Contents

0.1 Acknowledgements

My sincere thanks are due to my many correspondents and friends who have given me ideas, answered my queries and made helpful comments which have all contributed to the preparation of this book. In particular, I would like to thank David Wilson, who has made numerous suggestions (some of which are printable) and assisted with the final editing. I am also very grateful to Gerry Austin of PCW-World, who has managed to publish this book within two months of our initial meeting – incredible!

Geoffrey Childs, October 1989.

0.2 Part ONE

You may be a very good programmer, you may just think you are, or you may realise that you are only a beginner. You may have written thousands of programs... or just the one. It does not matter in the least. The less you know, the more you can learn from this book.

Will this book help you to write spectacular programs with mind-boggling graphics? Or will it just help you to write a program for your weekly budgeting that works efficiently? The answer lies solely in your hands. This book will give you the means to produce programs with all the bells and whistles. It will also help you to do the much simpler programming tasks well. What it will NOT do is to teach you how to PRINT "Hello". Nor will it give the syntax for every Mallard BASIC word. You need an elementary tutorial for the former, and a manual for the latter.

Part 1 is about the things that you CAN officially do with Mallard BASIC. It is hoped that the discussions will improve the proficiency of your programming, so that the simple things are done effectively. Part 2 is about the things you CANNOT do with Mallard BASIC. As we shall see, there is no such word as CAN'T! Admittedly, we shall have to make some recourse to machine code in Part 2: but have no fear, you will not be asked to understand it. Any BASIC programmer can use code routines that have already been written.

The final section is a series of appendices. This includes a simple extension to Mallard BASIC and a few other programs that do not conveniently fit into the text. There are plenty of short programs and routines that are included and explained in the context of the text. Each of these will be labelled with a name such as C4P2 (short for Chapter 4 Program 2). These can be found on the program disc supplied with this book.

In a book about computers, it is impossible to avoid 'jargon' completely. I hope that most of it will be explained as we progress. It may be helpful just to explain a few words immediately. MEMORY can be thought of as the brain cells of the computer. Each cell of memory (there are over 500,000 of them in the Amstrad PCW8512) contains a number 0-255 that the computer 'remembers' until it is changed, or until the computer is switched off. CODE is an all- purpose word that means symbols, letters, words, and language that the computer can operate upon (given a little bit of help by what is already in its memory). The PROCESSOR is the electronic device that turns the numbers in memory into simple activities that the computer can carry out. MACHINE CODE is the code that the processor can understand. A programming LANGUAGE converts the code that a programmer writes into code that the computer can use. Mallard BASIC is called an INTERPRETER as it does that conversion, line by line, each time a program runs. Other languages use a COMPILER which turns a complete program (SOURCE) into a new program (OBJECT). The object program is in machine code. We shall not be much concerned with compilers. RESERVED words are words that have a special meaning to the computer in languages such as BASIC. PRINT, RANDOMIZE, and SQR are examples of reserved words in BASIC. DO is a reserved word in the Pascal language, but means nothing in BASIC.

If you did not receive a program disc when you bought the book, please let us know at PCW-World ('Cotswold House', Cradley Heath, Warley, West Midlands B64 7NF) and we will provide a disc free of charge on receipt of some evidence of the sale.

Chapter 1

BASIC matters

Just before the second world war, LNER engine number 4468 achieved an all-time speed record for a steam engine of 126 mph (203 km/h) between Grantham and Peterborough. It could not have been done without 'streamlining' or a competent driver and fireman. Number 4468 was named "Mallard", as is the BASIC on your Amstrad PCW. Mallard BASIC has also been designed to be very fast and, when you program with it, you are its driver and fireman. The 'streamlining', I hope, will be found in the following pages.

1.1 Purpose

This book is not intended to be a user manual for Mallard BASIC, or even a tutorial. It is intended for Amstrad PCW users who have experimented with Mallard BASIC and attempted to write some serious, even if simple, programs of their own, but would like to move on to bigger and better ones. If you have not reached this stage, I suggest you study some tutorial material first. Your first choice will, obviously, be the official manual, and a book by Ian Sinclair has been generally recommended. For those preferring to learn at the keyboard, 'David Wilson Computronics' have produced a comprehensive disc entitled "BASIC Tutorial".

As I develop my approach, I will not be afraid to PEEK and POKE about in memory from time to time, but I will not assume that you have any knowledge of machine code programming – or even that you want to descend to that level! Very occasionally, a machine code sequence will be included in an example routine, but you will not need to understand it.

My central theme will be programming style, though some might think that I am the last person on Earth who ought to be writing about that (ROCHE¿ Including me, since Geoffrey Childs *NEVER* indents his programs!). If someone wanted to compliment my programming, I would accept – with typical modesty – such words as "knowledgeable", "ingenious", "economical", and even "eccentric". But not even my best friends – nor, for that matter, my worst enemies –

would call me a stylish programmer. However, what we are considering is your style, not mine. There are certain aspects of programming that I consider to be bad programming rather than bad style and, in these cases, I shall be quite dictatorial, telling you a few things that you must not do. Apart from that, I hope that you will accept most of what follows as suggestions, rather than instructions. Very often, the question is one of speed of operation, or the comfort of the user, which is what good footplate crew have always been concerned about.

Any form of programming is a skill and, however individual a skill may be, the best practitioners will try to learn from others. If you think that you always program perfectly, who am I to disagree? You may find some of my ideas and methods reprehensible – again, I will not disagree. What I can claim is experience – the fact that I wrote "Lightning BASIC" proves that – and that most of my programs do work, eventually.

1.2 BASIC extensions

Mallard is a powerful and fast implementation of BASIC. It is fair to say that the more I have used it, the more I like it. I did find it rather strange at first, after using Microsoft and Xtal BASICs on other machines. However, any new system needs patience before it becomes familiar.

Having said this, there is no question that Mallard is extremely clumsy at doing a few simple things, the most obvious of which is clearing the screen! (ROCHE¿ BASIC, and CP/M, were first used on computers which had no screen, only an ASR-33 Teletype through which all interaction with BASIC/CP/M was done. So, no screen = no CLS. Anyway, who needs a CLS when pressing [RETURN] the appropriate number of lines is enough to clear the screen?) There is a method behind Mallard's madness. It is not that Locomotive were incapable of devising a CLS which would have done the job. By using control codes, Mallard files become portable when saved in ASCII form, and can be used on computers other than the Amstrad PCW. (ROCHE¿ I have been using Mallard-86 BASIC for MS- DOS for 10 years, now, and have *NEVER* used ASC files, only BAS files when transfering my old Amstrad PCW programs...)

Your masterpiece (and mine) are probably only intended to run on the Amstrad PCW. (ROCHE¿ I started as a COBOL programmer on IBM Mainframes. Then ventured into minicomputers. Then finally used micro-computers (which went from 8 bits to 16 bits, then 32 bits!). For me, Mallard BASIC is a high-level way of porting my programs: they run under CP/M Plus, MS-DOS, and in a "DOS Box" under Windows.) It makes sense to make a few extensions to Mallard BASIC, so that the simple jobs can be done quickly.

Much of my programming on the Amstrad PCW has been devoted to writing various extensions to Mallard BASIC, so that not only does it do the simple jobs easily, but becomes more powerful generally.

I will admit that I do most of my BASIC programming using "Lightning BASIC" but, then, having written it myself, I have not had to pay for it! You would not like it if, having raided your piggybank to buy this book, I told you to send 24.95 Pounds to CP Software to buy Lightning BASIC. (But, of course, it is worth every penny...)

DWBAS is a much shorter extension (also written by me) – this can be RUN and LISTed from the program disc. You may have this for nothing. I do not like typing in programs with lots of figures, that is why I have provided DWBAS (and the other programs in this book) on disc. If you have the Amstrad PCW9512, use DW139 instead, this is also on the program disc. This is an amended version that runs with Mallard-80 BASIC Version 1.39. (The Amstrad PCW8256 and PCW8512 use Mallard-80 BASIC Version 1.29.) The explanation of these extensions is given in Appendix 3. The programs in the text do not rely on DWBAS, but those in Appendices 6 and 7 do need DWBAS loaded first.

One important feature of DWBAS (and Lightning BASIC) is a multiple POKE. The command POKE 50000,2,3,4 will POKE 2 into 50000, 3 into 50001, and 4 into 50002. I find this so important that I shall INSIST that you have a multiple POKE when we come to Part 2 of the book. If you refuse to use DWBAS, Appendix 4 gives a briefer program which will install this feature. Naturally, you will not need this if you use DWBAS.

1.3 Your own style

Part 1 of this book is intended to help you develop your own style of programming. There are usually many ways of achieving the desired result in a program. Often, the programmer has to make a choice between economies in memory and speed on the one hand, and attractive presentation and easily readable programs on the other. Programming style generally develops subconciously, but it is not a bad idea to have an introspective examination of one's programs from time to time. In this section, we have a preliminary look at some of the questions that you might consider.

Writing computer programs has parallels with writing English. It is easy to identify bad style, but good style is not so clear cut or uncontroversial. In most of our literary works, we would try to avoid spelling sausages as 'sosiges', but it would not matter if we put 'sosiges' on our weekly shopping list! In an idle quarter of an hour, last week, I used a computer program to work out the Daily Telegraph Brain Teaser. It does not matter in the slightest how badly the program was written, provided it works!

Another parallel with English is that the style of a computer program depends on the intended user. You would not write a love letter in the same style as a technical report, and it is unlikely that you would write a "Space Invaders" game in the same style as an accounts system.

Clearly, there are some general rules that all programmers should follow, and they can be summarised in one sentence. Write your program in a style that will be suitable for the people you expect to use it. The balance between lengthy prompts, excessive input checking and confirmation on the one hand, and a fast flowing program on the other, must be made with the expected experience of the average user in mind. Decorative screens and musical jingles (not so easy on the Amstrad PCW) must be balanced between pleasure and irritation potential! It is sometimes hard to remember that the person using your program, probably a philistine, will not be admiring your beautiful programming skills, but using your program as a means to an end.

Having made these general didactic points, it is time to emphasise the main point I want to make in this section: DEVELOP YOUR OWN STYLE.

What is right for you, may not be right for me, and vice versa. If you find that you can write programs that work well without bothering about structure and subroutines, then write that way – at least until you find that it might be wiser to modify your views. If you find that planning on paper and having a complete knowledge of exactly what your program will do is essential – do it that way. Academic writers about computer programming talk some sense, but I think that they talk a lot of rubbish, too!

I am going to finish this section by asking a number of questions. They are question which you may feel strongly about, and your answers may be opposite to mine. You may feel that the answer is sometimes 'yes' and sometimes 'no'. I hope that some will be questions that you have not even considered. In a short section like this, it is impossible to cover all the questions one might ask about style. Provided you are thinking about these propositions, it matters not whether you become more pig-headed in your views, or more willing to modify them.

Should variable names be long: "age.of.employee" or short: "a"?

Do you religiously, occasionally, or never put in REMs?

Do you leave blank lines, such as "200 :", to make your listings look pretty?

Should documentation be non-existent, in REMs, on-screen, or in a separate file or booklet?

Do you use "Press any key..." or a specific "Press SPACE..."?

Do you always use INPUT, even for one-letter replies?

Do you believe in right-justifying or centring screen printing?

Do you make sure the user sees your name on any programs you write?

Do you have any little habits by which a program you wrote would be identified as likely to be yours?

Should you avoid POKEs or CALLs that make your programs machine or model specific?

Do you use reverse video often, occasionally, or never?

Do you write your subroutines before the rest of your program, or whenever they crop up?

Do you think that you make too little or too much use of subroutines?

Do you like or dislike programs where the menu choice is made by cursor control (or a mouse), rather than by pressing a key?

Is it better to keep most program lines to a single statement?

What are your feelings about GOTO? That it should be never used. That is has occasional uses. That you do not care what the academics think, and it is just about your favorite BASIC word!

Do you use integer variables whenever possible, occasionally, or never?

Do you RENUM when a program is complete, or do you leave numbers as they stand, so that you may be able to recall them better later?

Do you ever use DEF FN, apart from escape sequences? Or PEEK, POKE, CALL, and VARPTR?

Do you use ON ERROR throughout a program, for occasional bits where a specific error could be expected, or never?

Do you protect your programs? If not, do you make your listings as easy (or as difficult!) as possible for the user to follow?

Which type of files do you use most often: sequential, random, or Jetsam?

Do you think Jetsam is the best thing about Mallard BASIC, could be used on occasions, or is too complex to try to understand?

What is your attitude to a finished program that works? "I am not going to touch it again", or "I will take every opportunity to gild the lily"?

Do you consider it bad programming if the screen scrolls during operation?

Some of the questions will not be discussed any further, but there are other questions on which I do have definite views, and which we shall think about in more detail. Where this happens, I shall be happy to preach to heathens, doubters, or the converted!

1.4 Planning

If you are a competent pianist, it is possible to sit down at a piano, put your hands on the keys, have no idea what you are going to play, and improvise happily and tunefully. You cannot do this with a computer. All programs do need an element of planning. At the very minimum, you must know what the program aims to achieve.

Flowcharts were once one of my pet hates. Perhaps I do not read the right books these days, but thankfully we do not hear so much about them, now. The idea of a flowchart is to put into words the various stages of a program, and use symbols and arrows to define its possible courses. For a simple program, this seems a waste of effort and, for a complex program, it is difficult to imagine the size of paper that you would need! I would find drawing the flowchart much harder than writing the program – which is not the purpose of the exercise! I know some people do find them helpful. If you like them, do not let me stop you using them. You are probably more methodical than I.

Some methods of planning are needed. If I am writing a Mallard BASIC program, I usually code at the computer. Occasionally, I may use the back of an envelope for a section of code that looks tricky. I may write a very rough menu outline for a long program on paper. If I am writing a machine code routine, I usually write it in assembler on an envelope, and translate it into code at the machine. Not recommended, unless you also are in an advanced state of madness! The stationers do not make a fortune out of me. I appreciate that most people would disagre with so little paper planning. Ask yourself if more time spent on paper planning would make the eventual writing easier or more efficient. Only you can decide.

You may have kindly thoughts like: "Yes, but you are such a genius that you do not need to plan." Or unkind ones like: "No wonder your programs turn out so badly." Wrong on both counts. I do plan: I probably spend more time on it that you do. If an idea for a substantial program comes to mind, I do not sit down at the computer to write it. I carry the idea, maybe for months, and think about it. I do nothing at the computer until I know that I have sufficient time to write the program with a minimum of breaks. By the time I start, I have a clear outline plan in my mind. While I am writing the program, I will be thinking about it and the little extras that can be introduced to improve on the outline. When the program is written, I hope to have nothing more to do. Usually, this is optimistic because it is often someone else who finds the bugs.

'Structured programming' is a computer buzz-word. To some people, it is closer to a religious utterance. A structured program is contained in modules. On a computer such as the BBC, these modules are called PROCEDURES. On the Amstrad PCWs, we have to make do with subroutines, which are nearly the same thing! The procedures are written, and all the main program does is to call them in the correct sequence. My advice is not to make structured programming into a religion, but make good use of subroutines, so that your programs have a semblance of structure – even if served with a helping of spaghetti!

If I use any formal technique, it is probably closer to the 'top-down' method than anything else. I usually a program at line 5000, where I put the general purpose subroutines – some will have been pinched from other programs. I may write some more specific subroutines, such as filing ones, starting at line 6000, which may only need to be chained to certain options on the menu. I use 7000+ for error traps. The main program is divided into sections, starting at line 1000, 2000, etc. I write the program a section at a time, when the subroutines have been written. Sometimes, I also use lines above 8000. Eventually, the main menu and housekeeping goes at the beginning of the program.

There is no magic in this particular numbering system, but there is plenty of point in establishing a pattern for yourself and sticking to it, if only for the fact that anything done consistently is more easily remembered.

However well you plan, you will probably spend at least as much time on testing and debugging your program as on writing it, but we will leave these little treasures until later.

Chapter 2

Everyday problems

In this chapter, we are going to look at some Mallard BASIC commands that most of you will have used. Familiarity can breed contempt, and good or bad programming is often more evident in the simple everyday operations that most programmers will be using frequently. The topics may seem disjointed, but the common theme is that we should think about the routines that we know we can program, as well as those that we know will offer a new challenge.

2.1 Escape sequences

One of the first things most of use notice about Mallard BASIC is that you cannot clear the screen with a simple CLS (or something similar). I have come to respect Mallard in many ways, but I still think that this is ridiculous, even though the reason is that Mallard BASIC is meant to be portable.

If you are using DWBAS, you will be happy not having to use escape sequences like

$$PRINTCHR\$(27) + "E" + CHR\$(27) + "H"$$

any more.

This will be the case if you only want to program for yourself. Some of you will wish to write programs for magazines, for example, and you cannot rely on readers having DWBAS. Life will be simpler with DWBAS, but there will still be times when you have to use the conventional sequences.

So, let me get rid of a couple of bees in my bonnet. I hate the people who write

$$esc\$ = CHR\$(27)$$

or even worse:

$$escape\$ = CHR\$(27)$$

The idea is, presumably, to shorten things, so WHY NOT

$$e\$ = CHR\$(27)?$$

It also irritates me greatly to see the PRINT AT sequence used without a DEF FN, and the horrible CHR$(32+r) appearing all over a program. If you must use escape sequences, my advice is to write a little subprogram including all the regular ones, and start each new program with it. Unused escape sequences can be deleted when the program is completed. (End of burst of bad temper!)

It is worth studying pages 140-141 (in my copy) of Manual 1 to note the rarer escape sequences. Some knowledge of the ones to use with the printer is also advisable. The effect of escape sequences is, of course, quite different in PRINT statements from that in LPRINT statements.

2.2 PRINT and LPRINT

Many programs will contain sections that offer the user a choice of printing on the screen, or sending the same text to the printer for hard copy. Quite often, these program sections are fairly long, and it seems to be wasted effort when the whole thing has to be written out again with all the PRINTs changed to LPRINTs. Of course, it is possible to save some of the burden by suitable use of AUTO and RENUM, but there is a much easier way.

The idea is very simple. We temporarily send all PRINT commands to the address for LPRINT commands. When we have done the LPRINTing, we change the address back to normal. POKE 18527,90 changes PRINT to LPRINT. POKE 18527,100 reverts to normal. On the Amstrad PCW9512, a different version of Mallard-80 BASIC (Version 1.39, as opposed to Version 1.29) is provided, and the addresses are different. The corresponding POKEs are POKE 18591,0 and POKE 18591,10. If you are writing for users who might be in either version of Mallard BASIC, you will have to test. The address I use (out of many possibles) is 5103. If PEEK (5103) = 197, then you are using Mallard-80 BASIC Version 1.39.

There is also a simple POKE to make the PRINTing go to the screen and then to the printer. POKE 8792,205 in Version 1.29 and POKE 29161,205 in Ver. 1.39. This make a machine code jump into a call (like a GOSUB): when it does the PRINT routine it returns, and where has it got to? The LPRINT routine! Making this change is rather more comprehensive than the previous POKE. Not only does PRINT get echoed, but also everything else that goes to the screen, including the "Ok" prompt! To go back to normal, you POKE 8792,195 with Ver. 1.29, and POKE 29161,195 with Ver. 1.39.

An omission in Mallard is that there is no official way to TYPE or DISPLAY a file to hard copy. Byt try this in Mallard 1.29:

POKE 8793, 234 : DISPLAY "FILENAME.TYP" : POKE 8793, 239

Change 8793 to 29162 with Ver. 1.39. If you run an Amstrad PCW9512, you will probably be using a version of CP/M Plus named 'J21CPM3.EMS' (found on your Amstrad master disc). In this case, you should replace the 234 and 239 mentioned above with 246 and 251, respectively.

List of Tables

List of Figures