Computing with the AMSTRAD

December, 1986
$3.60
A Database Publication

The independent magazine for Amstrad computer users

Registered by Australia Post – Publication No. VBP7811

Robot Ron and the Ice Monsters

By STEPHEN MARTIN

Regulars

First Steps : Basic for beginners
Sound : Learn all about the pitch envelope
CP/M : CCP keywords
Public Domain : All about communications

New!

Two new series start this month featuring Graphics and Machine Code - we'll make it easy to learn!

Listings

Robot Ron meets the Ice Monsters
Santa's Grotty
Word Count for Locoscript
Archiver - back up those disks!
Rem Strip - reclaim your memory
Full Forth listing
Reaction Timer - test your speed!

And lots more so get in now!

ALL NEW TITLES AT SPECIAL PRICES

NEW AMSTRAD TITLES AVAILABLE ONLY FROM STRATEGY SOFTWARE

1. GENESIS ADVENTURE CREATOR
DISK ONLY $39.95

Don't pay more! Genesis offers a complete system for writing Text/Graphic adventures with Music and Sound Effects. Features include: Text Compression, synonyms for commands and objects, sentence analyser, multiple graphic windows, variable screen modes, extensive graphic commands and storage of text and graphics on disk for large, disk-based adventures. Ideal for the novice wishing to create his/her own adventures. Used by commercial software houses in the U.K.

2. GRASP PLUS
DISK ONLY $29.95

All the features of Grasp, plus hi-res (mode 2) screen plots, improved labelling, various printer options, faster screen dumps, exploded pie charts and all the advantages of disk operation.

3. MUSICO
TAPE ONLY $17.95

An easy to use system for creating 3 part music and sound effects. Results may be used in your own programs.

4. DRUMKIT
TAPE ONLY $16.95

Percussion Instrument Simulator with eleven standard percussion sounds plus three user-defined sounds. Uses the Amstrad sound chip, no extra hardware required.

5. CHAOS FACTOR

TAPE $15.95
DISK $27.95

A graphic adventure game. featuring Nurd Fungus and Narsty & Crutch, the amazing cops who can solve any crime simply by saying 'Hi' a lot! Don't miss it!

6. EASY MUSIC
DISK ONLY $34.95

A combination of MUSICO and DRUMKIT (see above) together with the advantage of disk operation.

7..**AMSTRAD POT-POURRI VOLUME 1**

TAPE $ N/A
DISK $ 19.95

25 great games for your Amstrad. Fantastic value for money at less than 40¢ per program on tape. Volume 1 has a full-length text adventure which alone is worth the money!

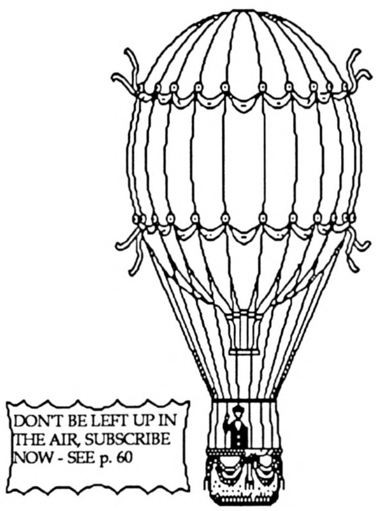
8. **AMSTRAD POT-POURRI VOLUME 2**
TAPE $ N/A
DISK $19.95

25 more previously un-released software on the same basis as Volume 1.Volume 2 includes a great machine-code Space Invaders.

EXTRA SPECIAL!!!

Buy both Volumes 1 & 2 and receive, absolutely free, a tape copier and tape to disk utility. Please note that these programs cannot be guaranteed to work with all tapes.

CALL [002] 29 4377 NOW!



Computing With The Amstrad

December 1986

Published monthly by Strategy Software under license from Database Publications Limited.

Telephone: [002] 29 4377
Telex: AA58134 (Attn. HT163)

Mail: P.O. Box 11
Blackmans Bay
Tasmania 7152

Annual Subscription (12 Issues)

Australia $40
New Zealand & South Pacific $A60

'Computing With The Amstrad' welcomes program listings and articles for publication. Material should be typed or computer printed, and preferably 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.

Contents

Features

Utilities

Contents

Computing With The Amstrad has moved.

Computing With The Amstrad (and, of course, Strategy Software) has moved to Tasmania. As a result of this move we now have more room in which to work and employ more staff to service your needs. Improvements in service will become apparent in the early part of 1987 and we ask for your indulgence during our re-organization period. Please take note of our new address and telephone number.

Count your blessings

JOHN MILSON writes a word counter for LocoScript

ALMOST a year has gone by and we still haven't seen any add-ons for LocoScript. One of the facilities sadly missing from it is a built-in word counter, and though such programs are available commercially for the PCW they all come as part of other packages like spelling checkers and alternative word processors.

So until now if you didn't feel like spending money on software you don't really need, you had to be content with counting your blessings and making rough estimates of the number of words in a file.

But here's a word count program offered by *Computing with the Amstrad* for free. It's written in Mallard Basic, which every PCW owner has. It won't handle an unmodified LocoScript file – there isn't yet a program of any description that will – but it can be used with an Ascii file created using the f7 option on the disc management screen. Use the simple Aciii, not the page image option.

First type the Basic program into the machine and save it on a blank, formatted disc with the name WCOUNT. Then using the Pip utility copy from the system disc the files J14CPM3.EMS, BASIC.COM and SUBMIT.COM.

Should you have another version of CP/M, then its filename would be slightly different. The name would also need to be corrected in line 220 of the program.

Now using the Basic RPED utility set up a file with the name PROFILE.SUB containing the single command line BASIC WCOUNT. The use of these utilities is described in the manual. The word counter is now complete, with all the necessary files on the one disc.

Let us suppose that you are using the word processor and want to count words. First you have to copy your document into the first group of drive M, using the f3 option of the disc management screen. If you have two disc drives or a PCW8512, this step can be bypassed.

Next you replace the LocoScript disc with your word counting disc, not forgetting to press f1. Then you make a simple Ascii text file, using the menu called up by f7. The destination of this must be the first group – Group 0 – of drive A.

This procedure is explained in the document called READ.ME. Now you have a suitable version of your document on the same disc as the word counting program.. Just reset the machine with Shift+Extra+Exit and watch things happen.

The screen should come to rest showing an option menu and notes, and the word counter is now ready to use. Just follow the instructions, requiring merely a few single keystrokes.

The use of the program could be simplified a little by modifying it to run, after resetting, without a pause through the file finding and word counting routines. However as it is you can handle documents in batches if you wish, or if you have a very large document – say, more than 90k – you can put it on a disc of its own.

PROGRAM STRUCTURE

220	FNprog(f$) returns zero unless *f$* holds the name of one of the program files. This is used to hide from view the files that do the work.
1080	Mallard Basic automatically adjusts the return stack when a jump is made out of a loop. In the absence of a Pascal style block structure this seems the easiest way to handle these options.
2030	Gives you something to pass the time while the machine is churning – calculating how long it will take seems as good an occupation as any.
2220	This seems a good place to force a garbage collection, while you are noting the result. It might save a little time if a second document is to be counted.
3010	Names of read-only files have bit seven set in one character. This line forestalls a Basic error.
4100	Retry would repeat the error. Cancel would return you to the system, losing Basic. Ignore returns you to Basic where the program can easily be restarted using RUN.
5000	Files must not be left open. As the return from the error routine is not back into the subroutine where the error occurs the return stack must be cleared by changing its size. Merely restating the size won't do, hence the two memory statements.
5060	Allows the printing of the Basic error message.

From Page 6

```
10 ' ************ WCOUNT ************
20 '    WORD COUNTER FOR LOCOSCRIPT
25 '             DOCUMENTS
30 '    * John Milsom April 1986  *
40 '      *   East Grinstead    *
50 ' ********************************
60 '
70 ON ERROR GOTO 5000
80 WIDTH 90
90 '
180 ' **** "Print" controls etc. ****
190 '
200 esc$ = CHR$(27): bel$ = CHR$(7)
210 hom$ = esc$ + "H": cl$ = esc$ + "
E": cls$ = cl$ + hom$
220 DEF FNprog(f$)=INSTR(1,"J14CPM3.E
MS,BASIC.COM,SUBMIT.COM,PROFILE.SUB,W
COUNT.BAS",f$)
230 PRINT cls$: GOSUB 4000
240 '
470 '
480 ' ******** Option Point ********
490 '
500 WHILE -1
510 PRINT bel$; "Press key for requir
ed option.";
520 IF file$ = "" THEN PRINT "  (You
need to choose a file. Option 1 or F)
" ELSE PRINT " To show Menu, press 0
or M.": PRINT"(Current file is ";file
$;")"
530 k$ = ""
540 WHILE k$ = "":k$ = INKEY$:WEND
550 PRINT cls$
560 ON INSTR(1,"0MmiFf2Cc3Ee",k$) GOS
UB 4000,4000,4000,1000,1000,1000,2000
,2000,2000,3000,3000,3000
570 PRINT
580 WEND
590 '
970 '
980 ' ******* 1. Find file *********
990 '
1000 n%=1
1010 PRINT "Do you want to do anythin
g with this file (y/n)?"
1020 f$ = (FIND$("*.*",1))
1030 WHILE f$ <> ""
1040 IF FNprog(STRIP$(f$)) <> 0 THEN
GOTO 1100
1050 PRINT f$;"  ?  ";
1060 k$=""
1070 WHILE k$="":k$=INKEY$:WEND
1080 ON INSTR(1,"YyXx",k$)GOTO 1120,
1120,1130,1130
1090 PRINT "No"
1100 n%=n%+1: f$=(FIND$("*.*",n%))
1110 WEND: RETURN
1120 file$=f$: PRINT"Yes":RETURN
1130 PRINT "No": PRINT: PRINT "Search
abandoned": PRINT: RETURN
1140 '
1970 '
1980 ' ** 2. Word counting routine **
1990 '
2000 IF file$= "" THEN PRINT "You nee
d to find a file name (Option 1 or F
(.": RETURN
2010 PRINT "      Words in   ";file$;"
being counted."
2020 PRINT TAB(20); " Please wait."
2030 PRINT "Approx. 12 seconds for ea
ch kilobyte of original document."
2040 '
2050 ' Initialise counting variables
2060 '
2070 wc%=0: l%=0: c%=0
2080 '
2090 ' ********** Count *************
2100 '
2110 OPEN "i",1,file$
2120 ch$ = INPUT$(1,1)
2130 WHILE NOT EOF(1)
2140 IF INSTR(1," Xx",INKEY$)>1 THEN
CLOSE 1: PRINT: PRINT "Count abandone
d": PRINT: RETURN
2150 IF ASC(ch$) <= 32 THEN c%=0 ELSE
c%=1
2160 IF l% = 0 THEN IF c% = 1 THEN wc
% = wc% + 1
2170 l%=c%
2180 ch$ = INPUT$(1,1)
2190 WEND
2200 CLOSE 1
2210 PRINT bel$;cls$; "Number of word
s in  ";file$;" = ";wc%
2220 free = FRE("")
2230 RETURN
2240 '
2970 '
2980 ' ******* 3. Erase File ********
2990 '
3000 IF file$ = "" THEN PRINT "First
please find file name (Option 1 or F
).": RETURN
3010 IF file$ <> STRIP$(file$)THEN PR
INT "You are not allowed to erase thi
s file": RETURN 3020 PRINT cls$;"Are
you sure you want to erase  ";file$;
"  ? (y/n)"
3030 k$ = ""
3040 WHILE k$ = ""
3050 k$ = INKEY$
3060 IF k$ <> "" AND INSTR(1,"Yy",k$)
<> 0 THEN KILL file$
3070 WEND
3080 PRINT cls$
3090 RETURN
3100 '
3970 '
3980 ' ********** 0. Menu ***********
3990 '
4000 PRINT "Menu of Options": PRINT
4010 PRINT "0. Menu: Display the opti
ons"
4020 PRINT "1. Find the file you want
 to use"
4030 PRINT "2. Count the number of wo
rds in the file"
4040 PRINT "3. Erase the file"
4050 PRINT: PRINT "To call option, pr
ess option No. or initial letter."
4060 PRINT: PRINT "To abandon operati
on in progress, hit x": PRINT
4070 PRINT "It is a good idea to eras
e files created especially for word c
ounting"
4080 PRINT "after they have been used
, in order to avoid cluttering the di
sc.": PRINT
4090 PRINT "If, on trying to erase a
file, you get a message at the bottom
 of the screen"
4100 PRINT "offering the options: Ret
ry, Ignore or Cancel, press I and ch
eck your disc."
4110 PRINT "You'll probably find it t
o be write-protected.": PRINT
4120 RETURN
4130 '
4970 '
4980 ' ******* Error routine ********
4990 '
5000 PRINT: CLOSE: MEMORY,256: MEMORY
,512
5010 IF ERR = 53 THEN PRINT file$;
"  not found on this disc.": PRINT: R
ESUME 500
5020 IF ERR = 62 THEN PRINT file$;
"  is an empty file.": PRINT: RESUME
500
5030 IF ERR = 64 THEN PRINT file$;
"  is not a valid file name.": PRINT:
RESUME 500
5040 PRINT bel$;"Error No. ";ERR;". T
his error was not foreseen."; bel$
5050 PRINT "Try to restart with 'RUN
<RETURN>'";bel$
5060 ON ERROR GOTO 0
5070 END
```

Third in MIKE BIBBY's guide through the micro programming jungle

WE saw last month how to write our own programs, however primitive. Now we'll look at some ways of improving them. I don't guarantee that you'll be able to produce spectacular programs by the end of this article, but you will certainly be well on the way to an understanding of Basic.

First, though, let's recap a little: We saw last month that a Basic program consists of a numbered sequence of instructions to the computer.

To enter one of these instructions we simply type the correct line number, followed by the appropriate Basic keyword, then press Enter.

As we discovered, because of the line number the Amstrad doesn't do what you tell it immediately but remembers it as part of the program.

To see all the instructions in a program we type:

LIST [Enter].

To actually get the Amstrad to carry out the sequence of instructions we type:

RUN [Enter].

To clear a program from memory (and we should do this before entering a new program) we use:

NEW [Enter].

We saw that we tended to enter line numbers in steps of 10 to allow us to fit in other instructions between them if necessary. Also we found that we could replace a line with a better version by simply giving the new version the line number of the old one.

Finally, to delete a line completely we simply type the line number and press Enter.

Program I is the one we started with last month. Before we continue, type it in and run it, to make sure you know what's going on:

```
10 PRINT "PROGRAMMING"
20 PRINT "IS"
30 PRINT "EASY"
```

Program I

Program II is another way of achieving exactly the same output. Type it in and try it.

Apart from its being an incredibly long-winded way of doing things, what else is going on?

```
10 A$="PROGRAMMING"
20 B$="IS"
30 C$="EASY"
40 PRINT A$
50 PRINT B$
60 PRINT C$
```

Program II

Well, as you will recall from the first article in this series, the words inside quotes are known as strings – because the computer simply remembers them as strings. That is, it considers HAMSTER as H, followed by A, followed by M and so on, with no idea of the word's meaning.

I don't think that it takes all that much imagination to see that when your computer is printing a lot of output you might be using the same string rather a lot.

For example, in a business letter you might use the name of the company fairly frequently – for example, BBC for British Broadcasting Corporation. The Amstrad's Basic allows us to use much the same idea, but more as labels than abbreviations.

For instance, in line 10 of the above program we have labelled the string "PROGRAMMING" with the label A$.

In computer terms we have assigned to A$ the value "PROGRAMMING".

All this means is that from now on wherever I want to use "PROGRAMMING" in my program I can replace it with A$. So line 40, which is

40 PRINT A$

causes the micro to print out "PROGRAMMING".

Admittedly in this example this technique of labelling doesn't save much space or effort, but if the program uses the word "PROGRAMMING" 100 times there would be a substantial saving in using A$ instead of the string itself.

Similarly, line 20 causes B$ to label "IS" and line 30 labels "EASY" with C$, so that lines 50 and 60 give the appropriate printout.

Notice the following points:

- We have chosen our labels so that

they consist of a letter of the alphabet followed by the "$" sign. Actually, we don't have to restrict ourselves to just one letter, as we shall see, but our label must end with the "$" sign, since this warns the computer that we are labelling a string. (We'll see later how to label other things.)

- While I used A$ for the first label, B$ for the second and C$ for the third, this was totally arbitrary on my part – labels don't have to follow alphabetic or any other kind of order.
- Although we use an equals sign ("=") to connect the label with what it is labelling, it is safer, as we shall see, not to think of it as an equals sign – think in terms of *A$ becomes "PROGRAMMING" rather than A$ equals "PROGRAMMING".*
- We must have the label on the left and what is labelled on the right of the equals sign. A line such as:

10 "PROGRAMMING" = A$

just does not make sense to the CPC464. Try it for yourself!

- When labelling we put the string inside quotes, as we did previously when using the PRINT statement to print out strings. So line 10 reads:

10 A$ = "PROGRAMMING"

From now on A$ completely replaces "PROGRAMMING", quotes and all, so that when we say

PRINT A$

we don't have to use any quotes – they're already there, implicit in the label A$.

Have a look at Program III. It's virtually identical to Program II except for lines 40 to 60. Here, instead of using A$,B$ and C$, we use the lower cased versions, a$, b$ and c$.

```
10 A$="PROGRAMMING"
20 B$="IS"
30 C$="EASY"
40 PRINT a$
50 PRINT b$
60 PRINT c$
```

Program III

However when you run the program this makes no difference – the output is the same as in Program II. This is rather odd – you have, for instance, given a value to A$ in line 10, and managed to print it out using a$, in line 40!

The point is that as far as the Amstrad is concerned labels that contain the same letters are identical – whether they are in upper or lower case. So:

```
PRINT A$
```

is the same as

```
PRINT a$
```

Beware – not all micros are like this . . .

Now when we label a string the label refers to whatever is inside the quotes, including spaces, as you will see if you run Program IV:

```
10 REM PROGRAM IV
20 MODE 1
30 A$ = "TEST"
40 B$ = " TEST"
50 C$ = "  TEST"
60 D$ = "   TEST"
70 PRINT A$; B$; C$ ;D$
80 PRINT"01234567890123456789012345678901234567890123456789"
```

Notice that our punctuation – semicolons and apostrophes – works for labelled strings just as it worked on its own.

Notice also that we have introduced a new Basic keyword in line 10 – REM. We use REM, which is short for REMark, to add comments or headings to our programs.

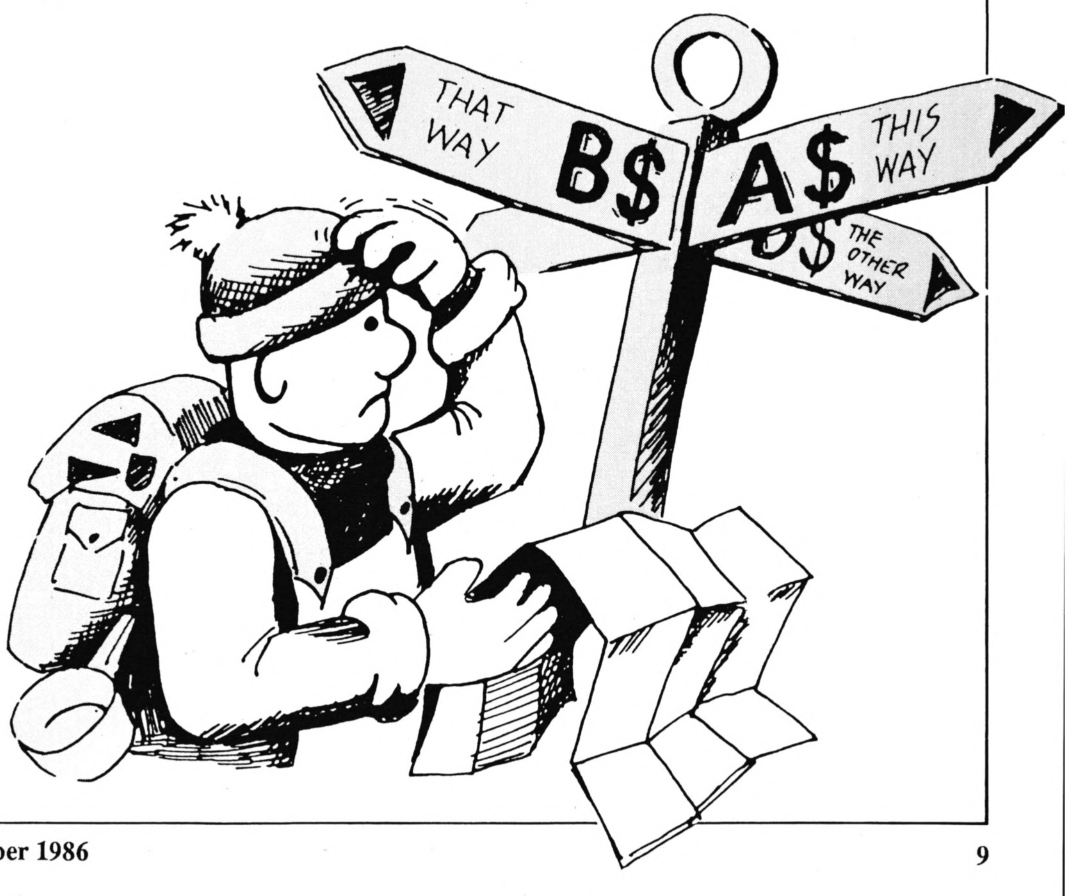
When the Amstrad encounters REM in a line it ignores everything else after it on the same line. This means we can write whatever we want after REM (providing it is on the same line) without fear of the micro giving us an error message – the CPC464 doesn't "read" the line beyond the REM.

If we use REM to prefix our comments on the program we can annotate our program. Certainly each main subdivision should have one or more REM statements explaining what is going on.

Since the Amstrad ignores the contents of REM statements you could leave them out of your program entirely and it will work as effectively. However it is good programming practice to include them.

In the program examples I have used a single REM at the beginning of the program, as it is so short. Bear in mind however, that REM can appear on any line in a program.

Now for some jargon. From now on we shall refer to our labels as variables. Don't be put off by the mathematical sound of that – they are still just labels! And instead of saying we are labelling, we say we are assigning, as we have mentioned



previously. The actual string involved is known as the value of the variable.

So:

A$ = "TEST"

reads *"the string variable A$ has assigned to it the value 'TEST'"*. The actual act of giving a variable a value is called an assignment.

To return to the world of actual programs, you can mix and match string variables and actual strings however you want. Program V illustrates the point:

```
10 REM PROGRAM V
20 MODE 1
30 A$ = "MY NAME IS"
40 B$ = " MIKE"
50 PRINT A$; B$
60 PRINT "MY NAME IS";B$
70 PRINT A$;" MIKE"
```

Notice the space of the beginning of the string assigned to B$ – you need this otherwise the output looks rather odd. Leave it out if you don't believe me!

'you can mix and match string variables however you want'

As we saw last month, a semi-colon at the end of a line causes the next output to start immediately after the last and not on a new line – as it would do in the absence of the semi-colon. That is, it "glues" the strings together.

The internal semi-colons of lines 50, 60 and 70 do much of the same, "gluing" variables to strings, etc.

While this is grammatically correct Basic, the Amstrad assumes (unless you tell it otherwise) that variables and strings mentioned in the same PRINT statement are meant to be output continuously on the same line. To prove this run Program IV omitting all the semi-colons.

You've got to be careful here, though. If you typed line 50 as:

```
50 PRINT A$B$
```

– that is, with no space between the variables – the program would still work. This is because the Amstrad recognises the "$" as a delimiter of the string.

Be careful of running variables together like this though. It can cause problems later – and it makes your programs very hard to read. Stick to:

```
50 PRINT A$ B$
```

if you want to do this sort of thing.

Also, while we're on the subject of grammatical propriety, when we're assigning variables we should use the LET statement. So line 40 should read

40 LET B$ = "MIKE

As you've already discovered, we can omit LET altogether.

● *Next month, more on variables and INPUT – which opens the door to effective programming.*

THINK SLIM

Pack more into a column with ROBIN NIXON's character cruncher

HAVE you ever wanted to have the use of 16 colours in Mode 1, or wanted 40 columns per line in Mode 0? Well with this utility you can have both. And, as well as that, if you've got very good eyesight, you can have 160 columns per line in Mode 2!

"How is this possible?" you may well ask. The answer is simple. As you know, Amstrad characters are defined by an 8 × 8 grid which can be re-defined using the SYMBOL command.

However with a little cunning it is possible to squeeze a complete character set into a 4 × 8 grid.

This is exactly what I've done in Program I. The left hand sides of characters 32 to 127 have been re-defined and the right hand sides have been left blank.

This means that once one character in a string has been placed on the screen the next character can be placed over the right hand side of it, over writing the blank part of the first character, thus allowing for twice the number of characters per line.

Program II is a demonstration of how to use the new definitions. It first sets up four variables:

ZM The current screen mode.
ZX, ZY The X and Y coordinates of where text is to be printed. As with LOCATE X,Y.
ZZ$ The string you wish to print.

It then GOSUBs 60000 in order to display the slim characters.

So, to use Slim Characters in your program – a game for example – first type or load in Program I, the one containing all the character data. Change the filename in line 310 to that of the program it is to be incorporated in, such as SPACINV and SAVE it.

Now load in your program SPACINV and add line 60000 from Program II to the end of it, along with the necessary extra lines to pass the parameters *ZM, ZX, ZY,* and *ZZ$*. Then SAVE the final version of the game.

Bear in mind that the SYMBOL AFTER 32 command has already been initiated in this utility and if you have used it in your program it must be removed from your original or the slim character set you are trying to use will be destroyed.

```
100 REM ************************
110 REM *                      *
120 REM *  SLIM  CHARACTERS    *
130 REM *                      *
140 REM *   By Robin Nixon     *
150 REM *                      *
160 REM * (c) Computing with   *
170 REM *     the  Amstrad     *
180 REM *                      *
190 REM ************************
200 REM
210 MODE 1:LOCATE 14,12:PRINT"Please
wait...":DEFINT A-Z:DIM A(8)
220 SYMBOL AFTER 256
230 SYMBOL AFTER 32
240 FOR X=32 TO 127
250 READ A$
260 FOR Y=1 TO 8
270 A(Y)=VAL("&"+MID$(A$,Y,1)+"0")
280 NEXT Y
290 SYMBOL X,A(1),A(2),A(3),A(4),A(5)
,A(6),A(7),A(8)
300 NEXT X
310 RUN "SLIMFILE"
320 END
1000 DATA 00000000
1010 DATA 44444040
1020 DATA AAA00000
1030 DATA 0AE0EA00
1040 DATA 4EC46E40
1050 DATA A2448A00
1060 DATA AEA0A4A0
1070 DATA 26400000
1080 DATA 48888840
1090 DATA 42222240
1100 DATA 04E4E400
1110 DATA 044E4400
1120 DATA 00000448
1130 DATA 000E0000
1140 DATA 00000440
1150 DATA 22448800
1160 DATA 4AAAAA40
1170 DATA 4C4444E0
1180 DATA EA2E88E0
1190 DATA EA242AE0
1200 DATA 88AAE220
1210 DATA E8E22AE0
1220 DATA EA8EAAE0
1230 DATA E2222220
1240 DATA EAA4AAE0
1250 DATA EAAE2AE0
1260 DATA 04400440
1270 DATA 04400448
1280 DATA 02484200
1290 DATA 00E0E000
1300 DATA 08424800
1310 DATA EA264040
1320 DATA EAAE8AE0
1330 DATA EAAEAAA0
1340 DATA CAACAAC0
1350 DATA EA888AE0
1360 DATA CAAAAAC0
1370 DATA EA8C8AE0
1380 DATA EA8C8880
1390 DATA EA8AAAE0
1400 DATA AAAEAAA0
1410 DATA E44444E0
1420 DATA E2222AE0
1430 DATA AACCAAA0
1440 DATA 88888AE0
1450 DATA AEEAAAA0
1460 DATA AAEEEAA0
1470 DATA EAAAAAE0
1480 DATA EAAE8880
1490 DATA EAAAAC20
1500 DATA EAAECAA0
1510 DATA EA8E2AE0
1520 DATA E4444440
1530 DATA AAAAAAE0
1540 DATA AAAAAA40
1550 DATA AAAAEEA0
1560 DATA AA444AA0
1570 DATA AAE44440
1580 DATA EA248AE0
1590 DATA E88888E0
1600 DATA 88442200
1610 DATA E22222E0
1620 DATA 4EA00000
1630 DATA 0000000E
1640 DATA EA8848E0
1650 DATA 00E2EAE0
1660 DATA 88EAAAE0
1670 DATA 00EA8AE0
1680 DATA 22EAAAE0
1690 DATA 00EAE8E0
1700 DATA 00E8C880
1710 DATA 00EAAE2E
1720 DATA 88EAAAA0
1730 DATA 40C444E0
1740 DATA 206222AE
1750 DATA 88AACAA0
1760 DATA C44444E0
1770 DATA 00AEEAA0
1780 DATA 00EAAAA0
1790 DATA 00EAAAE0
1800 DATA 00EAAE88
1810 DATA 00EAAE22
1820 DATA 00EA8880
1830 DATA 00E8E2E0
1840 DATA 44E44460
1850 DATA 00AAAAE0
1860 DATA 00AAA440
1870 DATA 00AAEEA0
1880 DATA 00AA4AA0
1890 DATA 00AAAE2E
1900 DATA 00E248E0
1910 DATA 264C4620
1920 DATA 44404440
1930 DATA 8C464C80
1940 DATA AE800000
1950 DATA 00000000
```

Program I

```
10 MODE 0:ZM=0
20 ZX=10:ZY=10:ZZ$="Hello there!!":GO
SUB 60000
30 END
60000 ZM1=2^(3-ZM)*2:ZX1=(ZX-1)*ZM1-Z
M1:ZY1=398-(ZY-1)*16:TAG:FOR ZZ=1 TO
LEN(ZZ$):MOVE ZX1+ZZ*ZM1,ZY1:PRINT MI
D$(ZZ$,ZZ,1);:NEXT:TAGOFF:RETURN
```

Program II

Making capital out of longer variables...

Fourth in MIKE BIBBY's helpful guide through the micro programming jungle

WE saw last month how to label strings with variables. This meant that if we were using a string several times in a program we could use a variable instead of it.

For example:

```
A$="AUSTRALIA"
```

means that, from now on, instead of using "AUSTRALIA" in full in our programs, we can use *A$*.

```
PRINT A$
```

will print out AUSTRALIA for you.

The labels we used last month were all single letters of the alphabet followed by '$'. The dollar sign tells the computer that it is a string we are labelling – such a variable is called a string variable.

It is called a variable because the "contents" of a variable (in technical terms, its value) can vary throughout a program.

Program I should illustrate the point:

```
10 REM PROGRAM I
20 MODE 1
30 A$ = "AUSTRALIA"
40 PRINT A$
50 A$ = "AMERICA"
60 PRINT A$
70 A$ = "AFRICA"
80 PRINT A$
```

Program I

As you will see when you RUN it, the value of *A$* varies as we reassign it during the program. *A$* always takes the last value assigned to it. You may wonder why on earth you would want to use the same variable for different things, rather than label everything separately.

As we shall see, it can be extremely useful.

```
10 REM PROGRAM II
20 MODE 1
30 name$ = "Mr.Smith,"
40 fact$ = "You owe me money."
50 threat$= "Pay up or else."
60 PRINT
70 PRINT "Dear "name$
80 PRINT TAB(5) fact$;threat$
90 PRINT TAB(15) "cordially yours,"
100 PRINT TAB(20) "Mike"
```

Program II

So far we have restricted our string variables to single letters of the alphabet followed by the $ sign, such as *A$, B$* and *C$*.

However there is no need for such a limit – provided we follow them with $. String variables can be made up of several letters, even words.

Program II illustrates the point. It is our most sophisticated program to date, and is well worth having a close look at.

Perhaps the first thing to remark upon is that we are now working in lower case letters as well as capitals. Infuriating as this is at first for the non-typist (myself included), it really is worthwhile.

Notice that in the programs all the Basic keywords are in capitals.

Now the Amstrad forces this on you. Keywords are always LISTed in capitals. If you entered a line such as:

```
10 print "Hello"
```

it would be LISTed as:

```
10 PRINT "Hello"
```

That is, the keyword would be translated into upper case. Notice that there's no such translation of the "Hello". Since this is a string, in quotes, it remains inviolate.

In fact, all the following:

```
10 pRinT "Hello"
10 pRINt "Hello"
10 PRInt "Hello"
```

would be listed as:

```
10 PRINT "Hello"
```

All the variables of Program II *(names$, fact$, threat$)* are in lower case.

This is because I have entered them this way. There's an important point to be made here:

On the Amstrad:

```
10 PRINT threat$
```

and

```
10 PRINT THREAT$
```

would appear different when listed, since the variable would remain in the same case – upper or lower – as when it was typed in. Only keywords are altered.

However, *both lines are equivalent.* The Amstrad doesn't recognise any difference between the variables *threat$* and *THREAT$.* In variable names capitals and lower case letters are equivalent, so:

```
threat$
THREAT$
thrEAT$
```

are considered by the micro to be the same variable.

```
10 REM PROGRAM III
20 MODE 1
30 threat$ = "First"
40 THREAT$ = "Second"
50 PRINT threat$
60 PRINT THREAT$
```

Program III

Program III shows the idea. Line 30 and 40 assign strings to the same variable, despite appearances. The value assigned in line 60 replaces that assigned in line 50 as we saw in Program I.

My recommendation is that you enter all variables in lower case. In this way, when you LIST the program, the lower case variables will stand out from the keywords, which are LISTed in capitals.

This may not make for easy typing, but it is good programming practice, since you can tell at a glance what's what in a program.

Take a close look at those variable names – we are using actual words for the labels in this program. Again, it is good programming practice to do so, since we can make the label describe what it is labelling. Programs make more sense this way.

Thus we use *name$* to label "Mr. Smith", *fact$* to label "You owe me money", and *threat$* for "Pay up or else".

This may seem long-winded, but it really does help to make your programs more readable, and hence easier to decipher. For example:

```
70 PRINT "Dear " name$
```

really tells you what the line is doing, far more than:

```
70 PRINT "Dear " A$
```

Similarly:

```
PRINT threat$
```

is more meaningful than

```
PRINT B$
```

The moral is, use words for variables (labels) as much as possible – and preferably lower case words.

Actually you can use capitals for variable names and intermix them with lower case letters and also numbers. The rules for doing so are as follows:

- All variable names must begin with a letter, though you can follow this with any mixture of letters, numbers and dots. Letters may be upper or lower case. They will however be considered equivalent.
- You cannot put spaces in the middle of variable names.
- Variables should be kept separate from Basic keywords.

The commonest error is to run a variable into a keyword.

One advantage of using variables instead of directly using strings is that we can easily alter the output of the program.

In the case of Program II, if we want another victim to be the recipient of our letter, just change line 30. For example:

```
30 name$ = "Mr. Jones"
```

From then on all uses of *name$* in the program will refer to Mr. Jones.

In this short program it doesn't make a great deal of difference, but in larger ones, if you had used the string "Mr. Smith" every time, instead of *name$,* you would be in for a lot of retyping.

You'd have to alter every single reference to "Mr. Smith" in the program. If you'd used *name$* throughout, all you'd have to do is change the line assigning *name$.*

Keeping variables in lower case can help us with a perennial problem for newcomers to the Amstrad – mistakenly tagging keywords onto variables.

Enter Program IV exactly as shown – making sure there's a space after print in line 40.

```
10 REM PROGRAM IV
20 MODE 1
30 word$ = "Hello"
40 print word$
```

When you LIST it, you'll see:

```
10 REM PROGRAM IV
20 MODE 1
30 word$ = "Hello"
40 PRINT word$
```

Program IV

RUN the program and all will be fine.

Suppose, though, that you'd missed out that space between print and word$. Enter this version of line 40:

```
40 printword$
```

Now LIST and RUN it. You'll get the error message "Syntax error in 40". As you can see, line 40 is now:

```
40 printword$
```

and the micro doesn't know what to do with this "variable".

More generally, variables tend to "absorb" keywords that run into them, to create larger variables. The keyword in effect "disappears" from the line, leaving the micro in some doubt as to what to do.

So to stop this confusion, leave spaces after – and before – keywords. Actually you can use characters other than space as separators, or delimiters as they are known. Quotes act as delimiters, as in:

```
PRINT"Hello"
```

If you're in the habit of entering variables in lower case, when you list a line, all the words in capitals must be keywords. If there aren't any words in capitals on a line you can see immediately you've slipped up – by forgetting the keyword, or missing out a separator.

Right, armed with that advice, let's return to Program II.

This introduces another new idea, the use of the TAB( ) function. This allows you to specify how far along a line you want the output of a PRINT statement to start.

In Mode 1 there are 40 characters to a line, so the screen can be considered to be 40 columns wide. TAB( ) decides in which column the printout starts. The 40 columns are numbered 1 to 40.

When you change mode the number of characters across the screen – that is the number of columns – changes. For example, Mode 0 only supports 20 columns.

Try running the program in this mode by changing line 20 to:

```
20 MODE 0
```

Can you see what is happening?

After a while TAB( ) becomes second nature. All too often potentially good programs are spoiled because they are set out badly on the screen. Careful use of TAB( ) can avoid this.

To give you some practice, try Program V. This prints out a triangle of asterisks. Can you devise a similar program, using TAB( ), to create a diamond of asterisks in the centre of the screen?

Before you continue, you might find it easier on the eyes if you return to Mode 1 with:

```
MODE 1 [ENTER]
```

So far we have talked about string variables. However there is another kind of variable called a numeric variable.

These are labels just as much as string variables are, only they label

```
10 REM PROGRAM V
20 MODE 1
30 PRINT
40 PRINT TAB(5)"*"
50 PRINT TAB(4)"**"
60 PRINT TAB(3)"***"
70 PRINT TAB(1)"*****"
```

Program V

numbers in such a fashion that we can do sums with them. Try running Program VI.

Line 30 uses the numeric variable *A* to label the number 10. Notice that for a numeric variable we can simply use a letter of the alphabet without following it with the $ sign necessary for a string.

Also since it isn't a string, the value we are giving the variable doesn't have to be in quotes. Hence line 30 is simply:

```
30 A = 10
```

Line 40 prints out, not *A*, of course, but the value that *A* labels, which is 10.

The most interesting part is line 50. Here we multiply the number that *A* labels by two, so that the line prints out 20.

That's the useful thing about numeric variables – you can do sums with them!

Notice that the micro did the calculation, then printed out the result. It didn't do anything wild such as printing out 2 * 10 or whatever.

A sum such as 2 * A or A + 8 is known as a numeric expression. When it encounters a numeric expression, the micro works it out and prints the answer, rather than printing the expression itself.

Try running Program VI with the following versions of line 50:

```
50 PRINT A + 8
50 PRINT A / 4
50 PRINT A * A
```

If you've been following what I've said so far you could be forgiven for thinking that string variables are for labelling words, and numeric variables for numbers.

```
10 REM PROGRAM VI
20 MODE 1
30 A = 10
40 PRINT A
50 PRINT 2 * A
```

Program VI

Life is never that simple. You can, and often do, use string variables for labelling numbers – the point is that you can't do sums with them. Try Program VII, which is based on Program VI, using the string *A$* instead of the numeric *A*.

The "Type mismatch in 50" that you receive shows that you are attempting to do a sum with the wrong type of variable – string instead of numeric.

```
10 REM PROGRAM VII
20 MODE 1
30 A$ = "10"
40 PRINT A$
50 PRINT 2 * A$
```

Program VII

As with string variables, we do not have to (and should not) restrict ourselves to single-letter labels for numeric variables.

We can use words in a manner strictly analogous to string variables, save that we omit the final $ sign. And, of course, we don't put what we are labelling in quotes, since it isn't a string.

Again, capitals and lower case are considered to be identical so *A* is the same as *a*.

Have a look at Program VIII. This is meant to be a cheery greeting for someone when they RUN the program in the computer – the sort of thing I often use in my classes.

```
10 REM PROGRAM VIII
20 MODE 1
30 name$ = "Mike"
40 PRINT "Good to see you, " name$
```

Program VIII

However as it stands it's a bit restricted – after all, only a small percentage of my students are called MIKE. What's really needed is some way for the Amstrad to find out the name of the person so that it can tailor the message to suit.

Program IX fits the bill. The trick here is the use of INPUT name$ in line 40. In Program VIII, line 30 put the value MIKE into *name$*. In Program IX the variable isn't actually attached to a specific value – if you like, you give the program a label, but neglect to tell it what it's labelling.

Instead you type:

```
INPUT name$
```

When the Amstrad reaches this line it waits until you PUT IN, or INPUT, the value you want *name$* to have by typing the value in.

To put it another way, when the computer meets an INPUT statement

```
10 REM PROGRAM IX
20 MODE 1
30 PRINT "What is your name";
40 INPUT name$
50 PRINT "Good to see you, " name$
```

Program IX

followed by a variable, it asks you what you want the variable to be – in fact, it actually puts a question mark on the screen.

You are then supposed to type in the answer followed by Enter, which, as always, sends it to the computer, which then carries on with the rest of the program.

So when you run the above program line 30 asks: "What is your name". Notice that we don't need a question mark – the INPUT statement of line 40 supplies that.

The micro then waits for us to type our reply and send it by pressing Enter. Whatever we have typed in then becomes the value of *name$* – even if we have lied!

Line 50 then prints out the message.

The point of all this is that in Program IX, as opposed to Program VIII, the value of *name$* is not fixed initially, but is decided during the program by the response to INPUT.

This means that every student in the class can now run the program and have the message tailored to themselves.

Incidentally, line 30 is not strictly necessary, but it is only polite to tell people what kind of response you expect them to make. Otherwise they will be met with just a question mark, followed by a cursor – not too "user-friendly" as the jargon has it.

The semi-colon at the end of line 30 "glues" the question mark, or prompt, as it is known, to the preceding "message". Running the program with it omitted should make this clear.

Remember, when you run Program IX and it asks for your name, you must type your reply then press Enter. If you omit Enter the Amstrad won't receive your answer and will continue waiting. This could be incredibly boring!

If you make a typing mistake before you press Enter, you can erase it with Delete. Once you've pressed Enter, though, you're stuck with what you've typed.

You can use INPUT with numeric variables as well as strings. Program X demonstrates this. When you get the prompt, try typing in a word rather than a number and see what happens.

A slightly more serious application of INPUT allows you to calculate the

```
10 REM PROGRAM X
20 MODE 1
30 PRINT "How old are you";
40 INPUT age
50 PRINT "I don't believe you are "; a
ge
```

Program X

product of two numbers, as Program XI demonstrates.

Look carefully at line 70 and see if you can work out what's happening. *first* isn't in quotes, and so the micro will print the number that *first* labels. "Multiplied by" is printed literally since it is in quotes.

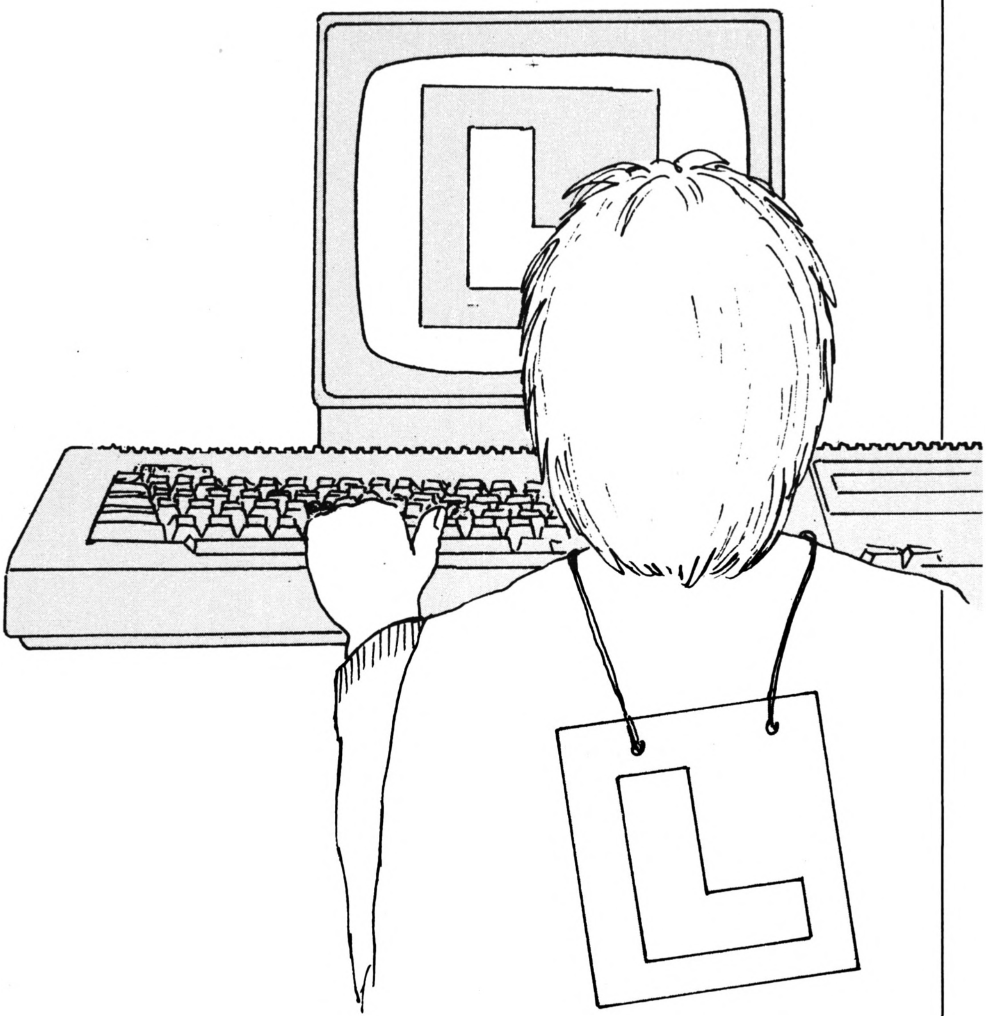
The numeric variable *second* is not in quotes – it may have them on either side, but the quotes on the left are already paired with the quotes on the far left, so they don't count. The micro will, therefore, print out the value of *second*.

"Is" is printed literally, since it is in quotes. *first*second* isn't in quotes, so the sum is done and the answer printed out. Figure I should help to make this clearer.

Finally, try altering Program XI so that it adds or subtracts pairs of numbers.

We've covered an enormous amount of ground here and I suggest that you spend a good while going over the programs. If you are having problems, re-reading the earlier articles will probably help.

Above all, remember it's a "hands-on" course – you can't expect the examples to make sense until you've typed them in!



```
10 REM PROGRAM XI
20 MODE 1
30 PRINT "First number";
40 INPUT first
50 PRINT "Second number";
60 INPUT second
70 PRINT first " multiplied by " secon
d " gives " first* second
```

Program XI

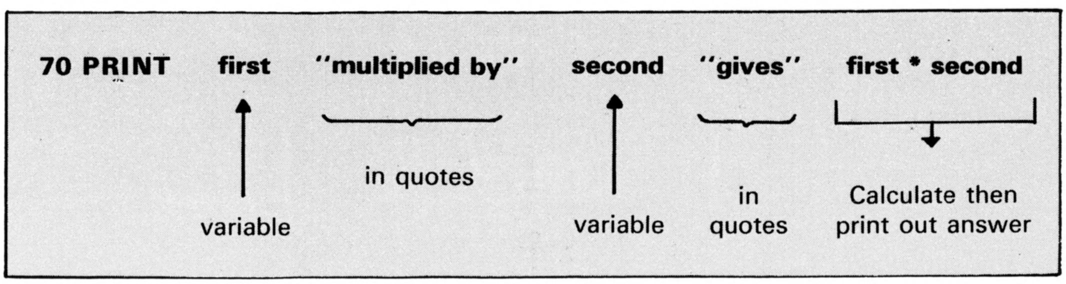


Figure I: Mixing variables and strings in PRINT statements

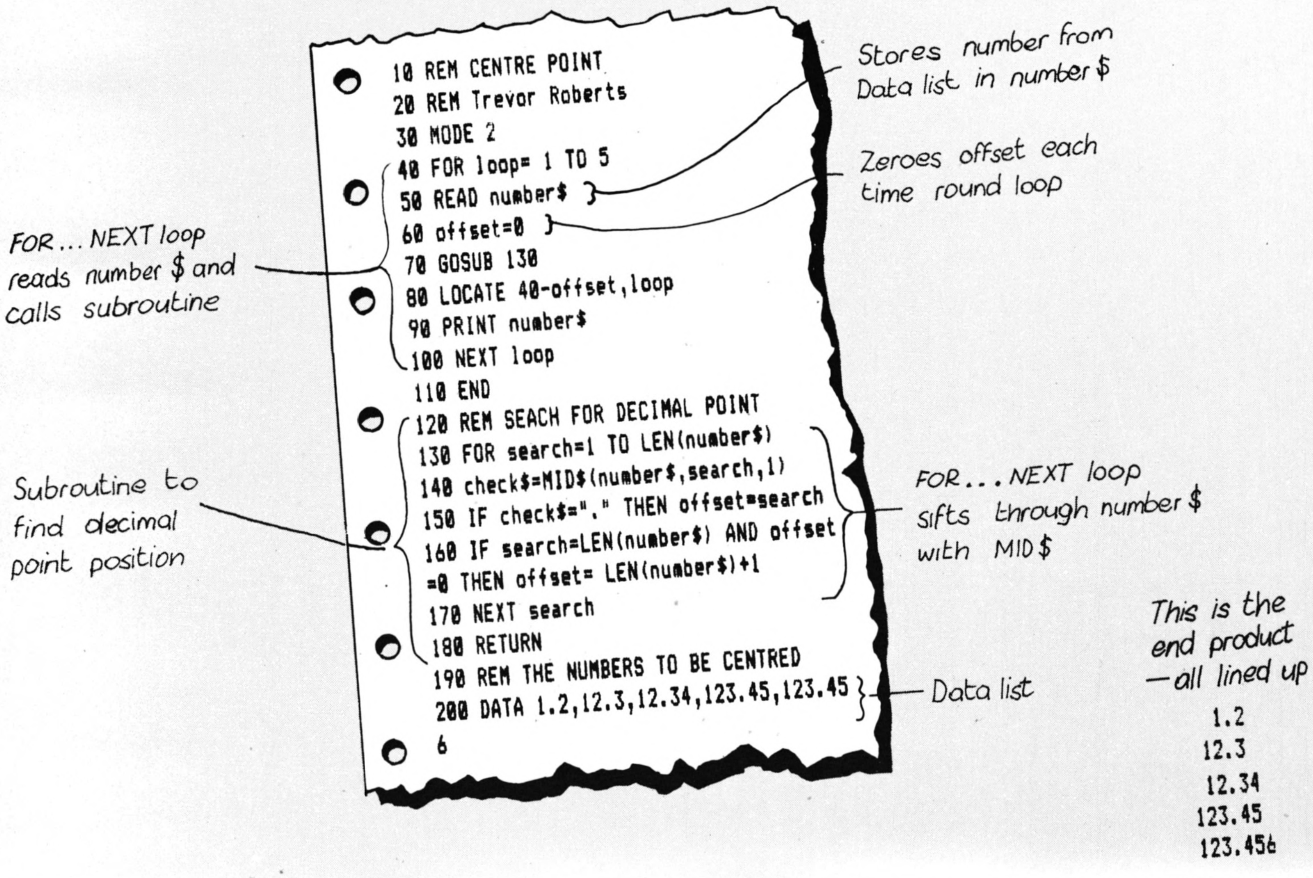
Get right to the point

with Trevor Roberts

HAVE you ever wanted to display a set of numbers with all the decimal points aligned under each other? That's the task set this month. Centre Point solves the problem using the *LEN* and *MID$* commands to dissect the numbers and find where the decimal point occurs.

Line number

10,20	Tell humans the title of the program and who wrote it. The Amstrad itself ignores everything after the REM.
30	Puts the micro into Mode 2, the 80 column mode.
40-100	Form a FOR ... NEXT loop which cycles five times to deal with each of the five numbers in turn.
50	Each time round the loop reads the next number from the data list and stores it in the string variable *number$.*
60	Ensures that *offset* is set to zero each time round the loop.
70	Calls the subroutine that locates the position of the decimal point – if any – in *number$.*
80	Ensures that all the decimal points are printed under each other. This is done by making the print position 40 minus the number of figures left of the decimal point in *number$.*
90	Prints out *number$* each time round the loop cycle.
110	Stops the program from crashing into the subroutine below.
120-180	Form the subroutine that searches *number$* for its decimal point.
130-170	Make up a FOR ... NEXT loop which cycles once for each character in *number$.*
140	Takes one character from *number$* and stores it in *check$.* By the time the loop has finished, every character in *number$* will have been stored in *check$.*
150	If *check$* is a decimal point then its position – given by the value of *search* – is stored in *offset.*
160	If the whole string has been checked and *offset* is still zero then the number has no decimal point. One is added to *offset* to give the position of the invisible decimal point at the end of a whole number.
180	Ends the subroutine.
200	Holds the five numbers to be printed.



```
10 REM CENTRE POINT
20 REM Trevor Roberts
30 MODE 2
40 FOR loop= 1 TO 5
50 READ number$
60 offset=0
70 GOSUB 130
80 LOCATE 40-offset,loop
90 PRINT number$
100 NEXT loop
110 END
120 REM SEACH FOR DECIMAL POINT
130 FOR search=1 TO LEN(number$)
140 check$=MID$(number$,search,1)
150 IF check$="." THEN offset=search
160 IF search=LEN(number$) AND offset
=0 THEN offset= LEN(number$)+1
170 NEXT search
180 RETURN
190 REM THE NUMBERS TO BE CENTRED
200 DATA 1.2,12.3,12.34,123.45,123.45
6
```

Robot Ron and the Ice Monsters

By STEPHEN MARTIN

NOT content with the scalps of many a sorry Weevil, Robot Ron goes in search of greater excitement and danger.

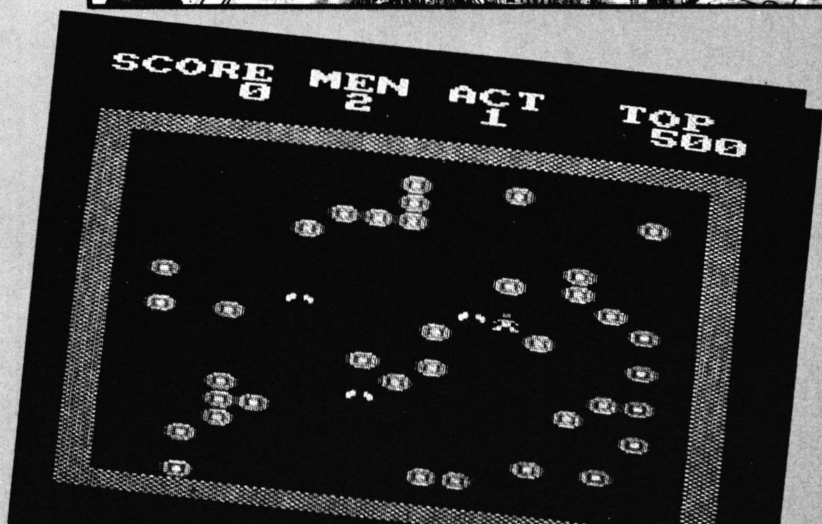
He stumbles into a huge ice maze inhabited by extremely dangerous super pink furry monsters – just what he was looking for.

Unfortunately in the extreme cold of this region Ron's trusty zap gun will not function, so he has to rely on his immense strength to push ice blocks over passing ice monsters.

Any keys or the joystick can be used to control Ron. You define the keys which suit you best by choosing option 4 on the menu. However the movement keys are set to the following when the program is first run:

Z	left
X	right
]	up
1	down
Enter/Return	push/crush

To pause the game press the Escape key once, then press any other key to resume.



VARIABLES

x,y	Ron's coordinates.
a (20,25)	Screen map.
r (20,1)	Monsters' coordinates.
n$(a)	Hi-scorers' names.
hi(a)	Hi-scorers' scores.
score	Score.
men	Lives left.
act	Level reached.
dead	Dead flag.

Full listing starts on Page 18

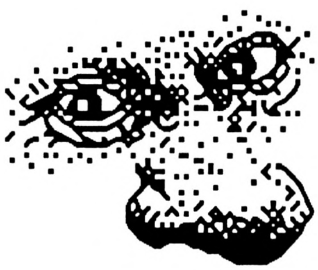
From Page 17

```
10 REM Robot Ron v Ice Monsters
20 REM By Steven Martin
30 REM (c) Computing With The Amstrad
40 REM
50 MODE 1:CALL &BC02:MEMORY &9FFF
60 d=18:e=19:f=22:g=71:h=63
70 GOSUB 820:REM initialise
80 GOSUB 440:REM data
90 GOSUB 1900:REM title screen
100 GOSUB 2050
110 MODE 0
120 GOSUB 2600:REM speed selection
130 MODE 0
140 GOSUB 260:REM menu
150 GOSUB 2050
160 MODE 0
170 GOSUB 880:REM set variables
180 GOSUB 910:REM wipe
190 GOSUB 640:REM set screen
200 GOSUB 1390:REM move monsters
210 GOSUB 930:REM push/crush
220 IF m=0 THEN GOSUB 1510:MODE 0:FOR
 s=1 TO 200:NEXT:GOTO 180
230 IF dead=1 THEN GOSUB 1560:MODE 0:
FOR s=1 TO 200:NEXT:GOTO 180
240 GOTO 200
250 REM -------- start screen ------
260 MODE 0:CALL &BB4E:CALL &BC02:DRAW
 0,398,4:DRAW 638,398:DRAW 638,0:DRAW
 0,0:PEN 1:LOCATE 9,2:PRINT "MENU"
270 PEN 2:BORDER 6:INK 2,24
280 LOCATE 5,7:PRINT"   up - ";:k=e:G
OSUB 400
290 LOCATE 5,9:PRINT" down - ";:k=f:G
OSUB 400
300 LOCATE 5,11:PRINT" left - ";:k=g:
GOSUB 400
310 LOCATE 5,13:PRINT"right - ";:k=h:
GOSUB 400
320 LOCATE 5,15:PRINT" push - ";:k=d:
GOSUB 400
330 PEN 4
340 LOCATE 4,19:PRINT"Space or Fire":
LOCATE 7,20:PRINT"to play"
350 LOCATE 2,22:PRINT"O for options m
enu"
360 IF INKEY(76)>-1 THEN d1=76:e1=72:
f1=73:g1=74:h1=75:ky=0:RETURN
370 IF INKEY(47)>-1 THEN d1=d:e1=e:f1
=f:g1=g:h1=h:ky=1:RETURN
380 IF INKEY(34)>-1 OR INKEY(32)>-1 T
HEN GOTO 1150
390 GOTO 360
400 RESTORE 2440:FOR a=1 TO 71:READ t
410 IF t=k THEN PRINT ke$(a):a=71
420 NEXT:RETURN
430 REM -------- data --------
440 RESTORE 470:FOR i=0 TO 98
450 READ a$:POKE &A000+i,VAL("&"+a$)
460 NEXT
470 DATA dd,7e,00,87,87,87,87,87,32,4
7,a0,26,c0,dd,7e,04,3d,87,87,6f,11,50
,00,dd,46,02,05,19,10,fd,e5,26,c0,dd,
7e,08,3d,87,87,6f,11,50,00,dd,46,06,0
5,19,10,fd,11,fd,07,06,08,36,00,23,36
,00,23,36,00,23,36,00
480 DATA 19,10,f2,e1,11,00,a1,06,08,1
a,77,13,23,1a,77,13,23,1a,77,13,23,1a
,77,13,78,01,fd,07,09,47,10,e9,c9
490 FOR i=0 TO 127:READ j:POKE &A100+
i,j:NEXT
500 DATA 0,65,130,0,0,65,130,0,0,0,0,
0,0,211,227,0,65,81,162,130,0,85,170,
0,0,170,85,0,68,136,68,136
510 DATA 0,240,240,0,80,60,60,160,180
,51,51,120,180,102,153,120,180,102,15
3,120,180,51,51,120,80,60,60,160,0,24
0,240,0
520 DATA 0,15,15,0,5,15,15,10,15,173,
94,15,94,173,94,173,94,47,31,173,15,1
5,15,15,15,15,15,15,5,10,5,10
530 DATA 0,0,0,0,0,3,252,0,1,86,252,1
68,1,86,252,168,1,87,252,168,1,86,252
,168,1,86,252,168,243,243,243,243
540 RETURN
550 REM -------- move man --------
560 i=x+(INKEY(g1)>-1)-(INKEY(h1)>-1)
570 j=y+(INKEY(e1)>-1)-(INKEY(f1)>-1)
580 IF a(i,j)>1 THEN RETURN
590 CALL &A000,x,y,i,j,0
600 x=i:y=j
610 FOR t=0 TO sped:NEXT
620 RETURN
630 REM -------- screen --------
640 BORDER 0:FOR t= 1 TO 19:a(t,4)=3:
a(t,24)=3:NEXT:FOR t=4 TO 24:a(1,t)=3
:a(19,t)=3:NEXT
650 RESTORE 650:FOR i=0 TO 15:READ j:
INK i,j:NEXT:DATA 0,18,6,24,2,8,20,26
,15,16,7,9,13,22,2,26
660 PAPER 4:PEN 3
670 LOCATE 1,4
680 PRINT STRING$(19,207)
700 FOR t=5 TO 23
710 LOCATE 1,t:PRINT CHR$(207):LOCATE
 19,t:PRINT CHR$(207)
720 NEXT
730 LOCATE 1,24
740 PRINT STRING$(19,207):PAPER 0
750 LOCATE 1,1:PEN 7:PRINT "SCORE MEN
 ACT  TOP":GOSUB 1310:GOSUB 1330:GOSU
B 1350:GOSUB 1370
760 FOR t=1 TO 40:rx=RND(1):ry=RND(1)
:rx=(rx*16)+2:ry=(ry*18)+5
770 CALL &A000,10,12,rx,ry,1:a(rx,ry)
=4:NEXT
780 FOR t=0 TO 2:rx=RND(1):ry=RND(1):
rx=(rx*16)+2:ry=(ry*18)+5
790 CALL &A000,10,12,rx,ry,2:a(rx,ry)
=5:r(t,0)=rx:r(t,1)=ry:NEXT
800 a(x,y)=0:RETURN
810 REM -------- initialise --------
820 DIM a(20,25),r(20,1),n$(9),hi(9):
FOR t= 1 TO 9:n$(t)="Computing with t
he Amstrad":hi(t)=500:NEXT
830 ENV 1,20,1,5:ENT -6,3,1,1:ENV 2,1
5,-1,5
840 DIM ke$(71):RESTORE 850:FOR t=1 T
O 71:READ ke$(t):NEXT
850 DATA 1,2,3,4,5,6,7,8,9,0,-,^,clr,
del,tab,q,w,e,r,t,y,u,i,o,p,@,[,enter
,caps,a,s,d,f,g,h,j,k,l,*,+,],shft,z,
x,c,v,b,n,m,<,>,/,\,ctrl,copy,space,e
nter,f7,f8,f9,f4,f5,f6,f1,f2,f3,f0,cs
r-up,f-stop,csr-lft,csr-dwn,csr-rt
860 RETURN
870 REM ------ set variables -------
880 x=10:y=12:a(x,y)=0:score=0:men=3:
act=1:m=3:dead=0
890 RETURN
900 REM -------- wipe --------
910 ERASE a:DIM a(20,25):RETURN
920 REM -------- push/crush --------
930 IF INKEY(d1)<0 THEN RETURN
940 IF INKEY(g1)>-1 THEN p=x-1:q=y:a=
-1:b=0
950 IF INKEY(h1)>-1 THEN p=x+1:q=y:a=
1:b=0
960 IF INKEY(e1)>-1 THEN p=x:q=y-1:a=
0:b=-1
970 IF INKEY(f1)>-1 THEN p=x:q=y+1:a=
0:b=1
980 IF p<0 OR p>20 OR q<0 OR q>25 THE
N RETURN ELSE IF a(p,q)<>4 THEN RETUR
N
990 IF a(p+a,q+b)=4 THEN GOSUB 1030:R
ETURN
1000 SOUND 1,142,90,10,,6:GOSUB 1060
1010 RETURN
1020 REM -------- crush --------
1030 LOCATE p,q:PRINT CHR$(32):a(p,q)
=0
1040 RETURN
1050 REM -------- push --------
1060 a(p,q)=0
1070 p1=p+a:q1=q+b
1080 IF a(p1,q1)=5 THEN a(p,q)=0:scor
e=score+50:SOUND 131,0:SOUND 2,1095,1
25,15,2,,5:GOSUB 1310:GOTO 1100
1090 IF a(p1,q1)>0 THEN a(p,q)=4:SOUN
D 129,0:SOUND 1,4095,50,15,2,,31:RETU
RN
1100 CALL &A000,p,q,p1,q1,1
1110 p=p1:q=q1
1120 CALL &BD19
```

```
1130 GOTO 1060
1140 REM ------ options menu ------
1150 MODE 0:CALL &BB4E:CALL &BC02:DRA
W 0,398,4:DRAW 638,398:DRAW 638,0:DRA
W 0,0:PEN 1:LOCATE 4,2:PRINT "options
 menu"
1160 RESTORE 1210:PEN 2:FOR t= 1 TO 4
1170 READ a$
1180 LOCATE 3,(t*2)+4
1190 PRINT t;" ";a$
1200 NEXT
1210 DATA "High Scores","Redefine Key
s","Instructions","Main Menu"
1220 PEN 3:LOCATE 4,20:PRINT "Select
Option"
1230 ky=1
1240 a$=INKEY$
1250 IF a$="1" THEN GOSUB 2050:GOSUB
1640:GOTO 1150
1260 IF a$="2" THEN GOSUB 2050:GOSUB
2220:GOTO 1150
1270 IF a$="3" THEN GOSUB 2050:GOSUB
2450:GOTO 1150
1280 IF a$="4" THEN GOTO 260
1290 GOTO 1240
1300 REM ------- print score -------
1310 s$=STR$(score):LOCATE 6-LEN(s$),
2:PRINT score:RETURN
1320 REM -------- print top --------
1330 s$=STR$(hi(1)):LOCATE 20-LEN(s$)
,2:PRINT hi(1):RETURN
1340 REM -------- print men --------
1350 s$=STR$(men):LOCATE 9-LEN(s$),2:
PRINT men:RETURN
1360 REM -------- print act --------
1370 s$=STR$(act):LOCATE 13-LEN(s$),2
:PRINT act:RETURN
1380 REM ------ move monsters ------
1390 FOR n=0 TO 2
1400 IF dead=0 THEN GOSUB 560:REM mov
e man
1410 i=r(n,0):j=r(n,1)
1420 IF j=0 THEN 1480
1430 IF a(i,j)<>5 THEN b=0:r(n,1)=b:L
OCATE i,j:PRINT " ":a(i,j)=0:m=m-1:GO
TO 1480
1440 i=i-(i<x)+(i>x)
1450 j=j+(j>y)-(j<y)
1460 IF a(x,y)<>0 THEN dead=1
1470 IF a(i,j)=0 THEN CALL &A000,r(n,
0),r(n,1),i,j,2:a(r(n,0),r(n,1))=0:a(
i,j)=5:r(n,0)=i:r(n,1)=j
1480 NEXT
1490 RETURN
1500 REM -------- finished --------
1510 act=act+1:m=3:score=score+100
1520 FOR t=1 TO 16
1530 FOR s=1 TO 50:NEXT:BORDER t:SOUN
D 129,242,10,15,,,20:NEXT
1540 BORDER 1:RETURN
1550 REM -------- dead --------
1560 SOUND 129,1500,100,15,2,0,15
1570 men=men-1:dead=0:m=3
1580 CALL &A000,x,y,x,y,3
1590 FOR t=1 TO 100:BORDER RND*26:INK
 0,RND*26:NEXT:CALL &BC02
1600 IF men>0 THEN RETURN
1610 IF score>= hi(9) THEN GOSUB 1720
1620 GOTO 130
1630 REM ------- high scores -------
1640 MODE 1:GOSUB 2200
1650 DRAW 0,398,1:DRAW 638,398:DRAW 6
38,0:DRAW 0,0
1660 PAPER 3:PEN 1:LOCATE 14,2:PRINT
"Block Busters":PAPER 0
1670 FOR t=1 TO 9
1680 LOCATE 4,4+t*2:PEN 1:PRINT t;:PE
N 3:PRINT " ";n$(t);" ";:PEN 2:WHILE
POS(#0)<31:PRINT ".";:WEND:PEN 1:PRIN
T hi(t)
1690 NEXT:PAPER 1:PEN 0:LOCATE 14,24:
PRINT "Space for Menu":PAPER 0
1700 GOSUB 2150
1710 a$=INKEY$:IF a$<>" " THEN 1710 E
LSE MODE 0: RETURN
1720 REM -------- new high --------
1730 MODE 1:CALL &BB4E:CALL &BC02
1740 DRAW 0,398,1:DRAW 638,398:DRAW 6
38,0:DRAW 0,0:PAPER 2
1750 PEN 3:LOCATE 13,4:PRINT" New Hig
h Score "
1760 PAPER 0:PEN 1:LOCATE 10,10:PRINT
"Please Enter Your Name"
1770 PAPER 1:PEN 3
1780 LOCATE 10,15:PRINT"-------------
---------"
1790 LOCATE 10,15:k$="":n$(9)=""
1800 WHILE INKEY$<>"":WEND
1810 WHILE k$<>CHR$(13)
1820 IF k$>CHR$(31) AND k$<"z" AND LE
N(n$(9))<23 THEN n$(9)=n$(9)+k$:PRINT
 k$;
1830 IF k$=CHR$(127) AND LEN(n$(9)) T
HEN n$(9)=LEFT$(n$(9),LEN(n$(9))-1):P
RINT CHR$(8);CHR$(16);
1840 k$=INKEY$
1850 WEND:IF n$(9)="" THEN n$(9)="ANO
N,(don't blame you)"
1860 hi(9)=score:FOR i=9 TO 2 STEP-1
1870 IF hi(i)>hi(i-1) THEN k$=n$(i):n
$(i)=n$(i-1):n$(i-1)=k$:hi(i)=hi(i-1)
:hi(i-1)=score
1880 NEXT:RETURN
1890 REM ------- title screen -------
1900 MODE 1:CALL &BB4E:CALL &BC02
1910 RESTORE 2040
1920 PAPER 3:PEN 1:LOCATE 1,4
1930 FOR t=1 TO 5
1940 READ a$
1950 PRINT TAB(3);
1960 FOR p=1 TO 37
1970 IF MID$(a$,p,1)="1" THEN PRINT C
HR$(207); ELSE PRINT CHR$(32);
1980 NEXT
1990 PRINT
2000 NEXT
2010 PAPER 0:PEN 1:LOCATE 9,15:PRINT"
MEETS THE ICE MONSTERS!"
2020 FOR t=1 TO 3000:NEXT
2030 RETURN
2040 DATA 1100111011001110111000001100
1110100100,10101010101010100100000101
01010110100,11001010110010100100000011
001010111100,10101010101010100100000001
0101010101100,1010111011001110010000
10101110100100
2050 REM ------- clear screen -------
2060 FOR t=40 TO 0 STEP-1
2070 OUT &BC00,1
2080 OUT &BD00,t
2090 FOR p=1 TO 25:NEXT
2100 NEXT
2110 CLS
2120 OUT &BC00,1
2130 OUT &BD00,40
2140 RETURN
2150 FOR t=1 TO 40
2160 OUT &BC00,1
2170 OUT &BD00,t
2180 FOR p=1 TO 25:NEXT
2190 NEXT:RETURN
2200 OUT &BC00,1:OUT &BD00,0:RETURN
2210 REM ------- redefine keys ------
-
2220 MODE 0:CALL &BB4E:CALL &BC02
2230 BORDER 14:DRAW 0,398,1:DRAW 638,
398:DRAW 638,0:DRAW 0,0:PEN 7:PAPER 4
2240 LOCATE 4,2:PRINT" Redefine Keys
"
2250 PAPER 0:PEN 9
2260 LOCATE 3,7:PRINT"  up   -  ";
2270 e=0:k=-1:WHILE k=-1:GOSUB 2400:W
END:e=k
2280 LOCATE 3,9:PRINT" down  -  ";
2290 f=0:k=-1:WHILE k=-1:GOSUB 2400:W
END:f=k
2300 LOCATE 3,11:PRINT" left  -  ";
2310 g=0:k=-1:WHILE k=-1:GOSUB 2400:W
END:g=k
2320 LOCATE 3,13:PRINT"right  -  ";
2330 h=0:k=-1:WHILE k=-1:GOSUB 2400:W
END:h=k
2340 LOCATE 3,15:PRINT" push  -  ";
2350 d=0:k=-1:WHILE k=-1:GOSUB 2400:W
END:d=k
```

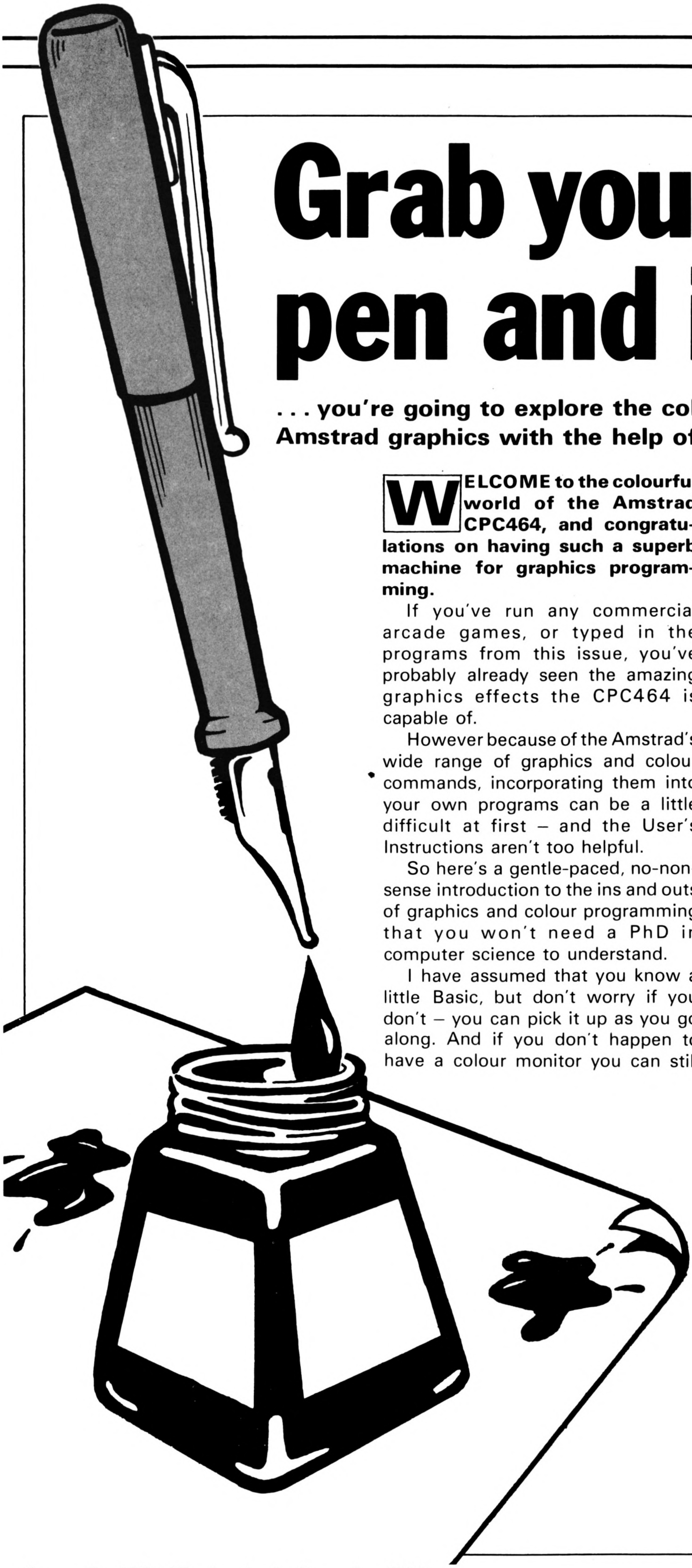
From Page 19

```
2360 PEN 3:LOCATE 5,20:PRINT "Correct
 ?"
2370 IF INKEY(43)>-1 THEN RETURN
2380 IF INKEY(46)>-1 THEN 2220
2390 GOTO 2370
2400 RESTORE 2440:FOR a=1 TO 71
2410 READ t
2420 IF INKEY(t)>-1 THEN k=t:PRINT ke
$(a):a=71:WHILE INKEY(t)>-1:WEND
2430 NEXT:RETURN
2440 DATA 64,65,57,56,49,48,41,40,33,
32,25,24,16,79,68,67,59,58,50,51,43,4
2,35,34,27,26,17,18,70,69,60,61,53,52
,44,45,37,36,29,28,19,21,71,63,62,55,
54,46,38,39,31,30,22,23,9,47,6,10,11,
3,20,12,4,13,14,5,0,7,8,2,1
2450 REM ------- instructions -------
2460 MODE 1:CALL &BB4E:CALL &BC02
2470 GOSUB 2200
2480 BORDER 14:PAPER 3:PEN 2
2490 LOCATE 15,2:PRINT" Robot Ron "
2500 PAPER 0:PEN 1
2510 PRINT:PRINT:PRINT:PRINT:PRINT "
Robot Ron has defeated all his previo
us"
2520 PRINT"opponents but now he has m
et his match.":PRINT
2530 PRINT" Ice monsters are the most
 feared beasts"
2540 PRINT"in the galaxy. And they ar
e extremely":PRINT:PRINT"difficult to
 kill.":PRINT
2550 PRINT" The only way this can be
achieved is to":PRINT"crush them by p
ushing an ice cube over":PRINT
2560 PRINT"them. GOOD LUCK!":PRINT
2570 PAPER 1:PEN 3:LOCATE 14,24:PRINT
"Space for Menu"
2580 GOSUB 2150
2590 a$=INKEY$:IF a$<>" " THEN 2590 E
LSE RETURN
2600 REM --- speed selection ---
2610 PEN 3:LOCATE 3,2:PRINT"Speed  Se
lection"
2620 RESTORE 2640:FOR t=1 TO 5
2630 READ a$:LOCATE 3,(2*t)+5:PEN t+5
:PRINT a$:NEXT
2640 DATA "1. Hand Breaking","2. Brea
th Taking","3. Mr Average","4. Slowww
w.","5. The A Team's"
2650 a$=INKEY$
2660 IF VAL(a$)>5 OR VAL(a$)<1 THEN 2
650
2670 zp=VAL(a$):zp=zp-1:sped=zp*100
2680 RETURN
```



Look for the new look
Computing **W**ith **T**he **A**mstrad

Available in the first week of February



Grab your paper pen and ink...

... you're going to explore the colourful world of Amstrad graphics with the help of MICHAEL NOELS

WELCOME to the colourful world of the Amstrad CPC464, and congratulations on having such a superb machine for graphics programming.

If you've run any commercial arcade games, or typed in the programs from this issue, you've probably already seen the amazing graphics effects the CPC464 is capable of.

However because of the Amstrad's wide range of graphics and colour commands, incorporating them into your own programs can be a little difficult at first – and the User's Instructions aren't too helpful.

So here's a gentle-paced, no-nonsense introduction to the ins and outs of graphics and colour programming that you won't need a PhD in computer science to understand.

I have assumed that you know a little Basic, but don't worry if you don't – you can pick it up as you go along. And if you don't happen to have a colour monitor you can still take advantage of the techniques we'll be using.

So switch on your Amstrad, or reset it if it's already on. To reset it press the Esc key while holding Ctrl and Shift down. You should see a blue screen with the familiar message appearing in yellow writing.

Let's see how many characters you can fit onto one line of the screen. Type:

```
0123456789
```

repeatedly, and you'll find that the screen has a width of exactly 40 characters.

You can actually have three basic types of screen – these screen types are known as modes – all with different screen widths. Enter:

```
mode 0
```

and see how many characters wide the screen is now. (Incidentally, if you've got a syntax error here, you've probably missed the space out between *mode* and *0*.)

As you'll have discovered, *mode 0* has 20 characters – rather fat ones at that. Now try:

```
mode 2
```

Again the screen is cleared, but this time we get a "skinny" *Ready*. If you're up to all the typing, you'll find that you can fit 80 characters across the screen.

When you first switch on or reset you are in Mode 1. Prove it now by entering:

```
mode 1
```

As you can see, we're back to normal size, and can fit 40 characters across the screen.

So we've got three modes – 0, 1 and 2. Notice it's not 1, 2 and 3. Remember, computers always start their counting at zero, not one.

The number of characters across the screen is not the only difference between modes – they also differ in the number of colours they allow on the screen at once.

Mode 0 allows 16 colours, Mode 1 four and Mode 2 permits two colours. Notice that the more colours you have the less characters you get across the screen, and vice versa.

When you think about it, it makes sense. You've only got a fixed amount of memory reserved for the screen, so if you're keeping track of a lot of colours you've not got much spare for remembering a lot of characters.

On the other hand, if you decrease the number of colours you've got to remember, there's more memory space available to keep tabs on a larger number of characters. Table I summarises it.

Mode	No. of characters	No. of colours
0	20	16
1	40	4
2	80	2

Table I: Mode characteristics

If you've been following so far, you should be in Mode 1. If not reset and we'll return to writing in yellow on a blue background, with a screen width of 40 characters.

Looking at it logically the smallest number of colours you can have in a mode is two. If you're going to see anything on the screen at all you'll need a foreground colour and a background colour.

For instance, in order for you to see the writing on this page, we've chosen black to be the foreground colour (that is, the colour that letters appear in) and white for the background (the colour of the paper).

Of course our printers could swop this round – and sometimes do – so that the letters appear in white on a black background, giving a sort of negative.

If we really wanted to go berserk we could print it in a white foreground on a white background, only you wouldn't be able to see it because of the lack of contrast. (Oddly enough, this sometimes comes in handy on the Amstrad.)

At the moment as far as the CPC464 is concerned I want you to imagine that we're writing on blue paper with a pen filled with yellow coloured ink.

All right, I'll come clean – the reason for the tortuous last sentence was that *pen, ink* and *paper* are special words as far as the Amstrad is concerned. Enter:

```
pen 2
```

and you'll see the Ready prompt as usual after a direct command, but it's changed colour. Now it appears in cyan.

If you try typing in a few characters – it doesn't matter which – they should all appear in cyan, though still on a blue background.

Next, try:

```
pen 3
```

and the writing should now appear in red. Then enter:

```
pen 1
```

and our characters will appear in yellow again.

Great! We've got three pens to write with, have we? No. We've actually got four – each filled with a different coloured ink – and, as per usual computer practice, we number them from 0 to 3.

If we want to change pens, we simply type pen followed by a space and then the number of the pen we want. So:

```
pen 3
```

puts out writing in bright red. Type:

```
pen 0
```

and try some writing. You won't be able to see a thing because – if you haven't guessed yet – *pen 0* is blue so you're writing in blue ink on blue paper.

So how do we get out of it? Well, press Enter to get you on a new line, then carefully type:

```
pen 1
```

and press Enter again. You should get back to yellow writing.

If you can't manage this – and it can be really awkward typing when you can't see what you're doing – reset the machine. Now try:

```
pen 4
```

Sorry about that! You're writing in blue on blue again. Oh well, at least you know how to get out of it this time – and you know that *pen 4* is the same as *pen 0*. Reset your machine and try:

```
pen 5
```

The writing's still in yellow. So *pen 5* is the same as *pen 1*. So what about *pen 6?* Try it:

```
pen 6
```

We're in cyan, the same as *pen 2*. No prizes for guessing that:

```
pen 7
```

gives you red. Let's explain: When you switch on or reset, you are in Mode 1. Now Mode 1 only allows four colours or inks on the screen at once. So when you've gone from *pen 0* to *pen 3* the CPC464 starts again by making *pen 4* equivalent to *pen 0* and so on. Similarly *pen 8* is equivalent to *pen 0,* and so on.

So in Mode 1, given a pen number, it's equivalent to the remainder left when that pen number is divided by 4 (since it's a four colour mode). Hence *pen 13* is equivalent to *pen 1*. I always imagine that the pen numbers are "wrapping round" to the start again.

If the above maths has you foxed don't worry. If you don't try anything fancy, and stick to the numbers 0 to 3 for your Mode 1 pens, you'll be all right. Table II summarises the colours

Pen number	Colour
0	Bright blue
1	Bright yellow
2	Bright cyan
3	Bright red

Table II: Default colours in Mode 1

associated with the pen numbers.

Now run Program I, which illustrates the different colours available. You could add the following lines to illustrate *pen 0:*

```
90 pen 0
100 PRINT "This is in pen 0"
```

but, of course, you wouldn't see it.

```
10 REM PROGRAM I
20 MODE 1
30 PEN 1
40 PRINT "This is in pen 1"
50 PEN 2
60 PRINT "This is in pen 2"
70 PEN 3
80 PRINT "This is in pen 3"
```

What would happen if we ran it in Mode 2? Change line 20 to:

```
20 mode 2
```

and see.

What happened to *pen 2,* and why's *pen 3* yellow? It used to be red! Well Mode 2 is a two colour mode. *Pen 0* gives us bright blue, *pen 1* gives us bright yellow – then we've run out of available colours, so we start again as we did when we ran out of colours in Mode 1 (only then there were four available).

So *pen 2* wraps round to blue and disappears against the blue background, while *pen 3* becomes yellow, and so on. Table III shows the colours associated with the pen numbers in Mode 2.

If we change line 20 to:

```
20 mode 0
```

there seems to be little difference from when you ran it in Mode 1, save for the fatter characters. Don't forget, though, this is a 16 colour mode – our pens should go from 0 to 15.

Program II illustrates the idea – showing all 16 colours – including the rather natty flashing colours of pens 14 and 15. Table IV shows the

```
10 REM PROGRAM II
20 MODE 0
30 FOR colour = 0 TO 15
40 PEN colour
50 PRINT "This is colour ";colour
60 NEXT colour
```

Pen number	Colour
0	Bright blue
1	Bright yellow

Table III: Default pen colours in Mode 2

colours associated with the pen numbers in Mode 0.

Try changing line 20 to give Modes 1 and 2 and you'll see how in modes with less colours the pen numbers wrap around. If you now enter *pen 16*:

```
Improper argument
```

will be hurled back at you. The CPC464 knows that the biggest pen number it can possibly have is 15, so it throws *pen 16* out. In Modes 1 and 2, as we've seen, it wraps the pen numbers round, but it still rejects numbers over 15.

So far all our work has been done on a nice blue background, but we aren't restricted to this. Let's investigate.

Reset your micro so you are back in Mode 1. Now so far we've been writing with pens filled with different coloured inks – on blue paper. Enter this:

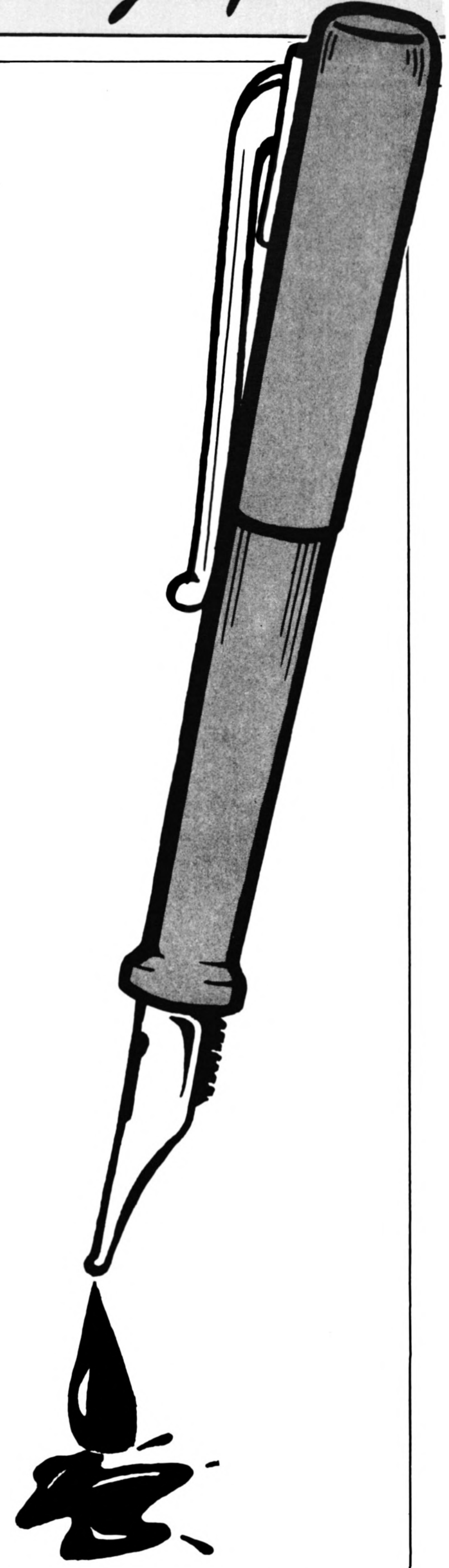
```
paper 3
```

All of a sudden Ready appears on red paper. That is, the letters still appear in yellow, but on a red background. You see:

```
paper 3
```

Pen number	Colour
0	Bright blue
1	Bright yellow
2	Bright cyan
3	Bright red
4	Bright white
5	Black
6	Bright blue
7	Bright magenta
8	Cyan
9	Yellow
10	Pastel blue
11	Pink
12	Bright green
13	Pastel green
14	Flashing blue/ bright yellow
15	Flashing pink/ sky blue

Table IV: Default pen colours in Mode 0



means "write on paper that's the same colour as the ink in *pen 3*".

Now, from Table I, *pen 3* is bright red, so *paper 3* sets the background to bright red. Type in some characters of your own if you don't believe me. Next try:

```
paper 2
```

The ink in *pen 2* is bright cyan, so our writing now appears on cyan paper. I find this terribly difficult to read, so let's make it clearer by changing our foreground colour to red. Remember how? It's:

pen 3

Paper colour is really quite easy to use – it works just as pen does, and follows the same restrictions as to mode. Just bear in mind: *paper n* means the background colour is that of the ink in *pen n*.

Notice that so far only the background of the characters you've typed has been in the new paper – the rest of the line stays in the old paper. When you've reached the bottom line, however, and the new line scrolls up, the whole of that line will be in the new paper.

After all, it's got to be in something, and as it's brand spanking new we may as well have it in the new paper.

There is a quicker way to get the screen in the new background colour. Enter:

paper 1 : cls

and the screen will clear to a yellow background *(paper 1)* with writing in the red foreground colour (we're still in *pen 3*).

You'll also notice something else if you haven't already – our yellow paper is surrounded by a blue border. You haven't noticed it before because our background's always been blue, matching the border.

We'll see later how you can change the border's colour. In fact there's not much else you can do with it – we can't actually write anything there . . .

Before we continue, have a look at Program III, which illlustrates how the various pen and paper combinations work.

```
10 REM PROGRAM III
20 MODE 0
30 FOR background = 0 TO 15
40 PAPER background
50 CLS
60 PRINT"This is paper "; background
70 PRINT:PRINT
80 FOR colour = 0 TO 15
90 PEN colour
100 PRINT "This is colour ";colour
110 NEXT colour
120 PRINT:PRINT:PRINT"Press any key"
130 delay$ = INKEY$ : IF delay$="" TH
EN GOTO 130
140 NEXT background
150 PAPER 0 : PEN 1
```

So far we've only seen 16 colours. However, when you bought your Amstrad you were promised 27. What's happened to them?

Program IV shows where they've been hiding. It successively steps the border through all 27 colours of inks, as they are known.

```
10 REM PROGRAM IV
20 MODE 1
30 FOR colour = 0 TO 26
40 BORDER colour
50 LOCATE 16,12
60 PRINT "Border "; colour
70 FOR delay = 0 TO 500 :NEXT delay
80 NEXT colour
```

As you'll have guessed *border* is the command that changes the colour of the border – you simply follow it with a space and the number of the colour you want the surround to be.

But beware, these numbers won't appear to have anything to do with the numbers you've been using for pen and paper. For example:

border 0

The border turns black – not blue as you would expect from pen 0 and paper 0.

This is a very important point – the numbers used with pen don't label colours – they label pens, which just happen to be filled with "coloured inks".

Just because *pen 3* has so far always given us red in Mode 1, it doesn't have to. It's just that, at the moment, *pen 3* happens to be filled with red ink.

Later on I'll show you how to fill a pen with, say, blue ink – in fact any coloured ink from our "palette" of 27 colours.

So the 3 in *pen 3* labels the pen, not the colour of the ink it is filled with. As in Mode 1 we're allowed four pens, and hence four corresponding papers. We can fill these pens with any four of the 27 – a sort of "perm any 4 from 27". In fact you can fill all four pens with the same colour if you want.

Much the same holds for the other modes, with their different number of pens.

Now the micro needs some way of referring to each of the 27 available colours. It could, of course, do it in

words – orange, bright red and so on.

Being a computer, it prefers to give the various coloured inks reference numbers, as shown in Table V.

As you can see, ink 0 is black and as the border command uses the ink number NOT the pen number:

border 0

turns the border black – and leaves the screen entirely alone. You must use pen or paper to affect the screen.

Next month I'll show you how to fill the pens with any ink you care to choose. For now, though, it's probably best if you just use the inks that the pens are "supplied" with when you switch on or reset – the default inks as they are known. Tables II, III and IV show them for each mode.

That should keep you busy enough until our next issue!

Ink number	Colour
0	Black
1	Blue
2	Bright blue
3	Red
4	Magenta
5	Mauve
6	Bright red
7	Purple
8	Bright magenta
9	Green
10	Cyan
11	Sky blue
12	Yellow
13	White
14	Pastel blue
15	Orange
16	Pink
17	Pastel magenta
18	Bright green
19	Sea green
20	Bright cyan
21	Lime green
22	Pastel green
23	Pastel cyan
24	Bright yellow
25	Pastel yellow
26	Bright white

Table V: Ink colours

PUBLIC DOMAIN

With Shane Kelly

O.K., this month we are going to communicate if it kIlls you. On side 1 of the disc you will find several files that are already configured for the various CPC machines called MDM730.DOC which will give you an idea of the functions that can be had and there are notations where this doc file differs from the programs supplied. Your recommended course of action if you are new to comms is to read the docfile, then read the RS232 manual, then read PROTOCOL. DOC from last months disc and then try running the program that is already configured for your AMSTRAD. If you already know all about comms then skip the above and get on with it! Back to the novices. For CPC ownwers (hands up you lot, don't be shy!) try firing up CPM 2.2 or 3.0• and then running M9CPCALL.COM. This program will run on all CPC's under both CPM's and it works. Now, there is one slight hitch with this program - it cannot do split baud rates and as there are now quite a few bulletin boards that use the 1200/75 format this would be a useful feature. Enter M9NOSET.COM ! To use M9NOSET you must set up the baud rate before running it. Under CPM 2.2 use the setup utility and keep this permanentlly altered system disc as your comms disc. Under CPM 3.0 use the SETSIO utility to achieve the same result.

Now you PCW owners, don't get impatient, it's your turn now. The file for you is MDM8000.COM. Now the bad news - I don't have a PCW so I can't test this for you. If any person would like to donate a PCW I would certainly take it with thanks.... no offers eh? Thought not.

Right, well what's the rest of this rubbish on the disc and why is it so? I don't know , I just write the column! No, OK (the editor just hit me over the head and told me to get serious - he's a bully isn't he?) I'll get on with it. The files on side 1 user 0 are:

M9CPCALL.COM All CPC RANGE modem program. CPM 2.2 & 3.0 (no split baud rates) This program sets the baud rate to 300/300 on start up

M9NOSET.COM All CPC Range modem program. CPM 2.0 & 3.0 To obtain split baud rates, set them up before entering this program.

MDM8000.COM A PCW modem program Untested by me but I am assured it works. CPM 3.0 ON PCW's ONLY! ! ! ! (no split baud rates)

The following files are in user 1 on side 1:

MDM730.DOC Explains the features and foibles of the modem progs with notes that show where M7XX and M8XXX differ.

M7LIB.DOC, M7RUB.MSG, MDM730.MSG, MDM730.NOT
These files are all short notes, messages and doc files to people who wish to bring up the modem program from scratch.

In user area 2 on side 1 we have:

MODEM.COM, Sripped down version of a modem program. Handy if you just want to talk between machines.

MODEM.DQC This is a squeezed doc file outlining the above program.

In case you are having trouble unsqueezing a file on single drive systems, I use the following method: Copy the squeezed file to a blank disc. Fire up NSWP (available on PD DISK1) and then log in the disc with the squeezed file on it, and then tag it, and subsequently unsqueeze it. NOTE this will only work if the unsqueezed version + the squeezed version is less in total length than the space on your disc- use a data disc if possible !

Now, on side 2 we have all the files you need to bring up a version of a modem program (MODEM798.COM). I have not used these files as there was not time before I had to send this column in. They are included only for the hackers and are not needed to get a working modem program. I f you only want a running modem program use the appropriate one as detailed at the begging of this article. The file M798-AMS.ASM is an overlay that is configured for a PCW 8512. This was done by John Dalstead and I know that John used it in its assembled form so if you want to bring up your own version it should not be too difficult.

Time to to, but before I do, I'll just say that next month I plan to bring to you a full featured MACRO ASSEMBLER (Z80 CODE) AND A DISASSEMBLER with a few utilities to change 8080 code to Z80 code. These are in the pipeline but no promises ! !

Shane Kelly

Editors Note:

If any readers have any particular requests in the Public Domain area (or if you have any Public Domain software you'd like to share with other Amstrad owners) you can contact Shane via this magazine using the address on Page 34 of this issue. We'd also like to take this opportunity to thank all those who have purchased PD Disk 1 and by doing so shown their support for this series. Please see P. 34 for details on how to order this month's disk.

Part 2 of COLIN FOSTER's exploration of CP/M 2.2 on the Amstrad

A whistlestop tour of CP/M

THIS month we'll look at what the different CCP (Console Command Processor) commands do, and how CP/M organises memory. First, however, let's go back to the beginning – always a good place to start! – and see just what happens when you type ICPM.

First, Basic hands over control to the cold boot routine in the BIOS ROM. (Cold boot is the CP/M term for the machine being completely reset and CP/M started up "from cold".

This routine looks for, and loads, the boot sector from the system tracks of the disc in drive A. Don't worry if you don't know what these terms mean – we'll explain them in later articles.

For now, just take them as meaning a special place on the disc (which you can't get at) where the programs which make up CP/M live.

Anyhow, this sector contains a short program, 512 bytes long, which initialises the computer. To do this, it first loads the configuration sector from the disc. This contains data put there by the program SETUP, which lets you customise the system to your own requirements.

We'll talk more about how to give your CP/M go-faster stripes next month.

Once the boot program has used this information to set up the computer as you want it, it passes control back to the BIOS ROM – to the warm boot routine, this time.

A warm boot occurs quite frequently in CP/M – every time a program finishes and hands control back to the system, or whenever you type Ctrl+C on the keyboard.

What it does is to load the main parts of CP/M from the disc, the CCP and BDOS, "log in" the disc to let the system know what's on it, and hand over control to the CCP.

Remember the CCP is just a program which acts as an "interface" between you and the computer.

Put a copy of your master disc – NEVER use the original – into drive A, side 1 up. Next reset the computer by pressing Ctrl+Shift+Esc and type Icpm. This performs the cold start, setting the computer up and loading the BDOS and CCP from disc.

The CCP tells you that it's alive and waiting for a command by displaying the ubiquitous A> prompt.

There are only six commands you can type in response to this which the CCP can understand and obey itself – these are called the resident, or built-in, commands and can be seen in Figure I. If you type anything else in response the CCP assumes that what

It's just a program which acts as an "interface" between you and the micro

you have typed is the name of a program on disc, and will attempt to load it into memory and execute it. These "commands by default" are called transient commands, because they change depending on which programs you have available on the disc in use.

Whenever CP/M is looking for input it provides some editing commands and other facilities. Ctrl+X deletes everything you have typed on the line, Ctrl+P will echo all console output to your printer (typing Ctrl+P again will turn this off, Ctrl+S will temporarily pause console output (restart it by pressing any key) and Ctrl+C will abort and warm boot.

DIR

The first of the built-in commands, DIR, you've probably met already.

Type:

```
A>dir
```

and you will get a directory listing of the files on side 1 of your disc. (Actually, you could have used dir instead of DIR as CP/M ignores case.)

This command is the CP/M equivalent of the Amsdos CAT command, but as you have probably noticed, does not tell you the sizes of files. You will notice a file in the list called STAT.COM.

Now type:

```
A>stat *.*
```

This is not one of the built-in commands – the CCP will recognise it as a transient command and so will load and run the program STAT.COM. As you will see from the screen, STAT gives us a much fuller directory listing than DIR.

We'll discuss what all the information means another time. For now just note that we get a list of files on the disc in alphabetical order, their sizes in kilobytes, and the amount of space still available on the disc.

The disadvantage of STAT is simply that it is a transient command – the program STAT.COM must be present on the disc for the command to work. DIR will work on any disc.

Notice the *.* we gave STAT as a parameter on the command line. This is an example of what CP/M calls an ambiguous file name, and simply asks STAT to give us information on all the files on the disc, whatever they're called.

SAVE

The second built-in command we come to is SAVE. This is dismissed in Amstrad's DDI-1 manual as being "for specialist use only". Well, you are all about to become "specialists". Make sure your disc is write-enabled with the little white tab pulled fully OUT and type:

```
A>save 24 fred.com
```

The disc will whirr and clank for a few seconds, then A> will return and nothing else seems to happen. What

DIR	Gives a limited directory listing of the files on a disc.
SAVE	Saves specified number of pages of memory, starting at &100, to disc with the specified name.
REN	Renames existing files.
ERA	Erases unwanted files. Use with caution!
USER	Changes current user area.
TYPE	Lists Ascii text files to the screen.

Figure I: Summary of CCP resident commands

&FFFF	
&C000	BIOS ROM
&BEC0	BIOS stack
&BE80	BIOS extended jumpblock
&AD33	Firmware and BIOS variables
&AD00	BIOS jumpblock
&9F00	BDOS
&9700	CCP
&0100	TPA
&0000	SPA

Figure II: Amstrad CP/M memory map

have we done? Well, type:

```
A>stat fred.com
```

This is another way of using STAT, this time with an unambiguous file name. It will provide information only on the file we specified.

You will see that we now have a 6k file on disc called FRED.COM! In fact, all SAVE does is to copy the number of pages of memory specified, from the start of the TPA onwards, into a disc file with the name we've given. (A page of memory is 256 bytes.)

Type:

```
A>fred *.*
```

FRED has the same effect as STAT! Well, it should do – they're identical. That's because after we first called STAT and it had executed it was still present in memory.

The command *save 24 fred.com* immediately afterwards simply copied STAT out of memory to a new file, FRED.COM (STAT just happens to be 24 pages long).

REN

The third built-in command is REN, or rename. This lets us change the name of a file by typing:

```
A>ren <newname>=<oldname>
```

So try:

```
A>ren jim.com=fred.com
```

and then use DIR or STAT *.* to check that FRED.COM has been renamed to JIM.COM. Run JIM by:

```
A>jim *.*
```

if you need convincing.

ERA

The fourth built-in command is ERA, or erase. BE WARNED – this one is dangerous! As the name suggests, it lets us erase and effectively destroy files which we no longer want. There is no simple way to recover something which you have erased by accident! Type:

```
A>era jim.con
```

and JIM.COM will cease to be. (Check this with DIR or STAT, as before.)

USER

The fifth resident command is not one we'll use much, and I won't go into it in any detail. The USER command:

```
A>user <n>
```

where <n> is a number from 0 to 15, allows us to split a disc up into 16 different user areas. Normally we work in USER 0 without knowing anything about it.

User areas in standard CP/M 2.2 are virtually useless, so for the moment we'll ignore them. Feel free to experiment, however – you'll soon discover the limitations.

TYPE

The last command in our whistle-stop tour of the CCP is TYPE. This lets us look at the contents of any files of Ascii text on the disc. (Ascii is the standard system of representing written text in computers.) Type:

```
A>type dump.asm
```

and we can read the text file containing the assembler source code for the transient utility DUMP.COM.

TYPE will not let us look at machine code – for instance .COM program files. Try it if you want and see what happens – the results tend to be spectacular.

Notice also that in general Amsdos Basic programs cannot be TYPEd successfully – this is because they are not stored as Ascii, but use a special coded format TYPE can't read!

So far I've explained a little about the different "bits" of CP/M, and what they each do. Figure II shows where each of them live in the Amstrad's memory while CP/M is running.

The bottom 256 bytes, page zero, make up the system parameter area. This contains a lot of data useful to both CP/M and programs.

The next, and largest, area of memory, starting at &100, is the transient program area where all programs, including any you might write, are loaded by the CCP when you type a transient command.

Above this is the CCP itself. However this area of memory is also available to a program as an "extra" bit of TPA, as once the CCP has loaded the program it is no longer needed.

When the program finishes a warm boot will occur to reload the CCP in case it has been overwritten.

Above the CCP is the BDOS, the main part of the operating system. This must never be overwritten by a program else the system will crash.

Above the BDOS lives the BIOS or machine specific parts of the system – the various jumpblocks and variables required to glue things together, and, at the top, the BIOS ROM lurking under the screen RAM.

● **Next month we'll move on to look at the programs present on disc and discover some tricks to make life easier if you only have one disc drive.**

Pitch in ... and give your tunes some tone

IF you've been following the series so far, by now you should be familiar with our old favourite:

```
SOUND 1,200,100,5
```

Hopefully you'll be able to see that this tells the Amstrad to make a sound on channel A that lasts for one second. The pitch of the note will be 200 and its volume will be 5.

As you'll recall, the SOUND command has the structure:

```
SOUND channel,pitch,duration,volume
```

and by altering these parameters we alter the resulting noise.

Things are never quite that simple and last month we saw that the volume of the note played could be changed by something called a volume envelope.

We can have 15 of these volume envelopes, defined by the ENV command and called up by attaching another parameter to the end of our basic SOUND statement.

So, by combining:

```
ENV 1,5,2,20
```

and:

```
SOUND 1,200,100,5,1
```

we get a note that lasts for one second, its volume getting louder as it plays.

The structure of the ENV command is:

```
ENV N,P,Q,R
```

where N just labels the envelope, P gives the number of steps, Q the volume change per step and R specifies how long each step will last.

Again however, things are never quite that simple and we saw that the ENV command could take up to 16 parameters in the form:

```
ENV N,P1,Q1,R1,P2,Q2,R2,
    P3,Q3,R3,P4,Q4,R4,
    P5,Q5,R5
```

This surfeit of parameters allows the volume envelope to have up to five stages. As if all this wasn't enough, the volume envelope isn't the only envelope that can affect our basic SOUND command.

There's another envelope called the pitch – or tone – envelope which affects the pitch of the note – how high or low it sounds. Before we go into how it works, let's hear it in action.

First, define a pitch envelope with:

```
ENT 1,5,10,20
```

Next type in:

```
SOUND 1,200,100,5,0,1
```

and press Enter.

If you've typed it all in correctly you should hear a noise that lasts for one second, getting lower and lower in pitch.

What's happened is that the final 1 in the SOUND command has called the pitch envelope labelled 1. This

	Channel	Pitch	Duration	Volume without envelope	Volume with envelope	Volume Envelope	Pitch Envelope
Range	1=A 2=B 4=C	0 to 4095	1 to 32767	0 to 7	0 to 15	0 to 15	0 to 15
Default	none	none	20	4	12	0	0

Table I: Parameter ranges for SOUND command

Parameter	Number S	Number of steps in section T	Pitch change per step V	Time length of each step W
Range	0 to 15	0 to 239	−128 to 127	0 to 255

Table II: Parameter ranges for ENT command

Part III of NIGEL PETERS' series on coaxing melodious sounds from the CPC464

previously-defined envelope then varies the pitch of the note produced by the SOUND statement in line with the pitch envelope's parameters.

You'll notice that we now have six parameters following the SOUND command. Table I shows the new parameter ranges for the SOUND command.

As you can see, the pitch envelope looks very similar to the volume envelope we dealt with previously. It takes the form:

```
ENT S,T,V,W
```

and as you might guess, S is just a number that labels the pitch envelope. You can define up to 15 of these pitch envelopes so S ranges from 1 to 15. A value of 0 leaves the note unchanged.

The T, V and W parameters again mimic those in the volume envelope but in this case they affect how the highness or lowness of the note varies, not its loudness.

The T parameter decides on the number of steps there are going to be in the pitch envelope. It can have values between 0 and 239.

The V parameter is the one that decides how much the pitch is going to vary at each step. The pitch can go either up or down, taking values between −128 and 127.

Finally the W parameter decides how long each step is to last. Measured in hundredths of a second, it can take values between 0 and 255. Table II sums up the parameters of the ENT command and the values they can take.

Now that we know what these parameters do, let's see how they worked on our old favourite sound. Figure I shows diagramatically the pitch of the note produced by:

```
SOUND 1,200,100,5
```

As you can see, the pitch stays steady at 200 for the second that the note lasts.

Now let's define a pitch envelope with:

```
ENT 1,5,10,20
```

and call it up with:

```
SOUND 1,200,100,5,0,1
```

As you'll hear, the sound descends in pitch in five steps during the second that it plays. Figure II shows the five steps of the pitch envelope graphically.

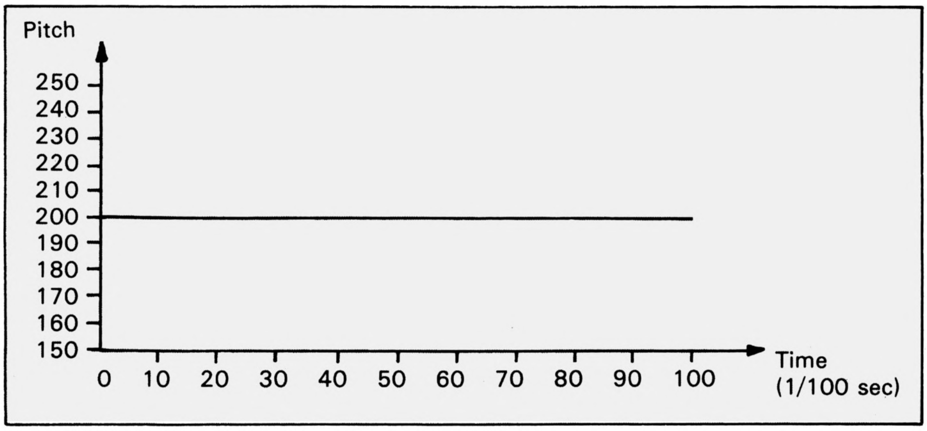


Figure I: SOUND 1,200,100,5

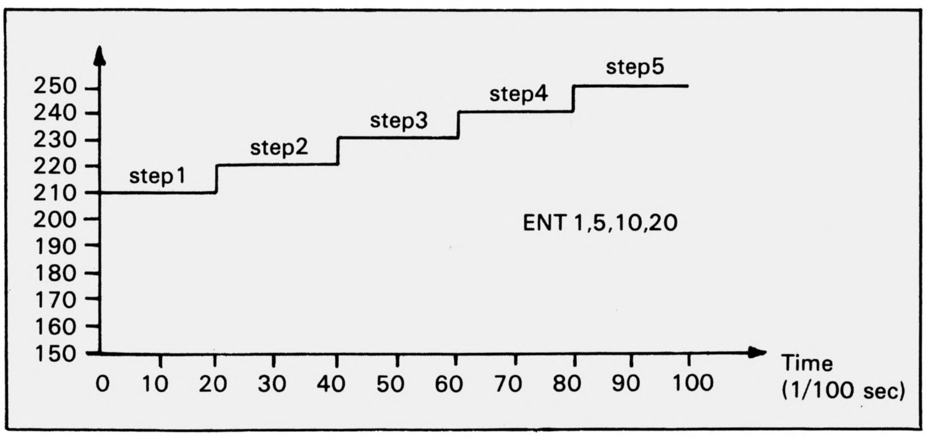


Figure II: SOUND 1,200,100,5,0,1

Let's take a look at this pitch envelope in detail. The T parameter is 5, ensuring that there will be five steps, while the W parameter of 20 ensures that each step will last for a fifth of a second.

The V parameter of 10 means that at each step 10 is added to the pitch of the note that is playing. In the case of:

```
SOUND 1,200,100,5,0,1
```

this means that there will be five

notes played with pitches of 210, 220, 230, 240 and 250. The envelope takes the pitch parameter of 200 from the SOUND command and successively adds 10 to it. As the value of the pitch parameter increases, so the note gets lower.

Notice that the pitch is incremented straight away – the sound starts at pitch 210, not 200 as you might expect. The pitch envelope takes effect immediately. Also notice that a single SOUND command has produced five notes courtesy of a previously-defined pitch envelope.

Before this we would have had to use five SOUND commands to get the same effect, as in Program I.

```
10 REM Program I
20 SOUND 1,210,20,5
30 SOUND 1,220,20,5
40 SOUND 1,230,20,5
50 SOUND 1,240,20,5
60 SOUND 1,250,20,5
```

Now however, we can get the same result by defining a pitch envelope with:

```
ENT 1,5,10,20
```

and calling it using:

```
SOUND 1,200,100,5,0,1
```

which is a lot simpler. And the same envelope can be used to vary the pitch of other notes in the same way. Try:

```
SOUND 1,100,100,5,0,1
```

which calls the same pitch envelope but starts at a higher pitch (110).

To sum up so far, we can define a pitch envelope using ENT. When this is called, it alters the pitch of the sound produced by a SOUND command.

In case you're wondering, you can have both volume and pitch envelopes operating at the same time. Try:

```
SOUND 1,200,100,5,1,1
```

and – unless you've cleared the envelopes out of your micro and have to re-enter them – you'll hear five descending notes getting louder as they get lower. The volume and pitch envelopes are working in unison.

As I said before, you can have up to 15 pitch envelopes so let's define another one with:

```
ENT 2,5,-10,20
```

Can you guess what its effect will be before you try it out on a SOUND command?

The T parameter is 5, so there will be five steps. Since the W parameter is 20, this means that each step will last for 20 hundredths of a second. The V parameter is –10 so the value of the pitch parameter will decrease by 10 for each step of the pitch envelope.

As the pitch parameter decreases in value, so the note paradoxically gets higher in pitch. So we'll get a note lasting one second, increasing in pitch by five stages. Call the envelope with:

```
SOUND 1,200,100,5,0,2
```

and hear for yourself.

Again, one simple pitch envelope has produced five notes of different pitch. If we didn't use an envelope we would have to resort to something like Program II to achieve our aims.

```
10 REM Program II
20 SOUND 1,190,20,5
30 SOUND 1,180,20,5
40 SOUND 1,170,20,5
50 SOUND 1,160,20,5
60 SOUND 1,150,20,5
```

As you can see:

```
SOUND 1,200,100,5,0,2
```

is much easier.

You'll probably have noticed that the pitch envelope expects the sound to last a certain time. So far our examples have always had the SOUND command last that amount of time. Suppose we defined a pitch envelope with:

```
ENT 3,5,20,40
```

As you can see from the T and W parameters, the envelope expects that there will be five steps and that each step will last 40 hundredths of a second. That means the whole pitch envelope will last two seconds.

But suppose the SOUND command that invokes the pitch envelope only has a duration parameter of a second? In other words, the duration of the SOUND command is less than that assumed in the pitch envelope. What happens?

As with most things in computing, the answer is to try it and see. Entering:

```
SOUND 1,200,100,5,0,3
```

will give you the answer. The noise still lasts only one second. The pitch envelope only gets through two and a half steps before it's cut off in its prime.

```
SOUND 1,200,200,5,0,3
```

which lasts two seconds, will let you hear all of the envelope's effects.

But what of the other case, where the pitch envelope lasts for a shorter time than the SOUND command? Enter:

```
ENT 4,5,-10,10
```

which defines a pitch envelope that expects to last half a second. Now call this newly-created envelope with:

```
SOUND 1,200,100,5,0,4
```

which should last one second.

As you can hear, the pitch envelope lasts for its full half second, the note rising in pitch. Then for the remaining half second the note remains at the final pitch.

The envelope has its way and then the SOUND command uses up the remaining time playing at the final pitch.

One other problem that might crop up is where the V parameter of a pitch envelope tries to take the pitch out of range.

As we know, the value of the pitch parameter can only range from 0 to 4095. So what happens if the increase or decrease of pitch in one of the envelope's steps tries to take it out of this range?

When we came across a similar problem in the volume envelope we saw that the Amstrad just wrapped round to values that were in range. This is also the case with the pitch envelope. Try:

```
ENT 5,5,-100,100
SOUND 1,300,500,5,0,5
```

and:

```
ENT 6,5,100,100
SOUND 1,3800,500,5,0,6
```

and you'll hear what I mean. The silent part occurs when the pitch parameter is equal to zero.

And that's about all for this month except to inform you that, as ever, the pitch envelope isn't as simple as I've made it seem. Like the volume envelope it can have up to five sections instead of just the one we've been using so far.

This means that instead of:

```
ENT S,T,V,W
```

the actual definition of a pitch envelope is:

```
ENT S,T1,V1,W1,T2,V2,W2,
    T3,V3,W3,T4,V4,W4,
    T5,V5,W5
```

Once again we've got a huge beast with 16 parameters. And once again let me tell you that it's not as bad as it looks.

Although we've got five sections each behaves exactly the way as the first one we've been looking at. The difference is that instead of T, V and W the first section has parameters T1, V1 and W1, the second T2, V2 and W2 and so on. Figure III shows how the parameters relate to the sections.

Although you can have five sections in a pitch envelope – as should be obvious from the above – you don't have to have all five in use. For illustration let's take a pitch envelope with three sections, such as the one defined with:

```
ENT 1,5,10,20,5,-5,20,5,5,20
```

This pitch envelope has the label 1 and is in three sections lasting a total of three seconds. Taking each section in turn you should be able to see what happens. When you think you've figured it out call the envelope with:

```
SOUND 1,200,300,5,0,1
```

and see if you were right.

Don't be worried by all the parameters of the pitch envelope. So long as you don't let it know you're afraid of it you'll be all right.

And to give you practice I leave you with Program III to help create your own pitch envelopes and hear what they're like.

```
10 REM PROGRAM III
20 REM TONE ENVELOPE
30 DIM T(5),V(5),W(5)
40 WHILE -1
50 MODE 1
60 INPUT "How many sections in tone envelope?", sections
70 IF sections<1 OR sections >5 THEN CLS:GOTO 60
80 CLS
90 FOR loop=1 TO sections
100 LOCATE 3,5:PRINT "Section" loop
110 LOCATE 3,8:PRINT "Number of steps?"
120 LOCATE 30,8:INPUT T(loop)
130 IF T(loop)<0 OR T(loop)>239 THEN LOCATE 30,8:PRINT SPACE$(8):GOTO 120
140 LOCATE 3,13:PRINT "Size of each step?"
150 LOCATE 30,13:INPUT V(loop)
160 IF V(loop)<-128 OR V(loop)>127 THEN LOCATE 30,13:PRINT SPACE$(8):GOTO 150
170 LOCATE 3,18:PRINT "Duration of step?"
180 LOCATE 30,18:INPUT W(loop)
190 IF W(loop)<0 OR W(loop)>255 THEN LOCATE 30,18:PRINT SPACE$(8):GOTO 180
200 LOCATE 14,23:PRINT "PRESS SPACE"
210 WHILE INKEY(47)=-1:WEND:CLS
220 WHILE INKEY$<>"":WEND
230 NEXT loop
240 ENT 1,T(1),V(1),W(1),T(2),V(2),W(2),T(3),V(3),W(3),T(4),V(4),W(4),T(5),V(5),W(5)
250 duration=T(1)*W(1)+T(2)*W(2)+T(3)*W(3)+T(4)*W(4)+T(5)*W(5)
260 SOUND 1,200,duration,5,0,1
270 CLS
280 duration$=RIGHT$(STR$(duration),LEN(STR$(duration))-1)
290 PRINT "SOUND 1,200,";duration$;",5,0,1"
300 FOR loop=1 TO sections
310 loop$=RIGHT$(STR$(loop),1)
320 PRINT "T(";loop$;")  ";T(loop)
330 PRINT "V(";loop$;")  ";V(loop)
340 PRINT "W(";loop$;")  ";W(loop)
350 NEXT
360 LOCATE 14,23:PRINT "PRESS SPACE"
370 WHILE INKEY(47)=-1:WEND:CLS
380 WEND
```

And that's all for this month, though it's not the end of our treatment of envelopes. After all, there are some very important questions needing an answer. Such as why have envelopes in the first place?

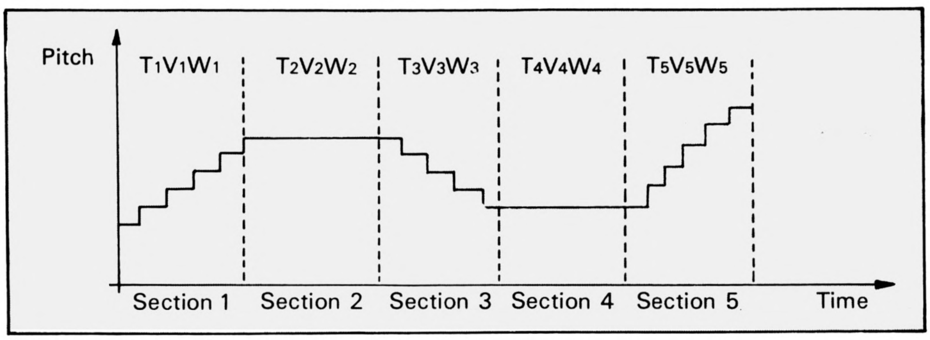


Figure III: Parameters for all five sections of a pitch envelope

ASCII stands for the American Standard Code for Information Exchange. The CPC464 can only deal with numbers. Therefore letters, punctuation marks and symbols have to be stored in memory as numbers. Obviously there has to be a list of which number stands for which symbol and Ascii is the one used in most micros. You can get the chart reproduced at the foot of the page using program I.

Get the facts at your fingertips . . . with the third of our ready reference charts

Ascii	Represents characters in number form.
CHR$	Gives the character from an Ascii code.
ASC	Gives the Ascii code for a character.

```
10 PRINT" Code";" ";"Character"
20 FOR loop=33 TO 99
30 PRINT " ";loop;"     "CHR$(loop)
40 NEXT loop
50 FOR loop=100 TO 126
60 PRINT loop;"     "CHR$(loop)
70 NEXT loop
```

Program I

```
10 LET variable=68
20 PRINT CHR$(variable)
```

Program II

```
10 LET a=63
20 LET b=6
30 PRINT CHR$(a+b)
```

Program III

CHR$

The keyword CHR$ allows you to find the character represented by a particular Ascii code. It takes the form:

```
CHR$(integer)
```

Entering:

```
PRINT CHR$(67)
```

gives the letter represented by 67 – which is C. CHR$ can also take a variable inside the brackets. See Program II.

It can even take an expression, as in Program III.

ASC

The keyword ASC returns the Ascii code of the first character of a string. It takes the form:

```
ASC(string)
```

If you want to know the Ascii code for C:

```
PRINT ASC("C")
```

will tell you. Note the inverted commas. ASC only returns the code of the first letter of the string.

```
PRINT ASC("CD")
```

only displays 67, the code for C. ASC will accept string variables inside the brackets:

```
10 LET a$="CD"
20 PRINT ASC(a$)
```

Notice the inverted commas aren't needed with a string variable.

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

Table I: Ascii codes and their associated characters

GO FORTH

...and here's an easy introduction

STEPHEN DEVINE points the way to a second language

MANY readers, having learned and used Basic on the Amstrad, will have quickly discovered that it is not the ideal programming language for all situations and that some applications, such as arcade-type games, call for a faster, more compact means of programming.

One solution is to write such programs in machine code, but for most people this is a very difficult and time-consuming task. A far easier and more enjoyable way is to use one of the many other languages which are becoming available for the Amstrad.

These include Pascal, Forth and Logo and, with the addition of a disc drive, Lisp, Prolog, Fortran, C and many, many more.

Each has its own advantages and disadvantages in different situations and, while one language may seem ideal for one particular application, it may prove to be too slow or take up too much memory in others.

What is needed is clearly a good all rounder, ideally a language which is fast enough for most requirements and not too wasteful of memory, while also being relatively easy to learn to use.

The language most fitting these requirements is Forth, and it is not surprising that it is the most popular second language among home computer users. It is a fast, compact, general purpose language, ideally suited to a variety of uses, and despite its unusual vocabulary and structure it is by no means difficult to learn.

Forth started life around 1969 and was originally used to control the complex movements of large telescopes. Since then it has been used by a wide and varied spectrum of users for an equally wide variety of uses.

Its main strength stems from the fact that although it contains many of the superior programming features of high level languages – such as loop structures and complex conditionals – it produces extremely compact programs which run at high speeds, typically 10 times as fast as Basic. In addition to this, you can modify and extend the language to suit any application you might require.

Your first task of course, before you can try out any of the following examples, will be to type in the program, which is a complete implementation of Forth for use on the Amstrad.

Note that this program does not provide you with a real version of Forth, it merely simulates its operation. It works by converting each new word into a special internal format which is then interpreted by Basic whenever the word is executed.

Since Basic is itself interpreted by the Amstrad this means that any Forth programs created will run extremely slowly. However this version is almost identical to real Forth systems and, as such, it will enable you to experiment with this powerful language, using the techniques outline in this article. You will then be able to decide if Forth is the language for you and, if so, you can then buy one of the commercial versions available for the Amstrad.

Forth is not without its drawbacks, and these are mainly due to its unusual vocabulary and its reversed method of operation. All commands in Forth, or words as they are known, expect to have their arguments – the variables or numbers which they operate on – given to them *before* each command – not after as is the case with most languages.

For example, if we wanted to add two numbers together in Basic and print out their result, we would use a statement in the form:

```
PRINT 3 + 8
```

but in Forth we would write this as:

```
3 8 + .
```

where the dot (.) is the Forth word for print. This form of arithmetic is known as Reverse Polish Notation, or RPN for short, and operates in conjunction with an arithmetic stack of numbers.

The way in which Forth interprets the above command is as follows. First the numbers to be added are put on to the stack – first the 3 then the 8 – then the Forth word + is executed.

This, like most Forth words, operates by taking numbers off the stack, processing them in some way, and returning the result to the stack for use by subsequent words.

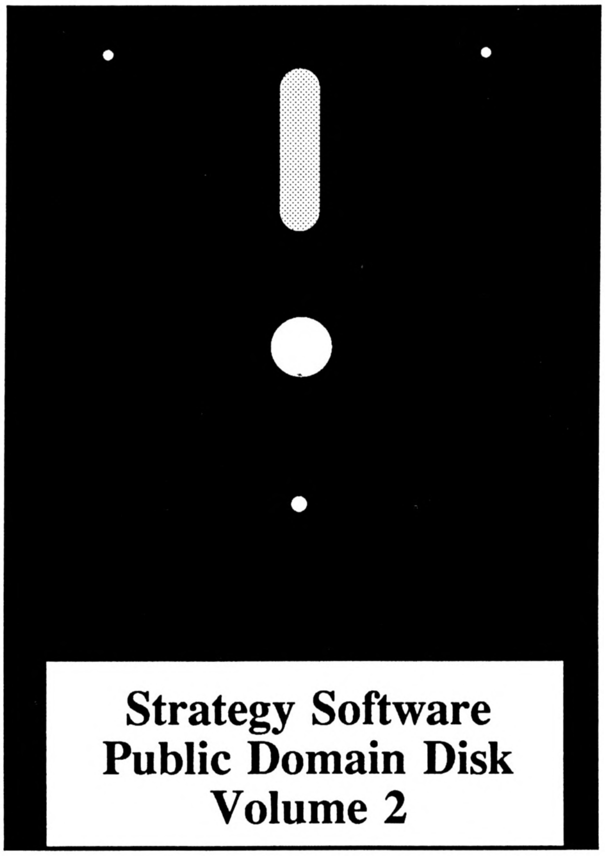
In this case the top two numbers are removed from the stack and added together and the result of this addition is then put back on to the stack.

The next word – . – then removes the topmost number from the stack and prints it out to the screen. Forth then prints the message OK to show that the statement has been executed without error. Note that all Forth words must be separated from each other by a space.

These operations leave the stack in exactly the same state as it was before. This is a very important feature since it allows subsequent words to operate on values which were put on the stack before the above sequence of instructions was executed.

The top of the stack always contains the last number put there

Public Domain Disk Volume 2



Only $19.95

Price includes P & P

Send cheque or order using Bankcard or Mastercard

Please quote Catalog #2802

Strategy Software
P.O. Box 11
Blackmans Bay
Tasmania 7152

[002] 29 4377

Suitable for both CPC and PCW computers

M9CPCALL.COM	All CPC range modem program. CP/M 2.2 & 3.0 (no split baud rates). This program sets the baud rate to 300/300 on startup.
M9NOSET.COM	All CPC range modem program. CP/M 2.2 & 3.0. To obtain split baud rates, set them up before entering this program.
MDM8000.COM	A PCW modem program - CP/M 3.0 on PCWs only.
MDM730.DOC	Explains the features and foibles of the modem programs with notes that show where M7** & M8** differ.
M7BELL.MSG M7FNK.DOC M7FNK.NOT M7LIB.DOC M7RUB.MSG MDM730.MSG MDM730.NOT	Files in this group consist short notes, messages and doc files for people who wish to bring up the modem program from scratch.
MODEM.COM	Stripped down version - useful for talking between two machines.
MODEM.DQC	Squeezed doc file for above.

From Page 33

and if a new value is put on to the stack the old number is pushed down so that the new value is now the topmost item.

You may well be wondering how Forth knew that we only wanted to add two numbers together and not three or four or even more. The answer to this is that the word + always operates on exactly two numbers and always returns just one result. This is also true of most arithmetic operations in Forth, such as multiplication and division.

This does not prevent us from using complex expressions in Forth, it just means that we have to be careful in deciding how to express them. For example, if we wanted Forth to evaluate the expression:

```
15 + 2 * 9
```

we would start by multiplying 2 by 9 to get an intermediate result which we add to 15 to produce the final value.

An alternative method is to add 15 and 2 together and then multiply this result by 9, but this will give us a different answer and is not the usual way to evaluate expressions of this type – Basic, and most other languages, would use the first method.

To multiply 2 by 9 in Forth we must type:

```
2 9 *
```

which will leave the result – 18 – on the top of the stack. This could be tested by printing the top stack number using . but since we need this value for the next part of the calculation, we will leave it where it is.

Next we must add 15 to the value on top of the stack by typing:

```
15 +
```

This leaves the result as the new top stack item, which can then be printed out. So our complete evaluation becomes:

```
2 9 * 15 + .
```

which prints out the correct answer of 33. Note that most versions of Forth use only whole number, or integer, arithmetic, with numbers usually in the range –32768 to +32767, and cannot normally handle floating-point or decimal numbers.

This method of arithmetic is not as complicated as it may seem. The best way to learn it is to try using it in Forth and, after some practice, you will find it almost as easy to use as normal arithmetic and just as powerful.

‘In the end the complete program might consist of just one word which need only be typed for all the associated words to be executed’

The real power of Forth, however, comes from being able to add new words to its vocabulary and to re-define existing ones.

Supposing you preferred to use English words for arithmetic, instead of the symbols +, –, and so on and would also like to use Basic's PRINT in place of Forth's dot (.). All you have to do is type:

```
: ADD + ;
: SUBTRACT - ;
```

and:

```
: MULTIPLY * ;
```

to create your new arithmetic words, and:

```
: PRINT . ;
```

to enable you to use a standard PRINT command.

Our arithmetic expression could now be evaluated using:

```
2 9 MULTIPLY 15 ADD PRINT
```

which would have exactly the same effect as the previous example.

In fact the previous example would still work, since we haven't actually re-defined the original arithmetic words but have simply created additional names for them.

All new Forth words are defined in this way, by bracketing the statements between a colon and semi-colon. The colon indicates to Forth that you are about to define a new word and it must be followed by the word's name.

Next come the actual Forth words which will be executed when the new word is used and these may be either standard words – such as the + and – of the previous example – or they can be other new words which have already been defined. Finally the whole definition is ended by a semi-colon. Quite complex words can be built up in this way, with some words being used in the definition of other words, which are themselves used in other definitions, and so on.

This is exactly how a program is constructed in Forth – by splitting each task into a series of smaller tasks. These can then be defined easily using standard Forth words, and all linked together in later definitions.

In the end the complete program might consist of just one word which need only be typed for all the associated words to be executed.

Forth programs, of course, do not just consist of arithmetic expressions – as in the previous examples – and many other standard words are provided.

These basic words, which come with all versions of Forth, are known as the core vocabulary, and include the facilities for implementing such features as variables and loops so that complex programs may be written.

You can see the complete vocabulary displayed by typing:

```
*VLIST
```

● *That's enough to be going on with this month. Next time we'll see how these Forth words are used.*

```
1 REM *****************
2 REM * Amstrad Forth  *
3 REM *       by       *
4 REM * Stephen Devine *
5 REM *****************
6 REM
10 REM (c) Computing with the Amstrad
100 REM Initialisation
110 MODE 2:BORDER 13:INK 1,0:INK 0,13
120 forth$=CHR$(12)+"Fred Forth V1.1"
130 OPENOUT "dummy":MEMORY HIMEM-1:CL
OSEOUT
140 temp=0:DEFINT a-z
150 DIM w$(130),p(130),beg(40),ff(40)
```

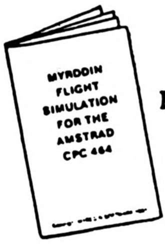
→

3D LANDMARKS YOU CAN FLY AROUND

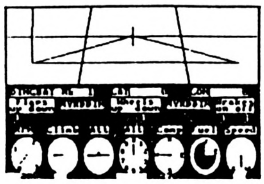
SUPERB REAL TIME SIMULATION

MYRDDIN FLIGHT SIMULATION

AMSTRAD CPC 464

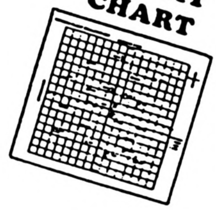


MANUAL

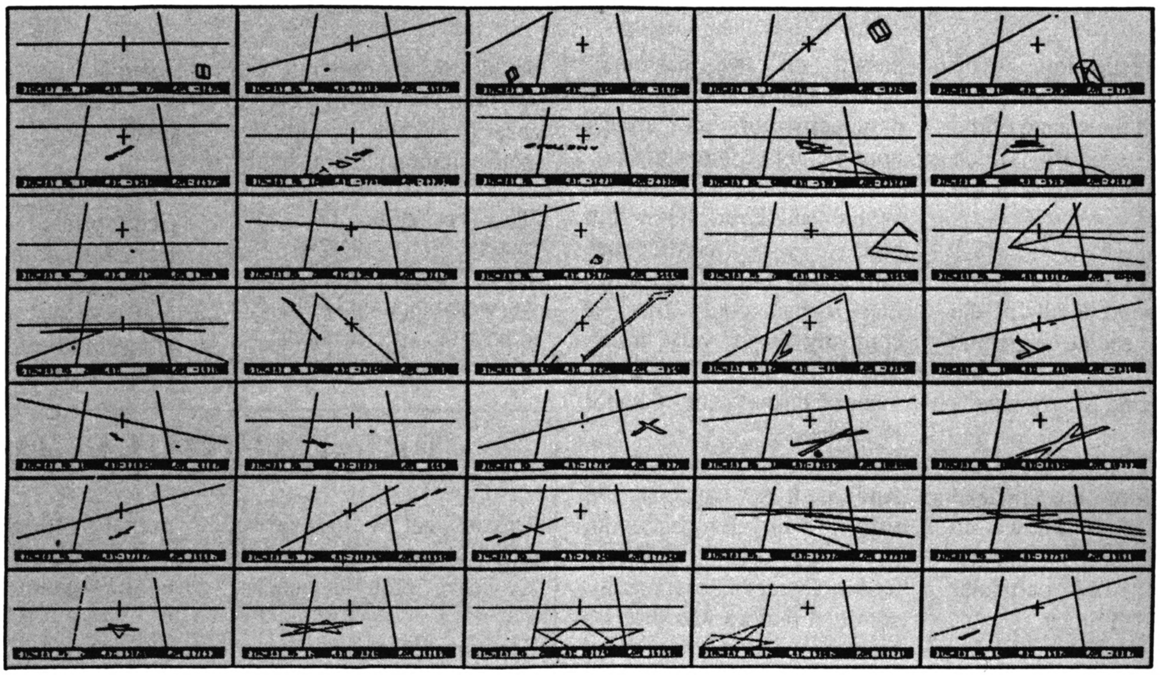


FULL SCREEN DISPLAY

FLIGHT CHART



Here are some screens from a typical flight showing the view from the cockpit (top half of screen) produced as printouts of the actual simulator.



A real time simulation with 3D graphics uses a massive 64000 x 64000 longitude & latitude flying area, making each flight completely different. Developed under pilot instruction to give realistic flight effect. The view through the cockpit gives moving 3D graphics.

Comprehensive instrument panel with moving needle meters & digital displays. 15 aircraft types with varying control sensitivities & speeds of between 100 – 500 knots.

3 runways available for refuelling, take off & landing. Ground and landmark orientation correct with all flying attitudes (rolls etc.).

The 3D graphics are still accurate when you fly upside down.

3D landmarks you can fly around.

Comes complete with manual & fully detailed chart of landmarks & airfields.

Joystick or keyboard operation.

Strategy Software
P.O. Box 11
Blackmans Bay
Tasmania 7152
[002] 29 4377

Tape $17.95 **Disk $29.95**

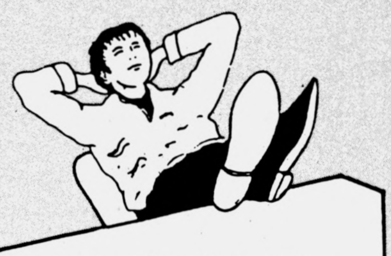
From Page 35

```
,df(40),loop(40),ll(40),li(40)
160 er$(0)="OK":er$(1)="Stack Underflow":er$(2)="Empty Stack"
170 er$(3)=" already defined":er$(4)=" - Illegal variable name":er$(5)=" - Bad Word":er$(6)="Stack Full":er$(7)="Return Stack Full"
180 spmax=100:DIM s(spmax):sp=-1
190 cvn=58:DIM cvoc$(cvn),dsp(cvn,1)
200 FOR i=0 TO cvn:READ cvoc$(i),dsp(i,0),dsp(i,1):NEXT
210 cvoc$(5)="."+CHR$(34)
220 umax=100:vmax=100:DIM uvoc$(umax),uvex$(umax),var$(vmax),var(vmax)
230 uvn=-1:vrn=-1:PRINT forth$:PRINT:PRINT er$(0)
240 ON ERROR GOTO 2530
250 REM Input Commands
260 w$="":er=0:LINE INPUT ln$:IF ln$="" THEN 690 ELSE IF LEN(ln$)>240 THEN PRINT"Line too long":GOTO 260
270 WHILE ASC(ln$)=32:IF LEN(ln$)>1 THEN ln$=RIGHT$(ln$,LEN(ln$)-1):WEND ELSE 690
280 WHILE RIGHT$(ln$,1)=CHR$(32):ln$=LEFT$(ln$,LEN(ln$)-1):WEND
290 ln$=UPPER$(ln$):IF ASC(ln$)=ASC("*") THEN IF LEN(ln$)>1 AND LEFT$(ln$,2)<>"* " THEN GOSUB 1700:IF er THEN 260 ELSE w$="":GOTO 690
300 ln$=ln$+CHR$(32):x$="":q=1:wn=-1:comp=0
310 WHILE q<LEN(ln$)
320 p=q:WHILE MID$(ln$,q,1)<>" ":q=q+1:WEND
330 w$=MID$(ln$,p,q-p):IF w$=":" THEN IF wn=-1 AND RIGHT$(ln$,2)="; " THEN comp=-1:GOTO 640 ELSE PRINT"Bad definition":GOTO 260
340 FOR i=cvn TO 0 STEP-1:IF cvoc$(i)<>w$ THEN NEXT:GOTO 400
350 IF comp THEN IF wn=0 THEN er=3:GOTO 690
360 IF wn>=0 THEN IF w$(wn)="VARIABLE" THEN er=4:GOTO 690
370 x$=x$+CHR$(0)+CHR$(i+14):IF w$<>"."+CHR$(34) THEN 640
380 te=INSTR(q,ln$,CHR$(34)+CHR$(32)):IF te=0 THEN PRINT".";CHR$(34);" without ";CHR$(34):GOTO 260
390 x$=x$+MID$(ln$,q+1,te-q-1)+CHR$(4):q=te+1:GOTO 640
400 FOR i=uvn TO 0 STEP-1:IF uvoc$(i)<>w$ THEN NEXT:GOTO 440
410 IF comp THEN IF wn=0 THEN er=3:GOTO 690
420 IF wn>=0 THEN IF w$(wn)="VARIABLE" THEN er=4:GOTO 690
430 x$=x$+CHR$(1)+CHR$(i+14):GOTO 640
440 FOR i=vrn TO 0 STEP-1:IF var$(i)<>w$ THEN NEXT:GOTO 480
450 IF comp THEN IF wn=0 THEN er=3:GOTO 690
460 IF wn>=0 THEN IF w$(wn)="VARIABLE" THEN er=3:GOTO 690
470 x$=x$+CHR$(2)+CHR$(i+14):GOTO 640
480 FOR i=1 TO LEN(w$)
490 IF i=1 AND (ASC(w$)=ASC("+") OR ASC(w$)=ASC("-")) AND LEN(w$)>1 THEN 510
500 IF MID$(w$,i,1)<"0" OR MID$(w$,i,1)>"9" THEN 560
510 NEXT i
520 IF comp AND wn=0 THEN er=5:GOTO 690
530 IF wn>=0 THEN IF w$(wn)="VARIABLE" THEN er=5:GOTO 690
540 IF VAL(w$)>32767 OR VAL(w$)<-32767 THEN PRINT"Number ";w$;" too large":GOTO 260
550 x$=x$+CHR$(3)+w$+CHR$(4):GOTO 640
560 IF wn<0 THEN 590 ELSE IF w$(wn)<>"VARIABLE" THEN 590
570 IF comp AND w$=w$(1) THEN er=3:GOTO 690
580 x$=x$+w$+CHR$(4):GOTO 640
590 IF w$<>";" THEN 610
600 IF comp=0 OR q<>LEN(ln$) THEN PRINT"Illegal semi-colon":GOTO 260 ELSE 640
610 IF comp THEN IF wn=0 THEN 640
620 IF comp THEN IF w$<>w$(1) THEN er=5:GOTO 690 ELSE x$=x$+CHR$(1)+CHR$(uvn+15):GOTO 640
630 er=5:GOTO 690
640 wn=wn+1:w$(wn)=w$
650 WHILE MID$(ln$,q,1)=" ":q=q+1:WEND
660 WEND
670 x$=x$+CHR$(13):IF comp THEN 720 ELSE 760
680 REM Error Routine
690 IF POS(#0)>1 THEN PRINT CHR$(32);
700 PRINT w$;er$(er):GOTO 260
710 REM Compile New Word
720 IF wn<3 THEN PRINT"Insufficient definition":GOTO 260
730 uvn=uvn+1:uvoc$(uvn)=w$(1):uvex$(uvn)=x$
740 w$="":GOTO 690
750 REM Execute Commands
760 ln=0:w$(ln)=x$:er=0
770 GOSUB 780:w$="":GOTO 690
780 p(ln)=1:ff(ln)=0:df(ln)=0
790 WHILE MID$(w$(ln),p(ln),1)<>CHR$(13)
800 class=ASC(MID$(w$(ln),p(ln),1)):p(ln)=p(ln)+1
810 IF class<>0 THEN 920
820 word=ASC(MID$(w$(ln),p(ln),1))-14:p(ln)=p(ln)+1
830 IF ff(ln)=0 OR word=37 OR word=39 OR word=40 THEN 860
840 IF word=5 OR word=32 THEN WHILE ASC(MID$(w$(ln),p(ln),1))<>4:p(ln)=p(ln)+1:WEND:p(ln)=p(ln)+1
850 GOTO 1060
860 IF sp+dsp(word,1)<-1 OR sp+dsp(word,1)>spmax THEN er=1:GOTO 1070
870 IF sp-dsp(word,0)<-1 THEN er=2:GOTO 1070
880 sp=sp+dsp(word,1)
890 IF word <43 THEN ON word+1 GOSUB 1090,1110,1120,1130,1140,1150,1160,1170,1180,1190,1200,1210,1220,1230,1240,1250,1260,1270,1280,1290,1300,1310,1320,1330,1340,1350,1360,1370,1380,1390,1400,1410,1420,1430,1440,1450,1460,1470,1480,1490,1500,1510,1520
900   IF word>42 THEN ON word-42 GOSUB 1530,1540,1550,1560,1570,1580,1590,1600,1610,1620,1630,1640,1650,1660,1670,1680
910 IF er=0 THEN 1060 ELSE 1070
920 IF class<>1 THEN 970
930 word=ASC(MID$(w$(ln),p(ln),1))-14:p(ln)=p(ln)+1
940 IF ff(ln) THEN 1060
950 IF ln<34 THEN ln=ln+1:w$(ln)=uvex$(word) ELSE er=7:RETURN
960 GOSUB 780:IF ln=0 OR er=0 THEN 1060 ELSE RETURN
970 IF class<>2 THEN 1010
980 word=ASC(MID$(w$(ln),p(ln),1))-14:p(ln)=p(ln)+1
990 IF ff(ln) THEN 1060
1000 sp=sp+1:s(sp)=@var(word):GOTO 1060
1010 IF class<>3 THEN er=1:GOTO 1070
1020 p=p(ln):WHILE ASC(MID$(w$(ln),p,1))<>4:p=p+1:WEND
1030 v=VAL(MID$(w$(ln),p(ln),p-p(ln)+1)):p(ln)=p+1
1040 IF ff(ln) THEN 1060
1050 sp=sp+1:s(sp)=v
1060 WEND
1070 ln=ln-1:RETURN
1080 REM Command List
1090 temp!=s(sp+1):IF temp!<0 THEN temp!=temp!+65536
1100 POKE s(sp+2),temp!-256*INT(temp!
```

From Page 37

```
/256):POKE s(sp+2)+1,INT(temp!/256):RETURN
1110 s(sp)=s(sp)*s(sp+1):RETURN
1120 s(sp)=s(sp)+s(sp+1):RETURN
1130 s(sp)=s(sp)-s(sp+1):RETURN
1140 PRINT s(sp+1);CHR$(8);:RETURN
1150 WHILE ASC(MID$(w$(ln),p(ln),1))<>4:PRINT MID$(w$(ln),p(ln),1);:p(ln)=p(ln)+1:WEND:p(ln)=p(ln)+1:RETURN
1160 s(sp)=INT(s(sp)/s(sp+1)):RETURN
1170 temp=s(sp):s(sp)=INT(s(sp-1)/s(sp)):s(sp-1)=s(sp-1)-s(sp)*temp:RETURN
1180 s(sp)=(s(sp)<0):RETURN
1190 s(sp)=(s(sp)=0):RETURN
1200 s(sp)=(s(sp)<s(sp+1)):RETURN
1210 s(sp)=(s(sp)=s(sp+1)):RETURN
1220 s(sp)=(s(sp)>s(sp+1)):RETURN
1230 temp!=PEEK(s(sp+1))+256*PEEK(s(sp+1)+1):IF temp!>32767 THEN temp!=temp!-65536:PRINT temp!;CHR$(8);:RETURN ELSE PRINT temp!;CHR$(8);:RETURN
1240 temp!=PEEK(s(sp))+256*PEEK(s(sp)+1):IF temp!>32767 THEN s(sp)=temp!-65536:RETURN ELSE s(sp)=temp!:RETURN
1250 s(sp)=ABS(s(sp)):RETURN
1260 s(sp)=s(sp) AND s(sp+1):RETURN
1270 s(sp)=PEEK(s(sp)):RETURN
1280 PRINT:RETURN
1290 RETURN
1300 s(sp)=s(sp-1):RETURN
1310 PRINT CHR$(s(sp+1));:RETURN
1320 in$=INKEY$:IF in$="" THEN 1320 ELSE s(sp)=ASC(in$):RETURN
1330 s(sp)=MAX(s(sp),s(sp+1)):RETURN
1340 s(sp)=MIN(s(sp),s(sp+1)):RETURN
1350 s(sp)=-s(sp):RETURN
1360 s(sp)=s(sp) MOD s(sp+1):RETURN
1370 s(sp)=s(sp) OR s(sp+1):RETURN
1380 s(sp)=s(sp-2):RETURN
1390 PRINT" ";:RETURN
1400 PRINT USING "&";SPACE$(s(sp+1)-256*INT(s(sp+1)/256));:RETURN
1410 temp=s(sp):s(sp)=s(sp-1):s(sp-1)=temp:RETURN
1420 vrn=vrn+1:var(vrn)=s(sp+1):WHILE ASC(MID$(w$(ln),p(ln),1))<>4:var$(vrn)=var$(vrn)+MID$(w$(ln),p(ln),1):p(ln)=p(ln)+1:WEND:p(ln)=p(ln)+1:RETURN
1430 s(sp)=s(sp) XOR s(sp+1):RETURN
1440 beg(ln)=p(ln):RETURN
1450 IF s(sp+1)=0 THEN p(ln)=beg(ln):RETURN ELSE RETURN
1460 IF s(sp+1)<>0 THEN RETURN ELSE ff(ln)=-1:RETURN
1470 IF ff(ln) THEN ff(ln)=0:RETURN ELSE p(ln)=beg(ln):RETURN
1480 IF s(sp+1)<>0 THEN RETURN ELSE ff(ln)=-1:RETURN
1490 ff(ln)=0:RETURN
1500 ff(ln)=-1-ff(ln):RETURN
1510 FOR i=0 TO uvn:uvoc$(i)="":uvex$(i)="":NEXT:uvn=-1:PRINT forth$:PRINT
1520 FOR i=0 TO vrn:var$(i)="":var(i)=0:NEXT:vrn=-1:RETURN
1530 temp=s(sp-2):s(sp-2)=s(sp-1):s(sp-1)=s(sp):s(sp)=temp:RETURN
1540 IF NOT df(ln) THEN df(ln)=-1:loop(ln)=p(ln):ll(ln)=s(sp+1):li(ln)=s(sp+2):RETURN ELSE RETURN
1550 li(ln)=li(ln)+1:IF li(ln)<ll(ln) THEN p(ln)=loop(ln):RETURN ELSE df(ln)=0:RETURN
1560 s(sp)=li(ln):RETURN
1570 CLG s(sp+1):RETURN
1580 DRAW s(sp+2),s(sp+1):RETURN
1590 DRAWR s(sp+2),s(sp+1):RETURN
1600 temp!=FRE(""):IF temp!>32767 THEN s(sp)=temp!-65536:RETURN ELSE s(sp)=temp!:RETURN
1610 MOVE s(sp+2),s(sp+1):RETURN
1620 MOVER s(sp+2),s(sp+1):RETURN
1630 PLOT s(sp+2),s(sp+1):RETURN
1640 PLOTR s(sp+2),s(sp+1):RETURN
1650 s(sp)=RND*32768:RETURN
1660 s(sp)=TEST(s(sp+1),s(sp)):RETURN
1670 s(sp)=TESTR(s(sp+1),s(sp)):RETURN
1680 x=XPOS:y=YPOS:PLOT 800,800,s(sp+1):MOVE x,y:RETURN
1690 REM Process Editing Commands
1700 er=0:w$=""
1710 IF ln$="*VLIST" THEN FOR i=cvn TO 0 STEP -1:PRINT cvoc$(i);"  ";:NEXT:PRINT:RETURN
1720 IF ln$<>"*LIST" THEN 1780
1730 FOR i=uvn TO 0 STEP -1
1740 PRINT uvoc$(i);"  ";
1750 IF INKEY$="" THEN 1770
1760 WHILE INKEY$="":WEND
1770 NEXT:PRINT:RETURN
1780 IF LEFT$(ln$,6)<>"*LIST " THEN 2040
1790 w$=RIGHT$(ln$,LEN(ln$)-6)
1800 WHILE ASC(w$)=32:w$=RIGHT$(w$,LEN(w$)-1):WEND
1810 FOR i=uvn TO 0 STEP -1:IF w$<>uvoc$(i) THEN NEXT:PRINT w$;" - Unknown word":er=-1:RETURN
1820 x$=uvex$(i)
1830 WHILE ASC(x$)<>13
1840 class=ASC(x$):x$=RIGHT$(x$,LEN(x$)-1)
1850 IF class<>0 THEN 1920
1860 word=ASC(x$)-14:x$=RIGHT$(x$,LEN(x$)-1)
1870 PRINT cvoc$(word);" ";
1880 IF word<>5 AND word<>32 THEN 2020
1890 WHILE ASC(x$)<>4:PRINT LEFT$(x$,1);:x$=RIGHT$(x$,LEN(x$)-1):WEND
1900 x$=RIGHT$(x$,LEN(x$)-1):IF word=5 AND class<>3 THEN PRINT CHR$(34);
1910 PRINT" ";:GOTO 2020
1920 IF class<>1 THEN 1960
1930 word=ASC(x$)-14:x$=RIGHT$(x$,LEN(x$)-1)
1940 PRINT uvoc$(word);" ";
1950 GOTO 2020
1960 IF class<>2 THEN 2000
1970 word=ASC(x$)-14:x$=RIGHT$(x$,LEN(x$)-1)
1980 PRINT var$(word);" ";
1990 GOTO 2020
2000 IF class<>3 THEN PRINT er$(1):GOTO 2030
2010 GOTO 1890
2020 WEND
2030 PRINT:RETURN
2040 IF LEFT$(ln$,8)<>"*FORGET " THEN 2130
2050 w$=RIGHT$(ln$,LEN(ln$)-8)
2060 WHILE ASC(w$)=32:w$=RIGHT$(w$,LEN(w$)-1):WEND
2070 FOR i=uvn TO 0 STEP -1:IF w$<>uvoc$(i) THEN NEXT:GOTO 2100
2080 FOR j=i TO uvn:uvoc$(j)="":exec$(j)="":NEXT
2090 uvn=i-1:RETURN
2100 FOR i=vrn TO 0 STEP -1:IF w$<>var$(i) THEN NEXT:PRINT w$;" - Unknown word":er=-1:RETURN
2110 FOR j=i TO vrn-1:var$(j)=var$(j+1):var(j)=var(j+1):NEXT
2120 vrn=vrn-1:RETURN
2130 IF ln$="*SAVE" THEN 2470
2140 IF LEFT$(ln$,6)<>"*SAVE " THEN 2290
2150 w$=RIGHT$(ln$,LEN(ln$)-6)
2160 WHILE ASC(w$)=32:w$=RIGHT$(w$,LEN(w$)-1):WEND
2170 dp=INSTR(w$,".")
2180 IF dp=0 THEN w1$=w$:w2$="4TH" ELSE w1$=LEFT$(w$,dp-1):w2$=RIGHT$(w$,LEN(w$)-dp)
2190 IF w1$="" OR LEN(w1$)>8 OR LEN(w2$)>3 THEN 2470
2200 IF w2$="" THEN w2$="4TH"
2210 w$=w1$+"."+w2$
2220 OPENOUT w$
2230 PRINT#9,uvn:PRINT#9,vrn
2240 FOR i=0 TO uvn:PRINT#9,uvoc$(i):PRINT#9,LEN(uvex$(i))
2250 FOR j=1 TO LEN(uvex$(i)):PRINT#9,ASC(MID$(uvex$(i),j,1));:NEXT j
2260 NEXT i
```

```
2270 FOR i=0 TO vrn:PRINT#9,var$(i):P
RINT#9,var(i):NEXT
2280 CLOSEOUT:RETURN
2290 IF ln$="|LOAD" THEN 2470
2300 IF LEFT$(ln$,6)<>"|LOAD " THEN 2
470
2310 w$=RIGHT$(ln$,LEN(ln$)-6)
2320 WHILE ASC(w$)=32:w$=RIGHT$(w$,LE
N(w$)-1):WEND
2330 dp=INSTR(w$,".")
2340 IF dp=0 THEN w1$=w$:w2$="4TH" EL
SE w1$=LEFT$(w$,dp-1):w2$=RIGHT$(w$,L
EN(w$)-dp)
2350 IF w1$="" OR LEN(w1$)>8 OR LEN(w
2$)>3 THEN 2470
2360 IF w2$="" THEN w2$="4TH"
2370 w$=w1$+"."+w2$
2380 OPENIN w$
2390 INPUT#9,uvn:INPUT#9,vrn
2400 ERASE uvoc$,uvex$,var$,var
2410 DIM uvoc$(umax),uvex$(umax),var$
(vmax),var(vmax)
2420 FOR i=0 TO uvn:INPUT#9,uvoc$(i):
INPUT#9,lux
2430 FOR j=1 TO lux:INPUT#9,temp:uvex
$(i)=uvex$(i)+CHR$(temp):NEXT j
2440 NEXT i
2450 FOR i=0 TO vrn:INPUT#9,var$(i):I
NPUT#9,var(i):NEXT
2460 CLOSEIN:RETURN
2470 IF ln$<>"|VARLIST" THEN PRINT "U
nknown or incomplete Command":er=-1:R
ETURN
2480 FOR i=vrn TO 0 STEP -1
2490 w$=var$(i)+CHR$(32)+STR$(var(i))
+SPACE$(2):PRINT w$;
2500 IF INKEY$="" THEN 2520
2510 WHILE INKEY$="":WEND
2520 NEXT:PRINT:RETURN
2530 er=6:IF sp>100 THEN sp=100:RESUM
E NEXT ELSE w$="":RESUME 690
2540 REM Data for Core Words
2550 DATA "!",2,-2,"*",2,-1,"+",2,-1,
"-",2,-1,".",0,-1,"?",0,0,"/",2,-1,"/
MOD",2,0,"0<",1,0,"0=",1,0,"<",2,-1
2560 DATA "=",2,-1,">",2,-1,"?",1,-1,
"@",1,0,"ABS",1,0,"AND",2,-1,"C@",1,0
,"CR",0,0,"DROP",1,-1,"DUP",1,1
2570 DATA "EMIT",1,-1,"KEY",0,1,"MAX"
,2,-1,"MIN",2,-1,"MINUS",1,0,"MOD",2,
-1,"OR",2,-1,"OVER",2,1
2580 DATA "SPACE",0,0,"SPACES",1,-1,"
SWAP",2,0,"VARIABLE",1,-1,"XOR",2,-1,
"BEGIN",0,0,"UNTIL",1,-1
2590 DATA "WHILE",1,-1,"REPEAT",0,0,"
IF",1,-1,"THEN",0,0,"ELSE",0,0,"FORTH
",0,0,"CLEAR",0,0,"ROT",3,0,"DO",2,-2
,"LOOP",0,0,"I",0,+1
2600 REM Data for Amstrad Words
2610 DATA "CLG",1,-1,"DRAW",2,-2,"DRA
WR",2,-2,"FRE",0,1,"MOVE",2,-2,"MOVER
",2,-2
2620 DATA "PLOT",2,-2,"PLOTR",2,-2,"R
ND",0,1,"TEST",2,-1,"TESTR",2,-1,"GRA
PEN",1,-1
```



Give your fingers a rest . . .

All the listings from this month's issue are available on cassette.

See our special offer on Page 67.

On your marks... get set... GO!

Test your reactions with ALAN McLACHLAN

FIND out how fast you are with our Amstrad reaction timer. Are you quicker off the mark than your friends? Does your reaction time vary as the day goes on? Kids, are you faster than your parents? Type in this listing and find out.

When you run the program a white bar will appear on the left of the screen. Watch it carefully because soon it's going to change colour, beeping every time to give you a warning.

At first it becomes red. Then, like a set of traffic lights, it will change to yellow. This tells you to be prepared because green is the next colour and you're going to have to react quickly.

As soon as it turns to green (but not before!) you have to press the space bar. The Amstrad will then tell you how fast your reactions are and what it thinks of them.

```
10 REM reaction timer
20 REM adapted by Alan McLachlan
25 BORDER 0
30 MODE 1:INK 0,1:INK 1,24:INK 2,20:I
NK 3,6:PEN 1:DIM m$(8):GOSUB 100
40 GOSUB 340
50 GOSUB 390
60 GOSUB 550
70 WHILE INKEY$="":WEND
80 RUN
90 END
100 REM######### initialise ##########
110 CLS:LOCATE 14,4:PEN 1:PRINT"REACT
ION TESTER"
120 LOCATE 14,5:PRINT"---------------
"
130 LOCATE 9,8:PEN 2:PRINT"Use this p
rogram to test"
140 LOCATE 14,10:PRINT"your reactions
."
150 LOCATE 5,12:PRINT"The white bar w
hich will appear "
160 LOCATE 8,14:PRINT"on the left of
the screen"
170 LOCATE 12,16:PRINT"will change co
lour."
180 LOCATE 4,20:PEN 3:PRINT"The seque
nce is RED, YELLOW, GREEN"
185 LOCATE 8,23:PEN 2:PRINT"Press any
 key to continue"
186 WHILE INKEY$="":WEND
187 CLS
190 LOCATE 7,6:PEN 2:PRINT"Each time
the colour changes"
200 LOCATE 10,8:PRINT"you will hear a
 beep."
210 LOCATE 11,12:PEN 1:PRINT"Press th
e space bar"
220 LOCATE 2,14:PRINT"as soon as the
colour changes to green"
230 LOCATE 12,16:PEN 3:PRINT"AT THE T
HIRD BEEP"
240 LOCATE 4,18:PRINT"Your time and r
ating will then be given"
250 LOCATE 8,23:PEN 2:PRINT"PRESS ANY
 KEY TO START TEST"
260 WHILE INKEY$="":WEND
270 CLS
275 INK 1,26:PEN 1
280 A$=STRING$(3,143)
290 PRINT:PRINT:FOR I%= 1 TO 18:PRINT
 TAB(8) A$:NEXT
300 B$=STRING$(3,32)
310 FOR I= 0 TO 7:READ M$(I):NEXT
320 I%=0:RANDOMIZE TIME
330 RETURN
340 REM#########WAIT###############
350 N%=6:GOSUB 480
355 LOCATE 18,11:PRINT"READY":SOUND 1
,100,20
360 N%=24:GOSUB 480
365 LOCATE 18,11:PRINT"STEADY":SOUND
1,75,20
370 N%=18:GOSUB 480:LOCATE 18,11:PRIN
T SPACE$(6):SOUND 1,50,50
380 RETURN
390 REM#############TEST############
400 IF INKEY$ <> "" THEN GOSUB 510:RU
N
420 TIMNOW=TIME
430 FOR I%= 1 TO 20
435 IF I% >19 OR INKEY(47)=0 THEN 460
440 LOCATE 8,I%:PRINT B$:NEXT
460 T%=INT(TIME-TIMNOW)/3
470 RETURN
480 REM##########colour##########
490 FOR I=1 TO 1000+INT(1000*RND)+1:N
EXT:INK 1,N%
500 RETURN
510 REM#########too soon#########
520 CLS:INK 1,24
530 LOCATE 1,10:PRINT"WAIT FOR THE GR
EEN LIGHT!!!":FOR I= 1 TO 1000:NEXT
540 RETURN
550 REM###########MESSAGE##########
560 CLS:INK 1,24:LOCATE 17,10:PRINT"Y
OU TOOK ":LOCATE 7,13:PRINT T%; " hun
dredths of a second":FOR DELAY = 1 TO
 500:NEXT:LOCATE 8,16:PRINT"You are "
;:PEN 2:PRINT M$(INT(T%/5))
565 IF INT(T%/5)=7 THEN FOR LOOP=1 TO
 20:SOUND 1,20,3:SOUND 1,70,3:NEXT
570 FOR x= 1 TO 3000:NEXT
580 LOCATE 8,23:INK 3,26,1:PEN 3:SPEE
D INK 50,20:PRINT"PRESS ANY KEY TO TR
Y AGAIN"
585 WHILE INKEY$="":WEND
590 RETURN
600 DATA OUT OF THIS WORLD,TOO GOOD T
O BE TRUE,EXCELLENT,VERY GOOD,GOOD,PO
OR,NEARLY ASLEEP,ASLEEP....WAKE UP!!
```

Give your fingers a rest . . .
All the listings from this month's issue are available on cassette.
See Order Form on Page 61

Back up your discs – and free valuable space

Nick Hinde describes Archiver, a disc to tape spooler for the CPC

WITH most computer systems in commercial use there is a facility to back-up program and data files to some form of tape device and have the option to restore the disc at a later date.

This provides a fairly reliable back-up for important or archived data and also frees expensive disc space.

Archiver is a program for the Amstrad CPC series which will allow whole discs to be recorded on tape and restored at a later date.

If the disc to be archived is fairly full it is a good idea to use a larger than normal cassette, such as a C60, for dumping the data, otherwise you may run out of tape halfway through.

The program is designed to work on system discs, but it could be altered for data discs quite easily by those who wish to modify the sector numbers.

When restoring a disc from tape it is important to use a disc which has nothing valuable on it as the first thing Archiver does when restoring is to wipe the disc on to which the tape data is going to be loaded.

To use the routine first type in the Basic program, Program I, and save it. If you have an assembler for the machine code use the source listing – Program II – and save the object code as **Maincode** with the origin at &9076 – the code length is &8B.

If you do not have an assembler type in Program III, save it and run it to produce the machine code, which will automatically be saved to disc as **Maincode.**

When you run Program I it will load the machine code and ask you for the date. It then gives you the options to archive or restore a disc. If you choose option 1 – archive a disc – it will ask for the disc to be archived to be installed in the drive, and a tape to be put in the cassette.

It will read the disc and check for empty spaces and write this information to the tape. Next it will read all the bytes from the useful sectors in a maximum of 45 sector chunks and save them to tape.

Bear in mind that when you delete a file from a disc you only remove its name from the directory. The program will still be physically there on the disc, although inaccessible, and the sectors it occupies will be read as live during the archiving process.

If you choose option 2 – restore a disc – the program will ask for the archive cassette and a spare disc, and will then wipe the disc and read the tape for the sector data. It then restores the new disc to the same state as when originally archived.

MAIN VARIABLES

track	Current read/write track number.
sector	Current read/write sector number.
date$	Date.
oldate$	Date last archived (read from tape on restoring).
logsec	Logical sector number counter – 1 to 360.
binlen	Length of binary file to save/load current batch of data.
block	Number of tape blocks remaining to load/save.
dsn$	Disc name being archived/restored.
start, **offset**, **count**	Markers for areas of memory to place sector information for reading/writing to/from tape.
live()	Holds a map of the live and dead disc sectors.

Program I

```
10 REM **********************
20 REM *                    *
30 REM *      Archiver      *
40 REM *                    *
50 REM *    by N.J.Hinde    *
60 REM *                    *
70 REM **********************
80 REM ***                ***
90 REM(c) Computing with the Amstrad
100 REM **********************
```

From Page 41

```
120 MEMORY &3473
130 DIM live(360):REM **This array holds flags for all live sectors**
140 vn=1.1
150 wid=80:WINDOW#1,1,wid,1,4:WINDOW#2,1,wid,5,25
160 LOAD "maincode",&9076
170 POKE &90BD,&DD:POKE &90DB,&84:POKE &90DC,&85
180 CALL &908E
190 LOCATE #1,2,1:PRINT#1,"Whole Disc Archive/Restore V";vn
200 LOCATE #2,2,4:INPUT#2,"Please enter date ";date$
210 CLS#2
220 LOCATE #2,2,4:PRINT#2,"Select option..... "
230 LOCATE #2,2,5:PRINT#2,"-------------------"
240 LOCATE #2,2,7:PRINT#2,"1. Backup disc to tape"
250 LOCATE #2,2,9:PRINT#2,"2. Restore disc from tape"
260 a$=INKEY$:IF a$="" THEN 260
270 IF a$<>"1" AND a$<>"2" THEN 260
280 ON VAL(a$) GOSUB 300,670
290 GOTO 210
300 REM **Dump Disc To Tape**
310 logsec=0:binlen=0
320 |DISC.IN:|TAPE.OUT:SPEED WRITE 1
330 CLS#2
340 LOCATE #2,2,4:PRINT#2,"Insert disc and a rewound tape"
350 LOCATE #2,2,6:PRINT#2,"Press RECORD and PLAY"
360 LOCATE #2,2,8:INPUT#2,"Enter filename for tape ";dsn$:dsn$="!"+dsn$
370 LOCATE #2,2,10:PRINT#2,"Hit any key when ready to start backup...."
380 WHILE INKEY$="":WEND:CLS#2
390 GOSUB 1280:REM **Check For Used Sectors And Put Flags In Array (live)**
400 LOCATE #2,2,4:PRINT#2,"Writing disc map to tape"
410 OPENOUT "!DISCMAP":REM **Save Disc Sector Config' To Tape**
420 FOR x=1 TO 360:WRITE#9,live(x):NEXT:WRITE#9,date$:CLOSEOUT
430 track=0:sector=65:start=&3673:logsec=0
440 CLS#2:LOCATE #2,2,4:PRINT#2,"Reading disc......"
450 FOR count=0 TO 44
460 logsec=logsec+1:IF logsec=361 THEN 580
470 IF live(logsec)=0 THEN GOSUB 1240:IF logsec=361 THEN 580 ELSE 470
480 LOCATE 2,12:PRINT"Track ";track;" Sector ";sector;
490 offset=(512*count)+start
500 GOSUB 1160:REM **Get Lo-byte Hi-Byte For Pokes**
510 POKE &907A,lobyte
520 POKE &907B,hibyte
530 POKE &90E5,track:POKE &90E4,0:POKE &90E6,sector:POKE &90E3,0
540 CALL &90B1:REM **Read Sector To Disc I/O Buffer**
550 CALL &9076:REM **Position Sector In Cassette I/O Buffer**
560 sector=sector+1:IF sector=74 THEN sector=65:track=track+1
570 NEXT count
580 CLS#2
590 LOCATE #2,2,4:PRINT#2,"Writing data to tape....."
600 size=binlen/2048
610 block=FIX(size)
620 IF block<>size THEN block=block+1
630 LOCATE #2,2,6:PRINT#2,"Saving ";block;" blocks "
640 SAVE dsn$,B,&3474,binlen:binlen=0
650 IF track<>40 THEN 440
660 RETURN
670 REM** Restore Disc From Tape**
680 logsec=0:binlen=0
690 |TAPE.IN:|DISC.OUT
700 CLS#2
710 LOCATE #2,2,4:PRINT#2,"Insert restore disc and tape"
720 LOCATE #2,2,6:PRINT#2,"Press PLAY "
730 LOCATE #2,2,8:PRINT#2,"Hit any key when ready to start restore...."
740 WHILE INKEY$="":WEND:CLS#2
750 LOCATE #2,2,4:PRINT#2,"Please wait.. ":LOCATE #2,2,6:PRINT#2,"Clearing restore disc ":GOSUB 1420:CLS#2
760 LOCATE #2,2,4:PRINT#2,"Reading disc map from tape"
770 OPENIN "!DISCMAP":REM **Load Disc Map From Tape**
780 FOR x=1 TO 360:INPUT#9,live(x):NEXT x:INPUT#9,oldate$:CLOSEIN
790 total=0:FOR x=1 TO 360
800 IF live(x)=1 THEN total=total+1
810 NEXT x
820 IF total<45 THEN binlen=&200*total ELSE binlen=&5A00
830 track=0:sector=65:start=&3474:logsec=0
840 CLS#2:LOCATE #2,2,4:PRINT#2,"Reading tape....."
850 over=0:FOR x=logsec+1 TO 360
860 IF live(x)=1 THEN over=over+1
870 NEXT
880 IF over<45 THEN binlen=over*512
890 size=binlen/2048
900 block=FIX(size)
910 IF block<>size THEN block=block+1
920 LOCATE #2,2,6:PRINT#2,"Loading ";block;" blocks "
930 LOAD "!",&3474:binlen=0
940 dsn$="":FOR title=&B807 TO &B816
950 IF PEEK(title)>31 THEN dsn$=dsn$+CHR$(PEEK(title))
960 NEXT title
970 CLS#2
980 LOCATE #2,2,4:PRINT#2,"Restoring : ";dsn$
990 LOCATE #2,2,6:PRINT#2,"Archived : ";oldate$
1000 LOCATE #2,2,8:PRINT#2,"Writing disc........"
1010 FOR count=0 TO 44
1020 logsec=logsec+1:IF logsec=361 THEN 1140
1030 IF live(logsec)=0 THEN GOSUB 1240:IF logsec=361 THEN 1140 ELSE 1030
1040 LOCATE #2,2,10:PRINT#2,"Track ";track;" Sector ";sector
1050 offset=(512*count)+start
1060 GOSUB 1160:REM **Get Lobyte Hi-Byte For Pokes**
1070 POKE &9083,lobyte
1080 POKE &9084,hibyte
1090 CALL &9082
1100 POKE &90E5,track:POKE &90E4,0:POKE &90E6,sector:POKE &90E3,0
1110 CALL &90C3:REM **Write Sector To Disc I/O Buffer**
1120 sector=sector+1:IF sector=74 THEN sector=65:track=track+1
1130 NEXT count
1140 IF track<>40 THEN 840
1150 RETURN
1160 REM **LO/HI**
1170 a$=HEX$(offset)
1180 lo$=MID$(a$,3,2)
1190 hi$=MID$(a$,1,2)
1200 hibyte=VAL("&"+hi$)
1210 lobyte=VAL("&"+lo$)
1220 binlen=binlen+&200:REM **Increment save length of cass I/O buffer**
1230 RETURN
```

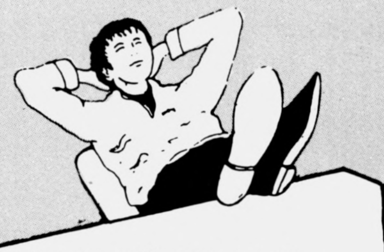
```
1240 REM **Live Sector Flag Found**
1250 logsec=logsec+1
1260 sector=sector+1:IF sector=74 THE
N sector=65:track=track+1
1270 RETURN
1280 REM **CHECK UNUSED SECTORS**
1290 track=0:sector=65
1300 LOCATE #2,2,4:PRINT#2,"Checking
For Unused Sectors "
1310 FOR logsec=1 TO 360
1320 LOCATE #2,2,6:PRINT#2, "Track ";
track;" Sector ";sector
1330 POKE &90E5,track:POKE &90E4,0:PO
KE &90E6,sector:POKE &90E3,0
1340 CALL &90B1:REM **read sector**
1350 POKE &9100,0
1360 CALL &90EA:REM **search routine*
*
1370 live(logsec)=PEEK(&9100)
1380 IF live(logsec)=1 THEN LOCATE #2
,2,8:PRINT#2,"Live sector found" ELSE
 LOCATE #2,2,8:PRINT#2,"Dead sector f
ound"
1390 sector=sector+1:IF sector=74 THE
N sector=65:track=track+1
1400 NEXT logsec
1410 CLS#2:RETURN
1420 REM **Wipe Restore Disc**
1430 FOR x=&8E75 TO &9074:POKE x,&E5:
NEXT
1440 track=0:sector=65
1450 POKE &90E5,track:POKE &90E4,0:PO
KE &90E6,sector:POKE &90E3,0
1460 CALL &90C3:REM **Write Sector**
1470 sector=sector+1:IF sector=74 THE
N sector=65:track=track+1
1480 IF track<>40 THEN GOTO 1450
1490 RETURN
```

Program II

```
Pass... 2           ORG &9076
9076:               .down
9076:21 74 90       LD HL,&9074
9079:11 00 00       LD DE,&0000
907C:01 00 02       LD BC,&0200
907F:ED B8          LDDR
9081:C9             RET
9082:               .up
9082:21 00 00       LD HL,&0000
9085:11 75 8E       LD DE,&8E75
9088:01 00 02       LD BC,&0200
908B:ED B0          LDIR
908D:C9             RET
908E:               .init
908E:DD 21 DD 90    LD IX,&90DD
9092:21 DB 90       LD HL,&90DB
9095:CD D4 BC       CALL &BCD4
9098:DD 75 00       LD (IX+00),L
909B:DD 74 01       LD (IX+01),H
909E:DD 71 02       LD (IX+02),C
90A1:21 DC 90       LD HL,&90DC
90A4:CD D4 BC       CALL &BCD4
90A7:DD 75 03       LD (IX+03),L
90AA:DD 74 04       LD (IX+04),H
90AD:DD 71 05       LD (IX+05),C
90B0:C9             RET
90B1:               .read
90B1:21 75 8E       LD HL,&8E75
90B4:ED 5B E4 90    LD DE,(&90E4)
90B8:3A E6 90       LD A,(&90E6)
90BB:4F             LD C,A
90BC:DF             RST 18
90BD:00             NOP
90BE:90             SUB B
90BF:D2 D5 90       JP NC,&90D5
90C2:C9             RET
90C3:               .write
90C3:21 75 8E       LD HL,&8E75
90C6:ED 5B E4 90    LD DE,(&90E4)
90CA:3A E6 90       LD A,(&90E6)
90CD:4F             LD C,A
90CE:DF             RST 18
90CF:E0             RET PO
90D0:90             SUB B
90D1:D2 D5 90       JP NC,&90D5
90D4:C9             RET
90D5:3E FF          LD A,&FF
90D7:32 E3 90       LD (&90E3),A
90DA:C9             RET
90DB:00             NOP
90DC:00             NOP
90DD:00             NOP
90DE:00             NOP
90DF:00             NOP
90E0:00             NOP
90E1:00             NOP
90E2:00             NOP
90E3:00             NOP
90E4:00             NOP
90E5:00             NOP
90E6:00             NOP
90E7:00             NOP
90E8:00             NOP
90E9:00             NOP
90EA:               .check
90EA:11 FF 01       LD DE,&01FF
90ED:21 74 8E       LD HL,&8E74
90F0:               .loop
90F0:23             INC HL
90F1:1B             DEC DE
90F2:7B             LD A,E
90F3:B2             OR  D
90F4:C8             RET Z
90F5:7E             LD A,(HL)
90F6:FE E5          CP  &E5
90F8:28 F6          JR Z,loop
90FA:21 00 91       LD HL,&9100
90FD:36 01          LD (HL),&01
90FF:C9             RET
9100:               END
```

Program III

```
10 REM ** Basic M/code Poker **
20 MODE 2
30 checksum=0
40 FOR address=&9076 TO &90FF
50 READ byte$
60 checksum=checksum+VAL("&"+byte$)
70 POKE address,VAL("&"+byte$)
80 NEXT
90 IF checksum<>15612 THEN CLS:PRINT
"DATA Error - Please debug":END
100 CLS:LOCATE 2,5:PRINT"Saving MAINC
ODE for use by Program I"
110 |DISC.OUT
120 SAVE "maincode",b,&9076,&8B
130 DATA 21,74,90,11,00,00,01,00
140 DATA 02,ED,B8,C9,21,00,00,11
150 DATA 75,8E,01,00,02,ED,B0,C9
160 DATA DD,21,DD,90,21,DB,90,CD
170 DATA D4,BC,DD,75,00,DD,74,01
180 DATA DD,71,02,21,DC,90,CD,D4
190 DATA BC,DD,75,03,DD,74,04,DD
200 DATA 71,05,C9,21,75,8E,ED,5B
210 DATA E4,90,3A,E6,90,4F,DF,00
220 DATA 90,D2,D5,90,C9,21,75,8E
230 DATA ED,5B,E4,90,3A,E6,90,4F
240 DATA DF,E0,90,D2,D5,90,C9,3E
250 DATA FF,32,E3,90,C9,00,00,00
260 DATA 00,00,00,00,00,00,00,00
270 DATA 00,00,00,00,11,FF,01,21
280 DATA 74,8E,23,1B,7B,B2,C8,7E
290 DATA FE,E5,28,F6,21,00,91,36
300 DATA 01,C9
```



Help Haggy through Santa's Grotty

IT'S Christmas Eve, and Santa's out on the town doing what Santa's best at during the festive season. He's also delivering presents to the kiddies.

He left home just as it went dark, but in his rush to get the job started left quite a few Christmas stars lying around the place.

That's not unusual, but the trouble is that the monsters from the infamous Grotty, that incredible lonely dark, dank land of the unbeliever, have gone out on the prowl, broken into his house and stolen the stars to sell on the "white" market.

You play the part of Haggy, Santa's right hand witch and part-time guardian of the Grotty. Your job is to get the stars back before anybody finds out they're missing.

To retrieve the stars you must enter the Grotty in your jet-pack powered suit, avoiding all the Meanies. You have to collect them all before the mists of darkness descend, as all caverns entered thereafter will be pitch black, and you won't be able to see your hand in front of your ugly face.

Good luck Haggy, you're going to need it.

VARIABLES

x,y	Haggy's coordinates.
x1,y1 / **x2,y2** / **x3,y3**	Monsters' coordinates.
gra1 / **gra2** / **gra3**	Monsters' shape number.
ll1,hl1 / **ll2,hl2** / **ll3,hl3**	High and low limit of monsters' movement.
dir1,dir2 / **dir3,dir4**	Monsters' direction: 1 = increase x, 2 = decrease x, 3 = increase y, 4 = decrease y.
star	Number of stars collected.
score	End score based on time and number of stars.
hs	High scores.
na$	High score names.
screen	Current screen number.
ti	Time left before darkness descends.

The Keys

Z	Left
X	Right
Space	Thrust

The sensational search for Santa's stars

By ARAMELLO CHAPMAN

SUBROUTINES

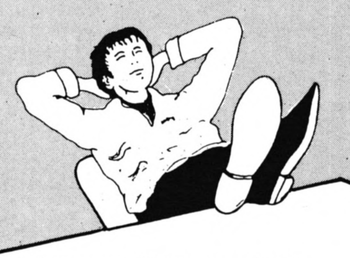
70	Main loop.
230	Moves Haggy.
700	Moves hazards.
990	Congratulations.
1040	End of game.
1180	Sets up screen.
1420	Initialises variables.
1480	Sets up screen border.
1650	Instructions.
1790	Sets up UDGs.
2050	Initial set up.
2870	High score.

```
1 REM Santas Grotty
2 REM By A.Chapman
3 REM Graphics Routine
4 REM By R.A.Waddilove
5 REM(c)Computing with the Amstrad
6 ON ERROR GOTO 30000
10 GOSUB 1790:REM U.D.G's
20 GOSUB 2050:REM Initial Set Up
30 GOSUB 1650:REM Instructions
40 GOSUB 1480:REM Set Up Border & Title
50 GOSUB 1420:REM Variables
60 GOSUB 1180:REM Set Up Screen
70 REM**********Main Loop************
80 th=0
90 GOSUB 230
100 IF exit=1 THEN exit=0:GOTO 60
110 GOSUB 940
120 GOSUB 700
130 GOSUB 940
140 IF star=8 THEN POKE 34857,3:POKE 34868,3:IF screen=5 THEN LOCATE 11,2:PRINT" ":LOCATE 11,3:PRINT" "
150 IF star=19 THEN POKE 36297,3
160 IF star=28 THEN 990
170 IF lives<1 THEN 1040
180 ti=ti-1:IF ti<0 THEN ti=0
190 PAPER#1,14:PEN#1,0
200 IF ti>-1 THEN LOCATE #1,14,17:PRINT#1,ti
210 IF st=1 THEN LOCATE #1,15,14:PAPER#1,14:PEN#1,0:PRINT#1,star:st=0
220 GOTO 70
230 REM*********Move Haggy***********
240 IF INKEY(71)<>0 AND INKEY(63)<>0 AND INKEY(47)<>0 THEN GOTO 280
250 IF INKEY(71)=0 THEN GOSUB 400:GOTO 280
260 IF INKEY(63)=0 THEN GOSUB 500:GOTO 280
270 IF INKEY(47)=0 THEN GOSUB 600
280 IF j=19 THEN 330
290 IF y>16 THEN 310
300 GOSUB 690:IF PEEK(s+(po+22))=4 THEN star=star+1:POKE(s+(po+22)),3:st=1
310 IF PEEK(s+(po+22))=3 THEN j=j+1:po=po+11
320 IF j<20 THEN 360
330 IF loc(screen,2)=0 THEN 360
340 f=screen:screen=loc(screen,2):GOSUB 690:IF PEEK(s+(x-1))<>3 OR PEEK(s+11+(x-1))<>3 THEN screen=f:GOTO 360
350 po=i-1:j=2:exit=1
360 IF exit=1 THEN CALL &A000,x,y,x,y,3:CALL &A000,x,y+1,x,y+1,3:GOTO 380
370 IF th=1 THEN CALL &A000,x,y,i,j,pic:CALL &A000,x,y+1,i,j+1,pic+1 ELSE IF th=0 THEN CALL &A000,x,y+1,i,j+1,pic+1:CALL &A000,x,y,i,j,pic
380 x=i:y=j
390 RETURN
400 pic=6:IF i=2 THEN 450
410 SOUND 132,0,15,3,1,4
420 GOSUB 690:IF PEEK(s+po-1)=4 THEN star=star+1:POKE (s+po-1),3:st=1 ELSE IF PEEK(s+po+10)=4 THEN star=star+1:POKE (s+po+10),3:st=1
430 IF PEEK(s+po-1)<>3 OR PEEK(s+(po+10))<>3 THEN RETURN
440 i=i-1:po=po-1:RETURN
450 IF loc(screen,4)=0 THEN RETURN
460 f=screen:screen=loc(screen,4)
470 GOSUB 690:IF PEEK(s+(po+10))<>3 OR PEEK(s+(po+21))<>3 THEN screen=f:RETURN
480 po=po-1
490 i=12:j=j-1:exit=1:RETURN
500 pic=4:IF i=12 THEN 550
510 SOUND 132,0,15,3,1,4
520 GOSUB 690:IF PEEK(s+po+1)=4 THEN star=star+1:POKE (s+po+1),3:st=1 ELSE IF PEEK(s+(po+11)+1)=4 THEN star=star+1:POKE (s+(po+11)+1),3:st=1
530 IF PEEK(s+po+1)<>3 OR PEEK(s+(po+12))<>3 THEN RETURN
540 i=i+1:po=po+1:RETURN
550 IF loc(screen,3)=0 THEN RETURN
560 f=screen:screen=loc(screen,3)
570 GOSUB 690:IF PEEK(s+(po-10))<>3 OR PEEK(s+(po+1))<>3 THEN screen=f:RETURN
580 po=po-10
590 i=2:exit=1:RETURN
600 SOUND 130,0,15,10,3,0,1:IF j=2 THEN 640
610 GOSUB 690:IF PEEK(s+(po-11))=4 THEN star=star+1:POKE(s+(po-11)),3:st=1
620 IF PEEK(s+(po-11))<>3 THEN RETURN
630 j=j-2:po=po-22:th=1:RETURN
640 IF loc(screen,1)=0 THEN RETURN
650 f=screen:screen=loc(screen,1)
660 GOSUB 690:IF PEEK(s+(198+(x-1)))<>3 OR PEEK(s+(187+(x-1)))<>3 THEN screen=f:RETURN
670 po=176+(i-1)
680 j=18:exit=1:RETURN
690 s=33999+((screen-1)*209):RETURN
700 REM********Move hazards**********
710 LET x4=x1:i1=x1:y4=y1:j1=y1:hl=hl1:dir=dir1:ll=111
720 ON dir1 GOSUB 860,880,900,920
730 CALL &A000,x1,y1,i1,j1,gra1
740 x1=i1:y1=j1:dir1=dir
750 IF x2=0 THEN RETURN
760 LET x4=x2:i1=x2:y4=y2:j1=y2:hl=hl2:dir=dir2:ll=112
770 ON dir2 GOSUB 860,880,900,920
780 CALL &A000,x2,y2,i1,j1,gra2
790 x2=i1:y2=j1:dir2=dir
800 IF x3=0 THEN RETURN
810 LET x4=x3:i1=x3:y4=y3:j1=y3:hl=hl3:dir=dir3:ll=113
820 ON dir3 GOSUB 860,880,900,920
830 CALL &A000,x3,y3,i1,j1,gra3
840 x3=i1:y3=j1:dir3=dir
850 RETURN
860 i1=x4+1:IF i1=hl THEN dir=2:IF screen=2 THEN j1=j1+1 ELSE IF screen=11 OR screen=13 THEN i1=ll:dir=1
870 RETURN
880 i1=x4-1:IF i1=ll THEN dir=1:IF screen=2 THEN j1=j1-1
890 RETURN
900 j1=y4+1:IF j1=hl THEN dir=4
910 RETURN
920 j1=y4-1:IF j1=ll THEN dir=3
930 RETURN
940 IF (x=x1 AND (y=y1 OR y+1=y1)) OR (x=x2 AND (y=y2 OR y+1=y2)) OR (x=x3 AND (y=y3 OR y+1=y3)) THEN GOSUB 960
950 RETURN
960 lives=lives-1:IF lives<1 THEN RETURN
970 OUT &BC00,8:OUT &BD00,1:SOUND 130,0,50,15,1,1,3:LOCATE #1,13+lives,20:PAPER #1,14:PRINT#1," ":LOCATE #1,13+lives,21:PRINT#1," ":OUT &BC00,8:OUT &BD00,0
980 RETURN
990 REM*******Congratulations********
1000 RESTORE 1030:FOR f=1 TO 24:READ n:SOUND 4,n,20,15,1:NEXT
1010 LET score=(star*100)+(ti*2)
1020 PRINT STRING$(32,11):LOCATE 2,7:PRINT"WELL DONE":FOR f=1 TO 2000:NEXT f:GOTO 2070
1030 DATA 60,53,47,45,60,0,45,47,45,40,53,0,53,47,45,36,40,40,45,45,47,53,47,60
1040 REM*********End of game**********
1050 LOCATE i-1,j-1:PEN 3:PRINT CHR$(236):LOCATE i-1,j:PRINT CHR$(236)
1060 FOR f=1 TO 200:NEXT
1070 SOUND 132,0,15,15,0,0,1
1080 LOCATE i-1,j-1:PRINT CHR$(235):LOCATE i-1,j:PRINT CHR$(235)
1090 SOUND 132,0,15,11,0,0,1
1100 FOR f=1 TO 200:NEXT
1110 LOCATE i-1,j-1:PRINT CHR$(234):LOCATE i-1,j:PRINT CHR$(234):SOUND 132,0,15,8,0,0,1
1120 FOR f=1 TO 200:NEXT
1130 LOCATE i-1,j-1:PRINT" ":LOCATE i-1,j:PRINT" ":FOR f=1 TO 200:NEXT
1140 FOR g=1 TO 2:RESTORE 1160:FOR f=1 TO 11:READ d,n:SOUND 1,n,d,7:SOUND
```

```
4,n+2,d,7:SOUND 5,0,3,0:NEXT f,g
1150 LET score=(star*100)
1160 DATA 50,1016,37,1016,12,1016,50,
1016,25,850,25,899,25,899,25,1016,25,
1016,25,1136,100,1016
1170 PRINT STRING$(40,11):LOCATE 2,7:
PRINT"GAME OVER":FOR f=1 TO 2000:NEXT
 f:GOTO 2870
1180 REM********Set Up Screen********
*
1190 PAPER 0:CLS
1200 IF ti=0 THEN GOTO 1240
1210 IF screen>6 AND screen<12 THEN R
ESTORE 2340 ELSE RESTORE 2190
1220 FOR f=0 TO 159:READ n$:POKE &A10
0+f,VAL("&"+n$):NEXT f
1230 LET s=33999+((screen-1)*209):FOR
 f=2 TO 20:FOR g=2 TO 12:s=s+1:CALL &
A000,g,f,g,f,PEEK(s):NEXT g,f
1240 LOCATE#1,2,22:PAPER#1,10:PEN#1,4
:PRINT#1,STRING$(11," "):LOCATE #1,2,
22:PRINT#1,sn$(screen):PEN 1
1250 RESTORE 2240
1260 FOR f=0 TO 31:READ n$:POKE &A180
+f,VAL("&"+n$):NEXT f
1270 IF screen>10 THEN GOSUB 1310 ELS
E GOSUB 1300
1280 GOSUB 1340
1290 RETURN
1300 addr=&A100:RESTORE 2280:GOSUB 13
20::addr=&A120:RESTORE 2290:GOSUB 132
0:addr=&A140:RESTORE 2300:GOSUB 1320:
RETURN
1310 addr=&A100:RESTORE 2310:GOSUB 13
20:addr=&A120:RESTORE 2320:GOSUB 1320
:addr=&A140:RESTORE 2330:GOSUB 1320:R
ETURN
1320 FOR f=0 TO 32:READ n$:POKE addr+
f,VAL("&"+n$):NEXT f
1330 RETURN
1340 no=((screen-1)*18)-5
1350 y1=ene(no+(6)):x1=ene(no+(6)+1)
1360 gra1=ene(no+(6)+2):dir1=ene(no+(
6)+3):l11=ene(no+(6)+4):h11=ene(no+(6
)+5)
1370 y2=ene(no+(12)):x2=ene(no+(12)+1
)
1380 gra2=ene(no+(12)+2):dir2=ene(no+
(12)+3):l12=ene(no+(12)+4):h12=ene(no
+(12)+5)
1390 y3=ene(no+(18)):x3=ene(no+(18)+1
)
1400 gra3=ene(no+(18)+2):dir3=ene(no+
(18)+3):l13=ene(no+(18)+4):h13=ene(no
+(18)+5)
1410 RETURN
1420 REM********Variables************
1430 screen=1:lives=7
1440 x=6:y=15:i=6:j=15
1450 star=0:pic=4:score=0
1460 ti=2000:po=140
1470 RETURN
1480 REM****Set Up Screen border*****
1490 MODE 0
1500 LOCATE 1,1:PAPER 1:PEN 3:PRINT S
TRING$(20,CHR$(240))
1510 FOR f=2 TO 22:LOCATE 1,f:PRINT C
HR$(240):LOCATE 13,f:PRINT CHR$(240):
LOCATE 20,f:PRINT CHR$(240):NEXT
1520 LOCATE 1,23:PRINT STRING$(20,CHR
$(240)):LOCATE 1,21:PRINT STRING$(12,
CHR$(240)):LOCATE 14,22:PRINT STRING$
(6,CHR$(240))
1530 PAPER 15:FOR f=2 TO 11:LOCATE 14
,f:PRINT STRING$(6,CHR$(238)):NEXT
1540 LOCATE 14,12:PAPER 1:PEN 3:PRINT
 STRING$(6,CHR$(240)):PAPER 2
1550 PEN 3:LOCATE 14,2:PRINT CHR$(241
):LOCATE 14,3:PRINT CHR$(242)CHR$(243
):LOCATE 14,4:PRINT " "CHR$(244)CHR$(
245)
1560 LOCATE 14,5:PRINT CHR$(249)" "CH
R$(246)CHR$(247):LOCATE 14,6:PRINT CH
R$(250)CHR$(251)" "CHR$(248)CHR$(243)
:LOCATE 18,7:PRINT CHR$(244)CHR$(241)
:LOCATE 19,8:PRINT CHR$(242)
1570 LOCATE 15,7:PRINT CHR$(252)CHR$(
253)" ":LOCATE 16,8:PRINT CHR$(254)CH
R$(247)" ":LOCATE 17,9:PRINT CHR$(248
)CHR$(247)" ":LOCATE 18,10:PRINT CHR$
(248)CHR$(255):LOCATE 19,11:PRINT CHR
$(239)
1580 FOR f=13 TO 21:PAPER 14:LOCATE 1
4,f:PRINT STRING$(6," "):NEXT:LOCATE
2,22:PAPER 10:PRINT STRING$(11," ")
1590 LOCATE 14,13:PAPER 1:PEN 5:PRINT
"STARS:"
1600 LOCATE 14,19:PRINT"SUITS:"
1610 LOCATE 14,16:PRINT" TIME "
1620 FOR f=1 TO 6:CALL &A000,13+f,20,
13+f,20,4:CALL &A000,13+f,21,13+f,21,
5:NEXT
1630 WINDOW #1,2,12,2,20:WINDOW SWAP
0,1
1640 RETURN
1650 REM**********Instructions*******
1660 MODE 1:INK 14,15:INK 15,24:INK 1
,18:INK 12,2,26
1670 LOCATE 1,1:PEN 1:PAPER 3:PRINT S
TRING$(40,CHR$(240)):FOR f=2 TO 5:LOC
ATE 1,f:PRINT CHR$(240):LOCATE 40,f:P
RINT CHR$(240):NEXT f:LOCATE 1,6:PRIN
T STRING$(40,CHR$(240))
1680 LOCATE 15,3:PEN 3:PAPER 0:PRINT"
SANTAS GROTTY":LOCATE 15,4:PEN 2:PRIN
T STRING$(13,CHR$(131))
1690 LOCATE 1,8
1700 PEN 1:PRINT"Guide Haggy around t
he caverns of SantasGrotty and collec
t all the stars which  Santa has left
 behind."
1710 PEN 3:PRINT"Beware of all who mo
ve as they are very dangerous and wil
l destroy one of your  suits every ti
me you hit them.
1720 PEN 2:PRINT"When the time counte
r reaches zero the  mists of darkness
 will descend leaving  new caverns in
visible."
1730 LOCATE 17,18:PEN 3:PRINT"THE KEY
S":LOCATE 17,19:PEN 2:PRINT STRING$(8
,CHR$(131))
1740 LOCATE 1,21:PEN 1:PRINT"'Z'-LEFT
   ";:PEN 2:PRINT"'<SPACE>'-THRUST
";:PEN 3:PRINT"'X'-RIGHT"
1750 LOCATE 1,22:PAPER 3:PEN 1:PRINT
STRING$(40,CHR$(240))
1760 LOCATE 1,23:PRINT CHR$(240):LOCA
TE 40,23:PRINT CHR$(240):LOCATE 1,24:
PRINT STRING$(40,CHR$(240))
1770 LOCATE 9,23:PEN 3:PAPER 0:PRINT"
Press <SPACE> to continue.."
1780 GOSUB 2650:GOTO 2870
1790 REM***********U.D.Gs************
1795 ON ERROR GOTO 1805
1800 SYMBOL AFTER 230
1805 ON ERROR GOTO 30000
1810 SYMBOL 240,66,165,90,60,60,90,16
5,66
1820 SYMBOL 241,126,255,255,231,226,2
24,224,254
1830 SYMBOL 242,255,127,7,71,231,255,
255,126
1840 SYMBOL 243,24,60,126,255,231,231
,231,231
1850 SYMBOL 244,255,231,231,231,231,2
31,231,102
1860 SYMBOL 245,0,24,60,126,231,231,2
31,231
1870 SYMBOL 246,231,231,231,231,231,2
31,231,66
1880 SYMBOL 247,0,56,56,56,254,254,56
,56
1890 SYMBOL 248,56,56,56,56,57,63,62,
60
1900 SYMBOL 249,126,255,255,230,224,2
24,224,224
1910 SYMBOL 250,224,239,239,230,230,2
54,254,124
1920 SYMBOL 251,62,127,231,231,231,25
5,254,240
1930 SYMBOL 252,240,240,236,236,230,2
30,227,227
1940 SYMBOL 253,24,60,126,231,231,231
,231,231
1950 SYMBOL 254,231,231,231,231,231,1
26,60,24
```

```
1960 SYMBOL 255,195,199,230,238,124,6
0,28,28
1970 SYMBOL 239,28,28,56,48,112,96,22
4,192
1980 SYMBOL 238,254,254,254,0,239,239
,239,0
1990 SYMBOL 237,146,84,56,254,56,84,1
46,0
2000 SYMBOL 236,144,70,16,132,66,20,8
0,137
2010 SYMBOL 235,0,68,0,136,1,16,132,1
2020 SYMBOL 234,8,0,34,0,0,8,65,0
2030 SYMBOL 233,198,165,198,165,6,40,
40,16
2040 RETURN
2050 REM*******Initial Set Up********
2060 INK 0,0:BORDER 0:PAPER 0:CLS:LOC
ATE 15,10:PEN 2:PRINT"PLEASE WAIT!!"
2070 RESTORE 2140
2080 MEMORY &84CF:check=0
2090 ENV 3,3,2,2,3,-2,2:ENT 4,5,-10,2
0,1,0,5:ENT 1,30,10,1:ENV 1,10,-1,2
2100 FOR f=0 TO 98
2110 READ n$:POKE &A000+f,VAL("&"+n$)
2120 check=check+VAL("&"+n$)
2130 NEXT
2140 DATA dd,7e,00,87,87,87,87,87,32,
47,a0,26,c0,dd,7e,04,3d,87,87,6f,11,5
0,00,dd,46,02,05,19,10,fd,e5,26,c0,dd
2150 DATA 7e,08,3d,87,87,6f,11,50,00,
dd,46,06,05,19,10,fd,11,fd,07,06,08,3
6,00,23,36,00,23,36,00,23,36,00
2160 DATA 19,10,f2,e1,11,00,a1,06,08,
1a,77,13,23,1a,77,13,23,1a,77,13,23,1
a,77,13,78,01,fd,07,09,47,10,e9,c9
2170 IF check<>7948 THEN LOCATE 10,10
:PRINT"ERROR IN DATA":PRINT CHR$(7):E
ND
2180 RESTORE 2240:FOR f=0 TO 128:READ
 n$:POKE &A180+f,VAL("&"+n$):NEXT
2190 DATA cd,cd,cd,cd,6a,6a,bf,95,bf,
95,95,bf,cd,bf,6a,6a,95,cd,95,6a,cd,8
0,cd,80,40,00,40,80,40,00,00,80
2200 DATA cf,cf,cf,cf,ff,ca,cd,df,95,
df,df,df,cd,ca,cd,df,bf,9f,7f,ca,cf,c
f,cf,cf,df,cd,9f,ea,df,7f,ca,d5
2210 DATA 6e,cc,cc,9d,3d,cc,cc,3e,3c,
6e,9d,3c,3c,6e,9d,3c,3c,6e,9d,3c,3c,6
e,9d,3c,3d,cc,cc,3e,6e,cc,cc,9d
2220 DATA 00,00,00,00,00,00,00,00,00,
00,00,00,00,00,00,00,00,00,00,00,00,0
0,00,00,00,00,00,00,00,00,00,00
2230 DATA 22,11,00,22,11,11,11,00,00,
33,22,00,33,33,33,22,00,33,22,00,11,1
1,11,00,22,11,00,22,00,00,00,00
2240 DATA 00,30,20,00,30,30,30,20,75,
25,88,20,71,64,cc,20,75,64,88,20,35,3
0,88,20,10,30,30,20,10,30,20,00
2250 DATA 10,71,20,00,3a,71,20,00,3f,
71,f3,e6,3f,30,20,00,3f,30,20,00,3a,3
0,30,20,00,00,10,20,00,00,40,c0
2260 DATA 00,10,30,00,10,30,30,30,10,
44,1a,ba,10,cc,98,b2,10,44,98,ba,10,4
4,30,3a,10,30,30,20,00,10,30,20
2270 DATA 00,10,b2,20,00,10,b2,35,d9,
f3,b2,3f,00,10,30,3f,00,10,30,3f,10,3
0,30,35,10,20,00,00,c0,80,00,00
2280 DATA 00,51,a2,00,00,f3,f3,00,51,
ab,57,a2,3f,3a,35,1f,2f,0f,0f,2f,05,8
f,4f,0a,00,8a,45,00,00,8a,45,00
2290 DATA 08,04,08,04,04,08,04,08,04,
aa,55,08,08,00,00,04,04,44,80,08,04,0
8,04,08,0c,04,08,0c,08,00,00,04
2300 DATA 20,00,00,10,10,00,00,20,30,
1a,25,30,25,0f,0f,1a,30,1a,25,30,10,3
0,30,20,00,3f,3f,00,44,ee,dd,88
2310 DATA 00,00,00,00,00,30,00,00,10,
30,20,00,30,71,64,cc,30,30,64,cc,10,3
0,20,00,00,30,00,00,00,30,20,00
2320 DATA 10,25,30,00,30,25,30,20,30,
25,30,20,30,25,0f,20,30,30,30,20,30,3
0,30,20,30,30,30,20,30,30,30,25
2330 DATA 40,c0,c0,80,80,00,00,40,cc,
88,44,cc,d5,10,00,ea,80,10,00,40,80,1
0,30,40,95,00,00,40,95,3f,6a,c0
2340 DATA 0d,00,00,0e,0e,0a,05,0d,05,
0d,0e,0a,00,0e,0d,00,00,0e,0d,00,05,0
d,0e,0a,0e,0a,05,0d,0d,00,00,0e
2350 DATA 3f,3f,b7,3f,7b,7b,fb,3f,f3,
f3,f3,fb,51,5b,f3,a2,51,f3,fb,00,00,f
b,a2,00,00,51,00,00,00,00,00,00
2360 DATA 00,40,00,00,00,40,00,00,00,
c0,80,00,00,c0,80,00,40,6a,6a,00,40,9
0,c0,00,d5,d5,d5,40,cc,cc,cc,cc
2370 DATA 00,00,00,00,00,00,00,00,00,
00,00,00,00,00,00,00,00,00,00,00,00,0
0,00,00,00,00,00,00,00,00,00,00
2380 DATA 22,11,00,22,11,11,11,00,00,
33,22,00,33,33,33,22,00,33,22,00,11,1
1,11,00,22,11,00,22,00,00,00,00
2390 RESTORE 2410
2400 no=33996:FOR f=1 TO 13:FOR g=0 T
O 52:no=no+4:READ n:POKE no,n\64:POKE
 no+1,(n AND 63)\16:POKE no+2,(n AND
15)\4:POKE no+3,n AND 3:NEXT g:no=no-
3:NEXT f
2410 DATA 170,170,171,127,255,252,255
,255,254,255,240,212,3,255,95,255,252
,63,255,255,42,63,255,63,255,255,255,
255,255,240,42,175,255,255,252,3,255,
255,255,255,255,255,85,255,253,255,25
5,247,254,160,0,0,0
2420 DATA 42,128,3,170,255,255,255,25
5,255,255,255,149,127,2,255,255,251,2
55,255,229,92,15,191,255,254,255,255,
249,87,255,239,255,255,191,240,2,255,
255,251,255,255,239,3,255,191,255,254
,255,255,245,95,85,86
2430 DATA 170,250,170,255,255,235,255
,255,160,0,14,191,255,250,3,255,235,2
52,0,175,255,254,191,255,250,0,0,235,
255,255,175,255,254,191,15,250,243,20
7,235,31,79,175,125,254,191,247,250,2
55,223,234,170,170,168
2440 DATA 128,0,2,255,255,248,255,243
,239,51,255,188,243,254,213,85,91,255
,255,239,255,255,181,85,94,223,255,12
3,127,253,236,255,243,188,255,62,252,
243,251,255,255,237,255,247,191,125,2
54,255,255,250,170,170,186
2450 DATA 170,170,171,255,255,239,255
,255,144,16,18,127,127,121,253,253,23
1,119,247,157,207,206,119,247,249,223
,223,224,0,7,191,255,222,0,0,59,255,2
55,239,255,255,191,255,254,0,0,251,25
5,255,234,170,170,186
2460 DATA 170,170,171,253,85,95,240,0
,47,255,255,179,255,254,252,0,27,255,
255,99,255,253,191,64,2,253,255,251,5
5,255,239,16,16,191,63,62,63,255,251,
255,127,111,192,0,191,255,246,255,255
,216,0,48,53
2470 DATA 85,87,87,255,255,207,255,25
5,63,255,252,254,171,243,240,15,207,2
07,63,63,60,252,255,255,243,207,255,2
07,254,255,48,0,252,255,255,243,243,2
55,207,255,255,63,63,252,255,255,242,
170,170,128,0,0,48
2480 DATA 0,0,3,255,255,255,255,255,2
34,175,254,255,255,247,255,255,245,95
,255,220,255,254,252,255,247,252,255,
252,252,255,240,252,250,112,252,195,2
55,255,15,255,188,63,236,0,255,48,0,2
36,192,0,0,0,48
2490 DATA 0,0,0,255,255,243,255,255,2
07,252,0,63,243,252,255,207,243,253,8
5,79,255,255,63,255,252,255,170,163,2
55,255,207,255,255,63,213,84,255,255,
243,235,255,207,3,59,48,0,12,0,0,0,0,
0,63
2500 DATA 255,255,51,255,252,206,174,
171,48,48,12,192,240,49,3,192,192,7,3
,0,60,12,85,245,115,255,252,207,255,3
,0,0,252,255,255,243,255,254,202,162,
135,63,95,252,255,255,243,191,238,192
,15,0,48
2510 DATA 255,63,255,252,255,252,3,0,
51,255,255,207,255,255,255,255,255,24
5,87,255,255,255,87,255,255,255,255,2
55,255,255,255,255,191,255,251,191,25
5,188,255,251,254,245,79,243,255,255,
243,255,255,197,85,85,127
2520 DATA 255,255,253,95,253,95,192,1
5,255,247,255,252,0,255,251,123,255,2
```

```
53,255,255,255,255,252,0,255,255,255,
255,253,255,213,127,255,255,247,127,2
55,93,255,253,117,255,196,215,247,207
,79,95,253,212,48,195,63
2530 DATA 255,255,255,235,235,255,255
,255,255,240,0,127,255,254,127,255,25
5,124,63,253,255,255,189,255,254,247,
195,251,7,255,239,223,240,191,223,254
,223,127,250,173,255,239,243,255,191,
255,254,223,255,208,0,0,0
2540 DIM ene(234):RESTORE 2550:FOR f=
1 TO 234:READ ene(f):NEXT f
2550 DATA 11,4,0,1,2,11,3,6,0,1,4,12,
14,4,1,3,14,19,4,8,2,1,4,12,8,8,1,2,6
,12,10,8,2,1,4,12,6,5,1,1,3,10,10,5,1
,2,3,10,0,0,0,0,0,0,10,3,0,3,5,19,10,
12,0,4,8,19,16,5,1,1,4,9,10,11,2,4,6,
19,0,0,0,0,0,0,0,0,0,0,0,0,5,5,1,3,3,
12,19,3,0,1,3,10,0,0,0,0,0,0
2560 DATA 7,3,0,3,3,12,10,10,0,1,2,11
,0,0,0,0,0,0,4,5,2,1,2,12,6,9,2,2,2,1
0,10,10,2,3,3,15,3,5,0,1,3,10,9,5,0,2
,3,12,12,5,0,1,3,10,2,5,2,2,2,9,11,4,
1,1,2,9,10,11,2,3,2,19,18,5,0,1,5,11,
19,5,1,1,5,11,7,7,2,1,2,12,9,7,2,1,3,
11,11,7,2,2,2,12,0,0,0,0,0,0
2570 DATA 18,5,0,1,5,12,19,5,1,1,5,12
,4,2,2,1,2,12
2580 RESTORE 2590:DIM sn$(15),loc(15,
4),na$(8),hs(8),x1(3),x2(3),dir(3),gr
a(3),l1(3),h1(3),j1(3),i1(3):FOR f=1
TO 15:READ sn$(f):NEXT
2590 DATA Haggies Pit,The Slobby,Bods
 Hobble,The Sewer,Confused???,Hello W
orld,Low down,The Pits,The Thrips,Bim
bleBom,The Eggloo,Mt.Chapman,Santa's
Hut
2600 RESTORE 2610:FOR f=1 TO 15:FOR g
=1 TO 4:READ loc(f,g):NEXT g,f
2610 DATA 0,0,5,2,0,3,1,4,2,0,0,0,0,0
,2,0,0,0,6,1,0,7,10,5,6,0,0,8,0,0,7,9
,0,0,8,0,11,3,0,6,12,10,0,13,0,11,0,0
,0,0,11,0
2620 FOR f=1 TO 8:na$(f)="BIG SCORES"
:hs(f)=3500-(400*f):NEXT
2630 te=0.6:o=1
2640 RETURN
2650 RESTORE 2800
2660 READ p,d:IF p=999 THEN RESTORE 2
800:GOTO 2660
2670 GOSUB 2770
2680 SOUND 1,pn,d*0.5,15,1
2690 p=100:GOSUB 2770
2700 SOUND 1,pn,5,15,1
2710 WHILE INKEY$<>"":WEND
2720 IF INKEY(47)<>0 THEN 2660
2730 RESTORE 2860:FOR f=1 TO 20:READ
g,f1:POKE 33999+((g-1)*209)+f1,4:NEXT
f
2740 POKE 34857,2:POKE 34860,2:POKE 3
6297,2
2750 LOCATE 1,1:PRINT STRING$(26,11)
2760 RETURN
2770 fr=440*(2^(0+((p-10)/12)))
2780 pn=ROUND(125000/fr)
2790 RETURN
2800 DATA 5,40,5,40,12,40,12,40,10,40
,8,40,7,40,5,40,3,40,5,40
2810 DATA 7,40,8,40,10,40,12,80,5,40,
5,40,12,40,12,40,10,40,8,40,7,40,5,40
,3,40,5,40,7,40,8,40,10,40
2820 DATA 12,80,12,40,13,40,10,40,12,
40,13,40,15,40,17,40,12,40,10,40
2830 DATA 8,40,5,40,7,40,8,40,10,80,8
,40,10,40,12,80,13,40,12,40,12,40,10,
40,8,40,7,40,5,80,8,20,7,20,5,40
2840 DATA 10,80,8,40,10,40,12,40,13,4
0,15,40,17,40,12,40,10,40,8,40,7,40,5
,80
2850 DATA 999,999
2860 DATA 1,59,1,110,2,68,2,90,3,148,
4,104,4,107,5,101,6,76,6,105,7,71,7,7
2,8,157,8,158,8,151,9,52,9,165,9,154,
10,166,11,178,11,140,12,33,12,23,12,6
0,12,62,13,9,13,67,13,68
2870 REM*********High Score*********
2880 MODE 1
2890 LOCATE 4,5:PEN 3:PRINT CHR$(150)
;STRING$(32,CHR$(154));CHR$(156)
2900 LOCATE 16,3:PEN 2:PRINT"HIGH SCO
RE":PEN 3
2910 FOR f=6 TO 15:LOCATE 4,f:PRINT C
HR$(149):LOCATE 37,f:PRINT CHR$(149):
NEXT
2920 LOCATE 4,16:PRINT CHR$(147);STRI
NG$(32,CHR$(154));CHR$(153)
2930 FOR f=1 TO 8
2940 IF score>hs(f) THEN GOSUB 3030:f
=10
2950 NEXT
2960 FOR f=1 TO 8:PEN 1:LOCATE 8,f+6:
PRINT na$(f):LOCATE 18,f+6:PEN 3:PRIN
T"..........";hs(f):NEXT
2970 LOCATE 1,17:PRINT STRING$(220,"
")
2980 IF INKEY$<>"" THEN GOTO 2980
2990 PEN 2:LOCATE 1,20:PRINT STRING$(
40,CHR$(154)):LOCATE 1,22:PRINT STRIN
G$(40,CHR$(154)):PEN 1
3000 LOCATE 1,23:PRINT STRING$(40," "
)
3010 LOCATE 11,21:PEN 3:PRINT"PRESS";
:PEN 1:PRINT"< SPACE >";:PEN 3:PRINT"
TO PLAY.":PEN 1
3020 GOSUB 2650:GOTO 40
3030 a$="ABCDEFGHIJKLMNOPQRSTUVWXYZ .
#%&()!{}?*+"+CHR$(233)
3040 c=19:LOCATE 1,20:PEN 1:PRINT a$
3050 LOCATE 3,17:PEN 1:PRINT"USE CURS
OR KEY'S LEFT,RIGHT AND COPY":LOCATE
4,18:PRINT"TO SELECT LETTERS.(MAXIMUM
 OF 10.)":PEN 2:LOCATE 1,19:PRINT STR
ING$(40,CHR$(154)):LOCATE 1,22:PRINT
STRING$(40,CHR$(154))
3060 LOCATE 12,23:PEN 3:PRINT"PRESS";
:PEN 1:PRINT"< 'X' >";:PEN 3:PRINT"TO
 EXIT.":PEN 1
3070 x$=""
3080 FOR z=1 TO 10
3090 LOCATE c,21:PEN 2:PRINT" "
3100 IF INKEY(1)=0 AND c<40 THEN c=c+
1
3110 IF INKEY(8)=0 AND c>1 THEN c=c-1
3120 IF INKEY(9)=0 AND c=40 THEN LOCA
TE 7,f+6:PRINT"           ":z=100:GOT
O 3170
3130 IF INKEY(63)=0 THEN z=11:GOTO 31
70
3140 IF INKEY(9)<>0 THEN LOCATE c,21:
PRINT"*":FOR a=1 TO 50:NEXT:GOTO 3090
3150 x$=x$+MID$(a$,c,1):LOCATE 7+z,f+
6:PEN 1:PRINT MID$(a$,c,1)
3160 FOR a=1 TO 200:NEXT
3170 NEXT
3180 IF z=101 THEN GOTO 3070
3190 hs(8)=score:na$(8)=x$
3200 f=0
3210 FOR z=1 TO 7
3220 IF hs(z)<hs(z+1) THEN t=hs(z+1):
hs(z+1)=hs(z):hs(z)=t:a$=na$(z+1):na$
(z+1)=na$(z):na$(z)=a$:f=1
3230 NEXT
3240 IF f=1 THEN GOTO 3200
3250 fr=FRE("")
3260 RETURN
30000 MODE 1
30001 INK 1,24:PAPER 0:PEN 1:PRINT "E
rror...";ERR;"at line";ERL
30002 END
```



PASCAL THEORY

PASCAL arose from investigations into possible developments resulting from the inclusion of data structuring facilities in an ALGOL-60 like language.

It was designed around 1970 mainly by Professor Niklaus Wirth working at the Institute for Informatics in Zurich, but also benefited by the inclusion of some of the ideas of C.A.R. Hoare who was also working on data structuring facilities in programming languages.

He published his language in 1971 and named it after the great seventeenth century French philosopher Blaise Pascal, who invented one of the earliest known calculators.

Two years later, in 1973, Hoare and Wirth attempted a formal definition of the language in response to user experience to shed light on areas of uncertainty. This led to a revision and extension of the original language.

As with all computer languages, Pascal was designed for a specific purpose. Niklaus Wirth's main objective was to produce a language better suited to teaching programming than any existing language at the time. He was successful in his aims and it soon became popular as a teaching language.

Very quickly user groups sprang up in several countries to exchange information and ideas on Pascal and the language was adopted by the University of California, San Diego in 1973/4 as their main teaching

... and practice: Hi-soft

By ROLAND WADDILOVE

THERE are two Amstrad versions of Hisoft Pascal, 4D and 4T. The first is a special version for disc owners which runs under CP/M, the second an ordinary version available on tape or disc that does not require CP/M.

The two implementations of the language are identical, the only differences are the editor, and storage of the source and object files. Perhaps I ought to make it clear that my own preference is for the CP/M version.

There are a total of 10 files on the CP/M Pascal disc. The two main utilities are HP80 the compiler, and ED80 a text editor.

The Pascal source text is written using ED80. This text editor is far superior to some word processors I've seen and could quite easily be used as such. The only missing functions are word wrap and justification, which in any case would be undesirable for writing programs.

The cursor can be moved throughout the text one character, word, line or screen at a time. Text can be entered in insert or overwrite mode and deleted one character, word or line at a time.

Markers can be placed around text to define a block. This can then be moved, copied, deleted, read from or written to disc. There's quite a powerful find and replace function which also allows the use of wildcards.

Where ever possible the functions are obtained by using the same keystrokes as within Wordstar. This means, for instance, that to move the cursor right you press Ctrl+D, left is Ctrl+S and quit is Ctrl+K.

Frankly the keys selected are appalling. Not to worry though, there's a file on the disc which enables you to alter almost every function of ED80 to your own personal taste. This is menu driven and very easy to operate.

The first thing most people will do is move those awful Wordstar cursor keys to their usual place on the Amstrad keyboard.

Having entered the Pascal source text and saved it to disc it can be compiled with HP80. This takes the source text from a .PAS file and places the object code in a .COM file.

Several options can be set affecting compilation. Listing can be enabled or disabled and sent to screen or printer. Error checking can be turned off or on and mathematical functions are reals, or just integers, selected.

A standard CP/M .COM file is produced and, as with all transient commands, it is executed by typing its name.

I'm a complete novice when it comes to CP/M, so being able to write CP/M utilities in Pascal is a great advantage. I can now type CLS to clear the screen, PEN 3, PAPER 4 and so on. I'm sure a CP/M expert would never do this, but it works for me!

There are two manuals for CP/M Pascal, one describing the editor and the other the compiler. Neither will teach you Pascal, but they do contain all the experienced programmer needs to know to use this particular implementation.

The Pascal itself is pretty standard

language. UCSD were responsible for implementing Pascal for a wide range of computers.

One of the main reasons for Pascal catching on so quickly is that it is concise – the rules of grammar can be written down on just four or five pages.

Pascal is fairly simple to learn although complete beginners may have trouble initially as the knowledge required to write your first program is greater than for Basic.

Pascal is a highly structured language with a rigid format that the programmer is required to adhere to. Everything is laid out so neatly and logically that it is difficult to go wrong.

It encourages a style of programming in which programs are built up step by step from small well defined procedures.

All programs start with the word 'program' followed by the name of the program. All the constants and variables used must be declared after the title, plus their type – for example, integer.

Any procedures used are defined following the variables and constants and the action part of the program commences with 'begin' and finishes with 'end'.

Pascal programs are very readable, being almost self documenting and needing very few comments. The program flow is easy to follow and the structure clear, making alterations, improvements and debugging very simple.

Lisp is quite interesting, Forth is fast and powerful, Basic just a Mickey Mouse toy for kids – but Pascal is a real programmer's language and a delight to use.

Pascal is a compiled language, not an interpreted one like Basic which means that Pascal programs run many times faster than their Basic equivalents.

There are two popular ways of implementing Pascal, each with its own advantages.

Either the text of the source program can be compiled to pure machine code – which makes it very fast but specific to that machine – or it can be compiled to P-Code which is then interpreted when run, not unlike Forth.

This is slower but more easily transferred to other machines.

Pascal

and virtually identical to ISO-Pascal on the BBC Micro and Electron. I borrowed a Pascal book from the editor of *Apple User* and tried a few examples – they all ran perfectly, and somewhat faster than the Apple II, I might add. So there are plenty of books and tutorial guides the novice can turn to even though none are specifically for the Amstrad.

Pascal 4T, tape or disc, differs in the way it operates, though the implementation of the language is identical.

The whole of the compiler, runtime routines and editor are loaded and are resident in memory at the same time. This leaves around 20k for both your source text and object code, which are both present at the same time. Compare this with around 30k for source and 30k for object code under CP/M.

The source text is entered in the same way as Basic with line numbers. Editing is with the cursor and Copy keys.

In addition to the normal Basic editor several other functions are available from a menu, including search and replace and a separate line editor. Text may be inserted, deleted, overwritten, deleted or abandoned, and can be saved to disc or tape.

The text can be compiled and the object code run or saved. The saved code can be run without the compiler being present, which means you can write your machine code routines in a high level language.

The advantages of these implementations are legion. Why use a Basic compiler when you can write in a high level language like Pascal? The Pascal compiler can cope with real numbers, arrays, SIN, COS, TAN, LOG and many more. Can any Basic compiler?

And if you want speed and haven't the time or the knowledge to write in machine code, then use Pascal.

As with the CP/M version, the manual is simply a reference guide for the Pascal programmer and not a tutorial, though there are several examples to type in.

A number of additional functions have been included in both versions of Pascal. PEEK and POKE are obvious, INLINE places Z80 machine code in the memory at the current compiler address, USER calls a machine code routine and NEW reserves space for a variable.

A turtle graphics package written in Pascal has been included with Pascal 4T. This, combined with Pascal's structure and wide range of commands, produces a powerful language for drawing quite complex patterns.

Being a compiled language, Pascal tends to be faster than Basic. In a test which simply involved counting from 0 to 30,000 Basic took 33 seconds whereas CP/M Pascal took only 20 – quite a significant increase in speed.

Forcing Basic to use an integer for the loop counter brings the time down to 13 seconds. By setting some compiler options, error checking within the Pascal program can be turned off and the program forced to use integers only. Pascal then took only 1 second – 13 times faster than Basic.

This won't always be the case, as it depends on what you are doing, but some speed increase is always assured.

There are a few restrictions with Hisoft Pascal. Neither version will allow procedures or functions as parameters, and a record type may not have a variant part. CP/M Pascal allows files of CHAR only, whereas 4T does not allow files, although variables may be stored on tape.

Pascal is a structured programmers language. I love it and would be quite happy to throw away Basic. Hisoft's versions are excellent, and I can thoroughly recommend them.

No comment!

DUDLEY BROOKE shows how to strip out those space-grabbing REMs after they have done their job

ONE of the nice things about listings in *Computing with the Amstrad* is the generous use of REM statements. These are an invaluable aid to the programmer.

Unfortunately as far as the Amstrad itself is concerned they are a waste of time and space. In fact a REM takes longer to be ignored than it takes a GOTO to be executed.

If you want to prove it you'll be glad to know that you can time any command using Program I by inserting the command in line 40 and running it. The time given will be for a single execution of the command.

What is really required is a method of removing REMs after a program has been typed in and fully debugged. Unfortunately it is almost impossible to use Basic for this task as the program would be modifying itself. The results of doing this are unpredictable and potentially disastrous.

So to have an independent program in memory and in the interests of speed we have to resort to machine code.

To discover how best to write REM Stripper I looked at the tokens used by Basic to store commands after they have been typed into a program. For further information on tokens I would recommend John Hughes' article published in the November and December 1985 issues of *Computing with the Amstrad.* Briefly though, Basic translates each command in the program into a single byte code.

A large part of a Basic program's work is jumping from line to line by means of such commands as GOTO, GOSUB and RUN. In most computers when you type a command with a line number, for example GOTO 20, the computer will store this as a token and the two bytes following will be the line number, as shown in Figure I.

When the program is running the interpreter finds the GOTO token and then picks up the line number which follows it. It then has to start at the bottom of the program and work up until it finds the relevent line.

This is very inefficient. Tutorials written for other computers recommend that the most used parts of the program are placed at the beginning and DATA statements, instructions and the such like are at the end. This type of structure is frequently seen on the CPC range of computers, but it is unnecessary and can even slow a program down when DATA statements are used frequently.

Looking at Figure II you will see that your Amstrad stores jumps in four bytes instead of three and this makes a significant difference. The &20 byte merely tells the interpreter to insert a space when listing – this is not really part of the instruction and may be disregarded.

Look at Program II and then compare this with the representation of line 10 in memory – Figure III. Note the extra byte (&1E) between the GOTO token and the line number which is the form of the command prior to running the program.

In Figure IV we see the command after it has been run – the extra byte has changed to &1D and the destination has changed to &0179. This number is the memory address of line 20 and the advantage of this is that the interpreter can now jump directly to the destination of the jump. Amazingly the GOTO in this form will now execute more quickly than a REM statement.

Some line dependent commands do not take advantage of this system, for example RESTORE. To allow you to investigate this further type in Program III and save it. It displays the bytes that make up line 10, the first line of the program. You can alter this as you wish.

464 owners might be alarmed at the DEC$ command. This is available but you must use an extra opening bracket, for example DEC$((n,"##") in line 110. It operates perfectly otherwise and makes formatting numbers very easy. To make the program easier to use I have redefined some keys on the keypad:

Key 0 executes a RUN so that line 10 is run.

Key 1 executes a RUN 60 so missing out the lines before.

Key 2 gives you a blank line 10 to fill as you wish.

Key 3 allows you to edit line 10.

When run you will see at the top of the screen the words Line Specifier and underneath some numbers. The numbers in red are the addresses in memory, those in light blue are the values at that address in hex and in decimal.

Finally there are some symbols corresponding to character values. Beneath these four bytes there is the word Tokens and more output in the same format – these numbers are the tokens and data used by the interpreter.

This aspect of Basic's operation contributes greatly to its speed but it is not the entire story.

It is a fairly simple matter to

Token	Line number
hh	hh hh

Figure I: Usual format for other computers

Token	Space	Address type	Address
A0	20	1E or 1D	hh hh

Figure II: Amstrad CPC format

Line number	Line length	GOTO token	Print space	Address type	Line number	Line end marker
0A 00	0A 00	A0	20	1E	14 00	00

Figure III: Line 10 before execution

Line number	Line length	GOTO token	Print space	Address type	Memory address	Line end marker
0A 00	0A 00	A0	20	1D	79 01	00

Figure IV: Line 10 after execution

identify and remove a REM or ' and then move the rest of the program down in memory to fill the gap. However it's not as simple as that because the REM might be the target of a line dependent command such as GOTO 90, RUN 50 or RESTORE 300. These then must be retargeted to point to the next non-REM line.

Program IV will eliminate all REM and ' statements and redirect all relevant commands with the exception of Delete as it seems pointless to retarget this if a line is already deleted.

It takes approximately 30 seconds for Rem Stripper to deREM a 10k program, depending on the number of REMs present. It is stored starting at location 30000 in memory and is 403 bytes long, including workspace. This should give ample space below for any Basic program and give plenty of room above for extension ROMs and RSXs.

To use Rem Stripper type in Program IV and run it. If you have incorrectly entered any of the data the program will inform you at which line this occurred and stop. Once the listing is correct you will be given the option to save or continue.

If you choose the save option it will save a loader program followed by the machine code as a binary file. The program will then return to the options. If you select the continue option the full stop on the key pad will be redefined to call the machine code when pressed, stripping all REMs.

My advice when using the program is to keep a master copy of the subject program containing the REMs in case any problems occur or you wish to go back and modify it at a later date. You'll find the original REMs invaluable.

```
10 REM ** Command Timer **
20 tim=TIME
30 FOR a=1 TO 1000
40 REM ** Put command here **
50 NEXT
60 tim=((TIME-tim)/300-1.09)/1000
70 PRINT "Time to execute = ";:PRINT
USING "#.#####";tim;:PRINT " secs"
```

Program I: Command timer

```
10 GOTO 20
20 PRINT "Hello!"
```

Program II: Example program

```
10 REM * First line
20 KEY 0,"run"+CHR$(13)
30 KEY 1,"run 60"+CHR$(13)
40 KEY 2,"10 "+CHR$(13)
50 KEY 3,"edit 10"+CHR$(13)
60 MODE 1:ZONE 20:z=0
70 FOR n=&170 TO &170+PEEK(&170)+256*
(PEEK(&171))-1
80 IF z=0 THEN PRINT:PRINT TAB(12) "L
INE SPECIFIER" ELSE IF z=4 THEN PRINT
:PRINT TAB(12) "COMMAND TOKENS"
90 PRINT TAB(11);
100 PEN 3:PRINT "&";HEX$(n);
110 PEN 2:PRINT " &";HEX$(PEEK(n),2);
STRING$(3,32);DEC$(PEEK(n),"###");
120 PEN 1:PRINT " "CHR$(1);CHR$(PEEK(
n))
130 z=z+1
140 NEXT
```

Program III: Line peeker

```
10 REM Rem Stripper
20 REM
30 REM By Dudley Brooke
40 REM (c) Computing with the Amstrad
50 REM
60 MODE 1:MEMORY 29999
70 lin=230
80 FOR addr=30000 TO 30400 STEP 8
90 FOR a=0 TO 7
100 READ code$:code=VAL("&"+code$):ch
eck=check+code
110 POKE addr+a,code
120 NEXT
130 READ chksum$:chksum=VAL("&"+chksu
m$)
140 IF chksum<>check THEN PRINT CHR$(
17);CHR$(12)"Error in line"lin:END
150 PRINT CHR$(11)"Line"lin"is correc
t"
160 lin=lin+10:check=0
170 NEXT
180 PRINT CHR$(11)"No Comment is now
coded":PRINT:PRINT "Press ( S ) to sa
ve or ( R ) to continue"
190 a$=UPPER$(INKEY$):IF a$="" THEN 1
90
200 IF a$="S" THEN SAVE "remkill":SAV
E "remcode",b,30000,410:CLS:GOTO 180:
REM Repeat
210 IF a$="R" THEN KEY 10,"call 30000
"+CHR$(13):CLS:PRINT "Press '.' on th
e keypad to call routine":END
220 GOTO 190
230 DATA FD,21,6F,01,FD,23,21,0,2CF
240 DATA 0,22,C0,76,FD,22,B9,76,3A6
250 DATA FD,4E,2,FD,46,3,ED,43,3C3
260 DATA BB,76,FD,4E,0,FD,46,1,3C0
270 DATA ED,43,BD,76,78,B1,C8,FD,551
280 DATA 7E,4,FE,1,28,6,FE,C5,372
290 DATA 28,53,18,B,FD,7E,5,FE,31C
300 DATA C0,28,4A,FE,C5,28,46,FD,460
310 DATA 23,FD,23,FD,23,FD,23,B,38E
320 DATA B,B,B,FD,23,B,78,B1,275
330 DATA 28,BA,FD,7E,0,FE,1,20,37C
340 DATA F2,FD,7E,1,FE,C0,28,6,45A
350 DATA FE,C5,28,2,18,E5,2A,BD,3D1
360 DATA 76,2B,22,BD,76,B7,ED,42,3DC
370 DATA EB,13,13,2A,B9,76,73,23,300
380 DATA 72,B,FD,E5,3E,FF,32,C2,490
390 DATA 76,FD,23,18,6,FD,E5,AF,445
400 DATA 32,C2,76,D1,AF,32,BF,76,451
410 DATA 2A,B9,76,ED,4B,BD,76,9,3CD
420 DATA E5,B7,ED,52,22,C0,76,E1,514
430 DATA 7E,12,23,13,B7,28,6,AF,25A
440 DATA 32,BF,76,18,F3,3A,BF,76,3E1
450 DATA FE,4,28,6,3C,32,BF,76,2D3
460 DATA 18,E6,CD,F0,75,C3,3C,75,4A4
470 DATA DD,21,6F,01,DD,23,DD,4E,399
480 DATA 0,DD,46,1,78,B1,C8,DD,3F2
490 DATA 23,DD,23,DD,23,DD,23,B,32E
500 DATA B,B,B,16,0,DD,7E,0,192
510 DATA CD,22,76,CC,4D,76,CC,75,435
520 DATA 76,DD,23,B,78,B1,28,D6,3A8
530 DATA 18,EB,FE,A0,C8,FE,9F,C8,5CE
540 DATA FE,EB,C8,FE,C8,C8,FE,CA,707
550 DATA C8,FE,C7,C8,FE,C6,C8,FE,6DF
560 DATA 96,C8,FE,A7,C8,FE,97,C8,628
570 DATA FE,81,C8,FE,2D,28,3,FE,49B
580 DATA 2C,C0,16,0,C9,DD,E5,E1,46E
590 DATA 7A,B7,28,14,2B,7E,FE,20,334
600 DATA 28,9,FE,1,28,5,FE,2C,287
610 DATA 28,1,C9,16,0,DD,E5,E1,3AB
620 DATA 23,7E,FE,20,28,FA,FE,1D,3FC
630 DATA C8,FE,1E,37,C9,F5,23,5E,45A
640 DATA 23,56,2B,F1,38,19,E5,2A,2F5
650 DATA B9,76,B7,ED,52,E1,D0,E5,5BB
660 DATA EB,ED,5B,C0,76,B7,ED,52,55F
670 DATA EB,E1,73,23,72,23,C9,E5,4A5
680 DATA 3A,C2,76,B7,20,17,2A,BB,345
690 DATA 76,7A,BC,20,10,7B,BD,20,334
700 DATA C,E1,FD,7E,2,77,23,FD,401
710 DATA 7E,3,77,23,C9,E1,23,23,30B
720 DATA C9,B8,47,0,0,0,0,0,1C8
730 DATA 0,0,0,0,0,0,0,0,0
```

Program IV: Basic version

It's all done by numbers

MIKE BIBBY helps make sense of machine code

I THINK the question I get asked most often by micro enthusiasts is: *"What exactly is machine code? I just can't make sense of all this LD (HL),nn and JP NZ business. I bought a book, but that didn't help".*

Well this series of articles is an attempt to answer that question. You may not be an accomplished machine code programmer at the end of it, but you will certainly know what machine code is, and be able to write your own simple programs.

Better than that, you'll be in a position to take advantage of the many excellent books on Z80 machine code currently on the market, and see how they fit in with your Amstrad. From then on you'll be able to teach yourself, and that's always the best way.

So what IS a machine code program?

Well, let me dodge the question by telling you that all programs are machine code, eventually, and we'll get round to exactly what that means in a minute.

First of all, tradition decrees that I tell you that the microprocessor at the heart of the Amstrad CPC464 is the Z80A, complete with 8 and 16 bit registers and a 16 bit address bus. I should then go on to discuss its arithmetic logical unit, its internal data bus and so on, referring you to an incomprehensible diagram showing its "architecture".

To heck with all that. Let's talk about it from the consumer's point of view – yours. You see, I'm not one of those "you can drive a car better if you know what's under the bonnet" freaks. I have it on good authority that gynaecologists do not make the best lovers.

So what is machine code all about? The fact is, it's all about numbers – lots of them. More precisely, it's about lots of numbers, each of which is between 0 and 255 in value.

Show me a machine code program and I'll show you a load of such numbers. Forget about LD and JP for the moment. Believe me, it's all done by numbers.

Let me explain. We're used to talking about a micro having memory aren't we. Well a micro's memory is composed of lots of individual memory cells, as is our own brain. And, just like our memory cells, a micro's memory cell can only remember so much.

In the case of the Z80, the cell can remember only one byte at a time – and a byte, you won't be surprised to learn, happens to be a number in the range 0 to 255.

An upper limit of 255 might seem a little arbitrary, but there's an excellent reason for it. It's all to do with the wiring. (Okay, we'll lift the bonnet just a little!)

Each memory cell, or location as it's more properly termed, consists of a set of eight switches, each of which can be either ON or OFF. Now by arranging the switches in various patterns of on and off, we can encode things – a sort of electrical semaphore. And what we code is – yes, you've guessed it – numbers!

Have a look at Table I. What it does is to link each switch with a number. We've labelled our switches switch 0, then switch 1 and so on up to switch 7. Notice that, yet again, computers start counting at 0. Even though we only go up to switch 7, there are eight switches in all.

Now below each switch in the table is the number linked to it (don't worry why we picked these particular numbers for the moment). Switch 0 is worth 1, switch 1 is worth 2, switch 2 is worth 4 and so on.

Notice how the value of each switch doubles as you go along. Given these values we can code numbers. For example, if switch 4 were ON, and all the others off, we'd be "hiding" the number 16. Similarly, if just switch 7 were on, we'd be coding 128.

Even better, by having more than one switch on at a time we can code other numbers than just the eight we've linked so far – we just add the values of all the switches that are ON, to arrive at the new number. For instance, if switch 5 and switch 1 were on simultaneously, and all the others were off, the number we've got is 34. Figure I shows why.

If you think about it for a moment, you'll see that the smallest number we can encode is 0 (all the switches OFF) and the largest number we can encode is 255 (all the switches on).

The really nice thing about the way we've chosen our numbers though, is that every number between 0 and 255 has its own unique pattern of switches, so there's never any confusion about the number you've coded, or stored – to use computerese – in the byte.

But, and it's a big but, writing 34 as

Switch	7	6	5	4	3	2	1	0
Value	128	64	32	16	8	4	2	1

Table I: Values associated with each switch

Switch	7	6	5	4	3	2	1	0
Value	128	64	32	16	8	4	2	1
State	off	off	on	off	off	off	on	off

gives - - - - > 32 + 2 = 34

Figure I: Encoding 34 with switches

off off on off off off on off is incredibly cumbersome. However mathematicians decided that since there were only two states for each switch (ON and OFF), they'd use the number 1 for ON and 0 for OFF. Using 1 and 0 in this way gives us what are called binary numbers. In this scheme of things 106 becomes %01101010. Figure II shows how.

You may be wondering why we've put the % in front of the 01101010. The reason is that otherwise we might mistake it for an incredibly large ordinary number. So if you see a % in front of a number it's coded in our binary way. Incidentally, each switch is known as a bit, and since a byte consists of eight such switches, we can say that there are eight bits in a byte. The article Bits and Bytes on Page 38 goes into it in more detail.

Now these eight bits in a memory byte allow us to store any number from 0 to 255 – 256 different numbers, if you remember to count 0. But if we're going to have a computer of any power we're going to need more than 256 bytes of memory.

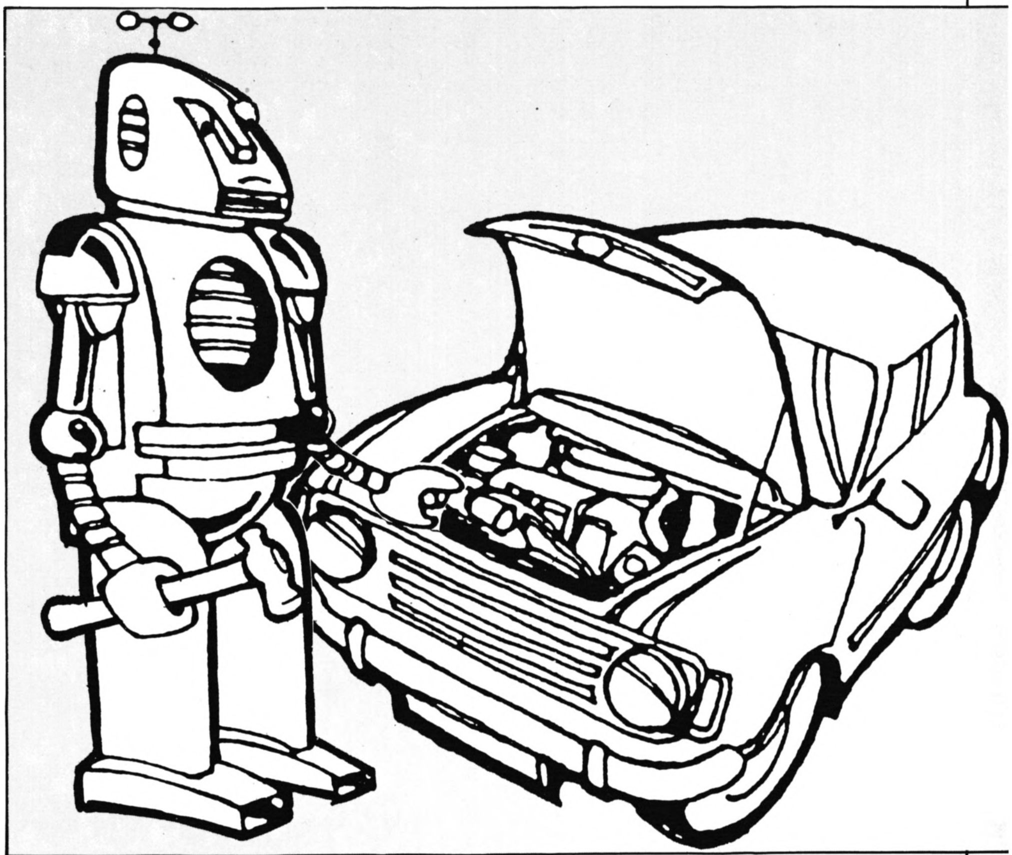
So what the micro does is to have 65536 different memory locations, numbered from 0 to 65535, to store its data in. Why 65536? Well, in order to keep track of its memory bytes, the computer has to do some more wiring.

We've already seen that having eight wires would only allow us to keep tabs on 256 locations. What the Z80 does is to double up the number of wires to 16 – which then gives it 65535 as its largest number. Look at Table II if you don't believe me.

As you can see, it's like the old Table I with an extra eight switches or bits added on top – that is, another byte.

To get the value of these extra bits we just keep on doubling. 128 was the last one, so it goes 256, 512 and so on up to 32768. The top (higher valued) set of eight bits is called the high byte of the address – hi byte for short. The bottom (lower valued) set of eight is called the low byte of the address – lo byte for short.

If all the switches are on – that is, if all the bits were set at 1 – the number these two bytes would code is:



%1111111111111111

= 32768 +		
16384		
8192		
4096	hi byte values	
2048		
1024		
512		
256		
128		
64		
32		
16	lo byte values	
8		
4		
2		
1		
65535		

Now do you believe me? The reason we've gone into so much detail is because, as I've said, machine code is all about numbers stored in the micro's memory. In fact machine code is mostly about moving those numbers (and hence the information encoded in them) around the memory of the computer – that is,

hi	bit	15	14	13	12	11	10	9	8
byte	value	32768	16384	8192	4096	2048	1024	512	256
lo	bit	7	6	5	4	3	2	1	0
byte	value	128	64	32	16	8	4	2	1

Table II: 16 bits explained – compare with Table I

Switch	7	6	5	4	3	2	1	0
Value	128	64	32	16	8	4	2	1
State	off	on	on	off	on	off	on	off
binary	0	1	1	0	1	0	1	0

Giving ----> 64 + 32 + 8 + 2 = 106 decimal
----> %01101010 binary

Figure II: The binary representation of 106 from one memory location to another.

For example, there's a large machine code program that actually runs your CPC464. It's called the operating system, or firmware. One of its jobs is to print that familiar "welcome" message on the screen when you first turn on the machine.

What happens is that the message is stored away in the micro's memory. When you switch on it copies the message from those locations into the memory reserved for the screen so you can see it. That is, the firmware machine code program moves the numbers that encode the message from one location to another.

This same sort of transfer of data occurs when you press a key. The firmware transfers the value of the key pressed from the location that remembers which key it was to the memory set aside for the screen.

When you save a Basic program the firmware's own machine code program moves the contents of the memory where the Basic program is stored out to the cassette port.

It's all about moving bytes of data around! More formally, most of machine code is concerned with moving bytes of information from one memory location to another. If you're a realist, you'll probably have guessed that there's a lot more to it than that. But have faith, most of what I'm telling you is true.

To investigate this movement of bytes further we need an analogy – in other words a meaningful lie. Suppose we have a tiny micro with only three memory locations. Figure III shows the sort of thing.

It's fairly easy to wire them up so that the numbers can move from one location to another – just join each byte to every other byte (in the figure, by the way, I've only shown one of the eight wires for clarity).

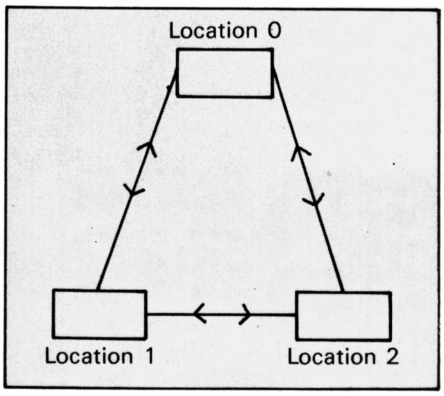


Figure III: Linking three memory locations

If you stretch your imagination you'll see that it looks a lot like a simple railway.

But suppose there were more locations, as in Figure IV. You can see the layout's getting complicated. And when you consider that the Z80 addresses tens of thousands of such locations, you can see that we've got problems – the wiring's far too complex.

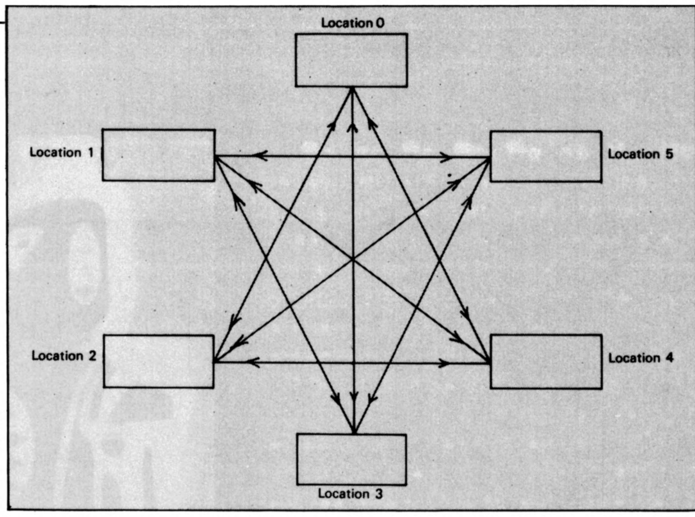


Figure IV: The complexities of linking six memory locations

Of course the answer is to stop giving each memory location its own direct lines to each and every other location. We'll do what railways do, and have junctions and branch lines.

Figure V shows such a layout. Our six locations are all connected via the major junction A. Everything passes through here. In fact there is such a memory location as A – a major junction through which traffic passes. It's deep in the heart of the Z80 and can hold one byte numbers. In computing jargon we call such a junction a register.

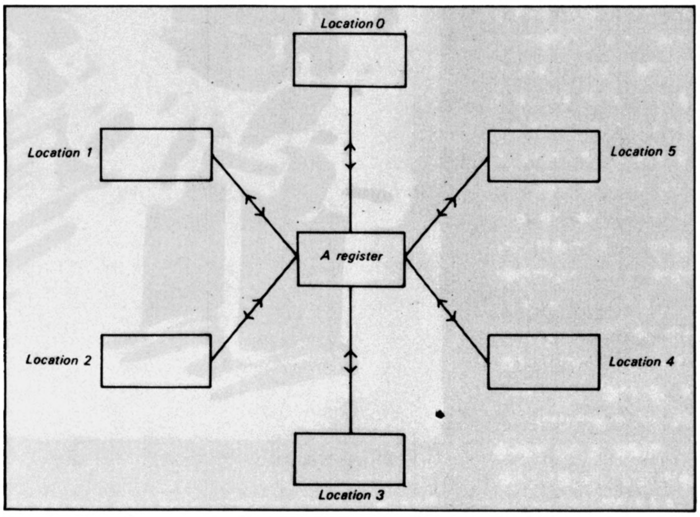


Figure V: Memory locations linked by A register

Now suppose you wanted to move a byte of information from memory location 0 to memory location 5. As you can see from the figure, you would go via register – that is, junction – A.

You would do this by giving the machine two instructions:

1. *Load register A with the number contained in memory location zero.*

2. *Load memory location 5 with the number that is in register A.*

It's a sort of microelectronic pass the parcel. The data goes from location 0 to A, then from A to 5. It's always a two-stage journey. All traffic passes through A.

In practice the actual layout is more like Figure VI, but there's still no direct traffic. Everything goes to A first and then back out.

To stretch our analogy a little further, a major rail junction like A would have lots of facilities that other junctions haven't. It's the same with the Z80 – once you've got a number in A you can do all sorts of things with it!

Also no rail designer worth his salt would depend on one major junction

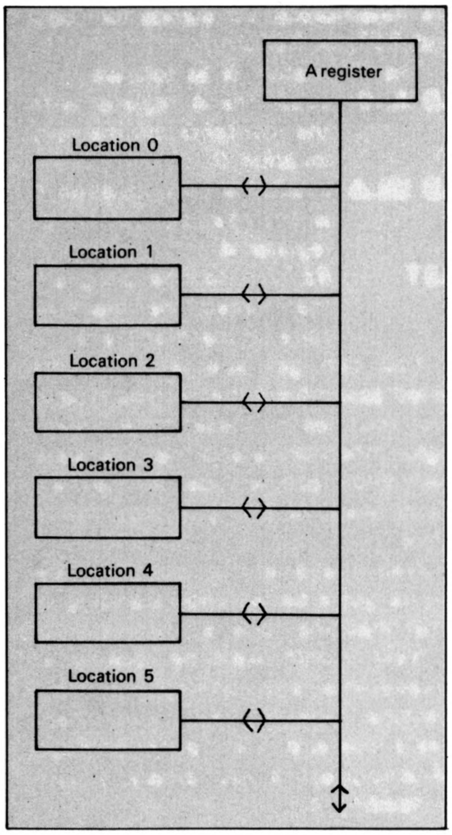
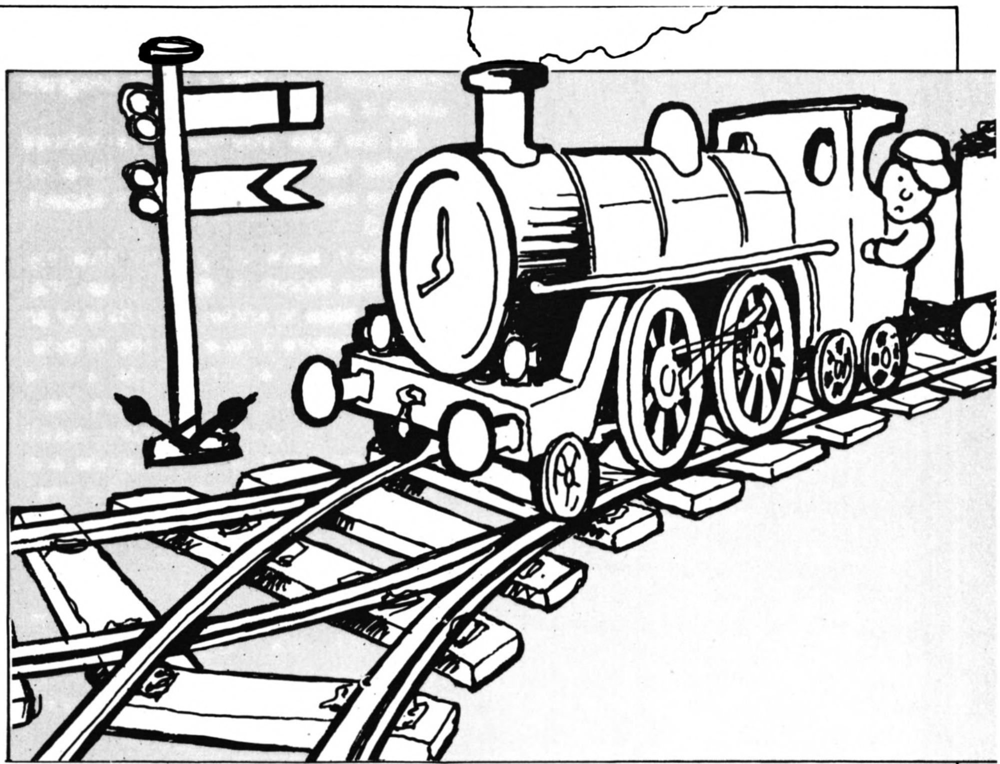


Figure VI: A more realistic representation of memory/ register linkage



– there'd be too much congestion. He'd have other junctions. Similarly the Z80 has registers other than A for much the same reasons.

And as it stands our junction/ register A doesn't have all that much capacity – just one eight bit number. This could be limiting if we wanted to deal with larger numbers, such as we use to label memory locations. Well the Z80 has got registers to handle these, too, as we'll see later on in the series.

By now I think I've convinced you that machine code's all about moving numbers around in memory. But how does the micro know what to do? How do you tell it to move these numbers, and where you want them putting?

The answer's simple – you give it a list of numbers stored in memory!

I'm not joking, honest.

The program itself is just a sequence of bytes in memory. The bytes have meaning to the Z80, you see – it's a sort of code, machine code in fact. All you do is point your Z80 at the first byte and say go. It then moves along the list of bytes doing what it's told.

Let's have a look at what this means in the context of the little program we discussed above – the one that transferred one byte from location 0 to location 5.

The actual string of bytes we need is:

58 0 0 50 5 0 201

I've written the numbers in decimal, as that's what we're used to – of course the micro reads them in binary. Figure VII explains what it's all about. When we point the Z80 at the location of the first byte of our program and tell it to go it knows that first byte is going to tell it to do something. The fancy name for this sort of "command" byte is an opcode – short for operation code.

Now 58 is an opcode that tells the Z80 to load register A with the contents of a particular location in memory. From the opcode itself, the Z80 knows that the address of this memory location will be contained in the two bytes directly following the opcode (remember you need two bytes to specify addresses).

So having understood the meaning of the opcode, the Z80 moves its attention along to these two bytes and works out the address they refer to (in this case, location 0). It then copies the contents of that address into the A register.

The Z80 has finished with the first three bytes and has done all the first opcode instructed it to. It now turns its attention to the fourth byte, which it knows must be an opcode since it has finished its previous task.

This time the byte is 50, which tells the micro to load the memory location specified in the next two bytes with the number in the A register. (In a sense, this is the mirror image of the last opcode. That loaded the accumulator from a memory

Contents of memory location	58	00	00	50	05	00	201
Meanings of above bytes	Load A with contents of the address that follows	These two bytes specify the address needed by previous opcode		Load the address following with the contents of A register	These two bytes specify the address referred to by the previous opcode		Return from whence you came

Figure VII: A simple machine code program explained location. This opcode loads a memory location from A.)

So having worked out what the opcode contained in the fourth byte wants it to do, the Z80 turns its attention to the next two bytes along, works out the address they store and copies the A register into that location.

Having finished that instruction, which used the fourth, fifth and sixth bytes, the Z80 then moves on to the seventh and last byte to find its next opcode.

The seventh byte contains 201, the opcode for return – which tells the Z80 to go back to where it was before it started or, as the jargon has it, called this progam.

This works in much the same way as RETURN does in a subroutine, causing the micro to rejoin the main flow of the program.

Notice that you don't need any extra bytes after this opcode to tell it where to return to. When this routine was called the Z80 carefully stored where it was up to for future reference – as does a Basic program when it meets a GOSUB.

By the way, you may have noticed that the two bytes specifying location five are not 0,5 as you might expect, but 5,0.

I don't want to go into this too much this month. Suffice it to say that the Z80 likes to know the lo byte of an address before it receives the hi byte.

Let's have another look at the machine code program we've developed. I'm going to split each instruction – that is each opcode and its data bytes – onto a separate line.

```
58 0 0
50 5 0
201
```

Doesn't make immediate sense, does it? Our brain is much more adept at making sense of words than numbers. Have a look at the program in a new form, that uses "words":

```
LD A,(0)  58 0 0
LD (5),A  50 5 0
RET       201
```

The symbols on the lefthand side are mnemonics. LD stands for LoaD and RET for RETurn.

The translation is as follows:

LD A,(0) *LoaD the A register with the contents of memory location 0.*

LD (5),A *LoaD memory location 5 with the contents of register A.*

RET *RETurn to the program that called the machine code in the first place.*

You can get special programs called assemblers that let you type in your routines in these more meaningful mnemonics and then translate them into machine code, but they're a luxury we'll be doing without for a while.

Well that's all for now. I hope you've got a better understanding of what machine code is.

● *Next month we'll be looking at hexadecimal and running your own machine code programs. Until then, take a look at Bits and Bytes on Page 38. And practice your binary – you'll be needing it!*

Printing labels with Mini Office II

PRODUCING labels with Mini Office II is delightfully simple. All that is required is to work methodically through the following three stages:

Enter the name and address details into the database, produce a mask or layout of the label and finally set the stationery parameters to fit the mask to the labels.

The first of these stages needs little explanation since the database has been discussed previously. The only point to note is that mask production is governed by the field number.

To determine the field numbers select Database from the main menu and place the data disc in the drive.

If a file containing names and addresses already exists continue as follows: Select Load/Save/Print followed by select Load data, enter filename, select Database menu and then Edit structure. The last step is advisable so you can make a note on a sheet of paper of the field numbers, titles and field length.

If no such file has yet been built go directly to Edit structure and when this has been done press Escape, select Edit data and enter the names and addresses on to the database.

```
 **** **** ***************
  *******************
   ***************
    ***************
     ************  **********
```

Figure I: The result of test print

When entry is complete press Escape, select Load/Save/Print and then Save all records.

Now that a file exists we can move on to producing the mask of the label as follows – select Database menu, select Mini Office II menu and change to the Mini Office II disc.

Now select Label Printer, change to the data disc, select Load file, enter filename and select Edit format.

To design the mask pick Edit label. At this point a window will appear on the screen with the cursor located in the top left corner of it.

Use the cursor keys to move round the screen and when in position press the # symbol and follow it by the appropriate field number. If you did not make a note of the field numbers you can press Tab and get a summary of the fields.

The only point to watch at this stage is that while setting up this mask you must ensure that you move the cursor sufficiently far between one field and the next to ensure that the back end of the first of these is not over-written by the second.

When satisfied with the design press Escape. This takes you back to the Edit label menu where you can match the mask with the stationery, and possibly your printer by entering the parameters prompted so far.

Label stationery comes in all different shapes and sizes, from one to five labels across the page, five to eight lines deep, 25 to 40 characters across and with one to three line gaps between them.

You may need a ruler to measure up some of the distances in order to be able to enter the correct values, particularly if you wish to switch to eight lines per inch from the default six.

When these entries have been completed select Label Print Menu, choose Save format and enter filename. You are now ready to print off your labels.

Select Print labels. The number of labels shown will be determined by the number of records you input to the database file.

Now choose Test print in order to check that you have the paper correctly adjusted in the printer and that the mask and stationery parameters are correct. The resulting printout is shown in Figure I. When satisfied with the layout select Print labels and the labels will be produced. The example shown in Figure II is the one used to produce a single label across the page.

```
Mr.  J B  Lewis
 14, Newton Way
  Boyle Lea
   Flemingshire
                   FL9 9ZZ

Mr.  Jo  Stork
 c/o Europress
  68, Chester Road
   Hazel Grove
    Stockport     SK7 5NY

         SAGESOFT PLC.
Regent Centre
 Gosforth
  Newcastle.
                   NE3 3DS

          COMPACT SOFTWARE
1, Ensbury Park Road
 Bournemouth

                   BH9 2SQ
```

Figure II: The final printout

If there was a problem with the test print return to Edit format and adjust accordingly or alternatively use the Send printer codes option to fit the database records on to the printer. Do not forget to save the working format again.

The only other point to make is that I've assumed that you would need to produce a format. If you already have a working format you would load this into the Mini Office II labelling module immediately after the address file has been loaded. ■

JO STORK continues his series on making the most of Mini Office II

SUBSCRIPTIONS & BACK ISSUES

AUGUST 1986 (#6008)

Our Premiere Edition and a true collectors item!
Game of The Month - Diamond Digger
Business Users got a look at Mini Office II, PCW Comms pack and lots more. You'd better hurry - there aren't too many left!

SEPTEMBER 1986 (#6009)

Game of the Month was Ice Front. September 1986 featured no less than 22 different and interesting articles and listings including: Hardware reviews on RS232 and an 8-bit printer port, a Fill utility for the 464, a simple music tutor, a Windows utility to help in key in those listings, reviews of Spreadsheets and Databases for the serious user and two articles on Locoscript including 10 useful tips on its use.

OCTOBER 1986 (#6010)

Our October issue featured one of the best games yet featured in a magazine listing (Da Bells) and as a bonus for 464 owners we presented a challenging Mouse game.
Two utilities were listed (Character Generator and Simple Sprites) as well as a full listing of the Pilot language! For those into the educational side of things we gave you Letter Litter and there was of course the first review of Amstrad's new PC to be published in Australia.

NOVEMBER 1986 (#6011)

A truly bumper 80 page edtion which featured 4 new series on CP/M, learning Basic, Sound and Public Domain Software. A super-fast Fill utility and a screen clock made up Novembers two utilities and there were reviews of Forth, and two disk drive options for CPC computers. Business users got the low-down on IBM compatibles, reviews of DR Draw and Pocket Wordstar as well as another look at one of the features of Mini Office II. PCW owners got their first game listing and Game of the Month was Discman.

Obviously there has been lots more including regular and irregular columns on Adventures, Puzzles and our monthly Software Survey to name but a few.

Back Issues and new subscriptions may be ordered on the form opposite or by 'phone using your Bankcard or Mastercard. If you don't wish to cut your magazine please feel free to photocopy the order form or just write us a note.

To obtain your copy of the above back issues please use the order form opposite or call [002] 29 4377

ORDER FORM

SUBSCRIPTIONS

5001	MAGAZINE ONLY	$40.00
5002	MAGAZINE + TAPE	$80.00
5003	MAGAZINE + QUARTERLY DISK	$105.00

SOFTWARE ON TAPE

1001	TASWORD	$36.95
1002	3D MONSTER CHASE	$ 9.95
1003	DRAGON'S GOLD	$ 9.95
1004	ALIEN BREAK-IN	$ 9.95
1005	ATOM SMASHER	$ 9.95
1006	TOOLBOX	$19.95
1007	FLEXIFREND	$19.95
1008	GRASP	$19.95
1009	CHAOS FACTOR	$15.95
1010	MUSICO	$17.95
1011	DRUMKIT	$16.95
1012	MUSIC COMPOSER	$17.95
1013	EASIDATA II	$29.45
1014	EASIDATA III	$41.45
1015	DATABASE/MAIL LIST	$29.45
1019	POT POURRI VOL. 1	N/A
1020	POT POURRI VOL. 2	N/A

SOFTWARE ON DISK

2001	TASWORD	$48.95
2009	CHAOS FACTOR	$27.95
2012	MUSIC COMPOSER	$29.95
2014	EASIDATA III	$53.45
2015	DATABASE/MAIL LIST	$41.45
2016	EASIWORD COMBO	$49.95
2017	GENESIS	$39.95
2018	EASY MUSIC	$34.95
2019	POT POURRI VOL. 1	$19.95
2020	POT POURRI VOL. 2	$19.95

SPECIALS

4001	4 OF A KIND(TAPE)	$29.95
4002	4 OF A KIND(DISK)	$39.95

BOOKS

3001	AMSTRAD HANDBOOK	$9.95
3002	AMSTRAD COMPUTING	$17.95

BACK ISSUES

6008	CWTA PREMIERE EDITION	$4.50
6009	CWTA SEPTEMBER ISSUE	$4.50
6010	CWTA OCTOBER ISSUE	$4.50
6011	CWTA NOVEMBER ISSUE	$4.50

TAPES

7008	TAPE (8/86) DIAMOND DIG	$7.50
7009	TAPE (9/86) ICE FRONT	$7.50
7010	TAPE (10/86) DA BELLS	$7.50
7011	TAPE(11/86) DISCMAN	$7.50

TITLE	CAT #	PRICE
	TOTAL	

Bankcard ☐ Mastercard ☐ Expiry ☐☐☐☐

☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Signed ____________________

Name ____________________

Address ____________________

State ____________ Postcode ____________

Mail orders to:

Strategy Software
P.O. BOX 11
Blackmans Bay
Tas. 7152 **or Call (002) 29 4377**

A couple of classics

ALEATOIRE advises on Knightlore, and poses Ramanujan's problem

THE classic format for arcade adventures appeared in 1984 when Ultimate unveiled Knightlore – the first 3D one to appear in Britain.

By now thousands of you must be familiar with the diamond shaped 3D view created by looking down to the middle of each room from one of the corners.

However there still must be loads of you who have failed to crack the game and, apart from a couple of cheat pokes, I have never seen a genuine analysis and fair solution of this original classic.

To play the game you must have a map (see Figure I), an understanding of how and where the eight different objects are scattered and the ability to anticipate what order they must be placed in the central cauldron.

The map gives one of the eight possible random scatters, and all you need do to generate the other seven is add 1 to each number and take the result modulo 8 – that is 0 –> 1, 1 –> 2, . . . 6 –> 7, 7 –> 0.

The order of objects into the cauldron always starts somewhere in the cyclic pattern 7 6 5 4 3 2 1 4 2 7 1 6 5 3 and back to 7. As there are 14 objects in all knowing the first two required means you know precisely the order of the remaining 12 that must be collected, and so you can plan your itinerary accordingly.

Of course moving around is not easy, but further analysis reveals that many of the rooms/areas can be ignored entirely because they are too dangerous, too awkward or simply a waste of time. Such rooms are marked with an X in Figure I.

Once all this is appreciated almost anyone who has a little expertise with the joystick and a lot of patience can solve the game. Note that patience is often essential. For example, the

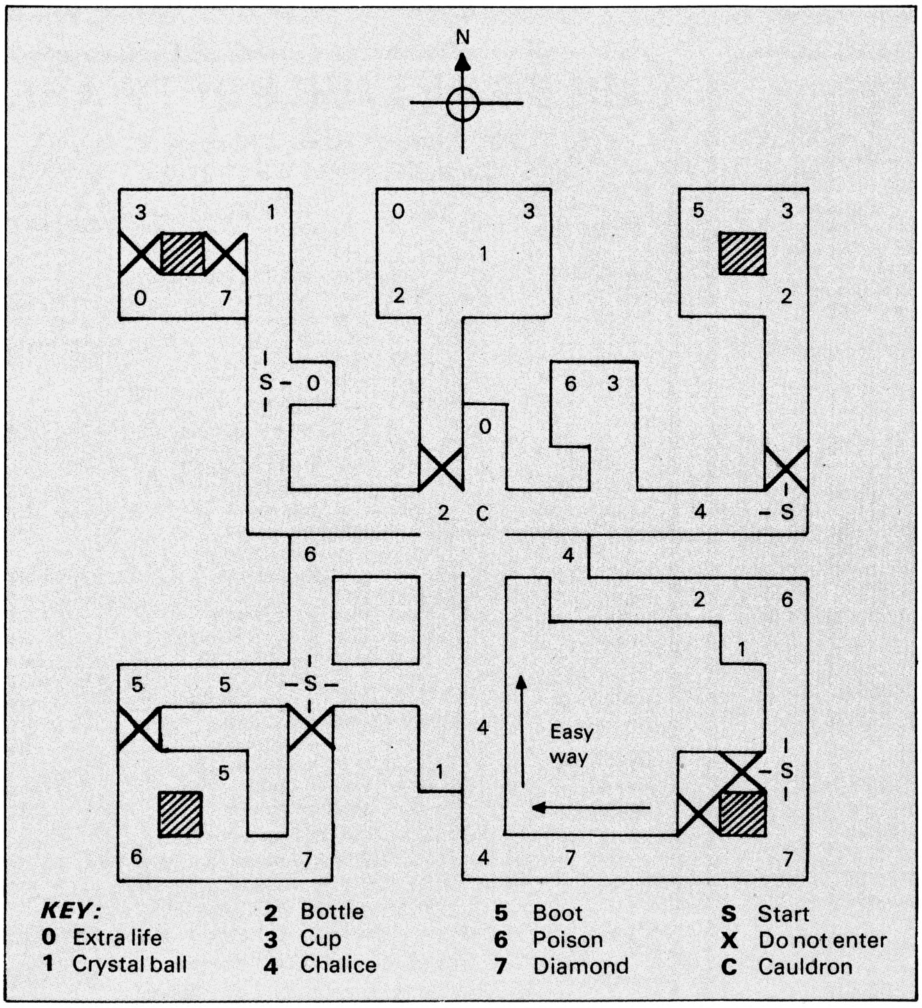


Figure I: Knightlore map

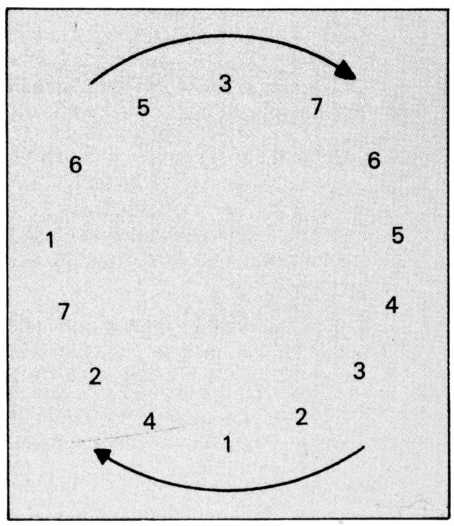


Figure II: Cyclic order of objects into the cauldron room in the top left hand corner cannot be reached if you are a werewolf – you must wait to metamorphise back into Sabreman.

Incidentally some of the objects are hidden, but you can verify their presence because the program never allows more than two objects to be dropped in a room.

To actually get hidden objects needs a little confidence as you drop into the unknown – but given that, plus a little practice, and you should succeed well within the 40 day limit. My best time is 21 days, so the game still has some interest even after the dust has danced and whirled around you for the first time and you have been told to "go forth to miremare".

Another classic puzzle was set many years ago by the British mathematician G.H. Hardy when visiting a sick friend in hospital. The friend, an Indian called Ramanujan, was a self-taught mathematical genius who lived and breathed number theory – mention almost any random number to him and he could give it a unique property.

Hardy, attempting to make conversation, remarked that the taxi he had just used had the rather uninteresting number 1729. "On the contrary", said Ramanujan, "it is the smallest number that can be represented as the sum of two cubes in two different ways" – that is, 1729 = 1^3+12^3 = 9^3+10^3.

"With the aid of a computer and a little bit of analysis you can solve it in less than an hour"

Hardy immediately asked what then was the smallest number representable as the sum of two biquadrates – that is, numbers to the power of four. Ramanujan replied that he did not know the answer, though he imagined it must be very large. A few months later he was dead.

Can you solve Ramanujan's problem? To encourage you I have calculated that 3262811042 = 7^4+239^4 = 157^4+227^4.

However this is not the smallest solution by a long way. The attraction of the problem is that with the aid of a computer and a little bit of analysis you can solve it in less than an hour. There is a prize for the first correct answer, but tackling the problem is a rewarding exercise in itself.

My advice is to study the two cubes in two different ways first, and then generate the 10 unique numbers less than 100,000 that can be expressed in this way. Having done that you should see how to efficiently solve the much bigger biquadrate case. ■

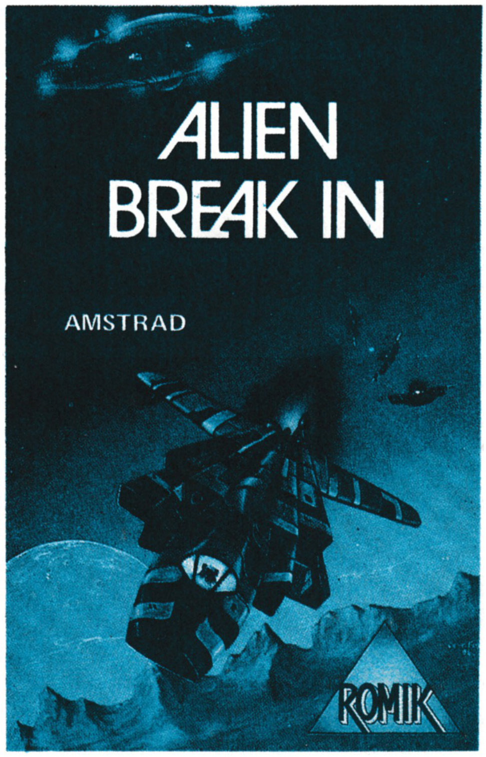
Next Month....

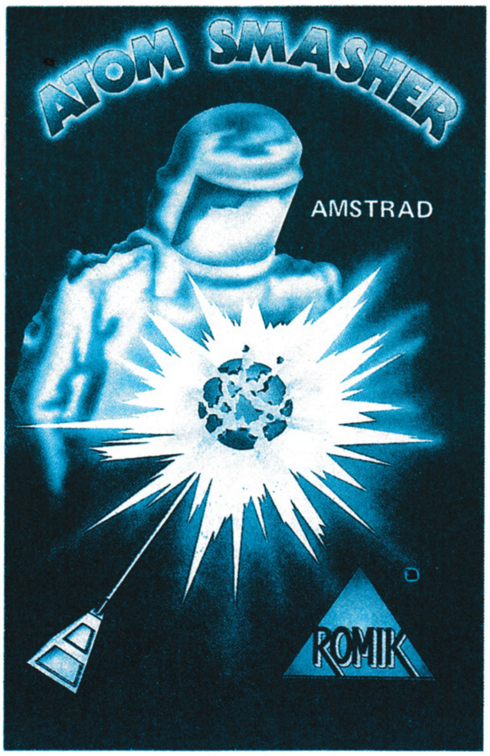
Another great issue featuring all the regulars with another two parts of First Steps - our beginners introduction to Basic, Shane Kelly's Public Domain column and we continue our regular series on Sound, CP/M, Machine Code and Graphics. There's also the regular features of Ready Reference and Analysis as well as a seven page Software Survey.

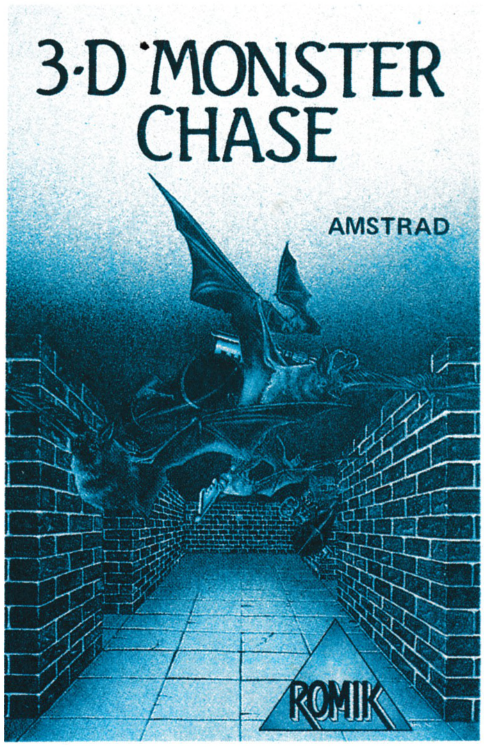
We present two utilities in Profiler to help speed up those Basic programs and Double Height which enables you to display double height characters on your CPC screen.

Two games are featured: Othello and Spiders Web, together with an educational listing for the young speller and a musical interlude for dk'Tronics light pen owners. We also continue our Forth article from this month's issue.

FOUR OF A KIND







$9.95 Each on cassette
$29.95 for all four on cassette
$39.95 for all four on one disk

Strategy Software
P.O. Box 11
Blackmans Bay
Tasmania 7152