.4283987           1.4284698
1.4285117           1.4285363
1.4285508           1.4285593
1.4285643           1.4285672
1.428569            1.42857
1.4285706           1.4285709
1.4285711           1.4285713
1.4285713           1.4285714
1.4285714           1.4285714
1.4285714           1.4285714
1.4285714           1.4285714
1.4285714           1.4285714
```

The second sum of series program sums the series 1↑2 + 2↑2 + 3↑2 . . . .N↑2:

```
  10 REM SUM OF SERIES
   1↑2 + 2↑2 + 3↑2...N↑2
  15 INPUT "Please enter value f
or N ";N
  20 LET X=0
  30 FOR M=1 TO N
  40 LET X=X+(M↑2)
  50 PRINT X
  60 NEXT M
```

Here it is in action for N = 16:

```
1
5
14
30
55
```

```
 91
 140
 204
 285
 385
 506
 650
 819
 1015
 1240
 1496
```

Square Roots and Sir Isaac

Initially, we'll look at three programs related to deducing square roots. The first one converges on the square root of two, using continued fractions, and the second program demonstrates the process. The third program, based on the first, is designed to determine the square root of N by continued fractions.

Give R some arbitrary value when running this program:

```
 20 INPUT "ENTER A NUMBER ";R
 30 PRINT "THE SQUARE ROOT OF 2
="
 40 PRINT TAB 23;R
 50 LET R=1+1/(1+R)
 60 GO TO 40
```

```
THE SQUARE ROOT OF 2 =
                        10000
                        1.0001
                        1.499975
                        1.400004
                        1.416666
                        1.4137932
                        1.4142857
                        1.4142012
                        1.4142157
                        1.4142132
                        1.4142136
                        1.4142136
                        1.4142136
                        1.4142136
```

This program, as you can see from the sample run which follows it, shows the process:

```
  5 REM CONTINUED FRACTIONS
 10 REM LET R=SQUARE ROOT OF 2
 20 REM R↑2-1=(R+1)(R-1)
 30 REM THUS (R+1)(R-1)=1
 40 REM SO R-1=1/(R+1)
 50 REM OR R=1+1/(1+R)
 60 PRINT AT 1,0;"R=1 +"
 70 FOR N=0 TO 18 STEP 2
 80 PRINT AT N,N+8;"1";AT N+1,N
+6;"-----";AT N+2,N+6;"2 +"
```



```
 90 NEXT N
100 PRINT AT 20,28;"etc."
```

```
          1
R=1 + -------
      2 + 1
          -------
        2 + 1
            -------
          2 + 1
              -------
            2 + 1
                -------
              2 + 1
                  -------
                2 + 1
                    -------
                  2 + 1
                      -------
                    2 + 1
                        -------
                      2 + 1
                          -------
                        2 + etc.
```

Square Roots by Continued Fractions

Our third program converges towards the square root of N using continued fractions. After the listing, you can see it in action working towards the square root of 4 (sample run one) and 9 (the second sample run):

```
  4 REM    SQUARE ROOT OF N
  5 REM  BY CONTINUED FRACTIONS
 10 INPUT "ENTER N ";N
 20 LET R=1
 30 LET R=1+(N-1)/(1+R)
 40 PRINT R
 50 GO TO 30
```

```
 2.5
 1.8571429
 2.05
 1.9836066
 2.0054945
 1.9981718
 2.0006098
 1.9997968
 2.0000677
 1.9999774
 2.0000075
 1.9999975
 2.0000008
 1.9999997
 2.0000001
 2
 2
```

```
5
2.3333333
3.4
2.8181818
3.0952381
2.9534884
3.0235294
2.9883041
3.0058651
2.9970717
3.0014652
2.9992677
3.0003662
2.9998169
3.0000916
2.9999542
3.0000229
2.9999886
3.0000057
2.9999971
2.9999993
3.0000004
2.9999998
3.0000001
3
3
```

Square Roots by Newton's Formula

In case you were wondering why Sir Isaac Newton gained a mention in our crossheading, here is the answer. Our next program is designed to find square roots by Newton's formula. After entering the number (N) you enter a guess (G) of the square root:

```
  10 REM TO FIND SQUARE ROOT BY
        NEWTON'S FORMULA
  20 INPUT "ENTER THE NUMBER ";N
  25 INPUT "YOUR NUMBER IS ";(N)
,"ENTER YOUR GUESS OF THE","SQUA
RE ROOT ";G
  30 LET S=(G+N/G)/2
  40 PRINT S,S↑2
  50 IF ABS (S-G)<S/1000000 THEN
 STOP
  60 LET G=S
  70 GO TO 30
```

Here is the program in action looking for the square root of 1000, with an entered guess of 17:

```
37.911765          1437.3019
32.144401          1033.2625
31.627009          1000.2677
31.622777          1000
31.622777
```


This time, I entered a guess of 1, to see what difference this would make:

```
500.5              250500.25
251.249            63126.06
127.61456          16285.475
67.725327          4586.72
41.245426          1701.1852
32.745269          1072.2527
31.642016          1001.2172
31.622782          1000.0004
31.622777          1000
```

It is often instructive, once you've got a program up and running correctly, to play around with the output of the program in an attempt to make it clear to the computer user what is going on. Working from the program listed above, I produced this following version of it, which gives a clear printout to show what is going on:

```
  10 PRINT "SQUARE ROOT BY
        NEWTON'S FORMULA"
  20 INPUT "ENTER THE NUMBER ";N
  25 INPUT "YOUR NUMBER IS ";(N)
,"ENTER YOUR GUESS OF THE","SQUA
RE ROOT ";G
  27 PRINT '"The number is ";N
  28 PRINT '"Your guess was ";G:
PRINT
  30 LET S=(G+N/G)/2
  40 PRINT S,S↑2
  50 IF ABS (S-G)<S/1000000 THEN
STOP
  60 LET G=S
  70 GO TO 30
```

Here is the output of one run. As you can see, it is instantly clear from the printout what is happening. As well, there is proper record of the values entered for N and G (a final improvement would be to have a line of the type 'The square root of . . . is . . .'):

```
SQUARE ROOT BY
NEWTON'S FORMULA

The number is 0.9894

Your guess was .03

16.505             272.41502
8.2824727          68.599355
4.2009649          17.648106
2.2182411          4.9205936
1.3321351          1.7745839
```

```
1.0374263          1.0762533
0.9955663          0.99115226
0.99468627         0.98940078
0.99468588         0.9894
```

Solving Equations by Newton's Method

The next section looks at programs which solve general equations using Newton's method. I never fail to be impressed by watching these programs in action, with the computer converging rapidly on the solution.

The first routine, written by Jeremy Ruston, expects the equation you want solved for X to be entered when prompted by line 1Ø. Then, in reply to the prompt from line 4Ø, you enter a starting position for the computer to work from. This should be (as in the square root examples given earlier) either an answer somewhere near what you believe the correct answer to be, or – if there is more than one correct answer – a number near the answer you are seeking. To try it out, enter X*X − 5 (to find the square root of five) or X ↑ 3 − 27.6 (to find the cube root of 27.6):

```
  5 REM Newton's method for
       solving equations
 10 REM By Jeremy Ruston
 20 INPUT "Enter a function in
terms of X ";F$
 30 PRINT F$
 40 INPUT "Enter a starting poi
nt ";S
 50 PRINT S
 70 INPUT "Enter maximum error
";ERR
 90 PRINT ERR
100 PRINT AT 10,10;S
110 LET X=S
120 IF ABS (VAL F$)<ERR THEN ST
OP
130 LET T=VAL F$
140 LET X=X + 0.00001
150 LET B=((VAL F$)-T)/0.00001
160 LET S=S-T/B
170 GO TO 100
```

The next version of this program is more flexible, allowing you to enter both sides of the equation, again with X as the unknown (note that the acceptable error – at the end of line 8Ø – is so small that some equations will not produce acceptable solutions; you may wish to increase this acceptable error to .ØØØ1):


```
  5 REM SOLVING EQUATIONS USING
       NEWTON'S FORMULA
 10 INPUT "ENTER LEFT HAND SIDE
OF","EQUATION ";L$
 20 INPUT "NOW PLEASE ENTER THE
RIGHT HAND SIDE OF YOUR EQUATIO
N ";R$
 30 PRINT AT 1,10;L$;"=";R$
 40 INPUT "ENTER A 'STARTING VA
LUE' ";S
 45 PRINT AT 5,11;".";AT 6,10;"
. . X="
 50 PRINT AT 6,16;S
 60 LET X=S
 70 LET F=VAL L$-VAL R$
 80 IF ABS F<.000001 THEN STOP
 90 LET X=X+1
100 LET S=S-F/(VAL L$-VAL R$-F)
110 GO TO 50
```

Here are three results of running the program:

```
X↑2-4*X=X↑6-44


. . X=1.8494857


X↑3=100


. . X=4.6415888


X↑2=100


. . X=10.000002
```

As I mentioned a little earlier, it is often a good idea to rewrite a program once you have it up and running to make the screen display as attractive and informative as possible. Another way of modifying a program is to rewrite it to dump to the printer, rather than the screen. To demonstrate this, I'll show you one way the above program could be rewritten to produce an informative output direct to the printer:

```
    5 REM SOLVING EQUATIONS USING
           NEWTON'S FORMULA
    6 REM ■ VERSION FOR PRINTER ■
   10 INPUT "ENTER LEFT HAND SIDE
 OF","EQUATION ";L$
   20 INPUT "NOW PLEASE ENTER THE
 RIGHT HAND SIDE OF YOUR EQUATIO
N ";R$
   30 LPRINT L$;"=";R$
   40 INPUT "ENTER A 'STARTING VA
LUE' ";S
   42 LPRINT : LPRINT "Starting v
alue = ";S
   50 LPRINT : LPRINT S
   60 LET X=S
   70 LET F=VAL L$-VAL R$
   80 IF ABS F<.0001 THEN GO TO 1
20
   90 LET X=X+1
  100 LET S=S-F/(VAL L$-VAL R$-F)
  110 GO TO 50
  120 LPRINT INVERSE 1;">>"; INVE
RSE 0;" X = ";S
```

Here is the program in action, looking for the fifth root of 1ØØØ:

```
X↑5=1000
Starting value = 4
4
3.9885769
3.984028
3.9822331
3.9815275
3.9812505
3.9811418
3.9810992
3.9810825
3.9810759
3.9810734
3.9810724
3.981072
3.9810718
3.9810717
>> X = 3.9810717
```


And again, solving a rather odd equation (to show it takes a lot to faze our Spectrum):

```
(2*PI/X)↑2=LN X
Starting value = 1
1
2.3028338
3.881935
4.9413868
4.9694722
4.9624762
4.9642559
4.9638055
>> X = 4.9638055
```

Here it is solving another odd equation, twice, with widely differing starting values (which shows that, in some cases, the starting value can be just about anything):

```
17*X-COS X/PI=5ØR X
Starting value = .02
.02
.027349178
.028491787
.028652158
.028674357
>> X = .028674357
17*X-COS X/PI=5ØR X
Starting value = 100
100
0.256304
.042804712
.030397919
.028911325
.028710073
```

```
.028682355
 X = .028682355
```

Finding Factors

Moving away from Sir Isaac, we'll now examine a number of different mathematical applications of the Spectrum. The first one, written by David Perry, finds factors. The program will prompt you, so you'll have no trouble using it:

```
  10 BORDER 1: PAPER 1: INK 7: B
RIGHT 8: CLS
  20 PRINT AT 0,7; PAPER 2;"  FA
CTORISING  "
  30 INPUT "Product? ";p: PRINT
AT 2,2; PAPER 0;"  Product:";p;"
 "
  40 INPUT "Sum? ";s: PRINT AT 4
,4; PAPER 0;"  Sum:";s;"  "
  50 FOR a=1 TO ABS (p)
  60 FOR b=ABS (p) TO 1 STEP -1
  70 IF (a*b)=ABS (p) THEN GO SU
B 120
  80 NEXT b
  90 NEXT a
 100 PRINT : PRINT "PRESS <ENTER
> TO RUN AGAIN"
 110 PAUSE 0: PAUSE 0: RUN
 120 IF (a*b)=p AND (a+b)=s THEN
 PRINT " Factors: +";a;",+";b
 130 IF ((-a)+b)=s AND ((-a)*b)=
p THEN PRINT " Factors: -";a;",+
";b
 140 IF ((-b)+a)=s AND ((-b)*a)=
p THEN PRINT " Factors: +";a;",-
";b
 150 IF ((-a)+(-b))=s AND ((-a)*
(-b))=p THEN PRINT " Factors: -"
;a;",-";b
 160 PRINT a,b
 170 RETURN
```

Calculating N! (factorial)

The next program calculates N! (factorial). The listing is followed by a few sample runs:

```
  10 REM TO PRINT N! (factorial)
  15 INPUT "Enter value for n ";
N
  20 PRINT N;" factorial ="
  30 LET X=1
  40 FOR P=1 TO N
  50 LET X=X*P
  60 NEXT P
  70 PRINT X
```



```
5 factorial =
120

12 factorial =
4.790016E+8


23 factorial =
2.5852017E+22

32 factorial =
2.6313084E+35
```

Related to the last program, this one prints factorials from 1 to M. A sample run follows the listing:

```
10 REM TO PRINT FACTORIALS
    FROM 1 TO M
15 INPUT "Please enter M ";M
20 FOR N=1 TO M
30 LET X=1
40 FOR P=1 TO N
50 LET X=X*P
60 NEXT P
70 PRINT N;" factorial = ";X
80 NEXT n
```

```
1 factorial = 1
2 factorial = 2
3 factorial = 6
4 factorial = 24
5 factorial = 120
6 factorial = 720
7 factorial = 5040
8 factorial = 40320
9 factorial = 362880
10 factorial = 3628800
11 factorial = 39916800
12 factorial = 4.790016E+8
13 factorial = 6.2270208E+9
14 factorial = 8.7178291E+10
15 factorial = 1.3076744E+12
16 factorial = 2.092279E+13
17 factorial = 3.5568743E+14
18 factorial = 6.4023737E+15
19 factorial = 1.216451E+17
20 factorial = 2.432902E+18
21 factorial = 5.1090942E+19
22 factorial = 1.1240007E+21
```

Solving Quadratic Equations

The resourceful Spectrum can solve quadratic equations more quickly than you can:

```
10 REM PROGRAM TO SOLVE
    QUADRATIC EQUATIONS
20 INPUT "ENTER A ";A
```

```
  30 INPUT "ENTER B ";B
  35 INPUT "ENTER C ";C
  37 PRINT "IF A = ";A;", B = ";
B;" and","C = ";C
  40 LET X1=(-B+SQR (B*B-4*A*C))
/(2*A)
  50 LET X2=(-B-SQR (B*B-4*A*C))
/(2*A)
  60 PRINT '"X = ";X1;" or ";X2


  IF A = 1, B = 9 and
  C = 2

  X = -0.22799813 or -8.7720019
```

Triangular Numbers

The next program generates Triangular numbers from one to 1ØØ as you can see from the output below the program listing:

```
  10 PRINT "TRIANGULAR NUMBERS"
  20 FOR N=1 TO 100
  30 PRINT N*(N+1)/2;" ";
  40 NEXT N
```

```
TRIANGULAR NUMBERS
1 3 6 10 15 21 28 36 45 55 66 78
 91 105 120 136 153 171 190 210
231 253 276 300 325 351 378 406
435 465 496 528 561 595 630 666
703 741 780 820 861 903 946 990
1035 1081 1128 1176 1225 1275 13
26 1378 1431 1485 1540 1596 1653
 1711 1770 1830 1891 1953 2016 2
080 2145 2211 2278 2346 2415 248
5 2556 2628 2701 2775 2850 2926
3003 3081 3160 3240 3321 3403 34
86 3570 3655 3741 3828 3916 4005
 4095 4186 4278 4371 4465 4560 4
656 4753 4851 4950 5050
```

If you prefer your output more neatly arranged, this second version may be of interest:

```
   5 REM ALTERNATIVE VERSION
     FOR OUTPUT IN COLUMNS
  10 PRINT "TRIANGULAR NUMBERS"
  20 FOR N=1 TO 21
  30 PRINT N;TAB 5;N*(N+1)/2;TAB
 10;(N+23)*(N+24)/2;TAB 15;(N+45
)*(N+46)/2;TAB 20;(N+67)*(N+68)/
2
  40 NEXT N
```



```
TRIANGULAR NUMBERS
1     1       300   1081  2346
2     3       325   1128  2415
3     6       351   1176  2485
4     10      378   1225  2556
5     15      406   1275  2628
6     21      435   1326  2701
7     28      465   1378  2775
8     36      496   1431  2850
9     45      528   1485  2926
10    55      561   1540  3003
11    66      595   1596  3081
12    78      630   1653  3160
13    91      666   1711  3240
14    105     703   1770  3321
15    120     741   1830  3403
16    136     780   1891  3486
17    153     820   1953  3570
```

Pascal's Triangle

Pascal's triangle can be printed by this program, in which line 3Ø gives the first number in each row, line 5Ø gives the rest of the numbers in the row, and line 6Ø is the 'recurrence' that must be found:

```
   5 REM Pascal's Triangle
  10 INPUT "How many rows? (<8)"
;M
  15 IF M>8 THEN LET M=8
  20 FOR N=0 TO M
  30 LET C=1
  40 PRINT AT 2*N,(15-2*N);C
  50 FOR R=1 TO N
  60 LET C=C*(N-R+1)/R
  70 PRINT AT N*2,(15-N*2+R*4);C
  80 NEXT R
  90 NEXT N
```

As you can see from these sample runs, the output of this program is limited by the screen width of the Spectrum display (you may wish to modify the program so it generates the numbers, but produces them in another form, so they can be 'manually' converted into a triangular display):

```
          1
        1   1
      1   2   1
    1   3   3   1
```

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5   10  10  5   1
    1   6   15  20  15  6   1
  1   7   21  35  35  21  7   1
1 8   28  56  70  56  28  8   1
```

Conversion Base 1Ø to Binary

This program is a demonstration of the conversion of base 1Ø numbers into binary. Read the remainders from bottom to top:

```
  10 INPUT "Enter your number in
 base 10 ";N
  20 PRINT "2 into ";N;" goes ";
INT (N/2);TAB 25;"Rem.: ";N-2*IN
T (N/2)
  30 LET N=INT (N/2)
  40 IF N>=1 THEN GO TO 20
```

This is the program in action:

```
2 into 16 goes 8          Rem.: 0
2 into 8 goes 4           Rem.: 0
2 into 4 goes 2           Rem.: 0
2 into 2 goes 1           Rem.: 0
2 into 1 goes 0           Rem.: 1


2 into 19 goes 9          Rem.: 1
2 into 9 goes 4           Rem.: 1
2 into 4 goes 2           Rem.: 0
2 into 2 goes 1           Rem.: 0
2 into 1 goes 0           Rem.: 1


2 into 94 goes 47         Rem.: 0
2 into 47 goes 23         Rem.: 1
2 into 23 goes 11         Rem.: 1
2 into 11 goes 5          Rem.: 1
2 into 5 goes 2           Rem.: 1
2 into 2 goes 1           Rem.: 0
2 into 1 goes 0           Rem.: 1
```


Our next listing converts base ten numbers into their binary equivalents:

```
  5 REM BASE 10 INTO BINARY
 10 INPUT "ENTER YOUR BASE 10 N
UMBER ";N
 20 PRINT N;" (base 10)"
 30 FOR j=0 TO INT (LN n/LN 2)
 40 PRINT AT 3,28-J;N-2*INT (N/
2);
 50 LET N=INT (N/2)
 60 NEXT J
```

Here is the program doing its work:

```
 123 (base 10) =          1111011

 3 (base 10) =                 11

 452 (base 10) =       111000100

 1234 (base 10) =   10011010010

 255 (base 10) =         11111111
```

Round to Nearest Whole Number

This short routine, as you can see from the sample run which follows the listing, rounds numbers to the nearest whole number:

```
  10 REM TO ROUND TO NEAREST
        WHOLE NUMBER
  20 INPUT "Please enter your nu
mber ";X
  30 PRINT X;" rounded to the ne
arest","whole number is ";INT (X
+.5000i)
  40 PRINT
  50 GO TO 20

12.3 rounded to the nearest
whole number is 12

12.75 rounded to the nearest
whole number is 13

0.6 rounded to the nearest
whole number is 1
```

```
0.3 rounded to the nearest
whole number is 0

0.5 rounded to the nearest
whole number is 1

.099 rounded to the nearest
whole number is 0
```

Express to N Decimal Places

This routine allows you to express a number to N decimal places:

```
   5 REM TO EXPRESS A NUMBER TO
        N DECIMAL PLACES
  10 INPUT "ENTER YOUR NUMBER ";
X
  20 INPUT "TO HOW MANY DECIMAL
PLACES ";N
  30 PRINT X;" to ";N;" decimal"
,"place(s) is ";
  40 PRINT INT (X*10↑N+.5)/10↑N

  24.689751 to 4 decimal
  place(s) is 24.6898


  .00034 to 4 decimal
  place(s) is .0003


  12399.99 to 0 decimal
  place(s) is 12400
```

Generating Prime Numbers

This program will generate as many prime numbers as you want:

```
   5 REM PRIME NUMBERS
  10 LET X=1: LET Y=2: LET D=3
  20 INPUT "HOW MANY PRIME NUMBE
RS DO" "YOU WANT? ";A
  40 PRINT "The first ";A;" prim
e numbers: "'1,2,3;
  60 FOR B=1 TO A-3
  70 LET D=D+Y
  80 LET C=Y+X
  90 LET E=INT (D/C)
 100 LET F=D-E*C
 110 IF NOT F THEN GO TO 70
 120 IF C>=E THEN GO TO 150
```



```
130 LET C=C+Y
140 GO TO 90
160 PRINT ,D;
170 NEXT B
```

The next program does much the same, this time using the Sieve of Eratosthenes, in a way which makes dramatic use of the visual output of the computer. It first prints up the numbers one to 1ØØ in this form:

```
 1  2  3  4  5  6  7  8  9  10
 11 12 13 14 15 16 17 18 19 20
 21 22 23 24 25 26 27 28 29 30
 31 32 33 34 35 36 37 38 39 40
 41 42 43 44 45 46 47 48 49 50
 51 52 53 54 55 56 57 58 59 60
 61 62 63 64 65 66 67 68 69 70
 71 72 73 74 75 76 77 78 79 80
 81 82 83 84 85 86 87 88 89 90
 91 92 93 94 95 96 97 98 99 100
```

Then, it works through them one by one, making each number flash briefly as it is checked to see if it is prime:

```
 1  2  3        5        7
 11       13              17       19
          23                       29
 31                       37
 41       43              47
          53              57 58 59 60
 61 62 63 64 65 66 67 68 69 70
 71 72 73 74 75 76 77 78 79 80
 81 82 83 84 85 86 87 88 89 90
 91 92 93 94 95 96 97 98 99 100
```

Any non-prime number is then erased, leaving this display of the prime numbers between 1 and 1ØØ:

```
1  2  3     5     7
11    13          17    19
      23                29
31                37
41    43          47
      53                59
61                67
71    73                79
      83                89
                  97
```

Here's the listing to work the magic:

```
  5 REM PRIME NUMBERS USING
      SIEVE OF ERATOSTHENES
 10 FOR R=0 TO 9
 20 FOR N=1 TO 10
 30 PRINT AT 2*R+3,3*N-1;10*R+N
 40 NEXT N
 50 NEXT R
 60 FOR R=0 TO 9
 70 FOR N=1 TO 10
 75 LET P=10*R+N: LET X=2*R+3:
LET Y=3*N-1
 80 FOR M=2 TO 7
100 IF P/M=INT (P/M) AND P<>2 A
ND P<>3 AND P<>5 AND P<>7 THEN P
RINT AT X,Y; INVERSE 1;P: PAUSE
10: PRINT AT X,Y; INVERSE 0;P;AT
 X,Y;"  "
110 NEXT M
120 NEXT N
130 NEXT R
```

Maths Tests: Decimals

We now have two complete programs, which contain full instructions within the listings, to generate maths tests. The first one, DECIMALS by Jim Walsh, allows you to specify the number of questions you wish to tackle, generates the questions and checks the answers, giving you a percentage score at the end. The program makes extremely effective use of such commands as BRIGHT to ensure the output is always lively:


```
  5 REM ..Decimals By Jim Walsh
 10 CLS : PRINT AT 11,3;"Decima
ls By Jim Walsh.";"'";"      Oct 19
82": PRINT AT 20,2;"Press any ke
y for prompts.": PAUSE 300
 15 GO SUB 1000
 50 LET l=10000: LET m=100: LET
 o=1000: LET go=go+1
 51 PAUSE 200: LET new=INT (RND
*4)+1: IF new=2 THEN GO SUB 300
 52 IF new=1 THEN GO SUB 100
 53 IF new=3 THEN GO SUB 400
 55 IF new=4 THEN GO SUB 500
 59 IF go=last THEN GO TO 2000
 60 FOR n=1 TO 100: NEXT n: CLS
: PRINT AT 10,5;INT (per/go*100
);"% Correct!!";AT 15,3;"Thats a
fter ";go;" sums.";AT 20,3;last-
go;" left to do.": PAUSE 100: GO
 TO 50
 99 REM ++++++ ADDITION ++++++
100 LET rnd=INT (RND*5)+2
105 DIM x(rnd): DIM y(rnd): DIM
z(rnd)
110 LET total=0: FOR n=1 TO rnd
: GO SUB 220: LET x(n)=tot: LET
total=total+tot: NEXT n: LET z$=
STR$ total
115 CLS : PRINT AT 0,27;"No.";g
o: PRINT AT 0,5;"Addition Of Dec
imals": PRINT OVER 1;AT 0,5;"
                ": PRINT AT 3,3
;"Add the following to 3 places"
: PRINT AT 5,0;" ";: FOR n=1 TO
rnd: PRINT x(n);: IF n<>rnd THEN
 PRINT " + ";
120 NEXT n: LET v$=STR$ total:
PRINT " = ??": INPUT "Type in yo
ur ans. "; LINE i$
125 IF i$=v$ THEN GO TO 180
129 REM wrong
130 CLS : PRINT AT 11,5;"SORRY
you are wrong"'"I will show y
ou the answer.": FOR k=1 TO 300:
 NEXT k: CLS : PRINT AT 5,0;" "
135 FOR n=1 TO rnd: LET p$=STR$
 INT x(n): PRINT TAB 10-LEN p$;x
(n): PAUSE 30: NEXT n: LET p$=ST
R$ INT total: PRINT TAB 9-LEN p$
;" ";: FOR n=1 TO LEN z$: PRINT
"-";: NEXT n: PAUSE 30: PRINT TA
B 10-LEN p$;total: PRINT TAB 9-L
EN p$;" ";: FOR n=1 TO LEN z$: P
RINT OVER 1;"-";: NEXT n: PRINT
"
140 FOR k=1 TO 500: NEXT k: PRI
NT AT 19,27;"O.K.": PAUSE 0: PRI
NT AT 21,3;"Hang on!!": RETURN
179 REM correct
180 LET per=per+1: CLS : PRINT
AT 11,4;"Well done!!!"'"
```

```
Bang On.": PAUSE 0: PRINT AT 21,
3;"Hang on!!": RETURN
 219 REM _____RANDOM NO._____
 220 LET a=INT (RND*l)+m
 225 LET b=INT (RND*o)+m: LET a$
=STR$ b: LET dec=b/10↑LEN a$: LE
T tot=a+dec
 226 IF b=10 OR b=100 OR b=1000
THEN GO TO 225
 230 RETURN
 299 REM --- SUBTRACTION---
 300 DIM x(2): FOR n=1 TO 2: GO
SUB 220: LET x(n)=tot: NEXT n: I
F x(1)<=x(2) THEN GO TO 300
 302 LET total=x(1)-x(2): IF tot
al<100 THEN GO TO 300
 305 CLS : PRINT AT 0,27;"No.";g
o: PRINT AT 0,0;"    Decimal Subt
raction";AT 0,3; OVER 1;"_______
_____"
 310 PRINT AT 11,3;x(1);" - ";x(
2);" = ??"
 315 INPUT "Type in the answer "
; LINE i$
 320 IF i$=STR$ total THEN GO TO
 380
 325 CLS : PRINT "WRONG!!!"''''"
    I will show the answer": PAU
SE 250: LET z$=STR$ INT x(1): LE
T y$=STR$ INT x(2): LET w$=STR$
INT total
 330 PRINT '''';TAB 10-INT LEN z$
;x(1)
 331 PAUSE 30
 332 PRINT TAB 10-INT LEN y$;x(2
);" -": PAUSE 30
 334 PRINT TAB 9-INT LEN w$;" ";
: LET v$=STR$ total: FOR n=1 TO
LEN v$: PRINT "_";: NEXT n: PRIN
T " "
 336 PRINT TAB 10-INT LEN w$;tot
al: PRINT TAB 9-INT LEN w$;" ";:
 FOR n=1 TO LEN v$: PRINT "_";:
NEXT n: PRINT " "
 340 FOR k=1 TO 400: NEXT k: PRI
NT AT 20,25;"O.K.": PAUSE 0: PRI
NT AT 21,3;"Hang on!!"
 345 RETURN
 380 LET per=per+1: CLS : PRINT
AT 11,0;"Well done"';'"     CORR
ECT!!": PAUSE 0: PRINT AT 21,3;"
Hang on!!": RETURN
 399 REM XXX MULTIPLICATION XXX
 400 DIM i(2): DIM x(2): DIM d(2
): LET len=0: LET l=100: LET m=1
: LET o=100
 405 FOR n=1 TO 2: GO SUB 220: L
ET i(n)=a: LET x(n)=tot: LET d(n
)=VAL a$: LET len=len+LEN a$: NE
XT n
 410 LET total=x(1)*x(2)
```



```
 415 CLS : PRINT AT 0,27;"No.";g
o: PRINT AT 0,0;"    MULTIPLICA
TION": PRINT AT 0,0; OVER 1;"
                "
 420 PRINT AT 5,0;"What is -:"'''
TAB 5;x(1);" X ";x(2);" = ??"'''
"Type in the answer.": INPUT LIN
E s$
 425 IF s$=STR$ total THEN GO TO
 490
 427 LET u$="I WILL SHOW YOU THE
 ANSWER": PRINT '''"WRONG!!!": PAU
SE 100: PRINT '''; FOR N=1 TO
 LEN U$: PRINT U$(N TO N);: PAUS
E 10: NEXT N: PAUSE 100: CLS
 430 PRINT I(1);".";  BRIGHT 1;D(
1): PRINT I(2);".";  BRIGHT 1;D(2
); BRIGHT 0;" X": FOR N=1 TO 6:
PRINT " ";: NEXT N: PRINT " ": P
AUSE 100: PRINT AT 2,10;"=": PAU
SE 50: PRINT AT 2,12;LEN;" decim
al places.": PAUSE 200: PRINT AT
 2,12;"                "
 432 LET m$=STR$ i(1)+STR$ d(1):
 LET n$=STR$ i(2)+STR$ d(2): PRI
NT TAB 20-LEN m$;m$: PRINT TAB 2
0-LEN n$;n$;" X": PRINT TAB 19-L
EN n$;"";: FOR n=1 TO LEN n$: P
AUSE 5: PRINT "_";: NEXT n: LET
p=0: PRINT " ": FOR n=LEN n$ TO
1 STEP -1: PAUSE 20: LET ans=VAL
 m$*VAL n$(n TO n)*10↑p: LET b$=
STR$ ans: PRINT TAB 20-LEN b$;b$
: LET p=p+1: NEXT n
 435 LET ans=VAL m$*VAL n$: LET
b$=STR$ ans: PRINT TAB 19-LEN b$
;" ";: FOR n=1 TO LEN b$: PAUSE
5: PRINT " ";: NEXT n: PRINT "
": PRINT TAB 20-LEN b$;b$: PRINT
TAB 19-LEN b$;" ";: FOR n=1 TO L
EN b$: PAUSE 5: PRINT "_";: NEXT
 n: PRINT " "
 440 PAUSE 100: PRINT AT 21,0;"
  =  ";total;"  ans.": FOR k=1
TO 250: NEXT k: PRINT AT 21,28;"
O.K.": PAUSE 0: PRINT AT 18,28;"
Hang": PRINT AT 19,28;"On!": RET
URN
 490 LET per=per+1: PRINT '"Well
 done CORRECT!": PAUSE 0: PRINT
'"Hang on!!": RETURN
 499 REM A A A DIVISION A A A
 500 REM
 505 GO SUB 220: LET l1=LEN a$::
LET b$=STR$ tot
 510 LET l=100: LET m=1: LET o=1
00: GO SUB 220: LET l2=LEN a$: L
ET c$=STR$ tot: LET l2=LEN a$: L
ET d$=STR$ (VAL b$/VAL c$)
 515 LET d=VAL d$: LET d1=INT (d
): LET d2=d-d1: LET e$=STR$ d2
```

```
 520 LET d3=VAL e$(6 TO 6): LET
d4=VAL e$(5 TO 5): IF d3>4 THEN
LET d4=d4+1
 525 LET e$(5 TO 5)=STR$ d4: LET
 e$=e$(1 TO 5): LET d1=d1+VAL e$
: LET f$=STR$ d1
 530 CLS : PRINT AT 0,0;"   DIVI
SION";AT 0,0; OVER 1;"
_";AT 0,27;"No.";go
 535 PRINT AT 4,3;"What is ";b$;
" A ";c$;" =?";AT 15,0;"Type in
 your answer correct ";"to thr
ee decimal places": INPUT LINE i
$
 540 IF i$=f$ THEN GO TO 490
 545 CLS : PRINT "WRONG!!": PAUS
E 100: PRINT AT 12,0;"I will sho
w you how to do it.": FOR n=1 TO
 200: NEXT n
 550 CLS : PRINT AT 0,0;c$;"B";b
$;: PAUSE 15: PRINT AT 0,LEN c$;
 OVER 1;" ";: FOR n=1 TO LEN b$:
 PAUSE 15: PRINT OVER 1;"C";: NE
XT n: PRINT " = "
 551 IF l1>l2 THEN LET l3=l1
 552 IF l2>l1 THEN LET l3=l2
 553 IF l1=l2 THEN LET l3=l1
 555 LET x=VAL c$*10↑l3: LET g$=
STR$ x: LET x=VAL b$*10↑l3: LET
h$=STR$ x
 559 PAUSE 100
 560 PRINT AT 0,LEN b$+LEN c$+4;
g$;"B";h$;AT 0,LEN b$+LEN c$+LEN
 g$+4; OVER 1;" ";: FOR n=1 TO L
EN h$: PAUSE 15: PRINT OVER 1;"C
";: NEXT n: PRINT " ="
 565 LET d=INT VAL d$: LET d1=VA
L d$-d: LET t$=STR$ d1: LET d$=S
TR$ d+"."+t$(3 TO 6)
 570 LET t=INT VAL d$: LET t$=ST
R$ t: PRINT AT 3,0;TAB 18-LEN t$
;d$
 580 PRINT AT 4,0;TAB 18-LEN h$;
h$
 585 PRINT AT 4,0;TAB 17-LEN h$;
"B";
 590 FOR n=1 TO LEN h$+LEN d$: P
AUSE 10: PRINT OVER 1;"C";: NEXT
 n
 595 PRINT AT 4,0;TAB 17-LEN h$-
LEN g$;g$
 600 PRINT AT 7,10;"=";AT 10,14;
" ";: FOR n=1 TO LEN f$: PRINT f
$(n TO n);: PAUSE 30: NEXT n: PR
INT "   ans.": PAUSE 300: PRINT
AT 21,27;"O.K."
 605 PAUSE 0: RETURN
1000 DATA 24,24,0,255,255,0,24,2
4
1001 DATA 31,16,8,8,8,8,8,16
1002 DATA 255,0,0,0,0,0,0,0
1005 RESTORE : FOR n=0 TO 23: RE
```



```
AD a: POKE 32600+n,a: NEXT n: LE
T per=0: LET go=0
1010 CLS : PRINT AT 5,2;"How man
y sums"''"do you want to do?";AT
 21,2;"Enter number": INPUT LINE
 i$: FOR n=1 TO LEN i$: LET a=CO
DE i$(n TO n): IF a<48 OR a>57 T
HEN GO TO 1010
1015 NEXT n: LET last=VAL i$: CL
S : PRINT "HANG ON!!": RETURN
2000 CLS : PRINT TAB 8;"Thats it
!!";AT 5,3;"Out of ";last;" sums
"''"     you got ";INT (per/last
*100);"%": PAUSE 200: PRINT AT 2
1,25;"O.K.": PAUSE 0: RUN
```

Gwyn Dewey's program – MATHS – is more of a game, but one which has considerable value in developing numerical skills. The game is designed for four players. There are ten tests for each player, and the winner is he or she who does best out of their round. The program uses the players' names, and prints the results of the test in a particularly effective way.

You'll see (line 5Ø6) that there are ten possible kinds of tests which can be produced by the program:

```
  1 REM @ 24/7/82 G.Dewey Maths
  2 LET d$="Maths"
  3 GO SUB 8000
  4 CLS
  5 DIM a(4)
  6 DIM c$(4,10)
  7 DIM a$(4,15)
 10 PRINT "Maths"
 20 PRINT '"for 4 players"
 30 PRINT '"there are 10 tests
to do of ten questions and a tab
le is shown  of questions correc
t"
 40 PRINT '"each person takes a
 test until  all have been done
then the     winner is found"
 50 PRINT ' FLASH 1;"Press to s
tart"
 60 IF INKEY$="" THEN GO TO 60
 70 CLS
 80 PRINT "Enter names"
 90 FOR i=1 TO 4
100 INPUT "Player ";(i);"?"; LI
NE a$(i)
110 NEXT i
130 CLS
135 IF c$(1,10)<>" " AND c$(2,1
0)<>" " AND c$(3,10)<>" " AND c$
(4,10)<>" " THEN GO TO 2000
140 PRINT "Maths"''"Menu card"
145 LET m=96+i*16
```

```
 150 PRINT '"1=Goto work card"
 160 PRINT "2=Print percentage"
 170 PRINT "3=Print Graph"
 175 PRINT "4=quit"
 180 INPUT "Which choice?"; LINE
 e$: IF e$<"1" OR e$>"4" THEN GO
 TO 180
 190 CLS
 191 IF e$="4" THEN GO TO 2000
 200 GO TO VAL e$*500
 500 INPUT "player(1-4)?"; LINE
s$: IF s$<"1" OR s$>"4" THEN GO
TO 500
 505 LET f=VAL s$
 506 PRINT "0=easy addition"'"1=
average addition"'"2=hard additi
on"'"3=easy subtraction"'"4=aver
age subtraction"'"5=hard subtrac
tion"'"6=easy multiplication"'"7
=hard multiplication"'"8=easy di
vision}round to nearest"'"9=hard
 division}whole number."
 510 INPUT "which work card(0-9)
?"; LINE g$: IF g$="q" THEN GO T
O 130
 511 IF LEN g$>1 THEN GO TO 510
 512 IF g$<"0" OR g$>"9" THEN GO
 TO 510
 515 FOR i=1 TO 10: IF c$(f,i)=g
$ THEN GO TO 130
 516 NEXT i
 520 LET b$="+++---**//"
 530 FOR i=1 TO 10: IF c$(f,i)<>
" " THEN NEXT i
 540 LET c$(f,i)=g$
 570 LET z=VAL g$+1
 580 IF z=1 OR z=4 OR z=7 THEN L
ET h=1
 590 IF z=2 OR z=5 OR z=8 OR z=9
 THEN LET h=2
 600 IF z=3 OR z=6 OR z=10 THEN
LET h=3
 610 CLS
 620 PRINT "Work card ";VAL g$
 630 PRINT ' FLASH 1;"press to s
tart"
 640 IF INKEY$="" THEN GO TO 640
 641 LET j=0
 645 FOR i=1 TO 10
 650 CLS
 655 PRINT "Question ";i
 656 PRINT j;" out of ";i-1
 657 PRINT "Player:";a$(f)
 660 LET a=INT (RND*10↑h)
 670 LET b=INT (RND*10↑h)


 675 IF a<b THEN IF z=4 OR z=5 O
R z=6 THEN LET y=a: LET a=b: LET
 b=y
 680 IF z=9 OR z=10 THEN LET b=I
NT (RND*10↑(h-1))
```



```
 681 IF b=0 THEN IF z=9 OR z=10
THEN GO TO 680
 690 LET z$=STR$ a+b$(VAL g$+1)+
STR$ b
 695 LET z$=STR$ (INT ((VAL z$)+
.5))
 696 GO TO 700
 697 PRINT AT 21,0;"Enter answer
 as a number please"
 700 INPUT "What is ";(a);(b$(VA
L g$+1));(b);"?"; LINE x$
 702 IF x$="" THEN GO TO 700
 705 FOR n=1 TO LEN x$: IF x$(n)
<"0" OR x$(n)>"9" THEN GO TO 697
:
 706 NEXT n
 707 PRINT AT 21,0;"
                    "
 710 LET k=VAL z$
 720 IF VAL x$=k THEN PRINT AT 2
1,0;"Right.": LET j=j+1
 730 IF VAL x$<>k THEN PRINT AT
21,0;"Wrong,The answer is ";k
 731 FOR l=1 TO 250
 732 NEXT l
 740 NEXT i
 750 LET a(f)=a(f)+j
 760 GO TO 130
1010 PRINT "-----------"
1020 FOR i=1 TO 4
1030 PRINT 'a$(i);" ";a(i);"%"
1040 NEXT i
1050 PRINT ' FLASH 1;"Press q to
 quit"
1060 IF INKEY$<>"q" AND INKEY$<>
"<=" AND INKEY$<>"0" THEN GO TO
1060
1070 GO TO 130
1500 FOR i=1 TO 4
1510 PRINT a$(i);"cards:";c$(i)
1520 PRINT PAPER 2;"
                    "
1530 NEXT i
1540 FOR i=1 TO 4
1545 LET m=176-i*16
1546 LET v=a(i)
1550 IF a(i)=0 THEN NEXT i
1560 FOR j=0 TO v*2-1
1565 FOR o=m TO m+7
1570 PLOT INK 4;j,o
1580 NEXT o
1590 NEXT j
1600 NEXT i
1610 PRINT AT 21,0; FLASH 1;"Pre
ss q to quit"
1620 IF INKEY$<>"q" AND INKEY$<>
"<=" AND INKEY$<>"0" THEN GO TO
1620
1630 GO TO 130
2000 CLS : PRINT "all finished"
2010 FOR i=1 TO 4
2020 IF a(i)>=a(1) AND a(i)>=a(2
```

```
) AND a(i)>=a(3) AND a(i)>=a(4)
THEN PRINT "Winner: ";a$(i)
2030 NEXT i
2040 GO TO 9000
8000 BORDER 7: PAPER 7: INK 0: C
LS
8001 FOR g=1 TO LEN d$
8005 LET f=USR "a"
8010 LET e=((CODE d$(g))-32)*8+1
5616
8020 FOR y=e TO e+7
8030 FOR z=1 TO 2
8040 POKE f,PEEK y
8050 LET f=f+1
8060 NEXT z
8070 NEXT y
8080 PRINT AT 9,g;"a"
8090 PRINT AT 10,g;"a"
8100 NEXT g
8200 PAUSE 300
8400 RETURN
9910 INPUT "Do you want to go to
 the next   program(y/n)?";z$
9920 IF z$="n" THEN RUN
9925 PRINT "Press play on tape r
ecorder"
9930 LOAD ""
```

Maths Tests: Correlation/Regression

Finally, in this chapter, we have the program CORETS (Correlation/Regression) by Paul Toland. Although you may not have a need for the program in this form in your lessons, it is included as it is a very good example of how a known mathematical process can be converted into a program. Regression is the process of fitting a best straight line through several points on a graph. This straight line will take an average path between all the points. If this line is then extended past the end of the known data, a forecast is made of the next, as yet unknown, figures.

Depending on the data given, some regression lines are a more accurate reflection of the original figures than others. The closeness of the 'fit' of a line is measured by the correlation coefficients.

When you run the program, you're asked if you will want the results sent to the printer in the course of the program. If you answer 'Y', you'll be given the opportunity to COPY the screen at all important points in the program. If 'N' is entered, then the question "PRINT THIS PAGE?" will never be asked.

Next you must enter the data. This is done in two stages,

with the X values first and then the Y values. The X values are independent figures. For example, for yearly sales' totals, the X data might be 1984, 1985, 1987 etc. It is possible to use other equivalent values without affecting the results, such as 1, 2, 4. You choose the base value (1 in this case) and then use similar increments as appear in the original data.

When entering data for time series calculations (time series calculations are used to take account of the fluctuations in data which are caused by seasonal factors; they are only useful for applications in which the data consists of a repeated pattern), it becomes essential to use this alternative notation for the X values. For example, for quarterly sales figures:

Year Quarter
'84 1 you must enter 1
2 you must enter 2
3 you must enter 3
4 you must enter 4
'85 1 you must enter 5
2 you must enter 6 . . . . and so on

Note that for time series calculations, missing seasons are not allowed. Therefore, in the above example, the quarters cannot be 1, 2, 4, although an entire missing period should not affect the results too adversely.

Once all the X values have been entered, enter 9999 to stop. The Y data are now entered in their original form. They are called the dependent data and might consist of, for example, the sales figures and production totals. A maximum of 1Ø sets of figures has been placed on the program.

Once all the data are entered, the program prints the correlation/regression results. The important ones are Pearson's coefficient, the coefficient of determination and the equation of the regression line.

You are then given the chance to interpolate or extrapolate on the regression line. You enter a value of X in the same notation as originally used when entering the X data, and you are given the value of Y at that point on the line. This can be used for forecasting. For example, if you have entered the sales for years 1 to 1Ø, then entering 11 will give you a forecast of the next year's sales figures.

The program then plots a graph with the original data and

the regression line superimposed.

After the graph has been drawn, you are asked if you wish to continue into the time series part of the program. It only makes sense to do so if the data is of a seasonal nature. If you decide to continue, you must enter the length of the period. That is, the number of values that make up one period (such as 4 for quarterly sales figures).

Next you enter the type of model you want to use, either multiplicative or additive (M or A). Once this is entered, the program plots in the graph of the moving averages of the data. As you will see, this varies throughout the data and so is more flexible. A freak value near the start of the data will not affect the end of the line.

The seasonal factors are then printed followed by the original data in seasonally adjusted form. Lastly, the program prints the forecast figures for the next two periods, before asking if you want a re-run of the program.

After all that, here is the listing (the background to the maths involved in the program is given after the listing, just in case you'd like to refresh your mind on it):

```
    2 REM CORETS © P.TOLAND.
    3 REM
    4 REM CAPS-ONLY FUNCTION.
    5 DEF FN Z$(X$)=(CHR$ (CODE (
X$)-32*(CODE X$>90)))
    6 REM ↑ CAPS-ONLY FUNCTION.
   10 PRINT "CORETS.- CORRELATION
/REGRESSION/          TIME SERIES
 PROGRAM."
   20 INPUT "WILL YOU WANT TO USE
 THE PRINTER";A$
   30 LET PCON=0
   40 IF FN Z$(A$)="Y" THEN LET P
CON=1
   50 GO TO 100
   60 IF NOT PCON THEN INPUT "Pre
ss ENTER";A$: RETURN
   70 INPUT "PRINT THIS PAGE ?";A
$
   80 IF FN Z$(A$)="Y" THEN COPY
: LPRINT "----------------------
----------"
   90 RETURN
  100 DIM X(100): DIM Y(100)
  110 PRINT "Enter X values in or
der -          End with 9999"
  120 FOR I=1 TO 100
  130 INPUT X
  140 IF X=9999 THEN GO TO 180
  150 PRINT X: LET X(I)=X
  160 NEXT I
```



```
  170 PRINT "100 IS MAXIMUM ALLOW
ED."
  180 LET I=I-1: CLS
  190 PRINT "          ";I;" VALUES    "
'"Enter Y values in order "
  195 LET SM=1E38: LET LA=4*1E-39
  200 FOR J=1 TO I
  205 PRINT X(J);
  210 INPUT Y(J): PRINT TAB 9,Y(J
)
  214 IF Y(J)>LA THEN LET LA=Y(J)
  216 IF Y(J)<SM THEN LET SM=Y(J)
  220 IF PEEK 23689=2 THEN GO SUB
 60: CLS
  225 NEXT J
  230 GO SUB 60
  240 LET SX=0: LET SY=0
  250 LET SXS=0: LET SYS=0
  260 LET SXY=0
  270 FOR J=1 TO I
  280 LET SX=SX+X(J)
  290 LET SY=SY+Y(J)
  300 LET SXS=SXS+X(J)*X(J)
  310 LET SYS=SYS+Y(J)*Y(J)
  320 LET SXY=SXY+X(J)*Y(J)
  330 NEXT J
  340 LET R=(I*SXY-SX*SY)/SQR ((I
*SXS-SX*SX)*(I*SYS-SY*SY))
  345 CLS
  350 PRINT "SUM X:";SX'"SUM Y:";
SY'"SUM X SQ.:";SXS'"SUM Y SQ.:"
;SYS'"SUM XY:";SXY
  360 PRINT '"PEARSONS CORRELATIO
N COEFFICIENT    ";R
  370 PRINT '"COEFFICIENT OF DET
ERMINATION       ";R*R
  380 LET B=(I*SXY-SX*SY)/(I*SXS-
SX*SX)
  390 LET A=SY/I-SX/I*B
  400 PRINT '"THE LINEAR REGRESS
ION EQUATION:"''"Y=";A;(" + " AND
B>=0);B;" X"
  410 GO SUB 60
  430 CLS
  440 PRINT "INTERPOLATION/EXTRAP
OLATION          9999 TO END"'
"     X          Y"
  450 INPUT "ENTER X VALUE ";X
  460 IF X=9999 THEN GO TO 495
  470 LET Y=A+B*X
  480 PRINT X;TAB 8;Y
  485 IF PEEK 23689=2 THEN GO SUB
 60: CLS
  490 GO TO 450
  495 GO SUB 60
  500 LET SL=A+B*X(1): LET EL=A+B
*X(I)
  502 IF SL>LA THEN LET LA=SL
  504 IF EL>LA THEN LET LA=EL
  506 IF SL<SM THEN LET SM=SL
  508 IF EL<SM THEN LET SM=EL
  510 LET SCX=253/(X(I)-X(1))
```

```
  520 LET SCY=173/(LA-SM)
  525 CLS
  530 FOR J=1 TO I
  540 CIRCLE 1+(X(J)-X(1))*SCX,1+
(Y(J)-SM)*SCY,1
  550 NEXT J
  560 PLOT 0,(SL-SM)*SCY
  570 DRAW 255,(EL-SM)*SCY-(SL-SM
)*SCY
  580 GO SUB 60
  590 INPUT "CONTINUE INTO TIME S
ERIES ?";A$
  600 IF FN Z$(A$)="N" THEN GO TO
 900
  610 INPUT "LENGTH OF PERIOD ?";
PER
  620 IF PER(=0 OR PER(>INT PER T
HEN GO TO 610
  630 INPUT "MULTIPLICATIVE OR AD
DITIVE MODEL (M OR A)?";M$
  640 LET M$=FN Z$(M$)
  650 IF M$(>"A" AND M$(>"M" THEN
 GO TO 630
  660 DIM T(PER): DIM A(I)
  670 LET IP=INT (I/PER)*PER
  680 LET END=IP-PER+1
  690 LET DIV=PER
  700 FOR J=1 TO END
  710 FOR K=J TO J+PER-1
  720 LET A(J)=A(J)+Y(K)
  730 NEXT K
  740 NEXT J
  750 LET ODD=(PER<>INT (PER/2)*2
)
  760 IF ODD THEN GO TO 820
  770 LET END=END-1
  780 FOR J=1 TO END
  790 LET A(J)=A(J)+A(J+1)
  800 NEXT J
  810 LET DIV=DIV*2
  830 FOR J=1 TO END
  840 LET A(J)=A(J)/DIV
  850 NEXT J
  860 LET STRT=INT (PER/2)+1
  870 LET X1=(X(STRT)-X(1))*SCX
  880 LET Y1=(A(1)-SM)*SCY
  890 PLOT X1,Y1
  900 FOR J=2 TO END
  910 LET X2=(X(STRT-1+J)-X(1))*S
CX
  920 LET Y2=(A(J)-SM)*SCY
  930 DRAW X2-X1,Y2-Y1
  940 LET X1=X2: LET Y1=Y2
  950 NEXT J
  960 GO SUB 60
  970 FOR J=1 TO END
  980 LET CP=STRT-1+J
  990 LET TP=(CP/PER-INT (CP/PER)
)*PER
  995 LET TP=TP+PER*(TP=0)
 1000 LET SALE=Y(CP)
 1010 LET MA=A(J)
```



```
 1020 LET T=SALE-MA
 1030 IF M$="M" THEN LET T=SALE/M
A
 1040 LET T(TP)=T(TP)+T
 1050 NEXT J
 1060 LET TOT=0
 1070 IF M$="M" THEN LET TOT=END
 1080 LET TTL=0
 1090 FOR J=1 TO PER
 1100 LET TTL=TTL+T(J)
 1110 NEXT J
 1120 LET ADJ=(TTL-TOT)/PER
 1125 PRINT "THE ";PER;" SEASONAL
 FACTORS ARE:"
 1130 FOR J=1 TO PER
 1140 LET T(J)=(T(J)-ADJ)/(END/PE
R)
 1150 PRINT J;"___";T(J)
 1160 NEXT J
 1170 PRINT '"SEASONALLY ADJUSTED
 FIGURES"'"X         Y         ADJUSTED
 Y"
 1175 LET TP=0
 1180 FOR J=1 TO I
 1190 LET TP=TP+1
 1200 LET SY=Y(J)-T(TP)
 1210 IF M$="M" THEN LET SY=Y(J)/
T(TP)
 1220 PRINT X(J);TAB 6;Y(J);TAB 1
6;SY
 1230 IF TP=PER THEN LET TP=0
 1240 IF PEEK 23689=2 THEN GO SUB
 60: CLS
 1250 NEXT J
 1260 GO SUB 60
 1270 CLS
 1280 PRINT "FORCAST FOR THE NEXT
 TWO PERIODS X    PERIOD   FORCAST
 Y"
 1290 LET INCX=X(I)-X(I-1)
 1300 LET TP=0
 1310 LET INCY=(A(END)-A(END-2))/
2
 1320 LET STX=X(I)+INCX
 1330 LET STY=A(END)+INCY
 1340 FOR J=END+STRT-1 TO I
 1350 LET STY=STY+INCY
 1360 NEXT J
 1365 LET TP=TP+I-IP
 1370 FOR J=STX TO STX+PER*2*INCX
-INCX STEP INCX
 1380 LET TP=TP+1
 1390 LET FY=STY+T(TP)
 1400 IF M$="M" THEN LET FY=STY*T
(TP)
 1410 PRINT J;TAB 6;TP;TAB 13;FY
 1420 LET STY=STY+INCY
 1430 IF TP=PER THEN LET TP=0
 1440 NEXT J
 1450 GO SUB 60
 1460 PRINT ' FLASH 1;"          CO
RETS COMPLETE"
```

```
1470 INPUT "DO YOU WANT TO RERUN
 THE PROGRAM ? (Y OR N)";A$
1480 IF FN Z$(A$)="Y" THEN RUN
```

As I said at the start of the introductory notes to this section, regression is the process of fitting a best straight line through several points on a graph. The straight line will take an average path between all the points. If this line is then extended past the end of the known data, a forecast is made of the next – as yet unknown – figures. Depending on the data given, some regression lines are a more accurate reflection of the original figures than others. The closeness of the 'fit' of a line is measured by the correlation coefficients.

Pearson's correlation coefficient, symbolised by the letter R, gives the strength of the linear correlation. The values of this coefficient lie in the range −1 to 1. A value close to either −1 or 1 means a high correlation, a good fit. Values close to ∅ mean a low correlation, a bad fit. Positive values indicate a line which is rising, and negative values indicate a falling trend. Among other things, a high correlation means – naturally enough – that any forecasting based on the observed data is likely to be reasonably accurate.

The coefficient of determination is very nearly the same as Pearson's; it is R squared. It falls in the range ∅ to 1. If R = ∅.8 then R ↑ 2 = ∅.64. This would mean that 64% of the variation in the observed values of Y is explained by the model.

Of course any calculations based on only a few figures will be meaningless. Several sets of data are needed to give some indication of the general trend.

Time Series

Time series calculations are used to take account of the fluctuations in the data which are caused by seasonal factors. Therefore, they are only useful for applications in which the data consist of a repeated pattern. Such patterns could be quarterly sales figures (each year would have four numbers attached) or daily production totals (which could have five, six or seven figures per week). In this last example, the *period* would be the week, and its *length* would be five (or six or seven).

The relationship between the size of a particular figure

and the season in which it falls can be calculated and is called the seasonal factor. By the way, a season in this context refers to the part of the period in which the number lies, and can refer to the day of the week or the quarter of the year.

When the data are affected by seasonal variation, the figures in one period cannot usefully be compared with each other. For example, if you sold toys, then you'd probably expect your Christmas sales to be better than your summer ones. If in one year you have an unusually quiet Christmas, and a strong summer, then comparing the two sales figures would not reveal this, since the Christmas figure would still be higher than the summer one.

If, however, you can remove the seasonal factor from the figures, the true situation can be assessed. In a term familiar to us from government pronouncements on unemployment, modifying the figures in this way is called *seasonal adjustment*. Forecasting from Time Series can lead to more satisfactory results as the figures produced will also follow the seasonal pattern.

Note that there are two types of time series model: Multiplicative and Additive. Multiplicative is used when the deviations from the trend per season vary greatly from period to period. The Additive model is used when there is a more or less constant seasonal variation.

Whatever the application, the results from any forecasting made using either the linear regression or the time series methods cannot be taken in isolation. Sales might fall because of a recent price rise, production may increase because of newly-installed automatic machinery. These events will affect the figures, but the computer cannot possibly know about them. You must therefore view the forecasts made by the Spectrum in the light of any other information you may have which is not reflected in the data you have used.

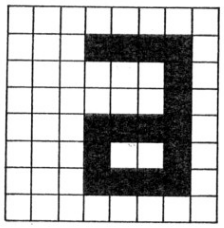
CHAPTER SIX

Using the Spectrum graphics effectively Part One

One of the great advantages of the Spectrum over many computers is that you can define your own graphics. This is of tremendous value in such subjects as languages, chemistry, geography and music.

The Spectrum graphics are formed on an eight by eight grid (of 'pixels'), with certain of the pixels being black, and others white. The patterns formed by the black and white pixels on the grid are actually the letters, numbers and so on you see on the TV screen. The Spectrum allows you to make your own patterns, to form musical notes, letters with accents, and any special symbols you need.

The information is stored as a series of zeroes (pixel turned off, or white) and ones (pixel turned on, or black). Here, to try and make that clear, is an eight by eight grid. On it is a form of the letter 'a', and beside it is the 'bit pattern' of ones and zeroes which forms the letter:



corresponding bit pattern

0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 1 1 1 1 0
0 0 0 1 0 0 1 0
0 0 0 1 1 1 1 0
0 0 0 0 0 0 0 0

As you can see, each of the bit patterns is actually a binary number. The computer has been programmed so that by giving it eight binary numbers where the ones correspond to the areas we want to have black in the character we're creating, and the zeroes correspond to the white spaces, we can form a letter.

Now as each row of eight zeroes and ones is a binary number, the maximum number we can have is 1 1 1 1 1 1 1 1 or 255 in decimal (128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255). Every possible combination of pattern has a unique number between zero and 255. By sketching a design on an eight by eight grid, and then working out the decimal equivalent for each line, the design can be assigned to certain keys on the Spectrum (any of the letter keys from A to U can be assigned a graphic character you've created, and the character is accessed by putting the computer in the graphics mode before pressing the relevant key.

Church Spire

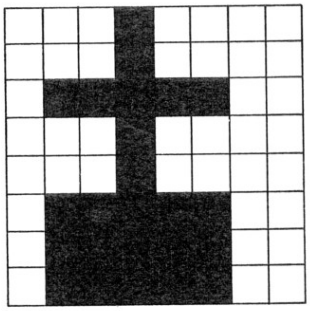
You put a character of your design into place using a simple program such as the following, in which the decimal equivalents of the binary numbers which form the pattern you want (and there is a binary/decimal conversion chart in the appendices to simplify this task) are placed in the DATA statement, and the key which you wish to assign to the character is placed in quotation marks after the word USR in the following program:

```
  5 REM CHURCH WITH SPIRE
 10 DATA 16,16,124,16,16,124,12
4,124
 20 FOR X=0 TO 7
 30 READ Z
 40 POKE USR "A"+X,Z
 50 NEXT X
```

This program produces a little symbol (of a church with spire) which would be suitable to use on a map. You get the computer to print the symbol either by pressing the 'A' key after getting into the graphics mode, or by telling the computer to PRINT CHR$ 144:

Here is the relevant pattern on the eight by eight grid,

with the binary numbers and their decimal equivalents beside them:



= 0 0 0 1 0 0 0 0 = 16
= 0 0 0 1 0 0 0 0 = 16
= 0 1 1 1 1 1 0 0 = 124
= 0 0 0 1 0 0 0 0 = 16
= 0 0 0 1 0 0 0 0 = 16
= 0 1 1 1 1 1 0 0 = 124
= 0 1 1 1 1 1 0 0 = 124
= 0 1 1 1 1 1 0 0 = 124

Coniferous Wood

It is sometimes necessary to print a defined character in two or more parts. This can happen if you decide that a single character cell is too small for your needs. The next program first defines a 'tree' character, and then uses it with existing Spectrum characters to create another Ordnance survey graphic – coniferous wood, unfenced:

```
Coniferous wood, unfenced
```

Here is the program to produce that effect, with the little trees in line 150 produced by getting into the graphics mode, then pressing the A key:

```
  10 REM CONIFEROUS WOOD
  20 FOR X=0 TO 7
  30 READ Z: POKE USR "A"+X,Z
  40 NEXT X
 100 PAPER 5: INK 4: BORDER 1: C
LS
 105 PRINT INK 9;AT 3,2;"Conifer
ous wood, unfenced"
 110 PRINT INK 9;AT 10,11;"-----
---";AT 11,10;"/";AT 11,19;"\"
 120 FOR x=1 TO 4
 130 PRINT INK 9;AT 11+X,9;"!";A
T 11+x,20;"!"
 135 NEXT x
 140 PRINT INK 9;AT 16,10;"\";AT
 16,19;"/"
 150 PRINT AT 12,14;"A   A";AT 1
4,12;"A  A   A";AT 15,13;"A   A"
 160 PRINT INK 9;AT 17,11;"-----
---"
 170 DATA 16,16,56,84,146,56,84,
16
```

Such graphics can be incorporated into a simplified map, with a key letter by the side, with the pupil being asked to identify each character, or they could even be placed into a multiple choice quiz program.

Chemistry Subscripts and Superscripts

The following program follows up this idea, but in a different subject area – Chemistry. In this subject, as you know, there are certain symbols which are unique and can only be obtained using a microcomputer with user-definable characters. As before, the characters were first drawn out on an eight by eight grid, and the resulting decimal number was placed into DATA statements. Because there are a number of characters to be defined, the READ and DATA are in a separate FOR/NEXT loop.

The defined character with its corresponding keyboard character is given in the following table. The keys 'A' to 'L' have been chosen so that a template could be constructed to lay over that row of keys to help remind you which symbol is assigned to which key.

SYMBOL	KEY	CHR§
Superscripts		
+	a	144
−	s	162
2+	d	147
2−	f	149
3+	g	15Ø
3−	h	151
Subscripts		
2	j	153
3	k	154
4	l	155

Here is the program which produces the Chemistry subscripts and superscripts:

```
  10 REM CHEMISTRY  SUBSCRIPTS
        AND SUPERSCRIPTS
  20 GO SUB 500
  25 INK 1: BORDER 1: PAPER 5: C
LS
  30 PRINT INK 9;"Get into Graph
ics Mode, then     try the keys A
 to L"
  40 STOP
 500 DATA "A",16,16,124,16,16,0,
0,0
 501 DATA "S",0,0,124,0,0,0,0,0
 502 DATA "D",34,82,23,34,114,0,
0,0
 503 DATA "F",32,80,23,32,112,0,
0,0
 504 DATA "G",114,18,55,18,114,0
,0,0
 505 DATA "H",112,16,55,16,112,0
,0,0
 506 DATA "J",0,0,0,32,80,16,32,
112
 507 DATA "K",0,0,0,112,16,48,16
,112
 508 DATA "L",0,0,0,16,48,80,120
,16
 510 FOR X=1 TO 9
 520 READ A$
 530 FOR A=0 TO 7
 540 READ B: POKE USR A$+A,B
 550 NEXT A
 560 NEXT X
 570 RETURN
```


Here is a small sample of what can be written when the program has been run. Note that a program to define characters can be deleted once it has been run, as the redefined characters remain even after 'NEWing'. They will stay there until you disconnect the power:

```
Chemical                  Symbol
1 - CopperII Sulphate     CuSO4
2 - IronIII Sulphate      Fe2(SO4)3
3 - Aluminium ion         Al3+

Neutralisation reaction:
H+(aq) + OH-(aq) -> H2O(l)
```

And if you wish to repeat this demonstration, here's the program I used to produce it:

```
1000 PRINT "Chemical";TAB 22;"Sy
mbol"
1010 PRINT '"1 - CopperII Sulpha
te";TAB 22;"CuSO4"
1020 PRINT '"2 - IronIII Sulphat
e";TAB 22;"Fe2(SO4)3"
1030 PRINT '"3 - Aluminium ion";
TAB 22;"Al3+"
1040 PRINT '''"Neutralisation re
action:"
1050 PRINT '"H+(aq) + OH-(aq)
 -> H2O(l)"
```

Molecular Weights

Our next program uses the defined symbols. The chemical formulae for five substances are stored in an array. The value for the weight of one mole of each of these substances is stored in another array. One of the substances is chosen at random from the array and at the same time a value between Ø.5 and 2.5 is chosen as a multiplying value for the molecular weights.

The student is given the formula for the chosen substance, and the number of moles. He or she is then asked to key in the correct mass in grams.

Once you've run the program in its present form, you'll

probably find a number of ways to improve it, such as (a) making it more 'friendly' by getting the computer to ask for, and use, the student's name; (b) giving a score of correct answers at the end, with a suitable comment depending on the number scored; (c) by increasing the number of substances; and (d) by printing on screen a page of atomic weights.

In this program, I've used the standard approximate values as used in C.S.E. and 'O' level examinations:

Calcium 40	Carbon 12	Hydrogen 1
Iron 56	Lead 207	Nitrogen 14
Oxygen 16	Sodium 23	Sulphur 32

In order to obtain the chemical subscripts and superscripts, the previous program is embedded in this program as you can see:

```
  10 REM Molecular weight
       calculations
  15 LET SC=0
  20 GO SUB 500
  25       PAPER 5: INK 1: BORDER
1
 100 FOR J=1 TO 10
 105 LET X=INT (RND*5)+1
 110 LET Y=INT (RND*5)*0.5+0.5
 115 CLS
 120 PRINT INVERSE 1;AT 0,8;"Pro
blem number ";J
 125 PRINT "Work out the mass in
 grams of:-"
 130 PRINT INVERSE 1;AT 4,2;"SUB
STANCE     NUMBER OF MOLES"
 135 PRINT AT 7,3;A$(X);AT 7,22;
Y
 140 PRINT INK 9;AT 14,0;"Key in
 your answer to the","nearest gr
am then press "; INVERSE 1;"ENTE
R"
 150 LET ANS=VAL (B$(X))*Y
 160 INPUT C$
 165 IF C$="" THEN GO TO 160
 170 IF CODE C$<49 OR CODE C$>58
 THEN GO TO 160
 180 IF VAL C$<ANS - 0.05 OR VAL
 C$ >ANS + 0.05 THEN GO TO 250
 200 CLS
 205 FOR X=1 TO 300
 210 PRINT FLASH 1;AT 10,5;"You
are correct!"
 220 NEXT X
 230 LET SC=SC+1
 235 NEXT J
 240 GO TO 700
 250 CLS
```



```
 260>PRINT INVERSE 1;AT 10,0;"I
am sorry, but you are wrong"
 265 PRINT '''"The answer is ";AN
S;" gms"
 270 PAUSE 100
 280 GO TO 235
 500 DATA "A",16,16,124,16,16,0,
0,0
 501 DATA "S",0,0,124,0,0,0,0,0
 502 DATA "D",34,82,25,34,114,0,
0,0
 503 DATA "F",32,80,23,32,112,0,
0,0
 504 DATA "G",114,18,55,18,114,0
,0,0
 505 DATA "H",112,16,55,16,112,0
,0,0
 506 DATA "J",0,0,0,32,80,16,32,
112
 507 DATA "K",0,0,0,112,16,48,16
,112
 508 DATA "L",0,0,0,16,48,80,120
,16
 510 FOR X=1 TO 9
 520 READ A$
 530 FOR A=0 TO 7
 540 READ B: POKE USA A$+A,B
 550 NEXT A
 560 NEXT X
 600 DIM A$(5,12)
 605 DIM B$(5,3)
 610 DATA "CUSO4.5H2O","CACO3","
FEIII(SO4)3","Pb(NO3)2","NaOH"
 615 FOR X=1 TO 5
 620 READ Z$: LET A$(X)=Z$
 625 NEXT X
 630 DATA "250","100","400","331
","40"
 635 FOR X=1 TO 5
 640 READ Z$: LET B$(X)=Z$
 645 NEXT X
 650 RETURN
 700 CLS
 705 PRINT AT 8,3;"You scored ";
SC;" out of 10"
 710 PRINT '''"Press any key for
another run"
 720 PAUSE 0
 730 RANDOMIZE
 740 GO TO 100
```

```
          Problem number 3
Work out the mass in grams of:-

  SUBSTANCE      NUMBER OF MOLES

   Pb(NO3)2              2.5

Key in your answer to the
nearest gram then press ENTER
```

```
PROBLEM NUMBER 3
Work out the mass in grams of:-

SUBSTANCE          NUMBER OF MOLES

NaOH                    0.5


Key in your answer to the
nearest gram then press ENTER


    You scored 7 out of 10
Press any key for another run
```

To make the program larger, more substances can be placed in the array A$, by being tagged onto the end of the DATA statement in line 61Ø. A$ should be redimensioned accordingly in line 6ØØ. The mass of one mole of the new substances should be placed in B$, via the DATA statement in line 63Ø, and B$ redimensioned accordingly in 6Ø5.

The two FOR/NEXT loops should then be changed in lines 615 and 635. These are the loops which read the DATA into the arrays. Line 1Ø5 should also be changed so that the random number 'X' is increased to allow for the extra chemicals.

As it stands, the program only allows for Ø.5 to 2.5 times the mass of one mole, in steps of Ø.5 moles. This is sufficient for third year and C.S.E. Chemistry, but to make it more difficult, the random number Y in line 11Ø could be altered. For example, if steps of Ø.1 are needed, the line should read:

```
Let Y = INT(RND*25)*Ø.1 + Ø.1
```

Other variables used in this program are:

J – counter for number of problems; set at 1Ø in line 1ØØ and can be changed if required.

ANS – this stores the correct answer and is generated in line 15Ø. To avoid problems with numbers not being stored exactly in the computer, line 18Ø allows for an error of Ø.Ø5 either way.

C$ – this is the answer entered by the student via line 16Ø. It is error–trapped in lines 165 and 17Ø so that (a) the computer will not just allow the ENTER key to be pressed without an answer being entered and (b) the first element of the answer must be a number (error trapping is discussed in detail in chapter IX).

SC – holds the score of number of problems correctly solved. It is incremented only if the pupil answers correctly in line 23Ø.

CHAPTER SEVEN

Using the Spectrum graphics effectively Part Two

In this second chapter on the graphics, we will attempt to answer a question posed by many teachers: 'What can you do with a microcomputer which you can't do with a blackboard?'

With limited graphic capabilities, a monochrome computer could rarely compete with a skilled teacher. However, now that we have the Spectrum, with its high resolution graphics and colour, there are ways of using it with which the blackboard could never compete. The most obvious area in which this claim is true is when a moving display can impart information in a way which a static drawing could not do. A wide range of subjects – from a beating heart, to a four-stroke engine – can be shown in motion by the Spectrum.

The Nitrogen Cycle

The following program is intended to provide a continuous display. You have no control over the program, in its present form, once it begins, although you can easily insert a PAUSE 0 line to hold the display until a key is pressed at the points in the program which separate one section from the next:

```
  10 REM THE NITROGEN CYCLE
  20 BORDER 6: PAPER 5: INK 1
 100 CLS
 105 PRINT INK 2;AT 1,2;"****The
 NITROGEN CYCLE****"
 110 PRINT AT 3,3;"N2 (79% of ai
r)"
 120 PRINT AT 7,0;"De-"
 125 PRINT AT 8,0;"Nitrifiers"
 130 PRINT AT 7,15;"Nitrogen"
 135 PRINT AT 8,19;"Fixers"
 140 FOR X=1 TO 5: PRINT INK 7;A
T 10-X,10;"*"
```



```
 150 PRINT INK 7;AT 4+X,18;"*":
PAUSE 20
 155 PRINT AT 10-X,10;" "
 160 PRINT AT 4+X,18;" "
 165 NEXT X
 170 PRINT INK 2; FLASH 1; INVER
SE 1;AT 10,10;"NITRATES"
 180 PRINT AT 11,0;"Lightning-->
"
 185 PRINT AT 11,17;"<--Fertiliz
ers"
 190 FOR X=1 TO 5: PRINT INK 7;A
T 10+X,16;"*"
 200 PAUSE 20: NEXT X
 210 FOR X=1 TO 5: PRINT AT 10+X
,16;" "
 215 PRINT INK 7;AT 15,15+X;"*"
 220 PAUSE 20: PRINT AT 15,15+X;
" "
 225 NEXT X
 230 PRINT AT 15,21;"Taken up";A
T 16,21;"by roots";AT 17,23;"("
 250 PRINT INK 2; INVERSE 1;AT 1
8,18;"PLANT PROTEIN"
 260 PRINT AT 19,18;"Eaten to fo
rm"
 270 PRINT INK 2; INVERSE 1;AT 2
1,18;"ANIMAL PROTEIN"
 280 PRINT AT 19,2;"Dead bodies<
--"
 290 PRINT AT 20,3;"+ Faeces  <-
--"
 295 PAUSE 50
 300 PRINT INK 7;AT 18,6;"↑"
 310 PRINT INK 2; INVERSE 1;AT 1
7,0;"Decomposers"
 320 PRINT INVERSE 1;AT 15,3;"NH
3"
 330 PRINT INK 2; INVERSE 1;AT 1
3,0;"Nitrifiers"
 335 PRINT INK 7;AT 18,5;"*"
 340 PAUSE 50
 345 PRINT AT 18,5;" "
 350 PRINT INK 7;AT 14,5;"*"
 355 PAUSE 50
 360 PRINT AT 14,5;" "
 370 FOR X=1 TO 3: PRINT INK 7;A
T 14-X,13;"*"
 375 PAUSE 60
 380 PRINT AT 14-X,13;" ": NEXT
X
 385 PAUSE 200
 390 IF INKEY$="" THEN GO TO 100
```

As is the case with many Spectrum programs, this excerpt from the program gives no real indication of how effective it is when up and running, with colour and animation:

```
    ****The NITROGEN CYCLE****
     N2 (79% of air)


De-                          Nitrogen
Nitrifiers                   Fixers

                NITRATES
Lightning-->            <--Fertilizers

Nitrifiers

    NH3                        Taken up
                               by roots
Decomposers                        |
      ↑                            ↓
  Dead bodies <--              PLANT PROTEIN
    + Faeces  <---             Eaten to form

                               ANIMAL PROTEIN
```

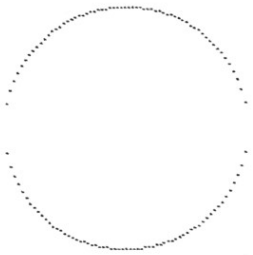
Plotting

The Spectrum is also superb for graphing curves and functions. Watching a shape unfold is far more dramatic than just tediously plotting it out on graph paper, or simply having it drawn fairly roughly on a blackboard.

The first example of this is a brief program showing the polar equation of a circle in action:

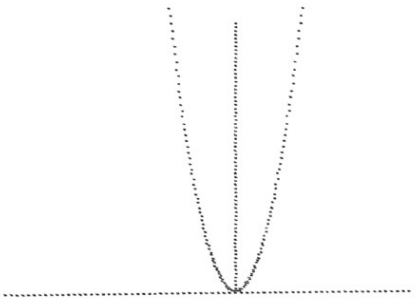
```
  5 REM      POLAR EQUATION
               FOR CIRCLE
 10 FOR A=0 TO 2*PI STEP PI/50
 20 PLOT 115+50*COS A,85+50*SIN
A
 30 NEXT A
```


This one is, by contrast, the cartesian equation for a circle:



```
  5 REM  CARTESIAN EQUATION
               FOR CIRCLE
 10 FOR X=-31 TO 31
 20 PLOT 115+2*X,85+2*SQR (1000
-X*X+1)
 30 PLOT 115+2*X,85-2*SQR (1000
-X*X+1)
 40 NEXT X
```

A program to produce a graph of Y = X ↑ 2 produced this result (with the listing after the sample run):

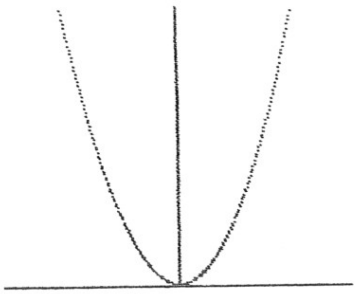


```
 10 REM GRAPH OF Y=X SQUARED
 15 FOR X=-7 TO 7 STEP .1
 20 PLOT 5*X+120,3*X*X+1
 30 PLOT 120,20*X
 40 PLOT 30*X,0
 50 NEXT X
```

To show that it is worth experimenting with a program after it is up and running – as I have advocated several times in the book, I worked on the *scale* a little longer, and produced another version of the program:

```
10 FOR N=0 TO 90 STEP .5
20 PLOT 120,1.62*N
30 PLOT 30+2*N,0
40 NEXT N
50 FOR X=-12 TO 12 STEP .1
60 PLOT 5*X+120,1+X*X
70 NEXT X
```

This is it in action:



This routine plots a tangent curve:

```
 10 REM TANGENT CURVE
 20 FOR X=0 TO 124
 30 PLOT 1.5*X,9*TAN (X/6.05)+2
0
 40 NEXT X
```

And this one a reciprocal graph:


```
10 PRINT "RECIPROCAL GRAPH"
20 FOR X=1 TO 10 STEP .01
30 PLOT 10*X,100/X
40 NEXT X
```

Sine Design

This program produces an evolving design based on a sine wave:

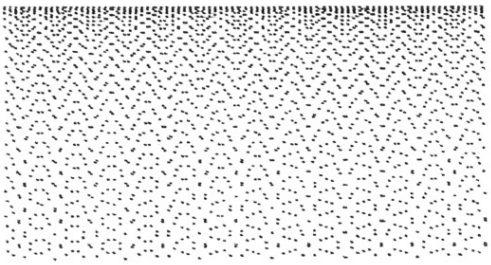
```
 10 REM SINE DESIGN
 15 FOR M=1 TO 2
 20 FOR N=0 TO 255
 30 IF M=1 THEN PLOT N,130*SIN
(N/20)
 35 IF M=2 THEN PLOT OVER 1;N,1
30*SIN (N/20)
 40 NEXT N
 50 NEXT M
```

Bouncing Ball

The next program is hard to explain, although the explanation should make perfect sense once you see it in motion. You need to imagine that a ball (which shows a remarkable tendency not to lose energy) is bouncing on the screen. At precise time intervals, a light flashes, casting a permanent, sharp shadow on the wall behind the ball. The ball moves slightly to the right as it bounces, so eventually it

leaves the screen on the right hand side. The program produces the pattern left on the wall by the captured shadows:

```
10 REM PATTERN LEFT BY
   CONSECUTIVE POSITIONS OF
   A BOUNCING BALL AT REGULAR
        TIME INTERVALS
20 FOR N=0 TO 250 STEP .1
30 PLOT N,130*COS N
40 NEXT N
```

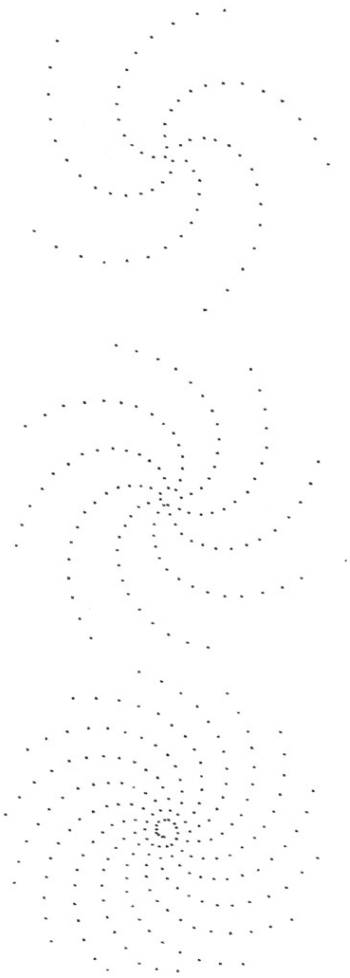


If you want to see the ball in motion, add line 35:

```
20 FOR N=0 TO 250 STEP .1
30 PLOT N,130*COS N
35 PLOT OVER 1;N,130*COS N
40 NEXT N
```

Scatter Spiral Plot

This program produces what I have called a 'scatter spiral plot'. Using a random step size, it first plots, then 'unplots' (using PLOT OVER !, see line 25) a Catherine-wheel-like design. You'll understand what I mean when you set it running. You can leave this program running for a long time:



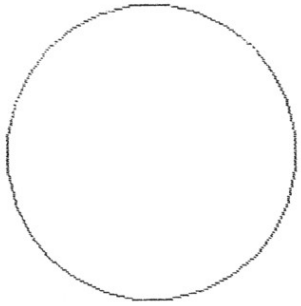
```
   5 REM SCATTER SPIRAL PLOT
   6 LET R=RND+0.25
   7 FOR S=1 TO 2
  10 FOR A=PI TO 30*PI STEP R
  20 IF S=1 THEN PLOT 125+0.9*A*
SIN A,80+0.9*A*COS A
  25 IF S=2 THEN PLOT OVER 1;125
+0.9*A*SIN A,80+0.9*A*COS A
  30 NEXT A
  40 NEXT S
```

Shapes

Now we'll look at ways of making five useful shapes with the Spectrum. You may well find you can use these shapes to 'dress up' the graphic display of your programs:

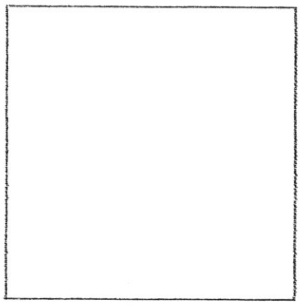
circle

```
10 CIRCLE 100,100,75
```



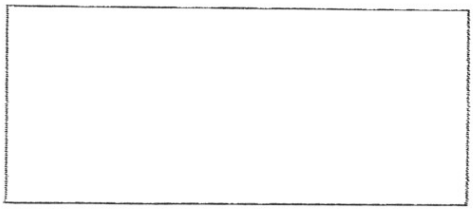
Square

```
10 DRAW 150,0
20 DRAW 0,150
30 DRAW -150,0
40 DRAW 0,-150
```



Oblong

```
10 DRAW 240,0
20 DRAW 0,100
30 DRAW -240,0
40 DRAW 0,-100
```

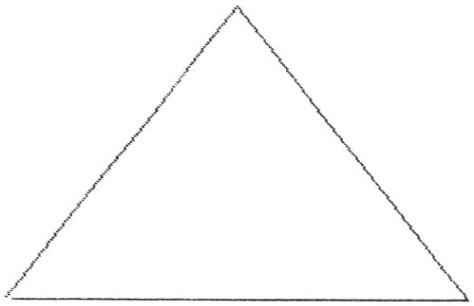


semicircle

```
10 PLOT 50,50: DRAW 75,75,PI
20 DRAW -75,-75
```

Triangle

```
10 DRAW 120,150
20 DRAW 120,-150
30 DRAW -235,0
```



If you are dealing with young children, a card to help them program (as explained in the section of the book on using the computer in infant school) can be of great help:

Making shapes

CIRCLE – use key H DRAW – use key W

Circle O

CAPS SYMBOL / CAPS CIRCLE 100 SYMBOL 9 100 SYMBOL 9 75 ENTER

Triangle △

10 DRAW 120 SYMBOL 9 150 ENTER
20 DRAW 120 SYMBOL 9 SYMBOL — 150 ENTER
30 DRAW SYMBOL — 235 SYMBOL 9 0 ENTER

Square □

10 DRAW 150 SYMBOL 9 0 ENTER
20 DRAW 0 SYMBOL 9 150 ENTER
30 DRAW SYMBOL — 150 SYMBOL 9 0 ENTER
40 DRAW 0 SYMBOL 9 SYMBOL — 150 ENTER

Oblong ▭

10 DRAW 240 SYMBOL 9 0 ENTER
20 DRAW 0 SYMBOL 9 100 ENTER
30 DRAW SYMBOL — 240 SYMBOL 9 0 ENTER
40 DRAW 0 SYMBOL 9 SYMBOL — 100 ENTER

Text Manipulation

There are a number of 'tricks' you can apply to text output to make it more interesting on the Spectrum, as these next four routines by David Perry demonstrate.

The first Perry routine turns writing upside-down:

```
 10 INPUT "ENTER A WORD";a$: IF
LEN a$>30 THEN GO TO 10
 15 PRINT AT 0,0;a$
```

```
  20 LET z=166: FOR a=167 TO 175
  30 FOR n=0 TO (LEN a$*8)
  40 IF POINT (n,a)=1 THEN PLOT
n,z
  50 NEXT n: LET z=z-1: NEXT a
```

This one allows you to write sideways:

```
  10 INPUT "ENTER A WORD";a$: IF
 LEN a$>20 THEN GO TO 10
  15 PRINT AT 0,0;a$
  20 LET z=0: FOR a=175 TO 168 S
TEP -1
  30 FOR n=(LEN a$*8) TO 0 STEP
-1
  40 IF POINT (n,a)=1 THEN PLOT
z,n
  50 NEXT n: LET z=z-1: NEXT a
```

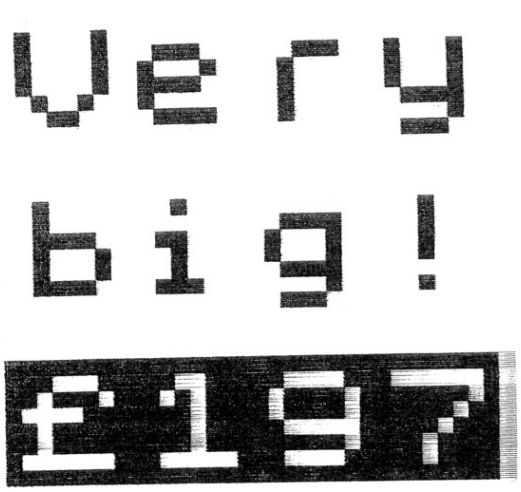
The third routine puts a frame around your words:

```
  10 BORDER 0: PAPER 0: INK 7: C
LS
  20 INPUT "WORD? ";a$
  30 INPUT "X-Co.";x: INPUT "Y-C
o.";y
  40 PRINT AT x,y;a$
  50 LET c=((LEN a$)*8): LET a=(
8*y)-2: LET b=(8*(21-x)-2)
  60 PLOT a,b: DRAW c+4,0: DRAW
0,12: DRAW -c-4,0: DRAW 0,-12
```

And the fourth and final Perry routine allows you to write in large letters on the screen:

```
   2 INPUT "Do you want normal l
etters (1)  or inverse ones (2)?
 ";Z
   3 IF Z>=2 THEN LET Z=0
  10 INPUT "Enter your word (no
more than   four letters) ";A$
  20 PRINT AT 0,0;A$
  25 LET L=LEN A$*8-1
  30 FOR X=168 TO 175: FOR Y=0 T
O L
  40 IF POINT (Y,X)=Z THEN PRINT
 AT X-166-20,Y;"■"
  50 NEXT Y: NEXT X
```

As you can imagine by looking at this printout, it can be very effective:



Character Generator

Finally, in this chapter, we have an outstanding character generator program, again written by David Perry, which should make your job of producing the most effective displays simpler than it would otherwise be.

The program has a large number of commands. You move the cursor with the 5, 6, 7 and 8 keys (in the direction of the arrows above those keys). Key Ø fills in the block the cursor is over, or 'empties' an already filled block.

You can pick up, and alter characters at the press of a key, invert the character presently on the grid and SAVE the characters on tape for your own programs.

When you run it, you'll see the program contains full details of the ways in which it can be used, along with clear instructions on which keys to use.

```
  10 REM CHARACTER GENERATOR
         DAVID PERRY 1983
  20 BORDER 1: PAPER 0: INK 7: B
RIGHT 1: CLS
```

```
  30 PRINT " A B C D E F G H I J
 K L M N O P"
  40 PRINT " ● ▚ ◣ ◥ ▣ ▣ ╳ ○ ▨ ▨
 ◖ ◗ ❦ ❣ ▩ ⅄"
  50 PRINT BRIGHT 8; INK 5;"  BI
NARY      INVERSE     NORMAL"
  60 PRINT "01234567 8     01234567
8 01234567 8"
  70 DIM z(8,8)
  80 FOR n=1 TO 8
  90 PRINT n; INK 2;"00000000
 "; INK 7;n; INK 5;"■■■■■■■■ ";
INK 7;n
 100 NEXT n
 110 GO SUB 530
 120 INPUT "X-Axis (1 to 8)",b:
IF b>8 OR b<1 THEN GO TO 120
 130 INPUT "Y-Axis (1 to 8)";a:
IF a>8 OR a<1 THEN GO TO 130
 140 PRINT AT a+3,b; INK 4;"X";A
T a+3,b+13;"X";AT a+3,b+23;"X"
 150 LET k$=INKEY$: IF k$="" THE
N GO TO 150
 160 IF (k$="0" AND z(a,b)=0) TH
EN LET z(a,b)=1: GO TO 180
 170 IF (k$="0" AND z(a,b)=1) TH
EN LET z(a,b)=0
 180 IF z(a,b)=0 THEN PRINT AT a
+3,b; INK 2;"0";AT a+3,b+13; INK
 5;"■";AT a+3,b+23; PAPER 0;" "
 190 IF z(a,b)=1 THEN PRINT AT a
+3,b; INK 7;"1";AT a+3,b+13; PAP
ER 0;" ";AT a+3,b+23; INK 3;"■"
 200 LET a=a+(INKEY$="6" AND a<8
)-(INKEY$="7" AND a>1)
 210 LET b=b+(INKEY$="8" AND b<8
)-(INKEY$="5" AND b>1)
 220 IF INKEY$="p" THEN COPY
 230 IF INKEY$="s" THEN GO SUB 5
90
 240 IF INKEY$="g" THEN GO TO 10
50
 250 IF INKEY$="i" THEN GO TO 96
0
 260 IF INKEY$="r" THEN GO TO 33
0
 270 IF INKEY$="x" THEN GO TO 12
0
 280 IF INKEY$="c" THEN GO SUB 3
40
 290 IF INKEY$="d" THEN GO SUB 1
060
 300 IF INKEY$="a" THEN GO SUB 7
40
 310 IF INKEY$="e" THEN GO SUB 9
20
 320 GO TO 140
 330 CLS : RUN 30
 340 LET z$="": INPUT "Letter fo
r character? ";D$: IF d$<"a" OR
 d$>"u" THEN GO TO 340
```



```
 350 DIM a$(8,8)
 360 FOR x=1 TO 8
 370 FOR y=1 TO 8
 380 LET a$(x,y)=SCREEN$ (x+3,y)
 390 NEXT y: NEXT x
 400 FOR x=1 TO 8
 410 LET z$=""
 420 FOR y=1 TO 8
 430 LET z$=z$+a$(x,y)
 440 NEXT y
 450 LET no=0
 460 IF z$(1)="1" THEN LET no=no
+128
 470 IF z$(2)="1" THEN LET no=no
+64
 480 IF z$(3)="1" THEN LET no=no
+32
 490 IF z$(4)="1" THEN LET no=no
+16
 500 IF z$(5)="1" THEN LET no=no
+8
 510 IF z$(6)="1" THEN LET no=no
+4
 520 IF z$(7)="1" THEN LET no=no
+2
 530 IF z$(8)="1" THEN LET no=no
+1
 540 POKE USR d$+x-1,no
 550 PRINT AT x+3,9; FLASH 1;no;
 FLASH 0;("   " AND no<=99)
 560 NEXT x
 570 PRINT AT 1,0;" ● ▚ ◣ ◥ ▣ ▣
╳ ○ ▨ ▨ ◖ ◗ ❦ ❣ ▩ ⅄"
 580 RETURN
 590 CLS : PRINT AT 20,0;"  CHEC
K LEADS,START RECORDER "
 600 INPUT "NAME?",A$
 610 SAVE A$CODE USR "A",168
 620 RUN
 630 PRINT AT 13,1;"I=INVERT CHA
RACTER"
 640 INK 6: PRINT AT 14,1;"S=SAV
E CHARACTERS"
 650 PRINT AT 12,1;"0=FILL/EMPTY
 BLOCK"
 660 PRINT AT 15,1;"P=PRINTER CO
PY      7"
 670 PRINT AT 16,1;"R=START AGAI
N       5X8"
 680 PRINT AT 17,1;"C=DEFINE CHA
RACTER    6"
 690 PRINT AT 18,1;"X=CHANGE CO-
ORDINATES"
 700 PRINT AT 19,1;"D=INPUT DECI
MAL DATA"
 710 PRINT AT 20,1;"A=PICK UP A
CHARACTER"
 720 PRINT AT 21,1;"E=ERASE A CH
ARACTER"
 730 RETURN
 740 REM pick up
```

```
  750 INPUT "Character to pick up
  A to U";r$: IF r$>"u" OR r$<"a"
  THEN GO TO 750
  760 LET r$=r$(1 TO 1)
  770 FOR y=0 TO 7
  780 LET s=PEEK (USR r$+y)
  790 LET l=128
  800 FOR x=0 TO 7
  810 LET t=INT (s/l)
  820 PRINT AT y+4,x+1; INK 2;("0
" AND NOT t=1); INK 7;("1" AND N
OT t<>1)
  830 PRINT AT y+4,x+14; INK 5;("
■" AND NOT t=1); PAPER 0;(" " AN
D NOT t<>1)
  840 PRINT AT y+4,x+24; PAPER 0;
(" " AND NOT t=1); INK 3;("■" AN
D NOT t<>1)
  845 IF SCREEN$ (y+4,x+1)="0" TH
EN LET z(y+1,x+1)=0
  846 IF SCREEN$ (y+4,x+1)="1" TH
EN LET z(y+1,x+1)=1
  850 LET s=s-t*l
  860 LET l=l/2
  870 NEXT x: NEXT y
  880 FOR n=1 TO 8: PRINT AT n+3,
9;"  ": NEXT n
  890 INPUT "X-Axis (1 to 8)";a:
IF a>8 OR a<1 THEN GO TO 120
  900 INPUT "Y-Axis (1 to 8)",A:
IF a>8 OR a<1 THEN GO TO 130
  910 RETURN
  920 INPUT "Character to ERASE ?
";d$: IF d$<"a" OR d$>"u" THEN G
O TO 920
  930 FOR f=0 TO 7: POKE USR d$+f
,0: NEXT f
  940 PRINT AT 1,0;" ● ◆ ◣ ◥ ◘ ◘
X O ⊠ ⊠ ◖ ◗ ♥ ☤ ▒ ☠"
  950 RETURN
  960 FOR u=1 TO 8
  970 FOR f=1 TO 8
  980 IF SCREEN$ (u+3,f)="0" THEN
  GO TO 1010
  990 LET z(u,f)=0: PRINT AT u+3,
f; INK 2;"0";AT u+3,f+13; INK 5;
"■";AT u+3,f+23; PAPER 0;" "
1000 GO TO 1020
1010 LET z(u,f)=1: PRINT AT u+3,
f; INK 7;"1";AT u+3,f+13; PAPER
0;" ";AT u+3,f+23; INK 3;"■"
1020 NEXT f
1030 NEXT u
1040 GO TO 120
1050 CLS : PRINT "●◆◣": PRINT "◥
◘◘": PRINT "XO⊠": PAUSE 0: CLS :
  RUN
1060 INPUT "LETTER ? ";l$: FOR f
=0 TO 7: INPUT "Decimal? ";s: PO
KE USR l$+f,s: NEXT f
1070 PRINT AT 1,0;" ● ◆ ◣ ◥ ◘ ◘
X O ⊠ ⊠ ◖ ◗ ♥ ☤ ▒ ☠": RETURN
```



```
 A B C D E F G H I J K L M N O P
 ● ◆ ◣ ◥ ◘ ◘ X O ⊠ ⊠ ◖ ◗ ♥ ☤ ▒ ☠
   BINARY          INVERSE      NORMAL
01234567 8     01234567 8  01234567 8
10000000 1     1          1
20011110 0     2          2
30111111 1     3          3
41111100 0     4          4
51111000 0     5          5
61111100 0     6          6
70111111 1     7          7
80011110 0     8          8
 0=FILL/EMPTY BLOCK
 I=INVERT CHARACTER
 S=SAVE CHARACTERS
 P=PRINTER COPY          7
 R=START AGAIN         5X8
 C=DEFINE CHARACTER      6
 X=CHANGE CO-ORDINATES
 D=INPUT DECIMAL DATA
 A=PICK UP A CHARACTER
 E=ERASE A CHARACTER
```

CHAPTER EIGHT

Using the Spectrum for English and Other Languages

The computer is a little more limited in this area than in the maths one, although with care and patience, you can develop programs which will be of benefit.

Spelling Program

I'll start with a spelling program which can also be used as a foreign language drill program. This program stores 12 commonly misspelled words, and gives 10 questions on them. Once you have the program running, you simply need to alter the DATA statements (from line 25Ø) to change the program.

To use it for foreign language drill, simply change the line which begins "Please choose the correct spelling for . . ." (line 11Ø) to something like "Please enter the French word for . . .".

Here is the program in action:

```
Please choose the correct
spelling from the alternatives
and type it in (in capital
letters)

ABSENSE

ABSCENSE

ABSENCE

Your spelling was ABSENCE

Well done, ABSENCE
is correct

You have 1 right
out of 1

Please stand by....
```



```
Please choose the correct
spelling from the alternatives
and type it in (in capital
letters)

IRREPARABLE

IREPARABLE

IRRAPERABLE

Your spelling was IRREPERABLE
but that is wrong.

The correct spelling is
IRREPARABLE

You have 1 right
out of 2

Please stand by....


Please choose the correct
spelling from the alternatives
and type it in (in capital
letters)

UNDERATE

UNDERRATE

UNDERAIT

Your spelling was UNDERRATE

Well done, UNDERRATE
is correct

You have 2 right
out of 3

Please stand by....
```

And this is the listing for it:

```
10 REM SPELLING
15 RANDOMIZE
20 LET SCORE=0
25 DIM F$(11,7)
30 FOR H=1 TO 10
40 RESTORE 250+10*INT (RND*9)
60 READ A$
65 FOR T=1 TO H
```

```
  65 IF RND>.5 THEN IF F$(T) ( TO
 7)=A$( TO 7) THEN GO TO 40
  70 NEXT T
  75 LET F$(H)=A$
  80 READ B$
  90 READ C$
 100 READ D$
 110 PRINT "Please choose the co
rrect        spelling from the al
ternatives   and type it in (in c
apital       letters)"
 120 PRINT 'B$
 130 PRINT 'C$
 140 PRINT 'D$
 150 INPUT E$
 155 PRINT '"Your spelling was "
;E$
 160 IF E$=A$ THEN PRINT '"Well
done, ";E$,"is correct": LET SCO
RE=SCORE+1
 170 IF E$<>A$ THEN PRINT "but t
hat is wrong."''"The correct spe
lling is ";A$
 180 IF SCORE>0 THEN PRINT ''"Yo
u have ";SCORE;" right"'"out of
";H
 190 PRINT '' FLASH 1;"Please st
and by...."
 195 INPUT a$: IF a$<>"" THEN CO
PY
 200 PAUSE 200
 210 CLS
 220 NEXT H
 230 PRINT '''"You managed to sp
ell ";SCORE'"word correctly"
 240 STOP
 250 DATA "ABSENCE","ABSENSE","A
BSCENSE","ABSENCE"
 260 DATA "BELIEVED","BELIEVED",
"BELEIVED","BEILIEVED"
 270 DATA "COLLEAGUES","COLEAGUE
S","COLLEAGUES","COLLEAGES"
 280 DATA "COMPARATIVE","COMPARA
TIVE","COMPARITIVE","COMPARATAVE
"
 290 DATA "CORROBORATE","CORROBO
RRATE","COROBORATE","CORROBORATE
"
 300 DATA "IRREPARABLE","IRREPAR
ABLE","IREPARABLE","IRRAPERABLE"
 310 DATA "REPLACEABLE","REPLACA
BLE","REPLACEABLE","REPLACIBLE"
 320 DATA "PARALLEL","PARALELL",
"PARALLEL","PARALEL"
 330 DATA "UNDERRATE","UNDERATE"
,"UNDERRATE","UNDERAIT"
 340 DATA "UNNECESSARY","UNECESS
ARY","UNNECCESSARY","UNNECESSARY
"
 350 DATA "WOOLLEN","WOOLEN","WO
OLLEN","WOOLIN"
```



```
 360 DATA "DISCREPANCY","DISCORE
PANCY","DISCREPENCY","DISCREPANC
Y"
```

The words used in the DATA statements were chosen from those words which are most commonly misspelled. Here is a list of several such words, from which you can create other spelling tests:

absence accessible accommodate
accommodation achieved acknowledge
acquainted acquiesce acquiescence
addresses aerial aggravate
aggregate agreeable analysis
analyses ancillary apparent
believed beneficial budgeted
category ceiling chaos choice
committee competent connoisseur
courtesy cursory deceive
definite dissatisfied embarrassed
exigency expenses extremely
fulfilment gauge grievance
guardian harassed independent
instalment irreparable
knowledge liaison maintenance
misspelled naive negotiate
niece noticeable omitted
parallel permanent preceding
preliminary professor
proprietary psychology
recommend regrettable replaceable
scarcely statutory supersede
tendency twelfth underrate
usually valuable withhold

If you want to modify this program for younger students, for whom the most-commonly misspelled words would be too difficult, you can replace them with words from this list selected from the most frequently used words in English:

which her had from they their
has were been will there who when
what your more would them some
than may upon its out into our
these like shall great now such
should other only any then can

about those made well old must
said time even new could very
much own might first after yet

Anagrams

The next program in this chapter is an ANAGRAM one written by Derek Cook. It contains full instructions, and is designed for use by children in the six to nine years age group:

```
 200 PAPER 0: CLS : PAPER 2: INK
 6
 210 PRINT AT 10,1;"Do you know
about ANAGRAMS?"
 220 PAUSE 100: CLS
 230 BORDER 1: PAPER 7: INK 0: C
LS
 300 PRINT "You take a word:"
 310 FOR a=1 TO 6
 315 READ a,a$
 320 INK a: PRINT AT 5,12+a;a$
 330 DATA 1,"R",2,"E",3,"S",4,"U
",5,"L",6,"T"
 340 NEXT a
 345 PAUSE 100
 350 INK 0: PRINT AT 3,0;"and mi
x the letters up:"
 360 RESTORE 400
 370 FOR z=1 TO 6
 380 READ b,b$
 390 INK b: BEEP .1,z: PRINT AT
6,12+z;b$
 395 PAUSE 50
 400 DATA 4,"U",5,"L",3,"S",6,"T
",2,"E",1,"R"
 410 NEXT z
 420 RESTORE 450
 430 FOR y=1 TO 6
 435 READ c,c$
 440 INK c: BEEP .1,y: PRINT AT
7,12+y;c$
 445 PAUSE 50
 450 DATA 5,"L",4,"U",3,"S",6,"T
",1,"R",2,"E"
 460 NEXT y
 470 RESTORE 500
 475 FOR x=1 TO 6
 480 READ d,d$
 490 INK d: BEEP .1,x: PRINT AT
8,12+x;d$
 500 DATA 1,"R",4,"U",3,"S",6,"T
",5,"L",2,"E"
 510 NEXT x
 512 PRINT AT 12,3;"to make othe
r words"
 515 PAUSE 200: CLS : BORDER 1:
PAPER 6: INK 2
```



```
 520 PRINT "The computer can mak
e anagrams  for you but most of
them will benonsense words"
 525 PRINT AT 5,0;"For instance:
"
 530 FLASH 1: PRINT AT 7,13;"STR
ULE"
 535 FLASH 0: PRINT AT 9,0;"Is t
here a SUTLER? Look it up inthe
dictionary!"
 540 PRINT AT 11,0;"Nonsense wor
ds can be fun;make  up your own
meanings for them.  If you want
me to make some     anagrams for
 you,press ENTER"
 550 INPUT j$
 560 CLS
 600 RANDOMIZE
 605 PAPER 0: CLS : PAPER 2: INK
 6
 610 INPUT "Type your word,then
press ENTER";a$
 620 LET l=LEN a$
 630 DIM b$(2,l)
 640 FOR n=1 TO 40
 650 FOR c=1 TO l
 660 LET b$(1,c)=a$(c)
 670 LET b$(2,c)="1"
 680 NEXT c
 690 FOR d=1 TO l
 700 LET r=INT (RND*l)+1
 710 IF b$(2,r)="0" THEN GO TO 7
00
 720 LET b$(2,r)="0"
 730 PRINT b$(1,r);
 740 NEXT d
 750 PRINT ,
 760 NEXT n
 770 BORDER 1: PAPER 0: INK 7: F
LASH 1: PRINT "press ENTER for i
nstructions"
 780 FLASH 0: INPUT i$
 790 IF i$="i" THEN GO TO 800
 800 CLS : BORDER 3: PAPER 6: IN
K 0: PRINT AT 5,0;"Enter:"
 810 PRINT AT 7,0;"1 for another
 40 anagrams of thesame word
                    2 for anagram
s of another word
                    3 to say Good
-bye"
 820 INPUT p
 830 IF p=1 THEN GO TO 620
 840 IF p=2 THEN GO TO 605
 850 CLS : BORDER 4: PAPER 3: IN
K 5: PRINT AT 11,12;"dogo-ybe"
```

Faster Reading

The final program in this chapter, written by Gordon

Armitt, is designed to aid in developing faster reading speeds. Here is the program in action:

```
NOW ENTER THE LETTERS YOU SAW

My letters were RZWE
Yours were REWO

No, you are wrong
You lose 10 points!
You have -10 points

Stand by for a new test

NOW ENTER THE LETTERS YOU SAW

My letters were TSOE
Yours were TSOE

Yes, you are right
You score 10 points!
You have 40 points

Stand by for a new test



NOW ENTER THE LETTERS YOU SAW

My letters were VJFC
Yours were VJFC

Yes, you are right
You score 10 points!
You have 50 points
YOU'VE PASSED WITH FLYING
COLOURS!
```



```
TYPE IN THE NUMBERS YOU SAW


4115       41154

You are WRONG and so you
lose 10 points


Your score is -10
STAND BY FOR A NEW TEST
```

And this is the listing for it:

```
  10 PRINT "  PRACTICE FOR FASTE
R READING"
  15 PRINT "  ------------------
----------"
  20 LET V=0
  30 PRINT AT 10,0;"PLEASE INDIC
ATE IF YOU WANT          NUMBERS
 OR LETTERS";AT 15,5;"ENTER N OR
 L"
  40 INPUT D$
  60 INPUT "ENTER A NUMBER BETWE
EN 1 AND 9  FOR THE SPEED OF YOU
R TEST.      1 IS THE FASTEST  ";
Z
  65 IF Z<1 OR Z>9 THEN GO TO 60
  70 IF D$="L" THEN GO TO 600
 210 PAUSE 80: CLS : PAUSE 10
 230 LET M=INT (RND*((10↑5)))
 232 LET P=INT (RND*20)
 234 LET Q=INT (RND*20)
 240 PRINT AT P,Q;M
 250 PAUSE 25*Z
 260 CLS
 270 PRINT ''"TYPE IN THE NUMBER
S YOU SAW"
 280 INPUT A
 290 PRINT AT 10,0;A;TAB 10;M
 300 IF A=M THEN LET V=V+10: PRI
NT '' INK 2; PAPER 6; FLASH 1;"Y
ou are correct and you score  "
; INVERSE 1;10; INVERSE 0;" poin
ts"
 310 IF A<>M THEN LET V=V-10: PR
INT '' INK 1; PAPER 7; FLASH 1;"
You are "; INVERSE 1;"WRONG"; IN
VERSE 0;" and so you";"lose ", I
NVERSE 1;10; INVERSE 0;" points"
 370 PRINT AT 18,0;"Your score i
s ";V
 380 IF V=-30 THEN PRINT "YOU'VE
 HIT ROCK BOTTOM!": STOP
```

```
 385 IF V=50 THEN PRINT "YOU'VE
PASSED WITH FLYING","COLOURS!":
STOP
 390 PRINT '"STAND BY FOR A NEW
TEST"
 400 PAUSE 300
 410 GO TO 210

 600 REM LETTERS
 610 PAUSE 140: CLS : PAUSE 10
 620 LET A$=CHR$ (INT (RND*26)+6
5)+CHR$ (INT (RND*26)+65)+CHR$ (
INT (RND*26)+65)+CHR$ (INT (RND*
26)+65)
 630 LET P=INT (RND*20)
 640 LET Q=INT (RND*20)
 650 PRINT AT P,Q;A$
 660 PAUSE 25*Z
 670 CLS
 680 PRINT "NOW ENTER THE LETTER
S YOU SAW"
 690 INPUT B$
 700 IF CODE B$<65 OR CODE B$>90
 OR LEN B$<>4 THEN GO TO 690
 710 PRINT '"My letters were ";
A$'"Yours were ";B$
 720 IF A$=B$ THEN LET V=V+10: P
RINT '"Yes, you are right"'"Yo
u score "; FLASH 1;10; FLASH 0;"
 points!"
 730 IF A$<>B$ THEN LET V=V-10:
PRINT '"No, you are wrong"'"Yo
u lose "; FLASH 1;10; FLASH 0;"
points!"
 735 PRINT '"You have "; BRIGHT
1;V; BRIGHT 0;" points"
 740 IF V=-30 THEN PRINT "YOU'VE
 HIT ROCK BOTTOM!": STOP
 750 IF V=50 THEN PRINT "YOU'VE
PASSED WITH FLYING","COLOURS!":
STOP
 770 PRINT '"Stand by for a new
 test"
 900 GO TO 610
```

CHAPTER NINE

Error trapping

Good computer programs contain mug-traps. Mug-traps are devices inserted in programs after inputs, designed to reject the entry of *inappropriate* data.

Many computer programs will crash if through operator error, or malevolence, the wrong kind of information is entered. For example, if a program expects a numeric input, and it is given a letter which has not been previously assigned as a variable name, the Spectrum will stop with a 'variable not found' report message.

There is an old saying about how difficult it is to make things foolproof 'because fools are so resourceful'. Unfortunately, this is the way the world is made. No matter how carefully you try to put mug-traps into your programs, you'll probably find some student manages to find a way to make the program crash.

The 'Molecular weight calculations' program in the first graphics chapter of this book contains a good example of error-trapping in lines 16ø through to 17ø. Here is the relevant section:

```
150 LET ANS=VAL (B$(X))*Y
160 INPUT C$
165 IF C$="" THEN GO TO 160
170 IF CODE C$<49 OR CODE C$>58
THEN GO TO 160
180 IF VAL C$<ANS - 0.05 OR VAL
C$ >ANS + 0.05 THEN GO TO 250
200 CLS
```

Although the computer wants, eventually, a numerical answer, it asks for a string. As you learned earlier in the book, the Spectrum can convert a string into its numerical equivalent by use of the function VAL. It is easier to reject string input than numeric input. Whereas entering a letter when numerical input is expected will either cause the computer to crash immediately, or to carry on with what

might well be a totally arbitrary value, a string input can be carefully checked before it is accepted.

In the program fragment printed above, line 165 rejects any 'non-input'. That is, if ENTER is just pressed without an answer being entered, line 165 will immediately go back to 16Ø for a new answer. The next line, 17Ø, checks the CODE of the entered string, and if this number does not lie between 49 (the CODE of '1') and 58, (the code of '9') the input is rejected. This, at least, rejects input which does not start with a number. It does not, and this is where the resourcefulness of fools' comes into play, reject input of the form '12z' or '9tricky1'.

Matchsticks

The next program – Matchsticks – is a variation on the old Nim games, in which players take it in turn to take matches away from a pile of them, with a limit on how many can be taken each time, with the loser being the player who is forced to take the last one.

It is included in this chapter as it contains some traps to catch bad input from the player. It is a little more difficult to crash this program than it is to crash the Molecular Weights one.

Enter it, and run it a few times, and then we'll discuss ways of making the program more robust:

```
            Matchsticks

The most you can take is 6

 1  2  3  4  5  6  7  8  9  10
 11  12  13  14  15  16
 17  18  19  20
 21

OK, you take 3

I'll take 4
```

```
  10 REM Matchsticks
  20 RANDOMIZE
  30 CLS
  40 LET M=0
  50 LET E=0
```



```
  60 LET Z=INT (RND*8)+16
  70 IF 2*INT (Z/2)=Z THEN LET Z
=Z+1
  80 LET H=3+INT (RND*4)
  90 CLS
 100 PRINT AT 3,10;"Matchsticks"
 110 PRINT AT 5,0;"The most you
can take is "; FLASH 1;H
 120 GO SUB 320
 130 IF E>0 THEN PRINT AT 7,0;"Y
ou took "; FLASH 1;E; FLASH 0;"
and I took "; FLASH 1;Q
 135 PRINT
 140 FOR K=1 TO Z: BEEP .03,2*K
 150 PRINT K;" ";: IF RND>.8 THE
N PRINT
 160 NEXT K
 170 GO SUB 320
 180 INPUT "How many do you want
to take? ";E$
 185 IF LEN E$>1 THEN GO TO 180
 186 IF E$<"1" OR E$>"9" THEN GO
TO 180
 187 LET E=VAL E$
 190 IF E>H THEN PRINT "You can'
t take that many": GO TO 180
 200 IF E>Z THEN PRINT "There ar
e not that many left": GO TO 180
 210 PRINT : PRINT "OK, you take
";E
 220 LET Z=Z-E
 230 GO SUB 320
 240 IF Z<1 THEN CLS : PRINT AT
5,0;"You took the last one,","so
 I'm the winner!": PRINT : PRINT
 "Thanks for the game": STOP
 250 LET Q=Z-1-INT ((Z-1)/(H+1))
*(H+1)-INT (RND*2)+INT (RND*2)
 260 IF Q<1 OR Q>H THEN GO TO 25
0
 270 PRINT : PRINT "I'll take ";
Q
 275 GO SUB 350
 280 LET Z=Z-Q
 290 IF Z<1 THEN CLS : PRINT AT
5,0;"I took the last one","so yo
u are the winner!": PRINT : PRIN
T FLASH 1;"Well done": STOP
 300 GO TO 90
 320 REM Delay subroutine
 340 PRINT
 350 FOR R=1 TO 100: NEXT R
 360 RETURN
```

You'll see when you run it that each new screen includes the message "The most you can take is . . .". It is worth reminding program users of any limits which exist on their input. And it does not matter if they are warned of this limit several times during the running of a program. So line 11Ø

prints up this message every time, thus, hopefully, stopping wrong input before it begins.

Line 18Ø asks for the input from the player as a string. The computer knows (because the game is set up in this way) that the input must be a single digit, between 1 and H (which is assigned in line 8Ø):Line 185 checks the length of the entered string. If it is greater than one (as it would be if 2RD2 or 3CP0 was entered) this input would be rejected (although, as we saw above, it would have got past the previous program mentioned). Having passed this hurdle, the entered string is checked for 'size'. Sinclair BASIC allows comparisons of 'greater than' and 'less than' to be made between strings. If the string is less than '1' or greater than '9' the input will be rejected. This effectively gets rid of all non-numeric input, as it will also reject a null-string (which you get when you simply press ENTER without previously entering anything). It will also reject a press from the SPACE/BREAK key.

Finally, in line 19Ø (after E$ has been turned into a numeric variable, E, by line 187) the value of the entered number is checked against the upper limit, H, to see if the entered number is acceptable.

While the program is far from perfect it does demonstrate the degree of robustness which you should aim at in developing programs for school use. Once you have your program running, and have made its display and 'conversation' as pleasant as possible, you may well wish to have a look at all the inputs, and make sure they have at least some degree of robustness, so a mistake (or a wilfully wrong entry) does not bring the whole lesson to an abrupt halt.

CHAPTER TEN

Multiple-choice quiz programs

There is a tendency to look upon the Spectrum and other microcomputers in schools only in the light of what they can do which is now done in another way. It takes a mental jump to look for things which are not being done in your class right now, but which can be implemented relatively easily, now that you have a Spectrum on hand.

Multiple-choice quiz programs are among the easiest to write, and because of this, predominate in commercially available software. As well, multiple-choice programs are those which are most generally criticised, as being a waste of computer potential. But this need not be the case.

Much software of this type does deserve criticism, because it is too limited, and perhaps not closely enough linked with class material. We have a major program in this chapter – Multiple Choice Master – which provides you with a program which will generate up to 1ØØ questions (on a 48K Spectrum, with a maximum of 15 on the 16K machine), each of which can support up to six different answers to choose from.

As well, the number of answers to choose from can change from question to question. The program has been deliberately written to be as flexible as possible, to allow you to use it in whichever subjects you choose. The program supports a full sentence question in each case, thus getting around one deficiency of many published programs in which the 'question' is reduced to 'Synonym for . . .?'.

This program has been proved very useful in practice, as it can be used for practically any subject, and for students of any age. It is self-prompting, and you should have little trouble in using it, and adapting it where needed, for your own subjects. The screen displays during creation of a test are designed to lead through the steps required as you can see from these samples:

```
          WORLD CAPITALS

HOW MANY QUESTIONS? (MAX. 100)

ENTER QUESTIONS,ANSWERS AND THE
CORRECT CHOICE AS REQUESTED,ANY
ERRORS MAY BE EDITED LATER.

MAXIMUM LENGTHS = 60 CHARACTERS
PER QUESTION, AND 50 CHARACTERS
PER ANSWER.
Press ENTER


QUESTION NO. 1

NO. OF CHOICES? (MAX. 6)
WHAT IS THE CAPITAL OF AUSTRALIA

ANSWER(A)MELBOURNE

ANSWER(B)SYDNEY

ANSWER(C)CANBERRA

CORRECT CHOICE?


QUESTION NO. 2

NO. OF CHOICES? (MAX. 6)
NAME THE CAPITAL OF ROMANIA

ANSWER(A)BUCHAREST

ANSWER(B)MADRID

CORRECT CHOICE?



QUESTION NO. 3

NO. OF CHOICES? (MAX. 6)
WHAT IS UGANDA'S CAPITAL CALLED?

ANSWER(A)KAMPALA

ANSWER(B)HANOI

ANSWER(C)KABUL

CORRECT CHOICE?
```


Once you have entered all your questions, you can check the program. Again, this feature is self-prompting, as you can see:

```
PROGRAM CHECK

QUESTION NO. 1
WHAT IS THE CAPITAL OF AUSTRALIA

(A)MELBOURNE

(B)SYDNEY

(C)CANBERRA

CORRECT CHOICE=C

Press ENTER TO CONTINUE,OR PRESS
  'E' TO EDIT
```

Finally, when a student sits down to use the program, the screen display is as follows, so there is little chance of error:

```
HELLO, TIM I AM GOING
TO ASK YOU SOME QUESTIONS AND
I WANT YOU TO CHOOSE THE
ANSWER WHICH YOU THINK IS
CORRECT, AND PRESS THE LETTER
SHOWN NEXT TO IT
Press ENTER
```

Here is the complete listing of the program:

```
   5 REM MULTIPLE CHOICE MASTER
  10 LET M=1
  20 PRINT "TITLE? (MAX. 20 CHR$
)"
  30 INPUT A$
  40 LET A=LEN A$+1
  50 LET B=((30-A)/2)
  60 GO SUB 1130
  70 PRINT "HOW MANY QUESTIONS?
(MAX. 100)"
  80 INPUT E: PRINT E
  90 DIM B$(E,60)
 100 DIM C$(E,50)
 110 DIM D$(E,50)
 120 DIM E$(E,50)
 130 DIM F$(E,50)
 140 DIM G$(E,50)
 150 DIM H$(E,50)
 160 DIM J$(E,50)
 170 DIM F(E)
 180 PRINT '"ENTER QUESTIONS,ANS
WERS AND THE CORRECT CHOICE AS R
```

```
EQUESTED,ANY ERRORS MAY BE EDITE
D LATER."
 190 PRINT '"MAXIMUM LENGTHS = 6
0 CHARACTERS PER QUESTION, AND 5
0 CHARACTERS PER ANSWER."
 200 GO SUB 1210
 210 INPUT Z$: CLS
 220 FOR D=1 TO E
 230 PRINT "QUESTION NO. ";D
 260 PRINT '"NO. OF CHOICES? (M
AX. 6)"
 270 INPUT F(D): PRINT F(D)
 280 PRINT AT 4,0;"ENTER QUESTIO
N (MAX. 60 CHARS)"
 290 INPUT B$(D)
 300 PRINT AT 4,0;B$(D);"     "
 310 FOR G=0 TO F(D)-1
 320 PRINT "ANSWER";
 330 GO SUB 710+40*G
 340 NEXT G
 350 PRINT "CORRECT CHOICE?"
 360 INPUT J$(D)
 370 CLS : NEXT D
 380 FOR D=1 TO E
 390 GO SUB 1130
 400 PRINT "PROGRAM CHECK"
 410 PRINT "
 420 GO SUB 950
 430 PRINT "CORRECT CHOICE=";J$(
D)
 440 GO SUB 1210
 450 PRINT "TO CONTINUE,OR PRESS
 'E' TO EDIT";
 460 INPUT Z$
 470 IF Z$="E" OR Z$="e" THEN CL
S : GO TO 230
 480 NEXT D
 490 CLS
 500 PRINT "EDIT FACILITIES END,
 DUMMY RUN  FOLLOWS"
 510 PAUSE 250
 520 LET S=0
 530 GO SUB 1130
 540 PRINT "HELLO, I'M SPECTRUM.
 WHAT'S  YOUR NAME?"
 550 INPUT K$
 560 CLS : PRINT "HELLO, ";K$;"
I AM GOING","TO ASK YOU SOME QUE
STIONS AND  I WANT YOU TO CHOOS
E THE  ANSWER WHICH YOU TH
INK IS  CORRECT, AND PRESS
THE LETTER  SHOWN NEXT TO IT"
 570 GO SUB 1210
 580 INPUT Z$
 590 FOR D=1 TO E
 600 GO SUB 1130
 610 PRINT AT 2,0;K$;" ";
 620 GO SUB 950
 630 PRINT "ANSWER=(?)"
 640 INPUT L$
 650 IF L$=J$(D) THEN GO TO 1270
```



```
 660 GO SUB 1230
 670 PRINT "PLEASE WAIT A MOMENT
."
 680 PAUSE 500
 690 NEXT D
 700 GO SUB 1310
 710 PRINT "(A)";
 720 INPUT C$(D)
 730 PRINT C$(D)
 740 RETURN
 750 PRINT "(B)";
 760 INPUT D$(D)
 770 PRINT D$(D)
 780 RETURN
 790 PRINT "(C)";
 800 INPUT E$(D)
 810 PRINT E$(D)
 820 RETURN
 830 PRINT "(D)";
 840 INPUT F$(D)
 850 PRINT F$(D)
 860 RETURN
 870 PRINT "(E)";
 880 INPUT G$(D)
 890 PRINT G$(D)
 900 RETURN
 910 PRINT "(F)";
 920 INPUT H$(D)
 930 PRINT H$(D)
 940 RETURN
 950 PRINT "QUESTION NO. ";D
 960 PRINT B$(D)
 970 FOR G=0 TO F(D)-1
 980 GO SUB 1010+20*G
 990 NEXT G
1000 RETURN
1010 PRINT "(A)";C$(D)
1020 RETURN
1030 PRINT "(B)";D$(D)
1040 RETURN
1050 PRINT "(C)";E$(D)
1060 RETURN
1070 PRINT "(D)";F$(D)
1080 RETURN
1090 PRINT "(E)";G$(D)
1100 RETURN
1110 PRINT "(F)";H$(D)
1120 RETURN
1130 CLS
1140 PRINT AT 0,B;A$
1150 PRINT AT 1,B-1;"
1160 FOR C=1 TO A
1170 PRINT "
1180 NEXT C
1190 PRINT
1200 RETURN
1210 PRINT "Press "; INVERSE 1;"
ENTER"; INVERSE 0;" ";
1220 RETURN
1230 PRINT "That is wrong ";K$;"
.","THE CORRECT ANSWER IS ";J$(D
)
```

```
1240 GO SUB 1210
1250 INPUT Z$
1260 RETURN
1270 LET S=S+1
1280 PRINT "WELL DONE ";K$;" THA
T","IS THE CORRECT ANSWER, SO YO
U","SCORE A POINT"
1290 GO TO 670
1300 GO SUB 1130
1310 PRINT K$;", YOU GOT ";S;" Q
UESTIONS","RIGHT OUT OF ";E;". P
LEASE","TELL YOUR TEACHER"
1320 IF M=1 THEN GO TO 1440
1330 PRINT AT 21,0;"N = NEXT CYC
LE"
1340 FOR Y=1 TO 10
1350 IF INKEY$="N" OR INKEY$="n"
 THEN GO TO 520
1360 NEXT Y
1370 PRINT AT 21,0;"
  "
1380 FOR Y=1 TO 10
1390 IF INKEY$="N" OR INKEY$="n"
 THEN GO TO 520
1400 NEXT Y
1410 GO TO 1330
1420 SAVE "TEST"
1430 PRINT "SAVE AGAIN ? (Y/N)"
1440 INPUT W$
1450 IF W$="Y" OR W$="y" THEN GO
 TO 1420
1460 LET M=0
1470 GO TO 520
```


CHAPTER ELEVEN

Other programs of interest

In this chapter, we'll look at several programs which, although they are not linked directly with a particular subject, are of interest. You may well be able to use them in your classes.

Histograms and Bar Charts

Lines or columns of different lengths are convenient ways of displaying information. It is simpler to show the information in bars which run across the page, and we will look at a program which does this first. However, it is not too difficult – using the Spectrum's PRINT AT – to produce a graph in which the bars are printed vertically, and our second program does this.

The first program plots the frequency with which numbers in the range 1 to 20 are generated by the random number function on the Spectrum. Here is the listing:

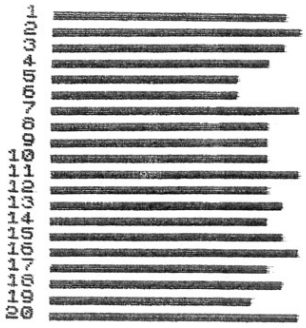
```
  10 REM  HISTOGRAMS
  20 DIM A(20)
  25 PRINT "Please stand by"
  30 FOR D=1 TO 300
  40 LET C=INT (RND*20)+1
  50 IF A(C)<30 THEN LET A(C)=A(
C)+1
  60 NEXT D
  70 REM   PRINT OUT
  75 CLS
  80 FOR B=1 TO 20
  90 IF B<10 THEN PRINT " ";
 100 PRINT B;" ";
 110 FOR C=1 TO A(B)
 120 PRINT "■";
 130 NEXT C
 140 PRINT
 150 NEXT B
```

Here is a run with a program in its present form:

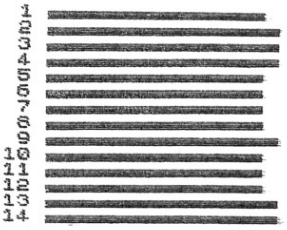
As you know, if the random number generator was working perfectly, and we had an infinite numer of trials, the frequency of each number generated would be equal. I rewrote the program so, instead of simply generating 300 numbers as the first one did, it would generate 3000. This is the altered listing:

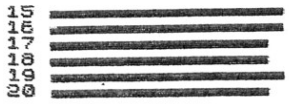
```
 10 REM  HISTOGRAMS
 15 REM  LARGER SAMPLE
 20 DIM A(20)
 25 PRINT "Please stand by"
 30 FOR D=1 TO 3000
 40 LET C=INT (RND*20)+1
 50 IF A(C)<300 THEN LET A(C)=A
(C)+1
 60 NEXT D
 70 REM   PRINT OUT
 75 CLS
 80 FOR B=1 TO 20
 90 IF B<10 THEN PRINT " ";
100 PRINT B;" ";
110 FOR C=1 TO A(B)/10
120 PRINT "■";
130 NEXT C
140 PRINT
150 NEXT B
```

And here is the result of two runs of the program. As you can see, the distribution of numbers more closely approaches the theoretical distribution:



Finally (and this called for some patience for the run to end), I modified it to generate 10,000 numbers, and produced this histogram:





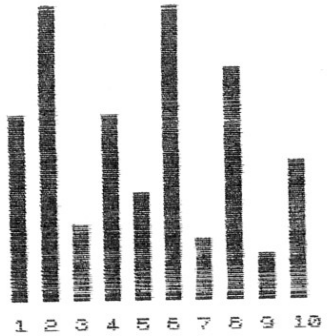
Column Graph

The next program – Column Graph – generates a bar graph in the way in which, perhaps, we expect to see it, with the origin in the bottom left hand corner. This program allows you to select the number of columns of data you want (up to 15) and then gets you to enter the data for each item on the graph.

This is the program listing:

```
   5 REM COLUMN GRAPH
   7 INPUT "HOW MANY COLUMNS? (U
P TO 15)";Q
   9 IF Q>15 THEN GO TO 7
  10 DIM A(Q)
  20 FOR G=1 TO Q
  30 INPUT A(G)
  35 IF A(G)>19 THEN LET A(G)=19
  40 NEXT G
  50 FOR G=1 TO Q
  55 PRINT AT 21,2*G;G
  60 FOR N=1 TO A(G)
  70 PRINT AT 20-N,2*G;"■"
  80 NEXT N
  90 NEXT G
```

Here's one run from it:



It may be interesting to modify this program so it produces a graph of the relative distribution of the numbers generated by the Spectrum's random number generator.

Sorting Routines

Next we have three sorting routines. The first one simply sorts numbers into order. If you wish to reverse the order of the final sequence, change the >= in line 8Ø into <=:

```
   5 REM NUMBER SORT
   7 INPUT "HOW MANY NUMBERS TO
SORT? ";N
  10 DIM A(N)
  20 FOR J=1 TO N
  30 INPUT "ENTER NUMBER ";(J);"
";A(J)
  40 PRINT A(J);" ";
  50 NEXT J
  55 CLS
  60 FOR Z=1 TO N
  70 FOR J=1 TO N-Z
  80 IF A(J)>=A(J+1) THEN GO TO
120
  90 LET T=A(J)
 100 LET A(J)=A(J+1)
 110 LET A(J+1)=T
 120 NEXT J
 150 PRINT A(J);" ";
 160 NEXT Z
```

The second sort puts strings into alphabetical order, as the sample run illustrates:

```
THIS                    IS
A                       TEST
TO                      ENSURE
THE                     PROGRAM
WORKS                   SATISFACTORILY
PLEASE STAND BY         FOR SORTED LIST



A                       ENSURE
IS                      PROGRAM
SATISFACTORILY          TEST
THE                     THIS
TO                      WORKS
```

Here is the listing (and note that input must follow the same format, that is, it must be all in upper or lower case, or if the first letter of any word is upper case with the rest in lower case, the first letter of all words must be in upper case with the rest of all words in lower case):

```
   5 REM ALPHABET SORT
   7 REM ALL INPUT MUST BE
         EITHER IN CAPS OR
         SMALL LETTERS
  10 INPUT "How many names to so
rt? ";X
  20 INPUT "And what is the leng
th of the","longest word (defaul
t is 15)";Y$: IF Y$="" THEN LET
Y$="15"
  25 LET Y=VAL Y$
  30 DIM A$(X,Y)
  40 FOR N=1 TO X
  50 INPUT INVERSE 1;"Enter word
 number ";(N);A$(N)
  60 PRINT A$(N),
  70 NEXT N
  80 FOR M=1 TO X-1
  90 FOR N=1 TO X-M
 100 IF A$(N)<=A$(N+1) THEN GO T
O 140
 110 LET B$=A$(N)
 120 LET A$(N)=A$(N+1)
 130 LET A$(N+1)=B$
 140 NEXT M
 150 NEXT N
 160 PRINT '"PLEASE STAND BY FOR
 SORTED LIST": PAUSE 200: CLS
 170 FOR N=1 TO X
 180 PRINT A$(N),
 190 NEXT N
```

Finally, we have an age order sort, in which names and ages are entered, and sorted in ascending age. This sort can easily be adapted for any situation where two variables are linked, and you want them sorted in terms of one of the variables:

```
Age Name
14 Jones
16 Smith
12 Harrison
104 Hartnell



Hartnell                      104
Smith                         16
Jones                         14
Harrison                      12
```



```
   5 REM  AGE ORDER SORT
   6 REM   OF X NAMES
  10 INPUT "How many names to so
rt? ";X
  20 DIM N$(X,26)
  30 DIM A(X)
  35 PRINT "Age Name"
  40 FOR J=1 TO X
  50 INPUT "Enter name number ";
(J);N$(J)
  60 INPUT "And how old is ";(N$
(J));A(J)
  70 PRINT A(J);" ";N$(J)
  80 NEXT J
  90 FOR Z=1 TO X-1
 100 FOR J=1 TO X-Z
 110 IF A(J)>=A(J+1) THEN GO TO
180
 120 LET T$=N$(J)
 130 LET N$(J)=N$(J+1)
 140 LET N$(J+1)=T$
 150 LET T=A(J)
 160 LET A(J)=A(J+1)
 170 LET A(J+1)=T
 180 NEXT J
 190 NEXT Z
 200 CLS
 205 FOR J=1 TO X
 210 PRINT N$(J);A(J)
 220 NEXT J
```

Comparing unlike quantities

This program arose from a question sent to my column 'Response Frame' in *Your Computer* magazine, when a reader in New Zealand said he wanted a program which would work out the time difference between two times (such as 9.17 am and 3.34 pm) and work out which was earlier (or later). The final program is of little intrinsic value (because a routine to detect 'AM' and 'PM' is all that is needed).

However, it is included here because it gives an insight into comparing quantities which are made up from units which do not necessarily compare easily (like pounds with ounces). It is important to remember, in any program in which you are comparing quantities which are made up of units with different values (such as pints and gallons, or hours and minutes), that you must render the values to be compared down to a single unit. You must also establish a common input format, so the computer knows what to expect.

```
  10 DIM A$(2,8): DIM A(2)
  20 FOR Z=1 TO 2
```

```
  30 INPUT A$(Z)
  40 LET A(Z)=VAL (A$(Z)( TO 5))
  50 IF A$(Z,7)="P" THEN LET A(Z
)=A(Z)+12
  60 LET B=INT (A(Z))
  70 LET A(Z)=60*B+100*A(Z)-60*B
  80 NEXT Z
  90 IF A(1)>A(2) THEN PRINT A$(
1);" IS LATER THAN ";A$(2)
 100 IF A(1)<A(2) THEN PRINT A$(
1);" IS EARLIER THAN ";A$(2)
```

This can be done fairly easily with the ZX Spectrum, because of its simple string handling (discussed in detail in an earlier chapter in this book). The program here for comparing times needs input in the form 09.37.AM or 12.04.PM. It then (line 4Ø) works out the value (as a number with a decimal point) of the time, adds 12 to the whole number to the left of the decimal point if the seventh element of the input string is a 'P' and then converts the number into minutes, comparing the two minutes' totals. The original strings are then used to print out the final information.

Super Sketch

This fine program, written by Gwyn Dewey, assists you to create pictures on the Spectrum using keys as indicated in the instructions provided by lines 6Ø to 13Ø. The keys zero to six control the colour. DRAW is controlled by the 'Q', 'W', 'E', 'A', 'D', 'Z', 'X' and 'C' keys. '8' erases and 'Ø' cancels, 'B' controls the brilliance and '9' sends a copy of the screen to the ZX Printer.

You exit from the program with the 'V' key, and you can SAVE your artwork on tape with 'K' and retrieve it with 'J'. Note that the program expects all input to be in lower case letters:

```
   1 REM sketch by G.Dewey
  10 LET a=100
  15 LET aa=0
  20 LET b=100
  30 LET c=0
  32 LET d=0
  33 LET z=0
  35 CLS
  40 PRINT "Super Sketch"
  50 REM Author G.Dewey
  60 PRINT : PRINT "This program
 helps you to draw  pictures usi
ng the following     keys.NOTE:on
```



```
ly one colour is     allowed in a
ny one square"
  70 PRINT '"0-6 colour control"
  80 PRINT '"Q,W,E,A,D,Z,X,C DRA
W"
  90 PRINT '"8 erases o cancels"
 100 PRINT '"B controls brillian
ce"
 110 PRINT '"9 sends a copy of t
he screen to the ZX PRINTER"
 120 PRINT '"V exits"
 130 PRINT '"K saves picture(J l
oads picture)"
 140 IF INKEY$="" THEN GO TO 140
 150 CLS
 151 POKE 23658,0
 154 INVERSE 0: PLOT a,b
 155 INVERSE aa: PLOT a,b
 156 LET b$=INKEY$
 157 IF b$="" THEN GO TO 154
 160 LET a=a-(b$="q" OR b$="a" O
R b$="z")+(b$="e" OR b$="d" OR b
$="c")
 170 LET b=b-(b$="z" OR b$="x" O
R b$="c")+(b$="q" OR b$="w" OR b
$="e")
 210 IF b$>="0" AND b$<="7" THEN
 INK (VAL b$)
 215 IF b$="8" THEN LET aa=1
 216 IF b$="o" THEN LET aa=0
 220 IF b$="b" AND d=0 THEN LET
d=1
 230 IF b$="b" AND d=1 THEN LET
d=0
 240 BRIGHT d
 250 IF b$="9" THEN COPY
 260 IF b$="v" THEN GO TO 9100
 270 IF b$="k" THEN SAVE "pictur
e"SCREEN$
 280 IF b$="j" THEN LOAD "pictur
e"SCREEN$
 290 GO TO 154
```

Interior angle of a regular polygon

This program works out the interior angle of a regular polygon, rounding the answer if needed (see sample run for seven sides) to two decimal places (it is interesting to try this for an imaginary polygon with just one or two sides):

```
A regular polygon of 3 sides
has interior angles of 60

A regular polygon of 4 sides
has interior angles of 90

A regular polygon of 5 sides
has interior angles of 108
```

```
A regular polygon of 6 sides
has interior angles of 120

A regular polygon of 7 sides
has interior angles of 128.57

A regular polygon of 8 sides
has interior angles of 135

A regular polygon of 9 sides
has interior angles of 140
```

```
  10 REM    INTERIOR ANGLE OF
  20 REM    A REGULAR POLYGON
  30 INPUT "How many sides? ";SI
DES
  40 LET SIDES=INT (SIDES)
  50 LET ANGLE=INT (100*(180-360
/SIDES))/100
  60 PRINT '"A regular polygon o
f ";SIDES;" sides"
  70 PRINT "has interior angles
of ";ANGLE
  80 GO TO 30
```

Straight line depreciation

The program is self-prompting. Note that line 28Ø (POKE 23692,−1) stops the computer from asking *Scroll* when the screen is full. Here's the program in action:

```
Purchase price is £759.75

Life of asset is 8 years

It depreciates £94.96 a year

Year          Worth
1984          £759.75
1985          £664.79
1986          £569.83
1987          £474.87
1988          £379.91
1989          £284.95
1990          £189.99
1991          £95.03
```

And this is the listing for it:

```
  10 REM STRAIGHT LINE
  20 REM  DEPRECIATION
  30 LET YEAR=0
  50 INPUT "Enter purchase price
";PRICE
  60 PRINT '"Purchase price is £
";PRICE
 100 INPUT "Enter life of asset
(in years) ";LIFE
 130 PRINT '"Life of asset is ";
LIFE;" years"
 140 LET DEPREC=(INT (PRICE*100/
LIFE))/100
 150 PRINT '"It depreciates £";D
EPREC;" a year"
 180 INPUT "Enter first year of
use",'"(as 1984) ";YEAR
 220 PRINT '"Year";TAB 12;"Worth
"
 240 PRINT YEAR;TAB 11;"£";PRICE
 250 LET PRICE=PRICE-DEPREC
 260 IF PRICE<1 THEN STOP : REM
     OR THEN COPY: STOP
 270 LET YEAR=YEAR+1
 280 POKE 23692,-1
 290 GO TO 240
```


Day of the week

This simple routine works out what day of the week a specified date falls on, as you can see from this sample run:

```
19/12/84 - Wed


30/4/83 - Sat

25/12/83 - Sun

1/1/99 - Tue
```

This is the listing:

```
  10 REM DAY OF THE WEEK
  20 LET A$="..MonTueWedThuFriSa
tSun"
  30 INPUT "Enter month (as 7) "
;M
  40 IF M<1 OR M>12 THEN GO TO 3
0
  50 INPUT "Enter day (as 23) ";
D
  60 IF D<1 OR D>31 OR M=2 AND D
>29 THEN GO TO 50
  70 INPUT "Enter year (as 1984)
";Y
  80 LET Q=Y+(M<3)
  90 LET K=Q/100
 100 LET T=M-12*(M<3)
 110 LET R=INT (13*(T+1)/5)+INT
(5*Q/4)-INT (K)+INT (K/4)+D+5
 120 LET R=R-(7*INT (R/7))+1
 130 PRINT TAB 6;D;"/";M;"/";Y-1
900;" - ";A$(R*3 TO R*3+2)
```

Seconds timer

Actually, the accuracy of this timer really precludes its use other than for demonstration purposes, but it succeeds admirably for that. The plotted point (see between the 5 and the 1Ø on the sample printout) moves around the circle, taking approximately a minute to make the circuit.

```
          60
     55         5


 50                  10


45                     15


 40                  20


     35         25
          30
```

The program prints out the numbers, and then stops (using PAUSE Ø in line 4Ø) to wait for a key press, at which it begins processing. If you want to make the timer more accurate, reduce the 39 in line 1ØØ to a 38, and add a line 1Ø5 in which the computer must do some calculation (such as raising a number to a power) which will slow it down enough to slow the time down. Your students will probably enjoy calibrating the timer accurately.

Here's the listing for that program:

```
  5 REM SECONDS TIMER
  10 FOR N=0 TO 60 STEP 5
  20 PRINT AT 10-10*COS (N/30*PI
),10+10*SIN (N/30*PI);N
  30 NEXT N
  40 PAUSE 0
  50 FOR T=0 TO 59
  60 PLOT 88+59*SIN (T/30*PI),90
+61*COS (T/30*PI)
  100 PAUSE 39
  110 PLOT OVER 1;88+59*SIN (T/30
*PI),90+61*COS (T/30*PI)
  120 NEXT T
  130 GO TO 50
```


Mean, standard deviation, variance

Finally, in this chapter, we have a program to determine mean, standard deviation and variance. The program is self-prompting, and the output is simple to interpret. You may wish to add a routine, similar to that used in the straight line depreciation program to limit the output to two decimal places. Here is a sample run:

```
24              4
25              3
26              2
23              3
22              2
21              1
27              1

The mean is 24
Variance is 2.5
Standard deviation is 1.5811388

230             7
228             9
235             3
236             1
225             2

The mean is 229.68182
Variance is 8.853302
Standard deviation is 2.9754499
```

This is the listing:

```
10 REM MEAN, STANDARD
20 REM DEVIATION,
30 REM VARIANCE
35 INPUT "How many items? ";N
40 DIM A(N)
50 DIM B(N)
80 FOR Z=1 TO N
90 POKE 23692,-1
```

```
 100 INPUT "Enter item ";(Z);" "
;A(Z)
 140 PRINT ;A(Z);
 180 INPUT "Enter frequency ";B(
Z)
 190 PRINT B(Z)
 200 NEXT Z
 270 LET M=0
 280 LET A=0
 290 LET B=0
 300 FOR Z=1 TO N
 310 LET M=M+B(Z)
 320 LET A=A+B(Z)*A(Z)
 330 LET B=B+B(Z)*A(Z)*A(Z)
 340 NEXT Z
 380 PRINT '''"The mean is ";A/M
 390 PRINT '"Variance is ";B/M-A
*A/(M*M)
 400 PRINT '"Standard deviation
is ";SQR (B/M-A*A/(M*M))
 410 STOP
1000 LET P=0
1010 LET Q=0
1020 LET L=0
```


CHAPTER TWELVE

Evaluating Software for the Spectrum

The Spectrum is a delightfully simple, but powerful, machine to use and to write programs for. Unfortunately, the market does not seem large enough to attract the major software houses into writing anything but games and utility-type programs. There are many of the 'quiz' programs, but many of these are of little serious educational value. It would be a terrible waste of a micro's immense power if it were only to be used for this kind of program.

Therefore, it is necessary for teachers to write software and to build up resource banks of programs which are tailor-made to suit the particular needs of an individual school. The ideal situation would be for teachers to become programmers, and for programmers to become teachers! In the meantime, the pressure is on the teaching profession to adapt to, and to utilise, the new technology as rapidly as possible. One difficulty here is that the vast majority of teachers, as I'm sure you realise, are newcomers to computing and are only just beginning to develop their programming skills. It is far too much to expect them to produce programs which are complex and robust enough to be of general use.

It is also unfortunate that far too many excellent ideas are lost simply because the program lacks sophistication, perhaps in error-trapping (discussed elsewhere in this book) or in screen layout. There are many other teachers who are not at all interested in programming, and simply wish to use computers as they would an overhead projector or a video recorder. Others still remain highly sceptical as to whether computers have any place in a classroom at all.

So where are the programs to come from for the immediate future? Fortunately, there are a few software houses which are producing good educational software for the Spectrum. Much of it is written by experienced teachers.

As a teacher, it is up to you to decide whether the Spectrum will be of use in a particular lesson, and then to decide what would be a good package to choose. As is valid for any classroom aid, the program should be at least as good as existing methods of covering the work in question, whether it be amplifying a particular point, or testing knowledge of a specific area of a subject. Another problem then comes to mind: Can the whole class participate in the use of the program at the same time?

With many schools able to afford only a small number of microcomputers, it can be hard to avoid the temptation to use them with large numbers of pupils at a time simply to exploit the computers' gimmick value. To enable an entire class to use a computer at once, you need a display which can be seen clearly by all those involved in the lesson. There is a distinct advantage to be gained from using a new and existing technology in the classroom, but not when the program chosen is only of marginal benefit.

It is far better to use a micro when it can do something which no other teaching method can. As I said in the introduction to this book, the colour facilities of the Spectrum, along with the computer's high-resolution graphics, make it ideal for producing and storing all manner of complex diagrams. This can be of great use particularly in subjects such as Biology and Child Welfare, and in the remedial department where large clear numbers and letters are required. It is extremely useful to have animated diagrams, particularly when the movement is under teacher control, so it can be halted when needed for explanation of what it is illustrating. A simplified flow diagram showing the process of photosynthesis, which seems the kind of program development you should aim at, is included in this book.

Once you've decided that it is appropriate to use computers in a lesson, what should you look for in commercial software? Here are five points to check.

1. The *purpose* of the program should be clearly stated. The package should contain information as to what age range it is suitable for, what prior knowledge is assumed before it can be used, and what *type* of program it is, i.e. whether it is an interactive program requiring input from students, or is a demonstration of a teaching point. The program could also be a testing program for just one student to use at a time.

2. The package should clearly state which machine (16K or 48K) the program demands, and should also point out if a printer is needed to get the maximum benefit from the program.

3. The packaging should clearly explain, as well as the points raised in (1) above, how to use the program, from loading it in the first place, to explaining what input is required when you've got it up and running.

4. The program itself should be fully error-trapped. There can be very few exceptions to this golden rule. For a program to accept an incorrect type of response – for example, to allow letters to be entered when a numerical input is required – and therefore crash can prove very disruptive in a lesson, similar in effect to a film breaking in a projector.

5. A program should, if possible, be capable of being adapted to the needs of a particular school and/or class. This is an ideal way for schools to obtain really useful software, provided that adapted programs are not then passed off as being original.

If all this sounds a little daunting, then help is at hand. Most of the really useful programs will have been evaluated at one time or another by experienced reviewers in one or more of the specialist magazines which deal with the Spectrum, such as *ZX Computing* and *Sinclair User*. They often contain programs which have been used in a classroom, and can therefore give an accurate assessment of usefulness. It is well worth buying these publications and keeping a record of any educational programs, articles or reviews.

It would also be useful to become a member of E.Z.U.G., the Educational ZX Users' Group (Eric Deeson, Highgate School, Birmingham 12), which is run under the auspices of M.U.S.E. The group only puts programs of an acceptable standard into its library. Therefore, you can buy the programs knowing that they have all been tried and tested. Now that there are a number of specialist retail outlets, it is becoming more and more possible to try out programs before purchasing them, or at least see them running. This is, of course, far more satisfactory that relying on information from advertisements or in catalogues. Most of the specialist shops are only too pleased to discuss the programs they sell. It is a good idea to call in frequently to

your local computer shop to see what is new, and what they recommend.

The questions you should ask yourself:

- Is it a suitable time in the teaching schemes to use a microcomputer?
  Is the program chosen relevant to the scheme of work?
- Can the television screen be seen clearly by everyone?
- Does the program have clear documentation which indicates the purpose of the program and shows how to use it?
- Is the program easy to use? Is it well error-trapped?
- Can it be adapted to the needs of a particular group?
- Is the package good value for money?

CHAPTER THIRTEEN

Using the Spectrum in Infant School, a case history

Christine S Johnson is headmistress of the Carlton Netherfield Infant and Nursery School in Netherfield, Nottingham. She has been using the Spectrum (and the ZX81) with children in the five to seven-and-a-half years old range for some time. In this section of the book, she explains how the computer became part of the curriculum, how it is used, and how the children respond to it.

Ask an elderly person if they can remember their very first classroom. They will probably describe a large room with drab walls and a high ceiling. There may have been steps at the rear of the room, and it's possible that the heavy wooden desks with iron legs were screwed to the floor. Perhaps there was a large open fireplace which the teacher had to keep supplied with coal. It was warm by the fire, but the children sitting at the other side of the room were shivering and their noses were blue. Every child had a blackboard, a piece of chalk, and a small scrap of waste material for 'rubbing out'. Books were few and silence was the rule.

How different from the lively, colourful, purposeful classrooms of today with cheerful pictures and meaningful displays. Children are encouraged to ask questions and to discover things for themselves, to use the attractive books and plentiful, stimulating apparatus wisely and well.

Just over thirty years ago, computers were extremely rare, were enormous, complicated pieces of machinery, cost thousands of pounds and were very limited in the way they could be used. At present, computers are plentiful, small, not too costly and very versatile. Their impact on the future of teaching is impossible to project at this time. Progress is so rapid that by the time the young children of today have grown old, almost everything in the world will have changed beyond recognition, and the computers in use now will certainly be museum pieces.

Today, however, they are fascinating, exciting, extremely useful items of equipment and it would not be fair to the children in our schools today to let them miss the opportunity of learning how to operate computers.

Most people would agree that it would not be possible for all children to become so competent that they could write complicated programs by the age of seven. However, if they are helped to use a keyboard, to follow clear instructions, to learn part of a simple computer language and to use programs connected with many aspects of the school curriculum, they will be ready and anxious to learn so much more by the time they reach the Junior School.

The main aims are encouragement and enjoyment. Today's children are computer-minded and are ready to receive all the assistance and information they can be offered.

Imagine that you are six years old. You watch the television at home each evening and enjoy what you see, but the following morning you have forgotten all about it because you were not directly involved. One day at school the teacher shows you a keyboard connected to the school television set, and tells you that you can make it work.

She asks you to touch a few keys and then to find the letters of your name. She helps you with a few more keys and suddenly your eyes sparkle as your name appears, not just once but twenty-two times.

```
10 BORDER 4
20 PAPER 7
30 PRINT "Dale Andrews"
40 PAUSE 80
50 GO TO 30
```

```
Dale Andrews
Dale Andrews
Dale Andrews
Dale Andrews
Dale Andrews
Dale Andrews
Dale Andrews
Dale Andrews
Dale Andrews
```

Other children have been watching and want to have a turn. Soon the teacher does not need to help at all because everyone is giving instructions: "Don't forget the quotation

marks", "Press ENTER at the end of the line" and so on.

One child holds a finger on a key for too long, and a whole row of letters comes onto the screen. Everyone looks worried, but the teacher says "We can always put right any mistakes" and shows how to press CAPS and DELETE. She shows you how to add two more lines to make a coloured screen and border.

Later she produces a card with simple instructions as a reminder, and asks you and your friends to help another group of children.

When school is over for the day you rush outside to a waiting parent, saying "Mummy, come and see what I've been doing. I can work a computer!"

What have the children learned from their first lesson?

1. They can make a computer work and that it is fun.
2. There is a 'safety code':
   - Only one child is allowed to use the keyboard at a time (although others can help with reading a program or spotting letters and symbols)
   - No child is allowed to unplug any leads or touch any switches (the tape recorder and printer are connected by the teacher before work begins)
   - If any real problem develops, the teacher must be informed
   - Anyone using the computer incorrectly, or upsetting anyone else who is using it, will be banned from using it for a few days
3. There is no need to be concerned if mistakes are made as they can be rectified
4. Instructions are given to the computer in lines, each of which begins with a number
5. It is possible to change from lower case letters to capitals by using CAPS SHIFT
6. It is possible to leave spaces between words
7. Quotation marks after the instruction PRINT are important
8. At the end of each line it is essential to press ENTER
9. Instructions may be changed to give different results by moving the cursor, CAPS and EDIT, for use in such things as names, colours and timing. At first, the children working on programs with me changed the whole line, but were then shown how to use the 'arrows' to help to change part of a line

10. It is possible to explain to someone else how to use the computer

How it started

It all began in 1981. In connection with a mathematics project we collected a box of 'interesting things'. A group of children showed a great deal of interest in such things as magnets and magnifying glasses, and we formed a lunch-time club. We soon progressed from a telephone made from yoghurt cartons and a piece of string to something more sophisticated which connected the Head Teacher's room and a corridor.

Our simple push-along car made from a construction kit soon changed to a battery-driven vehicle which reversed each time the end of a knitting needle struck a wall.

Any child in the school could join the club. They signed their names in the club book and wore small badges.

After a while, the box could no longer contain our collection as we added many more items, such as switches, batteries, wires and small bulbs, home-made musical instruments, calculators and (very important) two jumbo typewriters.

We also had a growing library of 'infant' type scientific books, so we had to use a large cupboard. We removed the doors so that all the items were easily accessible.

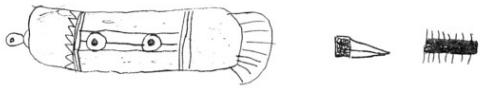
We acquired some valves and a TV repairman let us have a transistor and a microchip, and an interesting discussion developed. I asked a child to switch on and off an electric light, and then explained that a valve could work hundreds of times more quickly and a transistor even more rapidly.

We looked at the large television set and then a pocket-sized television with a tiny screen. The children were interested to know that computers were the size of a large room when their parents were little children.

They also had access to language masters, a 'Synchrofax' and various types of tape-recorder (reel to reel, cassette and a very small executive tape recorder). After the jumbo typewriters had been in use in the classroom for some time we set up the television set, computer and printer.

The only explanation I gave to the children was that a computer cannot tell the difference between 0 (alphabetical) and ∅ (numerical) so the latter had to be written as ∅ and is

Julie Morrell



This is a Valve This is a transistor This is a micro chip

here are the sizes of the Valve and the transistor and the micro chip.

called zero. We did a quick space rocket countdown – 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, zero – and the explanation was accepted.

The CAPS SHIFT key was pointed out and children took it in turns to write short sentences. They found this very exciting and rewarding as errors could be rectified with ease. Their efforts were recorded by the teacher on the printer. One copy for school, and another to take home to show parents.

```
Donna    I am 6
We did the computer. Mark
Lees can do it and Ian can.
I had a go today and it is
good. Melanie and Lorna did
not have a go of it they
looked at me.
```

As well as using the computer as an interesting typewriter, the children were shown how to 'write in' simple programs. (Although we started with a Sinclair ZX81, we progressed to a Spectrum shortly after it became available.)

Use of scrap books

We have two large scrap books. One is called 'Interesting

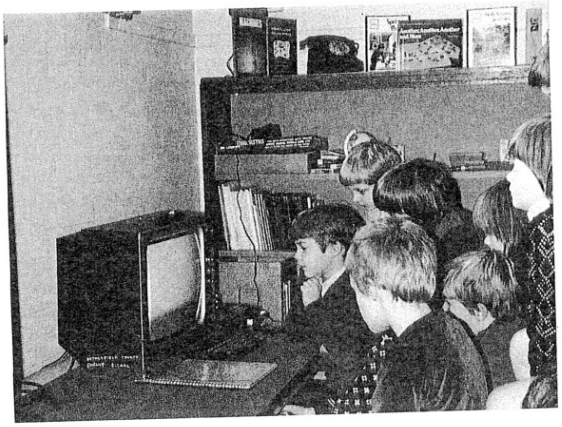
Things' and the other is labelled 'The Computer'. Any pictures or written work done by the children has been put into the books.

Alicia I am 7

We have a computer in our classroom and we do all sorts of things. One day we did Wall and we add numbers and we do Space Invaders. We write our names and some big boys called Trevor Martin and Patrick sent us a message that moved across the television it said Hello boys and girls of Netherfield School. We hope that you are all being good and working hard. love from Trevor Martin and Patrick.

Philip I am 7

This is for counting with and when people go by you press a buttern and it clicks and a number comes up so that you know how many it is and its called a HAND DESK tally counter. I have writtern 38 words



Work Cards

How can very young children faced with a keyboard which has 40 keys, and almost 200 letters, figures, words and symbols be expected to work (with the minimum of help from a busy teacher) when the first thing they want to do is gain 'hands on' experience?

A set of clear, simple to follow cards can be made, and these will help the children to 'type in' programs from the beginning.

As you'll see from the cards shown here, most of the instructions are straightforward, except that TO and = and the like are written in little boxes, with SYMBOL in the top half of the box, to remind the children to press and hold SYMBOL SHIFT as they press the TO or = or whatever.

Write your name A1

10	BORDER 4	ENTER
20	CAPS / SYMBOL, CAPS / PAPER	7 ENTER
30	PRINT SYMBOL / "	Write your name SYMBOL / " ENTER
40	PAUSE 80	ENTER
50	GOTO 30	ENTER
	RUN ENTER	

To clear the screen for the next person

CAPS / BREAK NEW ENTER

A2

Listen

10 FOR X [SYMBOL =] 0 [SYMBOL TO] 24 ENTER

20 [CAPS SYMBOL / BEEP SYMBOL] 1 [SYMBOL 9] X ENTER

30 NEXT X ENTER

RUN ENTER

Press RUN ENTER again.

How many sounds can you hear?

DELETE is worked in a similar way, but with the word CAPS above the word DELETE within a box. The more difficult BEEP and PAPER etc are written as a box divided into quarters, labelled CAPS SYMBOL BEEP SYMBOL to help the children to remember to press CAPS and SYMBOL together and then to hold SYMBOL before pressing BEEP.

Small fingers usually master this during the first two lessons. (David and Dale are 6 and wrote their names – and put them on the printer within a few minutes of seeing their first SPECTRUM.)

In the early stages, ENTER is written at the end of each line, but this can soon be omitted when the children have had more experience. After a few weeks the children can master some of the shorter programs from books and magazines.

A3

Look at the colours.

10 FOR X [SYMBOL =] 0 [SYMBOL TO] 7 ENTER

20 BORDER X ENTER

30 [CAPS SYMBOL / CAPS PAPER] 7 [SYMBOL —] X [SYMBOL :] [CAPS CLS] ENTER

40 PAUSE 150 ENTER

50 NEXT X ENTER

60 GOTO 10 ENTER

To look at the colours again RUN ENTER

```
10 FOR x=0 TO 7
20 BORDER x
30 PAPER 7-x: CLS
40 PAUSE 150
50 NEXT x
60 GO TO 10
```

As well as the 'work cards' it is useful to have a set of cards which give information to the children, such as those shown here.

At first they will have difficulty in finding some of the words and symbols, although they have no problem in finding figures and letters, especially if they have had some

Words and symbols

AND	Y	INPUT	I	"	P
AT	I	INVERSE	M	.	M
BEEP	Z	LET	L	,	N
BORDER	B	LIST	K	;	O
BRIGHT	B	LLIST	V	:	Z
CAPS LOCK	2	LOAD	J	+	K
CIRCLE	H	L PRINT	C	−	J
CLS	V	NEW	A	*	B
CONT	C	NEXT	N	/	V
COPY	Z	PAPER	C	=	L
DELETE	0	PAUSE	M	?	C
EDIT	1	PRINT	P	£	X
FLASH	V	RAND	T		
FOR	F	REM	E		
GOTO	G	RETURN	Y		
GRAPHICS	9	RND	T		
IF	U	RUN	R		
INK	X	STOP	A		
INKEY$	N	THEN	9		
		TO	F		

experience with a typewriter. The simplest card gives the most commonly-used words and symbols. Much later, the children should have access to cards giving the full range of words and symbols.

Children enjoy holding the cards and giving information to others when necessary.

Other cards which may be useful are:

- Instructions for loading a program from a tape recorder
- Deleting or changing lines in a program

Game wall

LOAD | SYMBOL " | wall | SYMBOL " | ENTER

Press PLAY on the tape recorder and watch the screen until you see STOP THE TAPE

Press STOP on the tape recorder.

To play the game press any key on the computer

Hold down CAPS and use
O for left ←
P for right →

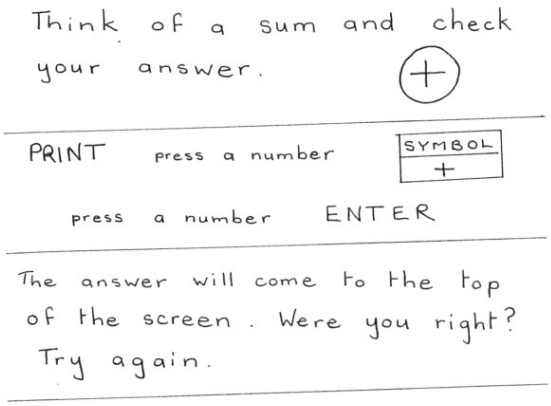
Write down your score.

Press Y to play the game again

- A cursor card
- A list of colours
- The priority of number functions, ( ) ^ * / + −, for older children

When the children play computer games such as 'Wall', it is useful to have a card for each game, to remind them which keys to press to work the game, such as 'O' to make the bat go to the left, and 'P' to get it to move to the right.

It helps if all tapes and work cards are marked with small



coloured stickers so that they can be put into groups or sets, such as:

1. Yellow – computer practice
2. Green – school work
3. Red – Computer games
4. Blue – Recording-keeping and teacher's aids

Group Two can be subdivided into school subjects, such as Mathematics, English, Pre-reading and so on.

Letters and numbers can be written on the stickers so that children can work at their own pace. For example, when a

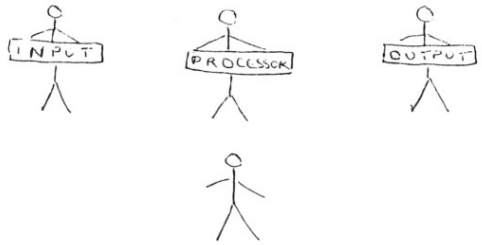
child has completed work cards A1, A2, A3 and the rest, he or she can move onto B1, B2, etc.

When I first acquired the Spectrum, I wrote out instructions for using the tape recorder – saving and loading – and had the full list of codewords and symbols near the computer. I felt much more confident when I was surrounded by parents and children that I would not plug in the incorrect leads, and would not be fumbling for keys. Later I found that a 'piano-type keyboard' set of instructions for DRAW and CIRCLE, and a squared, numbered 'screen' were very useful.

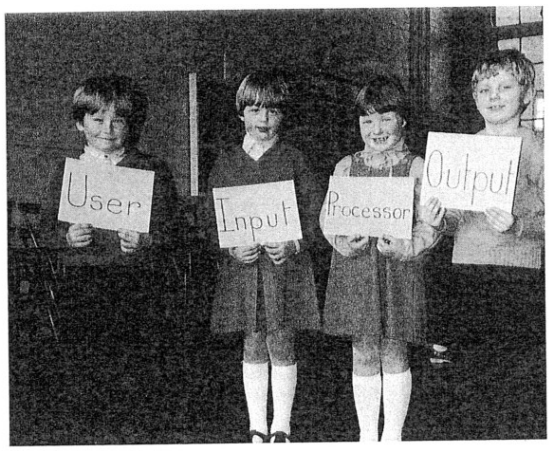
The Computer Game

This is a class activity which teaches young children about a computer in a very simple way.

Ask the children to stand in groups of four. One child is the 'User' and stands facing the other three, who each hold a sign:



'USER' can only speak to INPUT and can choose his or her own question. For example: "If I have four sweets and eat two, how many are left?" INPUT asks processor the same question. PROCESSOR must work out the answer while running up and down on the spot. The answer is passed to OUTPUT who in turn passes it back to user. The children enjoy this game, and can continue at their own speed, changing places occasionally. It is interesting to watch the PROCESSORs performing their little dances while working out their answers. The teacher walks from group to group checking that all is well.



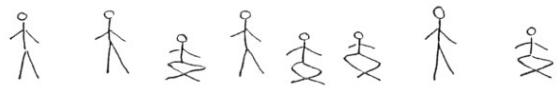
The Byte Game

Another game enjoyed by the children is one which will mean much more to them as they become older and more experienced.

Each child is called a 'bit' and moves around the hall to music. When the music stops, they have to try and form lines of eight. The children in the line can choose whether to stand up and be called "high powered" (or 1) or sit down and be called "low powered" (or Ø). Children in the first group to be ready receive a counter and the music begins again and the game continues.

At the end of the session the child with the most counters (the one who has been in the quickest 'byte' most often) is the outright winner.

To date, I have not given any more information about the binary system to the very young children, except to mention that a computer with a 1K memory has just over 1000 bytes and that our computer has a 16K memory.



Storing the equipment

Some computers have to be left in a permanent position in the school building, but a Spectrum can be carried around to be used with any television set (usually colour, but some programs work well on a black and white set if the colour set is already in use).

When people receive a computer for the first time, they nearly always put each item carefully into the original boxes for safe-keeping. This is the ideal method, but can be far too time-consuming in a busy school.

It is not always convenient for the equipment to be 'set up' for long periods of time, and it is sensible to put everything away each evening.

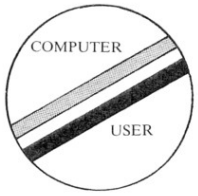
One easy way to store the computer, printer, tape recorder and a few cassettes, is to beg a strong, medium-sized, fairly shallow cardboard box from a supermarket. Experiment with different sizes of boxes until one is found which will hold everything in current use. Pad the base of the box with foam rubber of different thicknesses so that each item will fit into its own space. It will probably be possible for the printer and some of the leads to remain connected.

A large piece of hardboard or chipboard, covered with felt and with narrow strips of wood fixed around the edge, making a tray, can be used on the top of a table so that the items do not slip. Work cards can be kept in transparent pockets in a loose-leaf file.

Badges

All young children enjoy wearing badges. So why not reward them when they reach a certain standard?

A six- or seven-year-old child will be proud to wear a home-made badge when they have proved they can:



1. Copy a simple program from a card
2. Load a program from a tape recorder
3. Use the printer (The printer and cassette recorder must be connected to the computer at the beginning of each session)

If you have a school computer club, have a badge for each member. Note that small cardboard badges, complete with safety pins, suitable for this purpose, can be obtained from most large stationers.

A teacher may say that he or she cannot be with the computer group all the time. There is no real problem with this, as once the children have had experience in using the program cards, and can load a program from a tape recorder, they can be left in the corner of a classroom to follow instructions. Sometimes a program is left on the TV screen for a whole session, and can be used, as an example, to give instructions to the children as in this example:

```
Write about your visit to the
fire station.

     Words to help you

   fire engine       ladder

pole    hose    uniform   helmet

   boots      alarm      axe

water     quickly     dangerous

   burn     careful     safe
```

This helps with reading and free writing and the words on the screen can be easily changed.

CHAPTER FOURTEEN

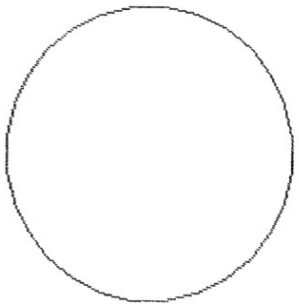
Specific Applications in Infant School

In this chapter, we outline a number of specific ideas for using the Spectrum more directly in education in infant school.

Handwriting

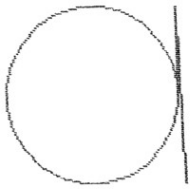
Some children have difficulty in forming letters correctly. If they are asked to enter into the computer the very simple 'MAGIC O' program, they can then follow the shape as it comes on the screen. When you run this, you'll see the 'O' is formed in a way which shows the direction of the line (anti-clockwise) very clearly. An older child or parent could write in the program for a younger child.

```
10 REM Magic O
20 CIRCLE 100,100,75
30 PAUSE 100
40 OVER 1
50 GO TO 20
```

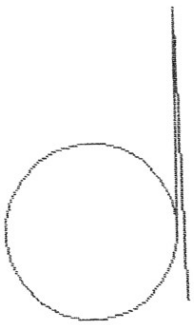


You can follow this with the 'a' and 'd' programs:

```
10 REM Letter a
20 CIRCLE 50,50,45
30 PAUSE 50
40 DRAW 0,50
50 PAUSE 80
60 DRAW 3,-90
70 PAUSE 200
80 CLS
90 GO TO 20
```



```
10 REM Letter d
20 CIRCLE 50,50,45
30 PAUSE 30
40 DRAW 0,120
50 PAUSE 80
60 DRAW 5,-150
70 PAUSE 200
80 CLS
90 GO TO 20
```



Here's the program card for the letter d program:

Make letter d

20 [CAPS][SYMBOL] [CAPS][CIRCLE] 50 [SYMBOL][7] 50 [SYMBOL][7] 45 ENTER

30 PAUSE 30 ENTER

40 DRAW 0 , 120 ENTER

50 PAUSE 80 ENTER

60 DRAW 5 [SYMBOL][7] [SYMBOL][—] 150 ENTER

70 PAUSE 200 ENTER

80 CLS ENTER

90 GOTO 20 ENTER

Follow the shape with your finger.

Remedial reading

Try using the TV screen as a reading aid for slower children, or for children who are lacking in confidence. Let a group of four or five children watch the teacher print a sentence or two using a few words from the more simple books from the school scheme, and include each child's name if possible. Leave the children to help each other and when they report that they can read the words on the screen, let each one press COPY and ENTER so that they have copies to stick

into their word books, or to take home.

As the children improve, longer 'pages' with more difficult words can be used. The teacher will probably find that the children are soon asking if they can write their own words and sentences.

```
David has a new football.
He would like to play for
Nottingham Forest
    Lindsay had a new doll
    for her birthday. She
    thinks it looks like a
    real baby.
Gareth likes to ride his
bicycle.
    David and Lindsay and
    Gareth like to play.
```


CHAPTER FIFTEEN

Maintaining Interest

There is little problem in keeping children interested in using the computer. However, it is just as well to use as many facilities of the Spectrum as possible, to ensure that the programs the children run are interesting.

Sound

An amusing piece of work was evolved around a few randomly chosen, repeating notes. The children said it sounded like someone running, so we coloured the BORDER and PAPER, left 'the patter of tiny feet' in the background, and wrote the following on the screen:

```
  Here come the men from one
of the planets. They run
very quickly and have very
tiny feet. They have long
arms and thin legs.

  I think that they are
running away from a big ugly
space monster.

  Can you draw a lot of
little men from space running
from a horrible monster?
```

This was a good reading exercise and the pictures were super. We taped the program so that we could use it again:

```
  10 REM Read 1
  20 PRINT "
```

```
 from one          Here come the men
ey run           of the planets. Th
ave very         very quickly and h
ve long          tiny feet. They ha
.                arms and thin legs

                   I think that they
 are             running away from a
 big ugly        space monster.
```

```
                    Can you draw a lo
t of                little men from spa
ce running          from a horrible mon
ster?"
  30 BORDER 5
  40 PAPER 6
  50 LET a=0
  60 PAUSE 10
  70 BEEP .01,a
  80 LET a=a+8
  90 IF a>50 THEN GO TO 50
 100 GO TO 60
```

It is possible to copy programs which help with the writing of simple tunes. Music familiar to the children may be used or they can experiment by writing tunes of their own. The group should work in a quiet corner, so as not to disturb the other children.

This next program plays the tune 'Twinkle, twinkle little star'. The star pattern changes at the end of each line. The colours and patterns of the stars can be changed very easily. We've left the program running in a 'Mums and toddlers' group so that the children could listen in, or have a quiet sing whenever they wished. The sound can be amplified by using a tape recorder. (I am sure that many other simple songs could be used in this way. Perhaps someone could make a graphic animal for each verse of 'Old MacDonald had a farm', so that everyone could join in for a sing song.)

```
  10 REM "Star"
  20 BORDER 3
  30 PAPER 1
  40 INK 6
  50 GO SUB 1000
  60 CLS
  70 GO SUB 2000
  80 BEEP 1,5: BEEP 1,5: BEEP 1,
12: BEEP 1,12: BEEP 1,14: BEEP 1
,14: BEEP 2,12.
  90 CLS
 100 GO SUB 1000
 110 BEEP 1,10: BEEP 1,10: BEEP
1,9: BEEP 1,9: BEEP 1,7: BEEP 1,
7: BEEP 2,5:
 120 CLS
 130 GO SUB 2000
 140 BEEP 1,12: BEEP 1,12: BEEP
1,10: BEEP 1,10: BEEP 1,9: BEEP
1,9: BEEP 2,7:
 150 CLS
 160 GO SUB 1000
 170 BEEP 1,12: BEEP 1,12: BEEP
1,10: BEEP 1,10: BEEP 1,9: BEEP
1,9: BEEP 2,7:
```

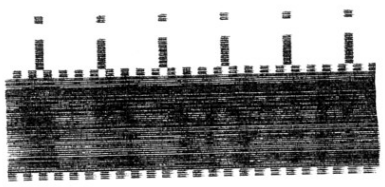


```
 180 CLS
 190 GO SUB 2000
 200 BEEP 1,5: BEEP 1,5: BEEP 1,
12: BEEP 1,12: BEEP 1,14: BEEP 1
,14: BEEP 2,12.
 210 CLS
 220 GO SUB 1000
 230 BEEP 1,10: BEEP 1,10: BEEP
1,9: BEEP 1,9: BEEP 1,7: BEEP 1,
7: BEEP 2,5:
 240 CLS
 250 GO SUB 2000
 260 CLS
 270 GO SUB 1000
 280 CLS
 290 GO TO 50
1000 PRINT "
                *
                       *
            *                    *
  *
 *
                     *           *
        *
           *                            *
   *                   *
             *
        *                            *
                      *  *
                                 *
      *"
1001 RETURN
2000 PRINT "               *         *
       *               *
   *            *                *
            *              *
             *     *
         *
 *       *         *                *
         *               *     *
           *                 *
    *               *
 *       *                             *
                *               *
         *             *
    *                             *
 *                                     *
               *                       *
    *                          *
             *            *
2001 RETURN   *            *"
```

Happy Birthday

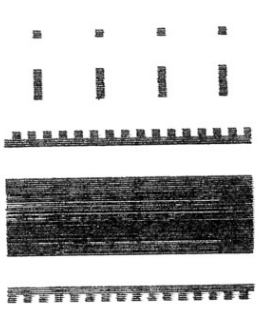
This is a program which can be used to delight a child on a special day. A birthday cake appears on the screen as follows:

```
Sarah is six today.
   Happy birthday.
```



The 'Happy Birthday' song is heard repeatedly. In this example, Sarah's cake has six candles, but the number of candles, the age and the name can easily be changed.

```
  10 BORDER 4
  20 PAPER 7
  30 PRINT "
                    Sarah is six t
oday."
  40 PRINT "            Happy birth
day."
                "
  50 INK 6
  60 PRINT "
    "
  70 PRINT "
   "
  80 INK 3
  90 PRINT "
"
 100 INK 5
 110 PRINT "
"
 120 INK 3
 130 PRINT "
"
```

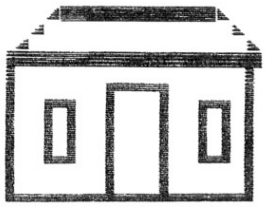


If you put the program on tape beforehand, using a number of different ages with the appropriate number of candles, and save them on tape as 'BIRTHDAY 4' and 'BIRTHDAY 5' and so on, it will be a simple matter to load the correct program. Line 3Ø can then be quickly edited to include the correct name.

Using the Spectrum's colours

The children can 'build' a house by following simple instructions and by using the colour keys, to answer questions such as "What colour would you like the roof?". After they have pressed the colour of their choice, the next question appears.

```
Thank you for colouring the
house.        Good-bye!
```



When the house is completed, the children are asked to press 'y' or 'n' (for 'yes' or 'no') to get another house to build.

```
  10 REM "House"
  20 FOR n=0 TO 7
  30 READ row: POKE USR "A"+n,ro
w
  40 NEXT n
  50 FOR p=0 TO 7
  60 READ row: POKE USR "B"+p,ro
w
  70 NEXT p
  80 CLS : BORDER 7
  90 PRINT AT 0,0;"What colour w
ould you like the  roof?"
 100 INPUT a: INK a
 110 LET z=5: LET y=7
 120 PRINT AT z,y;"◢"
 130 IF z=3 THEN GO TO 160
```

```
 140 LET z=z-1: LET y=y+1
 150 GO TO 120
 160 LET y=y+1
 170 PRINT AT z,y;"▓"
 180 IF y=19 THEN GO TO 210
 190 LET y=y+1
 200 GO TO 170
 210 LET y=y+1
 220 PRINT AT z,y;"▙"
 230 IF z=5 THEN GO TO 260
 240 LET z=z+1: LET y=y+1
 250 GO TO 220
 260 INK 0
 270 PRINT AT 0,0;"What colour w
ould you like the  walls?"
 280 INPUT b: INK b
 290 LET z=z+1
 300 PRINT AT z,y;"▐"
 310 IF z=14 THEN GO TO 340
 320 LET z=z+1
 330 GO TO 300
 340 LET z=z+1
 350 PRINT AT z,y;"▀"
 360 LET y=y-1
 370 PRINT AT z,y;"▀▀"
 380 IF y=7 THEN GO TO 410
 390 LET y=y-1
 400 GO TO 370
 410 LET y=y-1
 420 PRINT AT z,y;"▄▄"
 430 LET z=z-1
 440 PRINT AT z,y;"▐"
 450 IF z=6 THEN GO TO 480
 460 LET z=z-1
 470 GO TO 440
 480 LET y=y+1
 490 PRINT AT z,y;"▄"
 500 IF y=22 THEN GO TO 530
 510 LET y=y+1
 520 GO TO 490
 530 INK 0
 540 PRINT AT 0,0;"What colour w
ould you like the  windows?"
 550 INPUT c: INK c
 560 LET z=9: LET y=9
 570 PRINT AT z,y;"▛"
 580 LET y=y+1
 590 PRINT AT z,y;"▜"
 600 LET z=z+1
 610 PRINT AT z,y;"▐"
 620 IF z=11 THEN GO TO 650
 630 LET z=z+1
 640 GO TO 610
 650 LET z=z+1
 660 PRINT AT z,y;"▄"
 670 LET y=y-1
 680 PRINT AT z,y;"▙"
 690 LET z=z-1
 700 PRINT AT z,y;"▐"
 710 IF z=10 THEN GO TO 740
 720 LET z=z-1
 730 GO TO 700
```

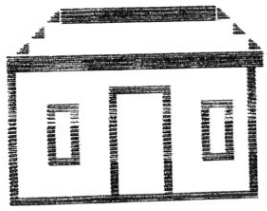


```
 740 LET z=8: LET y=19
 750 PRINT AT z,y;"▛"
 760 LET y=y+1
 770 PRINT AT z,y;"▜"
 780 LET z=z+1
 790 PRINT AT z,y;"▐"
 800 IF z=11 THEN GO TO 830
 810 LET z=z+1
 820 GO TO 790
 830 LET z=z+1
 840 PRINT AT z,y;"▄"
 850 LET y=y-1
 860 PRINT AT z,y;"▙"
 870 LET z=z-1
 880 PRINT AT z,y;"▐"
 890 IF z=10 THEN GO TO 920
 900 LET z=z-1
 910 GO TO 880
 920 INK 0
 930 PRINT AT 0,0;"What colour w
ould you like the  door?"
 940 INPUT d: INK d
 950 LET z=14: LET y=13
 960 PRINT AT z,y;"▟"
 970 IF z=9 THEN GO TO 1000
 980 LET z=z-1
 990 GO TO 960
1000 LET z=z-1
1010 PRINT AT z,y;"▛"
1020 LET y=y+1
1030 PRINT AT z,y;"▀▀"
1040 IF y=15 THEN GO TO 1070
1050 LET y=y+1
1060 GO TO 1030
1070 LET y=y+1
1080 PRINT AT z,y;"▜"
1090 LET z=z+1
1100 PRINT AT z,y;"▐"
1110 IF z=14 THEN GO TO 1140
1120 LET z=z+1
1130 GO TO 1100
1140 INK 0
1150 PRINT AT 0,0;"What colour w
ould you like the  border?"
1160 INPUT e: BORDER e
1170 PRINT AT 0,0;"The house is
finished now. Would you like to
make another one?   Please answe
r y or n."
1180 INPUT f$
1190 IF f$="y" THEN GO TO 80
1200 IF f$="n" THEN PRINT AT 0,0
;"Thank you for colouring the
  house.          Good-bye!

1210 GO TO 1240
1220 PRINT "Press y or n again p
lease."
1230 GO TO 1180
1240 STOP
1250 DATA BIN 00000000,BIN 00000
010,BIN 00000110,BIN 00001110,BI
```

```
N 00011110,BIN 00111110,BIN 0111
1110,BIN 00000000
1250 DATA BIN 00000000,BIN 01000
000,BIN 01100000,BIN 01110000,BI
N 01111000,BIN 01111100,BIN 0111
1110,BIN 00000000

Thank you for colouring the
house.          Good-bye!
```



House

Read the question.

Choose the colour and press :-

1 for BLUE
2 for RED
3 for MAGENTA
4 for GREEN
5 for CYAN
6 for YELLOW
0 for BLACK

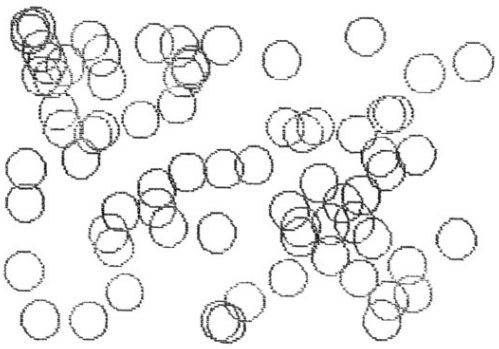
To make another house press y for yes.

To stop the program press n for no.

Bubbles

The next program (adapted from one in *The ZX Spectrum Explored*, Sinclair Browne, 1982) is a great favourite with the children. We have called it 'Bubbles' and have slowed it down, by adding a PAUSE line:

```
10 REM Bubbles
20 PAPER 0
30 LET x=(234*RND)+10
40 LET y=(154*RND)+10
50 LET z=(5*RND)+1
60 CIRCLE INK z;x,y,10
70 PAUSE 30
80 GO TO 30
```



Parents

Parents are welcome in schools, as they can help in many ways. They are curious, as well, especially about computers. They did not have the opportunity to operate computers when they were at school, although many of them use them now. Often parents and children will learn together.

After the children have shown that they can write their names on the computer, and can follow instructions from a

card, the parents usually want to show their keyboard skills. The Spectrum demonstration tape – 'Keyboard Trainer, Lesson One' – can be loaded and parents and children left to help each other. After using the simple instruction cards, parents are soon able to enter programs from books and magazines. One father spent the whole afternoon at school using the spectrum (for the first time) and wrote the following program:

```
 10 REM "Add"
 20 LET f=INT (RND*10)+1
 30 LET s=INT (RND*10)+1
 40 LET c=f+s
 50 LET t=0
 60 FOR t=1 TO 3
 70 PRINT f;s
 80 PRINT "What is the first nu
mber added to the second number?
"
 90 INPUT x
100 IF x<>c THEN PRINT "Sorry,
you are wrong."
110 PAUSE 100
120 IF x=c THEN GO TO 170
130 CLS
140 IF t<>3 THEN PRINT "The cor
rect answer is....";c
150 NEXT t
160 PAUSE 100
170 PRINT "You are right, well
done."
180 PAUSE 100
```

Parents and children enjoy writing messages to each other, leaving them on the screen, or using the printer.

I recently watched a man buying a Spectrum for his three children. The shop was busy and the assistant was looking anxiously at the queue. "If I plug it in," said the bewildered father, "does it play games right away? How does it know what to do? I shan't be able to show them how to use it when I give it to them." I reassured him that I felt just the same 18 months ago. Then I asked him the age of the children, and he told me that the oldest was 14 and had used a Spectrum at school. Everyone in the queue breathed sighs of relief, and told him that there would be no problems, and he would be lucky if they allowed him near the computer for several days. Someone asked him where he lived, and offered to go to the house to help. The man asked about tape recorders

and was told they saved a lot of time. Books were produced and more advice given. In fact, a computer club was almost formed on the spot.

I have found that most adults fall into one of two categories: those who are great experts, and those who say they want to run a mile at the mention of the word 'computer'. Children are all of one voice. They are keen, confident, and are anxious to get 'hands on' experience. The 'shy' parents are soon converted after they have watched their very young children using the computer without difficulty.

History

Children enjoy hearing about the first computer. I teach them a series of lessons based on the following material. The depth to which the subject is treated is dependent upon the age of the children.

Babbage's machines (1830s)

Charles Babbage was extremely annoyed at the inaccuracies in certain astronomical tables that had been calculated at the beginning of the nineteenth century. He realised that there was a need for a sophisticated device for calculations of greater accuracy.

He decided to build a mechanical computing device that could handle numbers with as many as 20 digits. One device, his 'Difference Engine' was partially completed, and he made plans for another, called the 'Analytical Engine'.

The engineering techniques of his time could not cope with his elaborate ideas, and because of this and the amount of money required, his projects failed.

Hollerith's Punched Cards (1890)

Herman Hollerith was employed by the American Census Bureau.

The 1880 census took seven years to complete, and Hollerith realised that the 1890 census would take a much longer time because of the very fast growth of the American population.

Unless mechanical devices were used, it was possible that

the results would not be completed before the 1900 census was due. Hollerith used a punched card code for the information, along with electro-mechanical devices which sorted the cards and then printed the data and totals. His important contribution meant that the 1890 census was completed in three years.

Aiken's development of Babbage's machine (1940)
The 'Automatic Sequence Controlled Calculator' was developed by Howard Aiken at Harvard University and was completed in 1944. It was electro-mechanical, consisting bascially of a large number of calculators, controlled by instructions stored on paper tape. The machine was used by the US Navy for 15 years.

```
    Matthew  I am 7

A long time ago there was a
man called Mr. Babbage. He
tried very hard to make a
machine that was very good at
doing sums very quickly.When
Mr.Babbage made his counting
machine it was so big that it
filled a whole room. He ran
out of money so he could not
finish making his machine.
```

Computers 1946–1959
The computers were built to use valves, and were relatively slow in operation. They consisted of a central processing unit and input and output devices. They used primitive but ingenious types of storage units which were later replaced by the use of ferrite cores.

Developments 1960–1964
The invention of the transistor meant that much smaller computers could be built. These had circuit boards with transistors, diodes, capacitors and resistors.

Transistors, like valves, are electronic switches through which current can be passed under certain circumstances. Because of their 'switching speed', transistors enabled the computer to work at a much faster speed than was possible with valves.

By this stage, magnetic tapes and discs were beginning to be used with computers.

Computers from 1964
Computer manufacturers began to spend as much money on the development of software as they did on the hardware.

The computer became even smaller with the introduction of integrated circuits in which transistors, diodes, capacitors and resistors were, in effect, fused together into a chip.

CHAPTER SIXTEEN

Brain Games

As you'll read in my 'final thoughts', some schools have found that the use their computers get above and beyond planned use within classes, depends to some extent upon the availability of software which is not locked to specific classes.

It would not be appropriate for me to provide you with listings of programs like 'Galactic Intruders'. Rather, here are a few games which have sufficient points in their favour – above and beyond the very important one that they are fun to operate – to merit inclusion in this book.

You'll find that these programs are also useful as 'icebreakers' to help students whose first contact with a computer is the one in the classroom.

Mind reader

This game asks the student to enter his or her name, and then follow a number of instructions, all of which demand a certain facility with mathematics. Having followed the instructions, and pressed ENTER after carrying out each step as instructed, the computer tells the student their age, and how much change he or she is carrying. The program does not take long to run (although, of course, the less competent the mathematician running it, the longer it will take) so several students can have a run through with it after it has been loaded.

This is the listing for MIND READER:

```
  10 REM MIND READER
  20 INPUT "What is your name? "
;A$
  30 PRINT '"Hi there, ";A$
  40 PRINT '"It's time to test
your skill","with numbers, and m
y skill as","a mind reader!"
  50 GO SUB 500
```



```
  60 PRINT "OK, ";A$;", start by
 multiplying","your age (in year
s) by two"
  70 GO SUB 500
  80 PRINT "Now add five to that
"
  90 GO SUB 500
 100 PRINT "And now multiply tha
t by 50"
 110 GO SUB 500
 120 PRINT "Now subtract 365 fro
m that"
 130 GO SUB 500
 140 PRINT "Now add the amount o
f loose","change in your pocket"
 150 GO SUB 500
 160 PRINT "Now give me the numb
er you've","ended up with"
 170 INPUT A
 180 GO SUB 500
 190 LET A=A+115
 200 LET B=INT (A/100)
 220 LET A=A-100*B
 230 PRINT "You have ";A;" chang
e"
 240 GO SUB 500
 250 PRINT '"And are ";B;" year
s old"
 260 STOP
 500 FOR G=1 TO 300: NEXT G
 510 PRINT '"Press "; INVERSE 1
;"ENTER"; INVERSE 0;" when you'r
e","ready to continue"
 520 INPUT Z$
 530 CLS
 540 RETURN
```

Here's what it looks like in action:

```
Hi there, Tim

It's time to test your skill
with numbers, and my skill as
a mind reader!

OK, Tim, start by multiplying
your age (in years) by two

Press ENTER when you're
ready to continue

You have 38 change


And are 17 years old
```

Code-breaker

The next program is a simplified form of the game Mastermind (the game and name are copyright Invicta Plastics). Invicta bought the rights to the game from an amateur mathematician – Mordechai Meirovich – in 1971. The game has been popular here in the UK for centuries under the name of 'Bulls and Cows'.

Whereas the commercial toy generally uses coloured plastic pegs, and has one player choosing a four-colour 'code' which the other player tries to break, this version uses digits instead of colours. The code is three-digits long, and – to make it easier to solve – the same digit is not repeated within a code. This means, for example, that 123 and 973 are valid codes, but 993 and 272 are not. Zero is not used.

Here's a sample run of 'Code-breaker':

```
This is guess number 1
1 black 0 whites
0 blacks 1 white
1 black 1 white
1 black 2 whites
0 blacks 3 whites
1 black 2 whites
0 blacks 3 whites
0 blacks 3 whites

You got it in 9 guesses

My code was 726
```

As you can see from this sample run, the computer awards a 'black' for any digit in the right position within the code, and a 'white' for any digit which appears in both the code and the student's guess, but not in the correct position. The player attempts to work out the code in less than 10 guesses, using the feedback the computer gives after each guess is entered.

This is the listing for 'Code-breaker':

```
  10 REM Code-breaker
  20 BORDER 2: PAPER 2: INK 6: C
LS
  30 DIM A(3): DIM B(3)
  90 FOR Z=1 TO 3
 100 LET A(Z)=INT (RND*9)+1
 110 NEXT Z
```



```
 120 IF A(1)=A(2) OR A(1)=A(3) O
R A(2)=A(3) THEN GO TO 90
 130 LET A=100*A(1)+10*A(2)+A(3)
 140 FOR C=1 TO 10
 150 PRINT AT 0,0;"This is guess
 number ";  INVERSE 1;C
 152 INPUT INVERSE 1;"Please ent
er your guess ";B
 155 IF B<100 OR B>999 THEN GO T
O 150
 157 PRINT AT C+1,1;  INVERSE 1;B
;  INVERSE 0;" ";
 160 LET B(1)=INT (B/100)
 170 LET B(2)=INT ((B-100*B(1))/
10)
 180 LET B(3)=INT (B-100*B(1)-10
*B(2))
 190 IF A=B THEN GO TO 430
 200 LET D=A
 210 LET N=0
 220 LET W=0
 230 FOR E=1 TO 3
 240 IF A(E)<>B(E) THEN GO TO 27
0
 250 LET N=N+1
 260 LET A(E)=0
 270 NEXT E
 280 FOR F=1 TO 3
 290 IF A(F)=0 THEN GO TO 340
 300 FOR E=1 TO 3
 310 IF B(F)<>A(E) THEN GO TO 33
0
 320 LET W=W+1
 330 NEXT E
 340 NEXT F
 350 LET A(1)=INT (D/100)
 360 LET A(2)=INT ((D-100*A(1))/
10)
 370 LET A(3)=INT (D-100*A(1)-10
*A(2))
 380 PRINT N;" black";
 385 IF N<>1 THEN PRINT "s ";
 386 IF N=1 THEN PRINT " ";
 390 PRINT W;" white";
 395 IF W<>1 THEN PRINT "s"
 400 NEXT C
 410 PRINT AT 12,0;"'"You didn't
get it"
 420 GO TO 440
 430 PRINT AT 12,0;"You got it i
n ";C;" guesses"
 440 PRINT "'"My code was ";A
```

Magic Star Sums

This program generates a five-pointed star, with a number on each of the intersections of the lines which make it up. The numbers in any particular line always sum to the same

total, much the same as, in a magic square, numbers in a straight line in any direction sum to the same total.

When you run the program, you'll see that two or three of the numbers on the star have been replaced with zeroes. It is up to you to work out what the needed numbers are in as few moves as possible.

Here are some 'snapshots' from one run:

```
Go number 1

        3
   14   13   15   0
     9      17
          0
       0    3
```

```
Go number 6

        3
   14   13   15   0
     9      17
          0
       0    3

You have 7 right so far
```

```
Go number 7

        3
   14   13   15   0
     9      17
          12
       0    3

You have 8 right so far
```



```
Go number 8

        3
   14   13   15   -4
     9      17
          12
       0    3

You have 9 right so far

You solved it in just 10 goes

        3
   14   13   15   -4
     9      17
          12
       13   3


OK, thanks for playing
```

And this is the listing so you can create your own magic stars:

```
  10 REM MAGIC STARS
  20 GO SUB 480: REM SET UP STAR
  30 GO SUB 340: REM PRINT STAR
  40 GO SUB 70: REM ASK FOR GUES
S
  50 GO TO 30
  60 REM ****************
  70 REM ASK FOR GUESS
  80 IF SCORE>0 AND SCORE<10 THE
N PRINT AT 18,0;"You have ";SCOR
E;" right so far  "
  90 LET GO=GO+1
 100 PRINT AT 0,0; INVERSE 1;"Go
number "; FLASH 1; INK 2; PAPER
6;GO
 110 INPUT FLASH 1; INK 2; PAPER
```

```
 6;"This is go number ";(go),"En
ter any number which you","think
 is part of the star";G
 130 LET SCORE=0
 140 FOR J=1 TO 10
 150 IF G=A(J) THEN LET B(J)=A(J
)
 160 IF B(J)<>0 THEN LET SCORE=S
CORE+1
 170 NEXT J
 180 FOR P=1 TO 100: NEXT P
 190 IF SCORE<10 THEN RETURN
 200 REM ***************
 210 GO SUB 340
 220 PRINT AT 18,0;"
                   ";AT 0,0;"You
solved it in just ";GO;" goes"
 240 FOR P=1 TO 100: NEXT P
 250 PRINT : PRINT FLASH 1;"Pres
s 'Y' if you'd like another","go
, or 'N' to stop..."
 260 LET A$=INKEY$
 270 IF A$<>"Y" AND A$<>"y" AND
A$<>"N" AND A$<>"n" THEN GO TO 2
60
 300 IF A$="Y" OR A$="y" THEN RU
N
 310 PRINT AT 18,0;"OK, thanks f
or playing": STOP
 330 REM ****************
 340 REM PRINT STAR
 350 PRINT AT 4,7;B(1)
 380 PRINT AT 6,2;B(2);"   ";B(3)
;"   ";B(4);"   ";B(5)
 400 PRINT AT 8,4;B(6);"     ";B(
7)
 420 PRINT AT 10,7;B(8)
 440 PRINT AT 12,5;B(9);"    ";B(
10)
 460 RETURN
 470 REM ****************
 480 REM SET UP STAR
 490 CLS
 500 RANDOMIZE
 510 DIM A(10): DIM B(10)
 520 LET GO=0
 530 LET SCORE=0
 540 LET A=INT (RND*9)+1
 550 LET B=INT (RND*9)+1
 560 LET C=INT (RND*9)+1
 570 LET D=INT (RND*9)+1
 580 LET E=INT (RND*9)+1
 590 IF A=B OR A=C OR A=D OR A=E
 THEN GO TO 550
 600 IF B=C OR B=D OR B=E THEN G
O TO 550
 610 IF C=D OR C=E THEN GO TO 55
0
 620 IF D=E THEN GO TO 550
 630 LET X=INT (RND*3)+1
 640 LET A(1)=X
```



```
 650 LET A(2)=X-B+C+D
 660 LET A(3)=A+E
 670 LET A(4)=A+D
 680 LET A(5)=X-B-C+E
 690 LET A(6)=A
 700 LET A(7)=A+C
 710 LET A(8)=A+B
 720 LET A(9)=X-2*B+2*D+E
 730 LET A(10)=X-2*B-C+D+2*E
 740 FOR J=1 TO 10
 750 LET B(J)=A(J)
 760 IF A(J)=0 THEN RUN
 770 NEXT J
 780 FOR J=1 TO 3
 790 LET B(INT (RND*10)+1)=0
 800 NEXT J
 810 RETURN
```

Hangman

Now we have another old computer favourite, 'Hangman', in which the Spectrum thinks of a word, and the student has to guess it before being hanged. The number of guesses is related to the length of the word.

As you can see from lines 5ØØ to 54Ø, the program comes complete with a starting vocabulary. However, you can make the game much more useful by replacing this vocabulary with a series of words which either relate specifically to your school, or to a particular subject (such as the names of minerals being studied).

The game is simple to play, and the *modus operandi* will be made clear if you study this sample run:

```
You have to guess my word in
just 11 guesses


-------


You have 0 correct letters



You have 11 chances left


You have to guess my word in
just 11 guesses
 E A M U O I
```

```
ME-MAI-


You have 5 correct letters


You have 10 chances left


You have to guess my word in
just 11 guesses
 E A M W O I D R

MERMAID


You got it in 8 guesses

My word was MERMAID


Whew. You've staved off
execution for another day.
```

Here's the listing of 'Hangman':

```
  10 REM HANGMAN
  20 PAPER 1: BORDER 1: INK 7: C
LS
  40 LET Y=0
  50 RANDOMIZE
  60 FOR G=1 TO RND*22+1
  90 READ A$
 100 NEXT G
 110 GO SUB 480
 120 LET N=LEN A$
 125 DIM B(N): DIM D(N)
 130 FOR G=1 TO N
 140 LET B(G)=CODE A$(G)
 150 LET D(G)=B(G)
 160 NEXT G
 170 LET Q=INT (N+N/2+.5)
 180 PRINT AT 0,0;"You have to g
uess my word in","just ";Q;" gue
sses"
```



```
 200 GO SUB 480
 210 FOR J=1 TO Q: LET Y=Y+1
 220 GO SUB 400
 230 IF H=N THEN GO TO 340
 240 PRINT AT 18,0;"You have ";Q
+1-J;" chances left  "
 250 INPUT "Enter your guess ";C
$
 255 PRINT AT 2,2*Y-1;C$
 260 LET F=CODE C$
 270 FOR G=1 TO N
 280 IF D(G)=F THEN LET D(G)=0:
LET J=J-1
 290 NEXT G: NEXT J
 300 GO SUB 480
 310 GO SUB 400
 320 PRINT AT 12,0;"
                   ";AT 18,0;"
                             ";AT
 18,0;"I'm sorry but you didn't
get it"
 330 GO TO 370
 340 PRINT AT 12,0;"
                   ";AT 18,0;"
                             ";AT 1
8,0;"Whew. You've staved off","e
xecution for another day."
 360 PRINT AT 10,0;"You got it i
n ";Y-1;" guesses"
 370 PRINT AT 12,0;"My word was
";A$
 390 STOP
 400 LET H=0: PRINT AT 4,0;
 410 FOR E=1 TO N
 420 IF B(E)=D(E) THEN PRINT "-"
;
 430 IF B(E)<>D(E) THEN PRINT FL
ASH 1; INK 2;CHR$ B(E); FLASH 0;
 INK 7;: LET H=H+1
 440 NEXT E
 450 PRINT AT 12,0;"You have ";H
;" correct letters"
 470 RETURN
 480 FOR P=1 TO 30: BEEP .008,RN
D*40: NEXT P
 490 RETURN
 500 DATA "MERIDIAN","MERIT","ME
RMAID","MERRIMENT"
 510 DATA "OVERSEER","OXIDANT","
OXYGEN","PALPABLE","UNORTHODOX"
 520 DATA "PANDEMONIUM","PANEGYR
IC","PARADOXICAL","PHEASANT"
 530 DATA "RUMPUS","RUMMAGE","SA
CRAMENT","SABRE","SCHEMATIC"
 540 DATA "SEDIMENT","SEXAGENARI
AN","TEMPERATE","TELESCOPE"
```

Kimspot

Finally in this chapter, we have 'The Kimspot Game', written by Derek Cook, who says it has given him and his family (aged 13 and 11) a great deal of mental stimulation.

The program is a combination of the old Kim's Game and the new 'Spot the Ball'. It can help teach children the rudiments of coordinate geometry, as well as the ability to recognise and retain shapes. As you'll see when you run it, the program makes very effective use of such Spectrum facilities as FLASH, BEEP and colour.

This is the listing for 'The Kimspot Game':

```
  10 PRINT "© D.L.Cook,1982"
  20 BORDER 2: INK 1: FLASH 1: P
RINT AT 11,8;"THE KIMSPOT GAME"
  30 PAUSE 100: BORDER 1: INK 2:
 PRINT AT 11,2;"Type ""ENTER"" f
or instructions"
  40 INPUT q$: FLASH 0
  80 BORDER 1
  90 RANDOMIZE
  95 CLS
 100 DIM d$(3,3)
 110 FOR a=1 TO 5
 120 LET b=INT (RND*3+1)
 130 LET c=INT (RND*3+1)
 135 IF d$(b,c)="*" THEN GO TO 1
20
 140 LET d$(b,c)="*"
 150 NEXT a
 155 PRINT "You are looking for:
-"
 160 FOR e=1 TO 3
 170 FOR f=1 TO 3
 180 IF d$(e,f)="*" THEN INK 2:
IF d$(e,f)="*" THEN FLASH 1: PRI
NT AT e,f;d$(e,f)
 190 NEXT f
 200 NEXT e
 210 FLASH 0
 230 PRINT "Remember the shape c
arefully!"
 250 PRINT "The shape is hidden
in a square:"
 260 INK 1: FOR a=1 TO 10
 270 PRINT "          "
 280 NEXT a
 300 PRINT "of 100 doors. To ope
n a door,     type its co-ordinat
es as x     <ENTER>,y <ENTER>. I
f you are    sure you remember th
e shape you are looking for,type
 ENTER    to play"
 310 INPUT c$
 320 CLS : LET v=0
 330 LET p=INT (RND*8+1)
```



```
 340 LET q=INT (RND*8+1)
 350 FOR a=0 TO 9
 360 PRINT AT 6+a,10;9-a
 370 FOR b=0 TO 9
 380 PRINT AT 6+b,11+a;"■"
 390 NEXT b
 400 PRINT AT 16,11+a;a
 410 NEXT a
 415 FOR n=1 TO 100
 420 INPUT "Type co-ordinates";x
 430 INPUT y
 432 PRINT AT 20,1;n;" goes"
 435 IF NOT ((x=p OR x=p+1 OR x=
p-1) AND (y=q OR y=q+1 OR y=q-1)
) THEN GO SUB 800
 440 IF d$(2+q-y,2-p+x)="*" THEN
 GO SUB 600
 450 IF d$(2+q-y,2-p+x)<>"*" THE
N GO SUB 700
 455 IF v=5 THEN GO TO 900
 460 NEXT n
 590 STOP
 600 FLASH 1: INK 2: PRINT AT 15
-y,11+x;"*"
 605 LET v=v+1
 607 BEEP .5,v
 610 RETURN
 690 STOP
 700 FLASH 0: INK 2: PRINT AT 15
-y,11+x;"■"
 710 RETURN
 800 FLASH 0: INK 2: PRINT AT 15
-y,11+x;"■"
 810 GO TO 460
 900 FLASH 1: PRINT AT 2,0;"You
found it in ";n;" goes"
 910 BEEP .5,20: BEEP .5,25: BEE
P .5,30: BEEP .5,35
 920 FLASH 0: PRINT AT 4,0;"Anot
her game? y/n"
 930 INPUT m$
 940 IF m$="y" THEN GO TO 90
 950 IF m$="n" THEN STOP
```

CHAPTER SEVENTEEN

Some Final Thoughts

Computers are particularly useful when viewed as aids to, rather than replacements for, teaching by teachers. In one area, that of individual coaching and testing, a machine can provide assistance of a type which would be impossible (because of the time and attention required) for a teacher with an entire class to look after.

One teacher who is enthusiastic about the use of microcomputers for work of this type says he uses his school's computers for review work for fourth and fifth year French students, and for remedial work with first year students. Any beginning student whose work is not satisfactory is required to go through 'drill' with the computer after school. The teacher reports that the number of students who failed the course dropped significantly.*

Another school entered the computer age with a cast-off machine from a local company. They found that student interest in using the machine grew as the number of available programs grew. Early programs were not particularly sophisticated, and included such things as finding objects on 10 by 10 grids, and working out biorhythms; yet the school found that student interest was very high long before the computer was integrated formally into the curriculum (see '*The Micro in a Small School*', Interface Age, October 1979, p. 64).

Although this was before microcomputers were widespread, and therefore computers had a somewhat higher novelty value than they have now, the lesson of interest being proportional to available programs probably holds true for the Spectrum.

**Computer Assisted Instruction – Worth The Effort?*, F Keplinger, Compute! magazine, August 1981, p. 40.

When writing programs for school use, keep in mind the fairly obvious fact that a student learns best (or at least is more likely to pay attention) when he or she is interested and involved. Do all you can to ensure your programs address the twin needs of interest and involvement.

A need to learn something (for example, in order to pass a test) can, however, be so pressing that a student will learn, at least to some extent, material whose presentation is entirely lacking in interest or any real involvement.

One of the great advantages of the computer-as-teacher over the human-as-teacher is the infinite patience of the machine. The Spectrum can be programmed to react to the learning speed of the student, ensuring that a slow student is not so frustrated that he or she is unable to sustain interest in the material.

You may find resistance within your school to the use of the Spectrum. Many things have been hailed as great aids to teachers in the past, only to be found wanting. An experience of this type, and a suspicion that students playng with computers are not doing anything which should be within the school curriculum can create attitudes which hinder a wider use of Spectrums within your school.

The school Spectrums must not be treated like immensely valuable objects, to be accessed only in formal class situations. Among other things, students are learning familiarity with computers. This is probably equally as important as any formal studies done with the machine. Computers will occupy a very large part of most of your students' futures, and anything which increases their ease in working with them seems to me to be worthwhile.

Therefore, check yourself if you are on the verge of stopping somebody using the Spectrum because they are 'just playing'. Feeling good about the machine will make it simpler to engage the student in other forms of computer use, and will therefore ultimately aid you in using the computer more directly for school work.

Of course a degree of supervision will probably be advisable (even if only to make sure the computers don't leave the room in school bags). However, if the Spectrum or Spectrums can be made available at lunchtimes, and after school, this is all to the good. Let your students make full use of the machines you've bought. Encourage them to write their own programs, and to bring original programs from

home to show to other students. Do anything to foster a situation in which access to the machines is easy, and the formalities of using them is kept at a minimum.

APPENDIX A

A suggested Test Paper on Spectrum BASIC

1. What do you find at the bottom left hand corner of the screen when you touch any key after first turning the computer on?
2. What does it signify?
3. What would you type into the computer to:
   (a) add 3 and 4 together
   (b) divide 8 by 4
   (c) multiply 7 by 8
4. What does mean?
5. What is the difference between the zero and the letter 'O' when printed by the computer?
6. (a) What does the CAPS SHIFT key do? Give an example?
   (b) What does the SYMBOL SHIFT key do? Give an example.
7. What do you see when you PRINT the following (after pressing the ENTER key?
   (a) 1 2 3 4
   (b) 1;2;3;4
   (c) 1,2,3,4
   (d) 1.2
   (e) 1,,2
8. What do you call a row of letters or numbers enclosed in " "?
9. If you wanted to PRINT the word SPECTRUM on the screen, how would you type it?
10. Re-write the following correctly, so they will work on the Spectrum.

(a) PRINT SWIMMING IS FUN
(b) PRINT "THE GIRL IS"17 YEARS OLD"

11. Name three of the *functions* available on the Spectrum keyboard.

12. (a) How do you obtain a function?
(b) Give one example.

13. What does PRINT AT 11, 16 mean?

14. Correct this line (so it will work on the Spectrum):
PRINT AT 11,16 "THE BUS IS RED

15. Imagine your Spectrum is turned on in front of you. You type in the following:
LET A = 1 (then press ENTER)
LET B = 2 (then press ENTER)
LET C = 3 (then press ENTER)
What would you see if you did the following:
(a) PRINT A;B;C
(b) PRINT A + B,C
(c) PRINT A + C − B

16. How do you get PI on the Spectrum?

17. Complete this program:
1Ø INPUT "radius ";r
2Ø PRINT "the area of a circle is ";

18. After line 1Ø below what should line 2Ø be to get the Spectrum to print the word DOG over and over again?
1Ø PRINT "DOG ";

19. Invent a simple program of four lines, and say what happens when you RUN it.

20. How do you count from 1 to 1Ø using FOR and NEXT? Include a simple program with your answer.

APPENDIX B

Ideas for Exercises and Programs

This appendix contains a number of ideas which you can convert into exercise material for your students to help them develop the skill of writing a program to solve a specific problem, or achieve a specific aim. Other ideas will perhaps strike you as more suitable to use as starting points for programs for you to develop to use in your classes.

Develop a program, or series of programs to demonstrate the relationship between speed, time and distance.

Illustrate the properties of light (shadows, reflections and the principles of light travelling in straight lines).

Demonstrate the basic theory of electric current graphically on the Spectrum.

Archimedes principle (floating, sinking and density) is another subject which would lend itself to an animated display.

Sentence construction, word sequencing and other language skills could be tested by programs (note that these are not particularly easy to write).

Race games can be written, in which children answer, say, maths questions, and each correct answer means that the child's 'car' advances one step. This would require at least two children to have access to the Spectrum at the same time.

Programs based on the '15-tiles' puzzles in which 'tiles' containing letters or numbers must be slid around to get into a pre-determined sequence have proved popular as games.

You could ask students to write programs to achieve the ends of many of the programs in this book, such as many of the shorter ones in the mathematics chapter. For example, you could ask your students to write a program, which, given a positive whole number, will calculate the factorial (N!) of that number.

Using the formula Interest = Principal * Rate * Time/100,

get students to write a program to show interest gained over various time periods, with differing principals and interest rates (you could then ask them to do the same thing with the formula for compound interest).

Produce a table of square- and cube-roots for numbers in a specified range.

APPENDIX C

Binary converter

This is the full list of decimal numbers from zero to 255, and their binary equivalents. It is designed to help you when you are producing your own graphics, so save you having to convert from binary numbers to their decimal equivalents to put into DATA statements.

Just in case you're interested, the program which produced the list is printed after it.

```
0         00000000
1         00000001
2         00000010
3         00000011
4         00000100
5         00000101
6         00000110
7         00000111
8         00001000
9         00001001
10        00001010
11        00001011
12        00001100
13        00001101
14        00001110
15        00001111
16        00010000
17        00010001
18        00010010
19        00010011
20        00010100
21        00010101
22        00010110
23        00010111
24        00011000
25        00011001
26        00011010
27        00011011
28        00011100
29        00011101
30        00011110
31        00011111
32        00100000
33        00100001
34        00100010
35        00100011
```

36	00100100
37	00100101
38	00100110
39	00100111
40	00101000
41	00101001
42	00101010
43	00101011
44	00101100
45	00101101
46	00101110
47	00101111
48	00110000
49	00110001
50	00110010
51	00110011
52	00110100
53	00110101
54	00110110
55	00110111
56	00111000
57	00111001
58	00111010
59	00111011
60	00111100
61	00111101
62	00111110
63	00111111
64	01000000
65	01000001
66	01000010
67	01000011
68	01000100
69	01000101
70	01000110
71	01000111
72	01001000
73	01001001
74	01001010
75	01001011
76	01001100
77	01001101
78	01001110
79	01001111
80	01010000
81	01010001
82	01010010
83	01010011
84	01010100
85	01010101
86	01010110
87	01010111
88	01011000
89	01011001
90	01011010
91	01011011
92	01011100
93	01011101
94	01011110
95	01011111
96	01100000
97	01100001

98	01100010
99	01100011
100	01100100
101	01100101
102	01100110
103	01100111
104	01101000
105	01101001
106	01101010
107	01101011
108	01101100
109	01101101
110	01101110
111	01101111
112	01110000
113	01110001
114	01110010
115	01110011
116	01110100
117	01110101
118	01110110
119	01110111
120	01111000
121	01111001
122	01111010
123	01111011
124	01111100
125	01111101
126	01111110
127	01111111
128	10000000
129	10000001
130	10000010
131	10000011
132	10000100
133	10000101
134	10000110
135	10000111
136	10001000
137	10001001
138	10001010
139	10001011
140	10001100
141	10001101
142	10001110
143	10001111
144	10010000
145	10010001
146	10010010
147	10010011
148	10010100
149	10010101
150	10010110
151	10010111
152	10011000
153	10011001
154	10011010
155	10011011
156	10011100
157	10011101
158	10011110
159	10011111

```
160     10100000
161     10100001
162     10100010
163     10100011
164     10100100
165     10100101
166     10100110
167     10100111
168     10101000
169     10101001
170     10101010
171     10101011
172     10101100
173     10101101
174     10101110
175     10101111
176     10110000
177     10110001
178     10110010
179     10110011
180     10110100
181     10110101
182     10110110
183     10110111
184     10111000
185     10111001
186     10111010
187     10111011
188     10111100
189     10111101
190     10111110
191     10111111
192     11000000
193     11000001
194     11000010
195     11000011
196     11000100
197     11000101
198     11000110
199     11000111
200     11001000
201     11001001
202     11001010
203     11001011
204     11001100
205     11001101
206     11001110
207     11001111
208     11010000
209     11010001
210     11010010
211     11010011
212     11010100
213     11010101
214     11010110
215     11010111
216     11011000
217     11011001
218     11011010
219     11011011
220     11011100
221     11011101
```



```
222     11011110
223     11011111
224     11100000
225     11100001
226     11100010
227     11100011
228     11100100
229     11100101
230     11100110
231     11100111
232     11101000
233     11101001
234     11101010
235     11101011
236     11101100
237     11101101
238     11101110
239     11101111
240     11110000
241     11110001
242     11110010
243     11110011
244     11110100
245     11110101
246     11110110
247     11110111
248     11111000
249     11111001
250     11111010
251     11111011
252     11111100
253     11111101
254     11111110
255     11111111
```

```
  5 REM BINARY NUMBERS
  6 REM By Jeremy Ruston
  7 REM from 'Programming the
                ZX Spectrum'
 10 FOR B=0 TO 255
 20 LET J=B
 30 LET A$=""
 40 FOR N=0 TO 7
 50 LET T=J-INT (J/2)*2
 60 IF T=0 THEN LET A$="0"+A$
 70 IF T<>0 THEN LET A$="1"+A$
 80 LET J=INT (J/2)
 90 NEXT N
100 LPRINT B;TAB 7;A$
110 NEXT B
```

APPENDIX D

Logo, and an introduction to turtle graphics

Logo is quite different from BASIC. It was designed with the lofty aim of being a language which would 'teach learning' and, to a certain extent, this aim has been realised.

Pioneered by Dr Simon Papert when he was a Professor of Mathematics and Education at the Massachusetts Institute of Technology in the US (he has now moved to France where he is one of the leaders of the World Computer Centre), Logo is intended to be the very first programming language a person learns. The first language you learn inevitably colours the way you program, and the way you think about programming, for the rest of our life. Proponents of Logo claim that the base provided by initial exposure to Logo is a far more suitable one for future programming excellence than is a language such as BASIC.

Is there a basis for such a claim? Papert says that many teachers have only seen computers as devices which can extend the transitional ways of doing things in the classroom, rather than as utterly new teaching tools. In contrast to this, Papert says Logo is a liberating device, which enables computers to be used to teach new and important skills, including the skill of 'learning how to learn'.

Following observations made by Jean Piaget that children are able to learn quite complex skills – such as being able to talk and walk – without formal training, and the fact that this highly-effective informality was absent in traditional classroom teaching, Papert set out to create a language which would remedy this deficiency. Papert says most school instruction in computer programming puts the child almost in the position of being programmed by the computer. Logo, by contrast, puts the child firmly in charge.

It does this by allowing the programmer to create new shapes and actions – such as one which draws a triangle –

and then get the computer to execute this on demand, simply by entering TRIANGLE. BASIC has no such way of creating new commands and functions.

Put just about anyone in France, and let them live there a while, and they will become skilled French speakers, even if they had a prior concept of themselves as 'not good with languages'. The same holds true for mathematics, claims Papert. Part of Logo's function is to allow user to 'live in Mathland' where there is no such thing as a person who is 'not good at Maths'. (In an article 'Logo in the Schools' (BYTE magazine, August 1982, pp. 116–134), Daniel Watt reports that 'teachers found that students who had taken part in the Logo classes were more willing to 'argue sensibly about mathematical issues' and to explain their 'mathematical difficulties clearly'.')

When computers were first developed, memory was at a premium. Programmers had to bend their thinking to the demands of the machine (such as integer variable name starting with specific letters), regardless of how much extra work this added. The thinking that human beings should continue to humble themselves before the computer's demands has continued. Although BASIC is relatively easy for a computer to interpret, and easy to teach, it is not a flexible language, and labyrinthine program constructions are sometimes needed. Papert and his team at MIT decided when developing Logo that they would not allow their work to be limited by computer technology. Rather than gear their thinking to the cheap (for the time) computers available when they began their work in the late sixties, the team worked with the biggest mainframes they could.

The most familiar aspect of Logo is 'turtle graphics' when the computer controls the movement of a 'turtle' (a triangular shape on the screen) which leaves a trail behind it as it moves. Therefore, if the turtle moves up the screen for an inch, turns through 90 degrees and moves another inch, turns and moves, turns and moves again, it will have traced out a square.

Turtles move in 'turtle steps' (with a screen being about 200 turtle steps high). A turtle command is often in the form of a direction (such as FORWARD) followed by the number of turtle steps, so FORWARD 1ØØ would cause the turtle to move half way up the screen (FORWARD is the direction the triangular cursor is facing).

With Logo, the computer can be taught a sequence of moves, such as the one we described to trace out a square, and the sequence can be 'remembered' by the computer under the name, say, SQUARE. Then, whenever we want the turtle to draw a square, we just enter SQUARE.

A sequence of moves like this is called a *procedure*. The process of drawing a square could be even simpler. Think of FORWARD 1ØØ. The computer draws a line up the screen. The word RIGHT followed by a number turns the turtle to the right the number of degrees specified by the following number, so RIGHT 9Ø will make it turn through a right angle. Moving FORWARD 1ØØ again will draw a line at right angles to the first. Follow this with RIGHT 9Ø and the turtle will turn through another 9Ø degrees (and will now be facing down the screen). Going through the sequence FORWARD 1ØØ RIGHT 9Ø four times will draw a square.

This should give us a hint as to how the procedure SQUARE can be created more simply. The Logo word REPEAT means just that. A number follows REPEAT, and the computer repeats the instruction which follows the number however many times are specified. So, to create a square-drawing program, we need the following:

```
TO SQUARE
REPEAT 4 [FORWARD 1ØØ RIGHT 9Ø]
END
```

Note that the first line of this program, TO SQUARE, is the procedure title line. Run it, and the computer then knows what a square is, and can produce one, whenever it encounters the command SQUARE. As I'm sure you can appreciate, there is no such facility in BASIC for creating new commands at will.

Logo has other useful commands, such as CLEARSCREEN, and PENUP (which 'lift the pen' from the screen) and PENDOWN. You can draw a line, lift the pen up and move it to another part of the screen, put the pen down and continue drawing.

Looking back to our definition of the procedure SQUARE above. You can see that if we had a way of entering the size (the 100 in our example) each time we ran the program, SQUARE could be used to draw squares of any size. Logo allows for this. If you include the variable name in the procedure title line, preceded by a colon, the

computer will wait for you to enter the required information.

```
TO SQUARE :LENGTH
REPEAT 4 [FORWARD :LENGTH RIGHT 9Ø]
END
```

To run this, you enter SQUARE 64 (replacing 64 with the side length you choose).

From this, it is easy to see that we could do much more than just change the length of the side. We could easily define a procedure which allows you to specify not only the length of the side, but the number of repeats, and the angle through which the turtle will turn. If you're creating mental pictures of the effects of each of these changes, you'll see what a powerful tool we now have on our hands.

```
TO SHAPE :MANY :ANGLE :LENGTH
REPEAT :MANY [FORWARD :LENGTH RIGHT
:ANGLE]
END
```

This simple procedure holds a wealth of extraordinary effects.

To draw a triangle, with sides 35 steps long, you'd just enter:

```
SHAPE 3 12Ø 35
```

A star with each line 55 steps long could be drawn with:

```
SHAPE 5 144 55
```

We will end our brief introduction to turtle graphics by pointing out that once the computer has been taught a word such as SQUARE, this procedure can be used within further definitions. That is, if you wanted the computer to print a shape, then move just a little to the side, then draw the shape again, and repeat this a number of times, you could define the following procedure (assuming that the procedure SQUARE had previously been defined):

```
TO AMAZING :MANY
REPEAT :MANY [SQUARE 5Ø FORWARD 1]
END
```

There are four important features of Logo:

PROCEDURES: the language works by defining

sequences of steps called *procedures* which are then *called*. Procedures can incorporate other procedures. The closest BASIC equivalent (and it is generally NOT helpful to learn Logo by drawing attention to barely-equivalent BASIC statements) would be a series of subroutines which were called by name (such as GOSUB PAUSE, where PAUSE was a variable which had previously been assigned a value of the line number where the subroutine PAUSE began).

INTERACTION: Any command, whether it is one which is part of the original language (such as FORWARD or PENUP) or one defined as a procedure, can be triggered just by entering the command, such as the word SHAPE or SQUARE.

LISTS: The language supports compound structures called *lists* which are much easier to manipulate than are data structures such as arrays. They can be manipulated very flexibly. Procedures can be handled as lists.

TURTLE GEOMETRY: The 'cybernetic animal', the turtle, will follow instructions to draw shapes on the screen. Turtle graphics have proved an ideal way of introducing the concept and practice of computer programming, and also as the basis upon which a computer-based mathematics curriculum can be built.

Further reading on Logo:

Mindstorms: Children, Computers and Powerful Ideas – Simon Papert (Basic Books, New York, 1980)

Logo for the Apple II – Harold Abelson (BYTE/McGraw Hill, Peterborough, 1982)

Learning Logo on the Apple II – Anne McDougall, Tony Adams, Pauline Adams (Prentice-Hall of Australia, 1982)

The August 1982 (Volume 7, number 8) issue of BYTE magazine is dedicated to Logo and is an extremely useful introduction to both the language, and to its implications.

APPENDIX E

PROLOG – PROgramming in LOGic

In 1972 work by Kowalski, Colmerauer and Robinson culminated in Colmerauer's implementation of a new computer language PROLOG. Logic has existed since Ancient Greek times as an accurate language for expressing problems. PROLOG allows problems descriptions written in a form of logic to be run on a computer.

Since 1972 many implementation of PROLOG have been written for different machines. Now, a version of the language, micro-PROLOG [Clark, Ennals and McCabe 1981], [Clark and McCabe 1983], has been produced for the Sinclair Spectrum. Micro-PROLOG is a full implementation of PROLOG for microcomputers and the Sinclair version also supports the sound and graphics capabilities of the machine.

The language can be used in many different ways and for many different applications. One of its most powerful features is the ease with which it can be customized for a particular application by constructing a 'front-end'. This is a program that allows the use of the language in a particular manner by providing various operations. As an example a 'front-end' might be written to provide LOGO-style graphics [Steel 1983a].

At Imperial College, London a front-end called Simple [Ennals 1982] was developed for use in schools. Work in this area is by no means complete and new versions are under development at the present time. Simple is provided as part of the micro-PROLOG package for the ZX Spectrum. All the PROLOG in this chapter will use the Simple front-end syntax.

Learning to describe problems

In order to learn to program in PROLOG very few

'traditional' computing ideas are necessary. What is necessary is the ability to describe things clearly and accurately. As an example, we will examine PROLOG's use in exploring woodland ecology [Steel 1983b]. This subject is sufficiently well understood by non experts to be explored without us having to learn the subject first. It has formed the basis for a number of introductory courses for teachers learning PROLOG.

A program usually starts with a collection of simple facts.

owl is-a bird
sparrow is-a bird
fly is-a insect
hogweed is-a herb
beech is-a tree
DDT is-a insecticide
mole is-a mammal
mole eats hogweed
sparrow eats seeds

Facts such as these can be given to the computer exactly as they are, using the Simple command *add ( )*.
e.g. add (owl is-a bird)

Small changes have to be made to the English sentences we would normally use in order that they are accepted by Simple PROLOG. Each sentence, in this case, has been split into three parts: the names of two objects and a relationship between them. Hyphens are used when we would normally use a space in English.

Rules, connecting these facts, can also be added.

x is-a plant *if* x is-a herb (i.e. something is a plant if it is a herb)
x is-a plant *if* x is-a tree
x is-a animal *if* x is-a bird
x is-a animal *if* x is-a mammal
x eats y
  if x is-a bird
  and y is-a insect

In these rules x and y are variables. They stand for unknown values. The last rule contains the information that all birds eat insects. We will complete our ecology example with a few more rules.

x bad-for y *if* x eats y
x bad-for y *if* x is-a insecticide
  and y is-a insect
x bad-for y *if* y is-a plant
  and x eats seeds

Once a collection of facts and rules have been established, the Simple front-end provides a set of questions that can be asked to use the information. The first sort of question checks that some fact is known by the computer. Does the computer know, for example, that a mole is an animal? We can ask this question as follows.

does (mole is-a animal)
YES

Only information known by the computer is used to answer the questions asked. Sometimes this will lead to surprising answers.

does (cat is-a animal)
NO

The computers answer NO because it does not know about a cat.

A much more powerful question is provided when a YES/NO answer is insufficient. It will allow us to find out what an owl eats.

which (x: owl eats x)

i.e. in English (give an x such that owl eats x)

answer is fly
no (more) answers

A which question may produce more than one answer and its form can be more general.

which ((x y): x bad-for y)

i.e. (give x and y such that x is bad for y)

answer is (mole hogweed)
answer is (sparrow seeds)
answer is (owl fly)
answer is (sparrow fly)
answer is (DDT fly)
answer is (sparrow hogweed)

```
answer is (sparrow beech)
no (more) answers
```

It is important to see that PROLOG uses rules to find answers to questions. Unlike database query packages much of the information can be generalised. Writing such general rules is a useful activity for a class learning a particular topic.

A third question that is particularly useful in a learning environment is why. This allows the computer to explain the rules it has used in obtaining an answer to a which or does question.

```
why (sparrow bad-for hogweed)
(sparrow bad-for hogweed) shown by
      (hogweed is-a plant) and
      (sparrow eats seeds)
(hogweed is-a plant) shown by
      (hogweed is-a herb)
(hogweed is-a herb) is given
(sparrow eats seeds) is given
```

Expert Systems

Programs, similar to this simple ecology example, could be written for numerous different school subject areas. On a larger scale programs using the same ideas are being written to help doctors, lawyers and planners. These are known as expert systems: systems that encapsulate the specialist knowledge of an expert. Once constructed these programs can be consulted to answer questions, they can explain their answers and with a different 'front-end' they can ask questions themselves, putting the expert system in the role of the teacher.

Within the next ten years we may see expert systems replacing the computing packages being used in schools today. By using PROLOG it is possible to explore these ideas now, constructing or consulting expertise in areas of history, geography or the sciences.

Acknowledgement

Work by the author, on Logic as a Computer Language for Children, is supported by Sinclair Research Ltd, Cambridge.

References

[Clark, Ennals and McCabe 1981]
Clark K L, Ennals J R, McCabe F G, A micro-PROLOG Primer, Logic Programming Associates Ltd, 1981.

[Clark, McCabe 1983]
Clark K L, McCabe F G (eds), micro-PROLOG: Programming in Logic, Prentice-Hall (to be published 1983).

[Ennals 1982]
Ennals J R, Beginning micro-PROLOG, Ellis Horwood and Heinemann Computers in Education 1982.

[Steel 1983a]
Steel B D, North Star Advantage Graphics in micro-PROLOG 3.0, Department of Computing, Imperial College.

[Steel 1983b]
Steel B D, An Expert System Workshop (unpublished).

APPENDIX F

Software houses with Spectrum educational software

(As with magazines, with new ones being born each week, new software houses appear to spring up each day. Refer to the magazines in the previous appendix for details of new companies. This list is of houses which have been around – comparatively speaking – a long time. The inclusion of a company in this list is not necessarily an endorsement of its products, and non-inclusion of any company in this list does not infer or imply anything regarding the products distributed by that company.)

Calpac Computer Software,
108 Hermitage Woods Crescent,
St Johns, Woking, Surrey GU21 1UF
Junior Education. Eight programs for 7 to 11–age group English, Maths, Science, etc. *O and CSE Chemistry.* Elements, structure, bonding, etc

E.Z.U.G. (Educational ZX Users' Group),
Highgate School, Birmingham 12
Large number of tested programs

Rose Cassettes,
148 Widney Lane, Solihull,
West Midlands, B91 3LH
Intermediate Maths 1 & 2, English 1 & 2, *O level* French revision, maths revision, arithmetic for the under eights, educational quiz

SCISOFT,
5 Minster Gardens, Newthorpe,
Eastwood, Notts., NG16 2AT
O and CSE study/revision packs in Physics, Chemistry, Biology, Computer studies (logic gates, etc.), Maths part 1 & 2, Teacher's Markbook, *junior* Jungle Maths, Astromaths, Magic Spell

Simon Software,
FREEPOST,
New End, Redditch, Worcestershire
11 to 16 age range. Maths 1, 2, 3, 4 & 5, covering area, ratio, percentages, angles, square roots, indices, equations, etc.

APPENDIX G

Publications which regularly include reviews of Spectrum software

(Please note this list is by no means comprehensive, although it represents the publications which – at the time the list was compiled – appeared to give regular, and indepth, reviews of software. As you know, new periodicals appear to be launched every week, so you should check your newsagent for the latest titles.)

Educational Computing,
MAGSUB,
Oakfield House,
Perrymount Road,
Haywards Heath,
West Sussex, RH16 3HD

Popular Computing Weekly,
Hobhouse Court,
19 Whitcombe Street,
London, WC2

Sinclair User,
ECC Publications,
30–31 Islington Green,
London, N1 8BJ

ZX Computing,
145 Charing Cross Road,
London, WC2 0EE

Personal Computing Today
145 Charing Cross Road,
London, WC2H 0EE
(This magazine contains a 'software checklist')

Your Computer.
IPC Electronic Press,
Quadrant House,
The Quadrant,
Sutton,
Surrey, SA2 5AS

APPENDIX H

Suggestions for further reading

Many magazine articles, and several books, have addressed themselves to the subject of computers and education. I have followed the published material with much interest. In this chapter are a number of avenues you may wish to pursue to continue to develop your knowledge in this area.

Magazine Articles:

Memory Recall Test (V Nasser), *Compute!*, December 1981, p. 129. The article contains a program written for the Superboard II but it would be easy to adapt for another BASIC.

Peripheral Vision Exerciser (Ron Kushnier), *Compute!*, September 1982, p. 28. Two versions of the program, which is intended to increase reading speed, are given. One in a general form of BASIC and another specifically for the Atari.

Learning With Computers (Glenn Keiman), *Compute!*, September 1982, pp. 96–99. General overview of the field at that time in America, with a valuable appendix of American periodicals which cover the field.

The Home Computer as a Teaching Aid (Joe Aitken), *Electronics & Computing*, March 1982, pp. 38–39. Article includes ZX BASIC listings for a letter recognition program, a flash card program, and a third program which demonstrates the workings of the N.P.N. transistor.

Which Computer Should a School Buy? (Dan Watt), *Popular Computing*, December 1982, pp. 140–144; surveys six cheap computers and assesses their value for education.

How computation process has changed (John Dawson), *Educational Computing*, October 1980, p. 15. From the

same issue: Proving that little is always best (Jim Cocallis), p. 26. Cocallis presents a solid case for buying a number of cheaper micros to ensure the maximum number of students have hands-on access to them.

The Micro-Mathematician (Richard R Parry), *Interface Age*, December 1980, pp. 36–40. Several methods of plotting functions are discussed, together with a program to dump graph output direct to a printer.

Final Exams – Let the Computer Write Them (Bernard Eisenberg), *Creative Computing*, November/December 1977, pp. 103–106. Fascinating discussion on how a program was written to generate a mathematics exam for students on a remedial course. One examination paper set by the computer is reproduced in full in the article.

Reading Matters (Mike and Wendy Cook), *Practical Computing*, April 1982, pp. 100–102. Article, with program, to allow the computer to function as a tachistoscope, presenting a series of letters or figures on the screen at a given rate to help children or adults with reading difficulties.

Apples, Computers and Teachers (Don Inman), *Interface Age*, October 1979, pp. 68–72. Article describes a short course given to teachers to acquaint them with possible uses of the computer in schools.

Instructional Software (Walter Koetke), *Microcomputing*, July 1981, pp. 20–22. General comments on the standard of some educational software on the US market, plus a detailed list of recommended programs.

Two Short Programs of CAI for Teaching BASIC (R. Hiatt), *Compute!*, March 1982, pp. 56–60.

Books:

An Introduction to Microcomputers in Teaching, Andrew Nash and Derek Ball (Hutchinson & Co. Ltd, London, 1982). Recommended without qualification. This is a detailed introduction to ways in which microcomputers can be used in teaching. Topics discussed include: Evaluating a program; Computer graphics versus other means of presenting pictures; The function of the computer in learning; Program code and portability.

Microcomputers in the Classroom, Alan Maddison (Hodder and Stoughton Educational, Sevenoaks, 1982). Another fine book, which discusses (among many other things) using the computer as an 'electronic blackboard' and computer managed learning.

Practical Guide to Computers in Education, Peter Coburn *et al.* (Addison-Wesley Publishing Company, London, 1982). This book, plus the two mentioned above, completes the cornerstone of a solid library on the field. An overview, complete with examples, of computer applications in the classroom indicates how wide the potential application of the machine can be in this area. A warning is also given not to expect too much when first introducing a computer into a particular class.

Microcomputers and the 3 R's, Christine Doerr (Hayden Book Co., Rochelle Park, New Jersey, 1979). Ways of using computers in schools to increase student motivation to learn, and a discussion of the capabilities and limitations of computers in school make this a worthwhile backup volume to the first three mentioned.

Computer Software for Schools, A Payne, B Hutchings, P Ayre (Pitman Books Ltd, London, 1982). The authors explain they were prompted to write this book by the scarcity of suitable programs for use in schools when they first investigated computer use in their own school (Oriel Grammar School). The book contains ten complete progams, all of them carefully annotated. Programs include DRAKE (a history simulation), MENU (nutritional analysis and VERB (French drill exercise).

Microcomputers in Education, ed. Christopher Smith (Ellis Horwood Ltd, Chichester, 1982). The book brings together a wide range of contributors, all of whom have practical experience of computer use in education. It covers many of the problems that derive from the use of micros in education and the training of teachers, looking at such applications as school administration, special education, computer graphics and classroom experiments.

Microcomputers in Science Teaching, R A Sparkes (Hutchinson and Co. Ltd, 1982). Although the book is biased towards use of Commodore PET computers for

teaching Physics, the clear (and numerous) programs give many leads for developing such programs for your own class use. Program titles include DOUBLE DENSITY GRAPH PLOT, CHEMICAL NAMES and TRIANGULAR WAVEFORM OUTPUT.

The ZX Spectrum Explored, Tim Hartnell (Sinclair Browne, London, 1982). Companion to this volume, covers a wide field (including business uses of the Spectrum and 3-D graphics, plus an introduction to machine code). One section (pp. 77–110) concentrates on educational uses of the computer. This section was written by Jeff Warren, an experienced teacher who runs a company selling educational software for the Spectrum.

Microl Spectrum, USE AND LEARN – 25 BASIC programs supplied on cassette with 118-page book designed to show the potential of the Spectrum, and help develop program-writing skills. Programs include BINARY SEARCH, SHELL SORT and WORLD ATLAS.

Pocket Computer Primer, Hank Librach (Micro Text Publications Inc. New York, 1982). Although designed for use with 'pocket computers', as its title indicates, this book contains a number of easy-to-adapt programs which could be of interest. Programs include a survival simulation, an algebra teacher and a program for working out classroom statistics.

Learning with the Spectrum, Eric Deeson (AVC Software, Harborne, 1982). Despite its small size, this booklet contains ten very useful progams, including a 'mini-LOGO' program. The notes on all programs are especially helpful.

Educare's 50 1K Programs for Primary Education on the ZX81, K S Goh (Educare, London, 1981). Although written for the ZX81, nearly all programs can easily be converted to the Spectrum. Programs are divided into categories, including arithmetic, mathematical concepts, fun with words, graphics, special topics and 'fun and games'.

Pocket Calculator Maths, J Shelton (Collins, Glasgow and London, 1981). Although written for calculator use, with care the book can be used with a computer, and should aid the development of numerical skills.

The Computer Tutor, Gary W Orwig and William S Hodges (Winthrop Publishers, Inc, Cambridge, Massachusetts, 1982). A number of worthwhile program ideas in this book, including a ballistics simulation one, and a quiz to test synonyms and antonyms.

APPENDIX I

Glossary of Computer Terms

Accumulator – part of the computer's logic unit which stores the intermediate results of computations
Address – a number which refers to a location, generally in the computer's memory, where information is stored
Algorithm – the sequence of steps used to solve a problem
Alphanumeric – generally used to describe a keyboard, and signifying that the keyboard has alphabetical and numerical keys. A numeric keypad, by contrast, only has keys for the digits one to nine, with some additional keys for arithmetic operations, much like a calculator
APL – this stands for A Programming Language, a language developed by Iverson in the early 1960s, which supports a large set of operators and data structures. It uses a non-standard set of characters
Application software – these are programs which are tailored for a specific task, such as word processing, or to handle mailing lists.
ASCII – stands for American Standard Code for Information Interchange. This is an almost universal code for letters, numbers and symbols, which has a number between 0 and 255 assigned to each of these, such as 65 for the letter A
Assembler – this is a program which converts another program written in an assembly language (which is a computer program in which a single instruction, such as ADD, converts into a single instruction for the computer) into the language the computer uses directly
BASIC – stands for Beginner's All-purpose Symbolic Instruction Code, the most common language used on microcomputers. It is easy to learn, with many of its statements being very close to English
Batch – a group of transactions which are to be processed by a computer in one lot, without interruption by an operator

Baud – a measure of the speed of transfer of data. It generally stands for the number of bits (discrete units of information) per second
Benchmark – a test which is used to measure some aspect of the performance of a computer, which can be compared to the result of running a similar test on a different computer
Bit – an abbreviation for binary digit. This is the smallest unit of information a computer circuit can recognise.
Binary – a system of counting in which there are only two symbols, 0 and 1 (as opposed to the ordinary decimal system, in which there are ten symbols, 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9). Your computer 'thinks' in binary
Boolean Algebra – the algebra of decision-making and logic, developed by English mathematician George Boole, and at the heart of your computer's ability to make decisions
Bootstrap – a program, run into the computer when it is first turned on, which puts the computer into the state where it can accept and understand other programs
Buffer – a storage mechanism which holds input from a device such as keyboard, then releases it at a rate which the computer dictates
Bug – an error in a program
Bus – a group of electrical connections used to link a computer with an ancillary device, or another computer
Byte – a group of bits, generally 8, which are written to or read from memory as a single unit. Information is always stored in whole bytes
Central Processing Unit (CPU) – the heart of the computer, where arithmetic, logic and control functions are carried out
Character code – the number in ASCII (see ASCII) which refers to a particular symbol, such as 32 for a space and 65 for the letter 'A'
COBOL – stands for Common Business Orientated Language, a standard programming language, close to English, which is used primarily for business
Compiler – a program which translates a program written in a high level (human-like) language into a machine language which the computer understands directly
Concatenate – to add two strings together
CP/M – stands for Control Program/Monitor, an almost universal disk operating system developed and marketed by Digital Research, Pacific Grove, California, whose trademark it is.

Data – a general term for information processed by a computer
Database – a collection of data, organised to permit rapid access by computer
Debug – to remove bugs (errors) from a program
Disk – a magnetic storage medium (further described as a 'hard disk', 'floppy disk' or even 'floppy') used to store computer information and programs. The disks resemble, to a limited extent, 45 rpm sound records, and are generally eight, five and a quarter, or three inches in diameter. Smaller 'microdisks' are also available for some systems
Documentation – the written instructions and explanations which accompany a program
DOS – stands for Disk Operating System (and generally pronounced 'doss'), the versatile program which allows a computer to control a disk system
Dot-matrix printer – a printer which forms the letters and symbols by a collection of dots, usually on an eight by eight, or seven by five, grid
Double-density – adjective used to describe disks when recorded using a special technique which, as the name suggests, doubles the amount of storage the disk can provide
Dynamic memory – computer memory which requires constant recharging to retain its contents
EPROM – stands for Erasable Programmable Read Only Memory, a device which contains computer information in a semi-permanent form, demanding sustained exposure to ultra-violet light to erase its contents
Error messages – information from the computer to the user, sometimes consisting only of numbers or a few letters, but generally of a phrase (such as 'Out of memory') which points out a programming or operational error which has caused the computer to halt progam executions
Field – a collection of characters which form a distinct group, such as an identifying code, a name or a date; a field is generally part of a record
File – a group of related records which are processed together, such as an inventory file or a student file
Firmware – The solid components of a computer system are often called the 'hardware', the programs, in machine-readable form on disk or cassette, are called the 'software', and programs which are hardwired into a circuit, are called 'firmware'. Firmware can be altered, to a limited extent, by

software in some circumstances
Flag – this is an indicator within a program, with the 'state of the flag' (i.e. the value it holds) giving information regarding a particular condition
Floppy disk – see disk
Flowchart – a written layout of program structure and flow, using various shapes, such as a rectangle with sloping sides for a computer action, and a diamond for a computer decision, is called a flow chart. A flowchart is generally written before any lines of program are entered into the computer
FORTRAN – a high level computer language, generally used for scientific work (from FORmula TRANslation)
Gate – a computer 'component' which makes decisions, allowing the circuit to flow in one direction or another, depending on the conditions to be satisfied
GIGO – acronym for 'Garbage In Garbage Out', suggesting that if rubbish or wrong data is fed into a computer, the result of its processing of such data (the output) must also be rubbish
Global – a set of conditions which affects the entire program is called 'global', as opposed to 'local'
Graphics – a term for any output of computer which is not alphanumeric, or symbolic
Hard copy – information dumped to paper by a printer.
Hardware – the solid parts of the computer (see 'software' and 'firmware')
Hexadecimal – a counting system much beloved by machine code programmers because it is closely related to the number storage methods used by computers, based on the number 16 as opposed to our 'ordinary' number system which is based on 10)
Hex pad – a keyboard, somewhat like a calculator, which is used for direct entry of hexadecimal numbers
High-level languages – programming languages which are close to English. Low-level languages are closer to those which the computer understands. Because high-level languages have to be compiled into a form which the computer can understand before they are processed, high-level languages usually run more slowly than their low-level counterparts
Input – any information which is fed into a program during execution

I/O – stands for Input/Output; an I/O port is a device the computer uses to communicate with the outside world

Instruction – an element of programming code, which tells the computer to carry out a specific task. An instruction in assembler language, for example, is ADD which (as you've probably guessed) tells the computer to carry out an addition

Interpreter – converts the high-level ('human-understandable') program into a form which the computer can understand

Joystick – an analogue device which feeds signal into a computer which is related to the position which the joystick is occupying; generally used in games programs

Kilobyte – a unit of storage measurement; one kilobyte (generally abbreviated as K) equals 1024 bytes.

Line printer – a printer which prints a complete line of characters at one time

Low-level language – a language which is close to that used within the computer (see high-level language)

Machine language – the step below a low-level language; the language which the computer understands directly

Mainframe – the term for 'giant' computers such as the IBM 370. Computers are also classed as mini-computers and microcomputers (such as the computer you own)

Memory – the device or devices used by a computer to hold information and programs being currently processed, and for the instruction set fixed within a computer which tells it how to carry out the demands of the program. There are basically two types of memory (see RAM and ROM)

Microprocessor – the 'chip' which lies at the heart of your computer. This does the 'thinking'

Modem – stands for MOdulator/DEModulator, and is a device which allows one computer to communicate with another via the telephone

Monitor – (a) a dedicated television-screen for use as a computer display unit, contains no tuning apparatus; (b) a program within a computer which enables it to understand and execute certain instructions

Motherboard – a unit, generally external, which has slots to allow additional 'boards' (circuits) to be plugged into the computer to provide facilities (such as high-resolution graphics, or 'robot control') which are not provided with the standard machine

Mouse – a control unit, slightly smaller than a box of cigarettes, which is rolled over the desk, moving an on-screen cursor in parallel to select options and make decisions within a program. 'Mouses' work either by sensing the action of their wheels, or by reading a grid pattern on the surface upon which they are moved

Network – a group of computers working in tandem

Numeric pad – a device primarily for entering numeric information into a computer, similar to a calculator

Octal – a numbering system based on eight (using the digits 0, 1, 2, 3, 4, 5, 6 and 7)

On-line – device which is under the direct control of the computer

Operating system – this is the 'big boss' program or series of programs within the computer which controls the computer's operation, doing such things as calling up routines when they are needed and assigning priorities

Output – any data produced by the computer while it is processing, whether this data is displayed on the screen or dumped to the printer, or is used internally

Pascal – a high level language, developed in the late 1960s by Niklaus Wirth, which encourages disciplined, structured programming

Port – an output or input 'hole' in the computer, through which data is transferred

Program – the series of instructions which the computer follows to carry out a predetermined task

PILOT – a high level language, generally used to develop computer programs for education

RAM – stands for Random Access Memory, and is the memory on board the computer which holds the current program. The contents of RAM can be changed, while the contents of ROM (Read Only Memory) cannot be changed under software control

Real-time – when a computer event is progressively in line with time in the 'real world', the event is said to be occurring in real time. An example would be a program which showed the development of a colony of bacteria which developed at the same rate that such a real colony would develop. Many games, which require reactions in real time, have been developed. Most 'arcade action' programs occur in real time

Refresh – The contents of dynamic memories (see memory) must receive periodic bursts of power in order for them to

maintain their contents. The signal which 'reminds' the memory of its contents is called the refresh signal

Register – a location in computer memory which holds data

Reset – a signal which returns the computer to the point it was in when first turned on

ROM – see RAM

RS-232 – a standard serial interface (defined by the Electronic Industries Association) which connects a modem and associated terminal equipment to a computer

S-100 bus – this is also a standard interface (see RS-232) made up of 100 parallel common communication lines which are used to connect circuit boards within micro-computers

SNOBOL – a high level language, developed by Bell Laboratories, which uses pattern recognition and string manipulation

Software – the program which the computer follows (see firmware)

Stack – the end point of a series of events which are accessed on a last in, first out basis'

Subroutine – a block of codes, or program, which is called up by another program

Syntax – as in human languages, the syntax is the structure rules which govern the use of a computer language

Systems software – sections of code which carry out administrative tasks, or assist with the writing of other programs, but which are not actually used to carry out the computer's final task

Thermal printer – a device which prints the output from the computer on heat-sensitive paper. Although thermal printers are quieter than other printers, the output is not always easy to read, nor is the used paper easy to store

Time-sharing – this term is used to refer to a large number of users, on independent terminals, making use of a single computer, which divides its time between the users in such a way that each of them appears to have the 'full attention' of the computer

Turnkey system – a computer system (generally for business use) which is ready to run when delivered, needing only the 'turn of a key' to get it working

Volatile memory – a memory device which loses its contents when the power supply is cut off (see memory, refresh, ROM and RAM)

Word processor – a dedicated computer (or a computer

operating a word-processing program) which gives access to an 'intelligent typewriter' with a large range of correction and adjustment features