AMSTRAD AMSTRAD AMSTRAD AMSTRAD

Computing with the AMSTRAD

INC. AMTIX!

$4.50

The independent magazine for Amstrad computer users

Registered by Australia Post – Publication No. VBP7811

AUGUST, 1987

A Database Publication



- **CENTIPODS — GAME OF THE MONTH**
- **MAKING MIDI MUSIC – DIGITAL DO, RAY, ME**
- **NEWS • GRAPHICS • WORD PROCESSING**
- **PULL-EM-DOWN MENUS • REVIEWS**

$0 $AVINGS ON $OFTWARE

AMSTRAD – AMSTRAD –

Contents

August 1987

4 FEATURE
IAN SHARPE investigates a useful addition to Locomotive Basic

6 REVIEW
IAN SHARPE reviews Arnor's CP/M Plus implementation

9 FIRST STEPS
Part 12 of PETE BABBY'S series on basic programming for beginners

13 CP/M
COLIN FOSTER'S exploration of CP/M 2.2

17 GRAPHICS
A regular feature of Amstrad graphics with GEOFF TURNER and MICHAEL NOELS

23 UK NEWS
What's going on in the world of Amstrad

26 10 LINERS
When the features editor gets 'creative'

27 AMTIX
News and reviews of new products released for Amstrad

35 ORDER FORMS

The Business Section will now appear bi-monthly.

36 AMTIPS
Tips and ideas on where to go when you're stuck

43 UTILITIES
Pull-em-down menus

46 GAME OF THE MONTH
After decades of being blown out of the skies along comes Centipods

52 MACHINE CODE
In Part VII of his series MIKE BIBBY looks at register pairs

56 SOFTWARE REVIEW
BrunWord — a word processing workhorse that'll last for years — reviewed by Jo Stork

58 SOUND
Let's Make Midi Music —Sample digital DO, RAY, ME, with IAN WAUGH.

63 IF ALL ELSE FAILS...
READ THE INSTRUCTIONS!! A whacky whimsical world — that's the world of Amstrad

65 PUBLIC DOMAIN
Extending your computer's memory under CP/M.

66 OUR SAY
The Editor has the last say

Published monthly by Planet Publications Pty Ltd under licence from Database Publications Ltd. Subscriptions – see centre page order form.

General enquiries, phone (002) 29 4377.
Telex: AA 58134, Attn. HT163
Mail: P.O. Box 11, Blackmans Bay, Tasmania 7152.

Annual subsciption (12 issues)
Australia $45
South Pacific (inc. NZ) $A65
Advertising enquiries phone (002) 31 0130.

Extra commands speed production of DIY databases

Ian Sharpe reviews a useful adjunct to Locomotive Basic

Instant Access is designed to add random access disk filing and other useful facilities to Locomotive Basic. It takes up about 10k of memory and provides 36 new RSX commands which fall into five main groups. These are summarised in Table I.

Amsdos — the Amstrad Disk Operating System — behaves in a similar way to the cassette system. It maintains compatibility between disk and tape based machines but fails to exploit the full potential of disk filing. The main problem is that Amsdos only supports sequential files.

Let's assume you had written a database which stored all your information on disk and you wanted to read a record part-way through. The only way to do it under Amsdos is to open the file and read all the records sequentially until you reach the one you want, just as if you were using cassette. It's like trying to look up Sharpe in a telephone directory by starting at the beginning and reading every name.

This is a slow process and you would probably only write a database in Locomotive Basic if the data were small enough to fit in memory. You would then read all the data into ram, manipulate it there and write all it back to disc at the end of a session.

This problem doesn't occur under CP/M, which lets you go straight to any part of a file and access it without having to load unwanted records. You can't alter the record as quickly as if it had been in memory, but the size of your database is only restricted by the amount of information you can get on a disk.

This is one of the strengths of

Random access

CREATE	Creates random access file.
OPEN	Opens random access file.
CLOSE	Closes current file.
INPUT	Inputs from file.
GFILE	Gets status about current file.
GPTR	Gets current random access pointer.
GFSIZE	Gets size of current file.
OFFSET	Sets origin for random access pointer.
PRINT	Sends string to file.
PTR	Sets absolute position of random access pointer.
PTRR	Sets relative position.

Sector editing

RDSEC	Reads specified sector into buffer.
SPEEK	Peeks byte in the buffer.
SPOKE	Pokes byte in the buffer.
WRSEC	Writes sector to disc.

Basic enhancement

EDIT	Edits string.
EPAR	Sets editor parameters.
EXEC	Executes specified Basic commands.
GKEY	Gets editor exit character.
GPOS	Gets last editor cursor position.
GVER	Gets version of Basic.
SPOS	Sets cursor start position in editor.

General disc

DDRIVE	Sets default drive.
DUMP	Dumps file to screen.
DUSER	Sets default user group.
FEXIST	Checks existence of file.
FORMAT	Formats specified track.
LOAD	Loads binary file without buffer.
SAVE	Saves binary file without buffer.

Error handling

ERROFF	Disables BIOS error messages.
EPRON	Enables BIOS error messages.
GERR	Gets error/line number for last error.
HELP	Displays available commands.
ONERR	Defines action on utility error.
REPORT	Displays last error message.
RETRY	Sets number of retrys disc makes for failed operation before generating error.

Table I: Summary of Commands

Mallard Basic supplied with the PCWs and available for the CPCs — at a price.

Random access is catered for by 11 RSXs which let you create, open and close random access files, input and output strings, move the file pointer to any location — either relative or absolute, obtain the pointer value and information about the current file.

The file pointer, as you may guess from the name, is a variable that holds the position in the random access file where the next piece of data will be written to or read from.

Included under the heading of general disk commands are those to set the default drive or user group, format particular tracks on a disk, load and save binary files without the need for a buffer, check for the existence of a file and dump a file to the screen.

The format option will let you specify the sector numbers to be used in formatting. These are used by Amsdos and CP/M to decide (among other things what type of format a disk is).

If they come across numbers that don't fit in with one of the standard types such as data format, they will stop with an error message.

The use of non-standard numbering on empty tracks of an otherwise normally formatted disk will prevent copying with Diskit but not one of the more sophisticated copiers.

Sector editing commands allow you to write your own disk utilities in Basic. A sector can be loaded into a buffer, individual bytes peeked and poked and the amended sector written back to disk. These could form the basis of a directory editor or program to recover files from faulty disks.

Seven commands are devoted to providing a greatly enhanced line editor which repalces INPUT and LINE INPUT from the keyboard. A string is fille d with characters to the same length as the maximum number you want to allow in the input field and :EDIT,(*a*string,x,y will present it for editing at the specified position.

Spaces are shown as graphics characters which indicate the extent of allowable cursor movement. Control+R toggles between overtype and insert modes (why not Basic's Control+Tab?), the Clr and Del keys behave as expected and Control+W wipes the line.

The input filed wraps round at the right hand edge of the window, so if a window is defined to contain the same number of characters as the input string you can edit a box rather than a single line.

For applications where it's desirable to move the cursor from field to field the character codes that exit the editor — maybe the ones generated by Control+cursor keys — can be specified. Another RSX returns the character that caused the editor to terminate, so it is easy to program these effects.

The final group concentrates on error handling, mainly relevant to those generated by the utilities.

A telelphone directory program is included on the disk which serves as a good example of how to write a simple database using this package.

The 31-page manual details each command and should not present any problems to an average Basic programmer. It also provides a line by line explanation of how the example program works.

Instant Access is a genuinely useful package, ideally suited to writing data-bases and disk utilites in Basic. If you are on the look out for a product to provide an alternative means of handling files I can certainly recommend this one.

UTILITY – 'Help' is on the way

The Help utility which was printed in the July edition can be found on the August tape (quarterly disc no. 4).

On the July tape was a Utility called Procedures (together with the game 'Mastermind'). This will be published in the September issue of CWTA.

MAXAM II — now the best gets even better

IAN SHARPE reviews Arnor's CP/M Plus

Early in 1985 Arnor introduced Maxam, an assembler, text editor and monitor for the CPC which was available as a plut-in rom cartridge as well as the normal tape and disk formats.

Not only was it the first software to appear on rom but it quickly won a reputation as the best assembler available, only rivalled in power by Hisoft's Devpac but certainly not matched in general ease of use.

Since then other assemblers have appeared, some with features that are noticeably lacking in Maxam. Despite that, I've stuck with Arnor's assembler because it's convenient having it in rom and so quick and easy to use.

For some time there have been rumours of a Maxam 11 that would use Protext as its text editor, leaving room for macros and enhanced debugging facilities. The rumours were nearly right, but Maxam 11 but isn't intended as a replacement for the original. It's a separate product for the CPC6128 running CP/M Plus and PCWs.

Like the original, Maxam II comes in three main parts — text editor, assembler and monitor. Each has been expanded and is at least as big as the original 16k package. There are also disk utilities and extensive help files.

The assembler, text editor and monitor are integrated so although they are individual programs it is possible to go from one to another without exiting to CP/M.

You can pass filenames, and invoking the assembler or monitor from the text editor will let you return to the editor with your text file already loaded. This makes the edit/assemble/debug/ edit cycle as quick as is practical under CP/M.

The text editor shown in Figure 1 is an implementation of Protext's program mode. I've used word processors and text editors on various machines and haven't seen anything that is both as powerful and as friendly as Protext. It's a delight to use and is excellent for entering text quickly.

It can work with two files in memory and it's possible to copy text from one to the other. Files larger than available memory can be handled, though I prefer to split a file into sections to avoid the reduction in speed caused by disk accesses.

Blocks of text can be loaded, saved, moved, copied and deleted. One thing I particularly like is the undelete function which allows recovery from those awkward situations which tend to arise when you haven't made a back up for a few hours.

There's also a facility to add or remove line numbers if you want to edit a Basic program save has Ascii.

The assembler takes its source code from disk or a file

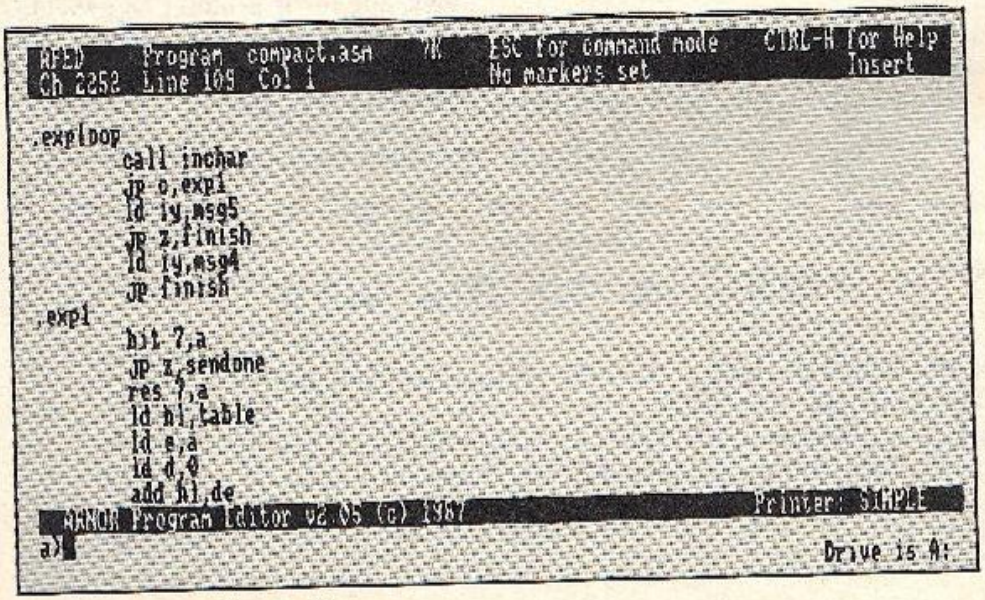


Figure I: APED text editor

in memory. Object code is written back to disk, the default being a .com file. Arnor intend to supply a utility to convert files to a format readable by Amsdos though the monitor won't be any use for non-CP/M programs.

A number of commands can be embedded in the source code which instruct the assembler to perform certain actions. These are known as pseudo-op codes or directives. All assemblers have these to some extent — ORG and DEFB being two examples.

Maxam II's extensive list of directives is almost a mini language. Conditional assembly and REPEAT...UNTIL loops let you produce different versions of a program from a single source file, and others give full control over the listing output.

Large files may be developed in sections and either assembled separately and linked together or the assembler can read multiple source files with the READ directive.

Macros are a welome addition because they can shorten the time it takes to write a program and make your code a lot more legible.

A macro is a sequence of instructions referred to by name. For example, if you wanted more flexibility in adding 16 bit register pairs you could define a macro to add the contents of BC to DE and leave the result in BC.

```
MACRO add_bcde
push h l
ldl,c
ldh,b
add hl,de
ldc,l
ldb,h
pop hl
MEND
```

With this macro defined at the start of your program *add_bcbe* can be used just like another mnemonic. Whenever the assembler meets it the list of instructions corresponding to the definition is inserted.

It's also possible to put your macros in a separate file so you can build up a library of useful routines. Another feature of macros is that the actual sequence of intructions can be varied according to a set of paramenters.

For instance, to add together any pair of eight bit registers leaving the result in the first register a general purpose macro could be written like this:

```
MACRO add_r1r2 &reg1 &reg2
ld a,&reg1
add &reg2
ld &reg1,a
MEND
```

So in one place you might use:

add r1r2 b c

which would add B to C and leave the result in B. Somewhere else in the program you could use:

add_r1r2 l d

which would do the smed of the L and D registers using the same macro definition. Parameters are only used during assembly and shouldn't be confused with those passed to functions and procedures in high level languages.

Another powerful feature is the ability to link machine code with files prosuced by Arnot C. The problem with putting in-line machine code in a C program is that C does not incorporate an assembler. You can write the code with an assembler and either incorporate the hex values in your C function or leave an area of memory free and load the code as a separate binary file. Alternatively, you can give all the opcodes difinitions such as:

#define ld_a Øx3E

and write pseudo-assembler using the labels.

None of these methods are very satisfactory, particularly when it comes to writing and debugging more than a few lines of code. The Arnor method is to use Maxam to write machine code and store it on disk as a link file.

In the file will also be a list of labels that you have declared with the assembler directive PUBLIC, along with their offset from the start of the code. The linker supplied with Arnor C can then access the labels and incorporate your machine code in a C program.

There are one or two constraints when writing assembly language for use with C. It has to be relocateable and can't be longer than 32k, though you can get round this by linking multiple files.

Arnor poducts seem to score again and again when it comes to combining powerful facilities with ease of use and they've excelled themselves with the monitor. The old monitor was rudimentary but the new one is in a different league.

In fact two monitors are supplied, small and large. The small monitor lacks the less commmonly used features of the large model to make room for larger object files which obviously have to be in memory at the same time.

Like the text editor, there is a separate command mode and

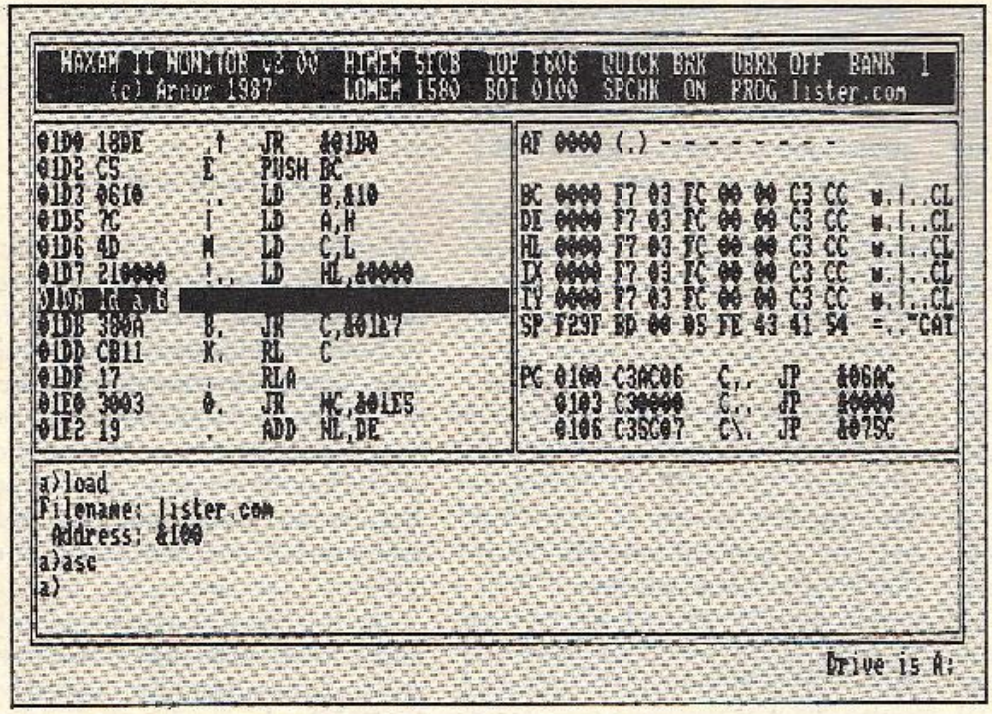


Figure II: Edit memory in hex or assembler mnemonics.

dozens of commands too numerous to list here.

Several debugging tools are available such as conditional breakpoints and single stepping. These make it possible to execute a program slowly or quickly with the monitor retaining control so you can stop the program with a key press or at a breakpoint.

This may be a particular address in the program or when a predefined condition arises, such as a register reaching or exceeding a certain value. You can examine and alter registers of memory and then continue execution.

There's a disassembler which can output to screen, printer or disk and a single pass assembler which lets you edit memory by typing in assembler mnemonics rather than hex codes. Figure II shows you the idea.

CP/M plus operates with banked memory allowing it to use more than the normal 64k available with the Z80 microprocessor. The monitor will let you investigate all the memory map by switching in banks of ram as required, and that includes the PCW's ram disk.

To round things off, several utilities can be accessed from the editor or monitor, including a disk copier and formatters. The latter will format a disk for either drive A or B on the PSWs and in CPC data format on any mode.

Files can be erased, copied or set to read only or read and write status. You can view the contents of a file with the TYPE command without erasing the file in memory and you can spool all screen output to disk or printer.

There's even a command to redefine a character matrix and you can construct exec files which contain commands that are exectued as if they were being typed in at the keyboard. In fact the automated installation procedure is done with an exec file.

Maxam II is accompained by a clearly written 157 page manual. It contains a tutorial session based on a worked example, which involved writing and debugging a short program, and is a great help in becoming familiar with the package.

Maxam II is going to appeal to professional programmers and very serious amateurs. Cheerful it is, but cheap it is not.

The thing that struck me when reviewing it was the amount of thought and effort that has gone into this product. Arnor have developed what was already considered to be the best available and deserves every success.

Taking arrays into another DIMension

Part 12 of Pete Bibby's series on Basic programming for beginners

This month we'll start with a simple program that should present no problems. All Program 1 does is to use a FOR ... NEXT loop to accept three numbers. Each is stored in turn in the aptly-named numeric variable *number*. A running total of the numbers is kept in *total*.

Although it's a simple program that's hardly likely to push back the frontiers of programming on the Amstrad when you look at it more closely it does raise some interesting problems.

```
10 REM Program I
20  total=0
30 FOR loop=1 TO 3
40 INPUT "Give me a number ",number
50 total=total+number
60 NEXT loop
70 PRINT "The total is"total
```

Program I

Notice how the same variable is used to store the three values you've typed in. If you enter 3, then 4 then 5 in response to the prompt, the value of *number* reflects this, becoming 3, then 4 and finally 5.

In this program that's all well and good, but what if it were a part of a more complex one? Suppose after finding the total, I wanted to do something else with the input numbers such as sorting them into order.

With the input method used in Program I, I'll be in trouble. I might go to look for the three numbers but when I get there the cupboard will be bare, or nearly so. The last figure typed in will still be stored in *number*, but the first two will have disappeared, replaced as *number* takes new values. Program II is an attempt to solve the problem.

```
10 REM Program II
20 INPUT "First? ",first
30 INPUT  "Second? ",second
40 INPUT  "Third?  ",third
50  total=first+second+third
60 PRINT  "The total is"total
```

Program II

Now the numbers are stored in first, second and third and summed up in line 50. The program works: three numbers ar e indeed entered and their usm is calculated. Also the figures are all still available for use, as a quick:

PRINT first,second, third

will show. The trouble is that although the program works, it's pretty unsatisfactory. We need a better solution, as the next problem shows.

Suppose we had to add 50 numbers. We could modify Program I to do the job with very little trouble. All we do is to change line 30 to:

30 FOR loop=1 TO 50

and we get the total.

With Program II, things are more difficult. As it has to have the values stored in different variables, it can't use a FOR ... NEXT loop. So to adapt it to summing 50 numbers you'd have to have 50 variables, going from first, second, third, all the way to forty-ninth and fiftieth. And line 50 would be massive, something like:

50 total=first+second+third+ ...+fortyninth+fiftieth

It's not too practical is it? What we need is a method of INPUT-ing numbers into a variable that will give us the efficiency of the first program's FOR ... NEXT loop but also let us keep a record of the numbers typed in. Program III points in the right direction.

```
10 REM Program III
20 INPUT  "First? ",mark1
30 INPUT  "Second? ",mark2
40 INPUT  "Third? ",mark3
50 total=mark1+mark2+mark3
60 PRINT  "The total is"total
```

Program III

Here there's no change in

method from Program II but the way the variables are named hints at a solution to our problem. The numbers typed in are stored in *mark1*, *mark2*, and *mark3*. Now these are completely separate variables, but their names obviously have something in common. They consist for the most part of the same word, *mark* and are only made different by the numbers stuck on the end.

And those numbers on the end look ripe for a FOR ... NEXT loop. They just seem to beg for a structure like:

```
FOR x=1 to 3
INPUT markx
NEXT x
```

where as *x* changes so does the variable name *markx*. It's a great idea and only has one drawback — it doesn't work. But don't worry, Amstrad Basic provides the means for making it work, as Program IV shows.

```
10 REM Program IV
20 DIM mark(3)
30 INPUT "First? ",mark(1)
40 INPUT "Second? ",mark(2)
50 INPUT "Third? ",mark(3)
60 total=mark(1)+mark(2)+mark(3)
70 PRINT "The total is"total
```

Program IV

This looks remarkably like Program III, but what is that DIM doing in line 20? And why are the numbers after the *mark* in brackets? The answers are that the DIM is there to dimension an array, and the numbers in the brackets are subscripts.

To be formal, an array is an ordered set of linked variables. All the variables in an array have similar names such as *item(1)*, *item(2)*, *item(3)* and so on. We call them the elements of the array. The root is the same — in this case it's *item* — only the numbers in the brackets are different.

These numbers are what are known as the subscripts of the array. They actually pick which element of the array you're talking about. A subscript of 1, as in *item(1)* means you're referring to the first variable or element in the array, 2 to the second *item(2)*, and 50 to the fiftieth *item(50)*, if there is one.

Don't worry if that seems a little complicated, arrays are quite simple and a lot easier to use than to read about. So let's see how Program IV uses the array *mark()*.

The DIM in line 20 is used to tell the Amstrad to set aside enough memory space for an array. The number inside the brackets tells it how many variables, or elements, will be in the array. So a line like:

DIM result(20)

will set aside memory space for 20 variables ranging from *result(1)* and *result (2)* all the way up to *result(20)*. Each element is given the initial value of 0.

There's a little more to DIM, but we'll come back to it later. For the moment let's see how the array *mark()* — with elements *mark(1)*, *mark(2)*, *mark(3)* — is used in Program IV.

Lines 30 to 50 are our familiar input lines, only now they'r estoring the numbers in the elements of an array. Line 60 adds up the values of all these elements and stores them in total with the line 70 displaying the result.

But, you may be thinking, Program IV is more or less the same as Program III. Why both dimensioning an array and then use its elements just as you would a normal variable?

It's a very good point. If you're going to dimension an array, then use it properly. If you cast your mind back, you'll remember that we were after a way of linking variables together so they could be used in a FOR ... NEXT loop.
Program V shows you how to do this by making proper use of an array, *mark()*.

Here we're combining an array with a FOR ... NEXT loop for the first time. As you'll see, it's a very powerful combination.

```
10 REM Program V
20 DIM mark(3)
30 FOR loop=1 TO 3
40 PRINT "Enter mark number"loop
50 INPUT mark(loop)
60 NEXT loop
70 total=mark(1)+mark(2)+mark(3)
80 PRINT "The total is"total
```

Program V

Line 20 dimensions an array, *mark()*. This will consist of three elements, *mark(1)*, *mark(2)*, and *mark(3)*. Lines 30 to 50 form a FOR ... NEXT loop with a control variable loop. This will increase in value from 1 to 2 and then to 3 as the loop cycles.

Inside the loop, line 40 just prompts you to type in a number. Line 50 is the heart of the matter, and it's a strange looking beast:

50 INPUT mark(loop)

The INPUT is familiar, it's just asking for a number which it will store in the following variable. It's this variable that looks odd. After all, we know we have *mark(1)*, *mark(2)*, and *mark(3)*

but what is *mark(loop)*?

The answer is that *mark(loop)* is either *mark(1)*, *mark(2)* or *mark(3)*, depending on the value of *loop*. You see, the number inside the brackets doesn't have to be a figure, it can be a numeric variable such as *loop*. And by giving different values to this variable you can deal with different elements of the array.

Again it's easier to see in practice than in theory, so let's look at what happens to *mark(loop)* as the FOR ... NEXT loop of Program V cycles.

The first time round, *loop* has the value 1 so:

5Ø INPUT mark(loop)

effectively becomes:

5Ø INPUT mark(1)

When you satisfy the INPUT by giving it a number, this number is stored in *mark(1)*. When *loop* is 2, *mark(loop)* becomes *mark(2)*. I'll leave it to you to figure out what element of the array is used to store the third number, input when *loop* is 3.

By using a variable inside the bracket of an array you can, by changing the value of the variable, get at every member of the array. Rather more grandly, this means that using a variable as a subscript allows every element of the array to be addressed as the variable changes in value.

And now we've got what we were searching for — a technique that allows us the flexibility of using FOR ... NEXT Loops to input values and also lets us get at those values later in the program.

Take another look at Program V. Line 70 looks a little ugly, doesn't it? Surely there's a better way of doing the addition now we're using an array? There is, and Program VI shows how it's done.

```
1Ø REM Program VI
2Ø DIM mark(3)
3Ø total=Ø
4Ø FOR loop=1 TO 3
5Ø PRINT "Enter mark number"loop
6Ø INPUT mark(loop)
7Ø NEXT loop
8Ø FOR loop=1 TO 3
9Ø total=total+mark(loop)
1ØØ NEXT loop
11Ø PRINT "The total is"total
```

Program VI

The second FOR ... NEXT loop of the program is the one that does the adding. Each time round the loop, *mark(loop)* is pointing to a different variable. The first time round the number stored in *mark(1)* is added to total. The second time round it's *mark(2)* that is accessed and the third time *mark(3)*.

While this may seem a rather longwinded way of doing things — after all Program V was shorter — it is more flexible. If you wanted to add together 100 numbers think of the problems you would have adapting Program V.

With Program VI all you have to do is to change the scope of the FOR ... NEXT loops with:

4Ø FOR loop=1 to 1ØØ
8Ø FOR loop=1 to 1ØØ

Well, almost. You also have to change line 20 to:

2Ø DIM mark(1ØØ)

which allows for the fact that you now want to use 100 elements ranging from *mark(1)* to *mark(100)*. Try changing the loops but not the DIM in Program VI and see what happens. You get the error message:

Subscript out of range in 6Ø

as you try to give a value to *mark(4)*. This is fair enough, after all, the Dim of line 20 has only told your micro to set aside space for three elements in the array. You can't complain if the Amstrad baulks when you suddenly try to use a fourth.

And this bring us to the further discussion of DIM promised earlier. First of all I must cough to something (as my mate Al would put it). When I told you that:

DIM mark(3)

set up three elements of an array, *mark(1)*, *mark(2)*, and *mark(3)* I didn't tell you the whole truth. In fact it sets up four elements. As well as the three we've already mentioned, it sets up a variable with a subscript of 0, mark(0). This is because while humans begin counting at 1, computers tend to start at 0.

So if we wanted an array of 10 elements we could do it with:

DIM score(9)

which would set aside memory space for 10 elements ranging from score(0) to score(9). This is fine and it works, but I find it a little obscure.

What I mean is, that if I want to use an array of 10 elements in a FOR ... NEXT lop, I find it easier to deal with a loop that goes from 1 to 10 rather than one which goes from 0 to 9. This is especially so in long programs when I may be using a lot of arrays in nested FOR ... NEXT loops. The Amstrad can keep track of things but I can't!

Because of this I tend to always dimension my arrays to the full number. If I want 12 elements of

an array *month()* I use:

DIM month(12)

and only ever use elements *month(1)* to *month(12)*. I ignore the element *month(0)*, which only confuses things. After all, what is *month(0)*? It's a fairly good guess that *month(1)* refers to something that happened in January and *month(12)* to December ... but *month(0)*?

So for clarity I ignore the elements with subscripts of zero. Of course this means there are some variables that aren't used and the memory used to hold them is wasted. However your Amstrad has so much memory that this is rarely a problem.

A point to bear in mind is that once you've dimensioned an array with a line like:

1Ø DIM cat(3)

then that' s your lot. You can't then go on to dimension it again if you find you need more elements later on in the program. An attempt to do this, such as:

1ØØ DIM cats(2Ø)

will result in the program crashing with the message:

Array already dimensioned in 1ØØ

for your pains. So when you dimension an array make sure that you allow for all the elements you need. And it's also good practice to keep all your DIMs at the head of the program where both you and the micro can get at them easily.

The last point I'll make about DIM is that you don't always need it. If you leave it out and then try to use an array element such as *notdimmed(3)* you'll get away with it.

This is because when the Amstrad comes across something like *notdimmed(3)* it realises that the brackets mean that it's part of an array. It looks to see if it's been DIMned and, if not, being a very friendly machine it does it for you. It sets up the array *notdimmed()* and the program runs. But beware, it only sets it up with 11 elements. In other words it does the equivalent of a:

DIM notdimmed(10)

If you try to use notdimmed(15) you're in trouble. The program crashes with a:

Subscript out of range

message. The moral is, while you'll get away with not dimensioning arrays, you can only go so far.

My advice is always DIM your arrays, even if they have so few elements that the Amstrad would normally do it for you. It makes things clearer and saves a lot of bother.

As an example of what I mean, all the programs in this article will work without the arrays being dimensioned. But would it be as easy to see how they worked?

And what if you decided to have Program VI adding 100 numbers? If there was no DIM sitting there at the top of the listing you might easily forget to:
DIM mark(100)

And now, after all that, we can see how Program VII, which we met at the end of the last article, works

Line 20 sets up a six element array *number ()*, with members ranging from *number(0)* to *number(5)*. The program ignores *number(0)* and just uses the other five elements.

Lines 30 to 50 form the familiar FOR ... NEXT loop, with control variable *loop* taking values from 1 to 5.

```
1Ø REM Program VII
2Ø DIM number (5)
3Ø FOR loop=1 TO 5
4Ø READ number (loop)
5Ø NEXT loop
6Ø FOR loop=1 TO 5
7Ø PRINT number(loop)
8Ø NEXT loop
9Ø DATA 1ØØ,2ØØ,3ØØ,4ØØ,5ØØ
```

Program VII

What's different about the program is that each time round the loop value is READ from the data list of line 90, rather than INPUT from the keyboard as before.

So the first time round the loop *number(1)* takes the value 100, the next time *number(2)* becomes 200, and so on. The second FOR ... NEXT loop just prints out the values held by each element of the array.

And that's the last program for this month. Why not use it to explore how to use arrays?

For a start, try changing the number of elements in the array. See what happens if you try to read in more numbers than there are elements in the array? And can you make the program print out the values of the elements in reverse order?

As I keep stressing, programming is an activity. You can read about it all you want, but to be able to do it you have to practice.

So don't just accept what I say about arrays, try it out for yourself. And after a few hours of arrays on your Amstrad you'll be ready for next time when we take our arrays into a whole new dimension.

A Z80 assembler and screen editor

Part VIII of COLIN FOSTER's exploration of CP/M 2.2

This month we'll look at ZSM.COM — a Z80 assembler for CP/M and EDIT.COM — a fairly basic but usable screen editor which you can use to create assembler source files.

Both of these programs are in the public domain and are available through the CP/M user groups and ur public domain disc.

ZSM.COM is a typical CP/M assembler, and much of what follows will be applicable to any other similar program you may have. If you are used to Amsdos editor/assembler programs such as DevPac you'll find ZSM a bit different.

For a start you need a separate editor to create the source file on disc which the assembler will use as input instead of using an editor which comes built-in to the assembler — almost all CP/M assemblers and language compilers work in this way.

Figure 1 shows a simple ZSM example program as it would be typed into an editor — this program uses BDOS function 9 to print a message string on the screen.

Also Amsdos assemblers normally produce executable machine code output, either in memory which you run there and then, or as a .BIN disc file — CP/M assemblers and languages normally do not.

Instead they produce a file on disc containing one or two common types of intermediate object code — either Intel HEXadecimal or Digital Research/Microsoft RELocatable, signified by .HEX and .REL filetypes respectively. ZSM produces .HEX format files.

HEX files are simple to understand, but difficult to explain to someone who is not used to the idea. Basically they contain the same machine code as a .COM file, but in a form which is readable instead of being executable — an Ascii rep-

```
;       Demo program for ZSM assembler
;       Uses BDOS function 9 to print message on Console

        org     100h

lf      equ     10              ; ASCII linefeed
cr      equ     13              ; ASCII carriage return
bdos    equ     5               ; BDOS entry address

        ld      de,text         ; Load DE with address of start of text
        ld      c,9             ; Select BDOS function 9
        call    bdos            ; Print the string
        jp      0000h           ; Exit back to CP/M by Warm Boot

text:   defb    cr,lf,cr,lf
        defb    'Hello, World!'
        defb    cr,lf,cr,lf
        defb    '$'

        end
```

Figure I: Example ZSM source program

```
First and subsequent blocks:

   : 08 0100 00 1234567890ABCDEF ED (CR)
   ¦ ¦  ¦  ¦      ¦              ¦  ¦
   ¦ ¦  ¦  ¦      ¦              ¦  ¦    Terminating carriage return
   ¦ ¦  ¦  ¦      ¦              Two's complement checksum of BYTES on line
   ¦ ¦  ¦  ¦      ASCII "hex" data
   ¦ ¦  ¦  "Type" - usually zero
   ¦ ¦  Start address for this block
   ¦ Number of DATA BYTES in block
   Colon - "Start of block" marker

Last block:

   : 00 0234 00 CA (CR)
   ¦ ¦  ¦  ¦  ¦  ¦
   ¦ ¦  ¦  ¦  ¦  ¦   Terminating carriage return
   ¦ ¦  ¦  ¦  ¦ Two's complement checksum
   ¦ ¦  ¦  Type
   ¦ ¦  Entry address of program (will be 0000 if no ENTRY specified)
   ¦ "Null" data length
   Start of block marker

 Note: Spaces in the above are for clarity only: they are NOT present
       in an actual .HEX file.
```

Figure II: Structure of a .HEX file

resentation of the binary numbers.

They also contain other information, such as the load address of the code and checksums. We can use the Type command to look at a .HEX file on the screen. Figure II shows how such a file is made up, and Figure III shows the HEX output from our example source program.

Microsoft .REL relocatable intermediate files are a common form of output from more powerful assemblers and most CP/M language compilers — for example Microsoft's Macro-80 assembler, Pascal MT/+, Fortran-80, and DR's Rmac assembler supplied with CP/M Plus.

.REL is a much more complex form of output than .HEX, and I won't go into it in any great detail here. Basically .REL format is designed to allow programs to be written as separate and independent modules, each of which can nevertheless refer to routines and variables in other modules or in existing library files.

The different modules are then put together into one program, and all the external references between modules resolved by a program called a linker. Examples of linkers are DR's Link, which is supplied with CP/M plus, and Microsoft's Link-80.

.HEX files are designed for transmission along RS232 or telephone cables, and for downloading to target hardware under development from another computer — their set Ascii format and checksums allow easy error checking when they are received at their destination.

Before we can run such files we must convert them into straight binary .COM files — CP/M's equivalent to AMSDOS .BIN program files. There are two ways of doing this.

Firstly we can use the utility LOAD.COM — if we type:

A>load our prog

then Load will read the OUR-PROG.HEX file created by the assembler and will created an OUR-PROG.COM program file on disc which we could then run. This is the best method if what we have written is a stand-alone program.

However if we have just written a small routine which we wish to insert into a larger program — this is what you will see referred to as patching — or we have not ORGed our program at &100 — Load always assumes you want a .COM file which will load at the start of the TPA — then we can use DDT.COM — the system debugger we looked at in a previous article.

I mentioned then, without going into detail, that DDT would convert a .HEX file into binary before loading it into memory for examination. If we type:

A>ddt ourprog.hex

then DDT will load the machine code content of the .HEX file to exactly the addresses specified by our assembler ORG instructions — which need not be to &100.

```
:1C01000011CB01CF D9CD05002C3D0000000A0D0A4B6564C6F2C2D576F726C642183
:05011C000D0A0D0A248C
:00000000FF
```

Figure III: .HEX output from ZSM for program in Figure I

So to insert a patch into an existing program we could load the main program into DDT and then load our patch, overwriting the section of program which was there before, as follows:

```
A>ddt mainprog.com
DDT VERS 2.2
Next PC
1FØØ Ø1ØØ
-iourpatch.hex
-r
NEXT PC
1FØØ ØØØØ
-^C
A>
```

DDT automatically takes care of loading the patch in the right place by using the Load Address information present in the .HEX file. Now DDT has told us the length of MAINPROG.COM when it was loaded: &1F00 — &100 = &1E00 bytes = 30 decimal pages — so we can save the modified program to disc by typing:

A>save 3Ø mainprog.com

immediately after existing DDT.

To assemble a program using ZSM you must first type it in to an editor and save it on disc with a filetype of .ZSM — otherwise ZSM will not be able to read it.

This month's disc also has the assembler test program ZSMTEST.ZSM already on it. You can assemble it — or your own programs once you have typed them in — by a command of the form:

A>zsm zsmtest

ZSM will then sign on and begin assembling the source file. As it goes along it tells you if you have made any mistakes in your source program. If you have you must edit your source to correct them and then reassemble.

This process may have to be repeated a few times until you have corrected all your errors. If you try assembling ZSMTEST.ZSM don't panic when all the error messages start coming up. This program has been deliberately written with lots of different errors in it to test the assembler's ability to detect them.

As an exercise you could try to spot the mistakes and correct them all, though even then ZSMTEST wouldn't run as it's not a real program.

After assembling a program look at a directory listing on your disc — you will find that ZSM has also created yet another file, this time with a .PRN extension.

This is the listing file, which contains both your source program and the Ascii equivalents of the machine code created for each line as it has been put into the .HEX file.

Again use the Type command to examine the .PRN file — you will see that if you have any comment lines the extra information inserted into the start of each line causes the lines to overflow on the screen and it looks a bit of a mess.

However if you list the .PRN file on a printer set to condensed mode — to give it more than 80 characters per line — you will see it as it really is. At the end of your listing file there is a Symbol table which lists all the labels and other symbols you have defined in your program and their values in HEX.

Listing files and their symbol tables are extremely useful

```
ZSM-2.8   Source file name: BDOS9                Page No:   1

0000            ;       Demo program for ZSM assembler
0000            ;       Uses BDOS function 9 to print message on Console
0000
0100 =                  org     100h
0100
000A =          lf      equ     10              ; ASCII linefeed
000D =          cr      equ     13              ; ASCII carriage return
0005 =          bdos    equ     5               ; BDOS entry address
0100
0100 110301             ld      de,text         ; Load DE with address of start of text
0103 0E09               ld      c,9             ; Select BDOS function 9
0105
0105 CD0500             call    bdos            ; Print the string
0108
0108 C30000             jp      0000h           ; Exit back to CP/M by Warm Boot
010B
010B 0D0A0D0A   text:   defb    cr,lf,cr,lf
010F 48656C6C           defb    'Hello, World!'
011C 0D0A0D0A           defb    cr,lf,cr,lf
0120 24                 defb    '$'
0121
0121 =                  end

Errors   0

Next address: 0121H

ZSM-2.8   Source file name: BDOS9                Page No:   2

BDOS      0005   CR       000D   LF       000A   TEXT     010B
```

Figure IV: .PRN listing file produced by ZSM from program in Figure I

while you are debugging programs, as they tell you where everything is. Figure IV shows what the .PRN file for our example program looks like.

We saw earlier that ZSM always looks for a file name type extension of .ZSM on our source files, so we didn't need to specify this when we ran the assembler.

In fact, like several other CP/M assemblers including ASM and Mac, the filename extension can be used, but to give ZSM extra parameters rather than different filetypes. The exact format is as follows:

A>zsm filename.pqr

P = the disc drive containing the source file FILENAME.ZSM — A or B for drives A and B respectively, or @ for the default drive.

q = the disc drive where the object code file FILENAME.HEX will be put — A, B, @ as before, or Z if object code is not required.

r = the disc drive where the listing file FILENAME.PRN is to be put — A, B, @ Z as before, P sends the file to the LST: device, X sends output to the console and Y sends the listing to the console but prints errors on the LST: device.

So, for example, the command:

A>zsm ourprog.aab

will tell ZSM to assemble the source file OURPROG.ZSM from the disc in drive A, putting the object file OURPROG.HEX also on to A and the listing file OURPROG.PRN on to drive B.

There is also a text file, ZSM.DOC, on this month's disc — this gives detailed information on ZSM's syntax and error messages. Use Type to look at it on the screen — remember you can pause the output with Control+S to give you time to read it — or list it on your printer using Pip. For example, typing:

A>pip lst:=zsm.doc[p66]

will list the file on your printer, automatically inserting a formfeed every 66 lines — April's article described Pip in more detail.

EDIT.COM is a screen editor — this means that you use the cursor keys to move about the screen and type what you want where you want it, just as if the screen were a piece of paper.

This is a lot more useful than the line editor Amsdos gives us to edit Basic programs. For a start we can type in anything — source code for any assembler or language compiler, letters, shopping lists and so on.

Because different computers all have different hardware controlling their screens and respond to different commands to do things like position the cursor, insert a line and so on, programs such as Edit must be configured specially for each machine on which they will run.

The version of EDIT.COM which we are supplying has already been configured for CP/M 2.2 on Amstrad CPC464/664/6128 computers — it will not work properly under CP/M Plus on a 6128 or 8256/8512.

Before you can run Edit you must use SETUP.COM to configure a disc to set up the keyboard properly when you cold boot.

This procedure was discussed in the article in the March edition of *Computing with the Amstrad*, but don't panic if you missed that — I've repeated the important bits briefly at the start of the editor document file EDGUIDE.DOC on the disc.

EDGUIDE.DOC is a comprehensive user's manual for the Edit program. If you don't have a printer I suggest you find a friend who does. You can read large files like this by typing them to the screen, but it's quite an effort.

If you can't get at a printer I suggest that you read as much of it as possible, then once you have a basic idea how Edit works refer instead to the other document file on disc, EDSUM.DOC. This provides a briefer summary of Edit's commands — useful once you understand what they do.

Edit is not a word processor unfortunately. Lack of quality programs of this sort is a great weakness in public domain software — that's a hint for anyone who fancies writing one.

If you already have a CP/M word processor such as NewWord or WordStar you can use it in N — Non-Document — mode to create program source code for any assembler or compiler.

The Ascii files produced by Tasword are also compatible with CP/M, so if that's all you've got you could edit all your source code under Amsdos and enter CP/M to assemble and run the programs.

• Next month we'll return to CP/M itself for a closer look at discs, to see how they actually store our programs.

Steam along and build up your character the easy way



Part VII of the Amstrad graphics series by GEOFF TURNER and MICHAEL NOELS

If you can cast your mind back that far, you'll remember that last month we entered the world of user defined graphics. I hope by now you've had some practice in designing your own characters.

```
10 REM PROGRAM 1
20 MODE 1
30 SYMBOL 240,0,0,0,0,0,0,0,0
40 SYMBOL 252,255,129,129,129,129,129
,129,255
50 SYMBOL 253,255,255,255,255,255,255
,255,255
60 SYMBOL 254,255,129,129,153,153,129
,129,255
70 SYMBOL 255,255,255,255,231,231,255
,255,255
80 FOR row=1 TO 8
90 FOR column=1 TO 8
100 LOCATE column,row
110 PRINT CHR$(252)
120 NEXT
130 LOCATE 12,row
140 PRINT USING "###";value(down)
150 LOCATE 18,row
160 PRINT BIN$(value(down),8)
170 NEXT
180 column=1:row=1
190 LOCATE column,row
200 PRINT CHR$(254)
210 WHILE -1
220 oldcolumn=column
230 oldrow=row
240 GOSUB 460 :REM update numbers
250 GOSUB 570 :REM print character
260 IF INKEY(0)=0 THEN row=row-1
270 IF row=0 THEN row=8
280 IF INKEY(2)=0 THEN row=row+1
290 IF row=9 THEN row=1
300 IF INKEY(8)=0 THEN column=column-
1
310 IF column=0 THEN column=8
320 IF INKEY(1)=0 THEN column=column+
1
330 IF column=9 THEN column=1
340 LOCATE oldcolumn,oldrow
350 PRINT CHR$(252+status(oldcolumn,o
ldrow))
360 LOCATE column,row
370 PRINT CHR$(254+status(column,row)
)
380 IF INKEY(9)=0 THEN GOSUB 400:REM
change pixel
390 WEND
400 REM plot / unplot pixel
410 PRINT CHR$(7)
420 status(column,row)=ABS(status(col
umn,row)-1)
430 LOCATE column,row
440 PRINT CHR$(254+status(column,row)
)
450 RETURN
460 REM update numbers
470 down=row
480 value(down)=0
490 FOR across=1 TO 8
500 IF status(across,down)=1 THEN val
ue(down)=value(down)+(2^(8-across))
510 NEXT
520 LOCATE 12,down
530 PRINT USING "###";value(down)
540 LOCATE 18,down
550 PRINT BIN$(value(down),8)
560 RETURN
570 REM print character
580 SYMBOL 240,value(1),value(2),valu
e(3),value(4),value(5),value(6),value
(7),value(8)
590 LOCATE 5,12
600 PRINT CHR$(240)
610 RETURN
```

Program 1

However if you're struggling with the calculations involved don't worry, you'll find our first program will help.

It is a simple version of a character generator, which takes all the hard work out of designing characters. The July 1987 issue of *Computing with the Amstrad* contains an all-singing, all-dancing character generator if you're feeling ambitious. For the present, though, Program I will suffice.

When you run it you'll find that it displays a large 8 by 8 character grid with a movable cursor. The Amstrad's cursor keys are used to guide this around the grid to any one of the 64 squares.

When you wish to fill or plot a square simply press the Copy key, and the currently selected square will be filled. The square is unplotted by pressing the Copy key once more. Designing your own characters now becomes easy. You just fill in the squares as needed to create your own character.

As you use it you'll see that the program takes care of all the necessary calculations. The current value of each row is displayed alongside the grid. Just for reference, the binary values are also displayed.

Notice how the ones and zeroes of the binary numbers correspond to the squares of the grid. A one means that the square is filled, a zero that it is left empty.

The character is printed below the grid in its actual size, so that you can see how it will look in your programs. Once you're satisfied with a character you should make a note of the eight values which make up the design. These can then be used with the SYMBOL command to produce the same character in your own programs.

Now we'll take a look at a few techniques which you may find useful when using your characters in programs.

In Program II, we've designed a steam engine. Reset the Amstrad to clear Program I before you run Program II. We wanted the character to be quite large, so we used six individual characters to make up the finished display. The overall size of the design is three characters across by two characters down.

Having designed the shapes, the next problem was how to get them on to the screen in the correct positions. The top row of the steam engine consists of characters 240, 241 and 242, while the bottom row is made up from 243, 244 and 245.

There are several ways of printing this character. Proba-

```
10 REM PROGRAM II
20 MODE 1
30 PAPER 2
40 PEN 3
50 CLS
60 SYMBOL 240,0,127,127,17,17,17,31,3
1
70 SYMBOL 241,0,0,0,0,24,60,60,255
80 SYMBOL 242,0,0,30,12,12,12,12,254
90 SYMBOL 243,127,127,127,127,127,127
,28,8
100 SYMBOL 244,255,255,255,255,33,115
,115,33
110 SYMBOL 245,255,255,255,255,8,156,
156,8
120 LOCATE 18,12
130 PRINT CHR$(240);CHR$(241);CHR$(24
2)
140 LOCATE 18,13
150 PRINT CHR$(243);CHR$(244);CHR$(24
5)
160 WHILE NOT true:WEND
```

Program II

```
10 REM PROGRAM III
20 MODE 1
30 PAPER 2
40 PEN 3
50 CLS
60 SYMBOL 240,0,127,127,17,17,17,31,3
1
70 SYMBOL 241,0,0,0,0,24,60,60,255
80 SYMBOL 242,0,0,30,12,12,12,12,254
90 SYMBOL 243,127,127,127,127,127,127
,28,8
100 SYMBOL 244,255,255,255,255,33,115
,115,33
110 SYMBOL 245,255,255,255,255,8,156,
156,8
120 Top$=CHR$(240)+CHR$(241)+CHR$(242
)
130 Bottom$=CHR$(243)+CHR$(244)+CHR$(
245)
140 Engine$=Top$+CHR$(10)+CHR$(8)+CHR
$(8)+CHR$(8)+Bottom$
150 LOCATE 18,12
160 PRINT Engine$
170 LOCATE 18,13
180 PRINT CHR$(243);CHR$(244);CHR$(24
5)
190 WHILE NOT true:WEND
```

Program III

bly the most obvious is to move the text cursor to the required position, then print the first three characters next to each other. The cursor can then be moved to the correct position on the next row down with a cunning LOCATE, and the final three characters printed.

Program II uses this method. While it obviously works, it's not very clear what the program is doing. This would be particularly so if it were part of a longer listing.

A much better way is to combine the six separate characters into a string variable, and then print the string all in one line. Program III demonstrates how this is done.

At line 120 we've concatenated or linked the top three characters into one string and called it *Top$*. The remaining three characters are combined into *BottomS* at line 130. Finally *Top$* and *Bottom$* are joined in line 140.

Notice that we have included some new characters in the finsihed string. Can you remember what CHR$(10) and CHR$(8) do? These are the cursor control codes. Here they are used to place the cursor in the correct position ready to print *Bottom$*. We are moving the cursor one row down (CHR$(10)), and three positions back (CHR$(8)). This places it directly under the beginning of *Top$*.

Finally we've given a meaningful name to the completed string and called it *Engine$*. Now whenever we wish to print the complete character, it's simply a matter of using the command.

PRINT Engine$

using LOCATE to give the position required.

On the face of it, it does seem a lot of work to produce the same result as Program II. However you will find as your programs become longer and more complex that it will be much easier to use this method.

One of the drawbacks with printing characters is that they can normally be printed in the current pen and paper colours. The foreground is printed in the pen colour, while the background is printed in the paper colour.

It might, however, be that we want to produce a character consisting of several different co-

```
10 REM PROGRAM IV
20 MODE 0
30 PAPER 5
40 CLS
50 REM face
60 SYMBOL 240,3,7,15,31,63,63,63,127
70 SYMBOL 241,192,224,240,248,248,248
,252,254
80 SYMBOL 242,127,255,255,255,255,127
,63,63
90 SYMBOL 243,254,255,255,255,255,254
,252,252
100 SYMBOL 244,63,63,63,31,31,31,7,3
110 SYMBOL 245,252,252,252,248,248,24
8,224,192
120 row$=CHR$(10)+CHR$(8)+CHR$(8)
130 face$=CHR$(240)+CHR$(241)+row$+CH
R$(242)+CHR$(243)+row$+CHR$(244)+CHR$
(245)
140 PEN 11
150 LOCATE 9,12
160 PRINT face$
170 WHILE INKEY$="":WEND
180 REM eyes
190 SYMBOL 246,0,0,0,0,4,14,0,0
200 SYMBOL 247,0,0,0,0,32,112
210 eye$=CHR$(246)+CHR$(247)
220 PEN 6
230 LOCATE 9,12
240 PRINT eye$
250 WHILE INKEY$="":WEND
260 REM nose
270 SYMBOL 246,0,0,0,1,1,0,0,0
280 SYMBOL 247,0,0,0,128,128,0,0,0
290 nose$=CHR$(246)+CHR$(247)
300 PEN 7
310 LOCATE 9,13
320 PRINT nose$
330 WHILE INKEY$="":WEND
340 REM mouth
350 SYMBOL 248,0,0,4,3,0,0,0,0
360 SYMBOL 249,0,0,32,192,0,0,0,0
370 mouth$=CHR$(248)+CHR$(249)
380 PEN 3
390 LOCATE 9,14
400 PRINT mouth$
```

Program IV

lours. We may want to print a pink face, with blue eyes and red lips. The trouble is that if we design the whole character in Basic using the SYMBOL command there's no facility to plot it in several colours.

To get around this we need to design several characters, each making up a different part of the face. The main symbol would be the head shape, other symbols representing the eyes, nose and mouth.

It seems fairly plausible that if we first select a suitable ink colour to print the face, then switch to blue ink and move the cursor, we can overprint the eyes in blue over the face. Unfortunately, as you'll see in Program IV, this method doesn't work satisfactorily.

The program works in stages, each requiring a key to be pressed before it moves on to the next part. First it prints the overall face shape. So far, so good. However when the eyes are printed the black background of the eye characters overwrites the previously printed pink face.

This is because characters printed at the text cursor always print a character square whose foreground and background is made up of the currently selected pen and paper. The same problem occurs with the nose and the mouth.

Happily there is a way of overcoming this. We can print characters in what's known as transparent mode, in which the background or paper colour of a character isn't printed. It is, in effect, transparent, so anything that was on the screen previously shows through.

This means that if we put the Amstrad into transparent mode we could put our blue eyes on to the face but still have the pink of the face showing through the transparent background of the eye character.

There is no basic keyword for transparent printing. We have to use control code 22 followed by a second number which turns it on or off. Transparent mode is turned on by:

```
PRINT CHR$(22);CHR$(1)
```

and turned off again by:

```
PRINT CHR$(22);CHR$(Ø)
```

When using transparent mode you should always remember to turn it off again at a suitable point in your program or strange things may happen to your displays.

To modify Program IV, add the following new lines:

```
175 PRINT CHR$(22);CHR$(1)
41Ø PRINT CHR$(22);CHR$(Ø)
```

The eyes, nose and mouth will now be printed without destroying any part of the face. This is because the paper they are printed on is see-through, letting the previous background show. So using this method we can build up multi-coloured characters.

Incidentally, the character could have been linked together into one long compilation of a string variable, as we did with our steam engine.

If you examine the list of control codes in the User Manual you will see that there are also control characters to select pen and paper inks.

Perhaps you might like to produce a single string variable which will print the multicolour face all in one go. It's possibe and, if you were using a lot of faces on the screen, could make things easier.

Another useful facility when printing characters on the screen is the ability to print at the position of the graphics cursor instead instead of the text cursor. The advantage of this is that characters may be then printed at any graphics coordinate and we are not limited to the usual, rather clumsy, text cursor locations.

This greater definition can be useful when labelling diagrams and graphs and so on.

To achieve printing at the graphics cursor we use the TAG command. Program V demonstrates how it affects the position of the printed text.

The first print command at line 30 occurs at the position of the text cursor. Having issued the TAG command at line 40 you'll see that the next print command is obeyed down at the bottom of the screen where the graphics cursor has been positioned by line 50.

```
10 REM PROGRAM V
20 MODE 1
30 PRINT "TAG OFF"
40 TAG
50 MOVE 0,64
60 PRINT "TAG ON"
70 TAGOFF
```

Program V

It was necesary to move the gaphics cursor slightly up the screen because if it was left at its home position, (0,0), the printed text would be below the bottom of the screen. Leave out line 50 and see what happens.

Notice that when printing at the graphics cursor the ink used belongs to the current graphics

```
10 REM PROGRAM VI
20 MODE 0
30 SYMBOL AFTER 65
40 LOCATE 1,8
50 PRINT"THIS IS MODE 0"
60 LOCATE 1,10
70 REM Normal size
80 PRINT"ABCDE"
90 SYMBOL 65,96,144,144,240,240,144,1
44,0
100 SYMBOL 66,224,144,144,224,144,144
,224,0
110 SYMBOL 67,96,144,128,128,128,144,
96,0
120 SYMBOL 68,224,144,144,144,144,144
,224,0
130 SYMBOL 69,240,128,128,224,128,128
,240,0
140 LOCATE 1,12
150 REM Redefined letters but gaps th
e same
160 PRINT "ABCDE"
170 PRINT
180 TAG
190 REM Adjusting the gaps
200 MOVE 0,192
210 PRINT"A";
220 MOVE 20,192
230 PRINT"B";
240 MOVE 40,192
250 PRINT"C";
260 MOVE 60,192
270 PRINT"D";
280 MOVE 80,192
290 PRINT"E";
300 TAGOFF
310 WHILE INKEY$="":WEND
320 SYMBOL AFTER 65
```

Program VI

pen and not the text pen. This presents a poblem if we wish to change the colour of text printed at the graphics cursor.

These's no direct way of changing the graphics pen except by plotting or drawing. This can be a nuisance as you don't always want to draw something over your nice display just to change the graphics colour.

One solution is to perform a dummy PLOT at a point off the screen. If you add a line like:

45 PLOT 1ØØØ,1ØØØ,3

to Program V then the graphics pen will be chanted to number 3, and all text printed at the graphics curor will now be in pen 3.

Observant readers will have noticed another effect of printing at the graphics cursor. After printing any characters the arrow symbols representing line feed and carriage return are also displayed. To suppress these control symbols we must place a semicolon at the end of any print statements followed by a TAG. Try doing this in program V.

In the last artricle we mentioned that it was unlikely that you would need to redesign all

```
10 REM PROGRAM VII
20 MODE 0
30 BORDER 0
40 REM PICTURE FRAME
50 PEN 1:PAPER 0
60 LOCATE 1,1
70 PRINT STRING$(20,207)
80 FOR y=2 TO 24
90 LOCATE 1,y
100 PRINT CHR$(207)
110 LOCATE 20,y
120 PRINT CHR$(207)
130 NEXT
140 LOCATE 1,25
150 PRINT STRING$(20,207);
160 REM SKY
170 WINDOW #0,2,19,2,24
180 INK 15,11
190 PAPER 15
200 CLS
210 REM SEA
220 WINDOW #0,2,19,12,24
230 PAPER 6
240 CLS
250 REM SAND
260 WINDOW #0,2,19,18,24
270 INK 13,15
280 PAPER 13
290 CLS
300 REM GRASS
310 WINDOW #0,2,19,2,24
320 SYMBOL 240,64,32,34,148,85,100,60
,126
330 PEN 12
340 FOR grass=1 TO 20
350 x=INT(RND(1)*17)+2
360 y=INT(RND(1)*6)+17
370 LOCATE x,y
380 PRINT CHR$(240)
390 NEXT
400 REM WAVES
410 SYMBOL 241,0,34,85,8,0,0,0,0
420 FOR waves =1 TO 20
430 PEN 4:PAPER 6
440 x=INT(RND(1)*15)+2
450 y=INT(RND(1)*5)+11
460 LOCATE x,y
470 PRINT STRING$(3,241)
480 NEXT
490 REM SHIP
500 SYMBOL 242,2,2,18,50,114,127,127,
63
510 SYMBOL 243,0,0,0,0,30,255,255,255
520 SYMBOL 244,4,4,4,4,15,254,252,248
530 ship$=CHR$(242)+CHR$(243)+CHR$(24
4)
540 LOCATE 5,10
550 PAPER 15:PEN 3
560 PRINT ship$
570 REM SUN
580 MOVE 500,300
590 DEG
600 FOR x=0 TO 360 STEP 20
610 DRAW 50*SIN(x)+500,50*COS(x)+300
620 MOVE 500,300
630 NEXT
640 FOR x=0 TO 360 STEP 4
650 MOVE 30*SIN(x)+500,30*COS(x)+300
660 DRAW 30*SIN(360-x)+500,30*COS(360
-x)+300
670 NEXT
680 GOTO 680
```

Program VII

the alphanumeric character set. But there is one application where this could be useful.

By now you should be familiar with the different screen modes. You're probably well aware that if you want a large selection of colours you need to use Mode O. Unfortunately this suffers from only allowing 20 characters per line.

Wouldn't it be nice to have the possibility of 16 colours and 40 characters per line? Well you may be surprised to learn that it can be done (well, almost).

In Mode O the characters are displayed twice their normal width, where "normal" means Mode 1. If we could redesign any character to occupy only the left hand side of a character cell then it would appear normal size when printed in Mode O. We could do this for each letter of the alphabet, using the character generator program, and we would then have a set of characters which were only half normal width.

We could therefore print a lot more than 20 of these characters on one line in Mode O.

There is, of course, a snag (there always is). If we print the characters at the text cursor they will still be printed at intervals relative to Mode O character cells. This means that we're still stuck with 20 per line.

If we redefined our characters in this way and used the command:

PRINT"ABC"

in mode O, it would appear as:

A B C

with a gap between each letter.

Have you worked out how to overcome this problem? Remember that we can use the TAG command and then print at graphics X and Y coordinates. So instead of the above we could first print A, MOVE the graphics cursor just to the right of the A and then print B. In effect, we close up the gaps between the characters.

Program VI demonstrates this. We've only redesigned a few characters to illustrate the technique. Obviously it's a lot harder to print text in this way, but it can be very useful when printing just a few words such as SCORE, BONUS or LIVES on the screen.

Unfortunately some characters such as M and W are difficult to portray in half normal width, so these will need to occupy a slightly wider space. The movement of the graphics cursor has to be adjusted to take account of this.

In Program VI there are 20 pixels per character, which will actually allow us 32 characters per line. Although we can't manage the full 40 characters per line we can at least improve on the standard 20.

A variation on this technique could be used to produce larger characters in Mode 2 — normally 80 characters per line — which whould be useful for producing large headings or titles. In this case we would need to spread each character over two character positions and print the two new characters side by side.

That's something for you to experiment with.

To finish for this month, Program VII brings together many of the topics covered in all the previous chapters. It's a sort of refresher course.

See if you can work through the program and understand all the techniques used. The program is broken down into sections for clarity, with appropriate REM statements so it shouldn't be too hard.

And by the time you've finished you should be ready for next month's discussion of logical colours.

Don't miss out...

When you want to keep up with the latest information on Amstrads you can't afford to miss an issue. Subscribe now for only $45. Don't miss out. Post the subscription order form in the centre of this issue.

Virgin is on the ball again

Football fever at Virgin Games has resulted in an update of last year's best-selling FA Cup Football program for the CPC.

It has new form figures for every team in the competition, re-computed by sports commentator Tony Williams from their current performance, to ensure the program gives the most realistic results.

The program also includes a new batch of strategic manager's questions.



Virgin programmers create their own version of Spot the Ball

Business Package Breaks Into Gallup's Games

A business software package available for the CPC has achieved a world first by storming its way into the all-important Gallup Top 40 chart.

Mini Office II, the award winning title from Database Software, has entered the best-selling list at number 25.

The chart is considered the most significant of all on the UK software scene because it includes all machine formats.

However Mini Office II's recent breakthrough in the games first division doesn't end there.

And although it was only released for the Commodore 64 in the last week of February, sales have been so spectacular that it is already number 25 in the Gallup Top 40 for the leading machine.

What is even more surprising about Mini Office II's chart successes is that priced at $48.95 it has to compete against games costing as little as $9.95.

"Yes we certainly have had quite an incredible week with Mini Office II," says Michael Meakin, head of Database Software. "But there again, the package has had a truly remarkable history".

Launched in October 1984, the original Mini Office was the finalist in two categories of the British Microcomputing Awards, its much enhanced successor, Mini Office II, took 26 man-years of programming and contains six modules — word processor, database, spreadsheet, graphs, label printing and communications.

Telecom golds

British Telecom's software titles have scooped an amazing total of 31 awards here and abroad in the past twelve months.

The accolades were awarded by reader polls and computer journalists' votes in magazines published in Britain, France, Germany and the United States.

More than two-thirds of Telecomsoft's sales of programs under the Rainbird, Firebird and Beyond labels are now overseas.

Latest release from Rainbird-venture of the Year at the Golden Joystick Awards.

It is set in the mythical land of Kerovnia, familiar to players of The Pawn, and features a large number of complex puzzles and 30 scene-setting illustrations.

Ian's home-brewed Basic

Home wine-making expert and CPC user Ian McFarlane has written a program in Basic to help others take a more scientific approach to their hobby and produce consistently good results.

Called Vinsoft, the program contains essential data for 90 commonly used ingredients, and formulae for calculating alcohol yields based on latest research findings.

Imperial to metric conversions are also built in.

The main feature is an automatic recipe formulator based on a choice of 16 standard commercial equivalents, with the option to modify each specification for sweetness, acidity and strength.



A recipe can be created from up to 90 different ingredients stored in the program. At any stage it will say how far away the recipe is from the selected style and, if requested, will automatically change the recipe to get the balance right.

A hard copy of the finished recipe and original specification can be taken if a printer is attached to the CPC.

Licensing Venture

Software house Activision has been scouring the globe for titles with which to launch itself into arcade games production for the CPC.

In its first venture into the licensing sector, the company has struck deals with Atari Games, Bally Sente, Data East, Nichibutsu and Sega.

This means an impressive line-up for the CPC in 1987 from Activision and its Electric Dreams and System 3 labels.

Galactic adventure Star Raiders II has been licensed from Atari and will sell as an Electric Dreams title.

Also on the Electric Dreams label will be Firetrap from Data East.

Top-selling coin-op car racing game Supersprint has been licensed from Atari and will be released by Electric Dreams on cassette and disc.

Licensed from Sega are Enduro Racer, Wonderboy and Quartet — a trilogy of arcade titles.

In addition, Abstract Concepts — producer of Bored of the Rings and The Boggit — has agreed to create CPC games for Activision.

Pride And Prejudice

Amstrad software supplier Pride Utilities has won five figure damages in the latest stage of its legal battle with former French suppliers.

Taking advantage of the large market for utilities in France, ESAT started its association with Pride well. But the large number of orders began to drop off sharply. Then, so too did the payments for goods, Pride claimed. So Pride sent someone over to the distributor's shop. It was discovered that ESAT was duplicating massive amounts of Pride software.

Despite the success in reclaiming £UK10,000 ($A25,000) owed and costs, Pride is continuing with its action alleging piracy.

Show reflecting Amstrad boom

The CPC range takes centre stage at the giant Amstrad Computer Show in London in July.

A survey of exhibitors booked to date has shown that companies in this sector of the market outnumber both PC and PCW suppliers by two to one.

It also revealed that more than 100 new products are currently under development for the CPC machines.

Though games make up the lion's share of these — some 54 per cent — a further 31 per cent are business programs or utilities, nine per cent are educational programs, while the remainder are peripherals.

And all the firms involved are hoping to complete them in time for launch at the show.

The three-day event will not only be the largest Amstrad Computer Show seen to date but almost certainly the most extensive machine-specific show ever held.

As a reflection of the current Amstrad boom, it has been moved to one of London's most famous venues, Alexandra Palace.

Housed in the spacious Alexandra Pavilion, there will be 50 per cent more exhibition space than at the previous location for the show, the Novotel at Hammersmith.

In line with this, the popular Amstrad Theatre will have its seating capacity doubled and provide virtually non-stop sessions during show hours.

Apart from being a historic location — it's where the world's first television service was launched 50 years ago — Alexandra Palace is easily accessible both from central London and the newly completed M25 and has more than 2,000 free parking spaces.

"We are anticipating an overwhelming response", said a spokesman for Database Exhibitons. "It is shaping up to be an event that Amstrad users will remember for years".

Games console on the way?

There are strong indications that Amstrad intends to follow Atari's recent move and bring out a games console.

While Amstrad policy is not to discuss product development, a spokesman for a leading software house told Computing with the Amstrad CPC: "I can confirm they are working on a machine of this type".

"I was briefed on the project by Amstrad recently, but they made me sign a non-disclosure agreement so I can't give you any details about it".

Other sources, however, say the device will use the sophisticated 68000 microprocessor found in upmarket machines like the Atari ST and Commodore Amiga.

This will allow it to run arcade games with all the graphics and sound features of the coin-op originals.

Apparently there are no plans to supply the console with a keyboard. It is expected to cost under $250 and come with joystick and some software.

Digging for Database

The world's first mass sod-cutting ceremony heralded the start of building a new £1 million ($A2.5m) home for Computing with the Amstrad CPC and its sister Database Publications magazines.

Every one of the 150 Database employees took part in the traditional act of ground-breaking to celebrate their role in the company's rise to prominence in computer magazine publishing.

The new site in the Cheshire countryside near Stockport echoed to the sound of scores of spades and shovels hitting the ground simultaneously at a signal from Database head Derek Meakin.

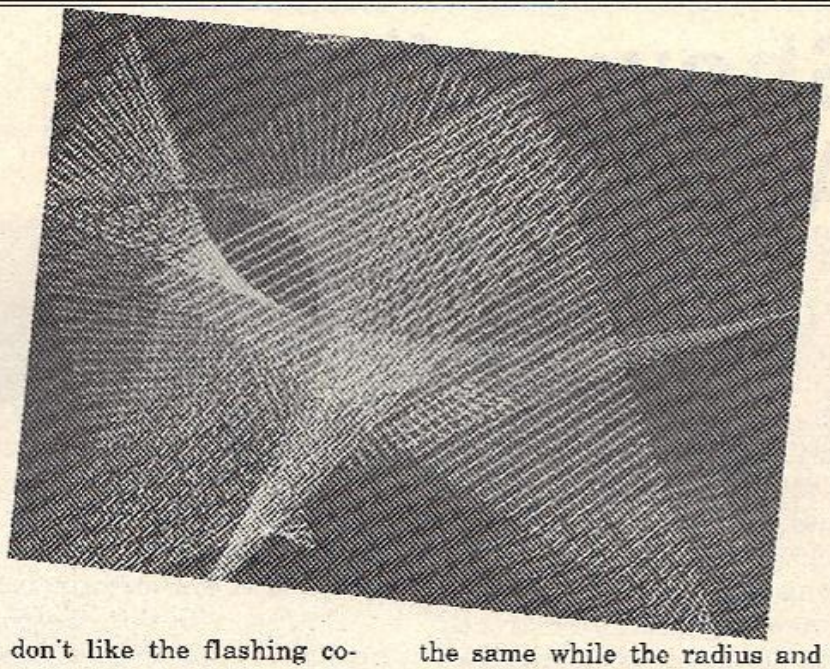
"We originally toyed with the idea of inviting a leading figure in the computer industry to perform the ceremony", he said, "but decided in the end to keep it within the company."

"All our employees have contributed to Database becoming the UK's leading publisher of computer magazines — so they all deserved to play a part on the big day".



Derek Meakin and 149 fellow diggers attack the green field site of the new headquarters.

10 LINERS



We've let the Features Editor loose on his Amstrad and he's come up with some pleasing graphics demonstrations — actually that's all he's capable of.

In the first, each screen display lasts for 30 seconds, but you can interrupt this at any time by pressing the spacebar and starting again. You can vary the length of this display by changing the variable xx in line 10 which corresponds to the number of seconds.

If you want to see a more colourful but chunky pattern, replace the MODE 1 in line 10 with MODE 0, and alter the 3 in line 50 to 15, or 12 if you don't like the flashing colours.

```
10 REM          fandango
20 REM (c) Computing with the Amstrad
30 MODE 1:DEFINT a-z:xx=30:g=1:x1=RND*64
0:y1=RND*400:x2=x1+RND*25-50:y2=y1+RND*7
5-50:AFTER xx*50 GOSUB 100
40 a=RND*36-25:b=RND*36-25:c=RND*35-25:d
=RND*36-25:WHILE INKEY(47)=-1:MOVE x1,y1
:DRAW x2,y2,g
50 IF g>3 THEN g=1
60 x1=x1+a:IF x1>=640 OR x1<=0 THEN a=-a
:g=g+1
70 y1=y1+b:IF y1>=400 OR y1<=0 THEN b=-b
:g=g+1
80 x2=x2+c:IF x2>=640 OR x2<=0 THEN c=-c
:g=g+1
90 y2=y2+d:IF y2>=400 OR y2<=0 THEN d=-d
:g=g+1
100 WEND:RUN
```

His final offering is a routine to provide either filled or outlined circles. The actual circle routine is based on one of Bresenham's algorithms (his line was used in Roland's PCW graphics article in May), and is contained in three subroutines that you can easily incorporate into your own programs.

Lines 40 and 50 make up the first subroutine and calculate the points on the circle's circumference. Lines 60 and 70 are the other two and create either a filled or an outlined circle dependent on your input.

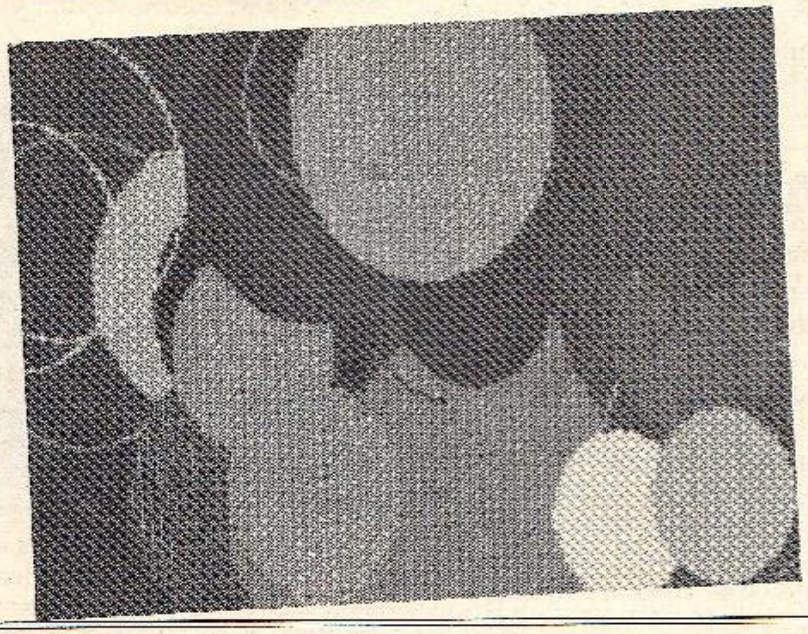
You must pass five parameters (xx,yy,r,c,a) to the subroutines and these dictate in sequence the x,y coordinates, radius, colour and type of circle.

Included in the 10 lines are four separate demonstrations — two of them in line 10. In the first, the x,y co ordinates and type of circle remain the same while the radius and ink numbers decrease. In the second all the parameters are generated by random numbers.

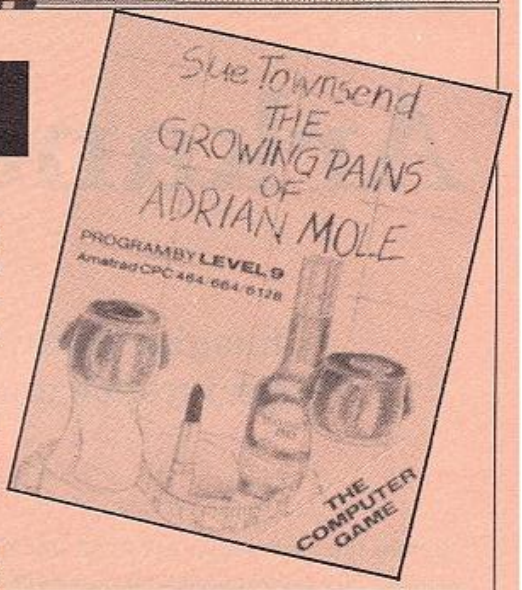
The third and fourth are little surprises to show you just what you can achieve with a little imagination and planning. All the circle parameters are read in from data contained in lines 80-100.

Now see what you can do!

```
10 MODE 0:xx=320:yy=200:r=12:FOR n= 150
TO 20 STEP -10:a=2:b=n-1:GOSUB 40:NEXT:W
HILE INKEY$="":WEND:CLS:FOR loop= 0 TO 4
0:xx=INT(RND*600):yy=INT(RND*350)+20:r=I
NT(RND*100)+20:c=(RND*12)+1:a=INT(RND*2)
+1:GOSUB 40:NEXT:WHILE INKEY$="":WEND
20 MODE 0:FOR loop=0 TO 15:READ xx,yy,r,
c,a:GOSUB 40:NEXT:WHILE INKEY$="":WEND:I
NK 3,6,26:WHILE INKEY$="":WEND:INK 3,6
30 MODE 0:FOR loop=0 TO 22:READ xx,yy,r,
c,a:GOSUB 40:NEXT:WHILE 1:WEND
40 ORIGIN xx,yy:x=0:y=r:d=3-2*y:WHILE x<
y:ON a GOSUB 60,70:IF d<0 THEN d=d+4*x+6
 ELSE d=d+4*(x-y)+10:y=y-1
50 x=x+1:WEND:IF x=y THEN ON a GOSUB 60,
70:RETURN
60 PLOT x,y,c:PLOT y,x:PLOT y,-x:PLOT x,
-y:PLOT -x,-y:PLOT -y,-x:PLOT -y,x:PLOT
-x,y:RETURN
70 MOVE x,y:DRAW x,-y,c:MOVE y,x:DRAW y,
-x:MOVE -x,y:DRAW -x,-y:MOVE -y,x:DRAW -
y,-x:RETURN
80 DATA 320,200,200,4,2,320,200,160,11,2
,320,200,150,5,2,320,200,130,11,2,320,20
0,120,4,2,320,300,80,4,2,240,260,90,4,2,
480,260,90,4,2,240,260,60,8,1,400,260,60
,8,1,260,260,40,8,2,380,260,40,8,2,270,2
60,20,5,2,370,260,20,5,2,320,180,60,3,2,
320,180,60,5,1
90 DATA 600,200,20,13,2,600,200,20,5,1,5
70,210,24,12,2,570,210,24,5,1,530,220,26
,1,2,530,220,26,5,1,480,210,30,13,2,480,
210,30,5,1,430,200,34,12,2,430,200,34,5,
1,370,190,40,1,2,370,190,40,5,1,280,180,
60,13,2,280,180,60,5,1
100 DATA 160,180,80,12,2,160,180,80,5,1,
160,150,40,5,2,150,150,50,12,2,150,200,3
0,4,2,150,200,30,5,1,140,190,20,5,2,100,
180,30,3,2,100,18
```



The growing pains of Adrian Mole



The Growing Pains of Adrian Mole is the second of Sue Townsend's novels to be converted into a game and has once again been written by the Level 9 programming team.

The aim of the game is to make Adrian as popular as possible with everyone. This is done using multiple choice questions, your answers to which affect your score, expressed as a percentage.

I can't really see that this qualifies as an adventure and the packaging information neatly sidesteps the issue by simply referring to it as a game — perhaps the best way to classify it would be as a strategy game.

It's in four parts and it isn't necessary to complete them in succession, though players who do jump a part are given an arbitrary score.

It isn't very easy to go into detail about the game. Anyone who has read the books or seen the television series will know what to expect: Zany, puerile humour.

It is possible to finish all the sections in about half an hour though it may take many hours of play to read all the responses to the various choices open to you.

My one major criticism is that although the programming is up to Level 9's usual standard and the graphics reasonable, the game ending is most unsatisfactory.

For about the last 10 days you are given your score and a rating and then told the game is over. "You have scored 69% and are a middling good poet" is not my idea of a satisfactory ending to the game. If you answer "no" when asked if you want to play the last section again, the computer hangs up.

Overall, I'm not very impressed and found that I soon got bored. Unless you are an Adrian Mole freak, I suggest you put the quite considerable amount of money you are expected to pay for it towards something better.

Presentation	3
Atmosphere	1
Frustration factor	0
Value for money	0
Overall	1

Grange Hill



Grange Hill is the latest TV series to be brought to life on a computer. Most people will know the series, but for those that don't, Grange Hill is a school.

Most of the problems that kids experience at school these days crop up regularly and one of the more serious has been used for this adaptation.

Gonch, the character you play in the game, has had his Walkman confiscated. The last time this happened it was stolen from the staff room. Since then his mum has insisted on seeing it as soon as he gets home.

As she is a somewhat over-bearing character, you decide that your best bet is to return to the school after it closes, enter the staff room and get it back.

One of your friends, Hollo, thinks that this could be good fun and decides that, to a point, he will accompany you in your quest.

The game is graphical: The top half of the screen shows Gonch and any other characters and objects while the bottom is used for messages from the computer. It also contains the drop-down menus you need to do anything other than moving around.

The options available are: Pick up, drop, examine, use, take, give, talk and exit menu. Selecting any of these brings down further menus which are used to select the object and/or person involved in your initial choice. Where an object is concerned, you can only use the menus if you're holding the object or it is readily accessible.

To give an example of these menus in action: A paper plane is encountered early on in the game; selecting pick up and paper plane on the second drop-down menu provokes the response "You can't reach it". The solution is to select the use option, and the history book from the object menu.

The program then asks you to enter text directly with "What do you want to do?" Entering "Stand on book" produces "You can now reach the plane. OK, you've got it".

The menus are awkward to use but they do have the advantage of telling you what objects are before you pick them up, as quite a few aren't easily recognisable on the screen.

You control Gonch, not by the usual verb/noun input, but by using the cursor keys. This does make movement a lot quicker and it is possible, indeed necessary, to jump and speed movement up further.

The lack of a save option is a real nuisance. There are several random deaths awaiting the unwary player and restarting the game from the begining each time you die is very off-putting. I suspect that my opinion of the game would be higher had this option been available.

Overall, a competent arcade adventure. I didn't like it but I prefer "real" adventures anyway, so my opinion is biased.

Presentation	7
Atmosphere	4
Frustration factor	6
Value for money	5
Overall	6

Arkanoid



Imagine Software, cass, joystick and keys

Before the mothership Arkanoid was destroyed by an alien attack, one small spaceship, the Vaus, managed to escape undetected. This escape was only temporary as she has found herself trapped in the void.

The Vaus's only chance of escape is to penetrate the 32 levels of the void. This completed she must destroy the mighty Dimension Changer, reverse time, and resurrect the Arkanoid. This wonderful work of fiction is **Imagine's** way of selling an enhanced version of *Breakout*.

The ageing computer hacks among you will no doubt remember *Breakout* from its days in the arcades. Using bat you repeatedly bounce a ball against a brick wall until you burst your way through. In this 1987 version, substitute spaceship for bat, throw in a few aliens and you have *Arkanoid*, an arcade hit from the *Taito Corporation*.

In order to sell a simple game like *Breakout* to today's increasingly demanding computer gamer you have to spice it up in a big way. **CRL** did it with *Ballbreaker* by using 3D graphics, **Imagine** has introduced variation by hiding special capsules within certain bricks. *Arkanoid* doesn't have the same graphical impact as *Ballbreaker*, but it ranks high in the addictive ratings.

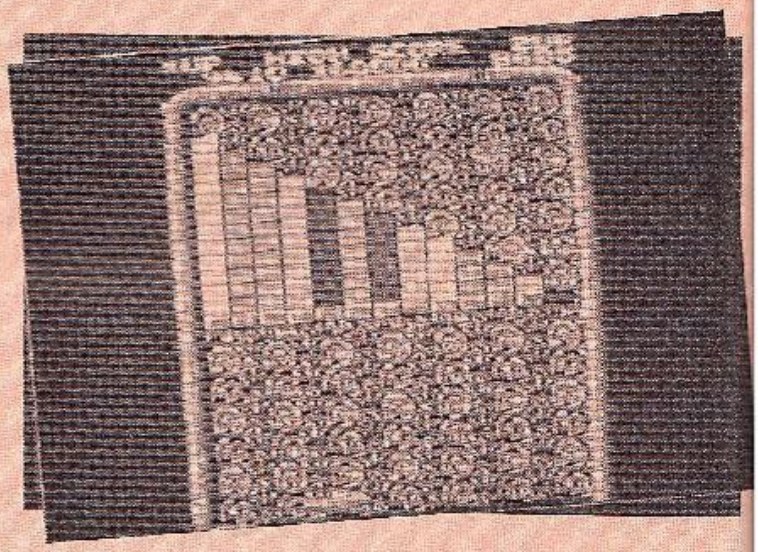
The barriers which you break through are constructed from coloured bricks. Grey ones must be struck twice before they disintegrate, orange ones are indestructable and are used to create simple mazes within the barrier. Great accuracy is needed if you are to angle the ball, or should I say energy bolt, into the maze opening.

Barrier design is extremely varied: screen one sees the traditional rectangular shape. The next barrier uses a wedge design, the lower edge consisting of grey bricks plus a single coloured one. An accurate service shot is very handy if you are to get the ball behind the wedge as soon as possible. Once behind a barrier the ball will ricochet between barrier and back wall causing tremendous damage.

Screen number three is the first of the maze designs. The indestructable orange bricks are used to construct a zig-zag channel up the screen, lined with a single layer of ordinary bricks. I found this screen to be the most difficult one that I tackled.

Coaxing the ball into the opening to the maze requires much skill and patience. The close proximity of the lower edge of the barrier means that a ball which misses the mouth of the channel returns to the bat very quickly.

The fourth barrier consists of two rectangles that are separated from the walls, and each other, by the width of a single brick. Launching the ball into this gap sends it machine



gunning up and around the back of the barrier. This is great fun when the ball is travelling away from you, but just wait till it rounds the corner and starts to head back.

I'm afraid that the fifth screen is as far as I got — due to lack of time, not skill, I hasten to add. This barrier is shaped like a giant space invader, with 90 per cent of the bricks being grey!

The small groups of coloured bricks which could be hiding useful capsules are all encased within the grey ones. This means that you've got to play the biggest part of the screen using your reflexes, and all the luck you can muster.

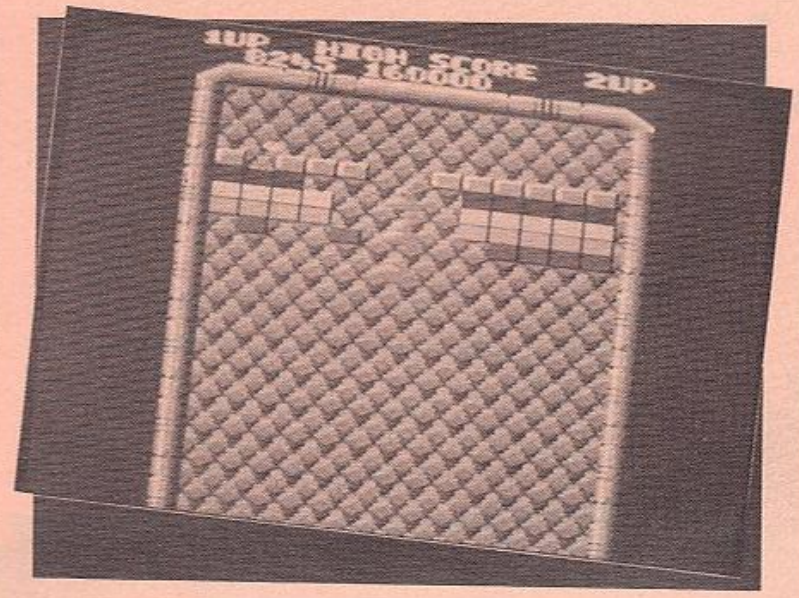
Hyperspatial masonry is not your only problem as you battle through the void — each screen has its complement of alien spacecraft. Up to level five these chaps don't fire at you and can be destroyed with either bat or ball.

Their hindrance value comes in the form of unexpected deflections and quick returns, if they fly near your bat. On the plus side you will sometimes get a bounce off an alien which sends the ball through an awkward little gap.

As I mentioned earlier, certain of the coloured bricks contain capsules. These fall towards the bottom of the screen when the bricks in which they are trapped are destroyed. There are seven types of capsule, each a different colour and embossed with a letter, specially for those of you with green screen monitors.

The capsule is activated by catching it on your bat, which is not as easy as it sounds as the ball is often returning at the same time. The powers endowed by the capsule remain in force until you catch a different capsule, or lose a life.

The first of the capsules is labelled with an S. This slows the rate of travel of the ball, making it that bit easier to hit.



Still on the easier-to-hit theme there is the E capsule which expands the bat to twice its size.

The C capsule comes into its own on the screen where you are trying to fire the ball into a narrow gap. It captuers the ball and holds it until you press the fire key. You can therefore catch the ball, carry it to the ideal position, and then fire it at the barrier.

D stands for disruption. It is a wonderful sight to behold if you can catch a D capsule while the ball is on the far side of the barrier. The ball splits into three pieces and causes incredible damage.

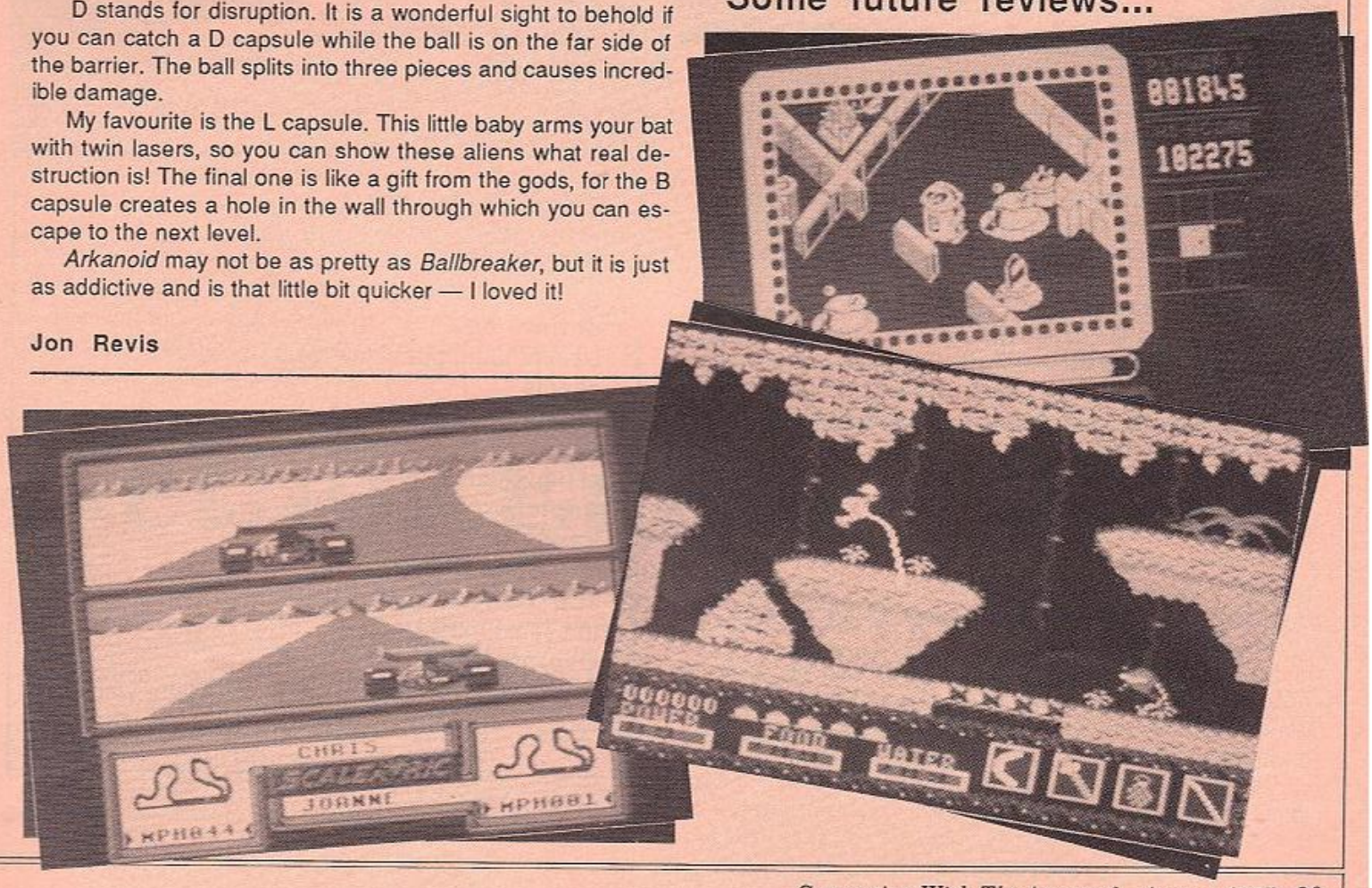
My favourite is the L capsule. This little baby arms your bat with twin lasers, so you can show these aliens what real destruction is! The final one is like a gift from the gods, for the B capsule creates a hole in the wall through which you can escape to the next level.

Arkanoid may not be as pretty as *Ballbreaker*, but it is just as addictive and is that little bit quicker — I loved it!

Jon Revis

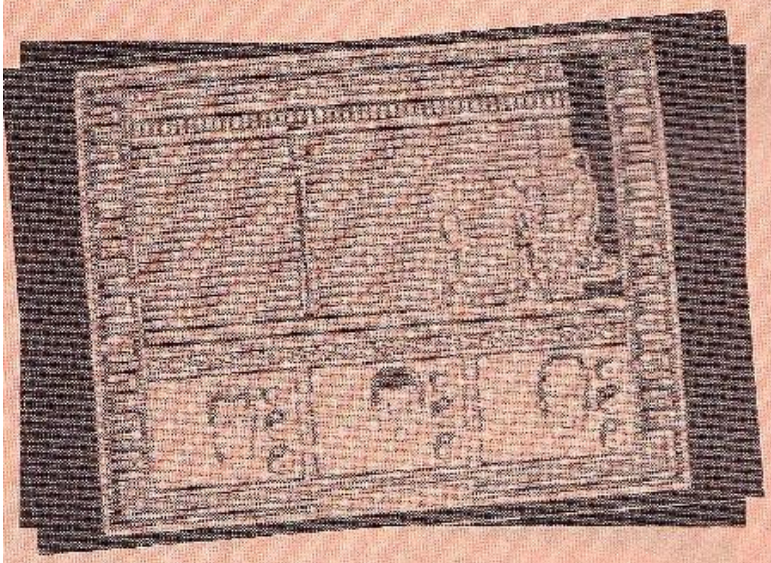
Presentation 85%
Nice simple controls, with a choice of one or two players.
Graphics 88%
Not quite state of the art, but fast and smooth.
Sound 85%
Good sound effects and a dinky title tune.
Playability 85%
Responds well to the keyboard, but joystick control
is dodgy.
Addictive qualities 95%
Go away I haven't finished playing yet!
Value for money 90%
Worth every penny.
Overall 92%
Don't just sit there, go out and buy it!

Some future reviews...



Big Trouble in Little China

Electric Dreams, cass, joystick or keys



The trend in licencing top American films continues. But what happens if the film on which the title is based turns out to be a turkey, in box office terms? And what if the film is brilliant, and the game is dreadful? You can play such guessing games until the cows place last orders and come home from the pub, but the real test is just how well the game plays.

Ignoring the fact that *Big Trouble in Little China* is based on said movie by the amazingly successful John Carpenter, immediate reactions confirm the fact that the game is basically a martial arts variant, where scrolling backgrounds and selectable choice of character control initially hide the fact that here is nothing really new or different.

The scenario: The villainous mandarin Lo Pan must appease a demon in order to secure a mortal body. To do this he must first marry a green-eyed girl, then sacrific her. Kidnapping two green eyed maidens, Gracie Law and Miao Yin, the girlfriends of the heroes Jack Burton and Wang Chi, Lo Pan takes the girls to the underground empire beneath the streets of Chinatown.

Now the film itself is a good old yarn, allowing the viewer to revel in the on-screen action and drama. But when magically transformed into the format of a computer game, all the excitement and mystery is suddenly squeezed out.

Despite the fact that during gameplay you have the option to control any one of three characters (including Egg Shen, the Chinese magician who travels on a mystical floating cloud, so introducing shoot-'em up, hack-'em up and zap-'em up elements into the game) this variation on a theme does little to enhance the game.

I am well known for my use of **Melbourne House's** *Exploding Fist* as a yardstick for judging other games where the martial arts contribute towards a large part of the gameplay. And here the use of extravagantly detailed storylines, combining voyages into the sewers of San Francisco with fanciful attempts at describing the excitement to be found therein, does little to hide the uninspired nature of the exercise.

Victor Laszlo

Presentation 75%
Pleasant four colour hi-res graphics.
Graphics 70%
Adequate animination.
Sound 65%
Decent tune, but virtually non-existent sound effects.
Playability 55%
Non-responsive controls, with no real feeling of participation.
Addictive qualities 40%
Pass.
Value for money 50%
You could see the film three times or more for the same money.
Overall 40%
Unexciting martial arts game, with little going for it.

Shao-lin's Road

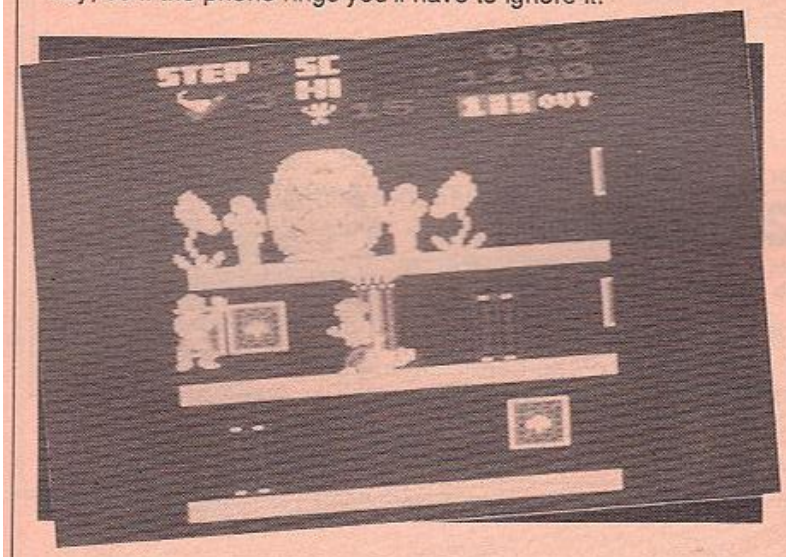
The Edge
cass, joystick

Shao-lin's Road (sounds like a soap opera to me) is the long-awaited follow up to *Yie Ar Kung Fu.* Lee has now mastered the secret martial art of Chin's Shao-lin and is ready to set out on the Shao-lin's road to freedom. Little does he know that the evil triad gangs are out to stop him.

The instructions supplied with the program are sparse.

You are told that the game can be played using keyboard or joystick, and that the keys are user definable. While on the title screen I pressed every key on the keyboard in an attempt to find which one would allow me to define my own keys — nothing worked so I used the joystick.

On top of all this there is no high score table on which to record your historic victories. The final omission is a pause key, so if the phone rings you'll have to ignore it.



The playing area is spread over two screens, the display scrolling to either side as you near the edge. Every screen is built on three levels, and Lee and the triad thugs can jump between any of these at will.

Assailants run at Lee from both directions and points are awarded for kicking them to death. There are bonus points to be had for kicking the birthday cakes which cross the screen from time to time. Most screens appear to have numerous minions to slay plus a secret weapon: On screen two this takes the shape of a 15 stone oriental punk — heavy!

One thing has me puzzled; if Lee has mastered this secret martial art, why is it that he can only remember one of the multitude of kicks and punches he used so skillfully in *Yie Ar1*? Lee's movements are limited to left/right, up/down and kick. In an attempt to maintain your interest Lee can acquire special powers by catching the spheres which occasionally emerge from a dead opponent.

Lee is provided with five lives, and loses one when he is hit four times. Gangsters can drop on Lee's head, or jump up and butt him from below. These manoeuvres are all acceptable and count as hits, but if Lee attempts the same moves the points are awarded to the bad guys — not fair!

Shao-lin's Road is a run-of-the-mill ladders and levels type game with a kung fu theme — nothing more.

Nev Astley

Presentation 65%
Sparse instructions and no user-defined keys.
Graphics 82%
Large, colourful and well animated.
Sound 80%
Sound effects without the tune would have been a welcome option.
Playability 75%
The bad guys have an unfair advantage.
Addictive qualities 70%
Bring back Yie Ar Kung Fu!
Value for money 75%
Try before you buy.
Overall 78%
The death knell for kung fu games.

Nether Earth

Argus, cass, joystick & keys

Since the Insignians emerged from the Earth's crust, the whole of humanity has known pain and suffering. In only a few short years they have taken complete control of the Earth, and only scattered remote settlements and bands of armed troops are left to fight the threat.

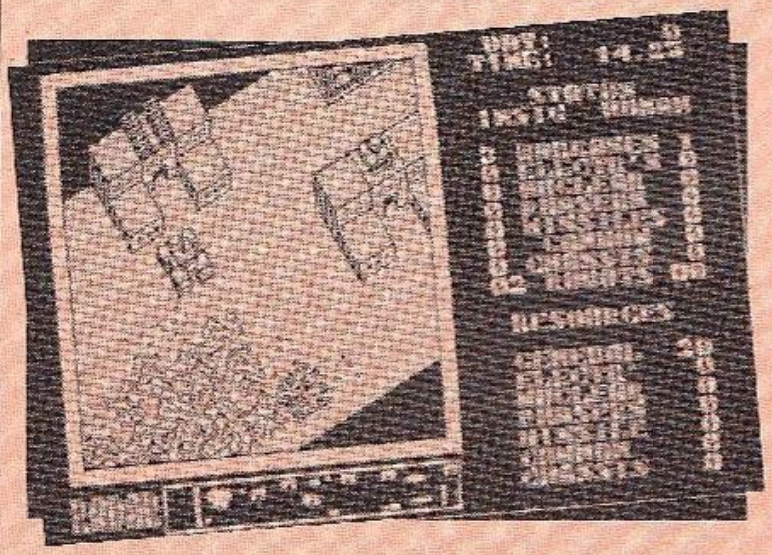
All other humans have been captured and are used as slave labour in the Insignian weapon factories.

Captain Furgus McCaffery has formalised a final plan to overthrow the oppressors. The only way to fight these machines is by using their own forces against them, so in a desperate battle the human liberation army took and held an Insignian base and began creating a new army.

The plan is to create small unarmed machines to occupy factories, liberate the human workforce and turn the factories into arsenals for the rebels.

You control a small hovering craft which can land on any of the human robots and give them instructions.

But before any robot can be controlled it must first be



created at the home war base. The robot is built in sections which take a specific amount of materials, depending on its power.

The robot requires a base to move it around the terrain, and these come in three types. The Bi-pod is the cheapest form of base and is very rugged but moves slowly, especially over rough terrain and cannot climb hills. A track system takes twice the resource units to build but is considerably faster over most terrain — hills are the exception. By far the best system of movement is the Anti-grav.

Four types of weapons are available, and up to three may be used on each robot built.

To complete the game you must travel 512 miles, destroying all three Insignian war bases. Luckily a save-game option is included, though a paper and pen may be needed to note position and type of your own robots at each save.

Nether Earth is both addictive and fun to play; very few people should be disappointed.

Tony Clarke

Presentation 87%
Keys can be redefined. Save-game option makes a nice touch.
Graphics 91%
Detailed graphics that are recognisable and move smoothly.
Sound 85%
An unusual title tune, a few white noise and laser sounds in the main game.
Addictive quality 90%
A little slow to start, but worth the effort.
Value for money 76%
A little pricey when more games are dropping to the $20 mark.
Overall 88%
One of the better arcade wargames.

US Gold, cass, joystick

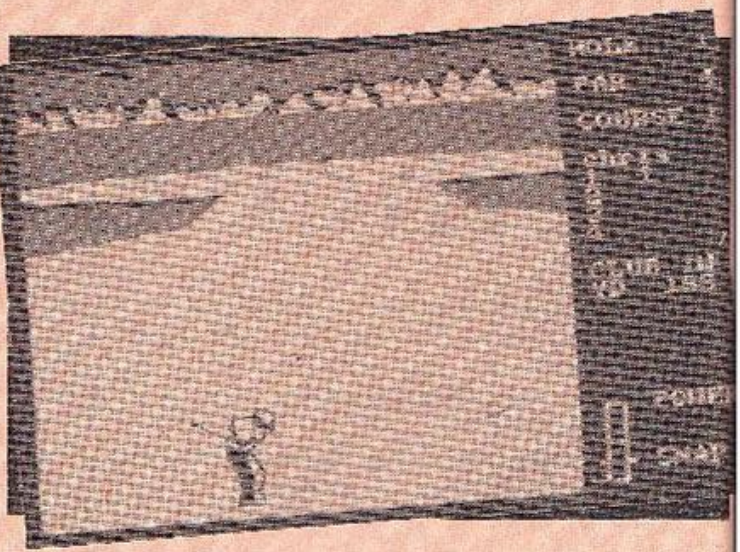
Leader Board

On its introduction in the summer of '86, *Leader Board* for the Commodore 64 immediately set the standard that all computer golf games had to match. As yet, no other title has managed to equal it in terms of marvellous playability and sheer realism.

The main problem with all computer golf games in the past was how to accurately convey the feel of a real golf game, with a joystick instead of a club. *Leader Board* achieves this by giving you a 3-D perspective over each hole, with a choice of four different courses.

On each consecutive shot, as the ball lands, the green actually draws itself in outline form, then quickly fills in with colour to depict your new position on the course.

To make the gameplay as real and detailed as possible, *Leader Board* lets you select your own skill level, the number of holes (from 18 to 72), choose your course, select a club from woods, irons and pitching wedges and control the power and "snap" of your shot.



Continues page 37

By careful control of the joystick it is perfectly possible to control the slice or hook of your shot: Practicing on the driving range is the best way to learn to hit the ball straight.

One special feature available only on the Professional Level is the presence of the Wind Factor. Here you must allow for the prevailing winds to target your shot properly, otherwise you tend to end up in the drink.

And just when you thought you might get away with a little cheating on the side, *Leader Board* penalises you for sending your ball out of bounds or ending up in the water.

If you land safely on the green, and within 64 feet of the hole, the computer removes the flag for you and forces the putter upon you to make your, hopefully, final shot.

I found *Leader Board* an extremely challenging program, providing just the right balance of gameplay, where practice eventually builds up your confidence enough to go for the Amateur and Professional levels.

I just wish that the Wind Factor had been a bit more predictable. Maybe then I would have ended up with a slightly better score rating. Bye for now, and back to the green...

Victor Laszlo

Presentation 90%
Good attention to detail and easy selection of options.
Graphics 87%
Simple, clean graphics with very realistic perspective.
Sound 70%
Not much in the way of FX here, but at least no silly music to distract you!
Playability 90%
The best feel and control of any computer golf game.
Addictive qualities 90%
How can you complain when you keep trying to better your rating?
Value for money 90%
Worth it for the relaxation.
Overall 90%
The best golf game yet

Pulsator

Martech, cass, joystick and keys

The maze has always featured in the mainstream of computer games; even classics such as *The Eidolon* and *Jet Set Willy* are simply maze games. *Pulsator* is yet another, but with an added twist.

It follows the traditional maze game format, with the player moving a character around a complex covering several screens, which are viewed from above in two dimensions. The game is spread over five levels — with 49 screens on each.

As you progress through the levels the aliens become more intelligent, more accurate and reduce the energy level of your Pulsator much more quickly.

And the deeper into the game you go, the more you have to remember.

Not every object in the maze is there to destroy you. Picking up oil will recharge the Pulsator to full power, and collecting three oil cans gives you an extra life. Transport

continues over page

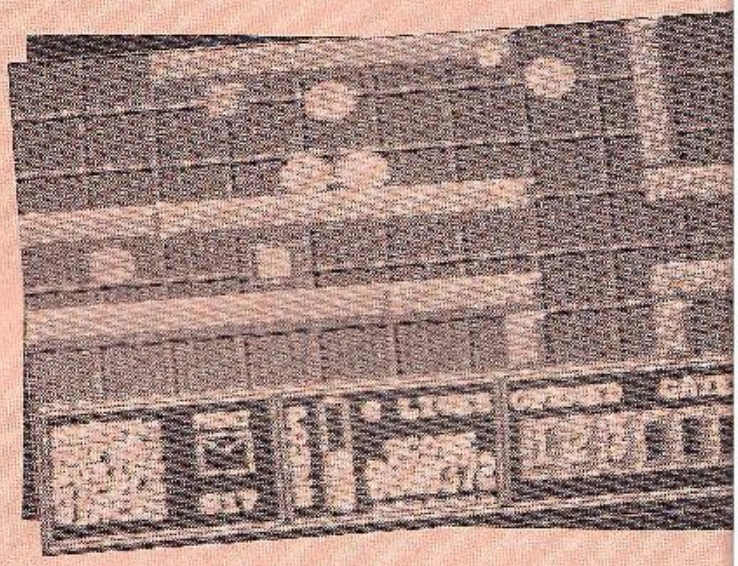
STOP PRESS: This photo should have appeared on page 38 and vice versa



Presentation 67%
A demo mode gives the player a taste of things to come, but the lack of a redefine key option spoils the game's polish.
Graphics 58%
Little more than pulsing spheres and other geometric shapes.
Sound 43%
A few bleeps, whistles and white noise explosions.
Playability 69%
The game control is a little sensitive, but otherwise simple to master.
Addictive qualities 51%
A good deal of strategy, but not much scope to keep the player going.
Value for money 55%
Would have been better priced in the $5 range.
Overall 55%
A good average game, though not my first choice

Ghost Hunter

Code Masters, cass, joystick and keys



There is a substantial prize awaiting anyone who can rid Nightmare Mansion of its ghostly occupants. Three days ago your brother took up this challenge and he has not been seen since. Driven on by brotherly love (or is it the money?) you entor the mansion in search of Chuck.

The aim of the game is to make your way to the heart of the mansion, free Chuck, and escape alive. To do this you need to activate numerous lifts and escalators, giving you accoss to new areas of the mansion.

The game can be controlled with user-defined keys or a joystick. The program allows you to move Hunk Studbuckle, the hero, or the sights of your "subcompact anti-matter phantom splatterer".

Moving the joystick enables Hunk to run, jump and climb. Holding down the fire button and moving the stick sprays gunfire around the screen, vapourising any ghoul it touches. The gunfire is excellently drawn and accompanied by realistic sound effects.

Two thirds of the screen is devoted to the playing window, the remainder displays a picture of yourself, looking like Rambo, and your score. It also contains two very important indicators: The first of these is the Macho Energy meter — if this falls to zero you die. In this game you only have one life so death is a pretty serious matter. The second indicator is the Terrometer. Looking like a speedometer it has a pointer which swings round in response to spectral activity within the room.

Your energy reserves can be replenished by consuming beakers of elixir. Drinking one of these is accompained by a digitised yell of Ghost Hunters from the depths of your keyboard.

The mansion's rotting residents are a delight to behold. Rising through the floor, the skeletons and ghouls walk with traditional stiff-limbed movements. A mummy emerges from its sarcophagus with an eerie creak.

Ghost Hunters is a complex ladders and levels game with a fair amount of zapping thrown in for good luck.

Steve Brook

Presentation 84%
A choice of user-defined keys of joystick.
Graphics 87%
Detailed and well-animated, but a bit on the small side.
Sound 88%
The software speech is very well done.
Playability 82%
Switching between character movement and gun movement takes a bit of getting used to.
Addictive Qualities 84%
Life is sometimes a little too short to make any real progress.
Value for money 90%
A genuine bargain.
Overall 86%
Horrific ladders and levels game.

---

from previous page

ers, dotted around the maze, move you from one place to another, hopefully putting you in a better position to complete the level.

Before the Pulsator can complete a level, it must first find a key to the prison on each level and set free a Pulsy — pulsies are images of your own Pulsator, but with sad mouths.

So far the whole game seems easy, but your task is far from simple. Throughout the maze are electronic doors, operated when the Pulsator passes over a pressure pad with the same number as the door.

The game plays very well and includes a fair amount of brain work along with joystick skills, though once a section is completed it is easy to finish it a second time. Screen change is very fast, as is the rate at which the aliens move, which all goes to make a furiously-paced game.

It is a pity that only five levels are included, as once the game has been completed there is no longer a challenge. A screen designer would have been a nice touch but was not included. Overall, *Pulsator* is an average game in all respects, but it is a well-polished piece of programming and worth a look.

Tony Clarke

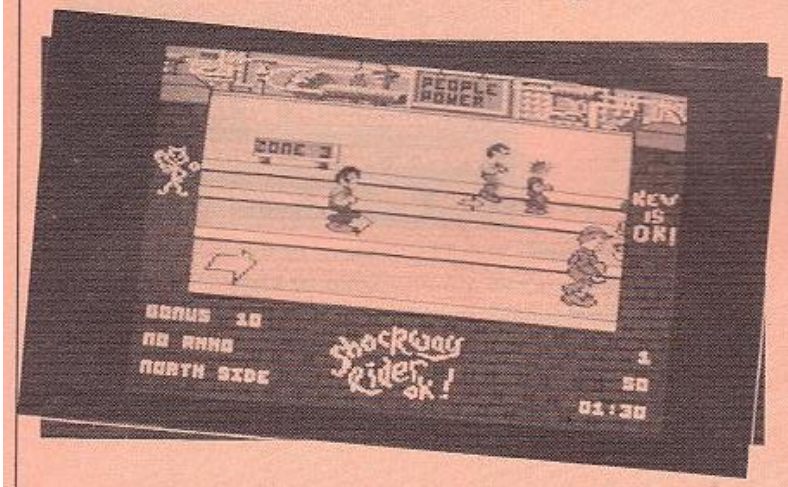
SHOCKWAY RIDER

Disc, joystick or keys

FTL's first offering on the CPC was Light Force, which topped the Amtix charts for some time. Now after a long wait comes *Shockway Rider*, the company's latest horizontal shoot-em-up.

Shockway Riders are the toughest members of the 21st century street gangs, and all want to complete the ultimate goal of going "Full Circle", a trip which takes them around the entire Megacity on the triple-speed walkways.

Each walkway — there are eight — consists of three tracks moving at different speeds, the slowest being the inner track.



You must avoid all objects and people, while moving fast enough to cover 12 zone gates (which come at one screen intervals) in under two minutes.

Most of the hazards found on the walkways are passive, such as railings which slice off the rider's head, and the policeman's ball which flies along any track which has been ridden too long.

The local residents — old ladies and punks — also ride the walkways, but they don't take to you speeding either, and will dispose of you neatly if you get in their way.

Not everything found on the walkways is harmful: Gold and money can also be found, along with an assortment of weapons from bricks to rocks which will kill a rider who is directly in front of or behind you.

Throwing ammunition sideways only harms pedestrians, but you gain bonus points if it scores a direct hit on any of the many targets that litter the walkways on the later levels.

The music proved to be very good, fitting in well with the plot, but tended to get repetitive after very few games. Music and sound FX can be toggled between, though the effects were nothing more than unimaginative beeps and whistles.

I think FTL seems to have let itself down a little with this one. The game was just too sparse for my liking, even though a lot was happening on the screen.

Shockway Rider can be an enjoyable game, but is only going to appeal to certain people — take a long look before buying.

Anthony Clarke

Presentation 81%
Good on screen instructions, easily handled keyboard layout.
Graphics 72%
Great music, limited spot FX.
Playability 73%
Practice mode makes the game much more playable.
Addictive qualities 59%
Only played for half an hour at a time, but an enjoyable half hour.
Value for money 61%
Less than most games, but would have been better as a budget line.
Overall 63%
Not FTL's best, but still one worth a look. A little sparse, but Teddy boy gang members look quite cool.

future review



Amtips
Anthony Clarke



Knight Tyme

(Mastertronic)

Here's some useful snippets from Peter White of Sheffield. To order some people around you will need a valid ID card. To get one you will need to do the following:

Tell Derby IV to help; pick up a blank ID card; get camera and film; unwear Cloak of Invisibility; give camera and film to S3 E3 or Klink; tell him to help; take photo; get glue from S3 E3.

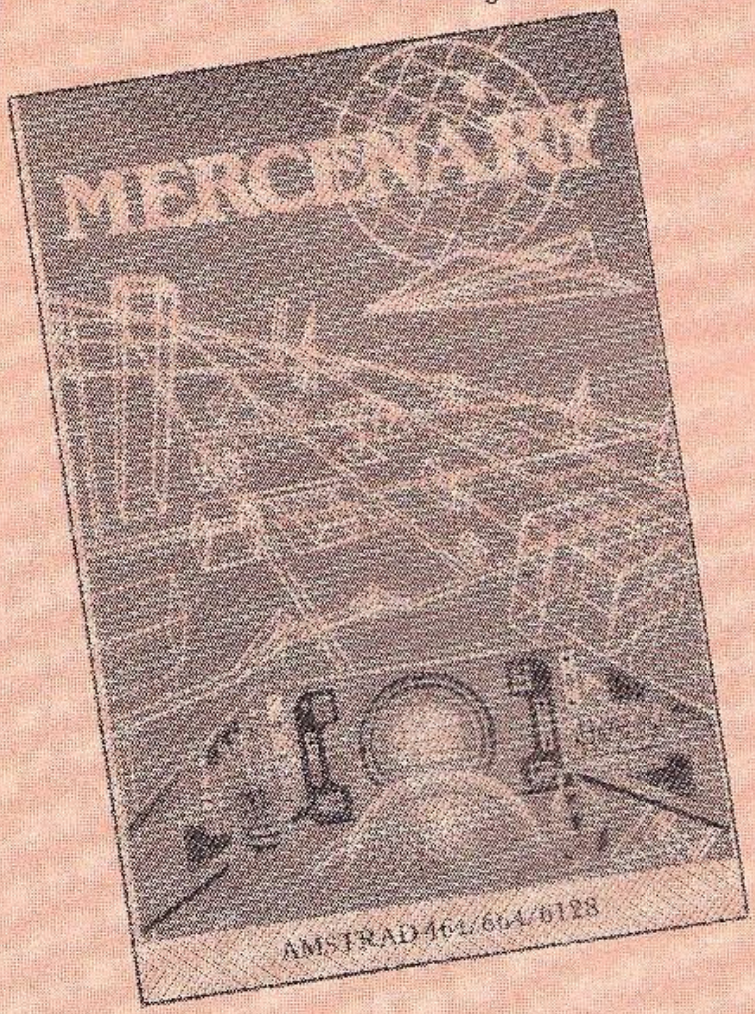
Sharon likes chocolate, and you can get a chocolate heart from Derby IV. To get the things from on top of the doors, drop the advert and use it to help you up.

MERCENARY

(Novagen)

Amtips is our monthly section that helps you to get more out of computer games. Just when you thought you had reached a hopeless situation, along comes a bright spark with all the answers, and a POKE or two to get you out of a jam.

MERCENARY is in the air! How many of you have been hooked by it? There must be quite a few. I've been playing it non-stop since I bought it some time ago.



Not that this is the only good game to have surfaced recently. It really has been a bumper month on the games scene. Remember that the more people that buy games the more profitable the Amstrad market will become and the better the games will be.

Just to help out those people who are still struggling to get going in the first few stages of the game, here are the locations of the most of the lifts into the underground complex.

09 06 Several keys should be found here.
03 00
09 05
81 35 A long way out in the desert, but well worth the trip.
03 15 Under this one is the interstellar ship, but the pass required is also in the same place – Hmmm!
**** Known only as the star base, it is Targ Arctic patrol station.

If you wish to get into the paylars colony craft then you need either a power amp for your dominion dart, a cheese-boy is that a Kraft, or concord.

The fastest ship is the cheese, which can also be carried in the pocket. Once on the colony craft you will need a key to continue into the rest of the craft. The web is really a cheat key that will open all doors, but you will need the kitchen sink to pick up anything that heavy.

Escape from singe's castle

To help you out on the screens that require precise movement, here is a full run-down of the moves.

Throne Room

Push left, right to dodge the hand and the fire. Press fire to dispatch the hand, push forward, left then pull down to dodge the fire three times.

A circle of energy will then begin to rotate around the orb, press fire each time to jump over it, but remember that as the circle speeds up you must jump a little sooner than the last time.

Push right to dodge the fire, and then use the same move as before to leap the second circle of fire, but this time you can let it get a little closer before jumping. Finally push right to sit in the chair and finish the game.

Doom Dungeon

First push forward to avoid the flames. Press fire to kill the snakes. Push right to avoid the fire once again. Press fire to kill the spider. Move left to dodge the lightning.

Press fire to kill the second spider. Pull back on the joystick to dodge the flames once more. Wait until the bench at the right of the screen is burnt, but then push left of you die in the flames.

Finally, hold the joystick to the right, dodge the spider and flames to leave by a hole in the wall.

Boulder Alley

There are very few tips for this one as it is more a game of timing. Try to stay near the boulder, but don't fall under it.

When a small boulder passes just in front of you, run forward, it will then be behind you. Let yourself move back to the big boulder and wait for the next small boulder.

Remember to avoid the pits in the floor by pressing forwards to leap them.

Mystic Mosaic

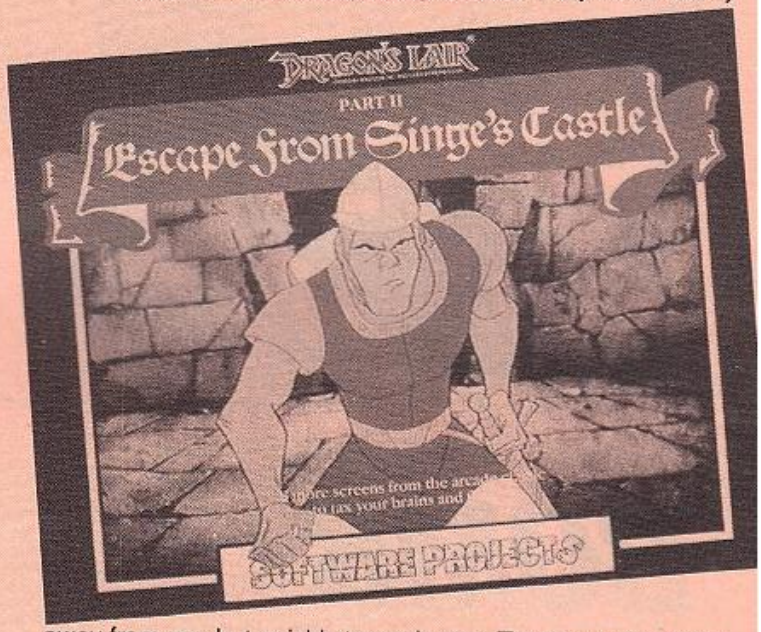
The flashing squares at the beginning of the game will give you the correct path to follow. Try to remember where they are and move from one to the next by the quickest route. Just because they are safe at one specific time, don't think they will always be safe.

If the bat gets too close then dispatch him quickly, but don't forget to keep moving to the next square. The door at the back is a trap — leave by the door on the far left.

The River Caves

Most people agree this is the hardest screen to complete. There are only two sequences of movement required to finish the rapids — try to remember them and work out, on the first screen, which to use.

The whirlpools are a little more difficult. They move slowly



away from you but quickly towards you. Try to remain behind them so that they are moving away from you.

If you find yourself in a no-win situation, try pulling back on the joystick to slow the boat down, and hopefully the whirlpool will move right past you.

Magical flying Horse

This is my favourite screen. The best idea is to simply weave back and forth each time you see a wall. This way you move away from the wall as soon as it appears. Don't worry about hitting the rocks and ice, even if you were to hit every single one, the reduction in speed would be so slight that most people won't notice it.

Dungeons of the Lizard King

It is important to remember that this level is made of only six rooms, arranged in a circle. Once you have visited every room — you need not visit both the top and bottom levels — a pot of gold and your sword will appear. Pick them up and then face the Lizard King. Hold down the fire button and the King will die.

The left and right doors on the back wall will take you to the other side of the set of six rooms. The centre door moves you between the top and bottom levels of a screen. If the rat comes too close Dirk will be bitten and fall about shivering.



Mud Monsters

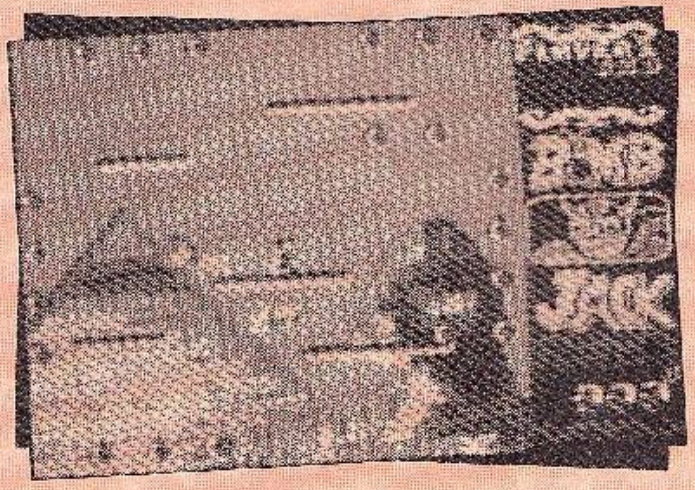
This is the final screen. Collect the ball that is spinning on the left-hand crater, then move Dirk to the far left of the screen and hold the up key until he reaches a point where the mud monster at the top right of the screen starts to throw mud. Press the left key to move back down the wall to a safe point.

When the Mud Monster stops throwing mud, move, using the up key, to a point where Dirk's back foot is touching the lowest spot of mud. Now hold the right button and he will run across the mud bridge and avoid the clutches of the Mud Monster that appears behind him.

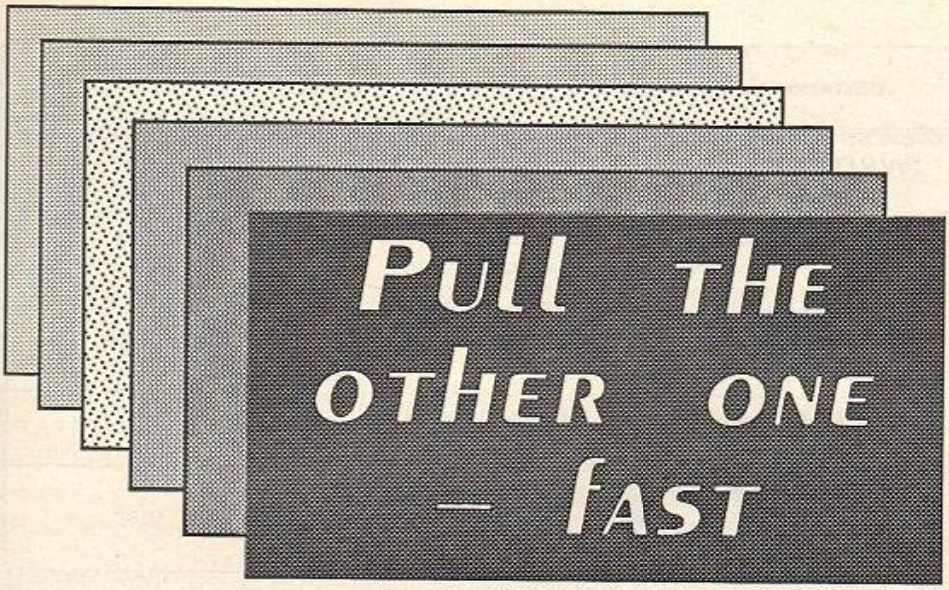
The problem now is how to cross the gap where hot mud flys into the air. Move to the cliff edge, where the cliff juts out most. Wait until the mist is high and then press the fire button. Dirk should now sail through the air and end up near the bottle.

BOMB JACK

(Elite)



```
110 ' BOMB JACK - POKES BY CY BOOKER
TAPE OR DISC
120 MEMORY 5999
130 MODE 0: BORDER 0: FOR I=0 TO 15:
READ A:INK I,A,A:NEXT I
140 LOAD "!BJSCREEN.BIN", 49152
150 LOAD "!BJCODE.BIN",6000
160 POKE &218F,1: REM ALL PILLS, EXTR
AS AND BONUSES APPEAR
170 POKE &1800,4: REM NUMBER OF LIVES
PLAYER 1
180 POKE &1811,4: REM NUMBER OF LIVES
PLAYER 2
190 POKE &19FD,0: REM INFINITE LIVES
200 POKE &1850,24: REM BOMBS TO COLLE
CT PER SCREEN
210 POKE &1F1F,&C3: REM INVULNERABILI
TY
220 POKE &2232,0: REM BIRD ONLY APPEA
RS
230 POKE &40E1,&C9: REM WATCH SCREEN
BEING DRAWN
240 CALL 6000
250
260 DATA 1,0,26,8,24,13,11,6,15,16,5,
2,6,3,20,10
```



A CPC6128 utility by Lawrence Rowe

Pull-down menus are definitely "in" and no program of any standing can afford to be without them. For best effect the pull-down needs to be fast and smooth and this is not so easy on the Amstrad, with its slow printing on screen.

One obvious solution on the CPC6128 is to store the menus in the spare RAM and to use the Bank Manager routines to copy them to the screen when required. Unfortunately Bank Manager can really only move whole screens which makes it unhelpful for single menus.

Another difficulty arises from the fact that the copying is done in one continuous run from start to finish, and because of the way the screen is mapped to memory this causes the infamous venetian blind appearance.

We need a set of routines that can move the contents of a selected window between the screen

```
1 REM Pull Down Menus
2 REM By Lawrence Rowe
3 REM (c) Computing with the Amstr
ad
4 REM
10 MEMORY &7FFF:add=&8000
20 FOR i=0 TO 15:t=0:READ a$,tn
30 FOR j=1 TO LEN(a$) STEP 2:a=VAL
("&"+MID$(a$,j,2)):POKE add,a:add=
add+1:t=t+a:NEXT
40 IF t<>tn THEN PRINT "ERROR in l
ine";1000+10*i:STOP ELSE PRINT "Li
ne";1000+10*i;"OK"
50 NEXT
60 SAVE "PULLMMC",B,&8000,&24B
70 END
1000 DATA 212000194E23462379B02854
E5606919E54E2346606919444DE1712370
E118E4,2929
1010 DATA 6100640070008C008F009200
95009800B100B400B700D300EF00F400F8
00FD00,2774
1020 DATA 1101210126013301430140C01
5C0171017B017E01B001BB01C101E601E9
010000,1770
1030 DATA 018C00218800C3D1BC218AB9
B7200C3AF901220900470E00C3B7BB1149
CBD521,3025
1040 DATA 8AB9220900C300B900000000
9A00C3B000C3EE00C33F01C33B01575249
5445CD,2818
1050 DATA 53574150CD50555348CD5055
4C4CCD00CDBB0032F90121790118B7B728
BDF5DD,3499
1060 DATA 7E01B72024DD7E00FE02381D
C602FE08301732F801F1FE023E00D8209E
DD7E03,3213
1070 DATA B72006DD7E02FE08D8114DCB
1890CDC001C5E5CDA101C511FB01EDB0CD
B90128,4014
1080 DATA 052100C0EDB0C1D16B62CBBC
E5C5EDB0CDB901280821004011 00C0EDB0
C1D1D5,4253
1090 DATA 21FB01EDB0CDB90128051100
40EDB0E1CBFCCDEE01C13D20B8185616FF
180216,3657
1100 DATA 00D5CDC001D1DD7201C5E5CD
A1015D54CBBADDCB01462801EBEDB0CDB9
01280F,4145
1110 DATA 2100C0110040DDCB01462801
EBEDB0E1CDEE01C13D20D21818F53AF801
CD9201,3607
1120 DATA 3E4032C6B7F1F3D9E1CD8DB9
3EC032C6B7AFF3D921D5B85677F6C0ED79
7AD9FB,5355
1130 DATA C9E5CBFC09E56069C1380301
0000ED43F901B7ED42444DE1C9ED4BF901
0C0DC9,3976
1140 DATA CDBB00CDB4BBCD69BB7A943C
4FCD11BC792803300287874F7B953C8787
870600,3586
1150 DATA C5F5CD1ABC3AF801CD9201F1
C1C901000809D00150C009C900000000000
000000,2864
```

Program 1

and banks 2 to 5 with a nice, even pull-down.

Before a menu can be pulled from one of the banks of spare RAM it must obviously first be put in there, and all this cunning is going to be wasted if the only way to do that is first print it out on the screen in full view.

So it would be handy if there were a way of printing, or plotting, or whatever, directly into the spare RAM without ever going on the screen. Of course you wouldn't be able to see what you were printing at the time, but you would have tried it out on the screen beforehand to make sure that your program was properly debugged.

PULLM is a suite of routines for the CPC 6128 that provides all these facilities. Program 1 enters the machine code, checks that it is correct and, when it is, saves it to disc as PULLMMC.BIN. Program 2 is the program which will load and initialise the RSXs. Type it in and save it on the disc as PULLMRSX.

The code is completely relocatable, so it can beloaded and run regardless of the value of HIMEM as long as it is not below &8000. After it has been called, the relocating routine is no longer needed and the space is reclaimed. You will now have a set of four RSXs and one CALL.

The first three RSXs, ¦PUSHM, ¦PULLM and ¦SWAPM, move data between the spare RAM and the screen. They all have the same syntax– ¦PUSHM, [<optional stream expression>,]<screen expression>.

If the stream is not specified the default value is Ø. As their names suggest, ¦PUSHM will push data from the front screen

command	action
¦WRITEM,n,s	Sends all print and graphics output to window *n* on rear screen *s*.
CALL closem	Restores print and graphics output to normal screen.
¦PUSHM,n,s	Pushes (copies) contents of window *n* from normal screen to rear screen *s*.
¦PULLM,n,s	Pulls (copies) contents of window *n* from rear screen *s* to normal screen.
¦SWAPM,n,s	Swaps contents of window *n* between normal screen and rear screen *s*.

The PULLM commands

the specified rear screen, ¦PULLM will pull data from the rear to the front and ¦SWAPM will swap the contents of the front and rear screens.

Thus, ¦PUSHM,3 pushes the contents of window Ø to screen 3, ¦PULLM,2,4 pulls the contents of window 2 from screen 4 to the front screen and ¦SWAPM,1,5, swaps the contents of window 1 between the front screen and screen 5.

As with Bank Manager you need to be careful of the screen hardware-roll effect. You should arrange that all PUSHing, PULLing and SWAPping is done with the screen set to the same hardware position.

The remaining RSX is ¦WRITEM, again with the same syntax, ¦WRITEM,[<optional stream expression>,]<screen expression>. This is the one that allows you to write invisibly on to one of the rear screens. It can be followed by any of the whole range of PRINT, PLOT and DRAW commands and you use them in the normal way.

When the program is run you will see nothing and all the printing and graphics will be sent to the rear screen of your choice. So ¦WRITEM,3,1 allows you to print directly to window 3 on screen 1.

But there is a price to pay for all this versatility. There are so many possible commands that it is hardly practicable to modify each one separately.

Instead the trick is to intercept LOW JUMP, which is a call to address &ØØØ8 through which the Basic interpreter gains access to the firmware routines. Then we can divert all the calls through our own switching routine to bring the extra banks of

```
1 REM Pull Down Menus - RSX
2 REM By Lawrence Rowe
3 REM (c) Computing with the Amstr
ad
4 REM
10 h=HIMEM-&24C:IF h<&8000 THEN PR
INT "Too low":END ELSE MEMORY h-1:
LOAD "PULLMMC",h
20 CALL h:h=h+&68:MEMORY h:closem=
h+1
```

Program 2

RAM in and out at the right times.

That deals with all the text and graphics routines, but it can make life difficult for many of the other commands that don't want to use the screen at all. You'll find that many of them won't work until ¦WRITEM has been switched out again using CALL *closem* which is described later.

For example –

```
¦WRITEM,1,3: LOCATE 3,3: PRINT "m":
CALL closem: WHILE INKEY$= "": WEND
```

will work perfectly, whereas:

```
¦WRITEM,1,3: LOCATE 3,3: PRINT "m":
WHILE INKEY$="": WEND: CALL closem
```

will cause the computer to hang.

In fact calling any RSX while ¦WRITEM is in operation is likely to crash the system. Even calling the ¦WRITEM is in operation is likely to crash the system. Even calling the ¦WRITEM command itself,

continues page 62

```
1 REM Pull Down Menus - DEMONSTRAT
ION
2 REM By Lawrence Rowe
3 REM (c) Computing with the Amstr
ad
4 REM
10 BORDER 26:INK 0,26:INK 1,0:MODE
 2:closem=HIMEM+1
20 WINDOW #3,30,51,1,11:WINDOW #4,
60,79,3,12
30 CLS #4:LOCATE 10,8:PRINT "This
is a demonstration of the ¦WRITEM
and ¦SWAPM commands"
40 PRINT:PRINT:PRINT TAB(21)"as us
ed to produce pull-down menus,"
50 PRINT:PRINT:PRINT TAB(20)"as we
ll as a simple animation routine"
60 PLOT 0,0:DRAWR 0,398:DRAWR 638,
0:DRAWR 0,-398:DRAWR -638,0
70 ¦WRITEM,3,2:CLS:PLOT 407,398:DR
AWR -175,0:DRAWR 0,-170:DRAWR 170,
0:DRAWR 0,170
80 DRAWR 1,-1:DRAWR 0,-170:DRAWR -
170,0:DRAWR 1,-1:DRAWR 170,0:DRAWR
 0,170:DRAWR 1,-1:DRAWR 0,-170:DRA
WR -170,0:DRAWR 1,-1:DRAWR 170
,0:DRAWR 0,170
90 LOCATE 3,2:PRINT "1.  PULLM"
100 LOCATE 3,4:PRINT "2.  SWAPM"
110 LOCATE 3,6:PRINT "3.  ANIMATIO
N"
120 LOCATE 3,8:PRINT "4.  QUIT"
130 LOCATE 3,10:PRINT "Select Opti
on":CALL closem
140 ¦WRITEM,4,3:PRINT CHR$(24);:CL
S:LOCATE 7,6:PRINT "SCREEN 3";CHR$
(24):CALL closem
150 LOCATE 27,17:PRINT CHR$(24);"
Press any key to start ";CHR$(24)
160 WHILE INKEY$="":WEND:LOCATE 27
,17:PRINT SPACE$(24);:¦SWAPM,3,2
170 i$="":WHILE i$="":i$=INKEY$:WE
ND:IF i$<"1" OR i$>"4" THEN 170
180 ON VAL(i$) GOTO 190,210,230,22
0
190 ¦SWAPM,3,2:¦PULLM,4,3:LOCATE 2
7,17:PRINT ">Press any key to star
t<"
200 WHILE INKEY$="":WEND:GOTO 30
210 ¦SWAPM,3,2:GOTO 150
220 END
230 BORDER 1:INK 0,1:INK 1,24:MODE
 1:WINDOW #2,11,30,5,21:ORIGIN 320
,184
240 PRINT CHR$(24);:CLS:LOCATE 2,2
4:PRINT "PLEASE WAIT.  DRAWING ON
FIVE SCREENS";CHR$(24):CLS #2
250 DEG:FOR s=1 TO 5:IF s>1 THEN ¦
WRITEM,s:CLS
260 LOCATE #2,7,2:PRINT #2,"SCREEN
";s
270 PLOT 100,0,s:FOR i=10 TO 360 S
TEP 10:DRAW 100*COS(i),100*SIN(i):
NEXT
280 FOR i=12*s-12 TO 360 STEP 60:P
LOT 0,0:DRAW 100*COS(i),100*SIN(i)
:NEXT
290 CALL closem:NEXT
300 LOCATE 2,24:PAPER 1:PEN 3:PRIN
T "   SWAPPING.  PRESS ANY KEY TO
STOP  ";:PAPER 0:PEN 1
310 WHILE INKEY$="":FOR i=2 TO 5:¦
SWAPM,2,i:NEXT:WEND
320 GOTO 10
```

Program 3



After decades of being blown out of the skies by humanity, the invaders built the ultimate weapon. It swung into orbit around planet Earth and promptly exploded when it was powered up. The invading horde retreated to examine the small print on the sticky tape warranty and Earth, for the most part, was safe.

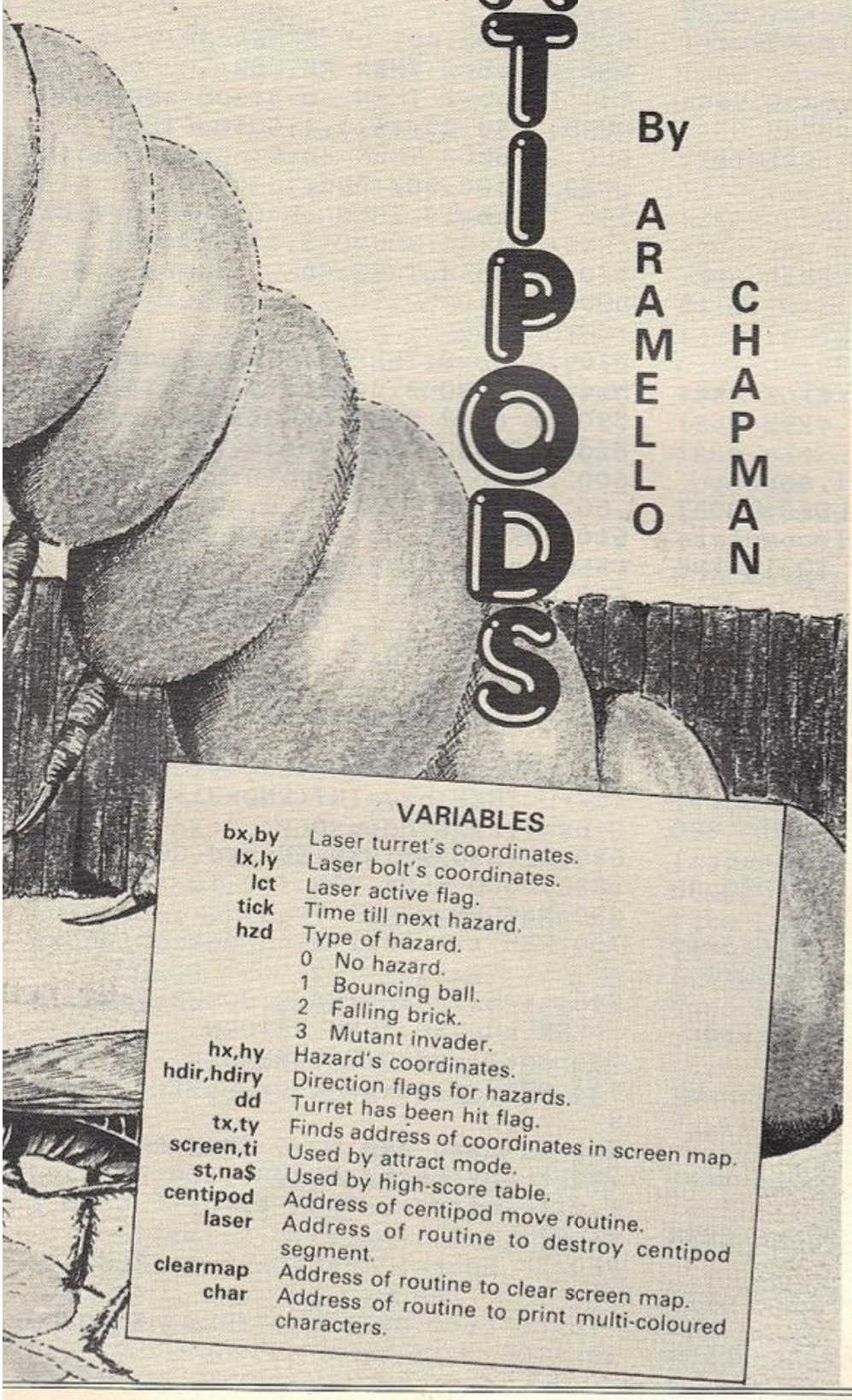
The explosion generated an intense wave of radiation which swept across your garden with interesting effects on the resident wildlife. Those little red bugs which used to eat you cabbages have mutated into giant Centipods with only one thing in mind — revenge.

Additional hazards come in the shape of falling brick, which appear out of nowhere and can deliver permanent headaches to the unwary.

In a desperate bid for survival you clamber into one of your three trusty mega-kill laset turrets. It can move left, right, up and down and fire a single laser bolt.

CENTIPODS

By ARAMELLO CHAPMAN



The bolt is powerful but will not destroy the falling bricks and loses strength by the time it reaches the far side of the garden.

When you hit a Centipod the segment dies leaving two live halves. When all the segments are dead bigger and meaner specimens are to be found lurking in other parts of the garden only too willing to take up the fray.

Points are scored for killing segments, any other life form in the garden and the boxes of goodies which are dropped by the rogue invader.

VARIABLES

bx,by	Laser turret's coordinates.
lx,ly	Laser bolt's coordinates.
lct	Laser active flag.
tick	Time till next hazard.
hzd	Type of hazard. 0 No hazard. 1 Bouncing ball. 2 Falling brick. 3 Mutant invader.
hx,hy	Hazard's coordinates.
hdir,hdiry	Direction flags for hazards.
dd	Turret has been hit flag.
tx,ty	Finds address of coordinates in screen map.
screen,ti	Used by attract mode.
st,na$	Used by high-score table.
centipod	Address of centipod move routine.
laser	Address of routine to destroy centipod segment.
clearmap	Address of routine to clear screen map.
char	Address of routine to print multi-coloured characters.

```
10 REM Centipods
20 REM by A. Chapman
30 REM (c) Computing
 With The Amstrad
40 REM ----------
 CPC only ----------
50 GOSUB 1620:REM
 initial set up
60 GOSUB 2730:REM
 attract mode
70 GOSUB 1260:REM
 set up variables
80 GOSUB 1410:REM
 print score line
90 GOSUB 1290:REM
 print screen
100 REM **********
 main loop **********
110 GOSUB 330:GOSUB
 710:IF dd THEN 160
120 GOSUB 410:GOSUB
 710:IF dd THEN 160
```

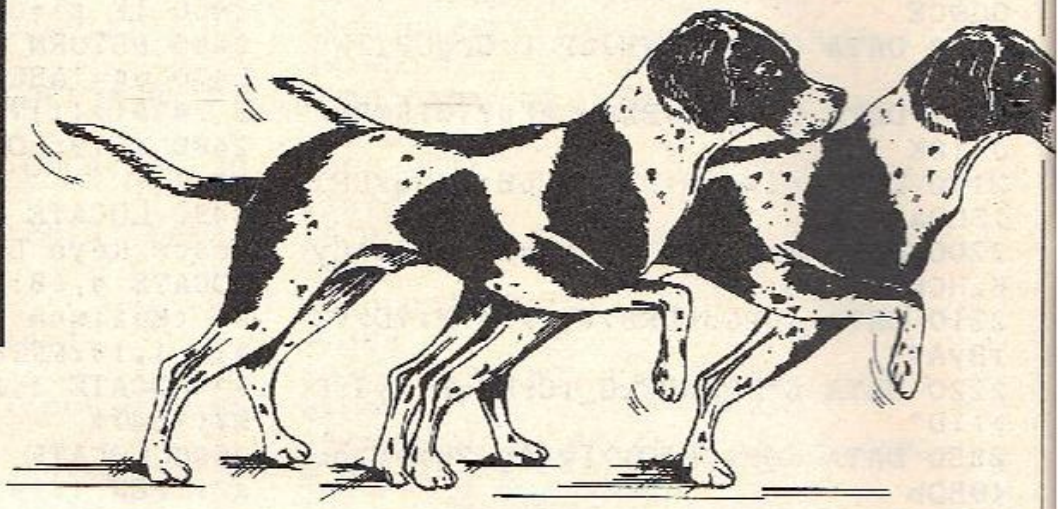
```
130 u=u XOR 1:IF u THEN 120
140 CALL cpod:IF PEEK(38501)=0 THE
N 260
150 GOSUB 670:GOSUB 710:IF dd <>1
THEN 110
160 REM ******** lost a life *****
***
170 SOUND 3,100,150,15,2,2,15:IF b
x=19 AND by=21 THEN 200
180 PEN 2:FOR g=250 TO 252:LOCATE
bx+1,by+1:PRINT CHR$(g):FOR f=1 TO
 100:NEXT f,g
190 FOR g=252 TO 250 STEP -1:LOCAT
E bx+1,by+1:PRINT CHR$(g): FOR f=1
 TO 50:NEXT f,g
200 CALL char,bx,by,1:lives=lives-
1
210 PEN 1:LOCATE #1,18,24:PRINT#1,
lives
220 FOR f=1 TO 1000:NEXT
230 IF lives>0 THEN 90
240 LOCATE 7,10:PEN 14:PRINT"Game
Over":FOR f=0 TO 100:FOR g=0 TO 15
:INK 14,g:NEXT g,f
250 GOTO 60
260 REM ******** next level ******
**
270 scr=scr+(100*level)
280 LOCATE#1,5,24:PRINT#1,scr
290 WINDOW#2,2,18,9,11:CLS#2:LOCAT
E 4,10:PEN 14:PRINT"Level complete
d":FOR f=1 TO 50:FOR g=0 TO 15:INK
 14,g:NEXT g,f
300 level=level+1:IF level>8 THEN
level=8
310 GOTO 90
320 REM ***** move laser base ****
*
330 tx=bx:ty=by:GOSUB 1400
340 IF INKEY(71)>-1 AND PEEK(adr-1
)=0 AND bx>0 THEN bx=bx-1:GOTO 380
350 IF INKEY(63)>-1 AND PEEK(adr+1
)=0 AND bx<19 THEN bx=bx+1:GOTO 38
0
360 IF INKEY(39)>-1 AND PEEK(adr+2
0)=0 AND by<21 THEN by=by+1:GOTO 3
80
370 IF INKEY(36)>-1 AND PEEK (adr-
20)=0 AND by>18 THEN by=by-1
380 CALL char,tx,ty,1:CALL char,bx
,by,5
390 RETURN
400 REM ***** move laser bolt ****
*
410 IF INKEY(47)>-1 AND lct=0 THEN
 470
420 IF lct=0 THEN FOR lp=1 TO 15:N
EXT:RETURN
430 ty=ly:tx=lx:GOSUB 520:IF lct=0
 THEN 500
440 ty=ly-1:tx=lx:IF ty=0 THEN lct
=0 ELSE GOSUB 520
450 CALL char,lx,ly,1:IF lct=1 THE
N CALL char,tx,ty,6:lx=tx:ly=ty
460 RETURN
470 lct=1:tx=bx:ty=by-1:GOSUB 520:
SOUND 2,10,20,10,2,1
480 IF lct=0 THEN RETURN
490 CALL char,tx,ty,6:lx=tx:ly=ty
500 RETURN
510 REM *** laser bolt collision ?
 ***
520 GOSUB 1400:ob=PEEK(adr)
530 IF ob=0 THEN RETURN
540 IF ob=2 THEN ob=3:scr=scr+10:S
OUND 3,50,20,15,1,1:GOTO 610
550 IF ob=3 THEN ob=0:scr=scr+15:S
OUND 3,100,50,15,1,1
560 IF ob=10 THEN ob=0:scr=scr+INT
(RND*50)+50:SOUND 3,50,25,15,3,1
570 IF ob=4 THEN CALL laser,tx,ty:
SOUND 3,10,50,10,2,2,15:scr=scr+25
:GOTO 630
580 IF ob=9 THEN ob=0:scr=scr+100:
hzd=0:SOUND 3,50,25,15,0,1,15
590 IF ob=7 THEN ob=0:scr=scr+200:
hzd=0:SOUND 3,50,20,15,2,1
600 IF ob=11 THEN ob=0:SOUND 3,100
,50,15,1,1
610 POKE adr,ob:IF ob=0 THEN ob=1:
CALL char,tx,ty,ob
620 CALL char,tx,ty,ob
630 lct=0
640 LOCATE#1,5,24:PRINT#1,scr
650 RETURN
660 REM ****** hazard manager ****
**
670 IF hzd=0 THEN tick=tick-1:IF t
ick=0 THEN hzd=INT(RND*3)+1:tick=5
0:hx=0:hy=0:hdir=0
680 IF hzd>0 THEN ON hzd GOSUB 730
,890,1000 ELSE FOR f=0 TO 20:NEXT
690 RETURN
700 REM *** collision detection **
*
710 tx=bx:ty=by:dd=0:GOSUB 1400:IF
 PEEK(adr)=0 THEN RETURN
720 dd=1:hzd=0:RETURN
730 REM ******** move ball *******
*
740 IF hx=0 AND hy=0 THEN 840
750 ty=hy:tx=hx:GOSUB 1400:POKE ad
r,0:IF hdiry=0 THEN ty=hy-1 ELSE t
y=hy+1
760 IF hdir=1 THEN tx=hx+1 ELSE tx
=hx-1
770 IF tx<0 OR tx>19 THEN hzd=0:GO
TO 800
```

```
780 IF ty=22 THEN ty=21:hdiry=0:SO
UND 2,50,20,15,3,2 ELSE IF ty=17 T
HEN ty=18:hdiry=1:SOUND 2,70,20,15
,3,2
790 GOSUB 1400:IF PEEK(adr)=4 THEN
 820
800 CALL char,hx,hy,1
810 hx=tx:hy=ty
820 IF hzd>0 THEN CALL char,hx,hy,
7:ty=hy:tx=hx:GOSUB 1400:POKE adr,
7
830 RETURN
840 ty=INT(RND*3)+18:IF RND>0.7 TH
EN tx=0:hdir=1 ELSE tx=19:hdir=0
850 GOSUB 1400:IF PEEK(adr)=4 THEN
 RETURN
860 POKE adr,7:hx=tx:hy=ty:CALL ch
ar,hx,hy,7
870 IF RND>0.5 THEN hdiry=1 ELSE h
diry=0
880 RETURN
890 REM **** move flying brick ***
*
900 IF hx=0 AND hy=0 THEN 960
910 ty=hy:tx=hx:GOSUB 1400:POKE ad
r,0:ty=ty+1:IF ty=22 THEN hzd=0:GO
TO 940 ELSE GOSUB 1400
920 IF PEEK(adr)=4 THEN ty=hy:GOSU
B 1400
930 POKE adr,8
940 CALL char,hx,hy,1:IF hzd>0 THE
N CALL char,tx,ty,8:hx=tx:hy=ty
950 RETURN
960 tx=INT(RND*17)+1:ty=0:GOSUB 14
00
970 IF PEEK(adr)<>0 THEN RETURN
980 hx=tx:hy=ty:CALL char,hx,hy,8
990 RETURN
1000 REM *** move mutant invader *
**
1010 IF hx=0 AND hy=0 THEN 1120
1020 ty=hy:IF hdir=0 THEN tx=hx+1
ELSE tx=hx-1
1030 ob=1:IF tx>19 OR tx<0 THEN hz
d=0:GOTO 1060
1040 GOSUB 1400:IF PEEK(adr)=4 THE
N tx=hx:ty=hy:ob=9:GOSUB 1400:GOTO
 1060
1050 ob=INT(RND*11)+1:IF ob<10 AND
 ob>5 THEN ob=2 ELSE IF ob<6 THEN
ob=1
1060 IF hzd>0 THEN CALL char,tx,ty
,9:POKE adr,9:SOUND 1,600,15,10,1,
1
1070 t1=tx:t2=ty:tx=hx:ty=hy:GOSUB
 1400
1080 CALL char,hx,hy,ob
1090 IF ob=1 THEN ob=0
1100 POKE adr,ob:hx=t1:hy=t2
1110 RETURN
1120 IF RND>0.6 THEN hdir=0:tx=0 E
LSE tx=19:hdir=1
1130 ty=INT(RND*20)+2:GOSUB 1400
1140 IF PEEK(adr)<>0 THEN RETURN
1150 hx=tx:hy=ty:CALL char,hx,hy,9
1160 RETURN
1170 REM ***** setup centipods ***
**
1180 POKE 38500,pods:adr=&8CA0
1190 FOR f=1 TO pods
1200 POKE adr,1:POKE adr+1,15+f
1210 POKE adr+2,255:POKE adr+3,0
1220 POKE adr+4,0:POKE adr+5,0
1230 adr=adr+6:NEXT
1240 POKE adr-2,1:POKE 36003,1
1250 RETURN
1260 REM **** setup variables ****
1270 level=1:scr=0:lives=3
1280 RETURN
1290 REM ****** setup screen *****
1300 CALL clearmap:CLS
1310 FOR f=1 TO (level*5)+10
1320 tx=INT(RND*18)+1:ty=INT(RND*2
0)+1:GOSUB 1400
1330 POKE adr,2:CALL char,tx,ty,2
1340 NEXT
1350 pods=10*level:IF pods>60 THEN
 pods=60
1360 GOSUB 1170
1370 bx=9:by=21:CALL char,bx,by,5:
lct=0:tick=50:hzd=0
1380 RETURN
1390 REM *** screen map address **
*
1400 adr=&9844+(tx+((ty*20)-1))+1:
RETURN
1410 REM ***** print scoreline ***
*
1420 PEN 1:MODE 0:WINDOW#1,1,20,1,
22
1430 FOR f=0 TO 19:CALL char,f,22,
8:NEXT
1440 LOCATE 1,24:PRINT"1up:"scr
1450 CALL char,15,23,5:LOCATE 17,2
4:PRINT":";lives
1460 WINDOW SWAP 1,0:RETURN
1470 REM ******* score table *****
1480 MODE 0
1490 LOCATE 5,1:PEN 6:PRINT"Score
Table":LOCATE 5,2:PEN 2:PRINT STRI
NG$(12,CHR$(131))
1500 CALL char,5,5,2:LOCATE 8,6:PR
INT"- 10 pts"
1510 CALL char,5,7,3:LOCATE 8,8:PR
INT"- 15 pts"
1520 CALL char,5,9,4:LOCATE 8,10:P
RINT"- 25 pts"
1530 CALL char,5,11,9:LOCATE 8,12:
```

```
PRINT"- 100 pts"
1540 CALL char,5,13,7:LOCATE 8,14:
PRINT"- 200 pts"
1550 CALL char,5,15,10:LOCATE 8,16
:PRINT"- ?? pts"
1560 CALL char,5,17,8:LOCATE 8,18:
PRINT"- 0 pts"
1570 CALL char,5,19,11:LOCATE 8,20
:PRINT"- 0 pts"
1580 PEN 7:LOCATE 5,25:PRINT"Press
 Space "
1590 GOSUB 2690
1600 IF ti=0 THEN screen=1
1610 RETURN
1620 REM ****** initial setup ****
*
1630 DEFINT a-z:MEMORY 35990
1640 MODE 1:FOR i=0 TO 15:READ c:I
NK i,c:NEXT
1650 BORDER 0:LOCATE 15,10:PEN 2:P
RINT"Please wait ..."
1660 DATA 0,18,6,24,2,8,20,26,15,1
6,7,9,13,22,0,0
1670 clearmap=&95AF:char=&95BE:las
er=&9534:cpod=&9470
1680 DIM st(8),na$(8):FOR f=1 TO 8
:na$(f)="Centipods":st(f)=9000-(f*
1000):NEXT
1690 SYMBOL 250,0,0,8,32,2,8,0,0
1700 SYMBOL 251,2,68,40,76,18,73,7
2,128
1710 SYMBOL 252,130,68,41,16,231,2
4,68,145
1720 SYMBOL 253,198,165,198,165,6,
40,40,16
1730 ENT 2,15,10,10:ENV 1,15,-1,1:
ENT -1,15,10,1,15,-10,1:ENV 2,15,-
1,10:ENV 1,1,0,2,11,-11,2:ENV 3,10
,-1,2
1740 ad1=&90A6:ad2=&91DE:ln=1880:G
OSUB 1780
1750 ad1=&9470:ad2=&9600:ln=2070:G
OTO 1780
1760 c(m)=ASC(MID$(a$,k+m,3))-59:R
ETURN
1770 PRINT"Error in Line ";ln:END
1780 FOR a=ad1 TO ad2 STEP 18:READ
 a$
1790 ch=0:FOR i=0 TO 8:j=i*2:k=i*3
1800 FOR m=1 TO 3:GOSUB 1760:ch=ch
+c(m)
1810 NEXT:p=a+j
1820 IF c(1)>15 OR c(2)>63 OR c(3)
>63 THEN 1770
1830 POKE p+1,c(1)*16+((c(2) AND 6
0)/4)
1840 POKE p,c(3)+((c(2) AND 3)*64)
:NEXT
1850 k=27:m=1:GOSUB 1760:m=2:GOSUB
 1760
1860 IF ch<>c(1)*64+c(2) THEN 1770
1870 ln=ln+10:NEXT:RETURN
1880 DATA ;;;;K;;=;G;;;>CG<?;\CG=?
GN;=O
1890 DATA =S;;>J?K;;=E<?;;=E;K;;=E
;K;=D
1900 DATA ;;[<[;;=S>;?;CS>;\=Kk=G;
;;n?d
1910 DATA ;K;;;]<?;;;];?;;;];>GGk;
;MG>c
1920 DATA ?kC<k;;;gB[?;]oB[?;]o<k;
;;gBs
1930 DATA ;MG?kC;>GGk;@K;;=c@K;;=c
@KC@;
1940 DATA ;Mc@K]<Ac@K]<AcE_]<@Q=Hq
F_NBt
1950 DATA @K]<AcEvzJxXEvzJxXIvzJzX
JzzJ;
1960 DATA JzzJzzJzzEvzJxXEvzJxXIvz
JzXLo
1970 DATA <;;;;[>K;;;s<KK=;c>ko>[w
>koB_
1980 DATA >[w=kK=;W>;;;;k<;;;;[;lG
;lGAH
1990 DATA ;lG;lG;lG;lGGnGGnG?kG?kG
?kGDE
2000 DATA ?kG?kG?kGGnGGnG?O;;=EGx@
CfJCE
2010 DATA ?P@CeE?P@CeEGw;;>J?PHCuE
;=IAK
2020 DATA ?o;;;G;k;J>kJ>kJmC?NwE]C
?K;BW
2030 DATA E]C?K;E]CInwE]C?LOJmC?Nw
J>kCo
2040 DATA J>k;KC;K;;;?C[CC[;;<??;;
;=;>L
2050 DATA ?;C;M;?<?C=;C?;;<C?F;?F;
;;;<_
2060 REM ------------------------
----
2070 DATA =BXCm[>EjDT'AKu?YQHr@;<y
;>yDO
2080 DATA >kcAOmHqQ;DaAvXHo<;Hy;By
>OcEY
2090 DATA AWuHqQ;DrASuHqQ;@r>DwDTb
>DxEw
2100 DATA DTaBvXJs?=[;>cLDTb>dbDTa
ExjEe
2110 DATA BzH;UPGo<DRB;SLHo;G?TFKK
BrDBj
2120 DATA ASmBmQAWmHqQ;Py;>y=CcJtx
=[NFG
2130 DATA IOjGogDQAJv\=;;=k'BRXHo<
;DoD[
2140 DATA ;KyBzH?YPGZHGIPDMcJtx=[;
IOHDS
2150 DATA GohDQAJv\=;;=o>HkSGo_DQA
;>yDR
```

```
2160 DATA <C[Jtw=;Q=S=Ho;;Py;BiBZX
GG@CE
2170 DATA DNc=O`CVHJuP=[;GrgDP]IWS
AVXDY
2180 DATA Ho;;Di=BXCm[AKu?YQIR@BvX
Js;DK
2190 DATA =[;HoH;Dy==wHoB;@yGmxDP]
;SLDa
2200 DATA Ho;I?T()<Gb\AvXHo<;Da;Cy
BzHCh
2210 DATA ;UPGo=DRB>VX;;;>VX;?D>VX
;ByAV
2220 DATA GrDDQHBZxG_;CrHJqP;<yJrD
>;\D^
2230 DATA JqS;;qJO;>lw<^y<?k;>y;oc
<@BDb
2240 DATA ;;O<bxJkK;;QJpXJ?T<BD;B[
?K\BS
2250 DATA >US=G;BcV==nGbs?VXHo;;Da
AvXCt
2260 DATA Bk?<byIRKA[\<AK;;[<;THBx
BrfCc
2270 DATA AXg<fH;uw;SCIO?BWU<G^JcK
Bn\DN
2280 DATA ;^A><bB_A@;<;b;;pJIW[;>D
;;;@G
2290 DATA ;;;;;;@<J@DL@LN@TP@\R@dT
=s[@i
2300 DATA rem ******** high score
*****
2310 MODE 1:LOCATE 4,5:PEN 3:PRINT
 CHR$(150);STRING$(32,CHR$(154));C
HR$(156)
2320 LOCATE 16,3:PEN 2:PRINT"High
Score ":PEN 3
2330 FOR f=6 TO 15:LOCATE 4,f:PRIN
T CHR$(149):LOCATE 37,f:PRINT CHR$
(149):NEXT
2340 LOCATE 4,16:PRINT CHR$(147);S
TRING$(32,CHR$(154));CHR$(153)
2350 FOR f=1 TO 8
2360 IF scr>st(f) THEN GOSUB 2470:
f=10:scr=0
2370 NEXT
2380 FOR f=1 TO 8:PEN 1:LOCATE 8,f
+6:PRINT na$(f):LOCATE 18,f+6:PEN
3:PRINT".........";st(f):NEXT
2390 LOCATE 1,17:PRINT STRING$(220
," ")
2400 IF INKEY$<>"" THEN 2400
2410 PEN 2:LOCATE 1,20:PRINT STRIN
G$(40,CHR$(154)):LOCATE 1,22:PRINT
 STRING$(40,CHR$(154)):PEN 1
2420 LOCATE 1,23:PRINT STRING$(40,
" ")
2430 LOCATE 11,21:PEN 3:PRINT"Pres
s";:PEN 1:PRINT"< SPACE >";:PEN 3:
PRINT" to play":PEN 1
2440 GOSUB 2690
2450 IF ti=0 THEN screen=2
2460 RETURN
2470 a$="ABCDEFGHIJKLMNOPQRSTUVWXY
Z .#%&()!{}?*+"+CHR$(253)
2480 c=19:LOCATE 1,20:PEN 1:PRINT
a$
2490 LOCATE 2,17:PEN 1:PRINT"Use c
ursor keys Left, Right and Enter":
LOCATE 4,18:PRINT"To select letter
s. (Maximum of 10.)":PEN 2:LOC
ATE 1,19:PRINT STRING$(40,CHR$(154
)):LOCATE 1,22:PRINT STRING$(40,CH
R$(154))
2500 LOCATE 12,23:PEN 3:PRINT"Pres
s";:PEN 1:PRINT"< X >";:PEN 3:PRIN
T" To Exit.":PEN 1
2510 x$=""
2520 FOR z=1 TO 10
2530 LOCATE c,21:PEN 2:PRINT" "
2540 IF INKEY(1)=0 AND c<40 THEN c
=c+1
2550 IF INKEY(8)=0 AND c>1 THEN c=
c-1
2560 IF INKEY(18)=0 AND c=40 THEN
LOCATE 7,f+6:PRINT"          ":z=1
00:GOTO 2610
2570 IF INKEY(63)=0 THEN z=11:GOTO
 2610
2580 IF INKEY(18)<>0 THEN LOCATE c
,21:PRINT"*":FOR a=1 TO 50:NEXT:GO
TO 2530
2590 x$=x$+MID$(a$,c,1):LOCATE 7+z
,f+6:PEN 1:PRINT MID$(a$,c,1)
2600 FOR a=1 TO 200:NEXT
2610 NEXT
2620 IF z=101 THEN 2510
2630 st(8)=scr:na$(8)=x$
2640 f=0:FOR z=1 TO 7
2650 IF st(z)<st(z+1) THEN t=st(z+
1):st(z+1)=st(z):st(z)=t:a$=na$(z+
1):na$(z+1)=na$(z):na$(z)=a$:f=1
2660 NEXT
2670 IF f=1 THEN 2640
2680 fr=FRE(""):RETURN
2690 WHILE (screen<>0) AND (ti>0)
2700 WHILE INKEY$<>"":WEND
2710 IF INKEY(47)<>-1 THEN screen=
0
2720 ti=ti-1:WEND:RETURN
2730 REM ***** attract mode *****
2740 screen=1:WHILE screen<>0
2750 ti=2000:ON screen GOSUB 2300,
1470
2760 WEND:RETURN
```

Part VII of Mike Bibby's introduction to machine code



Last month we saw how our single registers could be combined to form register pairs BC,DE, HL, each capable of holding a 16 bit number. This means a register pair can hold any number in the range &O — &FFFF (0 — 65535).

Loading the register pair with a constant couldn't be easier — just specify the relevant opcode for the pair in question, followed by two bytes containing the constant in our usual lo byte, hi byte fashion.

For example,

LD BC, &2FF8

would translate as:

Ø1 F8 2F

We also met a similar looking type of instruction:

LD BC, (&2FF8)

In this case, though , the &2FF8 isn't a constant — it's a pointer to a memory location. As we've seen, it's a sort of two byte peek, with C taking the value stored in memory location &2FF8 and B taking the value stored in &2FF9.

It translates as:

ED 4B F8 2F

The brackets make a considerable difference!

The register pairs' ability to specify any number in the range &O to &FFFF means that we can use them to point at, or index, any location in memory.

In fact we often use register pairs in this way. For instance, if HL contained &2FF8, the instruction:

LD A, (HL)

would load the A register with ths contents of memory location &2FF8. In effect it's doing an 8 bit peek, with the address to be peeked stored in register pair IIL.

And, just as you can "peek", so you can "poke":

LD (HL),A

would copy the contents of the A register into the memory location specified by HL.

Try Program I to see how this idea works:

```
Address hexcode  mnemonics
3ØØØ    21 F8 2F LD HL, &2FF8
3ØØ3    3E FF    LD  A,&FF
3ØØ5    77       LD  (HL),A
3ØØ6    C9       RET
```

Program I

Once it's executed, you should see that &2FF8 — the first location in Hexer's workspace — has the value &FF.

Line by Line, the program:

* Sets the pointer to memory by loading HL with &2FF8.
* Loads the A register with &FF — our "marker byte".
* Loads the location pointed to by HL — &2FF8 — with the contents of A.— &FF.
* Returns.

As you can see, all works as expected. You might be wondering why we didn't load A with &FF and "poke" directly, as in program 11:

```
Address hexcode  mnemonics
3ØØØ    3E FF    LD A,&FF
3ØØ2    32 F8 2F LD (&2FF8),A
3ØØ5    C9       RET
```

Program II

Actually, there's nothing wrong with doing it this way, and it does save you a couple of bytes. However, if you point at memory with HL rather than directly with a constant, you can take advantage of instructions such as:

INC HL (opcode &23)

and

DEC HL (opcode &2B)

to alter the memory location you're pointing at.

Use register pairs as pointers to specify a memory location

Program 111 shows the sort of thing I mean:

```
address  hex code   mnemonics
3ØØØ     21 F8 2F  LD HL, &2FF8
3ØØ3     3E FF     LD A, &FF
3ØØ5     77        LD (HL),A
3ØØ6     23        INC HL
3ØØ7     77        LD (HL),A
3ØØ8     23        INC HL
3ØØ9     77        LD (HL) ,A
3ØØA     C9        RET
```

Program III

This puts &FF into three consecutive bytes of workspace. In practice, as you'll know from your Basic programming, you'd normally palce the repeated:

```
LD (HL) ,A
INC HL
```

inside a loop, but more of that later.

To see if you've understood what we've done so far, try writing a program that has identical results to program 111, but using DEC HL instead of INC HL.

Then writen another version that puts the values 0,1,2, into consecutive memory location from &2FF8 onwards.

You'll probably need INC A — opcode & 3C).

One of the nice things about using HL to point to memory is that, in effect, the memory specified becomes an "extra register". That is, the instructions you've learned using single registers can often by used with (HL) substituted for one of those registers, the contents of LH pointing to the memory location you want to use.

For example, since:

INC A

and:

DEC A

exist, you won't be surprised to learn that there are instructions:

INC (HL) opcode &34

and:

DEC (HL) opcode &35

These work in exactly the same way as their single register counterparts except that instead of directly specifying the register you want, you give the address in memory you want to effect via the HL register.

Using HL in this way fills the "gaps" in our tables of opcodes. You may have noticed that there's a pattern to the opcodes. If not, have a look at last month's Ready Reference.

For instance, if you look at our old table for LD r,n you'll see that the opcode for the B register is &06, for C it's &OE, D is & 16, E is &1E, H is &26 and L is &2E. See the pattern? Suddenly, though, there's a jump to &3E for A. What happened to the missing &36?

Well, &36 is the opcode for:

LD (HL),n

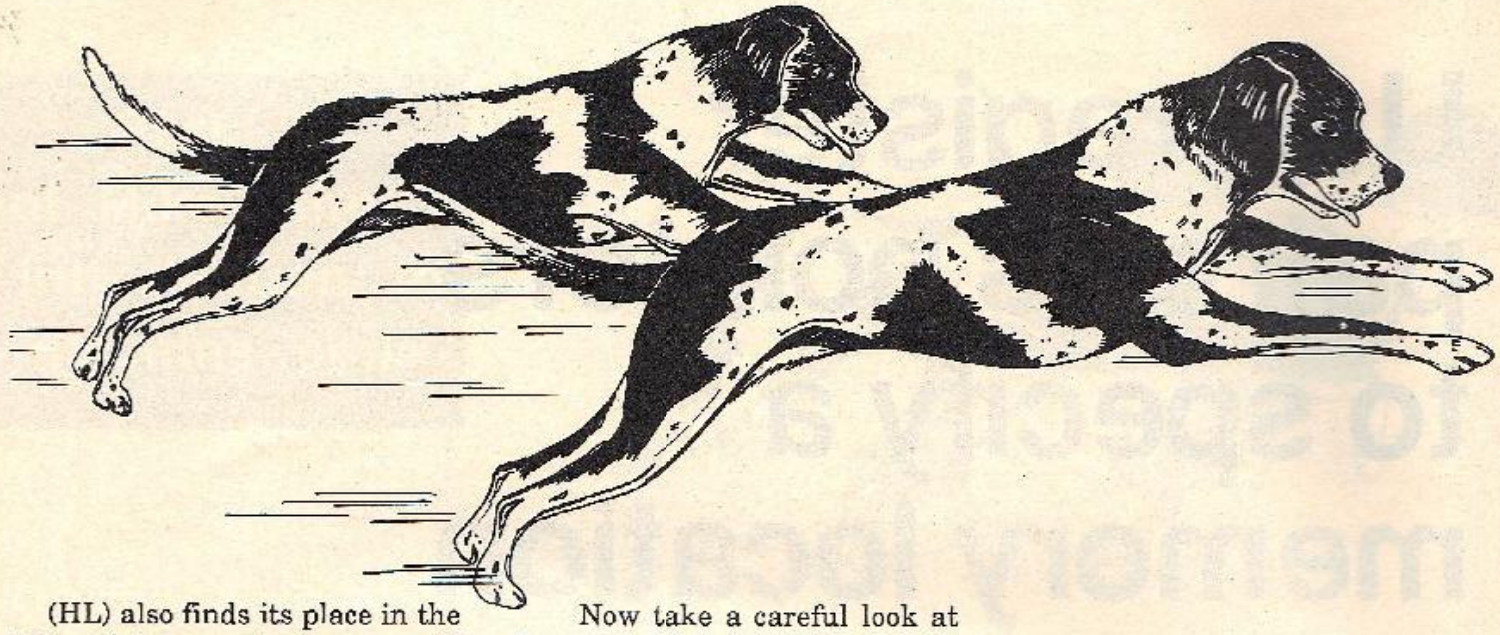
as you've probably already guessed. Table 1 shows the full range of opcodes for LD r,n including (HL).

LD B,n	Ø6	n
LD C,n	ØE	n
LD D,n	16	n
LD E,n	1E	n
LD H,n	26	n
LD L,n	2E	n
LD (HL),n	36	n
LD A,n	3E	n

Table I: LD r,n opcodes

	r'	B	C	D	E	H	L	(HL)	A
r	B	4Ø	41	42	43	44	45	46	47
	C	48	49	4A	4B	4C	4D	4E	4F
	D	5Ø	51	52	53	54	55	56	57
	E	58	59	5A	5B	5C	5D	5E	5F
	H	6Ø	61	62	63	64	65	66	67
	L	68	69	6A	6B	6C	6D	6E	6F
	(HL)	7Ø	71	72	73	74	75	*	77
	A	78	79	7A	7B	7C	7D	7E	7F

Table II: Opcodes for LD r,r'



(HL) also finds its place in the LD r,r' instructions, as you'll see in Table II. LD A, (HL) exists, as does LD (HL),A. Notice there's no opcode for LD (HL), (HL) — a meaningless instruction. I've placed an asterisk there.

To sound off our coverage, you won't be surprised to learn that you can:

ADD A,(HL) &86

and :

SUB (HL) &96

Using a register pair to point at memory this way is known as indirect addressing — you don't directly tell the Z80 the memory location you want, you tell it to go look at HL to find out!

As you'll see, indirect addressing can be very powerful, but the ideas take a bit of getting used to, so let's have some practice.

Program 1V uses indirect addressing to store &FF in memory location &2FF8. Not as we did in Program 1, by using the A register, but using LD (HL),&FF.

```
address hex code mnemonics
3ØØØ    21 F82F LD HL,&2FF8
3ØØ3    36 FF   LD (HL),&FF
3ØØ5    C9      RET
```

Program IV

Now take a careful look at Programs V and V1. They illustrate the difference between INC HL anfd INC (HL).

```
address hex code mnemonics
3ØØØ    21 F8 2F LD HL,&2FF8
3ØØ3    36 FF    LD (HL),&FF
3ØØ5    23       INC HL
3ØØ6    C9       RET
```

Program V

```
address hex code anemonics
3ØØØ    21 F8 2F LD HL,&2FF8
3ØØ3    36 FF    LD (HL),&FF
3ØØ5    34       INC (HL)
3ØØ6    C9       RET
```

Program V I

Program V is much the samc as program IV, except for an indeffectual INC HL at the end. The program simply puts &FF in &FF8, as you'll see if you examine memory with Hexer. The INC IIL increases the value in HL by one, to &2FF9.

In Program VI, however, after loading &FF into &2FF8, we then INC (HL). This, as you'll see, doesn't change to number in HL (which is still &2FF8), but increases the contents of the memory pointed to by HL by one.

And, since we've already loaded &2FF8 with &FF, increasing it by one wraps the value round to zero. Take a look with IIexer of you don't beleive me.

So, INC IIL increases the number in the register pair by one. INC (IIL) increases the value of the memory location specified by HL by one.

Right, before you run Programm VII, can you predict what it does?

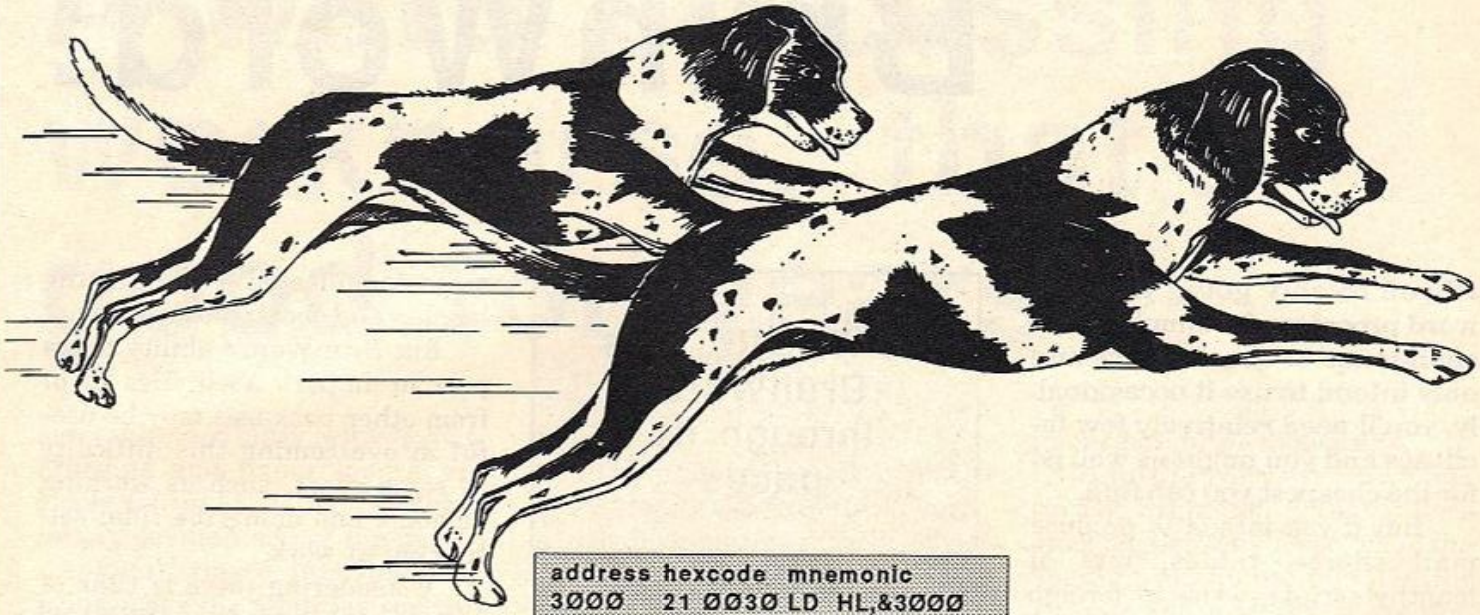
```
address hexcode  mmemonics
3ØØØ    21 F82F  LD HL,&2FF8
3ØØ3    34       INC (HL)
3Ø04    23       INC HL
3Ø05    34       INC (HL)
3ØØ6    23       INC HL
3Ø07    34       INC (HL)
3Ø08    C9       RET
```

Program VII

If you use Hexer to examine the workspace before and after you run the program you'll see that the first three bytes are increased by one.

So far, we've used the HL register as our pointer to memory. In fact, just as the A register has the most versatile range of instructions of the 8 bit registers, so HL is the "leader" of the 16 bit registers.

However, we can point into memory with BC, DE as well, but only in conjunction with the A register, as Table III shows.



mnemonic	opcode
LD (BC),A	Ø2
LD (DE),A	12
LD A,(BC)	ØA
LD A,(DE)	1A

Table III: Indirect addressing with BC, DE

```
address hexcode  mnemonic
3ØØØ    21 ØØ3Ø LD  HL,&3ØØØ
3ØØ3    11 F8 2F LD  DE,&2FF8
3ØØ6    7E       LD A, (HL)
3ØØ7    12       LD (DE),A
3ØØ8    13       INC DE
3ØØ9    23       INC HL
3ØØA    7E       LD A,(HL)
3ØØB    12       LD (DE),A
3ØØC    13       INC DE
3ØØD    23       INC HL
3ØØE    7E       LD A,(HL)
3ØØF    12       LD (DE),A
3Ø1Ø    C9       RET
```

Program VIII

Program VIII uses this idea to copy the first three bytes of the program (from &3000) down into the workspace. This may seem rather futile, but, as I said in the introduction to this series, much of machine code involves moving data from one location to another — and often in blocks.

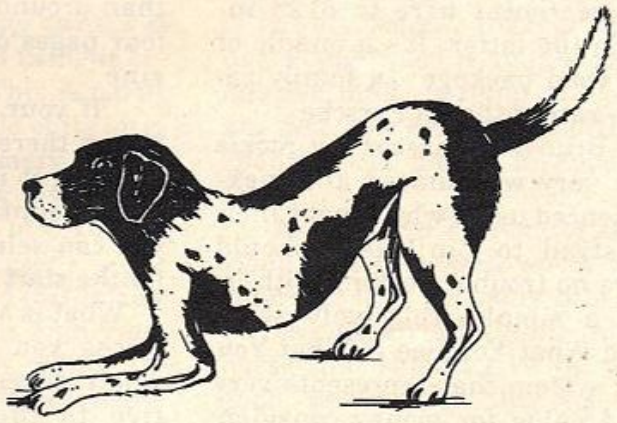
In fact, the Z80 has special instructions for moving blocks of data about in memory, which we'll be covering.

In Program VIII we use two pointers to memory. HL is loaded with &3000, the start of the program, where we are copying bytes from, while DE is loaded with &2FF8, the start of the workspace we are copying to.

Next, A is loaded with the byte pointed at by HL with LD A, (HL).

Then that value is copied into the location pointed at by DE with LD(DE),A.

Both DE and HL are increased by one so as to point to the next bytes along and the copying orocess repeated. DE and HL are then increased again and the copying done a final time.



BrunWord:

JO STORK puts BrunWord through its paces

IF you're only going to use a word processor for simple correspondence or short essays and only intend to use it occasionally, you'll need relatively few facilities and you might as well go for the cheapest you can find.

But if you intend to produce mail shorts, tables, lots of lengthy scripts, write in foreign languages or use mathematical symbols, then you're going to have to pay more.

All word processors are designed to carry out the same basic task: They allow you to type in and make simple corrections to text. And they provide search, replace, cut, paste and copy facilities for more advanced surgery.

But they do differ significantly in the extensions to these functions and for each extra you pay more, in money - and in learning time, for with increased complexity there are more instructions, codes and keystrokes to get used to.

The software reviewed here is BrunWord 6128. It will work on an expanded CPC464 or 664 and all references here to 6128 include the latter. It's a middle of the road package - a family saloon rather than a Porsche.

BrunWord passes my rookie test very well indeed and inexperienced users who bought their Amstrad to zap aliens should have no trouble working with it. It's a simple, relatively standard What You See Is What You Get system that represents very good value for money considering it comes with a spelling checker and mail merge. It is also easy to configure for any printer, whether dot matrix or daisy wheel.

There are some key combination idiosyncracies and unusual jargon but these are only likely to confuse someone already used to another word processor. And in the unlikely event that you become confused, pressing f7 at any time will give the help screen shown in Figure I.

The program works by keeping the complete text in memory. This makes for very fast movement through the text, particularly useful when interrupting work since on restarting you can be at the point where you left off in the blink of an eye.

Similarly you can find any point in the text in just a few seconds. However, the drawback is that your text can be no more than around 24k in size - about four pages of text in this magazine.

If your work is going to be longer there is no alternative to splitting it into sections - not an insurmountable problem since you can select the page number for the start of each chunk.

What is a nuisance is that this forces you to be very careful where pages start and end relative to the different files in which the various segments are stored. And this can become quite complicated if you use the header and footer facilities.

But BrunWord's ability to export or import Ascii files to or from other packages may be useful in overcoming this difficulty in some cases, such as working at home and doing the final setting out at work.

Considering there is 128k of memory available in a CPC6128, the 24k maximum is really rather poor. All the additional memory is used for what many suppliers regard as chargeable extras such as mail merge facilities, but even so Tasword, for example, gives you 64k of text space.

And the 24k limit is even more surprising since BrunWord can use part of it as a "memory file". You can temporarily save a file in memory, start work on another piece of text, bring back the first and finally save all finished work to disc.

This is certainly far quicker than working with the disc throughout, but I wonder if the seconds saved justify the space this coding has taken from potential single file size.

Despite being geared to the lower end of the word processor market, BrunWord offers con



Figure 1: The help screen

A word processing workhorse, that's good for years

siderable flexibility with layout. Margins and paper length are easily set and tabs can be placed in any position across the page.

Furthermore, provided your printer has the facilities you can dictate that portions of text be bold, italic, double width or underlined.

Where BrunWord's basic design structure is limited is in that you can't mix page styles within a document. So for example if you wanted to leave gaps in a document for diagrams, you would have to build up the complete text from smaller portions, each with their own separate layout. However, this is not too time-consuming, thanks to the memory filing capabilities described earlier.

Unusually for a word processor of this size, BrunWord comes with an excellent spelling checker, BrunSpell. The dictionary, held in memory, only offers 21,000 words but there is space available for you to add between 7,000 and 10,000 more.



Figure 2: The spelling checker

The problem for all spelling checkers is that English is such a rich language and most of us use a surprisingly large and varied vocabulary - typically, 40-50,000 words. This gives the software producer a major headache.

If the dictionary is large enough to cope comfortably without constantly alerting you that a word is misspelt merely because it is not included, it is likely to be very slow because of the sheer size of the word file that has to be searched.

What is worse is that this is not a task where the computer can be left unattended since at any moment the run may pause expecting a response to a non-match. Some spelling checkers read through a whole file store up mis-matched words, but you then lose the on-screen advantages of those which work like BrunWord.

For example, many owners of Taspell have stopped using it because it's too slow. This is partly because it has so little of its larger dictionary in memory at any one time, and partly because of anything up to 64k of text that may need to be processed.

As a test, I checked this review using BrunSpell in a highly acceptable 112 seconds, which included making the corrections. It found 47 words missing of which only four were actually mispelt - the reminder were val id but not in the dictionary.

A further very attractive feature of BrunSpell is that it does offer suggestions as to what the spelling might be, by checking against those words in its dictionary which are closest to the spelling you used.

So far I've made BrunWord appear competent, cost effective but not particularly outstanding. This is because I have left the best till last - a very easy-to-use Datafile.

While nowhere near a full database it is a card index system that's perfect for mail merges, since not only does it handle letter headings and labels but also permits some degree fo personalisation of correspondence. The Club secretary borrowing his son's CPC when he's not using it for landing F18s will find it ideal.

Summing up this product is not easy. I could list a whole raft of shortcomings and yet at heart it is very sound. Inexperienced users who need a simple package for mail merging and who are not too confident of their spelling will not be disappointed.

As a starter system for occasional use it is well worth considering. It is probably best described as an admirable workhorse that should give years of good service for the masses. That surely is the test by which all programs should ultimately be judged.



Let's Midi

Just when you think you're getting the hang of computing, along comes something else to confound and befuddle. In the world of music, Midi came along and upset everyone, musicians included. Musicians especially, in fact.

As a computer user, however, you have an adavantage over musicians because the Midi concept is a lot closer to computers than it is to keyboards.

Let's get definitions out of the way first. Midi is an acronym for Musical Instrument Digital Interface. From that you can probably work out what it does: It's simply a way of allowing musical instruments to communicate with each other. And as the name suggests it does it with digital signals.

Most keyboards, synthesisers and organs now have Midi built in. Communication with the outside world is via 5-pin din sockets of which there are three types: Midi In, Midi Out and Midi Thru. Midi In accepts incoming Midi signals and Midi Out transmits Midi signals from the instrument. That is straightforward and virtually all instruments have these two sockets.

Midi Thru is slightly less common. It transmits a duplicate of the signal arriving on the Midi In line and is used to daisychain instruments together. The Midi Thru of one instrument connects to the Midi In of another and the Midi Thru of that one plugs into the Midi In of a third and so on.

Note that although Midi stands for Musical Instrument Digital Interface we refer to the interface

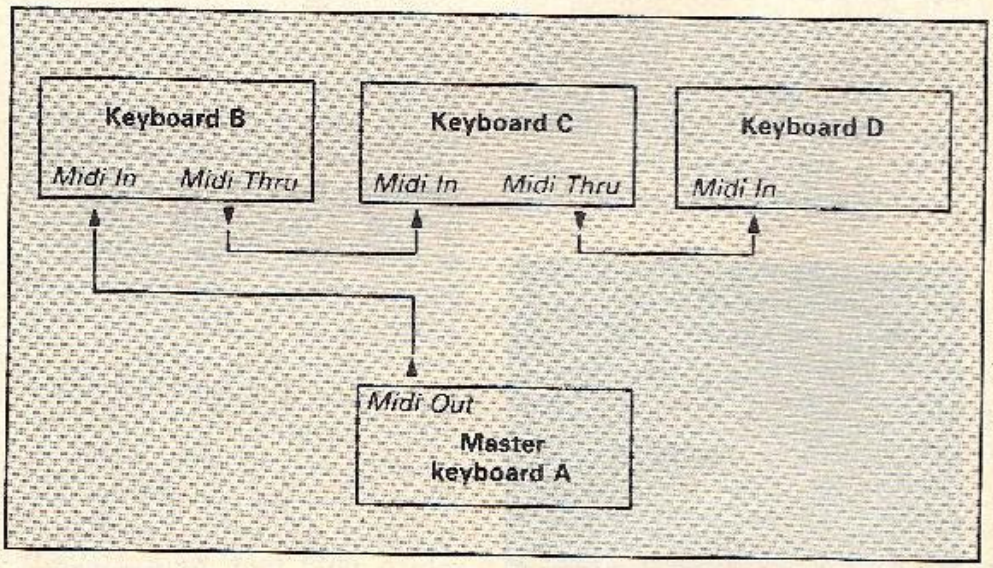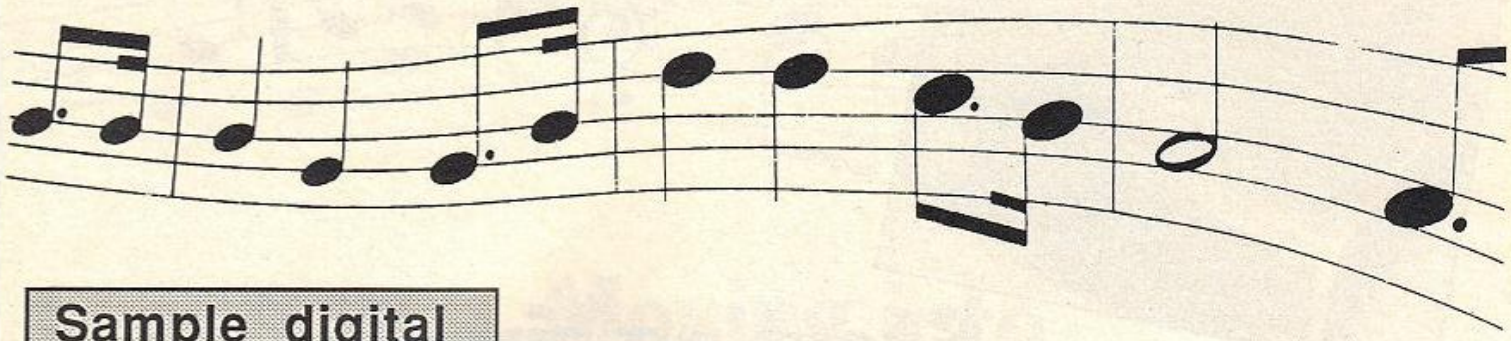


Figure I: Controlling three keyboards from a master keyboard with Midi Thru. Each keyboard receives exactly the same messages



Sample digital do ray me with Ian Waugh

make music

itself as a Midi interface and not just a Midi — such is the perversity of the acronym.

Midi was developed by the major musical instrument manufacturers to overcome the problem of incompatibility. Before Midi you could not, for example, plug one manufacturer's drum machine or keyboard into another's sequencer. If you could they probably wouldn't work.

Now you can plug just about anything into almost anything else and the door to musical instrument interfacing has been thrown wide open. Midi signals carry all sorts of musical messages. The most important are Note On and Note Off information but they also handle such things as pitch bend, modulation, velocity, after touch and patch change.

It's imprtant to realise that Midi deals with digital signals, not audio ones. There is never any actual music floating around in the system and you need an instrument to turn the signals into music. If the Midi Out of keyboard A is plugged into the Midi In of keyboard B, keyboard B will mimic whatever is played on keyboard A. Listeners will think you have four hands. You can daisychain instruments together to play three, four or more keyboards by remote control.

Let's take this a step further. It may be very ego-boosting — and ear-shattering — to play several keyboards together but it's not terribly practical. What would be better is the ability to switch control from one remote keyboard to another — without leaping up and swapping all the leads around.

Midi can do this quite easily as it allows instruments to be tuned into one of 16 channels, rather like tuning in a TV set. So you can set remote keyboard B to receive on channel 1, keyboard C on channel 2 and so on, and to play them you just select the relevant channel on your main keyboard and transmit on that.

This is all jolly nice and lots of fun but it's only the tip of the

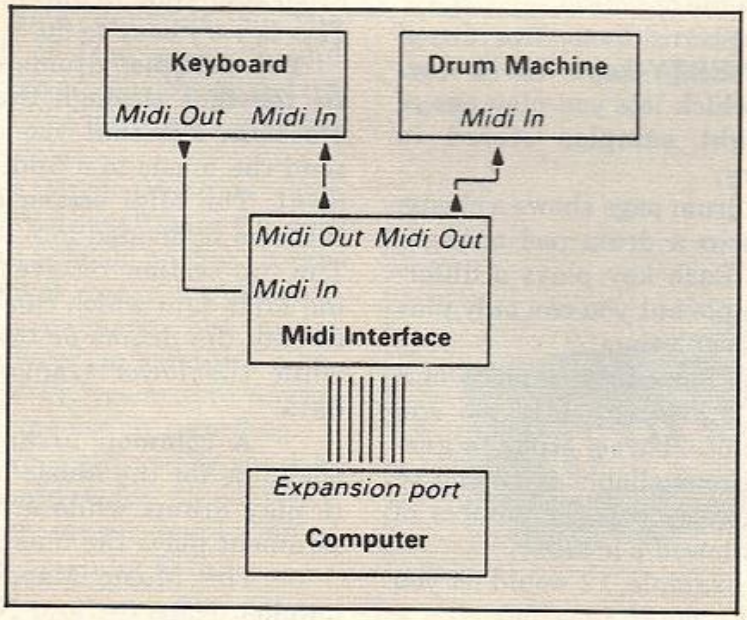
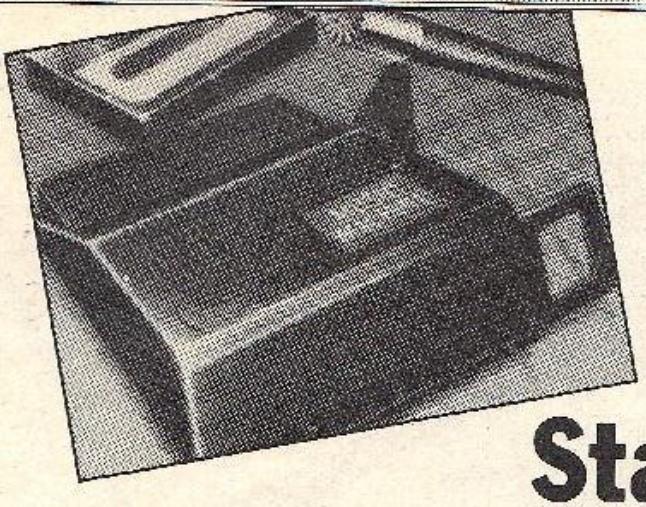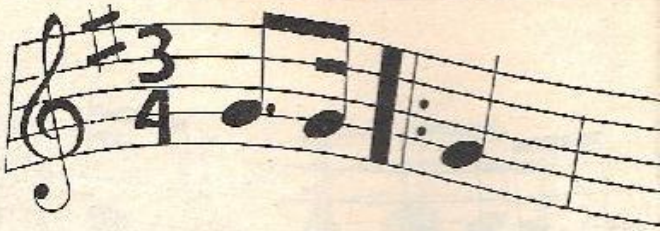


Figure II: Using a computer and Midi interace to control a keyboard and a drum machine. The Midi In on the interface is used to receive information from the keyboard during recording.





Starting simply

Midi iceberg. As signals are digital they are ideally suited to computer control. You have only one pair of hands (if you have more please let us know) and therefore can only play one musical part or instrument at a time.

A computer, however, could send out 16 sets of signals, one on each Midi channel, letting you create a 16-part arrangement.

Realistically, that's probably a little ambitious as not many of us can afford 16 synthesisers. However, there are one or two on the market that are ideally suited to Midi control and perfect for anyone making music with a computer. These have the ability to play more than one sound at a time under computer control. Technically they are known as multi-timbral (meaning more than one tone) instruments and one of the cheapest and most popular is Casio's CZ-101 at around $860.00.

Yamaha's FB-01 (which is currently undergoing an update) at $735.00 is rather more sophisticated but also more difficult to understand initially. It is an expander — that is, it does not have its own keyboard, so it can only be played from another keyboard or from a computer.

If you have an instrument with Midi, how do you get started? Well, you need a Midi interface for your Amstrad and some software.

RAM Electronic's Music Machine was designed by Flare technology, a company formed by ex-Sinclair employees, so you would expect them to know what they are doing — more or less. The Music Machine is quite remarkable in that it offers sound sampling with editing facilities, echo effects, a two-voice music editor and tune composer and Midi compatibility.

The program is menu-driven and options are selected by pressing the capital letter associated with your choice. The menu looks like this: Play, Midi, ecHo, Load/save, tuNe editor, Bar editor, Drums, sAmple editor, pianO, basic, dElete, drUm editor and sampleR. The software loads with seven drum sounds which are quite good and a synth sound which is not very interesting. Let's quickly run through the options.

Piano turns the top two rows of the QWERTY keys into a keyboard which lets you play one of the eight samples stored in memory.

The drum page shows a cluster of keys in a drum pad configuration. Each key plays a different sample but you can only play one note at a time.

The bar editor consists of a double stave with a treble and bass clef. Three "time signatures" are available: 8, 12 and 16 but these are just the number of notes allowed per bar.

For example, 12 would let you palce 12 notes in a bar. It's a very rough and ready system with little scope for any musical subtlety. Notes appear as lines instead of dots.

A bar can hold two lines, both of which use the same voice. You can set different tempos for each bar so you can speed up the music or slow it down. Deleting a bar is a bit messy as you have to go through several screens to select the delete option.

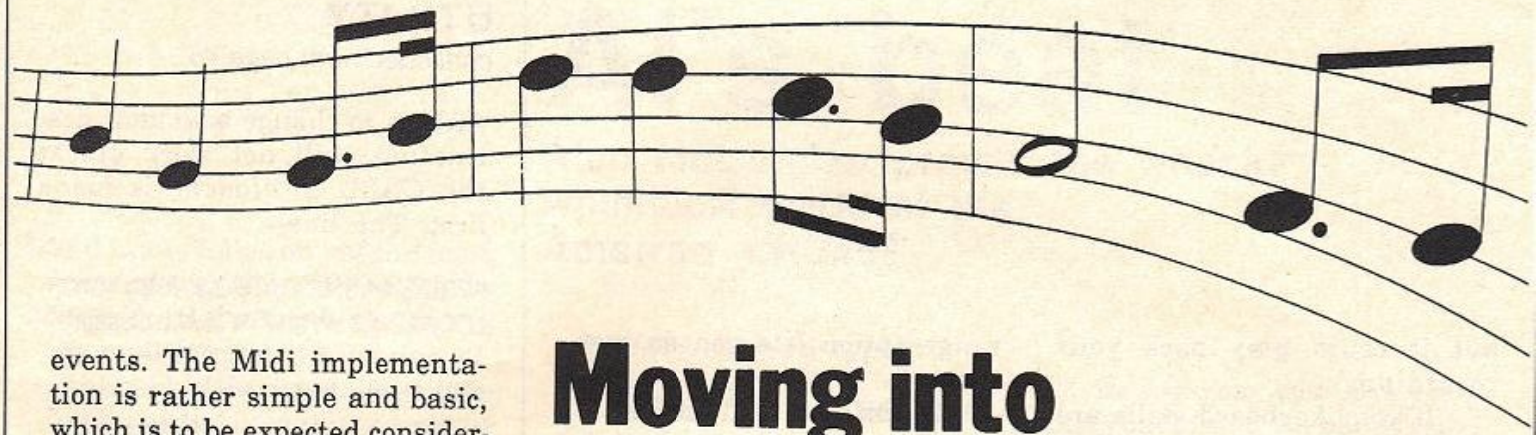
The drum editor is similar to the bar editor but with only one stave of eight lines, each representing a different sample — but you can play three samples at once.

There are two sections to the tune editor, one for drums and one for music. Here up to 255 bars of music and drum patterns can be chained together to form a tune, a process which will be familiar to anyone who has used a programmable drum machine. This is very easy to do.

You can't play drums and music together through the Music Machine. For that you need to send the music to a Midi instrument. The Midi screen controls this and is divided into sections. The top section receives incoming Midi data which can be used to play the drums or the music, while the lower transmits the data.

A common arrangement would be for the Music Machine to play drums while a Midi instrument plays the music.

The Music Machine ony handles Note On and Note Off



events. The Midi implementation is rather simple and basic, which is to be expected considering what it costs.

Again, bearing in mind the price, the sampler is quite sophisticated. It can hold up to eight samples in memory at once but the total storage time is only 1.1 seconds.

You'll probably get more out of the system by using it to store several drum type samples rather than one orchestral one. I found it difficult to make clean samples with the microphone (a cheap 'n' cheerful one is supplied) and got better results by plugging directly into my hi-fi system.

The editor gives a graphic display of the sample and lets you remove sections from the front and end. There's also a useful zoom facility. You can play a sample backwards, too, and loop it — but you'll be lucky to get a glitch-free loop.

Finally, the echo acts on signals from the microphone. The delay time depends on the amount of free memory and can be varied, but there always seemed to be quite a lot of background noise.

The Music Machine costs $125.95 with software on cassette and $159.95 on disk. It certainly gives you a lot of facilities for your money but they're not really up to semi-pro standard and I'd hesitate to recommend it for use in recording.

However, if you're short of money and want to join the computer music revolution it will give you a lot of fun.

Moving into the big band sound

If you want to get involved with Midi at a more professional level then EMR's Miditrack Performer could be for you. It is a real-time sequencing program which means you enter notes into the system "live" and is suitable for Amstrad CPC464, 664 and 6128 computers.

EMR's Midi interface plugs into the Amstrad's expansion port. It has a Midi In, two Midi Outs and a clock Start/Stop socket which is used for controlling a drum machine.

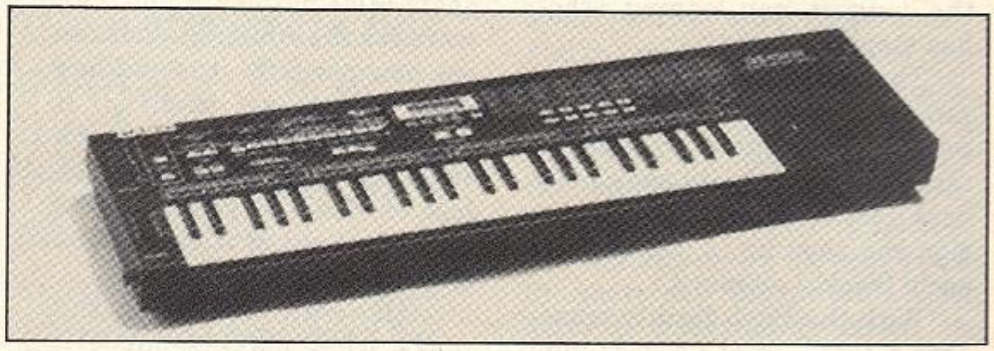
There is only one screen — this keeps things simple — and fashionable icons are used to select options. Some excellent demonstration files are supplied for you to play. You'll either go green with envy or they'll spur you on to create great things yourself.

The Performer lets you record on up to eight tracks which are in some ways similar to tracks on a multi-track tape recorder.

You can do a lot more with these digital tracks, though, than audio ones. The six main options are: Play, Channel, Loop, Control, Pitch and Arrange. We'll look at these one at a time and you will see how powerful Midi can be.

Each line or track of music can be assigned to a different Midi channel so on playback each can be given a different sound. This is where a multi-timbral synth comes in handy and the Performer works well with CZ synths.

Recording is a straightforward process but obviously you need a keyboard (Yamaha's keyboardless FB-01 would be unsuitable



The Casio CZ230S is an economically price Midi keyboard

but it could play back your recordings).

If your keyboard skills are not as good as you would like, the Performer can help in several ways. You can record a piece slowly then speed it up on playback. There is a Punch-In facility which lets you drop into a track and record from that point onwards.

There is also a Time Correct feature, technically know as quantisation which puts notes "on the beat" as you play. The resolution can be altered from quarter noes to 32nds notes. For many musicians, this feature is one of the most useful aspects of Midi.

The recording process is similar in many ways to audio recording but you build up a series of individual tracks inside the computer instead of on tape.

You can bounce tracks too, a process which the Performer calls track merging. This puts all the note information into one track. Unlike audio recording that track an only be played through one channel so it's no good merging a bass line and a lead line as they will play with the same sound.

You can make a track loop in which case it will repeat until the longest non-looping track has stopped playing.

The Control option is used to remove unwanted performance control data. The storage of aftertouch and velocity sensitivity uses a lot of ram and their omission can save up to one third of each track's memory.

With the Pitch facility transpose a track up or down by any number of semitones. The Arrange option lets you do just that.

If, for instance, you had a bass line on track 1, a rhythm accompaniment on track 2 and a solo line on track 3, you could arrange track 1 to play as an introduction. If you wanted to bring in the rhythm, the next arrangement would be tracks 1 and 2.

The third arrangement, tracks 1, 2 and 3, would bring in the solo line and arragement 4 could simply be tracks 2 and 3 — solo and rhythm without the bass. It's far easier than it sounds and you can create up to 64 arrangements.

There are other features too, such as a metronome, a count-in (so you start each part at the right time), adjustable tempo, a repeat option (the piece can play up to 255 times) and a text option so you can store a few words about the piece — ideal if, like me, you never keep notes.

EMR is constantly developing new programs and an Editor, Notator (to print out the music) and step-time Composer (input notes one at a time) are under development.

Computer music is a growth area and now at last we have some Midi hardware and software for the Amstrad. You can see how powerful it is and how easy it is to get involved. It needn't cost you an arm and a scholarship to the Royal College of Music — you can get a great deal out of it even if you only have one keyboard.

Above all Midi is fun and that's what making music is all about. Plug in today!

UTILITY

continued from page 45

such as to change a window destination, will not work unless the CALL to *closem* is made first. The line–

```
|WRITEM,1,3: LOCATE 3,3: PRINT "m":
LOCATE 4,3: PRINT "n":CALL closem
```

will work, but–

```
|WRITEM,1,3: LOCATE 3,3: PRINT "m":
|WRITEM,2,3: LOCATE 4,3: PRINT "n":
CALL closem
```

will not, without a:

```
CALL closem
```

before the second RSX command.

The fifth and last command in the PULLM set is CLOSEM, the one we've just met briefly and which restores normal printing and resets LOW JUMP back to its proper value. Unfortunately when |WRITEM is called it will also interfere with the RSX mechanism and that means that CLOSEM cannot be an RSX.

Instead CLOSEM has to be a simple CALL, but to make life easier the address to call is picked up in the variable *closem* – note no | – when the program is set up. If you forget and run the program and lose your variables it is still there as HIMEM+1 – unless of course you have also moved HIMEM.

To see PULLM in action, once you have PULLMC.BIN and PULLMRSX.BAS safely on disc reset the computer and run PULLMRSX. That will set up all the routines and it can then be deleted – but don't use NEW or you will lose the value in *closem*.

Type in Program 3 and save it. Don't use RUN, but GOTO 10 to prevent the value of *closem* from being reset. Now PULLM will show off its tricks, including a piece of simple but impressive animation.

Al's beat

Alan McLachlan prowls the whacky whimsical world of the Amstrad CPC464

You know folks, one of the most entertaining times of the day in the editorial office is the morning mail session. To the accompaniment of the sparrows coughing outside I do a quick scan of your letters, separating them into categories such as Postbag, submissions for publication, comments and suggestions, and listing queries.

This is done so that later, when everybody has come down off the ceiling (editor included) the A Team can tackle the letters together.

During the last few months we've received sackfuls of mail on all sorts of topics. Particularly enjoyable are those that praise the magazine content, those that offer constructive criticism and suggestions for improvement and in particular those that contain praise for the games written by amateurs not on our staff (as opposed to the amateurs on the staff).

Queries from people who have typed in listings incorrectly and so can't get the programs to run have introduced us to a host of wonderful characters whose existence we'd never suspected.

I'd like to introduce you to some of these by quoting some short sections from letters, and if on reading them you recognize yourself, please don't take offence as none is intended.

The first character is the **chess player** — *"I always check everything thoroughly, I've checked and double checked but . . ."*

A variant is the **railway employee:** *"I've checked every line and there isn't a single fault to be seen"*. He's probably been reading Mike's Z80 articles. We usually reply that he's lost track of the listing and he normally gets the point.

Our favourite is the **typist:**

"As I have years of experience using a typewriter the chance of any errors being mine are exstreamly unlikely" .

That one's followed closely by the **postman:***"I've checked every single letter and I can't find anything wrong"*.

More dubious is the **pot smoker:** *"I've passed it around among my friends and they all agree there's nothing wrong with it"*.

Rather more serious is the **doting parent:**

"I got my Johnny an Amstrad of Xmas and he's had a lot of fun with it until he typed in your Trapper. He has spend hours typing it in and it's not fair as he cannot get it to work. There must be a misteak in it as it keeps coming up 'Syntax Error' all over the place".

He's the one who says everyone's out of step but our Johnny.

A similar case is the **iceskater:** *"I've been backwards and forwards over it till I'm blue in the face, and I'm still no further"*.

And, of course, the inevitable **accuser:** *"Having tried all weekend to get your program to run, I'm sure you put deliberate errors in to make people buy your monthly tapes."*

A more dreaded character is the **improver:** *"Your program does not work properly. The following additions will stop the Grumpies going outside the maze"*. (They never did on the original version, but you try convincing him).

As a Speilberg freak the editor is fond of the **film fan:** *"We've searched high and low for Gremlins, but can't find any. They seem to be hidden away in the listing and we're just not good enough to find them"*.

Of course there's the **pacifist:** *"We are not normally the type of family to complain but all we have is 'Arguments' all over the place"*.

We all like the **Rentokill man:** *"I've got rid of all my bugs but I think you must still have some, and I can't find them"*, and the **motor maniac:** *"I've run over it time and time again and I'm sure I'm right"*.

These are closely rivalled by the **search party:** *"We've checked every entry and there's nothing unusual to be seen"*, and the more evil **masochist:** *"I've made a painstaking check of my listing and can only assume your's to be in error"*.

There's the **out of work GP:** *"I must admit I have run out of patience"* (think about it) and the **racing driver:** *"I've been through it time and time again and it's driving me round the bend"*.

Saddest is the **optimist:**

"I have been told by your telephone receptionist to write in as I am desperate. I have typed in four of your games and can't get any of them to work.

"I have saved them all on cassette and ask you to look at them for me and find out where I've gone wrong. I am an absolute beginner and don't know where to start looking for errors".

At least this last lady admitted that the faults were probably hers. Unfortunately we just don't have the time to provide a debugging service, and certainly not over the phone.

Out of curiosity I took this particular cassette home one weekend and found in the four programs no less than 47 typing errors and four missing lines.

It took me 10 hours looking carefully through the listing and checking every entry against that in the magazine. Apart from four errors which I couldn't find without some useful debugging techniques, all the mistakes were plainly visible.

OK, I agree that to an experienced eye they will be more obvious than to an inexperienced one, but they were there to be spotted with careful scrutiny.

I'm not going to keep the debugging techniques to myself and intend, with the assistance of the rest of the A Team, to continue with a series of articles on hints and tips to find your own programming errors.

Here is an example to be going on with. Let us assume you have typed in line 30:

```
3Ø MODE 1:INK Ø,1:INK
1,24:INK
2,2Ø:INK 3,6:PEN 1:DIM
m$(8):GOSUB1ØØ
```

Your micro will respond "Syntax error in 30" and the mistake should be fairly obvious — GOSUB 100 needs a space.

But what if there were more statements in the line and it were not so obvious? A simple hint is to split the line in two:

```
3Ø MODE 1:INK Ø,1:INK
1,24:INK 2,2Ø
31 INK 3,6:PEN 1:DIM
m$(8):GOSUB1ØØ
```

Now you will get "syntax error in 31". This can now be split:

```
32 INK 3,6:PEN 1
33 DIM m$(8):GOSUB1ØØ
```

and the result will be "Syntax error in 33".

Gradual elimination will narrow things down until you are left with the offending statement on its own. Your micro can't tell you what is wrong with it but with careful checking and perhaps experiment you should come up with the solution.

This is a fairly easy technique which can be used for many error messages. We'll look at some more another time.

By the way, if all your debugging sessions have failed and you have been the postman, the pot smoker, the ice skater and any other of my friends, then drop us a line. But a few words of advice:

- Don't expect an immediate reply. We're as fast as we can be, but we are snowed under.
- Do include a self-addressed envelope complete with stamp.

We can't promise to solve all your problems, but we try out best or at least admit defeat. The request for an SAE also applies to any other type of mail if you want a personal reply.

Well that's all for the moment – now I'm going to get some practice at finding some of your errors, which are amazingly like my own.

Public Domain

with Shane Kelly

Hello again. This month we are going to devote most of our space to extended machines, that is machines that are running:-

a) Extended memory (dktronics or other compatible)

b) CPM + or a version of CPM 2.2 that uses the extened memory

c) 2 disk drives (or one if you are using CPM+)

This means that as an absolute minimum we need a 464 with DDI — 1 and extra memory with CPM+. to run this months selection of programs. Actually they are a repeat of two popular programs showing considerable improvements in the user inerface over the originally.

Ths first of the months programs is FATCAT, the fatter but faster disk catalogue program. This program, is considerably easier to use than Ward Christensen's original CAT program, but it is written in Turbo-Pascal and that is a notoriously memory hungry language, hence the need for the larger TPA for this program. Fatcat is designed so that you can simply insert disk after disk without waiting for the computer to update the catalogue. Then, when all the disks have been read, the update portion of the program is invoked and while the computer is doing this lengthy task, you may do something else.

First things first. Fatcat as supplied will run on the Amstrad CPM+ terminal emulation. But you will need to configure it for your number of disk drives and to achieve the most convenience from the program. I know that documentation is usually the last before you start is there. OK, thats out of the way. Now, I don't know how many disks you have, but I have over 100 5.25 disks and I can never remember where I put a particular file after about 3 weeks of not using it. So for me a catalogue is an absolute necessity. What I have done with Fatcat is to create several catalogues that match the category of the files in it. For instance, all my PD programs are in the PUBLIC catalogue while all my utilities are in the UTES catalogue. Fatcat allows this flesibility and in fact I find it encouraged you to keep your disks organised more than is normal.

The other thing I like about Fatcat is the fact that you can actually alter the disk before you catalogue it, using Fatcat's clean-up mode. In this Mode you can delete files, rename them, check their size in k's, and add a file. This is most useful as for this program to work properly, all disks must be given a name (usually starting with —) and an extension which must be numberic and 3 digits long, with all blanks filled by zeros. For instance, -FRED.001 is OK, but -FRED.01 is not as the extension is not fully expanded. To help in the search for a file, I have split my catalogues into areas, with a range of extension numbers allocated to each catalogue. My PD catalogue is from 001 to 299, allowing 300 disks in this catalogue, (if I don't stop collecting PD software, this won't be enough!!!) my UTES catalogue is allocated 300to 399, and so on. This makes searching for a particular file faster, as Fatcat uses and it is fairly obvious what to do, but *please* print out and read the .DOC file before you start as this is a sophisticated piece of software that will repay study of the instructions.

Our second program is a dissassembler that is also so easy to use that it is now my main weapon in the fight to customise software for the Amstrad from the public domain. DAZZLESTAR or DZ for short uses the wordstar user interface which is nice if you know wordstar, just a pain if you don't.

For many of you who are not programmers the inclusion of yet another dissassembler may provoke groans of annoyance. That's only because you can see no use for it. If you have obtained any PD software that does not work properly on the Amstrad screen then you may be able to patch it using a dissassembler, given a little help from the local user club. Also, you sometimes see in magazines articles telling you haw to "patch" a program so that it works smarter or better or faster or just works. A dissassembler is not the usual way to do this sort of work, but it can provide an insight into how the program works. If nothing else, it may convince you that machine code is not impossible to bring to heel.

DZ is a particularly good example of something that is user friendly written under CPM. It has help menus and a large DOC file that explains how to use this program to the best advantage. It also has a "typical use" section for those who are dipping their toes into machine code for the

first time. As always, the best way to learn is to experiment, after reading the documentation!!

Our last offering for you extended users this month is AUSOPOLY.BAS, an MBASIC (and hence a Mallard basic) program that play monolpoly with up to 10 players. The computer may take as many players as you like and will even play itself if you so desire. I had quite a bit of fun with this program, but I couldn't win. If you are new to programming, have a look at the way AUSOPOLY is written, and note the usually single statement lines, the use of remarks and the way it is laid out. It may help you if you are struggling to understand BASIC.

That's all for this month. Next month I'll be trying to get a bit more software for the minimal type system but I still do not have much feedback from you out there, so I guess you'll just have to put up with my choice, won't you?

What MORE would you like?

Well, it's good to see our second issue of Computing With The Amstrad just about ready for you. And many thanks to all those who've written and phoned to express their support for the magazine – its content, and its new look. The new AMTIX section, this time with book reviews as well, is proving very popular, and we're looking at expanding this section.

Now that we're (touch wood) over the worst of the hiccups of taking over CWTA it's time to look more closely at the balance of articles in each issue – and would welcome your feedback on this. What articles do you enjoy the most? Graphics? Amtix? Maybe more on Sound ... or perhaps less Series and more features? Or vice versa? Waddayarekn?

Let us know – the address is at the bottom of this page – after all, we work on this magazine for you! Drop a line now – if others think the same way you do you're sure to see more of the sort of articles that you want.

Happy hacking...

Rob McKenzie

POSTBAG

July's Game of the Month – Roulette requires some corrections.

Line 380 should be –
spin=0:num=RND*36
(NOT 37)

Line 620
FOR i=0 TO 1:MOVE
16+(i*336),10:DRAW
16+(i*336),366:NEXT

Line 1920
1=36:h=0

In description panel for terms of betting – Douxaine select either 1, 13, or 25, not any number in that range.

Also Deux Douxaine 1 or 13
Pass?Manque 1 or 19

Computing With The Amstrad

'Computing With The Amstrad' welcomes program listings and articles for publication. Material should be typed or computer printed, and must be double spaced. Program listings should be accompanied by cassette tape or disk. Please enclose a stamped addressed envelope or the return of material cannot be guaranteed. Contributions accepted for publication by Database Publication or its licensee will be on an all-rights basis.

© Database Publications and Planet Publishing Pty Ltd. No material may be reproduced in whole or part without permission. While every care is taken, the publishers cannot be held responsible for any errors in articles, listings or advertisements.
'Computing With The Amstrad' is an independent publication and neither 'Computing With The Amstrad' and neither Amstrad plc or Amsoft or their distributors are responsible for any of the articles in this issue or for any of the opinions expressed.