Introduction to Microcomputing with the PET

J. Arotsky, J. Taylor and D. W. Glassbrook

Introduction to Microcomputing with the PET

J Arotsky, J Taylor and D W Glassbrook

The St Helens College of Technology

Edward Arnold

First published 1983
by Edward Arnold (Publishers) Ltd
41 Bedford Square, London WC1B 3DQ

British Library Cataloguing in Publication Data
Arotsky, J.
Introduction to microcomputing with the PET.
1. PET (Computer)—Programming
2. Basic (Computer program language)
I. Title II. Taylor, J.
III. Glassbrook, D.W.
001.64'24 QA76.8.P47

ISBN 0-7131-3475-5

Filmset in 10/11 pt English Times by
CK Typesetters Ltd., Sutton, Surrey

Printed in Great Britain by
Thomson Litho Ltd, East Kilbride, Scotland

Preface

Computing originally implied large, expensive installations locked away behind closed doors and ruled over by the data processing manager. The only interface that the public had with the computer was when they received through the post a final demand for $0 or fifteen identical circulars. Gradually this changed as time sharing was introduced, allowing numbers of relatively cheap terminals to share the use of one very expensive computer processor. The first people to benefit from this were scientists and engineers, who used the computer to do very complex calculations very quickly. Principally because 'amateurs' could now get access to the computer, demand mushroomed for different types of computing. The advent of the microprocessor has changed everything still further. Microprocessors are computer processors which are contained on a single chip of silicon. Although the microprocessor-based computer is not the ultimate in power it is still a very good and very useful machine, costing hundreds instead of hundreds of thousands of pounds.

This book is written for the person who has access to a microcomputer, specifically the Commodore PET. It should be possible to use it with most other microcomputers which support BASIC: there may be a number of differences in what you are allowed to do, but with judicious use of the instruction manual for your machine you should be able to overcome any problems.

The book is intended primarily for beginners who want to learn how to program a computer using BASIC but who are not necessarily mathematical. We have not aimed at any syllabus and we are not trying to prepare the student to pass any examination. We have designed the book with the home computer enthusiast in mind: someone who has only a basic machine with a cassette unit. If you have more computer equipment then you will be able to do more eventually.

It should take you a total time of 30 to 100 hours to work through the book. If you do it in less than 30 hours then you are probably cheating and wasting your time. It may well take you longer than 100 hours; do not despair.

St Helens
April 1982

JA
JT
DWG

Acknowledgements

We are grateful to the following, without whom this book would have never been written. The Governors of The St Helens College of Technology; the Principal Mr C P Gray and in particular Mr A C Hayward, the Vice Principal, who created the environment; the Computer staff, Miss J Bushell and Mr D Fraser, for their constant encouragement and assistance; and last but not least our long-suffering wives, in particular Mrs C I Arotsky who typed most of the manuscript.

Contents

1 Introduction

Computers actually work via a series of pulses of electricity. The computer is designed internally so that the patterns made and the paths taken by the pulses can be interpreted to mean something. If we want the computer to do something, then we must tell it exactly how to do it. Rather than talk to the machine we give it instructions. If you ask a person to do something but they do not speak the same language as you then you will have some difficulties. Similarly, to instruct a computer and for the computer to be able to carry out the instructions we need a computer language which both the computer and the programmer can understand. There are two types of language, high level and low level.

Low level languages are the next step up from setting each piece of information into the computer by hand. They are extremely flexible, but require a large number of instructions in order to carry out the most simple tasks. It is also rather difficult to discover what mistakes you have made in writing a program in a low level language. Do not mistake the problems of low level languages with inefficiencies. Low level languages are very efficient in terms of the speed at which a program is executed and the amount of memory that is used. The reason why people don't use them is that they are somewhat hard to use.

The next step was to make programming easier; hence high level languages were developed to make the programmer's life easier. When using a high level language one communicates with the computer in a language quite close to English, and the operating system which controls the computer converts your instructions to the machine code which the computer can understand. There are many methods of conversion: two of the most important types are compilation and interpretation.

Most programming languages are of the compiled type, where the whole of the program is read and translated in full into machine code. Compiled programs run very fast and efficiently, but to run a compiler usually requires a means of storing the intermediate codes and final results. This either takes a considerable time on tape or requires the use of a faster access device such as a disk, which is expensive (or used to be).

The alternative to the compiled language is an interpretive method. This is where each instruction is converted to machine code and acted upon, and the program then moves to the next instruction. This form of language is a bit easier to use than a compiled language and does not require additional equipment such as tapes and disks. In most microcomputers the software writers have decided to use this concept of interpretive language, particularly for using BASIC (Beginners All-purpose Symbolic Instruction Code) which was originally developed in 1963 at Dartmouth College in the USA.

Why we wrote this book

What do you use computers and computing for? If we were writing a book about what computers are currently used for in industry then the answer to this question would be easy. We could talk about payrolls, stock control, tax demands, vehicle licensing, etc. We could talk about the prediction of flow rates in rivers, the structures of atoms, and the stresses on bridges. But this does not answer the question of what to do with a group of people who want to study programming for one evening a week. What we did first was to assume the common denominator that one normally assumes in a college. This is mathematics: we showed how the computer can be used to solve mathematical equations. Unfortunately the majority of our students had either not done mathematics for twenty years or had never done it at all. It was then that we discovered a very interesting and very obvious fact. It is very easy to talk about computing to a homogeneous group of people such as engineers, or bankers, or primary school teachers. It is very difficult to talk about computing to a heterogeneous group where the only common interest is computing. We also realised how informed and gifted the modern data processing specialist must be: he is required to know as much about accounts as an accountant, as much about personnel files as the personnel manager, as much about stock control as the chief purchasing officer, etc. With microcomputers we are in a different environment, particularly in a book like this where we are participating in teaching people how to program. We are not, however, aiming at producing data processing specialists. We are aiming at showing people how to use microcomputers. This means that we want people to use microcomputers for their own interests. The difficulty is to illustrate programming with problems which will be of general interest; accordingly we have used examples such as games. We make no apologies for this since we assume that everyone plays games. We have also used examples in home finance, drawing diagrams and some general teaching. We have avoided programs of specific interest because the extra discussion which would be required to explain the specialised principles involved would make the book very large and very boring.

The book, therefore, is intended to provide an introduction to the general principles of computing as they are applied to microcomputers. Once the reader has completed and understood the book he should be able to apply the principles to his own specific area of interest.

More about computing

The closest discipline that we feel approximates to computing is gardening. Gardening is intellectually very easy, but to be a successful gardener requires careful attention to detail and a lot of hard work. However, anyone can be a good gardener provided that they pay attention to detail and put in the necessary hours.

The same is true of computing. The actual process of computing is not intellectually demanding at the level of writing a program. The intellectual requirement is mastering the problem that we want to program. Later in the book there is an example of a program for a family budget. Before we can write a program for a family budget we need to answer lots of questions like:

1) what is a family budget?
2) what sort of information must be fed in?
3) what sort of information do we want to get out?
4) in what form do we want the information?

The process of understanding computing therefore requires a lot of work and careful attention to detail. One of the problems with computers which use BASIC is that it is possible for someone to write a program that works after half an hour's instruction or 15 minutes with an instruction manual. This is a problem because many people then go away assuming that they know all about computing. We consider that most people who have not had a formal course in computing will benefit by working through this book. Those 'experts' in BASIC should be able to get through it more quickly. We have been quite surprised at finding the gaps in the knowledge of people who are writing quite complex programs. It is important to realise that as far as we are concerned you should not try to write any programs, other than the problems given in the text, until you have worked through the whole book. Once you have worked through the book then you should be equipped with the necessary knowledge to be able to write programs in BASIC.

We have written the book around the Commodore computers*. We consider that it is necessary to write the book to a specific version of BASIC. When you become more expert you will realise that there are a very large number of different BASIC dialects. The differences are not very great and we have found that it is very easy to adapt from one BASIC tc another. In our college students use up to three different forms of BASIC with three different computers. However we think it would be very confusing to try and point out all the variants of BASIC dialect within the one book. If you are using this book with a different computer then you will need to cross-reference the book with the instruction manual of your computer. This will ensure that you are using the appropriate BASIC statements or commands. If you are considering purchasing a microcomputer then get one which has the following specifications:

a) at least 8K bytes of RAM memory (RAM stands for Random Access Memory. This is the memory in which the computer stores all variables that are necessary for it to function. Such variables include the program, the values of variables, what should be on the screen, etc. The size of RAM is normally quoted in *bytes*. Each byte corresponds roughly to 1 character; 8K bytes thus correspond to 8000 characters.);
b) most of the standard Dartmouth BASIC instructions;
c) the ability to accept one character at a time from the keyboard, e.g. GET;
d) facilities to attach a tape cassette plus additional output ports, e.g. parallel port or RS232 port;
e) the ability to perform string manipulation functions, e.g. MID$ or SEG$;
f) supports data file handling and statements like OPEN and CLOSE.

How to use this book

Writing computer programs takes a long time. Some people talk in terms of man-years for writing complex programs. It follows that it will take a fair time for you to work through this book; if you cannot spare the time then take up another hobby. As we shall demonstrate, programming is not enormously difficult. We have indicated in the Preface that we expect it to take between 30 and 100 hours to complete the book, but we would not be surprised if 100 hours turned out to be a low estimate. The book is designed so that you can work at your own pace; we hope that we have written it in such a way that you will enjoy the work.

The main part of the book is highly sectionalised. After the introductory chapters, it is divided into two parts, one covering the fundamentals (Chapter 3) and the second covering more advanced topics (Chapters 4 to 11). Each chapter is broken down into numbered sections which you should work through in order. Chapter 3, Simple Programming, is also broken up into logical groupings which allow the reader to stop and reinforce what he has learnt by trying some examples and exercises; you might aim to cover one such grouping in, say, a week. You should try each worked example before looking at our solution and you should attempt all the problems. If you do not then you may not have gained any benefit from the book. If you think that you already know some computing then you should be able to complete the tasks very quickly. *Do not try to do the exercises before you have read the text.*

There are also sets of exercises at the ends of Chapters 4 (Arrays) and 5 (Strings). In addition, there are a number of problems scattered throughout the book for which solutions will be found in the appendices at the back. Do not despair if you cannot do these at the first attempt. The solutions are to be used as a last resort, if you get really stuck; their principal

*Note however that the text of this book will work with the VIC 64.

use should be to compare the elegance of your solution. You will also find here listings of some of the programs which appear in the main text, together with example runs.

Other appendices give a summary of the BASIC statements and functions, and some suggestions for further reading. There is also a summary of error messages, which will appear on the screen when you do something wrong. (Note 'when': everyone programming a computer gets errors, particularly in the beginning.)

What skills you need to learn computing

The following skills are required before you can learn to program—pause here with baited breath to see whether you are one of the select few.

1) The ability to read and write.
2) A knowledge of the four basic arithmetic operators, add, subtract, multiply and divide. In principle you need only a knowledge of these: the computer will do the sums for you. You should, however, be aware that 2 × 2 is approximately equal to 5 and not to 5000.
3) A knowledge of the keyboard. We were originally going to say 'an ability to type' but we decided that that was a bit strong. We have no intention of turning part of the book into a PET instruction manual: the reader is referred to the machine handbook for a detailed discussion of the layout of the keyboard. We emphasise here the following.
   (a) The keyboard is the means by which the operator, in this case you, converses with the computer. It is not the only way you can input data into the computer; the cassette unit, for example, is another.
   (b) The letters of the keyboard are laid out more or less according to the standard typewriter. It is called the QWERTY keyboard after the left-hand middle row of letters.
   (c) The numeric keypad, with the numbers 0 to 9, is on the right-hand side of the keyboard. The numeric keypad also houses the arithmetic operators +, −, * and / for adding, subtracting, multiplying and dividing.
   (d) Along the top of the keyboard are various other keys: the most important are the inverted commas and brackets.
   (e) Along the top of the numeric keypad are the four keys for controlling the cursor; we shall deal with these later.
   (f) Initially we will use only capital letters and we shall generally deal with keys in their unshifted mode. The use of the SHIFT key will be dealt with later.

The book is designed so that you will do a lot of typing. In fact one of the uses of a microcomputer is as a word processor. Strictly speaking it is best that you learn to type (the in-phrase is that you should acquire keyboard skills). If you do not wish to do so then this will be a disadvantage, but not a serious one. It is, however, very important that you use two hands when inputting data into the computer. We think that the minimum should be one finger on each hand and your thumbs (which normally operate the space bar at the bottom).

Software

The computer itself and its peripheral units such as disk drives and printers are known as hardware. The term software is used to describe collectively the programs which control the computer and those which the user might incorporate into his programs to save effort and to improve the efficiency of his programs. It could even be taken to mean any device which helps the programmer to make the best use of his machine, and thus books and manuals associated with computing are often considered to be part of the software. In fact, now that computer programs are often published in book form, books could well comprise the major part of any software. Essential books are the manual for the particular machine; and, of course, this one. Owners of microcomputers generally do not realise how much software is required to run any computer system efficiently. Software costs are always an appreciable part of the total cost of a computer; for example software contributes 50% of the overall cost of the minicomputer that we have at the college.

One of the biggest criticisms of microcomputers is of the quality of the manuals. In fact the manuals are often inadequate rather than poor. If you want to make the best use of your computer you need access to a library of books which supplement your instruction manual. We have included a list of books in the appendices. You do not need to purchase all of them, but we think that you should be prepared to spend a minimum of 10% of the cost of a computer on additional books.

2
The direct mode

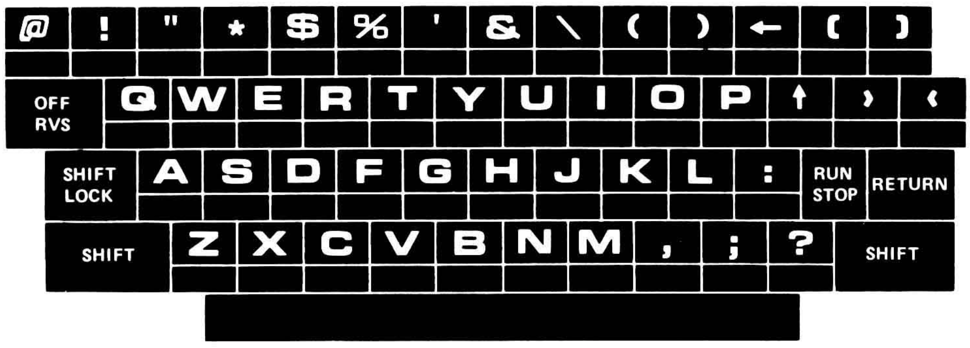
● **1** Switch on the PET. Note that it responds with

```
***COMMODORE BASIC***
```

and the number of bytes free.

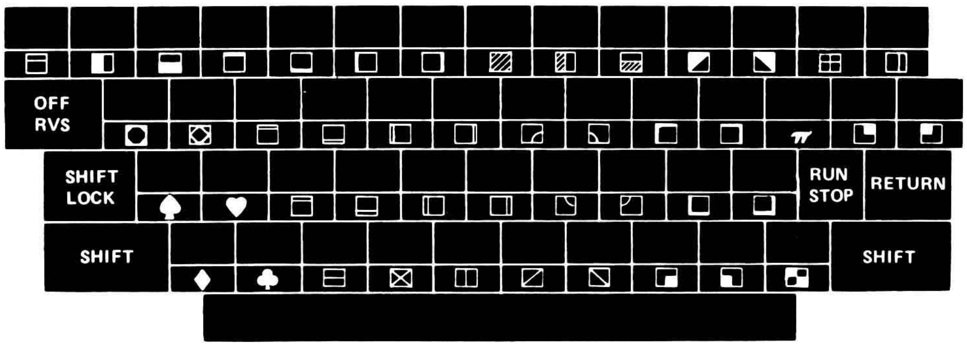
We are going to use the computer first of all in the direct mode as opposed to the programmed mode. The direct mode is also called the calculating or interpretive mode.

● **2** For a computer to function it must receive an input. On the PET, input from the screen is normally achieved by typing in your requirements (e.g. program) and pressing the RETURN key. This is the large key situated on the right-hand side of the alphabetic keypad (see Fig. 2.1).



@ ! " * $ % ' & \ ( ) ← [ ]
OFF RVS Q W E R T Y U I O P ↑ > <
SHIFT LOCK A S D F G H J K L : RUN STOP RETURN
SHIFT Z X C V B N M , ; ? SHIFT

(a)



(b)

Fig. 2.1 Alphabetic keypad on the PET: (a) without SHIFTing and (b) with the SHIFT key held down.

● **3** Unless the RETURN key is depressed the computer will not receive the data from the screen. In this book we shall use the symbol [RETURN] to denote that you should depress the RETURN key. If you have typed something, and the cursor (the white flashing square) remains on the same line, then you have not pressed the RETURN key. Alternatively, you may have tried typing in several lines at once and have omitted to press the RETURN key. Later on in the book we will assume that you understand when to press the RETURN key. If the book prints a series of lines such as those following, you must produce an image on the screen which is similar to the one in the book, i.e. you should press RETURN at the end of each line. (Do not attempt to input this program into your computer.)

```
100 INPUT A
110 INPUT B
120 PRINT A + B
```

● **4** Go back and read section (3) again and make sure that you have understood it. The principal mistake that beginners make is either never to press RETURN or to press it at the wrong time.

● **5** Calculations—the arithmetic symbols on the PET are:

+ for addition = for equals
− for subtraction * for multiplication and
/ (slash) for division; note the way this slopes because there are two obliques on the keyboard

All of these keys will be found on the numeric keypad (see Fig. 2.2).

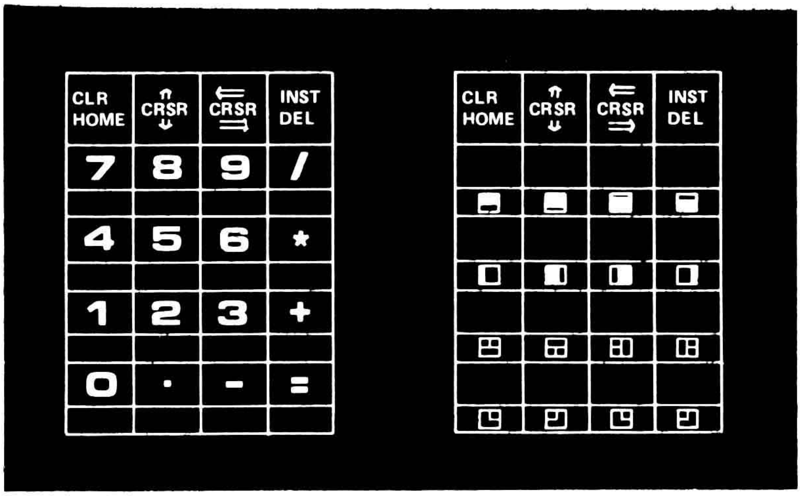


Fig. 2.2 Numeric keypad on the PET: (a) without SHIFTing and (b) with the SHIFT key held down.

● **6** Try typing

```
PRINT 123+456 [RETURN]
```

Note that the answer 579 appears as soon as you have pressed the RETURN key. The word PRINT is a command in BASIC. It tells the computer to PRINT the answer to the sum of 123 + 456. Now try

```
PRINT 456-123 RETURN
PRINT 2*4 RETURN
PRINT 4/2 RETURN
```

If you do type anything wrong then simply type it again, or, if you prefer, use the delete key DEL. This can be found on the numeric keypad.

● **7** Do the following sums and check that you get the correct answer:
(a) 7014 + 8202 + 9345 + 12072 = 36633
(Have you used PRINT before the sum? Do not use the equals sign at the end of the sum.)
(b) 15432 − 13650 + 11781 = 13563
(c) 65536/256 = 256
If you cannot do these, repeat section (6).
Note: you should have the PRINT command at the beginning and no equals sign at the end of each sum.

● **8** Use of numeric variables.
One of the powerful features of a computer is its ability to assign variables. In the PET there is an ability to give names to several tens of thousands of different variables. At the moment we are going to restrict ourselves to the letters A to Z.
Try this:

```
LET A = 123 RETURN
LET B = 456 RETURN
PRINT A + B RETURN
PRINT A, B, A + B RETURN
PRINT A; B; A - B RETURN
A = 789 RETURN
PRINT A + B RETURN
```

Note the following aspects of what we have done.
(a) In the first two lines we have defined the variable A as 123 and the variable B as 456. The use of the statement LET is not essential in PET BASIC, as we can see on the sixth line where A has been redefined as 789.
(b) Once a variable has a defined value then it will keep that value until we do something to change it. (We shall see later how we can set all variables to 0.) In this case the variable A retains the value 123 until at line six it is redefined as 789.
(c) The fourth and fifth lines are illustrations of two of the ways in which PRINT can be modified. The comma on line four causes ten spaces to be left between each set of numbers, while the semicolon on line five causes two spaces to be left between each set of numbers. (Note between each *number*. For non-numeric data no spaces are left if a semicolon is used.)
(d) The variables will remain stored in the machine; for example,

```
PRINT A/B RETURN
```

This returns A divided by B.

● **9** Use of non-numeric variables.
Non-numeric variables are termed strings. They are distinguished from numeric variables by the dollar sign ($), which on some keyboards is replaced by the pound sign.
Try this:

```
A$ = "HELLO" RETURN
```

Note the use of inverted commas (this is the second key from the left on the top row of the PET keyboard and is used unshifted, unlike most typewriters). The inverted commas are to distinguish the string from a BASIC statement or command.

```
B$ = "PET" [RETURN]
PRINT A$ + B$ [RETURN]
```

Note that there is no space. A space to the computer is every bit as much a character as any letter, therefore if we want a space we must define one. Try

```
C$ = "∇" [RETURN]
```

The symbol ∇ denotes a space and is one depression of the space key which is situated at the bottom of the PET in the middle (on the typewriter-style keyboards it is unlabelled). Thus C$ = "∇∇" would imply two spaces.

```
PRINT A$ + C$ + B$ [RETURN]
A$ = "WOTCHA" [RETURN]
PRINT A$ + C$ + B$ [RETURN]
```

● **10** Note that the only string arithmetic instruction you are allowed to use is the plus sign. This plus sign does something quite different in string arithmetic to number arithmetic.

The space key produces a character which is treated in the same way as the other characters. However, BASIC itself is structured so that it normally ignores spaces. For example

```
B$=∇∇"PET"
```

is treated the same as

```
B$="PET"   or   B$=∇∇∇"PET"   or   B$∇=∇"PET"
```

In the same way,

```
PRINT∇A$∇+∇C$∇+∇B$
```

is identical in BASIC to the expression

```
PRINT∇A$+C$+B$
```

which has no spaces. In fact you can leave as many or as few spaces as you wish. Spaces are normally put in to allow you to understand the expression more easily.

Once the space is enclosed in quotes, however, it becomes part of the expression, as we have seen already; thus

```
PRINT "HULLOPET" [RETURN]
```

is different to

```
PRINT "HULLO∇PET" [RETURN]
```

You must realise that the number of spaces between PRINT and the first inverted commas is not relevant.

● **11** The = (equals) sign. The equality sign on the computer has a different concept to that in normal arithmetic. It approximates most closely to 'is replaced by'. Thus in initially defining the variable A as being equal to 123, we have specified that the value of the variable A (which was initially zero) should be replaced by a value of 123. In a similar way we earlier replaced the value of the variable A, which was 123, by 789.

The right-hand variable always replaces the left-hand variable.

This concept is very important in understanding programming concepts. For example, later on we are going to use concepts such as N = N + 1.

Try this:

```
A = 123 RETURN
PRINT A RETURN
A = A + 5 RETURN
PRINT A RETURN
```

Note that A has been replaced by A + 5.

Part A

Simple programming