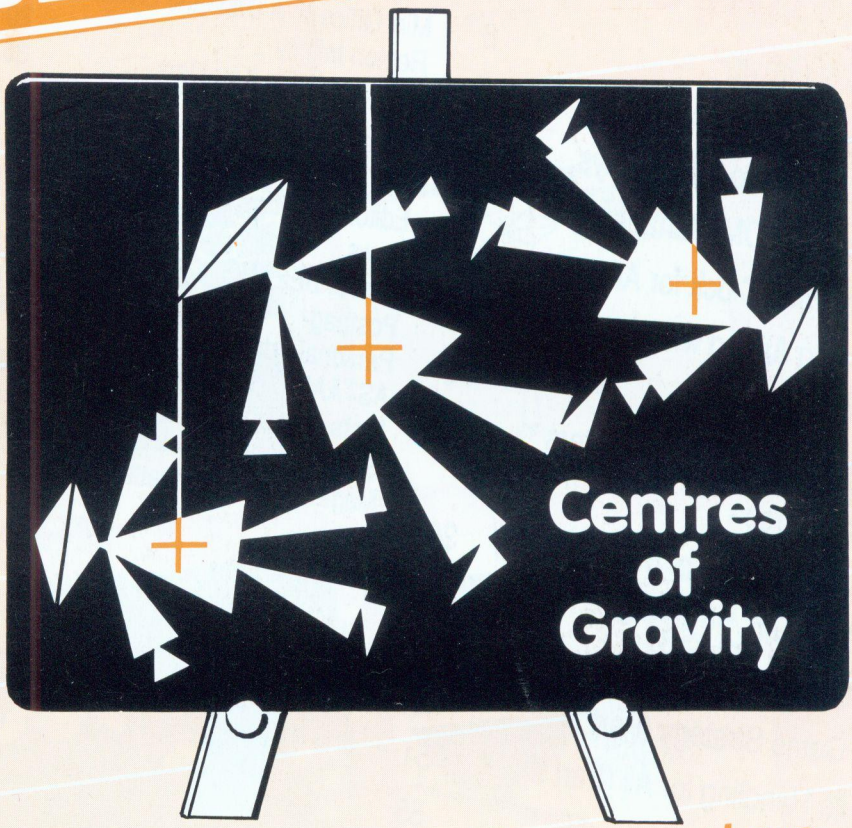
Vol.7 No.8 January/February 1989

BEEBUG

FOR THE BBC MICRO & MASTER SERIES



Programs for Education

- ADFS DIRECTORY ZAPPER
- DRAUGHTS
- AUDIO TAPE INLAYS
- REPTON INFINITY

BEEBUG Vol.7 No.8 January/February 1989

FEATURES

REVIEWS

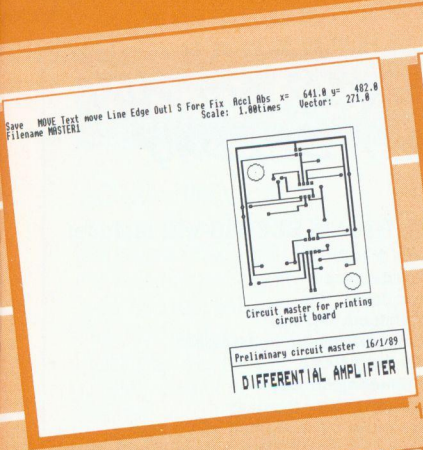
REGULAR ITEMS

HINTS & TIPS

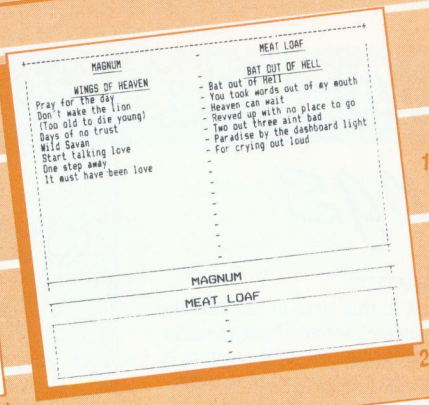
PROGRAM INFORMATION

All listings published in BEEBUG magazine are produced directly from working programs. They are formatted using LISTO 1 and WIDTH 40. The space following the line number is to aid readability only, and may be omitted when the program is typed in. However, the rest of each line should be entered exactly as printed, and checked carefully. When entering a listing, pay special attention to the difference between the digit one and a lower case l (L). Also note that the vertical bar character (Shift \) is reproduced in listings as |.

All programs in BEEBUG magazine will run on any BBC micro with Basic II or later, unless otherwise indicated. Members with Basic I are referred to the article on page 44 of BEEBUG Vol.7 No.2 (reprints

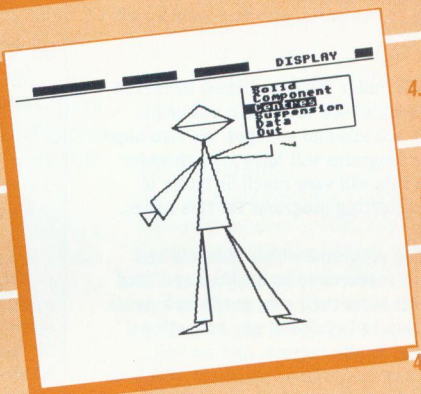


1.

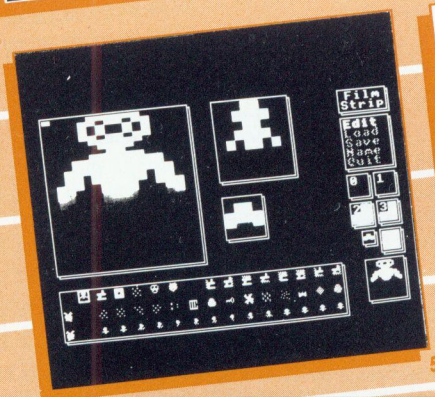


2.

3.



4.



5.

6.

available on receipt of an A5 SAE), and are strongly advised to upgrade to Basic II. Any second processor fitted to the computer should be turned off before the programs are run.

Where a program requires a certain configuration, this is indicated by symbols at the beginning of the article (as shown opposite). Any other requirements are referred to explicitly in the text of the article.

Program will not function on a cassette-based system.

Program needs at least one bank of sideways RAM.

Program is for Master 128 and Compact only.

Editor's Jottings

PROGRAMS FOR EDUCATION

We are pleased to present in this issue of BEEBUG the first of our recently announced series of programs for 1989 with an educational theme. The first program in the series deals with the topic of Centres of Gravity, and will be covered in two parts, one this month and one next.

We expect to include under the educational banner, a variety of programs which we anticipate will be of interest to those in schools and colleges. We also hope that many of these programs will have a much wider appeal, and indeed this will very much be borne in mind when we are selecting programs for this series.

Our aim is to publish programs which teachers and others can view as a resource to be adapted and used in whatever way best suits their own needs and styles. It is not our intention to indulge in any educational theory; we leave that to the experts. And we will certainly listen to any feedback or comment which we receive. We would also be interested to evaluate any of your own programs for publication in this series.

BEST OF BEEBUG - ASTAAD

We are publishing this month a further disc in our *Best of BEEBUG* range, to add to the two existing discs already available. The latest disc is a little different to the previous ones as it focuses solely on the ASTAAD CAD package (for the Master 128 and Compact), which we have been publishing in recent issues of BEEBUG. The many illustrations have shown just how much potential ASTAAD has, and we believe that this system provides many of the features to be found in costly commercial packages.

The new ASTAAD disc contains all the programs needed, updated for improved performance, together with sample files and supporting documentation and keystrips. This puts everything you need to use ASTAAD on one convenient disc, and at a price to BEEBUG members which is far less than would be charged commercially. See page 35 for further details.

This month's telesoftware password is *halfpenny*.

LOOKING AHEAD WITH BEEBUG

The March Issue of BEEBUG will include:

Applications & Utilities:
- 3D Landscapes
- Disc Indexer
- Dual Column Listings
- Decimal Arithmetic in Assembler
- Roller Rally Game
- File Handling for All
- Workshop - Game Strategy
- First Course - Timed Input and Output
- BEEBUG Education - the National Curriculum
- 512 Forum

Reviews:
- Heritage - a genealogy package from Bel Tech
- Colossus 4 Bridge - CDS Software
- ShareVAL - share investment analysis from Nasco Software

Plus News, Postbag, Hints and more.

RISC USER

RISC User is the largest circulation magazine devoted entirely to the Acorn Archimedes range. It is available on subscription to all BEEBUG members at a substantially reduced rate (see page 67).

The March Issue of Risc USER will include:

Features:
- New Fonts
- Toolbox Disc Searches
- Image Manipulation
- What's New in RISC OS
- RISC OS Multi-tasking
- RISC OS Applications Directories
- Archimedes Visuals
- Sprite File Manager
- Using Logistix

Reviews:
- Art/Graphics Packages
- Archimedes Games Survey
- Clares' Toolbox Plus
- RISC Forth

and more.

RISC User is the ideal magazine to keep you up to date with the Archimedes scene in every respect, and is particularly useful if you are contemplating the purchase of an Archimedes in the near future.

*The latest details of the contents and distribution of both magazines are contained in the BEEBUG area of Micronet. Just type *BEEBUG# when on-line.*

News News News News News

ACORN UNIX

Acorn are shortly to launch their UNIX-based workstation. The system, called the R140, runs Berkeley 4.3 UNIX with the various extensions needed to bring it up to system 5 level. Also supplied is the *X-Desktop* window system from IXI. The R140's hardware is basically that of an Archimedes 440, but with the hard-disc capacity increased to 50Mb, most of this being used for the system software. Despite the similarity, Acorn claim that there will be no upgrade path for 440 machines to UNIX. The R140 is supplied with compilers for C, Fortran and Pascal, as well as Uniplex productivity software. Future hardware upgrades will include an Ethernet interface and a floating point co-processor, both of which will also work with existing Archimedes systems.

The R140 will cost around £4000 (ex. VAT) including a monitor, and will be officially launched at the Which Computer Show at the NEC, Birmingham, from 21st to 24th February.

IT'S ALL A QUESTION OF SPORT

Avid viewers of the BBC TV quiz show 'A Question of Sport' can now buy a computerised version of the game. You can either play the computer, or a human opponent, and each player has to choose his team from a selection of sports personalities. All the normal rounds of the quiz are implemented, and digitised pictures of the team members are included for added realism.

A Question of Sport is produced by Superior Software under the Elite banner, and is available in several different formats for a whole range of computers. The cassette version for an Electron or BBC Micro costs £12.95, while the 5.25" disc version for the BBC is £14.95, and the 3.5" Compact version is £19.95 (all prices inc. VAT).

SUMMER SCHOOL

Bradford and Ilkley Community College are running a summer school in computing for beginners. The residential courses last for eight days, with starting dates of 8th July or 15th July. Each course will be based around BBC micros, and will cover all aspects from keyboard familiarity, through applications and games, up to actual programming. The cost of the course is £183.52, including full board etc., and there is a limit of twelve places on each course. For more details contact The Bradford and Ilkley Community College, Great Horton Road, Bradford BD7 1AY, tel. (0274) 546812.

SCHOOL'S HOT-LINE

Acorn has started a special *School's Service* to provide immediate help for teachers using Acorn machines. The service includes technical support, training, an education news letter, and up-to-the-minute product information. Acorn has offered all the schools in the country the chance to join this service free of charge.

Z88 MODEM

The *Pocket Stradcom* from Dataflex Design is the first BABT approved modem specifically for use with Cambridge Computer's Z88. The battery powered modem supports V21 and V22 standards (300/300 and 1200/1200 baud), and is supplied with the *Zterm* comms package from Wordmongers. The *Pocket Stradcom* costs £274.85 inc. VAT, and includes a mains adaptor and free registration to Telecom Gold. While this is dearer than the official Cambridge Computer modem (see review in BEEBUG Volume 7 No.6), it does have the advantage of being BABT approved. Further details can be obtained by ringing Dataflex Design on 01-543 6417.

COMPUTER COMMUNICATIONS

Transfolink is a novel kit for transferring data between computers. The system includes all the cables and software needed to transfer data between a BBC micro, an IBM PC (or compatible), and an Atari ST, with versions for the Apple Macintosh, Commodore Amiga and Z88 promised shortly. *Transfolink* is supplied in a neat looking briefcase, with a 'Filofax compatible' manual. The whole system costs £90.85 inc. VAT, and is available from Silicon Croft, PO Box 1258, Birmingham B13 8NU, tel. 021-442 4042.

FAX*FILE PRICE

In the review last month of the *Fax*File Extension Disc* from Mewsoft, it was stated that the price for the ADFS version of the package (including the original Fax*File) was £22.09 inc. VAT. This should have been £22.90 inc. VAT.

Programs for Education

Centres of Gravity (Part 1)

by David Lowndes Williams

This ingenious package, specially written for BEEBUG's new educational series, is aimed at acquainting students with the topic of Centres of Gravity. It is presented in two parts.

INTRODUCTION

It isn't always obvious where the centre of gravity of an object lies. It may even lie outside the confines of the object altogether. This package enables you to generate a two dimensional shape (composed of triangles), edit it, and display the individual and composite centres of gravity.

WHAT IT DOES

The package is too long to fit in a BBC or Master in a single program, it is therefore presented in two parts. This will require a disc system to move between the two parts of the program, and store the data in a file (called "wkfile") between programs. Part 1 this month allows you to create and edit 2D objects. Part 2 (next issue) performs the mathematics, and allows your object to be displayed in a number of ways - one of which is to suspend the shape from a point, the shape is then drawn as if it was hanging from this point under the influence of gravity. Other display options include drawing the shape and its centre of mass (assuming the shape is cut out of a uniform lamina of negligible thickness), and another option prints out a list of co-ordinates of the individual triangles which make up the shape. It also gives the co-ordinates of the centre of mass.

LIMITATIONS

The maximum number of triangles that can be used to create the shape is determined automatically by the PAGE setting of your machine. The resulting number of triangles is displayed at the top right of the screen. A PAGE setting of &1900 or below is necessary in order to allow a realistic number of triangles. You may find the ROM Manager program elsewhere in this issue useful in disabling those ROMs that may be claiming workspace thus raising the value of PAGE above &1900.

ENTERING THE PROGRAM

The program CENTRE listed here is an editor for creating shapes. The co-ordinates of the corners of each triangle are stored in the array, d%(C,T) where T=1 represents the data for triangle 1, T=2 for triangle 2, etc. The information relating to the number of points making up the entire shape is stored in d%(C,0). See Table 1 for a full explanation.

E% is set to zero before entering the program to let it know that it is not being entered from the other program. E% is set to TRUE (-1) once the DISPLAY program (to follow in Part 2) has been called so that when re-entering DISPLAY, the temporary file which holds the data for the current shape, can be re-loaded.

Type in the program and save it away to disc under the name CENTRE. In order to try the program out you will need to type:

```
E%=0:CHAIN"CENTRE"
```

C / T	0	1	2	3	4	5	6	7	8
0	Current Maximum Point	Angle of Rot (deg)	Point last Visited x	y	Point of Suspension x	y	General Centre of Gravity x	y	Area of Entire Shape
1	x1	y2	x2	y2	x3	y3	cx	cy	Area
2	x1	y2	x2	y2	x3	y3	cx	cy	Area
.									
11	x1	y2	x2	y2	x3	y3	cx	cy	Area

Table 1. Data storage in array d%(C,T)

Upon start up, the four words CREATE, EDIT, FILE, and DISPLAY will be displayed, and the cursor keys and Return can be used to select one of these. The Escape key has been disabled to prevent accidental loss of data, however you may quit the program by pressing the Shift and Escape keys together.

If CREATE is selected, a cross will appear which can be moved around the screen. Pressing Return will select this point. Another cursor will appear connected via a 'rubber band' to the first point, which can also be selected by pressing Return. A third cursor will then appear; to complete the triangle, move this point to the desired position and press Return again. This procedure enables you to enter a shape as a succession of triangles. Any attempt to enter more than the maximum permitted number of triangles for your machine (as displayed top right) will be ignored.

Selecting EDIT allows you to edit points already entered. This can be done by positioning the cursor on (or very close to) the point that you wish to change. If more than one corner of a triangle lies at one point then only one of the triangle's corners can be edited at a time.

Selecting FILE enables you to load and save data. Selecting 'L' when prompted loads data into memory. This will result in the current data being 'overwritten' and hence lost. Selecting 'S' at this stage saves the current data.

DISPLAY will fail (giving a "not found" error) until Part 2 of the program has been typed in from the next issue. It will perform all of the centre of gravity calculations and also display the shape, the 'suspension' display being of particular interest, as the shape has to be rotated by the correct angle to make it appear to be hanging. The current data is saved under the filename "wkfile" and then re-loaded when the DISPLAY program is called.

You can prepare some shapes using the CREATE program supplied this month. And we have also included on the magazine disc a shape file called "person" which is depicted in Figure 1.

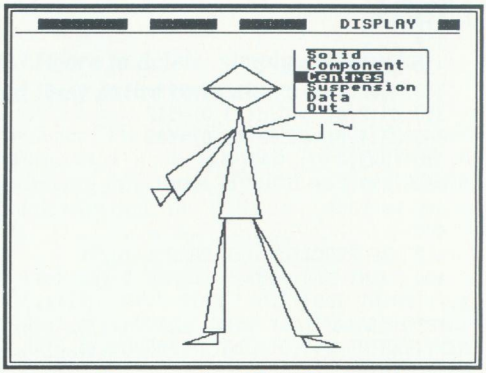


Figure 1. The object displayed

```
   10 REM Program CREATE CENTRES
   20 REM Version B1.5
   30 REM Author  David Lowndes Williams
   40 REM BEEBUG  Jan/Feb 1989
   50 REM Program subject to copyright
   60 :
   70 ON ERROR GOTO 260
   80 maxt=INT((&1B00-PAGE)/36)
   90 IF maxt<5 CLS:VDU7:PRINT TAB(0,10)
"Not enough memory to run the package,"'
"PAGE needs to be at &1900 or below":GOT
O 300
  100 MODE1:DIM d%(8,maxt):PROCinit
  110 VDU24,0;0;1279;976;:a=1:*FX4,1
  120 IF E%=TRUE PROCload_save("L","wkfi
le"):PROCdisp2 ELSE E%=TRUE
  130 REPEAT:REPEAT:PROClight(a):g=GET
  140 IF g=136 a=a-1
  150 IF g=137a=a+1
  160 IF a>4 a=4
  170 IF a<1 a=1
  180 UNTIL g=13:COLOUR128:COLOUR3
  190 IF a=1 PROCcreate
  200 IF a=2 PROCedit
  210 IF a=3 PROCfile
  220 IF a=4 PROCload_save("S","wkfile")
:CHAIN"DISPLAY"
  230 UNTIL FALSE
  240 END
  250 :
  260 IF ERR=17 AND INKEY-1 THEN 300
  270 IF ERR=17 COLOUR128:VDU26:CLS:MOVE
0,980:DRAW1280,980:VDU24,0;0;1279;976;:P
ROCdisp2:GOTO 130
  280 IF ERR=222 VDU7:COLOUR128:VDU26:CL
S:MOVE0,980:DRAW1280,980:VDU24,0;0;1279;
976;:PROCdisp2:GOTO 130
  290 REPORT:PRINT" at line ";ERL
  300 *FX4
```

```
  310 END
  320 :
  330 DEF PROCinit
  340 MOVE0,980:DRAW1280,980:d%(0,0)=0
  350 d%(2,0)=640:d%(3,0)=512
  360 d%(4,0)=640:d%(5,0)=512
  370 *FX9,4
  380 *FX10,4
  390 ENDPROC
  400 :
  410 DEF PROClight(a):PROCcol(a,1)
  420 PRINT TAB(2,0) " CREATE ":PROCcol(
a,2):PRINT TAB(12,0)" EDIT ":PROCcol(a,3
):PRINT TAB(20,0)" FILE ":PROCcol(a,4):P
RINT TAB(28,0)" DISPLAY ":PROCcol(a,5):P
RINT TAB(38,0)STR$maxt
  430 ENDPROC
  440 :
  450 DEF PROCcol(a,b)
  460 IF a<>b COLOUR129:COLOUR3 ELSE COL
OUR131:COLOUR0
  470 ENDPROC
  480 :
  490 DEF PROCcursor(x,y,r):MOVEx-r,y
  500 DRAWx+r,y:MOVEx,y-r:DRAWx,y+r
  510 ENDPROC
  520 :
  530 DEF PROCcreate
  540 IF d%(0,0)>=maxt VDU7:ENDPROC
  550 LOCALx,y,Ox,Oy,k,Ax,Ay,Bx,By,Cx,Cy
,cur
  560 COLOUR128:PRINT TAB(2,0)" CREATE "
  570 SOUND 1,-10,200,2:x=d%(2,0)
  580 y=d%(3,0):GCOL3,2
  590 PROCcursor(x,y,16)
  600 REPEAT:Ox=x:Oy=y:k=FNkeyboard
  610 PROCcursor(Ox,Oy,16)
  620 PROCcursor(x,y,16):UNTIL k
  630 Ax=x:Ay=y
  640 SOUND 1,-5,200,2:GCOL 3,2
  650 REPEAT:Ox=x:Oy=y:k=FNkeyboard
  660 MOVE Ax,Ay:DRAW Ox,Oy
  670 PROCcursor(Ox,Oy,16):MOVE Ax,Ay
  680 DRAW x,y:PROCcursor(x,y,16)
  690 UNTIL k:GCOL 0,3:MOVE Ax,Ay
  700 DRAW x,y:GCOL 3,2:Bx=x:By=y
  710 SOUND1,-5,200,2:GCOL 3,2
  720 REPEAT:Ox=x:Oy=y:k=FNkeyboard
  730 MOVE Ax,Ay:DRAW Ox,Oy:DRAW Bx,By
  740 PROCcursor(Ox,Oy,16):MOVE Ax,Ay
  750 DRAW x,y:DRAW Bx,By
  760 PROCcursor(x,y,16)
  770 UNTIL k
  780 IF Ax=Bx AND Ax=x AND Ay=By AND Ay
=y PROCdisp2:ENDPROC
  790 PROCcursor(x,y,16):Cx=x:Cy=y
  800 GCOL 0,3:MOVE Ax,Ay:DRAW Cx,Cy
  810 DRAW Bx,By:DRAW Ax,Ay:cur=d%(0,0)
  820 cur=cur+1:d%(0,cur)=Ax
  830 d%(1,cur)=Ay:d%(2,cur)=Bx
  840 d%(3,cur)=By:d%(4,cur)=Cx
  850 d%(5,cur)=Cy:d%(0,0)=cur
  860 d%(2,0)=x:d%(3,0)=y:ENDPROC
  870 :
  880 DEF FNkeyboard:IF GET=13 =TRUE
  890 IFINKEY-58 y=y+4:IFINKEY-1 y=y+16
  900 IFINKEY-42 y=y-4:IFINKEY-1 y=y-16
  910 IFINKEY-26 x=x-4:IFINKEY-1 x=x-16
  920 IFINKEY-122 x=x+4:IFINKEY-1 x=x+16
  930 IF x<0 x=0
  940 IF x>1279 x=1279
  950 IF y<0 y=0
  960 IF y>980 y=980
  970 *FX21,0
  980 =FALSE
  990 :
 1000 DEF PROCdisp2:LOCAL a,b
 1010 b=d%(0,0):CLG:GCOL0,3
 1020 FOR a=1 TO b:MOVE d%(0,a),d%(1,a)
 1030 DRAW d%(2,a),d%(3,a)
 1040 DRAW d%(4,a),d%(5,a)
 1050 DRAW d%(0,a),d%(1,a)
 1060 NEXT:ENDPROC
 1070 :
 1080 DEF PROCfile
 1090 LOCAL G$
 1100 COLOUR128:PRINTTAB(20,0)" FILE "
 1110 PRINTTAB(10,3)"S)ave OR L)oad ? ";
 1120 G$=CHR$(GET AND 223)
 1130 IF G$="S" PRINT"Save" ELSE IF G$="
L" PRINT"Load" ELSE GOTO 1160
 1140 INPUT'"Enter Filename: ",f$
 1150 PROCload_save(G$,f$)
 1160 PROCdisp2
 1170 ENDPROC
 1180 :
 1190 DEF PROCload_save(d$,f$)
 1200 LOCAL a,b,f
 1210 IF d$="S" f=OPENOUT f$ ELSE f=OPEN
IN f$
 1220 IF d$="L" AND f=0 VDU7:PRINT"File
not found":OSCLI("CAT"):PRINT'"Press spa
ce when ready":REPEATUNTILGET=32:ENDPROC
 1230 FOR b=0 TO 5:IF b=1 THEN 1250
 1240 IF d$="S" PRINT#f,d%(b,0) ELSE INP
UT#f,d%(b,0)
 1250 NEXT
 1260 FOR a=1 TO d%(0,0)
 1270 FOR b=0 TO 5
 1280 IF d$="S" PRINT#f,d%(b,a) ELSE INP
UT#f,d%(b,a)
 1290 NEXT:NEXT:CLOSE#f
 1300 ENDPROC
```

Continued on page 42

An ADFS Directory Deleter

Use this powerful utility by R.A.Smith and A.J.Moore to delete simply and quickly unwanted ADFS directories and their entire contents.

The ADFS allows you to create many directories, which can be extremely useful in the organisation of a disc. Unfortunately, a complex directory structure can be a headache if you want to delete part of it, especially when the 'Dir not empty' error occurs.

The program ZAPDIR is a utility which allows the deletion of a directory and its contents, including any nested sub-directories and files. Enter the program in the normal way, but save the Basic program before running with a name other than ZAPDIR, as this is the name of the machine code file generated by this program.

DELETING DIRECTORIES

To delete a directory including any files and sub-directories use *RUN ZAPDIR (or just *ZAPDIR) followed by the name of the directory to be deleted.

ZAPDIR accepts all the file specifications that are used in the ADFS filing system, as shown below with examples:

```
  @ -    Currently selected directory
         *ZAPDIR @
  ^ -    Parent directory
         *ZAPDIR ^
  . -    A pathname
         *ZAPDIR Fred.Bill
  * -    A wildcard
         *ZAPDIR Fred.B*
  # -    A wildcard
         *ZAPDIR Part#
 :0. -   A drive specification
         *ZAPDIR :0.Utils
    -    Or a mixture, e.g.
         *ZAPDIR ^.Games.*
         *ZAPDIR :1.Games#
```

Note that on a single drive system, ZAPDIR can only be used to zap a different disc by including a quotes (") character (ASCII 34) somewhere in the directory specification. When this is included, then once ZAPDIR has been loaded from disc, the message "Press a key" appears on the screen. Insert the disc which contains the directory to be deleted, and then press any key. ZAPDIR will then function normally with the new disc. The quotes character (") is never used in an actual directory name, and it is ignored by ZAPDIR in specifying directories. Typical uses of ZAPDIR in this situation are:

```
*ZAPDIR "Viewfiles
*ZAPDIR BAS"IC.Utils
*ZAPDIR B*.C*."#RT
```

Once started the only way to abort the operation of ZAPDIR is by pressing Break, but any files and directories already displayed on the screen will have been deleted, and further use of the disc may well result in error messages such as "Broken directory" or "Bad FS map"!

ERROR REPORTING

Certain error states may occur in the execution of the program as listed below.

'ADFS not selected' - An attempt has been made to *ZAPDIR a directory while using another filing system. ADFS must be selected.

'Aborted' - This error is given if any character other than a "Y" is given to answer the 'Go (Y/N)?' prompt.

'Too complex' - This is produced if either the number of nested sub-directories reaches more than 253 or the selected directory specification is longer than given in the 'length' variable (79 in the program).

PROGRAM NOTES

The code resides at &9BE and overwrites the speech, cassette output, RS423 input and Econet workspace. The Econet filing system, like the ADFS, has a hierarchical directory structure, but due to certain assumptions made by the program, like the exact format of data returned from filing system calls, and because it overwrites Econet workspace, it will not function on Econet without modification.

So that any errors made whilst entering the program may be detected, a check is made after assembly of the length of the resulting code. As listed, the program should be exactly &343 bytes long.

WARNING

ZAPPING ADFS DIRECTORIES IS A POWERFUL BUT IRREVERSIBLE PROCESS. TAKE CARE.

```
   10 REM Program ZAPDIR
   20 REM Version B1.04
   30 REM Authors R.A.Smith & A.J.Moore
   40 REM BEEBUG Jan/Feb 1989
   50 REM Program subject to copyright
   60 :
  100 oswrch=&FFEE:osasci=&FFE3
  110 oscli=&FFF7:osbyte=&FFF4
  120 osnewl=&FFE7:osfile=&FFDD
  130 osargs=&FFDA:osgbpb=&FFD1
  140 osrdch=&FFE0:length=79:REM Do NOT
increase size of length
  150 PROCassemble
  160 IF P%>&D00 PRINT'"Please check cod
e as it is too long":END
  170 OSCLI"SAVE ZAPDIR "+STR$~code%+" "
+STR$~P%+" "+STR$~zapdir
  180 END
  190 :
 1000 DEFPROCassemble
 1010 code%=&9BD
 1020 FOR pass%=0 TO 3 STEP 3
 1030 P%=code%
 1040 [
 1050 OPT pass%
 1060 .zapdir
 1070 LDA #0:LDY #0
 1080 JSR osargs:CMP #8
 1090 BEQ adfs:BRK
 1100 EQUB 248:EQUS "ADFS not selected"
 1110 EQUB 0
 1120 .pause
 1130 DEX:INC level
 1140 BEQ retpause
 1150 .error
 1160 BRK:EQUB 145
 1170 EQUS "Too complex":EQUB 0
 1180 .adfs
 1190 LDA #15:JSR oswrch
 1200 LDA #1:LDX #&F2
 1210 LDY #0:JSR osargs
 1220 LDY #0:LDX #4
 1230 .copydirloop
 1240 LDA dir,X:STA dirstore,X
 1250 DEX:BPL copydirloop
 1260 LDX #0
 1270 .storedirloop
 1280 LDA (&F2),Y:CMP #&22
 1290 BEQ pause:STA dirstore+4,X
 1300 .retpause
 1310 INY:INX
 1320 CPX #length-1:BCS error
 1330 CMP #13:BNE storedirloop
 1340 LDA level:BNE missedthat
 1350 JSR print:EQUS "Press a key"+CHR$1
3+"@"
 1360 JSR osrdch:LDX #mount MOD 256
 1370 LDY #mount DIV 256:JSR oscli
 1380 .missedthat
 1390 LDX #dirstore MOD 256:LDY #dirstor
e DIV 256
 1400 JSR oscli:LDA #139
 1410 LDX #1:LDY #0:STY level
 1420 JSR osbyte
 1430 JSR directoryname:JSR print:EQUS "
Selecting   :@"
 1440 JSR printname:JSR osnewl
 1450 JSR print:EQUS "Go (Y/N)?@"
 1460 JSR osrdch:JSR osasci
 1470 CMP #ASC("Y"):BEQ ok
 1480 BRK:EQUB 146
 1490 EQUS CHR$10+"Aborted":EQUB 0
 1500 .ok
 1510 JSR osnewl
 1520 LDX #14
 1530 .copyloop
 1540 LDA dirname,X:STA dirstore+1,X
 1550 DEX:CPX #2
 1560 BNE copyloop:LDA #13
 1570 STA dirstore+14:JSR access
 1580 CMP #0:BEQ jmpup
 1590 .mainloop
 1600 JSR getfilename:BCS jmpup
 1610 JSR isdir:CMP #2
 1620 BNE remove:JMP selectdir
 1630 .remove
 1640 JSR print:EQUS "   Removing :@"
 1650 JSR printfile:JSR deletefile
 1660 JMP mainloop
 1670 .selectdir
 1680 INC level:LDA level
 1690 CMP #254:BCS jerror
 1700 LDX #dirstar MOD 256:LDY #dirstar
DIV 256
 1710 JSR oscli:JSR access
 1720 CMP #0:BEQ jmpup
 1730 JMP mainloop
 1740 .jerror
 1750 JMP error
 1760 .jmpup
 1770 DEC level:LDA level
 1780 CMP #&FF:BEQ startdir
 1790 LDX #moveup MOD 256:LDY #moveup DI
V 256
 1800 JSR oscli:JSR getfilename
 1810 JSR deletefile:JSR print:EQUS "Zap
ping dir :@"
 1820 JSR printfile:JMP mainloop
 1830 .startdir
 1840 JSR directoryname:LDA #ASC(":")
 1850 JSR oswrch:JSR printname
 1860 JSR print:EQUS " now zapped"+CHR$1
3+"@"
 1870 LDX dirname:INX
 1880 INX:LDA dirname,X
 1890 CMP #ASC("$"):BEQ root
 1900 LDX #moveup MOD 256:LDY #moveup DI
```

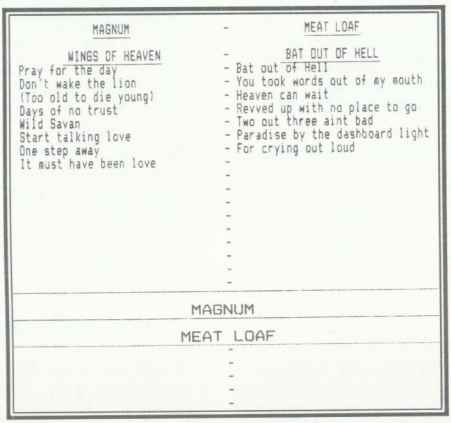
```
V 256
 1910 JSR oscli:LDA #(dirstore+4) MOD 25
6
 1920 STA fileblock:LDA #(dirstore+4) DI
V 256
 1930 STA fileblock+1:LDA #0:STA fileblo
ck+14
 1940 LDA #4:LDX #fileblock MOD 256
 1950 LDY #fileblock DIV 256:JSR osfile
 1960 LDA #6:JSR osfile
 1970 .root
 1980 RTS
 1990 .mount
 2000 EQUS "MOUNT":EQUB 13
 2010 .dir
 2020 EQUS "DIR "
 2030 .moveup
 2040 EQUS "DIR ^":EQUB 13
 2050 .dirstar
 2060 EQUS "DIR *":EQUB 13
 2070 .access
 2080 LDA #star MOD 256:STA fileblock
 2090 LDA #star DIV 256:STA fileblock+1
 2100 LDA #5:LDX #fileblock MOD 256
 2110 LDY #fileblock DIV 256:JSR osfile
 2120 CMP #0:BEQ accessrts
 2130 LDX #accessall MOD 256:LDY #access
all DIV 256
 2140 JSR oscli:LDA #1
 2150 .accessrts
 2160 RTS
 2170 .accessall
 2180 EQUS "ACCESS "
 2190 .star
 2200 EQUS "*":EQUB 13
 2210 .getfilename
 2220 JSR storezero:LDA #filename MOD 25
6
 2230 STA gbpbblock+1:LDA #filename DIV
256
 2240 STA gbpbblock+2:LDA #1
 2250 STA gbpbblock+5:LDA #8
 2260 LDX #gbpbblock MOD 256:LDY #gbpbbl
ock DIV 256
 2270 JSR osgbpb:PHP
 2280 LDX filename:INX
 2290 LDA #13:STA filename,X
 2300 PLP:RTS
 2310 .isdir
 2320 LDA #(filename+1) MOD 256:STA file
block
 2330 LDA #(filename+1) DIV 256:STA file
block+1
 2340 LDA #5:LDX #fileblock MOD 256
 2350 LDY #fileblock DIV 256:JSR osfile
 2360 RTS
 2370 .deletefile
 2380 LDA #(filename+1) MOD 256:STA file
block
 2390 LDA #(filename+1) DIV 256:STA file
block+1
 2400 LDA #6:LDX #fileblock MOD 256
 2410 LDY #fileblock DIV 256:JSR osfile
 2420 RTS
 2430 .directoryname
 2440 JSR storezero:LDA #dirname MOD 256
 2450 STA gbpbblock+1:LDA #dirname DIV 2
56
 2460 STA gbpbblock+2:LDA #6
 2470 LDX #gbpbblock MOD 256:LDY #gbpbbl
ock DIV 256
 2480 JSR osgbpb:RTS
 2490 .printname
 2500 LDX #1:LDA dirname,X
 2510 JSR oswrch:LDA #ASC(".")
 2520 JSR oswrch:INX
 2530 INX
 2540 .fname
 2550 LDA dirname,X:JSR oswrch
 2560 INX:CMP #32
 2570 BNE fname:RTS
 2580 .storezero
 2590 LDX#13:LDA #0
 2600 .storezeroloop
 2610 STA gbpbblock,X:DEX
 2620 BPL storezeroloop:RTS
 2630 .print
 2640 LDY #0:PLA:STA &AA
 2650 PLA:STA &AB
 2660 INC &AA:BNE printloop
 2670 INC &AB
 2680 .printloop
 2690 LDA (&AA),Y:CMP #64
 2700 BEQ endprint:JSR osasci
 2710 INC &AA:BNE printloop
 2720 INC &AB:BNE printloop
 2730 .endprint
 2740 INC &AA:BNE endprint2
 2750 INC &AB
 2760 .endprint2
 2770 JMP (&AA)
 2780 .printfile
 2790 LDX #1
 2800 .printfileloop
 2810 LDA filename,X:JSR osasci
 2820 INX:CMP #13
 2830 BNE printfileloop:RTS
 2840 .dirstore
 2850 EQUS STRING$(length+5,CHR$0)
 2860 .level
 2870 EQUB 255
 2880 .gbpbblock
 2890 EQUS STRING$(14,CHR$0)
 2900 .fileblock
 2910 EQUS STRING$(19,CHR$0)
 2920 .dirname
 2930 EQUS STRING$(14,CHR$0)
 2940 .filename
 2950 EQUS STRING$(15,CHR$0)
 2960 ]
 2970 NEXT
 2980 ENDPROC
```

Producing Audio Tape Inlays

Use your computer with this application from Paul Timson to edit and print cassette labels for audio tapes.

From the discipline of making backups of programs to avoid the disaster of losing a program through accidental erasure or bad disc, etc. I decided to extend this idea to my record collection. Transferring the albums to tape gives me the advantage of maintaining the original album in a 'once played' condition avoiding the almost inevitable scratches through regular use and the convenience of using the tape copy. Having only the copy at risk is also more important when the original is a tape version of the album, which is invariably of relatively poor quality in both tape and mechanism, and is then at the mercy of the unpredictably voracious appetite of the tape player.

With all blank tapes comes the need to fill in the tracks on the inverted tape inlay. This program allows the user to produce attractive and readable tape inlays for either one or two albums recorded on the same tape.



A sample tape inlay

THE TAPE INLAY PRODUCER

There are three programs listed here. The first is the Menu program, the second the Tape Inlay Editor, and finally the Tape Inlay Printer routine. The magazine disc/tape also contains a fourth program (called LOADER) which can be used to generate a Help text file for use with the editor. This is not essential to using the other programs.

TYPING IN THE PROGRAMS

Type in and save the menu program (using the name MENU). Next type in the editor program, being careful to exclude unnecessary spaces as memory space is at a premium, and save as EDIT. Finally, type in the printer program and save as PRINT. When using the system always start by chaining MENU.

MENU

The Tape Inlay Producer is supported by a menu which displays the following three options:

1. Edit new or existing tape inlay
2. Print out tape inlay
3. Exit tape inlay producer

The menu program allows access to the editor and the printer routines, and the program can be also be terminated at this point. These options are selected by pressing the appropriate number key.

EDITOR

The Editor allows tape inlays to be loaded, entered, edited and subsequently saved for printing or further editing.

The editor initially displays a tape inlay, which is twenty three lines deep and two columns of thirty three characters wide. It also marks the division between the main list, and the titles which are folded round the back of the cassette. This is where the tunes/games that appear on the tape are entered. Below this are two boxes labelled Title 1 and Title 2, where the album title(s) are entered. Selection of either one or two titles on the tape allows for a general heading for the whole tape or one for each side, the tape titles being printed on the 'spine'. On the inlay, track titles can be inserted, deleted, copied, moved and edited. See table 1 for a complete list.

When saving and loading tape inlays, a disc catalogue is displayed to aid filename selection. At any time, information on the commands can be called up by pressing the 'H' key (i.e. Help),

provided the Help file has been created (with the Loader program supplied only on the magazine disc/tape), and is accessible on disc. Delete line 1640 to completely disable the Help facility.

When editing lines (i.e. not titles) a dash (-) typed before the item means that it will be centred and underlined when printed out. When copying or moving a cell - Press the 'C' or 'M' key on the cell to be copied/moved, and then move the cursor to the destination and press Return to complete the operation, or the space bar to exit.

E	- Edit line occupied by cursor
1	- Edit title number 1
2	- Edit title number 2
T	- Alternate between 1 and 2 titles
I	- Insert a blank line at the cursor
D	- Delete line at cursor
C	- Copy line to another position
M	- Move line to a new position
L	- Load previous tape inlay
S	- Save current tape inlay
*	- Catalogue tape/disc
Z	- Zap tape inlay (i.e. clear all titles)
X	- Exit program
Cursor keys	- Move cursor
<Delete>	- Delete left
<Copy>	- Delete right

Table 1. Editor command list

PRINTER

The printer program prints out tape inlays which have been created and saved with the editor. A disc catalogue is displayed to aid filename selection.

The printer codes used are for EPSON compatible printers, but can easily be modified to suit other printers by altering the variables in DEF PROCinit.

These variables are used as follows:

dc$ - Double-strike, condensed mode
de$ - Double-strike, enlarged mode
oe$ - One-eighth inch line-space
ul$ - Underline mode on
xul$ - Underline mode off
init$ - Initialise printer
bell$ - Sound printer bell

```
   10 REM Program Tape Inlay Menu
   20 REM Version B1.1
   30 REM Author  Paul Timson
   40 REM BEEBUG  Jan/Feb 1989
   50 REM Program Subject To Copyright
   60 :
  100 IF PAGE>&1300 THEN PAGE=&1300:CHAI
N"MENU"
  110 ON ERROR GOTO 120
  120 MODE7
  130 *FX 4,2
  140 FOR count%=0 TO 1
  150 PRINTTAB(9,count%);CHR$(141);CHR$(
129);"TAPE INLAY PRODUCER"
  160 NEXT count%
  170 PRINTTAB(12,3);CHR$(129);"By Paul
Timson"
  180 PRINTTAB(0,8);"1. Edit New Or Exis
ting Tape Inlay."
  190 PRINTTAB(0,13);"2. Print Out Tape
Inlay."
  200 PRINTTAB(0,18);"3. Exit Tape Inlay
 Producer."
  210 PRINTTAB(0,24);"Please Select Opti
on (1 - 3). ";
  220 temp%=GET-48
  230 IF temp%<1 OR temp%>3 GOTO 220
  240 PRINTSTR$(temp%);
  250 IF temp%=1 CHAIN"EDIT"
  260 IF temp%=2 CHAIN"PRINT"
  270 MODE7:*FX4
  280 END
  290 :
  300 IF ERR=17 GOTO120
  310 MODE7:REPORT:PRINT" at line ";ERL
  320 END
```

```
   10 REM Program Tape Inlay Editor
   20 REM Version B1.2
   30 REM Author  Paul Timson
   40 REM BEEBUG  Jan/Feb 1989
   50 REM Program Subject To Copyright
   60 :
  100 ONERRORCLOSE#0:CLS:GOTO130
  110 MODE0
  120 PROCinit
  130 PROCtape:REPEAT
  140 PRINTTAB(48,26);"Function (H is He
lp) :-";SPC(10);TAB(72,26);
  150 PROCcom(1)
  160 UNTILtem%>=15ANDtem%<=18
  170 PRINTTAB(48,26);SPC(23);TAB(48,26)
;"Are You Sure? ";
  180 tem$=GET$
  190 IFINSTR("YyNn",tem$)=0GOTO180
  200 PRINTtem$
  210 IFtem$="N"ORtem$="n"GOTO130
  220 IFtem%>16CLEAR:GOTO120
```

```
  230 CHAIN"MENU"
  240 END
  250 :
 1000 DEFPROCinit
 1010 DIMtr$(46)
 1020 t1$="":t2$=""
 1030 last$=""
 1040 *FX 4,1
 1050 *FX 11
 1060 VDU23,1,0;0;0;0;
 1070 pos%=1
 1080 t%=1
 1090 ENDPROC
 1100 :
 1110 DEFPROCtape
 1120 PRINTTAB(22,0);"Tape Inlay Produce
r - By Paul Timson"
 1130 PRINTTAB(5);"+";STRING$(33,"-");"+
";STRING$(33,"-");"+"
 1140 FORad%=1TO23:PROCline(ad%):NEXTad%
 1150 PRINTTAB(5);"+";STRING$(33,"-");"+
";STRING$(6,"-");"+";STRING$(26,"-");"+"
 1160 PRINTTAB(5);"|Title 1="+t1$+STRING
$(32-LEN(t1$)," ")+"|"
 1170 IFt%=1PRINTTAB(5);"|Title 2=Not In
 Use";STRING$(22," ")+"|"ELSEPRINTTAB(5)
;"|Title 2="+t2$+STRING$(32-LEN(t2$)," "
)+"|"
 1180 PRINTTAB(5);"+";STRING$(40,"-");"+
"
 1190 ENDPROC
 1200 :
 1210 DEFPROCline(ad%)
 1220 IFad%+23>46ad%=ad%-23
 1230 PRINTTAB(4,ad%+1);" |";tr$(ad%);ST
RING$(33-LEN(tr$(ad%))," ");"|";tr$(ad%+
23);STRING$(33-LEN(tr$(ad%+23))," ");"|
"
 1240 IFpos%=ad%PRINTTAB(4,ad%+1);"*"ELS
EIFpos%=ad%+23PRINTTAB(74,ad%+1);"*"
 1250 ENDPROC
 1260 :
 1270 DEFPROClist
 1280 PRINTTAB(5,30);"Enter song/game ti
tle :- ";
 1290 y%=pos%+1:IFpos%>23y%=pos%-22
 1300 x%=6:IFpos%>23x%=40
 1310 var$=tr$(pos%)
 1320 PROCalt(x%,y%,33,1)
 1330 tr$(pos%)=var$
 1340 ENDPROC
 1350 :
 1360 DEFPROCcopy
 1370 from%=pos%
 1380 PRINTTAB(5,30);"Move cursor to des
tination and press <Return> - <Space Bar
> to Exit.";
 1390 PROCcom(2)
 1400 PROCline(pos%)
 1410 IFtem$=CHR$(32)GOTO1440
 1420 tem$=tr$(pos%)
 1430 tr$(pos%)=tr$(from%)
 1440 ENDPROC
 1450 :
 1460 DEFPROCins
 1470 IF(pos%<24ANDtr$(23)<>"")=-1OR(pos
%>23ANDtr$(46)<>"")=-1ORpos%=23ORpos%=46
GOTO1490
 1480 IFpos%<24FORad%=23TOpos%+1STEP-1:t
r$(ad%)=tr$(ad%-1):NEXTad%:tr$(pos%)=""E
LSEFORad%=46TOpos%+1STEP-1:tr$(ad%)=tr$(
ad%-1):NEXTad%:tr$(pos%)=""
 1490 ENDPROC
 1500 :
 1510 DEFPROCdel
 1520 IFpos%<23FORad%=pos%+1TO23:tr$(ad%
-1)=tr$(ad%):NEXTad%ELSEIFpos%<45ANDpos%
>23FORad%=pos%+1TO46:tr$(ad%-1)=tr$(ad%)
:NEXTad%
 1530 IFpos%<24tr$(23)=""ELSEtr$(46)=""
 1540 ENDPROC
 1550 :
 1560 DEFPROCmove
 1570 PROCcopy
 1580 IFpos%=from%GOTO1630
 1590 IF(from%<24ANDpos%>23ANDtr$(46)<>"
")=-1OR(from%>23ANDpos%<24ANDtr$(23)<>""
)=-1tr$(pos%)=tem$:GOTO1630
 1600 typ%=pos%:pos%=from%:PROCdel:pos%=
typ%
 1610 pos%=pos%+(from%<typ%AND(typ%>23AN
Dfrom%<24)=0)
 1620 pos%=pos%+1:PROCins:tr$(pos%)=tem$
:pos%=pos%-(from%<typ%AND(typ%>23ANDfrom
%<24)=0)-1
 1630 ENDPROC
 1640 :
 1650 DEFPROCcom(T%)
 1660 tem$=GET$
 1670 tem%=ASC(tem$):IFtem%>=136ANDtem%<
=139GOTO1720
 1680 IFT%=2AND(tem$=CHR$(13)ORtem$=CHR$
(32))=-1tem%=0:GOTO1840
 1690 IFT%=2GOTO1660
 1700 tem%=INSTR("HhLlSsIiDdCcEeXxZz**Mm
Tt1122",tem$):IFtem%=0GOTO1660
 1710 IF(tem%/2)=INT(tem%/2)tem$=CHR$(AS
C(tem$)-32)
 1720 IFT%=2GOTO1740
 1730 IFtem%=139PRINT"Up"ELSEIFtem%=138P
RINT"Down"ELSEIFtem%=136PRINT"Left"ELSEI
Ftem%=137PRINT"Right"
 1740 IFtem%<136GOTO1780
 1750 A%=pos%:pos%=0:PROCline(A%):pos%=A
%
 1760 IFtem%=139ANDpos%<>1ANDpos%<>24pos
```

```
%=pos%-1ELSEIFtem%=138ANDpos%<>23ANDpos%
<>46pos%=pos%+1ELSEIFtem%=136ANDpos%>23p
os%=pos%-23ELSEIFtem%=137ANDpos%<24pos%=
pos%+23
 1770 PROCline(pos%):GOTO1830
 1780 IFtem$="E"PRINT"Edit":PROClist ELS
EIFtem$="S"PROCfile(1) ELSEIFtem$="L"PRO
Cfile(2) ELSEIFtem$="*"PROCcat ELSEIFtem
$="H"PROChelp ELSEIFtem$="I"PRINT"Insert
":PROCins
 1790 IFtem$="D"PRINT"Delete":PROCdel EL
SEIFtem$="C"PRINT"Copy":PROCcopy ELSEIFt
em$="M"PRINT"Move":PROCmove ELSEIFtem$="
T"PRINT"Titles":PROCtitles ELSEIFtem$="1
"PRINT"Edit 1":PROCed(1) ELSEIFtem$="2"P
RINT"Edit 2":PROCed(2)
 1800 IFtem$="X"PRINT"Exit":GOTO1840ELSE
IFtem$="Z"PRINT"Zap":GOTO1840
 1810 tem%=INT((tem%-1)/2):IFtem%<>6ANDt
em%<>12ANDtem%<>13PROCtape
 1820 PRINTTAB(5,30);STRING$(70," ");
 1830 IFT%=2GOTO1660
 1840 ENDPROC
 1850 :
 1860 DEFPROCalt(x%,y%,max%,typ%)
 1870 *FX 12
 1880 VDU23,1,1;0;0;0;
 1890 at%=1:ad%=LEN(var$):IFad%>1at%=ad%
+1
 1900 ad%=LEN(var$):IFat%=max%+1OR(at%=m
ax%ANDtyp%=1ANDLEFT$(var$,1)<>"-")at%=at
%-1
 1910 PRINTTAB(x%,y%);SPC(max%);TAB(x%,y
%);var$:PRINTTAB(x%+at%-1,y%);
 1920 tem$=GET$
 1930 IFtem$=CHR$(13)GOTO2030
 1940 IFtem$=CHR$(137)ANDat%<ad%+1at%=at
%+1:GOTO2020
 1950 IFtem$=CHR$(136)ANDat%<>1at%=at%-1
:GOTO2020
 1960 IFtem$=CHR$(127)ANDat%=1GOTO1920
 1970 IFtem$=CHR$(135)ANDat%>ad%GOTO1920
 1980 IFtem$=CHR$(135)var$=LEFT$(var$,at
%-1)+RIGHT$(var$,LEN(var$)-at%):GOTO2020
 1990 IFtem$=CHR$(127)var$=LEFT$(var$,at
%-2)+RIGHT$(var$,LEN(var$)-at%+1):at%=at
%-1:GOTO2020
 2000 IFad%=max%OR(ad%=max%-1ANDtyp%=1AN
D((LEFT$(var$,1)<>"-"ANDat%<>1)=-1OR(LEF
T$(var$,1)<>"-"ANDat%=1ANDtem$<>"-")=-1)
=-1)=-1GOTO1920
 2010 IFtem$>CHR$(31)ANDtem$<CHR$(128)va
r$=LEFT$(var$,at%-1)+tem$+RIGHT$(var$,LE
N(var$)-at%+1):at%=at%+1
 2020 GOTO1900
 2030 VDU23,1,0;0;0;0;
 2040 *FX 11
 2050 ENDPROC
 2060 :
 2070 DEFPROCtitles
 2080 IFt%=1t%=2ELSEt%=1
 2090 ENDPROC
 2100 :
 2110 DEFPROCed(typ%)
 2120 PRINTTAB(5,30);"Enter cassette tit
le ";typ%;":- ";:IFtyp%=2t%=2
 2130 IFtyp%=1var$=t1$ELSEvar$=t2$
 2140 PROCalt(14,25+typ%,32,2)
 2150 IFtyp%=1t1$=var$ELSEt2$=var$
 2160 ENDPROC
 2170 :
 2180 DEFPROCfile(typ%)
 2190 CLS:IFtyp%=1PRINTTAB(33,0);"SAVE";
ELSEPRINTTAB(33,0);"LOAD";
 2200 PRINT" TAPE INLAY";TAB(33,1);"----
----------"'
 2210 *CAT
 2220 PRINTTAB(0,26);"Enter file name :-
"
 2230 var$=last$
 2240 PRINT'">";:PROCalt(1,28,7,2)
 2250 last$=var$
 2260 IFtyp%=1X%=OPENOUT(last$)ELSEX%=OP
ENIN(last$)
 2270 FORad%=1TO46
 2280 IFtyp%=1PRINT#X%,tr$(ad%)ELSEINPUT
#X%,tr$(ad%)
 2290 NEXTad%
 2300 IFtyp%=1PRINT#X%,t1$,t2$,t%,pos%EL
SEINPUT#X%,t1$,t2$,t%,pos%
 2310 CLOSE#X%
 2320 CLS
 2330 ENDPROC
 2340 :
 2350 DEFPROChelp
 2360 CLS
 2370 X%=OPENIN("HELPSHE")
 2380 FORtyp%=1TO32
 2390 INPUT#X%,tem$
 2400 IFtyp%<32PRINTtem$ELSEPRINTtem$;
 2410 NEXTtyp%
 2420 CLOSE#X%
 2430 tem$=GET$
 2440 CLS
 2450 ENDPROC
 2460 :
 2470 DEFPROCcat
 2480 CLS
 2490 PRINTTAB(33,0);"CATALOGUE DISC";TA
B(33,1);STRING$(14,"-")'
 2500 *CAT
 2510 PRINTTAB(0,30);"PRESS ANY KEY"'STR
ING$(13,"-");
 2520 tem$=GET$
 2530 CLS
 2540 ENDPROC
```

```
  10 REM Program Tape Inlay Print
  20 REM Version B1.2
  30 REM Author  Paul Timson
  40 REM BEEBUG  Jan/Feb 1989
  50 REM Program Subject To Copyright
  60 :
 100 ON ERROR PROCerror:IF pass%=1 GOTO
230 ELSE GOTO 130
 110 MODE0
 120 PROCinit
 130 REPEAT
 140 CLS
 150 PRINTTAB(32,0);"PRINT TAPE INLAY"'
TAB(32,1);STRING$(16,"-")'
 160 *CAT
 170 PRINTTAB(2,26);"Enter Tape Inlay T
o Print - <Escape> Exits"
 180 INPUT'">"last$
 190 *FX 14,6
 200 PROCfile:PROCprint
 210 *FX 13,6
 220 UNTIL FALSE
 230 PRINTTAB(0,30);"Are You Sure? ";
 240 temp$=GET$
 250 IFINSTR("YyNn",temp$)=0GOTO240
 260 PRINTtemp$
 270 IFtemp$="N"ORtemp$="n"GOTO130
 280 CHAIN"MENU"
 290 END
 300 :
1000 DEFPROCinit
1010 DIMlist$(46)
1020 t1$="":t2$=""
1030 last$=""
1040 dc$=CHR$(1)+CHR$(27)+CHR$(1)+CHR$(
33)+CHR$(1)+CHR$(20)
1050 oe$=CHR$(1)+CHR$(27)+CHR$(1)+CHR$(
48)
1060 ul$=CHR$(1)+CHR$(27)+CHR$(1)+CHR$(
45)+CHR$(1)+CHR$(49)
1070 de$=CHR$(1)+CHR$(27)+CHR$(1)+CHR$(
33)+CHR$(1)+CHR$(30)
1080 xul$=CHR$(1)+CHR$(27)+CHR$(1)+CHR$
(45)+CHR$(1)+CHR$(48)
1090 init$=CHR$(1)+CHR$(27)+CHR$(1)+CHR
$(64)
1100 bell$=CHR$(1)+CHR$(7)
1110 *FX 4
1120 *FX 11
1130 VDU23,1,0;0;0;0;
1140 t%=1
1150 pass%=0
1160 ENDPROC
1170 :
1180 DEFPROCprint
1190 CLS:VDU2
1200 PRINTinit$;dc$;oe$;"+";STRING$(68,
"-");"+"
1210 IFt%=1PRINTde$;" ";SPC(INT((39-LEN
(t1$))/2));ul$;t1$;xul$;SPC(39-LEN(t1$)-
INT((39-LEN(t1$))/2));" "
1220 IFt%=2PRINT" ";SPC(INT((33-LEN(t1$
))/2));ul$;t1$;xul$;SPC(33-LEN(t1$)-INT(
(33-LEN(t1$))/2));" - ";SPC(INT((32-LEN(
t2$))/2));ul$;t2$;xul$;SPC(32-LEN(t2$)-I
NT((32-LEN(t2$))/2));" "
1230 st%=1:fin%=18:PROCindex
1240 PRINTxul$;"|";STRING$(68," ");"|"
1250 IFt%=2GOTO1300
1260 PRINTde$;" ";SPC(INT((39-LEN(t1$))
/2));ul$;t1$;xul$;SPC(39-LEN(t1$)-INT((3
9-LEN(t1$))/2));" "
1270 PRINTdc$;"|";STRING$(68," ");"|"
1280 PRINTul$;"|";STRING$(68," ");"|"
1290 GOTO1330
1300 PRINTde$;ul$;" ";SPC(INT((39-LEN(t
1$))/2));t1$;SPC(39-LEN(t1$)-INT((39-LEN
(t1$))/2));" ";xul$
1310 PRINTdc$;"|";STRING$(68," ");"|"
1320 PRINTde$;ul$;" ";SPC(INT((39-LEN(t
2$))/2));t2$;SPC(39-LEN(t2$)-INT((39-LEN
(t2$))/2));" "
1330 st%=19:fin%=23:PROCindex
1340 PRINTbell$;init$
1350 VDU3
1360 ENDPROC
1370 :
1380 DEFPROCindex
1390 PRINTdc$;xul$;
1400 IFst%=1PRINT"|";STRING$(68," ");"|
"
1410 FORad%=st%TOfin%
1420 IFad%=18ORad%=23PRINTul$;
1430 FORtemp%=ad%TOad%+23STEP23
1440 IFLEFT$(list$(temp%),1)="-"temp$="
"+RIGHT$(list$(temp%),LEN(list$(temp%))
-1):temp$=STRING$(INT((33-LEN(temp$))/2)
," ")+ul$+temp$+xul$+STRING$(33-LEN(temp
$)-INT((33-LEN(temp$))/2)," ")
1450 IFLEFT$(list$(temp%),1)<>"-"temp$=
" "+list$(temp%)+STRING$(33-LEN(list$(te
mp%))-1," ")
1460 IFtemp%<24one$=temp$ELSEtwo$=temp$
1470 NEXTtemp%
1480 PRINT"|";one$;" -";two$;"|"
1490 NEXTad%
1500 ENDPROC
1510 :
1520 DEFPROCfile
1530 X%=OPENIN(last$)
1540 FORad%=1TO46
1550 INPUT#X%,list$(ad%)
1560 NEXTad%
1570 INPUT#X%,t1$,t2$,t%,pos%
1580 CLOSE#X%
1590 ENDPROC
1600 :
1610 DEFPROCerror
1620 CLOSE #0
1630 VDU3
1640 *FX 13,6
1650 IF ERR=17 pass%=1
1660 ENDPROC
```

Mini Office II: A Dabhand Guide

Reviewed by Peter Rochford

Title	**Mini Office II: A Dabhand Guide**
Publisher	**Dabs Press** **5 Victoria Lane, Whitefield,** **Manchester M25 6AL.** **Tel. 061-766 8423**
Price	**£9.95 (book)** **£7.95 (program disc)**

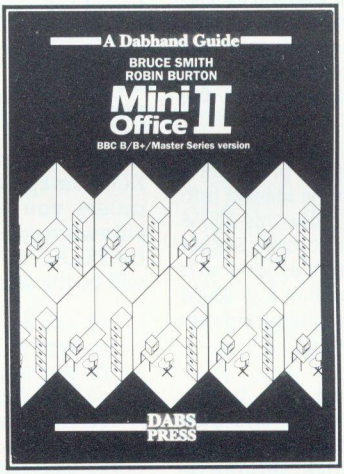
It is a number of years now since Database Publications first released their budget-priced business software package called Mini Office. Originally written for the Beeb, it has now been converted for many other computers including the PC, and has received wide acclaim.

The latest version, Mini Office II, is indeed a comprehensive package, and a remarkably powerful one considering the low price (see review in BEEBUG Vol.5 No.8). This has made it a consistent best-seller.

The original documentation supplied with the package is in the form of a small ring-bound manual of some 98 pages. It is quite a reasonable guide, but certainly not comprehensive, and does not do full justice to the software. Neither does the manual serve well the needs of first time users of computers who would, I expect, make up a large proportion of purchasers of the Mini Office II.

Dabs Press has published some excellent guides for several of the popular Beeb software packages, and has now produced its own guide to Mini Office II for those who want more information. The book, written by Bruce Smith and Robin Burton, consists of some 250 pages with a very good index, and is realistically priced at £9.95.

It has separate sections on the use of each of the modules in Mini Office along with worked examples, and each section concludes with a chapter of useful hints and tips.



The layout and presentation of the book are good. It is well written in an informal and chatty style that I am sure will appeal to most people.

Dabs has rightly assumed that many purchasers of Mini Office will be first-time users of such software, so a good introduction has been included in the book, and each section begins with a clear and easy-to-follow explanation of what this part of the package can do for you.

The sections on labelling and the use of printers deserve special mention as they are really excellent, yet it is often areas like these that cause problems for so many in using packages like Mini Office II. There is also a chapter on the use of Dabs' own MiniDriver printer ROM, which is written for Mini Office and thus takes the pain out of getting good printed results from your work.

There is a programs disc available to go with the book priced at £7.95 which contains all the worked examples referred to in the text, and a number of utilities to enhance the package. One utility worthy of mention allows the sideways printing of spreadsheets, overcoming the problem of printers limited to 80 columns only.

CONCLUSION

This is another excellent book indeed from Dabs Press. It has much to commend it in terms of both content and presentation. I am sure it will be a most useful buy for newcomers to the Mini Office II package, and for those who are already serious users and want to get even more from it. Recommended. B

A Versatile ROM Manager

Model B owners can now manage the ROMs in their machine with four new star commands implemented with Bernard Hill's latest program.

With the general acceptance of BEEBUG's optional prefix notation (W for Watford, B for Beebug, Z for solidisk, etc) for sideways ROM commands, command clashes are not as serious a problem as they were in the early days of the model B. However, *workspace* clashes are becoming something of an annoyance.

If all my ROMs were enabled in my model B, then PAGE would be at &2300 since many claim dynamic workspace - not much room for programs! Now with this handy utility you can *UNPLUG your ADFS ROM, GXR ROM and any others which are causing unnecessary raising of OSHWM. Of course, when required, the ROMs can be reINSERTed, just as in the Master.

The program listed with this article produces a sideways ROM image which can be loaded into sideways RAM as required. Ideally this ROM image should be in the highest priority socket (as on the ATPL board), but is fine as long as it sits in a socket of higher priority than any ROM you wish to disable, and which claims private workspace (such as ADFS or GXR).

First type in the program (and save before running - use the name SMANGER), and then run. You will find that it produces a file called MANAGER on your disc which should be loaded to sideways RAM with the appropriate command (see the instructions supplied with your sideways RAM).

COMMANDS

Once initialised with Break the new 'ROM' responds to four commands:

```
*INSERT <ROM number>
*UNPLUG <ROM number>
*ROMS
*POWERUP
```

The first two commands do the main work and behave just as in the Master, except that the ROM number should be expressed in Hex (e.g. *UNPLUG F, not *UNPLUG 15). You do not need to press Break after inserting or unplugging, but you should note that the program will give a "Bad ROM Number" message if you try to unplug the RAM bank which contains it!

The command *ROMS gives a list of ROMs in exactly the same style as the Master (complete with Unplug status) and *POWERUP is a routine which simulates a power-on reset on the system. All commands may be abbreviated with a full stop, e.g. *POW.

The program works by intercepting service call 1, and before passing it on, checking a table of 16 ROM status bits (held in 2 bytes) clearing the relevant byte of the ROM type table at &2A1 if the ROM has 'unplug' status. Thus the ROMs remain UNPLUGged through Break, Ctrl-Break and even the simulated *POWERUP, but of course a real power-down will cause loss of the ROM image and so loss of unplug status - unless you have battery-backing for your sideways RAM, in which case your UNPLUG/INSERT status is permanent, and will be the same when you switch back on.

```
 10 REM Program SMANAGER
 20 REM Version B1.0
 30 REM Author  Bernard Hill
 40 REM BEEBUG  Jan/Feb 1989
 50 REM Program subject to copyright
 60 :
100 MODE7:HIMEM=&7800
110 mask=&A8:flag=&A9:temp=&AA
120 :
130 FOR opt=4 TO 7 STEP 3
140 P%=&8000:O%=HIMEM:Q%=O%
150 [ OPT opt
160 BRK:BRK:BRK
170 JMP service
180 EQUB &82
190 EQUB copyright
200 EQUB 1
210 EQUS "Manager ROM"
220 .copyright
230 EQUB 0
240 EQUS "(C) BEEBUG, 1989"
250 EQUB 0
260 .byte1 EQUB 0
270 .byte2 EQUB 0
280 .service
290 CMP #1:BEQ service1
```

```
300 CMP #4:BEQ service4:RTS
310 .service1 TYA:PHA
320 LDX #&80:STX mask:LDY #&F
330 .lp LDA byte1:AND mask:BEQ endlp1
340 LDA #0:STA &2A1,Y
350 .endlp1 CLC:ROR mask:DEY:CPY #7
360 BNE lp:LDX #&80:STX mask
370 .lp LDA byte2:AND mask:BEQ endlp2
380 LDA #0:STA &2A1,Y
390 .endlp2 CLC:ROR mask:DEY:BPL lp
400 PLA:TAY:LDA #1:LDX &F4:RTS
410 .service4
420 TYA:PHA:LDX #0
430 .ncmd PLA:TAY:PHA:DEX:DEY
440 .nch INX:INY:LDA(&F2),Y:CMP #&61
450 BCC noc:CMP #&7B:BCS noc:AND #&DF
460 .noc EOR cmds,X:BEQ nch:BPL on
470 LDA (&F2),Y:BEQ do
480 CMP #13:BEQ do:CMP #32:BEQ do
490 .on CPX #0:BEQ x:LDA (&F2),Y
500 CMP #&2E:BEQ skip
510 .x LDA cmds,X:BMI ov:INX:JMP x
520 .ov INX:INX:LDA cmds,X:BNE ncmd
530 PLA:TAY:LDX &F4:LDA #4:RTS
540 .skip INX:LDA cmds,X:BPL skip
550 .do LDA cmds,X:PHA
560 LDA cmds+1,X:PHA:RTS
570 .insert LDA #&80:STA flag:BNE doc
580 .unplug LDA #0:STA flag
590 .doc LDA #1:STA mask:JSR rm
600 LDA flag:BMI set0:STA &2A1,X
610 BEQ domask
620 .set0 LDA #6:JSR rombyt
630 .set0 LDA #6:JSR rombyt:STA &2A1,X
640 .domask LDY #0:CPX #8:BCC less8
650 .lp CPX #8:BEQ fin1:ASL mask
660 DEX:BPL lp
670 .fin1 LDA mask:BIT flag:BPL set1
680 EOR #&FF:AND byte1:JMP f1
690 .set1 ORA byte1
700 .f1 STA byte1:JMP fincmd1
710 .less8 CPX #0:BEQ fin2:ASL mask
720 DEX:BPL less8
730 .fin2 LDA mask:BIT flag:BPL set2
740 EOR #&FF:AND byte2:JMP f2
750 .set2 ORA byte2
760 .f2 STA byte2
770 .fincmd1 LDX &F4:PLA:TAY
780 LDA #0:RTS
790 .rombyt STA &F6:STY temp
800 TXA:PHA:TAY:LDA #&80:STA &F7
810 JSR &FFB9:TAY:PLA:TAX:TYA:LDY temp
820 RTS
830 .roms LDX #15
840 .romlp
850 LDA #ASC"R":JSR &FFE3
860 LDA #ASC"O":JSR &FFE3
870 LDA #ASC"M":JSR &FFE3
880 JSR space:TXA:JSR hex0:JSR space
890 LDA #7:JSR rombyt:TAY:INY:TYA
900 JSR rombyt:CMP #ASC"(":BNE endinfo
910 INY:TYA
920 JSR rombyt:CMP #ASC"C":BNE endinfo
930 INY:TYA
940 JSR rombyt:CMP #ASC")":BNE endinfo
950 LDA #8:TAY
960 .lp INY:TYA:JSR rombyt:CMP #0
970 BEQ endt:CMP #127:BCC ok1
980 .ok2 LDA #ASC"?"
990 .ok1 CMP #32:BCC ok2:JSR &FFE3
1000 BNE lp
1010 .endt JSR space:LDA #8:JSR rombyt
1020 JSR hex:JSR space:LDA &2A1,X
1030 BNE endinfo:LDY #0
1040 .lp LDA unpl,Y:BEQ endinfo
1050 JSR &FFE3:INY:BNE lp
1060 .endinfo:JSR &FFE7:DEX:BPL romlp
1070 PLA:TAY:LDX &F4:LDA #0:RTS
1080 .power LDA #151:LDX #78:LDY #127
1090 JSR &FFF4:JMP (&FFFC)
1100 .cmds
1110 EQUS "UNPLUG":EQUW FNdec(unplug)
1120 EQUS "INSERT":EQUW FNdec(insert)
1130 EQUS "ROMS"  :EQUW FNdec(roms)
1140 EQUS "POWERUP":EQUW FNdec(power)
1150 EQUB 0
1160 .hex PHA:AND #&F0:ROR A:ROR A
1170 ROR A:ROR A:JSR hex0:PLA
1180 .hex0 AND #&F:STX temp:TAX
1190 LDA hexc,X:JSR &FFE3:LDX temp:RTS
1200 .hexc EQUS "0123456789ABCDEF"
1210 .unpl EQUS "unplugged"+CHR$0
1220 .bad EQUS "Bad rom number"+CHR$0
1230 .space LDA #32:JMP &FFE3
1240 .rm INY:LDA (&F2),Y:CMP #32:BEQ rm
1250 .str CMP #48:BCC badrom
1260 CMP #58:BCC decrom
1270 CMP #ASC"A":BCC badrom
1280 CMP #ASC"G":BCC hexrom
1290 .badrom LDX #0:STX &100:STX &101
1300 .lp INX:LDA bad-1,X:STA &101,X
1310 BNE lp:JMP &100
1320 .decrom SBC #47:JMP endr
1330 .hexrom SBC #54
1340 .endr CMP &F4:BEQ badrom:TAX:RTS
1350 ]:NEXT
1360 c$="SAVE MANAGER "+STR$~Q%+" "+STR
$~O%+" 8000 8000"
1370 PRINT"*"+c$:OSCLIc$
1380 END
1390 :
2000 DEFFNdec(a)=256*((a-1) MOD 256)+(a
-1) DIV 256
```

Repton Infinity

David Spencer looks at more than just a game.

Product	**Repton Infinity**
Supplier	**Superior Software**
	Regent House, Skinner Lane,
	Leeds LS7 1AX.
	Tel. (0532) 459453
Price	**£12.95 BBC/Electron tape**
	£14.95 BBC/Master disc
	£19.95 Master Compact disc
	(all prices inc. VAT)

THE STORY SO FAR

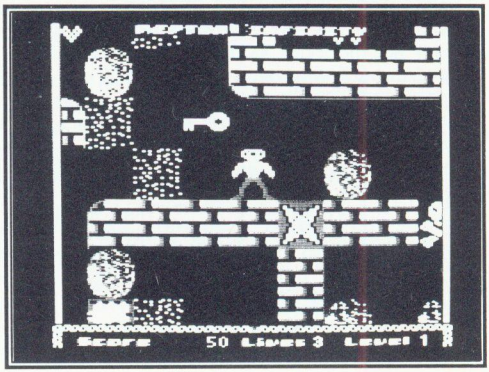
The original Repton was little more than a slightly non-conformist arcade game. It differed from space invaders and the like because a certain degree of thought and planning was required by the player. It was not enough to simply blast everything in sight.

Repton 2 was very different. Although it looked similar to the original, the new game was in fact a massive logic-puzzle. The object was to move Repton around the landscape collecting diamonds and earth, and killing monsters, while avoiding falling rocks. Some of the speed and action of an arcade game was still present, but so were the problem solving situations found in adventure games. The puzzle posed by Repton 2 was so challenging that Superior Software offered a substantial prize for the first person to solve it.

Repton 3 was a progression from Repton 2. The same problem-solving and arcade combination were retained, but instead of one large puzzle, there was a series of individual stages, each made up of eight screens. The stages could be attempted in any order, but to complete a stage, each of its screens had to be completed in succession. Major additions included in Repton 3 were character and screen editors. These allowed you to design your own screens, and alter the character to anything you liked. This quickly led to many new sets of screens being produced and marketed as add-ons. However, the behaviour of characters could not be changed, only their appearance. So, for example, rocks still fell if unsupported, even if they were redefined to look like flying saucers.

ROLL ON REPTON INFINITY

In some ways Repton Infinity, the latest offering from Superior, is a logical progression from Repton 3. The game is still there of course, as are the character and screen editors. However, new to Repton Infinity is the game editor which I shall cover in detail later.



The Repton 4 game

The Repton Infinity package is available for the model B and Master on cassette or disc, and for the Electron on cassette only. I used the disc version on a Master 128. The software is supplied on three discs. The game disc is a 40/80 track flippy with the standard version on one side, and an enhanced Master version on the other. I will explain the difference later. The other two discs contain the actual game files, and the other files that the editors produce. One of these is a single sided 80 track disc, the other is a 40 track flippy; both contain exactly the same data. The instruction manual is 72 pages long, and there is also a reference card.

When the disc is booted, there is an initial title screen followed by a menu. This offers five options: Play game, Land Scape, Film Strip, Blue Print, and File Link. I will deal with each in turn.

Play Game starts the actual game. There are in fact four different games supplied. The first is a version of Repton 3, while the second, called

Repton 4, adds some extra features while still maintaining the same theme. Robbo, the third game, sees you playing a robot in a 'time-space puzzle vortex', while in the fourth game, Trakker, you drive around in a latest hi-tech bulldozer killing hideous 'Jaggas'.

Land Scape is the screen layout editor. This allows a set of four game screens to be created and saved together. Each screen is edited by selecting a character and then using the cursor keys to position it on the screen. Existing screen files can be loaded, and the first four screens of Repton 3 are included to experiment with.

Film Strip is the character definer. Each character in the game actually exists in three forms. There is the large character shown on the playing screen, the small character used when displaying a map of the screen, and an editor character. Editor characters can be loaded into the editors in place of the default set as an aide memoir when editing screen layouts and character definitions.

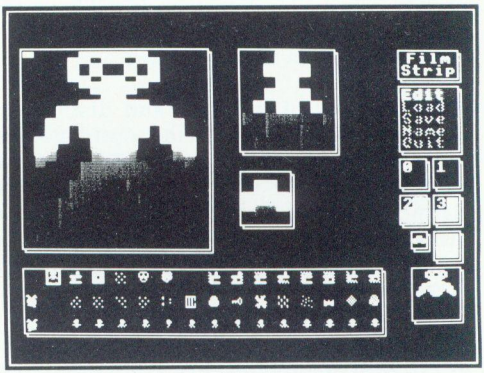
Blue Print is the game definer, which is the major feature new to Repton Infinity. The behaviour of each character in the game can be defined using a fairly sophisticated programming language called *Reptol*. The best way to explain this is with an example, namely the definition for a diamond in Repton 3:

```
NAME Diamond

DEFINE TYPE
      CurvedLeft
      CurvedRight

DEFINE HITS
      IF HITBY Repton
            SCORE (5)
            SOUND (242)
      ENDIF
```

The first line defines the name of the character, while the TYPE definition specifies certain attributes for that character. 'CurvedLeft' and 'CurvedRight' are flags which are used by the definition of a rock to control how it falls. They basically say that a rock can fall off a diamond to either side. The HITS definition determines what happens when another character hits the diamond. In this case, if the character is Repton himself, his score is increased by five and a 'ping' sound produced.

The action of each character is defined in this way, and *Reptol* includes some powerful features for effects such as random delays.

File Link joins the screen layouts, the character designs, and their actions together to form a complete game which can then be saved to disc. On the enhanced Master version, games can be tried without linking and saving them, but linking is compulsory each time on the other versions. This is the only advantage of the enhanced version.



Film Strip - the character designer

CONCLUSION

I see Repton Infinity as more than just a game, and indeed as more than just a successor to Repton 3. Repton Infinity provides a programming language in the form of *Reptol* which, while being fairly easy to learn and use, includes many of the features found in 'real' languages such as C. Add to this the fact that the results of your programming effort are (hopefully) an enjoyable game, then in my opinion, you have the perfect teaching tool. Pupils in first year IT lessons could be taught *Reptol* as an introductory language, and could be allowed to develop and try their own games.

You mustn't forget, though, that Repton is primarily a game, and a very good one at that. I am sure that the four games which accompany Repton Infinity will provide hours of pleasure to anyone. B

A Real-Time Clock for All

No real-time clock in your machine? This can be a thing of the past, as Kevin Harding's clever program provides both clock and alarm display for all users.

Anyone who has used one of the many computers which have the time constantly displayed somewhere on the screen will know how useful this is, especially if it is coupled with an alarm which can be set to remind you of meetings, picking someone up or just to stop you sitting at the computer all night when you get deeply involved. It can even be used to wake you up in the morning. That is what this program does. It performs the job of a real-time clock in software for those who do not have one built in to their system (as on the Master 128), but it will work on all machines.

It keeps track of the time and displays it digitally (24 hour clock) in the top right-hand corner of the screen in *any* screen mode. It also features an alarm which sounds when the alarm time is reached. The time can also be easily extracted in Basic, and can therefore be used within Basic programs. The program is Break proof, and maintains the time even if Ctrl-Break is pressed. It works not only in Basic but also in other languages such as Pascal and BCPL, with applications like Wordwise, View and Exmon, and Basic and machine code programs.

The program can assemble both a version to run in main memory, and also a version which will work from sideways RAM/EPROM. The sideways RAM/EPROM version also works with many commercial games (the main memory version tends to get overwritten by them). If the program is assembled on a Master computer, additional code is added to deal with shadow RAM. Additionally, the Master 128 version can 'grab' the time from the internal real-time clock, rather than prompting the user for it.

ENTERING THE PROGRAM

The program should be typed in and saved. When run, it will ask whether you want a main memory or sideways ROM version to be assembled. After selection, the program assembles the appropriate version and displays the *SAVE command necessary to save the code. If your machine runs out of memory during assembly, lower the value of PAGE (PAGE=&1200 for example) and try again.

USING THE PROGRAM

For the memory version type:

```
*RUN <name>
```

or `*<name>`

where <name> is the file name of the saved code. This sets up the vectors and prompts for the start time, unless it is running on a Master 128. If an invalid time is entered it uses the default which is midnight (00:00:00). This version should be used on systems where PAGE is normally set to &1900 by default, and which do not have sideways RAM.

For the ROM version the saved ROM image should be loaded using your normal sideways RAM loader (*SRLOAD on a Master or Compact for example), and Break or Ctrl-Break should be pressed to initialise it, during which it will prompt for the start time, unless the computer is a Master 128.

Once loaded, your clock can be controlled with six different star commands.

COMMANDS

*SETTIME <HH:MM:SS> sets the time shown on the clock. The time is specified in the 24 hour clock system, and preceding zeros must be used for single digit values. For example:

```
*SETTIME 07:00:00 (NOT *TIME 7:00:00)
```

On a Master 128, if no time is given, then the time is read from the internal real-time clock.

*ALARM <HH:MM:SS> sets the time at which the alarm will sound. An '=' is displayed after the time on the screen to show that the alarm is set. *ALARM without a time de-activates the alarm; it also stops the alarm while it is sounding.

*COFF deactivates the screen display. Correct time will be maintained but will not be displayed on the screen until *CON is used.

*CON reactivates the screen display after executing a *COFF.

*CSTOP disables the event the clock uses to maintain the time. This means that the time is not updated and also that it is not displayed. This can be used if the clock program causes other programs (certain machine-code games) to crash.

*CSTART re-enables the event and starts the clock. You will be prompted for the new time, unless the program is running on a Master 128.

The time can also be read from BASIC using the statement 'TI$=$&38D'. This makes TI$ a six-digit string, the first two digits, the hours, the second two the minutes, and the last two the seconds. These can be extracted using LEFT$, MID$ and RIGHT$.

MEMORY USAGE

Both versions use the same areas of workspace when running; these are &A8 to &AB during star commands, &90-&95 during event processing and &380-&39A for general storage. The &380-&39A area is in the cassette workspace used for BPUT operations. If you are using this cassette command, this workspace will have to be moved. Other usable areas are from &880 if you are not using a printer or from &8C0 if you are not using sound envelopes. To change them just change the value of other in line 190.

The main memory version of the program resides between &12C0 and &1700. This is part of the memory the DFS uses as a file buffer. This means that if you are using the memory version, you cannot open more than one file at a time as it would overwrite the code. Using the main memory version with cassette is rather difficult. It needs to be assembled at &E00. This can be done by changing the second K% in line 1010 from &12C0 to &E00, setting PAGE to &1400, loading the program and running it and saving the code produced. Then whenever you use the code you need to set PAGE to &1400 and *RUN the code. PAGE must then always be reset to &1400 whenever <Break> or <Ctrl-Break> is pressed.

HOW IT WORKS

The program uses the internal interval timer to generate an event every second. The event is vectored through &220/&221 and is redirected to the clock program. Each time the routine is entered it increments the time by one, displays the clock on the screen (if enabled) and checks the alarm time to see if it has been reached; if it has, the number of beeps to be sounded is set to 30 and the first beep is sounded, and on consecutive events a beep is again sounded until all 30 have been produced.

```
   10 REM Program Clock/Alarm
   20 REM Version B1.01
   30 REM Author  Kevin Harding
   40 REM BEEBUG  Jan/Feb 1989
   50 REM Program subject to copyright
   60 :
  100 MODE7:DIM code &600
  110 PRINT"ASSEMBLE FOR ROM OR MEMORY (
R/M)? : ";
  120 q$=CHR$(GET AND &DF)
  130 IFq$<>"R" AND q$<>"M" GOTO120
  140 PRINTq$'
  150 FORx=1 TO 2:PRINTCHR$141;SPC(10);"
ASSEMBLING TO";CHR$136;
  160 IF q$="M" PRINT"MEMORY" ELSE PRINT
"ROM"
  170 NEXT
  180 VDU28,0,24,39,5
  190 starzp=&A8:eventzp=&90:other=&380
  200 sstart=eventzp:sblank=eventzp+2
  210 cinp=starzp:cpoint=starzp+2
  220 xtemp=other:ytemp=other+1
  230 atemp=other+2:temp=other+3
  240 pchar=other+4:index=other+5
  250 vcolumns=other+6:clockflag=other+7
  260 beeps=other+8:oevent=other+9
  270 ocliv=other+11:clock=other+13
  280 alarm=other+20:tstack=other+26
  290 A%=0:X%=1:version=(USR &FFF4 AND &
FF00) DIV &100
  300 PROCassemble
  310 PRINT'"TO SAVE CODE, TYPE :"
  320 PRINT"*SAVE <Filename> ";~code;" "
;~O%;
  330 IF q$="M" PRINT" 12C0 12C0" ELSE P
RINT
  340 END
  350 :
 1000 DEFPROCassemble
 1010 IFq$="R" THEN K%=&8000 ELSE K%=&12
```

```
C0
 1020 FORI%=4 TO 7 STEP 3:P%=K%:O%=code
 1030 IFq$="R" PROCrombreak:GOTO 1110
 1040 [OPTI%:SEI:LDA#249:LDY#0
 1050 LDX#brk DIV 256:JSR&FFF4:LDA#248
 1060 LDY#0:LDX#brk MOD 256:JSR&FFF4
 1070 LDA#247:LDY#0:LDX#&4C:JSR&FFF4
 1080 LDA#13:STA clock+6:LDA#1
 1090 STA clockflag:JSR zero:JSR sev
 1100 JSR stup:CLI:JMP tii:]
 1110 [OPTI%:.zero LDX#6:LDA#&30
 1120 .bclk STA clock-1,X:DEX:BNE bclk
 1130 RTS:.sti LDX#cset MOD 256
 1140 LDY#cset DIV 256:LDA#4:JMP &FFF1:]
 1150 IFq$="R" PROCromevents:GOTO1290
 1160 [OPTI%:.sev JSR sti:LDA &220
 1170 STA oevent:LDA &221:STA oevent+1
 1180 LDA#event MOD256:STA &220
 1190 LDA#event DIV256:STA &221:LDA#14
 1200 LDX#5:JMP&FFF4:.stup LDA &208
 1210 STA ocliv:LDA &209:STA ocliv+1
 1220 LDA#scomm MOD256:STA &208
 1230 LDA#scomm DIV256:STA &209:RTS
 1240 .brk BCS rst:RTS
 1250 .rst PHP:PHA:TXA:PHA:TYA:PHA:SEI
 1260 JSR stup:LDA#8:BIT clockflag
 1270 BNE chalt:JSR sev:.chalt CLI
 1280 JMP rescur:]
 1290 IF q$="R" PROCromstarcommand:GOTO1
320
 1300 [OPTI%:.scomm STX cinp:TXA:PHA
 1310 STY cinp+1:TYA:PHA:]
 1320 [OPTI%:LDA#aco MOD 256:STA cpoint
 1330 LDA#aco DIV 256:STA cpoint+1
 1340 LDY#&FF:.nxts INY:LDA (cinp),Y
 1350 CMP#ASC"*":BEQ nxts:TYA:CLC
 1360 ADC cinp:STA cinp:BCC anc
 1370 INC cinp+1:.anc LDX#&FA
 1380 .cnxt LDY#0:.snxt LDA (cpoint),Y
 1390 BNE cch:JMP cmatch
 1400 .cch LDA (cinp),Y:JSR ctc
 1410 CMP (cpoint),Y:BNE cun:INY
 1420 JMP snxt:.cun LDA (cinp),Y
 1430 CMP#ASC".":BNE fnxt:INY:CPY#2
 1440 BCS cmatch:.fnxt INY:LDA(cpoint),Y
 1450 BNE fnxt:]
 1460 IFq$<>"R" GOTO 1490
 1470 [OPTI%:.fnxt2 INY:LDA(cpoint),Y
 1480 BNE fnxt2:]
 1490 [OPTI%:INY:TYA:CLC:ADC cpoint
 1500 STA cpoint:BCC nxt:INC cpoint+1
 1510 .nxt INX:BNE cnxt:]
 1520 IFq$="R" PROCromstarexit:GOTO1550
 1530 [OPTI%:PLA:TAY:PLA:TAX:JMP (ocliv)
 1540 ]
 1550 IF version<>3 THEN 1600
 1560 [OPT I%:.tii LDA cblk:STA cinp
 1570 LDA cblk+1:STA cinp+1:LDY #0
 1580 LDA #13:STA (cinp),Y:]
 1590 GOTO1650
 1600 [OPTI%:.tii LDY#0:.wnxt LDA ftm,Y
 1610 BEQ fmw:JSR&FFEE:INY:JMP wnxt
 1620 .fmw LDA#0:LDX#cblk MOD 256
 1630 LDY#cblk DIV 256:JSR &FFF1
 1640 BCC sval:JMP invt:]
 1650 [OPT I%:.sval LDA cblk
 1660 STA cinp:LDA cblk+1:STA cinp+1
 1670 LDY#0:JSR ft:BCC tnc:JMP invt
 1680 .ftm EQUS "Enter time : ":EQUB &00
 1690 .cmatch PLA:PLA:]
 1700 IFq$<>"R" GOTO1720
 1710 [OPTI%:PLA:]
 1720 [OPTI%:CPX#251:BEQ cs:BCS seld
 1730 LDA#&41:STA clockflag:JSR zero
 1740 JSR sev:]
 1750 IFq$<>"R" [OPTI%:JMP tii:]:GOTO177
0
 1760 [OPTI%:JSR tii:LDA#0:RTS:]
 1770 [OPTI%:.cs LDA#13:LDX#5:JSR&FFF4
 1780 JSR wab:LDA#8:]
 1790 IFq$="R" [OPTI%:JSR sf:LDA#0:RTS:]
 1800 [OPTI%:.sf ORA clockflag
 1810 STA clockflag:RTS:.seld CPX#254
 1820 BCS sut:CPX#252:BEQ co:LDA#&FE
 1830 JSR cf:]
 1840 IFq$<>"R" [OPTI%:JMP wab:]:GOTO186
0
 1850 [OPTI%:JSR wab:LDA#0:RTS:]
 1860 [OPTI%:.co LDA#1:]
 1870 IFq$<>"R" [OPTI%:JMP sf:]:GOTO1890
 1880 [OPTI%:JSR sf:LDA#0:RTS:]
 1890 [OPTI%:.sut CPX#254:BNE sa:JSR ft
 1900 BCS invt:SEI:.tnc LDA (cinp),Y
 1910 CMP#ASC":":BEQ nxty:STA clock,X
 1920 INX:.nxty INY:CPX#6:BNE tnc:CLI:]
 1930 IFq$<>"R" GOTO1950
 1940 [OPTI%:LDA#0:]
 1950 [OPTI%:RTS:.sa JSR ft2:BCC sat
 1960 LDA#&F9:]
 1970 IFq$="R" [OPTI%:JSR cf:LDA#0:RTS:]
 1980 [OPTI%:.cf AND clockflag
 1990 STA clockflag:RTS
 2000 .sat LDA (cinp),Y:CMP#ASC":"
 2010 BEQ nxty2:STA alarm,X:INX
 2020 .nxty2 INY:CPX#6:BNE sat:LDA#2:]
 2030 IFq$<>"R" [OPTI%:JMP sf:]:GOTO2050
 2040 [OPTI%:JSR sf:LDA#0:RTS:]
 2050 IFq$="R" PROCromerror:GOTO2070
 2060 [OPTI%:.invt EQUB &00:]
 2070 [OPTI%:.emess EQUB &70
```

```
 2080 EQUS "Invalid Time":EQUB &00
 2090 .ft:]IF version=3 THEN PROCpulltim
e
 2100 [OPT I%
 2110 .ft2 LDA (cinp),Y:CMP#13:BEQ bt
 2120 JSR cvn:BCC ffn:CMP#32:BNE invt
 2130 INY:JMP ft2:.bt SEC:RTS
 2140 .ffn STY ytemp:LDX#0
 2150 .fnd LDA (cinp),Y:JSR cvn:BCS inj
 2160 CMP cmax,X:BCS inj:INY:INX:CPX#2
 2170 BEQ ccol:CPX#4:BEQ ccol:CPX#6
 2180 BNE fnd:LDY ytemp:LDA (cinp),Y
 2190 CMP#&32:BNE fvt:INY:LDA (cinp),Y
 2200 CMP#&34:BCC fvt:.inj JMP invt:.fvt
 LDY ytemp
 2210 LDX#0:RTS:.ccol LDA(cinp),Y
 2220 CMP#ASC":":BEQ vtime:JMP invt
 2230 .vtime INY:JMP fnd:.cvn CMP#48
 2240 BCC inv:CMP#58:RTS:.inv SEC:RTS:]
 2250 IFq$="R" PROCcommands:GOTO 2310
 2260 [OPTI%:.aco EQUS "CSTART":EQUB &00
 2270 EQUS "CSTOP":EQUB &00:EQUS "CON"
 2280 EQUB &00:EQUS "COFF":EQUB &00
 2290 EQUS "SETTIME":EQUB &00:EQUS "ALAR
M"
 2300 EQUB &00:]
 2310 [OPTI%:.ctc CMP#ASC"a":BCC charok
 2320 CMP#ASC"z"+1:BCS charok:AND#&DF
 2330 .charok RTS:.event PHP:CMP#5
 2340 BEQ ceven:PLP:JMP(oevent)
 2350 .ceven PHA:TXA:PHA:TYA:PHA:JSR sti
 2360 LDX#5:.incc INC clock,X
 2370 LDA clock,X:CMP cmax,X:BNE chk24
 2380 LDA#48:STA clock,X:DEX:BPL incc
 2390 .chk24 LDA clock:CMP#&32:BNE wc
 2400 LDA clock+1:CMP#&34:BNE wc:LDA#48
 2410 STA clock:STA clock+1:.wc LDA#2
 2420 BIT clockflag:BEQ chkp:LDY#6
 2430 .chka LDA clock-1,Y:CMP alarm-1,Y
 2440 BNE chkp:DEY:BNE chka:LDA#30
 2450 STA beeps:LDA#4:JSR sf
 2460 .chkp LDA#4:BIT clockflag:BEQ chkd
 2470 LDA#7:JSR&FFEE:DEC beeps:BNE chkd
 2480 LDA#&F9:JSR cf:.chkd LDA#1
 2490 BIT clockflag:BEQ rescur
 2500 ]IF version>2 PROCsaveacccon
 2510 [OPT I%:JSR wab:LDY#2
 2520 .addchar JSR cnss:DEY:BNE addchar
 2530 .nxtcb LDA clock,X:JSR wchar:INX
 2540 INY:CPX#6:BEQ cas:CPX#2:BEQ wcol
 2550 CPX#4:BEQ wcol:JMP nxtcb
 2560 .wcol LDA#ASC":":JSR wchar:INY
 2570 JMP nxtcb:.cas LDA#2:BIT clockflag
 2580 BEQ swpage:LDA#ASC"=":JSR wchar
 2590 .swpage:]IF version>2 PROCresetacc
con
 2600 [OPT I%:.rescur PLA:TAY:PLA:TAX
 2610 PLA:PLP:RTS
 2620 .wchar STA pchar:TYA:PHA:TXA:PHA
 2630 LDA&34F:CMP#1:BNE ntele:LDA pchar
 2640 LDY#0:STA (sstart),Y:JMP ewchar
 2650 .ntele PHA:LDA pchar:STA &13F
 2660 LDX #&3F:LDY #1:LDA #10:JSR &FFF1
 2670 LDX #0:LDY#0:PLA:CMP#8:BNE n2c
 2680 .wnl LDA &140,X:STA (sstart),Y
 2690 INX:INY:CPY#8:BNE wnl:JMP ewchar
 2700 .n2c CMP#16:BNE n4c:LDY#0
 2710 .nll LDA &140,X:AND#&F0:STA temp
 2720 LSR A:LSR A:LSR A:LSR A:ADC temp
 2730 STA (sstart),Y:INX:INY:CPY#8
 2740 BNE nll:LDX #0:.nrl LDA &140,X
 2750 AND#&F:STA temp:ASL A:ASL A:ASL A
 2760 ASL A:ADC temp:STA (sstart),Y
 2770 INX:INY:CPY#16:BNE nrl:BEQ ewchar
 2780 .n4c STY index:LDX#0:.ncol LDY #0
 2790 LDA#8:STA vcolumns:.nxt2 LDA#0
 2800 STA pchar:LDA &140,Y:PHA:STY xtemp
 2810 AND table,X:BEQ nxtp:LDA pchar
 2820 ORA#&2A:STA pchar:.nxtp PLA
 2830 AND table+1,X:BEQ p2b:LDA pchar
 2840 ORA#&15:STA pchar:.p2b LDA pchar
 2850 LDY index:STA (sstart),Y:INC index
 2860 LDY xtemp:INY:DEC vcolumns
 2870 BNE nxt2:INX:INX:CPX#8:BNE ncol
 2880 .ewchar PLA:TAX:PLA:TAY:.cnss CLC
 2890 LDA sstart:ADC &34F:STA sstart
 2900 BCC frout:INC sstart+1:.frout RTS
 2910 .wab LDA &355:ASL A:TAY:LDX#&7E
 2920 .anb LDA &350-&7E,X:ADC soff,Y
 2930 STA sstart-&7E,X:STA sblank-&7E,X
 2940 INY:INX:BPL anb:LDX#11:.wnbc LDY#0
 2950 LDA#0:.wns STA (sblank),Y
 2960 INY:CPY &34F:BNE wns:DEX:BNE cns
 2970 RTS:.cns CLC:LDA sblank:ADC &34F
 2980 STA sblank:BCC wnbc:INC sblank+1
 2990 JMP wnbc:.cset EQUD &FFFFFF9C
 3000 EQUB &FF:.cmax EQUD &3A363A33
 3010 EQUW &3A36:.soff EQUW &228
 3020 EQUW &1D0:EQUW &120:EQUW &228
 3030 EQUW &0E8:EQUW &090:EQUW &0E8
 3040 EQUW &01D:.table EQUD &10204080
 3050 EQUD &01020408:.cblk EQUW &700
 3060 EQUB &8:EQUB &20:EQUB &7E:]
 3070 NEXT
 3080 ENDPROC
 3090 :
 3100 DEFPROCromerror
 3110 [OPTI%:.invt LDA clockflag
 3120 CMP#&61:BNE weo:TSX:INX:INX
 3130 CPX tstack:BEQ ertine:PLA:PLA
```

```
3140 .ertine RTS:.weo LDY#&FF:LDA#0
3150 STA&100:.cem INY:LDA emess,Y
3160 STA &101,Y:BNE cem:JMP &100:]
3170 ENDPROC
3180 :
3190 DEFPROCrombreak
3200 [OPTI%:EQUB &00:EQUW &0000
3210 JMP service:EQUB &82
3220 EQUB cp MOD 256:EQUB &01
3230 EQUS "CLOCK/ALARM ROM":EQUB &00
3240 EQUS "1.01":.cp EQUB &00
3250 EQUS "(C)":EQUS " KAH":EQUB &00
3260 .service CMP#4:BNE chkh:JMP scomm
3270 .chkh CMP#9:BEQ hcomm:CMP#&FE
3280 BEQ bpres:RTS
3290 .bpres PHA:TYA:PHA:TXA:PHA:LDA#&E0
3300 AND clockflag:CMP#&40:BNE iset
3310 LDA#8:BIT clockflag:BNE etg
3320 JSR sev:.etg JMP rstr
3330 .iset LDA#&61:STA clockflag:LDA#13
3340 STA clock+6:JSR zero:JSR sev
3350 TSX:STX tstack:JSR tii:LDA#&41
3360 STA clockflag:.spres LDA#&79
3370 LDX#201:JSR&FFF4:CPX#0:BMI spres
3380 LDA#12:JSR&FFE3:JMP rstr
3390 .hcomm PHA:TYA:PHA:TXA:PHA:DEY
3400 .shcom INY:LDA(&F2),Y:CMP#32
3410 BEQ shcom:CMP#13:BNE acfc:JSR won
3420 LDY#0:.wncc LDA command-2,Y
3430 BEQ cww:JSR &FFE3:INY:BNE wncc
3440 .cww JSR &FFE7:JSR&FFE7:JMP rstr
3450 EQUW &2020:.command EQUS "CLOCK"
3460 EQUB &00:.acfc LDX#0
3470 .cncc LDA (&F2),Y:JSR ctc:CMP#13
3480 BEQ cend:CMP#32:BEQ cend
3490 CMP command,X:BNE cfd:INY:INX
3500 BNE cncc:.cfd CMP#ASC".":BNE acnm
3510 CPX#2:BCS acmatch
3520 .acnm JMP rstr:.cend LDA command,X
3530 BNE acnm:.acmatch JSR won:LDY#0
3540 .wnc LDA#32:JSR&FFE3:JSR&FFE3
3550 .wncl LDA aco,Y:BEQ wsyn:JSR&FFE3
3560 INY:JMP wncl:.wsyn LDA#32:JSR&FFE3
3570 .wnsyn INY:LDA aco,Y:BEQ synw
3580 JSR &FFE3:JMP wnsyn
3590 .synw JSR &FFE7:INY:LDA aco,Y
3600 BEQ allc:BNE wnc:.allc JSR &FFE7
3610 .rstr PLA:TAX:PLA:TAY:PLA:RTS
3620 .won JSR&FFE7:LDY#8:.wnt INY
3630 LDA&8000,Y:BEQ ftit:JSR&FFE3
3640 JMP wnt:.ftit LDA#32:JSR&FFE3
3650 CPY&8007:BNE wnt:JSR &FFE7:RTS:]
3660 ENDPROC
3670 :
3680 DEFPROCcommands
3690 [OPTI%:.aco EQUS "CSTART"
3700 EQUW &0000:EQUS "CSTOP"
3710 EQUW &0000:EQUS "CON"
3720 EQUW &0000:EQUS "COFF"
3730 EQUW &0000:EQUS "SETTIME"
3740 EQUB &00:EQUS "<HH:MM:SS>"
3750 EQUB &00:EQUS "ALARM"
3760 EQUB &00:EQUS "(HH:MM:SS)"
3770 EQUB &00:EQUB &00:]
3780 ENDPROC
3790 :
3800 DEFPROCromstarcommand
3810 [OPTI%:.scomm PHA:TYA:PHA:TXA:PHA
3820 TYA:CLC:ADC &F2:STA cinp:LDA &F3
3830 ADC#0:STA cinp+1:]
3840 ENDPROC
3850 :
3860 DEFPROCromstarexit
3870 [OPTI%:PLA:TAX:PLA:TAY:PLA:RTS:]
3880 ENDPROC
3890 :
3900 DEFPROCromevents
3910 [OPTI%:.sev JSR sti:LDA &220
3920 STA oevent:LDA &221:STA oevent+1
3930 LDA#&30:STA&220:LDA#&FF:STA&221
3940 LDA#&A8:LDX#0:LDY#&FF:JSR&FFF4
3950 STX cpoint:STY cpoint+1:LDY#&30
3960 LDA#event MOD 256:STA (cpoint),Y
3970 INY:LDA#event DIV 256
3980 STA (cpoint),Y:INY:LDA &F4
3990 STA (cpoint),Y:LDA#14:LDX#5
4000 JMP&FFF4:]
4010 ENDPROC
4020 :
4030 DEF PROCsaveacccon
4040 [OPT I%:LDA &FE34:PHA:AND #2
4050 BEQ sacc:LDA &FE34:ORA #4
4060 BNE sacc2:.sacc:LDA &FE34:AND #&FB
4070 .sacc2:STA &FE34:]
4080 ENDPROC
4090 :
4100 DEF PROCresetacccon
4110 [OPT I%:PLA:STA &FE34:]
4120 ENDPROC
4130 :
4140 DEF PROCpulltime
4150 [OPT I%:.ptt TYA:PHA:DEY:.ptt2 INY
4160 LDA (cinp),Y:CMP #32:BEQ ptt2
4170 BCC ptt3:PLA:TAY:JMP ft2
4180 .ptt3 PLA:LDX#64:LDY#0:STY &140
4190 INY:LDA#14
4200 JSR&FFF1:LDY#1:STY cinp+1
4210 LDA #&50:STA cinp:DEY:]
4220 ENDPROC
```

Steaming Ahead with ASTAAD

David Demaine extends the ASTAAD environment with a supporting program to manipulate screen images.

In the December issue (Vol.7 No.7), we published the third and final part of the ASTAAD3 CAD program. This month the associated program STEAMS3 is published in full. The name originated from the concept of the distortions that can be imparted to materials by steaming them under stress. If you must have an acronym for the name then try Shrink, Twist, Expand And Mix Screens!

The STEAMS program provides facilities to identify a rectangular area of the screen, and to distort this by moving, copying, rotating, enlarging or reducing the enclosed image. STEAMS maintains two images as main and shadow screens, and one image can overlay or replace another.

STEAMS3 is used in association with ASTAAD3 and the two programs must reside in the same directory. To prepare for the use of the program published this month you must:

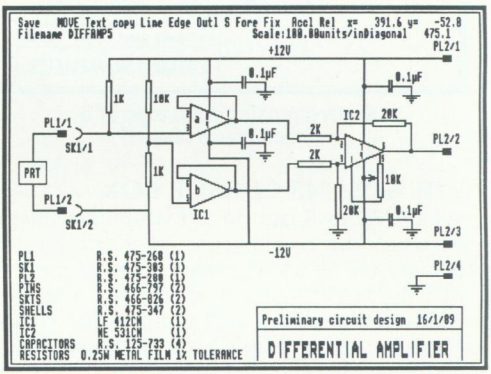
1. Make sure your copy of ASTAAD is named as ASTAAD3.
2. Delete line 6190 in ASTAAD3,which inhibits the 'Link' function.
3. Change line 6220 in ASTAAD3 to read:
   `VDU4:J%=&7FFFFF:CHAIN"STEAMS3":ENDPROC`

THE STEAMS PROGRAM

This program is much shorter than ASTAAD, though quite substantial in its own right. It has to be kept within limits because it uses two full mode 0 screens, one in main memory commencing at &3000, and the other in shadow memory. HIMEM is set to &3000 while STEAMS is running. This reduces the Master 128 down to nearly the capacity of the BBC model B. To save space it has been necessary to abbreviate some function and variable names, which does unfortunately reduce readability.

I have set the screen backgrounds to different colours, white for the SHADOW screen and blue for the MAIN screen, so that it is easy to see which screen is visible at any instant. From ASTAAD, Ctrl-f7 will now link to STEAMS, but there will be no change in the display, except for a change of colour to black and white. Any image you had on your ASTAAD screen will still be there, but with the appearance of lines around the edge.

As with ASTAAD there is only one screen format, a 1280 X 960 graphics screen with a two line header along the top. The STEAMS program is controlled by the function keys, the Escape, Tab and Copy keys, and the cursor keys for cursor movement.



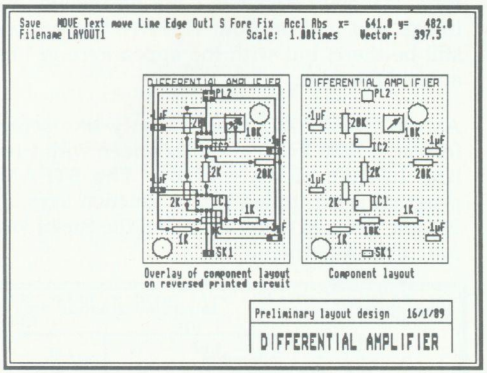
Designing a circuit

Keystrips for both ASTAAD and STEAMS are printed on page 62 of this issue. You will see that f0 on the STEAMS keystrip is marked 'Switch screens'. Press this once and a blue background screen will appear with MAIN in the header. You can use Ctrl-f9 to load a screen from memory to either the MAIN or the SHADOW screen. If you link back to ASTAAD using Ctrl-f7 at any time, the MAIN screen will be lost and the white SHADOW screen will turn back to yellow.

CORRESPONDING KEYS TO ASTAAD

You will soon see that the Save, Load and Print screen functions are identical to ASTAAD and on the same function keys. Other familiar keys are Margins, Origin rel/abs, Delete box and Area move with its associated toggle to Copy or

Move. These are indeed all identical to the equivalent ASTAAD functions, except that operations are immediate without the opportunity for second thoughts. There are no cursor drawing functions, or any way of placing text on the screen.



Superimposing an image of a printed circuit board

THE STEAMS CURSOR BOX

What STEAMS can do is manipulate images between the two screens. If you have just entered the STEAMS program, press the Tab, and the lower and left hand lines down the edges of the screen will jump to the cursor. Move the cursor a little way, both horizontally and vertically and then press Copy. This will bring the other two lines to the new cursor position, defining a box on the screen. To make the box disappear, press Tab and then Copy without moving the cursor. The box lines have no effect on the image you see on the screen, and are not stored when you save the image. They are used to define the area of operation of the principal STEAMS functions.

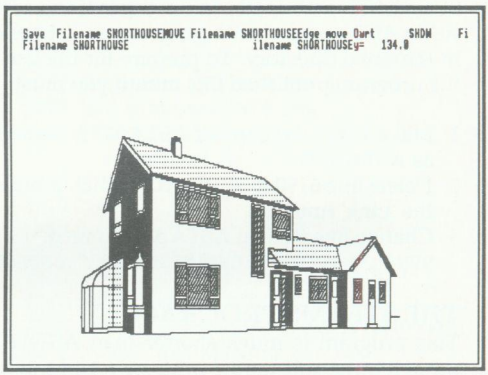
STEAMS FUNCTIONS

Function keys f1 and Shift-f1 provide you with the ability to move or copy any part of the screen. The cursor box defines the area which will be moved or copied, and itself moves to the new position. The margins toggle, Ctrl-f3, allows you to move the cursor off screen, which may be necessary in some cases.

IMAGE ROTATION

f2 is the 'Mix screens' function. The area in the box on the screen you are viewing is moved onto the hidden screen. The display switches to the hidden screen. When you call this function, a header message gives you the opportunity to specify a rotation angle. If you enter a magnitude greater than 360 degrees the function is aborted. Rotation takes place about the centre of the box. The result of rotation will make changes to your image because of the imperfect match of the rotated pixels to their new positions, but the results are not unacceptable. The 2 to 1 aspect ratio of the pixels is a particular cause of problems, and some simple precautions have been taken in the program to minimise adverse effects.

Shift-f2 is a toggle associated with the 'Mix screens' function, and gives the user the opportunity of overwriting or overlaying the image on the hidden screen. An example of this is shown by the illustration of a printed circuit board, where both sides are superimposed to prove registration.



A screen with two rotations and anamorphic shrink

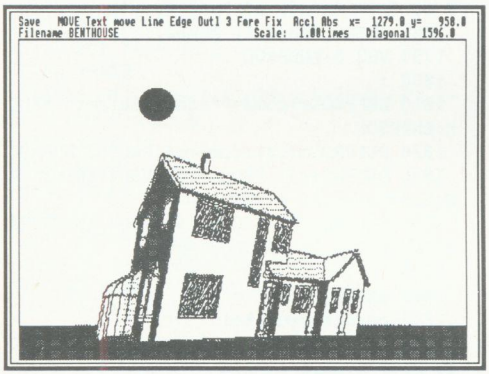
IMAGE SIZE CHANGE

A 'Shrink screen' function is provided by f3, which allows you to shrink or expand any boxed area of a screen. The new image is created on the hidden screen with the bottom left-hand corner of the area in the same position, and the box moves to the new shape and position. Before you specify the magnitude of the size change, a message in the header gives you the opportunity to restrict the change to either the x or the y dimension only. This enables anamorphic distortion of the picture

with results as illustrated on the house projection, taken from last months issue. If you enter zero for the shrink factor, the function is aborted. If you expand the image with a 'shrink factor' greater than unity, gaps will appear in the transformed image. This function, like all the others may operate on either the MAIN or the SHADOW screen.

f8 deletes the borders around the box. Incidentally there are no checks in STEAMS to warn the user that he may seriously damage his image, but these should not be necessary as the screen is easily reloaded in cases of accident.

As with ASTAAD, STEAMS is a highly modular Basic program which allows further functions to be added by the user as required.



A screen with two rotations and anamorphic horizontal expand

This program concludes our coverage of revised ASTAAD CAD package. We believe that it provides many of the features available in commercial offerings, and we hope that the illustrations accompanying the articles will not only have demonstrated the power of ASTAAD, but will spur readers on to try it out for themselves.

STEAMS appears on this month's magazine disc/tape, but we have also prepared a self-contained ASTAAD disc which includes ASTAAD3 and STEAMS3 together with a number of demonstration screens to get you started. The disc is accompanied by supporting documentation and keystrips for both programs. See page 35 for order details.

```
   10 REM Program STEAMS3
   20 REM Version A1.14
   30 REM Author  David Demaine
   40 REM BEEBUG  Jan/Feb 1989
   50 REM Program Subject to copyright
   60 :
  100 IF J%=&7FFFFF chain%=TRUE ELSE cha
in%=FALSE
  110 J%=13
  120 IF NOT chain% MODE128
  130 HIMEM=&3000
  140 DIM OS% 40
  150 PROCinit
  160 PROCsetup
  170 ON ERROR PROCerror
  180 quit%=FALSE
  190 :
  200 REPEAT
  210 PROCruler:PROCcursor
  220 key%=INKEY(0):J%=key%-128:IF key%<
=0 THEN UNTIL FALSE
  230 *FX15,1
  240 IF TIME<10 PROCaccel
  250 IF TIME>=10 count%=0:s%=4
  260 ON J%-12 PROCleft,PROCright,PROCdo
wn,PROCup ELSE 280
  270 UNTIL FALSE
  280 SOUND17,-10,125,1
  290 IF key%=27 AND NOT INKEY-1 AND NOT
 INKEY-2 PROChome:UNTIL FALSE
  300 IF key%=27 AND INKEY-1 AND NOT INK
EY-2 PROCbox:CLG:PROCbox:UNTIL FALSE
  310 IF key%=27 AND INKEY-1 AND INKEY-2
 PROCquit:UNTIL FALSE
  320 IF NOT (key%=9 OR (J%>0 AND J%<43)
) UNTIL FALSE
  330 PROCbox
  340 IF key%=9 x1=x:y1=y:PROCbox:UNTIL
FALSE
  350 ON ((J% DIV 16)+1) PROCf,PROCsf,PR
OCcf
  360 PROCbox
  370 UNTIL FALSE
  380 :
 1000 DEFPROCerror
 1010 PROCbox
 1020 IF ERR=17 AND NOT INKEY-1 PROChome
:PROCbox:ENDPROC
 1030 VDU4,31,6,0
 1040 REPORT:PRINT " at line "; ERL;" ha
s occurred;"'"Continue? (Y/N)";
 1050 IF quit% GOTO1070
 1060 IF (GET AND &DF)=ASC"Y" VDU12,5:PR
OCr:ENDPROC
 1070 *FX4,0
 1080 *FX229,0
```

```
 1090 VDU22,128
 1100 REPORT:PRINT " at line "; ERL'
 1110 PRINT "NEERCS ";:ON ERROR OFF
 1120 *DELETE NEERCS
 1130 END
 1140 :
 1150 DEFPROCsave:IF J%<>41 ENDPROC
 1160 fn$="Save  "
 1170 $OS%=FNc("File name?")
 1180 PROCm("Filename "+$OS%)
 1190 IF NOT main%:*FX108,1
 1200 *SAVE NEERCS 3000 7FFF
 1210 IF NOT main%:*FX108,0
 1220 OSCLI("RENAME NEERCS "+$OS%)
 1230 PROCr:ENDPROC
 1240 :
 1250 DEFPROCload:IF J%<>42 ENDPROC
 1260 fn$="Load  "
 1270 Fname$=FNc("File name?"):PROCm("Lo
ading "+Fname$)
 1280 OSCLI("RENAME "+Fname$+"  NEERCS")
 1290 IF NOT main%:*FX108,1
 1300 *LOAD NEERCS 3000
 1310 IF NOT main%:*FX108,0
 1320 OSCLI("RENAME NEERCS "+Fname$)
 1330 PROCr:ENDPROC
 1340 :
 1350 DEFPROCinit
 1360 b$=CHR$32:c$=STRING$(2,b$):d$=STRI
NG$(4,b$)
 1370 p$=STRING$(10,b$):Fname$=p$
 1380 @%=5
 1390 lx%=1280:ly%=960
 1400 VDU3,4
 1410 VDU29|
 1420 VDU24,0;0;lx%-1;ly%-1;
 1430 VDU18,0,128
 1440 IF NOT chain% CLG
 1450 VDU28,0,1,79,0
 1460 VDU23,1|
 1470 VDU19,0,7|19,1,0|
 1480 *FX229,1
 1490 *FX4,2
 1500 *FX225,129
 1510 *FX226,145
 1520 *FX227,161
 1530 *FX228,177
 1540 *FX112,1
 1550 CLG
 1560 *FX112,2
 1570 ENDPROC
 1580 :
 1590 DEFPROCsetup
 1600 x=lx%/2+1:y=ly%/2+2
 1610 J%=13
 1620 x1=0:y1=0:x2=1278:y2=956:x4=0:y4=0
:vector=0
 1630 s%=4:t%=50:count%=0
 1640 fn$="Setup ":com$="Double screen e
dit has been initialised"
 1650 areacode%=0
 1660 main%=FALSE:main$="SHADOW"
 1670 copy%=FALSE:copy$="move"
 1680 owrt%=TRUE:owrt$="Owrt"
 1690 orig$="Abs "
 1700 vector%=TRUE:vector$=" Vector:"
 1710 marg%=FALSE:marg$="Edge"
 1720 PROCh:PROCd:PROCr
 1730 PROCbox:ENDPROC
 1740 :
 1750 DEFPROCh:VDU4,12
 1760 coma$=com$+STRING$(41-LEN(com$),b$
)
 1770 PRINT TAB(0,0) fn$ d$ main$ d$ mar
g$ c$ copy$ c$ owrt$ STRING$(17,b$) orig
$ d$ "x=" (x-x4) d$ "y=" (y-y4);
 1780 PRINT TAB(0,1) coma$;
 1790 VDU 5:ENDPROC
 1800 :
 1810 DEFPROCr:com$="":fn$="Cursor":PROC
h:ENDPROC
 1820 DEFPROCm(c$):com$=c$:PROCh:ENDPROC
 1830 DEFFNi(c$):PROCm(c$):VDU4:INPUT TA
B(LEN(c$)+1,1) p:VDU5:=p
 1840 DEFFNc(c$):PROCm(c$):VDU4:INPUT TA
B(LEN(c$)+1,1) p$:VDU5:=p$
 1850 :
 1860 DEFPROCaccel
 1870 count%=count%+1
 1880 IF count%<61 s%=4+(count%*count% D
IV 256)*4 ELSE s%=256
 1890 TIME=0:ENDPROC
 1900 :
 1910 DEFPROCleft:x=x-s%*((x>=s%) OR mar
g%)/((s%=4)-1)+(x-1)*((x<s%) AND NOT mar
g%):PLOT4,x,y:ENDPROC
 1920 DEFPROCright:x=x+s%*((x<lx%-s%) OR
 marg%)/((s%=4)-1)-(lx%-x-1)*((x>=lx%-s%
) AND NOT marg%):PLOT4,x,y:ENDPROC
 1930 DEFPROCdown:y=y+s%*((y>=s%) OR mar
g%)+(y-2)*((y<s%) AND NOT marg%):PLOT4,x
,y:ENDPROC
 1940 DEFPROCup:y=y-s%*((y<ly%-s%) OR ma
rg%)-(ly%-y-2)*((y>=ly%-s%) AND NOT marg
%):PLOT4,x,y:ENDPROC
 1950 DEFPROCf
 1960 ON J% PROCswitch,PROCareamove,PROC
mix,PROCshrink,PROCn,PROCn,PROCn,PROCn,P
ROCborder,PROCdelete,PROCn,PROCcpy ELSE
ENDPROC
 1970 ENDPROC
 1980 :
```

```
 1990 DEFPROCsf
 2000 ON J%-16 PROCn,PROCcopymove,PROCov
erlay,PROCn,PROCn,PROCn,PROCn,PROCn,PROC
n,PROCorigin ELSE ENDPROC
 2010 ENDPROC
 2020 :
 2030 DEFPROCcf
 2040 ON J%-32 PROCdump,PROCn,PROCn,PROC
margins,PROCn,PROCn,PROCn,PROClink,PROCs
ave,PROCload ELSE ENDPROC
 2050 ENDPROC
 2060 :
 2070 DEF PROCbox
 2080 PLOT4,x1,0:PLOT6,x1,956:PLOT4,0,y1
:PLOT6,1276,y1
 2090 PLOT4,x2,0:PLOT6,x2,956:PLOT4,0,y2
:PLOT6,1276,y2
 2100 PLOT4,x,y
 2110 ENDPROC
 2120 :
 2130 DEFPROCcursor
 2140 FOR I%=0 TO 1
 2150 *FX19
 2160 PLOT4,x,y+15:PLOT6,x,y-15:PLOT4,x-
15,y:PLOT6,x+15,y
 2170 NEXT I%
 2180 PLOT4,x,y
 2190 ENDPROC
 2200 :
 2210 DEFPROChome
 2220 IF x4=0 AND y4=0 THEN x=lx%/2+1:y=
ly%/2+2 ELSE x=x4:y=y4
 2230 J%=13:ENDPROC
 2240 :
 2250 DEFPROCruler:IF J%<13 OR J%>18 END
PROC
 2260 IF J%=17 ENDPROC
 2270 VDU4:X=x1-x:Y=y1-y
 2280 PRINT TAB(63,0) (x-x4) TAB(74,0) (
y-y4);
 2290 TIME=0:VDU 5:ENDPROC
 2300 :
 2310 DEFPROCswitch:IF J%<>1 ENDPROC
 2320 IF main% main%=FALSE:main$="SHADOW
":PROCshad ELSE main%=TRUE:main$="MAIN"+
c$:PROCmain
 2330 PROCr:ENDPROC
 2340 :
 2350 DEFPROCmain
 2360 VDU19,0,4|19,1,7|
 2370 *FX112,1
 2380 *FX113,1
 2390 ENDPROC
 2400 :
 2410 DEFPROCshad
 2420 VDU19,0,7|19,1,0|
 2430 *FX112,2
 2440 *FX113,2
 2450 ENDPROC
 2460 :
 2470 DEFPROCareamove:IF J%<>2 ENDPROC
 2480 IF copy% fn$="Arcopy" ELSE fn$="Ar
Move"
 2490 PROCh
 2500 PROCm("Area "+copy$)
 2510 MOVE x2,y2:MOVE x1,y1:PLOT188+area
code%,x,y
 2520 x2=x+ABS(x2-x1):y2=y+ABS(y2-y1):x1
=x:y1=y
 2530 PROCr:ENDPROC
 2540 :
 2550 DEFPROCmix:IF J%<>3 ENDPROC
 2560 fn$="Mix    ":a=FNi("Rotation angle
?"):IF ABS(a)>360.9 PROCr:ENDPROC
 2570 J%=1:PROCswitch
 2580 ca=COS(RAD(a)):sa=SIN(RAD(a))
 2590 PROCcentre
 2600 IF owrt% PROCdel:PROCm("Overwritin
g screen") ELSE PROCm("Overlaying screen
")
 2610 IF main% PROCvdus ELSE PROCvdum
 2620 FOR X=X1 TO X2 STEP 2
 2630 FOR Y=Y1 TO Y2 STEP 2-2*(ABS(sa)<0
.71)
 2640 IF POINT(X,Y) PROCpnt
 2650 NEXT Y:NEXT X
 2660 IF main% PROCvdum ELSE PROCvdus
 2670 PROCr:ENDPROC
 2680 :
 2690 DEFPROCdel
 2700 Yv=Y2-Y1:X1d=X1-Xc:Yd=Y1-Yc:X2d=X2
-Xc
 2710 MOVE X1d*ca-Yd*sa+Xc,Yd*ca+X1d*sa+
Yc
 2720 PLOT0,-Yv*sa,Yv*ca
 2730 PLOT87,X2d*ca-Yd*sa+Xc,Yd*ca+X2d*s
a+Yc
 2740 PLOT83,-Yv*sa,Yv*ca
 2750 ENDPROC
 2760 :
 2770 DEFPROCpnt
 2780 IF main% PROCvdum ELSE PROCvdus
 2790 IF a=0 PLOT69,X,Y ELSE IF a=180 PL
OT69,Xe-X,Ye-Y ELSE Xd=X-Xc:Yd=Y-Yc:PLOT
69,Xd*ca-Yd*sa+Xc,Yd*ca+Xd*sa+Yc
 2800 IF main% PROCvdus ELSE PROCvdum
 2810 ENDPROC
 2820 :
 2830 DEFPROCvdum
 2840 *FX112,1
 2850 ENDPROC
 2860 :
```

```
 2870 DEFPROCvdus
 2880 *FX112,2
 2890 ENDPROC
 2900 :
 2910 DEFPROCshrink:IF J%<>4 ENDPROC
 2920 fn$="Shrink":q$=FNc("Anamorphic? (
X/Y)")
 2930 shr=FNi("Factor?"):IF shr=0 PROCr:
ENDPROC
 2940 shrx=shr-(1-shr)*(q$="Y" OR q$="y"
):shry=shr-(1-shr)*(q$="X" OR q$="x")
 2950 J%=1:PROCswitch
 2960 PROCcentre
 2970 x2=(X2-X1)*shrx+X1:y2=(Y2-Y1)*shry
+Y1
 2980 x1=X1:y1=Y1:x=X1:y=Y1
 2990 J%=10:PROCdelete
 3000 IF shr<1 PROCm("Shrinking") ELSE P
ROCm("Expanding")
 3010 IF main% PROCvdus ELSE PROCvdum
 3020 FOR X=X1 TO X2 STEP 2
 3030 FOR Y=Y1 TO Y2 STEP 4
 3040 IF POINT(X,Y) PROCshr
 3050 NEXT Y:NEXT X
 3060 IF main% PROCvdum ELSE PROCvdus
 3070 PROCr:ENDPROC
 3080 :
 3090 DEFPROCshr
 3100 IF main% PROCvdum ELSE PROCvdus
 3110 PLOT69,(X-x1)*shrx+x1,(Y-y1)*shry+
y1
 3120 IF main% PROCvdus ELSE PROCvdum
 3130 ENDPROC
 3140 :
 3150 DEFPROCborder:IF J%<>9 ENDPROC
 3160 fn$="Border":PROCh
 3170 PROCcentre
 3180 MOVE-2,0:PLOT103,X1-2,956
 3190 MOVE X2+2,0:PLOT103,1280,956
 3200 MOVE0,-4:PLOT103,1278,Y1-4
 3210 MOVE0,Y2+4:PLOT103,1278,960
 3220 PROCr:ENDPROC
 3230 ENDPROC
 3240 :
 3250 DEFPROCcentre
 3260 IF x1<x2 X1=x1:X2=x2 ELSE X1=x2:X2
=x1
 3270 IF y1<y2 Y1=y1:Y2=y2 ELSE Y1=y2:Y2
=y1
 3280 Xe=X1+X2:Ye=Y1+Y2:Xc=Xe/2:Yc=Ye/2
 3290 ENDPROC
 3300 :
 3310 DEFPROCdelete:IF J%<>10 ENDPROC
 3320 fn$="DELETE":PROCh
 3330 MOVE x1,y1:PLOT103,x2,y2
 3340 PROCr:ENDPROC
 3350 :
 3360 DEFPROCcpy
 3370 IF J%<>12 ENDPROC
 3380 x2=x:y2=y:ENDPROC
 3390 :
 3400 DEFPROCcopymove:IF J%<>18 ENDPROC
 3410 IF copy% copy%=FALSE:copy$="move":
areacode%=0 ELSE copy%=TRUE:copy$="copy"
:areacode%=2
 3420 PROCr:ENDPROC
 3430 :
 3440 DEFPROCoverlay:IF J%<>19 ENDPROC
 3450 IF owrt% owrt%=FALSE:owrt$="Olay"
ELSE owrt%=TRUE:owrt$="Owrt"
 3460 PROCr:ENDPROC
 3470 ENDPROC
 3480 :
 3490 DEFPROCorigin:IF J%<>26 ENDPROC
 3500 IF x4=0 AND y4=0 THEN x4=x:y4=y:or
ig$="Rel " ELSE x4=0:y4=0:orig$="Abs "
 3510 J%=13:PROCr:ENDPROC
 3520 :
 3530 DEFPROCdump:IF J%<>33 ENDPROC
 3540 fn$="Dump  ":PROCh
 3550 IF NOT main%:*BPRINT P5 F I W2 L10
0
 3560 IF main%:*BPRINT P5 F W2 L100
 3570 ENDPROC
 3580 :
 3590 DEFPROCmargins:IF J%<>36 ENDPROC
 3600 IF marg% marg%=FALSE:marg$="Edge"
ELSE marg%=TRUE:marg$="MarR"
 3610 PROCr:ENDPROC
 3620 ENDPROC
 3630 :
 3640 DEF PROClink:IF J%<>40 ENDPROC
 3650 IF main% main%=FALSE:main$="SHDW":
PROCshad
 3660 HIMEM=&7FFF:fn$="Link  "
 3670 *FX229,0
 3680 VDU4:J%=&7FFFFF
 3690 PROCm("Chaining to ASTAAD")
 3700 CHAIN "ASTAAD3":ENDPROC
 3710 :
 3720 DEFPROCquit:PROCm("Quitting"):quit
%=TRUE:CRASH:END
 3730 DEFPROCn:PROCm("No function"):PROC
d:PROCr:ENDPROC
 3740 :
 3750 DEFPROCd:T%=TIME
 3760 REPEAT UNTIL TIME>T%+150 OR ADVAL-
1
 3770 ENDPROC
```

The Comms Spot

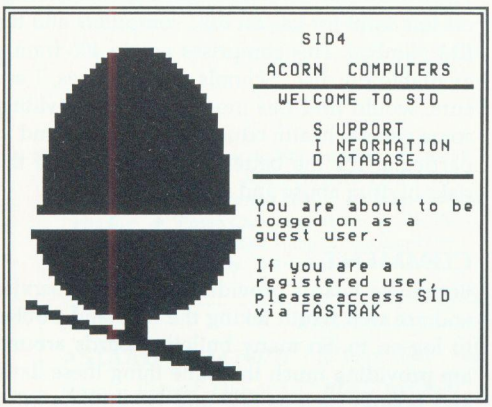
by Peter Rochford

In this month's Comms Spot, we get away from the technical jargon of the last two articles and take a detailed look at a couple of bulletin boards.

Bulletin boards are numerous in the UK now, as a quick look at the ever-growing list published in BEEBUG will confirm. Many of these boards provide very similar features and facilities. Amongst them though are some specialist interest boards, and I am going to take a look at two of these.

SID

This board, with the rather odd name, is owned and run by Acorn Computers (SID stands for Support Information Database). It will therefore be of particular interest to BEEBUG and RISC User members. It has been in operation for quite some time now, but strangely does not seem to get strongly promoted by Acorn itself.



SID's Title Page

The board operates from Acorn's headquarters in Cambridge, and can be dialled up on (0223) 243642. It is a viewdata board, and you will need Prestel type terminal software to access it along with a 1200/75 baud modem. Incidentally, just like Prestel, pages can be accessed by numbers or with keywords.

Access to the board is open to everyone, and no special password is needed. There is the facility to become a registered user with extra benefits, but more about that later.

As one may expect, the purpose of the board is to promote Acorn products and to provide information for current and potential users of Acorn computers.

Guest users have access only to a limited number of facilities and information on the system, but there is still plenty of interest. There are sections on Acorn products, news from Acorn, a dealers list with a search facility and an on-line magazine called El Sid. El Sid has several sections, including Commbase for comms, educational computing and both books and games reviews.

Registered users of the system have access to a lot more. They are issued with a mailbox number for sending Email to other users and can send technical and sales enquiries to Acorn direct via the system. A notice board for hints and tips is also available to promote the exchange of information between users of SID, along with a general bulletin board section.

As a registered user you also have access to the telesoftware section. This includes technical information in the form of downloadable text files.

To register on SID you will have to log on and fill in a response frame. Details will then be sent to you by post. Apart from the extra facilities already mentioned, being a registered user enables you to log on at local call rates via a service called Fastrak which costs 8p per

minute. There is no subscription charge, but you will be charged a minimum of £10 plus VAT per quarter on the line time incurred.

Having had a good look around SID, I would say that it is a very worthwhile service. It is both interesting and informative and should prove beneficial to many users of Acorn computers. There is scope for plenty more to be included yet, and it is hinted that Acorn are going to expand the system. This, by the way, includes the setting up of closed-user groups for specialist interest areas.

HEALTHDATA

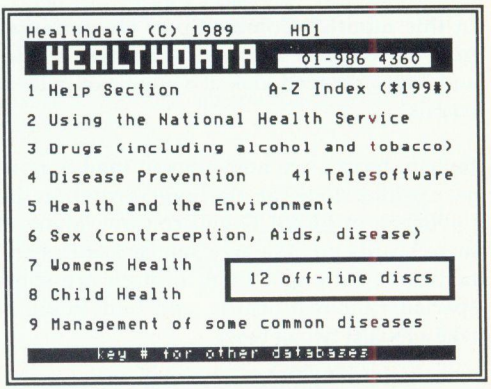
This board has been up and running since as long ago as 1985. It was set up by a general practitioner to provide medical information and advice to anyone who needs it. The system is viewdata based, and to use it you will again need Prestel type terminal software and a 1200/75 baud modem.

Access to the system is open to all by dialling 01-986 4360, which is a London number. No password is required and there is no need to become a registered user. In my experience, this board is very popular, and you may well find it often engaged during the evenings.

The content of the board covers a very wide variety of health topics - everything from coping with the common cold to information on the risk of AIDS. The topics are not covered in great detail but are certainly informative and useful, and there are always suggestions on how to get further advice.

Healthdata is not, I should point out, an electronic version of the 'Family Doctor' book, and is certainly not for hypochondriacs. It gives good down-to-earth advice even though at times it can be a bit patronising.

The amount of information on the system is quite surprising, and once on-line you may find you have spent longer than you thought. You keep on finding topics that you always wanted to know a bit more about, and the whole thing becomes quite absorbing. It is certainly an interesting board to just browse through.



Healthdata's Main Menu

I feel that this board provides a very useful service and full marks to Chris Dobbing, the Healthdata sysop for putting it together in such a well-written and well-presented way. It must have involved a great deal of work.

The contents of the whole board are available in off-line form for use on BBC computers and the RM Nimbus. This comprises nearly 400 frames available on disc. Schools and colleges, I am sure, would find this most useful in providing some general health education for pupils and in particular for the better understanding of the risks of drug abuse and AIDS.

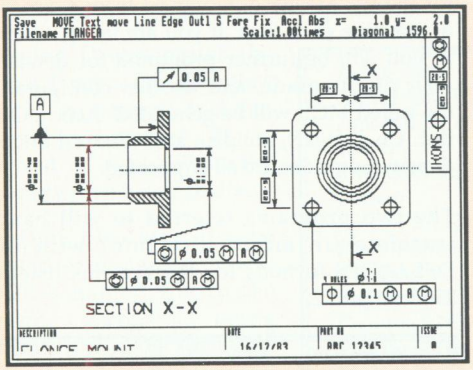
COMMENT

Both these boards provide a very useful service and are well worth taking the time and trouble to log-on to. So many bulletin boards around are providing much the same thing these days, and boards such as SID and Healthdata are a welcome change. I hope that other BB's will emerge offering similar types of service to Healthdata on other subjects. It is a question, of course, of people with the right interest, finding the time and money to set up and maintain the system. B

ASTAAD

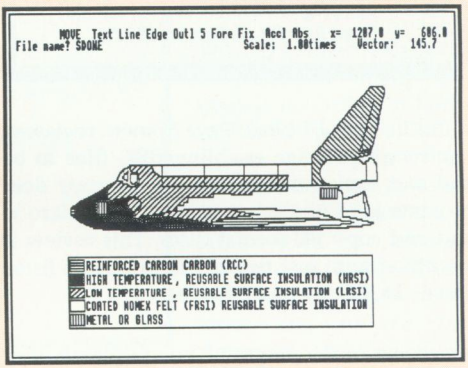
The Best of BEEBUG

Our new BEST OF BEEBUG disc is based on the recent series of articles and programs dealing with ASTAAD, our comprehensive CAD program for the Master and Compact. The ASTAAD disc contains everything you need to use this very impressive piece of software, including the enhanced versions of the ASTAAD and STEAMS programs. This is supplied complete with documentation and printed keystrips for use with ASTAAD and STEAMS.



ASTAAD £9.95

* Enhanced ASTAAD CAD program now including full mouse and joystick control, built-in printer dump and speed improvement.
* STEAMS image manipulator
* 8 page manual covering the use of the system.
* Keystrips for ASTAAD and STEAMS
* Sample picture files to experiment with.



Recent magazines have shown many examples of the truly impressive results that can be achieved with ASTAAD for producing technical drawings and diagrams including perspective work. The disc also contains several example drawings for you to load and modify, for easy familiarisation with the software. All this is at a fraction of the price that you would need to pay for equivalent commercial software.

Our first two releases under the Best of BEEBUG label are still available:

General Utilities £5.75
Best of BEEBUG Disc 1

* Printer Buffer * * ROM Controller * Sprite Editor/Animator * Multi-Character Printer Driver for View * Mode 7 Screen Editor * Multi-Column Printing * Epson Character Definer * BEEBUG MiniWimp † * ROM Filing System Generator

*** Master series only. † Requires sideways RAM.**

Applications £5.75
Best of BEEBUG Disc 2

* Business Graphics * Video Cataloguer
* World by Night and Day * Phone Book
* Page Designer * Personalised Letter-Heads
* Mapping the British Isles * Selective Breeding
* Appointments Diary * The Earth from Space
* Personalised Address Book

Please rush me my Best of BEEBUG discs:

ASTAAD Code 1407A *(80 track DFS)* ❑
(State if 40 track DFS or 3.5" ADFS required)

General Utilities Disc Code 1605A ❑

Applications Disc Code 1604A ❑

Name _____

Address _____

_____

Memership No _____

Total £_____

Postage (£0.60 + £0.30 for every other disc) £_____

Grand Total £_____

I enclose a cheque for £_____ OR please debit my Access, Visa or Connect account, Card No_____/_____/_____/_____ Expiry____/____ Signed_____

Return to BEEBUG Ltd, Dolphin Place, Holywell Hill, St Albans, Herts AL1 1EX. Telephone (0727) 40303.

PC to BBC File Transfers

Bernard Hickman reviews a piece of software which solves his PC to BBC file transfer problems.

Title	**Discopy**
Format	**DOSCopy (BBC and MS-DOS)** **CPMBeeb (BBC and CP/M)** **CPMCopy (CP/M and MS-DOS)** **TorchCopy (Torch CPN/MCP and MS/DOS)**
Supplier	**BAKsoft** **20 Leys Avenue,** **Cambridge CB4 2AW.**
Price	**Any combination supplied on one disc: one program £19.50; two for £34.50; three £49.50; all four for £59.50.**

In BEEBUG Vol.7 No.2 Dave Somers reviewed a software package enabling BBC files to be read and copied on an IBM PC. Discopy does the same kind of job but using a BBC micro to read and copy PC format discs. This review is concerned with just two of the programs listed above - DOSCopy and CPMBeeb.

Whichever of the programs you purchase, they come on one DFS disc and share a common menu system. Since this is a dual-format 40/80 disc, it cannot be backed up. However, a file BACK is thoughtfully included to enable you to make a working copy. This latter can then be copied to an ADFS if you are using that filing system.

The programs will run on a B+, Master or Compact, and on a Model B equipped with the 1770 disc controller (available as an upgrade). To maximise the options, an 80-track double-sided drive is needed, and preferably dual drives to obviate the need for disc-swapping.

When the program is first booted, it prompts for details of your drive configuration, which it then stores in a file called TRACKS. If you subsequently change the drive equipment, you simply generate a new TRACKS file.

This done, the program detects which model it is running in, and loads the appropriate routine for the B+, Master or Compact from a selection of machine code files. If you are using a Model B, you will be further prompted for details of your disc upgrade, and another configuration file called MCB will be generated. Acorn, Opus (inc. Challenger), Solidisk and Watford double-density upgrades are all supported.

The two programs referred to will handle transfers (in either direction) with BBC DFS/ADFS formats for the systems listed in Tables 1 and 2.

Acorn 640k,	800k
Apricot	315k, 720k
Atari ST	360k, 720k
Duet	720k
IBM PC	160k, 320k, 180k, 360k
IBM PS/2	720k
MacDougall	
NEC	
Nimbus	720k
Olivetti	720k A,
Olivetti	720k B
Rainbow	400k

Table 1. Accepted MS-DOS formats

The program uses an ASCII file called DISCDEF which contains a list of formats with the relevant format data for each. The structure of the file is fully explained in the documentation and it can be edited to add further formats (e.g. with the Master Editor or a word processor). BAKsoft offer to help with the necessary information if you supply a sample disc of whatever strange format you wish to read.

Note, however, that IBM 1.2 Mbyte high density, Sirius, Apple (inc. Macintosh) and Amiga cannot be accommodated, a limitation of the Beeb's 1770 disc controller rather than this software.

Once configured for your particular hardware, the program could scarcely be simpler to use. The two formats to be used are selected from the screen menu, e.g. BBC for drive 0, IBM 360K for drive 1, and the work discs inserted. This done, a menu of commands is displayed, which can be returned to at any time by pressing Escape, which will also abort any operation.

Pressing D catalogues the MS-DOS disc. The directory is held in memory and displayed whenever D is pressed. N will reload the directory if you change discs.

MS-DOS files may also be deleted or re-named. T will TYPE a named file to the screen, scrolling being controlled with the space bar. Hex dumps can also be produced. If the program was entered while in an 80-column mode, then the screen output will be in 80-column format (provided your machine has shadow RAM for the screen display).

Acorn Z80
Alphatronic PC
Amstrad SS, DS
CDOS
Cifer 2683 40tr, Cifer 2683 80tr
Comart
Epson PX8, QX10
IBM CP/M-86
ICL Quattro
Kaypro S/S,D/S
Osborne SS SD, Osborne SS DD
Rair Black Box
RML 380Z SD, 480Z DD
Sharp MZ80B
SM1
Superbrain QD
Tatung TPC-2000
Televideo
Transtec Krypton
Zorba

Table 2. Accepted CP/M formats

F will format an MS-DOS disc. If this is to be read on a PC, it is advisable to use a new, unused, disc, or one that has been "unformatted", e.g. with Advanced Disc Investigator or a similar utility.

P toggles the printer on and off, enabling screen output to be printed.

The main item, of course, is C for copy. This produces prompts for the direction of transfer (from/to BBC) and the filenames to be used. MS/DOS filenames must be given in the form shown in the directory, e.g. CHAPT3.ASC, but any spaces may be omitted.

Copying then proceeds, a 20K text file taking about 35 seconds. Wildcards, alone or with partial filenames, may be very usefully employed. If you reply to the prompts as follows:

```
MS-DOS Filename: *.*
BBC filename: *
```

then an entire MS-DOS disc will be copied to BBC format without further intervention. The BBC filenames will be truncated to seven or ten characters as necessary.

Also from the menu, * commands will be applied directly to the BBC disc, e.g: *CAT *DRIVE *DIR *ACCESS *RENAME *DELETE *INFO *EX. However, *COPY *BACKUP and *COMPACT should not be attempted from within the program.

The CPMBeeb routine is very similar, except that there is no facility to format a CP/M disc.

As with other programs of this type, the Beeb cannot cope with a "foreign" directory structure. Therefore files can only be written from and to the root directory on MS-DOS and CP/M discs. Text files are best supplied in pure ASCII format. Most word processors have a facility to produce files in this form.

Problems can arise with line-endings. Whereas the Beeb uses RET or LF/RET, MS-DOS and CP/M commonly use RET/LF. BAKsoft include a Basic program CONVERT to sort this out for you and also remove unwanted parity bits.

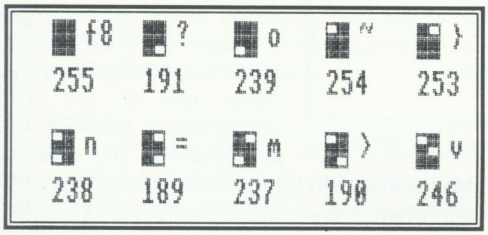
I would emphasize that I have so far only used these two programs regularly on a Master 128, with text files (probably the most common use for programs of this type), and with a selection of the disc formats listed above. However, in this context it has given complete satisfaction, and in my opinion offers excellent "user-friendliness" and value for money. B

Mode 7 Graphics Characters Displayed

If you ever have difficulty finding the right mode 7 graphics character, Paul Hogarth's short utility may be just the thing you need.

Although a mode 7 teletext editor, such as that published in BEEBUG Vol.6 No.3 is very useful, it is often difficult to locate (in the User Guide or elsewhere) just the character you are looking for, as they are usually arranged in numerical code order, rather than in any logical graphic order.

This program will display all the mode 7 graphics characters in a much more useful way, and if you have a suitable printer dump ROM installed, the program will also print a copy for future reference. The display also shows the code value and corresponding lower-case character for each teletext graphics character. The order in which the characters is displayed is determined quite simply by the order of their code values in the DATA statements from line 9000 onwards. A blank position in the display is produced by including a code value of 100. If necessary, just edit the DATA statements to display the characters in a different order.



To use the program, just type it in and run it (but do save it first for safety). If you don't have a printer ROM, delete line 190. The program as listed will work with BEEBUG's Dumpmaster, and this line is also the one to change if you have a different print ROM (or other than Epson FX80 compatible printer).

```
   10 REM Program GRAPH 7
   20 REM Version B1.4
   30 REM Author  Paul Hogarth
   40 REM BEEBUG  Jan/Feb 1989
   50 REM Program subject to copyright
   60 :
  100 MODE 0
  110 ON ERROR GOTO 220
  120 PRINTTAB(22,0)"MODE 7 GRAPHICS CHA
RACTERS"
  130 PRINTTAB(23,2)"For BEEBUG Screen e
ditor"
  140 VDU5
  150 FOR x%=61 TO 1160 STEP 100
  160 FOR y%=840 TO 120 STEP -120
  170 READ c%:PROCblock:PROCfillA
  180 NEXT y%,x%
  190 *BPRINT V
  200 END
  210 :
  220 MODE3:REPORT:PRINT" at line ";ERL
  230 END
  240 :
 1000 DEF PROCblock:LOCAL z%
 1010 IF c%=100 THEN ENDPROC
 1020 MOVEx%+36,y%+36
 1030 IF c%=255 PRINT"f8" ELSE VDU ((c%-
128)-61*(c%=163)+(c%=224))
 1040 FOR z%=0 TO 24 STEP 12
 1050 MOVEx%+z%,y%
 1060 DRAWx%+z%,y%+36
 1070 NEXT
 1080 FOR z%=0 TO 36 STEP 12
 1090 MOVEx%,y%+z%
 1100 DRAWx%+24,y%+z%
 1110 NEXT
 1120 MOVEx%,y%-16:PRINT STR$(c%)
 1130 ENDPROC
 1145 :
 1200 DEF PROCfillA
 1210 IF c%>=224 PROCfillB(x%+16,y%+4):c
%=c%-64
 1220 IF c%>=176 PROCfillB(x%+8,y%+4):c%
=c%-16
 1230 IF c%>=168 PROCfillB(x%+16,y%+16):
c%=c%-8
 1240 IF c%>=164 PROCfillB(x%+8,y%+16):c
%=c%-4
 1250 IF c%>=162 PROCfillB(x%+16,y%+28):
c%=c%-2
 1260 IF c%>=161 PROCfillB(x%+8,y%+28)
 1270 ENDPROC
 1280 :
 1400 DEF PROCfillB(x,y)
 1410 LOCAL y1
 1420 FOR y1=y TO y+4 STEP 4
 1430 PLOT77,x,y1
 1440 NEXT
 1450 ENDPROC
 1460 :
 9000 DATA 255,238,230,252,175,243,176
 9010 DATA 191,189,229,188,167,241,164
 9020 DATA 239,237,186,236,171,242,161
 9030 DATA 254,190,182,248,174,227,100
 9040 DATA 253,246,233,244,173,179,224
 9050 DATA 247,249,100,232,170,177,168
 9060 DATA 251,187,100,180,165,178,162
 9070 DATA 183,231,100,228,166,225,100
 9080 DATA 235,181,100,184,169,226,240
 9090 DATA 250,234,100,100,100,100,172
 9100 DATA 245,185,100,100,100,100,163
```

The Ins and Outs of Basic (Part 2)

This month Mike Williams looks at the use of the pseudo variable @% in print formatting, and examines some of the alternatives to PRINT and INPUT.

In dealing with methods of input and output, I started last month with the obvious INPUT and PRINT functions in Basic. I described how both numbers and strings are formatted in fields controlled partly by the use of the comma or semi-colon as a separator.

Although by default each field in the display is 10 characters in width, this can be changed by using the pseudo variable @%, which can also determine the format in which numbers will appear. Although it may seem more of a hindrance than a help, it really is better to deal with the value of @% in hexadecimal. This value then consists of 5 digits, the first selecting one of three different formats, the next two the number of decimal places (or digits) to be shown, and the last two digits the size of the field (see figure 1). The default is &0090A (&00A0A in Basic I), or more simply &90A. This indicates a 'normal' format with a field width of 10 (&0A in hex).

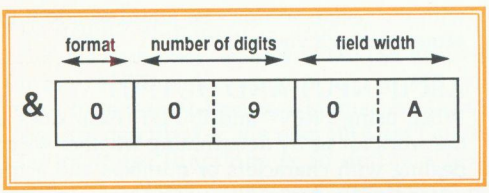


Figure 1. Composition of @% in hex (showing default value)

The three formats determined by the first digit are:

0 'normal' format
1 exponential form
2 fixed number of decimal places

In the so-called *normal* format, numbers are displayed in full, if there is room to do so within the current field width. If the accurate representation would require more digits than can be accommodated, then exponential form is used. This is often called scientific notation because of its use by scientists and engineers. It consists of a number, usually with a fractional part called the mantissa, multiplied by a power of 10 (the exponent). For example, the number 123.456 in exponential form becomes 1.23456E2 meaning 1.23456 multiplied by 10^2.

Consider the following short program:

```
100 FOR I=1 TO 100
110 PRINT I^5;
120 NEXT
130 PRINT
140 END
```

If you run this you will find the resulting numbers neatly lined up in columns until the point is reached where Basic is forced into using exponents. Once this happens the numbers no longer fit the default field width of 10.

Remember that unless we have changed it, @% will have the value &90A. The '0A' (in hex) indicates a field width of 10. The figure 9 determines the number of significant figures. So all our exponential numbers have 9 digits plus a decimal point plus the exponent (2 or 3 characters depending on size), giving a total of 12 in the output from our example program. If Basic cannot fit a number within a field as defined, then it allows the number to overflow in the interests of not losing information.

If you type:

```
@%=&914
```

before running the program again, you should find all the numbers now fit their fields whether or not exponential form is used. This is because the '14' (in hex) gives a field width of 20 (in decimal) which is now ample. And because the number of characters per line is an exact multiple of 20 (in either a 40 or 80 column mode), the numbers line up neatly in columns. If you choose a sufficiently large field width, but one that does not easily fit into the line length, the numbers will all fit individually, but will no longer be in neat columns.

The solution to the above problem is to add an extra line to the program. For example, suppose

we first set @%=&90F in an 80 column mode. The field width indicated by &0F is 15 in decimal, and we can fit five such complete fields on one 80 column line. So add the following line to the program:

```
115 IF I MOD 5=0 THEN PRINT
```

and once again all the numbers should line up in columns. We have simply forced Basic to execute CR/LF (Carriage Return/Line Feed) after every fifth number is output.

Now change @% to &60F and run the program again. All should be as before, except that where numbers are displayed in exponential form, only six significant figures are given, not 9 as before.

Now it's all very well leaving Basic to decide whether numbers should be displayed in exponential form or not, but often tables look better if all numbers follow the same format. Up to now a *normal* format has been assumed (code 0). To see all the numbers displayed in exponential form set @%=1060F. The first digit (the '1') determines the format of the display, the '06' determines the number of significant figures, and the '0F' the field width, the latter two as before. Try this and run the program again to convince yourself. All the numbers will now be displayed in exponential form.

Now try setting @%=&2020F and run the program once more. The initial '2' indicates that format 2 is to be used, working to a fixed number of decimal places. The next two digits (the '02') indicate the number of fractional figures to be displayed. The field width is as before. You should find that all starts off as expected, but that after several rows of figures the decimal point and any following figures simply disappear. Once again, the number has become too large for Basic to display it in the form requested and within the specified field width.

Format 2 is particularly useful when dealing with money sums, by setting the format to show just two decimal places. Where necessary, Basic rounds numbers up, but do remember that most real numbers (those with a fractional part) cannot be represented in any digital computer with 100% accuracy. BBC Basic works to a maximum of nine significant figures. We can conveniently package up the use of format 2 in a simple procedure.

```
1000 DEF PROCprint(n,width%,ndec%)
1010 LOCAL @%
1020 @%=&20000+&100*width%+ndec%
1030 PRINT n;
1040 ENDPROC
```

This will print any value 'n' with the specified number of decimal places and field width. Note that by making @% local to the procedure, any other setting of @% in the calling program is unaffected.

A minor consequence of using formats 1 or 2 is that any line numbers referenced in error messages will also be shown in the same format. It may look odd, but there is little you can do about it, apart from resetting @% within any error handling routine of your own. Also bear in mind that the setting of @% determines the format of numbers converted to string format using STR$.

I hope that this has given enough insight into the purpose of @%. However, even experienced programmers usually end up trying various values for this variable before they are satisfied with the result, so don't be afraid to adopt the same approach yourself.

ASCII INPUT AND OUTPUT

When using PRINT and INPUT, Basic does all the work for us by determining whether we are dealing with characters or numbers and acting accordingly. Now although this is all very obvious and highly useful, there are circumstances where these keywords are cumbersome, or even quite incapable of dealing with the requirements in hand.

The alternative to the use of PRINT and INPUT, which deal with strings of characters and whole numbers, is to use VDU for output and either GET or GET$ for input. Whichever is used, only one character at a time can be handled, and that is mostly dealt with in terms of an ASCII code rather than as a character.

For example, suppose in mode 7 we want to display text in blue against a cyan background. The teletext codes to do this are 134, 157, and

132, and these need to be positioned as the first three characters on each line of the display. One way of writing this would be:

```
100 FOR I=1 TO 24
110 VDU 134,157,132,13,10
120 NEXT
130 VDU 28,3,24,39,0
```

You should be able to see one immediate difference between the use of VDU and PRINT. PRINT always causes CR/LF to be output after any text or numbers, unless inhibited by a semi-colon. On the other hand, VDU only outputs precisely what you specify, so CR/LF must be explicitly included, hence the codes 13 and 10 (the ASCII codes for Return and Linefeed) at the end of the VDU sequence in line 110.

As a comparison, we can write the same routine using PRINT:

```
100 FOR I=1 TO 100
110 PRINT CHR$(134);CHR$(157);CHR$(132)
120 NEXT
130 VDU 28,3,24,39,0
```

Note the use of the CHR$ function. PRINT can only output strings of characters; VDU outputs codes. In both cases line 130 is included to redefine the text window to protect the three codes down the side of the screen. That in turn illustrates another advantage of using VDU.

The PRINT statement is used largely with printable characters, whereas VDU can be used for any ASCII code, printable or not. All the codes from 0 to 31 have special meanings, and these are described in the relevant User Guide. One of these functions is to define a text window, as above. In addition, ASCII code 12 is equivalent to CLS, clear text area to background colour. So to define a text window and clear this we could write:

```
VDU 28,0,20,79,5,12
```

The sequence 0, 20, 79, 5 defines a window, which is then cleared with the 12 at the end.

In the same way that VDU corresponds to PRINT, so GET (or GET$) corresponds to INPUT. GET returns the ASCII code of a character typed in via the keyboard, while GET$ returns the character. In neither case is the character echoed on the screen (very useful for entering passwords). In the examples which follow we shall concentrate on GET but sometimes we will have to convert the code to a character with the CHR$ function. We could in most cases use GET$, and then convert the character to its ASCII code when needed using the ASC function. My preference is to use GET.

For example, you will often see in many programs a line like:

```
PRINT "Press any key to continue":G=GET
```

As soon as any key is pressed, Basic will continue to the next line of the program, but nothing is displayed on the screen.

A line like G=GET can also be useful when debugging Basic programs - just include such a line at any point where you want to pause the program. The program will then wait when that point is reached until any key is pressed, and then continue. For greater assistance write:

```
PRINT"n";:G=GET
```

where n is the line number of this extra line of Basic. Not only will the program pause but will identify which pause line has occurred.

Using GET can also be useful when we want to exercise more control over input than can be provided by INPUT. For example, the function below would restrict input to those characters specified in the string accept$.

```
DEF FNinput1(accept$)
LOCAL G,p$:p$=""
REPEAT
G=GET
IF G=127 AND LEN(p$)>0 THEN
   VDU8,32,8:p$=LEFT$(p$,LEN(p$)-1)
IF INSTR(accept$,CHR$(G)) THEN
   p$=p$+CHR$(G):VDU G
UNTIL G=13
=p$
```

This would build up a string of acceptable characters until Return was pressed, but the Return character itself would not be included in the resulting string returned by the function, though this could be changed. A call to this function, such as:

```
data$=FNinput1("0123456789")
```

would ensure that all input consisted of the numeric digits 0-9 only, but note that as coded the function returns a string; use the VAL function to convert this to a number if needed.

Notice also the first IF statement in the above function. This tests for Delete (ASCII 127)

having been pressed. Provided at least one character has already been typed in (the length of p$ greater than zero), the last character is removed from the screen by the VDU sequence which follows, and from the string p$ being created. VDU8,32,8 moves the cursor back one character (so that the cursor is under the character to be deleted), outputs a space (to delete the character, but the cursor will now have been moved on one position again), and moves the cursor back one position ready for the next keypress.

Another variation on an input function is where input is to be restricted to a specific position and field width on the screen. In such cases the input field is often shown on the screen by some marker character (a pad character).

The first IF statement checks for the Delete key as before, but replaces the character to be deleted with the pad character rather than a space. The second IF checks that we have input a printable character, and that we have not already reached the limit of the input field, before adding the character to the input string and echoing it to the screen. This function could be called, for example, as:

```
data$=FNinput2(10,15,12,"-")
```

using the hyphen as the marker character.

```
DEF FNinput2(x,y,w,pad$)
LOCAL G,p$:p$=""
PRINTTAB(x,y) STRING$(w,pad$);
PRINTTAB(x,y);
REPEAT
G=GET
IF G=127 AND LEN(p$)>0 THEN
   VDU8,ASC(pad$),8:
   p$=LEFT$(p$,LEN(p$)-1)
IF G>31 AND G<127 AND LEN(p$)<w THEN
   p$=p$+CHR$(G):VDU G
UNTIL G=13
=p$
```

That is all we have space for this month. If you look at the last two examples you should see that we have intentionally used both PRINT and VDU. In practice, that's not untypical. You need to be aware of both methods, and to mix them judiciously to your advantage. Next time we will look at timed input with the INKEY function. B

Centres of Gravity (continued from page 8)

```
 1310 :
 1320 DEF PROCedit
 1330 LOCAL a
 1340 COLOUR 128:COLOUR 3
 1350 PRINT TAB(12,0)" EDIT "
 1360 IF d%(0,0)=0 ENDPROC
 1370 x=d%(2,0):y=d%(3,0)
 1380 GCOL 3,2:PROCcursor(x,y,16)
 1390 REPEAT:Ox=x:Oy=y:k=FNkeyboard
 1400 PROCcursor(Ox,Oy,16)
 1410 PROCcursor(x,y,16):UNTIL k
 1420 Ax=x:Ay=y:S=0:PROCcursor(x,y,16)
 1430 FOR a=1 TO d%(0,0)
 1440 IF FNdist(Ax,Ay,d%(0,a),d%(1,a))<1
6 T=a:S=1
 1450 IF FNdist(Ax,Ay,d%(2,a),d%(3,a))<1
6 T=a:S=2
 1460 IF FNdist(Ax,Ay,d%(4,a),d%(5,a))<1
6 T=a:S=3
 1470 NEXT
 1480 IF S=1 PROCtriangle(d%(0,T),d%(1,T
),d%(2,T),d%(3,T),d%(4,T),d%(5,T))
 1490 IF S=2 PROCtriangle(d%(2,T),d%(3,T
),d%(4,T),d%(5,T),d%(0,T),d%(1,T))
 1500 IF S=3 PROCtriangle(d%(4,T),d%(5,T
),d%(0,T),d%(1,T),d%(2,T),d%(3,T))
 1510 PROCdisp2
 1520 ENDPROC
 1530 :
 1540 DEF FNdist(Gx,Gy,Hx,Hy):=SQR((Hx-G
x)^2+(Hy-Gy)^2)
 1550 :
 1560 DEFPROCtriangle(Fx,Fy,Gx,Gy,Hx,Hy)
 1570 SOUND1,-15,200,2:GCOL3,2:x=Fx:y=Fy
 1580 PROCcursor(x,y,16)
 1590 REPEAT:Ox=x:Oy=y:k=FNkeyboard
 1600 MOVE Gx,Gy:DRAW Ox,Oy:DRAW Hx,Hy
 1610 PROCcursor(Ox,Oy,16)
 1620 MOVE Gx,Gy
 1630 DRAW x,y:DRAW Hx,Hy
 1640 PROCcursor(x,y,16):UNTIL k
 1650 Fx=x:Fy=y
 1660 d%(0,T)=Fx:d%(1,T)=Fy
 1670 d%(2,T)=Gx:d%(3,T)=Gy
 1680 d%(4,T)=Hx:d%(5,T)=Hy
 1690 ENDPROC                          B
```

512 Forum

Robin Burton takes a look at an MS/DOS implementation of BBC Basic on the 512 co-processor.

BBC micro users have been spoiled. Not only is BBC Basic included with the machine, but it's acknowledged as an excellent implementation.

Many micros have no native language, a problem the BBC user hasn't been 'brought up' to expect. No one feels this more than 512 users. The fact that Acorn did supply BBC Basic with the Z80 only rubs salt into the wound.

Several implementations of Basic work on the 512, but even any resemblance to BBC Basic is virtually incidental. I was interested therefore, to be able to borrow a copy of BBCBasic(86) from M-TEC. This is an MS-DOS version of BBC Basic IV, and during the last month I've been trying it in the 512.

Product	**BBCBasic(86) PLUS**
Supplier	**M-TEC Computer Services UK, The Market Place, Reepham, Norfolk NR10 4JJ.**
Price	**£96.60 inc. VAT and p&p.**

The first point to make clear is that this isn't a BBC package transferred to the 512. No-one should be surprised, but if you did expect an exact replica of BBC Basic implemented on the 512 purely to gain memory or speed, you'd be disappointed.

This isn't a criticism, but users of *real* BBC Basic have expectations (and facilities) that DOS users don't, so let's put things in perspective.

BBCBasic(86) V4.6 is an MS-DOS package which emulates the language that most BBC users cut their teeth on. The fact that, in the 512 with its DOS+ environment and very non-standard hardware it runs well is a credit to the code and its authors (one of these is Richard Russell who wrote the Z80's BBC Basic).

The acid test for a BBC Basic look-alike must be, 'can you transfer programs with little or no change and expect them to work?' The answer here is largely 'Yes'.

The main difference is simple. Remember that in BBCBasic(86) the screen modes are those of an IBM-PC, not a BBC micro, and there are few problems. The fact that a real BBC sits on the other end of the tube is irrelevant. I had to remind myself of this, because so much looks familiar.

IN USE

The software is on a single 360K disc, with over 500 pages of documentation in an A5 ring binder. This is extremely thorough and the information is well presented. Most BBC users will skim through quite large portions of it because so much is similar.

I would have liked the contents list at the front and normal page numbering. Finding items isn't easy - the first job is to find the contents list! It immediately precedes the excellent index, but I'd move the lot to the front of the manual and put dividers between sections to aid searches.

The first job was to make a working disc of the software, which was entirely without incident. The software and numerous example programs are all in a directory 'BBCBasic', so I transferred this to an 800K disc and copied 'COMMAND.COM' to the root directory. No other installation or setting up is needed - just 'CD BBCBasic' and off you go.

There are two ways to run programs. Firstly, you can run 'BBCBasic.EXE' to enter the BBCBasic shell. After this, with a few exceptions, most operations are much as they are in the BBC.

Alternatively, you can type 'BBCBasic <progname>' as a composite DOS+ command. The language is entered and the named program is chained. When it terminates you

remain in the BBCBasic(86) environment. *QUIT exits BBCBasic and returns to the DOS prompt.

LOAD, CHAIN and SAVE are the same as in the BBC micro, except for the filing system. The extension '.BBC' is reserved for Basic programs, but can be omitted from commands.Native DOS commands are preceded by a * to run them. Discs can be catalogued by either '*.', showing .BBC files, or '*DIR' for other extensions as required.

The cursor keys aren't the same as in BBC mode of course. Pressing the TAB key enters edit mode, while repeated presses provide the same facility as COPY. Left and right cursor then behave as you'd expect, while up or down cursor take you to the start or end of a line - very simple.

There's also a Basic editor which *feels* very much like Acorn's. Entering or amending programs by either means is very easy, and no-one could possibly have trouble with these facilities.

The graphics demonstrations provided worked well, and quite quickly using PC screen modes. Teletext emulation is included, with several set-up programs for different PCs, but it requires an EGA which rules out the 512. That said, no-one would use 512 graphics in preference to the BBC's, so it isn't a factor in evaluating the package.

Among the programs supplied are the old PCW benchmarks. I ran these and was informed that my processor ran at 9.44MHz. Naturally I decided to find out how fast BBCBasic(86) programs do run compared with the real thing, and to test program compatibility at the same time.

COMPATIBILITY

Using MOVE I transferred three BBC programs directly to DOS with no changes, not even the names. I then used the supplied program (FCONVERT) to convert from tokenised BBC format to tokenised BBCBasic(86) format. Everything proceeded without a hitch. What's more the resulting programs worked.

There was, I hardly need say, no BBC assembler, graphics or teletext in these programs. They consist of moderately involved arithmetic and/or array juggling with some text output. In-line assembly is supported, but of course BBCBasic(86) doesn't handle 6502 assembler code, but 8086 code instead.

One program finds the highest prime number equal to or below any value up to 9 digits long. Another creates an integer array of 6000 elements and stores randomly ascending values in it. It then offers the choices of finding a value either by a serial search or a recursive dicotomic search (often called a binary-chop) - in which the list is repeatedly divided in half to find the required entry.

The third program creates a similar array, but stores completely random values in it. It then offers three methods of sorting into ascending or descending order before displaying the results in paged mode.

These were experimental programs, written to test the relative speed of various techniques using Cobol on mainframe systems. As such they make extensive use of integer arithmetic, procedures, REPEAT-UNTIL and FOR-NEXT loops and relational/conditional tests including TRUE/FALSE, all of which have direct analogies in Cobol.

I also transferred a program which opens a named file, reads it and filters the data, removing unwanted codes (double spaces, printer codes, tabs etc) creating a new, text only file.

All the programs performed perfectly and immediately, with not a single alteration. Even exits with Escape using ON ERROR and IFERR=17 worked properly, as did DIM, OPENIN, OPENUP, BGET/BPUT, multiple statements, NEXTs with implicit variables and so on.

PERFORMANCE

One unwelcome surprise was that there was no gain in speed. On the contrary, screen output was drastically slower, especially scrolling. I decided to organise some very direct tests of

several aspects to isolate various program functions.

The results are tabulated (see Table 1), but the bottom line is that speed is a real disappointment all round. Apart from disc handling, which of course is quicker (except OPEN/CLOSE) because of the 800K format, almost everything took as long or longer than in my updated (but ancient) series 7 model B.

There's no obvious reason why greater advantage isn't gained from the 512's 10MHz. After all, it's running at five times the clock-rate of the BBC, doing precisely the same job. The tube can't affect RAM or processor functions, and clearly doesn't in fact affect anything much anyway.

To investigate further, three tests were also run on an 8MHz Amstrad PC, with the following results. Test 1 - 263.37 seconds, test 2 - 2.58 seconds and test 5 - 50.00 seconds. These figures are worse than BBCBasic(86) and clearly demonstrate that neither the 512 nor DOS+ is at fault for the slow processor/RAM operations, (quite the opposite in fact) and it doesn't explain slow screen output either. What it does underline is the excellence of the original 6502-based BBC Basic.

2MHz 65C02 model B vs 10MHz 80186 Master 512

	Test	Run Times in seconds Basic IV	BBCB(86)	CB86I	CB86C
	MEMORY/PROCESSOR				
1.	FORI%=1TO 1,000,000.	159.98	177.86	41.35	5.10
2.	FORI%=1TO 10,000	1.61	1.78	5.11	*0.05
3.	DIM A%(6000)+load data	31.83	25.73	7.02	*0.01
4.	Serial search, 6000 items	32.00	26.01	7.20	*0.01
	SCREEN OUTPUT				
5.	PRINT"A": 1,000 times.	6.91	128.14	12.01	11.20
6.	PRINT"A to Z": 1,000 times.	15.54	142.84	26.40	24.23
	DISC FILING				
7.	OPEN:BPUT10,000 bytes:CLOSE	18.36	14.30	**	**
	ADFS	35.93			
8.	OPEN:BPUT1,000 bytes:CLOSE	2.56	6.08		
	ADFS	10.41			
9.	BPUT10,000 bytes (file open)	17.36	8.36		
	ADFS	32.50			
10.	OPEN:CLOSE	1.00	5.94		
	ADFS	3.43			

* CBasic86 - Compiled tests 2 to 4 too fast for reliable timing.
** Sequential/random access to records (and fields) only.
Byte access to files is not supported.

Table 1. Table of comparative run times for BBC Basic, BBCBasic(86), Digital Research CBasic86 I(nterpreted) and C(ompiled).

On balance it's doubtful that BBCBasic(86) would outpace a model B or Master, definitely not if text output to screen is required. Don't think that the 512 isn't fast. The reasons are within the package, but I can't guess what they are.

The tests in BBC Basic were identical for each machine and all were entered as single lines. For tests 7 to 10 two sets of figures are given, the first for single density DFS systems and the second for the ADFS (and marked thus).

Tests 1 to 4 were pure memory and CPU operations, with tests 1 and 2 of the form:

```
TIME=0:FOR I%=1 TO 100000:NEXT:
    PRINT TIME/100
```

doing nothing but incrementing a loop, and the BBC is clearly faster. However, indexing an array seems faster in the 512, (tests 3 and 4) so this area is swings and roundabouts, depending on your program.

The much slower screen handling is shown by tests 5 and 6. I assume that legal DOS calls are the reason. The 512 supports direct screen updates and it's pretty clear why. Scrolling causes an enormous overhead, yet I used PC mode 7 for the tests, as it's the quickest (e.g. Test 6 in PC mode 3 took 151.78 seconds.)

When 26 times the data is displayed per line (test 6) the gap between the BBC and 512

reduces, but it's still huge (the 512 figures are not misprints, they both comfortably exceed two minutes.)

512 file handling used the 800K format, because it's the quickest. Comparing tests 7 to 10, it's clear that opening and closing files is where time is lost. Tests 9 and 10 separate the two operations. When a file is open (or large) DOS+ scores well over DFS and extremely well over ADFS. Not a surprise, but it only shows that disc handling isn't hampered by BBCBasic(86).

OTHER FACILITIES

Two versions of the interpreter are provided in the package, the standard one allowing programs of up to 64K, with page at &900 and HIMEM at &10000.

A second version, 'BIGBasic', utilizes all memory up to 640K, giving 265,392 bytes in an un-expanded 512, but the manual cautions that this version is slower! I tried test 1 again, and without actually using extra memory the time increased by five seconds. This is because the processor has to set up the segment registers as well as the actual address, and this all takes time.

The disc also includes utilities to 'unlist' and to compact programs, both operations making programs unlistable. There's a utility to produce ASCII files from listable programs or vice-versa, the equivalent of the BBC's *SPOOL+LIST and *EXEC, and a program to perform a *DUMP.

There are also two conversion programs (one for 64K Basic, one 'big' version) to produce run-only code. This includes all the necessary routines so that programs can be run as 'EXE' files without needing a copy of BBCBasic(86). A licence is needed if you use this to distribute programs, or if you use any of the package in more than one machine.

The manual suggests that run-only programs are the next best thing to compiled code, so I duly 'CRUNCHed' test 1 (see below) and converted it. The result was that the original 29 bytes expanded to over 31K, but it took exactly the same time to run.

NOTE: Because of this statement I also ran tests 1 to 6 using Digital Research's CBasic86. The results are shown in the CB86 columns, 'I' means interpreted, 'C' is compiled. All the compiled programs were smaller than 6K. The times show exactly how 'slow' the 512 and the tube really are if driven hard. It's not the hardware or DOS+ that slows things down (not even the screen!).

For curiosity I also tried test 1 in Basic 5 on an Archimedes with RISC OS. It took just 14.74 seconds.

CONCLUSION

BBCBasic(86) is an excellent implementation of BBC Basic in terms of compatibility and syntax. To be able to run (some) programs without altering a single byte is as much as anyone could hope for. Restrictions are pre-dictable and attributable to hardware, DOS or the filing system. It can't be faulted on that score. It also proved robust and trouble free on the 512.

The documentation is comprehensive, and no one who writes Basic for the BBC micro, with even a rudimentary knowledge of the 512, will have the slightest difficulty in using this package.

Conversely, performance is a great disappointment; I expected to see speeds gains by factors of 2-3. If your objective is handling large amounts of data quickly BBCBasic(86) is not the answer. You do get the benefit of 512 disc handling, but that's equally true for any DOS package.

If your only concerns are larger quantities of data than your BBC micro can handle, but using a familiar language, BBCBasic(86) satisfies this, especially the large memory version. The penalty is that run times will probably be longer than on the BBC micro, especially when screen output is involved.

In spite of the obvious attractions of this package, I would strongly suggest that you investigate at least one other DOS Basic, ideally one with a compiler, before you make any decision. B

Game Strategy (Part 2)

David Spencer explains how to write a strategy game using the Minimax technique.

To recap, last month we looked at the principles needed to develop a strategy game, in particular chess, and developed the *static evaluator* and the *legal move generator*. Armed with these tools, the next step is to develop a method that allows the computer to choose the best moves to make. This is what we shall look at this month.

GAME TREES

Before going any further we need to look at a data structure called the *game tree*. As its name suggests, this is a tree structure, and you might find it useful to refresh your memory by reading the earlier workshop on trees (Vol.7 No.6).

Consider a tree with the following rules:

1. Each node is a record that can hold a board position and a score of the type produced by the static evaluator.

2. The branches out of any node represent one move each.

3. Each node (except a leaf node) has one branch for each move that can be made from the position stored in that node.

4. The nodes at the end of any branch contain the board positions that result from applying the move associated with that branch to the position held in the parent node.

5. The root node contains the current board position.

The resulting tree, known as a *game tree*, will have the current position as the root, with the first level down containing all the positions that can be reached from the current position by the player whose turn it is. The second level will contain the positions that the opponent can reach in reply to each of the first player's possible moves and so on.

To make this concept clearer, we will construct the first three levels of the game tree for the position shown in figure 1. This is a slightly unorthodox position, not least because neither player has a king on the board! There isn't the space here to draw pictures for all the board positions, so instead we will list the occupied squares together with the piece on them. So, the position in figure 1, and hence the root node of the game tree, would be written as:

(2,2) Pw
(3,3) Pb
(5,5) Pw

Where the 'P' indicates a pawn, and the suffix shows its colour.

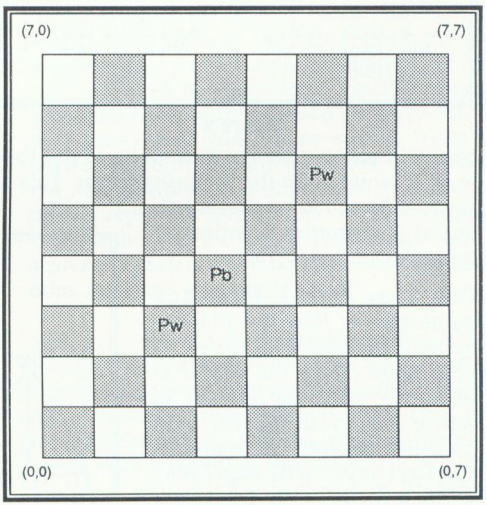


Figure 1

Assuming that white is to move, there are three possibilities. Firstly, he can advance his pawn on (2,2), totally ignoring black. Secondly, he can advance the pawn on (5,5), again ignoring

black, or thirdly, he can take black's pawn. Hence, there are three nodes in the first level of the game tree for this position. Figure 2 shows the first three levels of the game tree. The letters on the lines refer to the table of moves given at the bottom of the diagram, and each board position is shown using the above format.

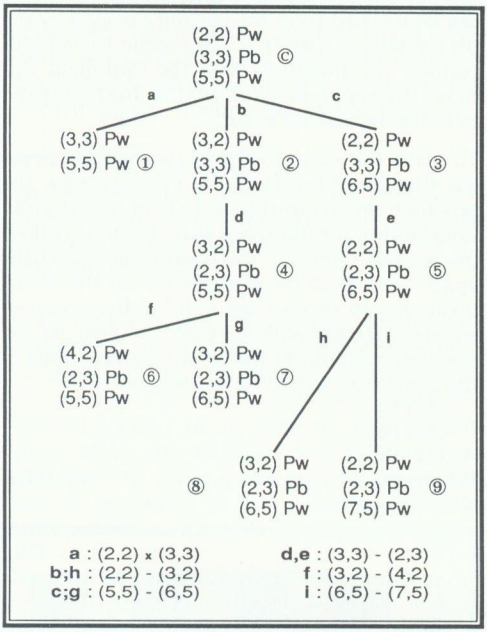


Figure 2

You will notice that the tree stops at the first level if white takes the opposing pawn. This is because black has no pieces left, and it is therefore pointless to try and continue further.

One thing to note about any game tree is that the players making the move alternate as you move down the tree. For example, suppose that it is white's turn to move. The branches between the root and the first level will represent white's moves, the branches between the first and second level are black's moves, those between the second and third are white's and so on. This is shown in figure 3.

Obviously, we can continue the game tree until the point where there are no legal moves that can be made. This represents the checkmate or stalemate situation, depending on which player's turn it is. In some cases, rather than reaching a dead end, the game will develop into a cat and mouse situation with the tree continuing deeper and deeper. This situation is indicative of a draw.

If we were to take as the current position the start of the game, and continue constructing a game tree until every branch hit a dead end or a position started to repeat, then it should be apparent that contained within this tree are all possible games of chess. Looked at another way, every game of chess *ever* played will be found somewhere by choosing a path down from the root of the tree! However, while there is a finite number of ways a game of chess can proceed, this number is very high. An average game of chess lasts for 35 moves, or 70 ply, and from any position there are an average of 37 possible moves that can be made. This means that, roughly speaking, the number of nodes in the game tree for the whole game is:

$$37^{70}+37^{69}+37^{68}+...+37^{2}+37^{1}+37^{0}$$

Don't bother trying this on your pocket calculator. The result is an astronomically large number. A researcher in Artificial Intelligence once said: "If a super-computer consisting of all the molecules in the universe had been examining all possible chess positions since the Big Bang, then it would only be a fraction of the way into the task."

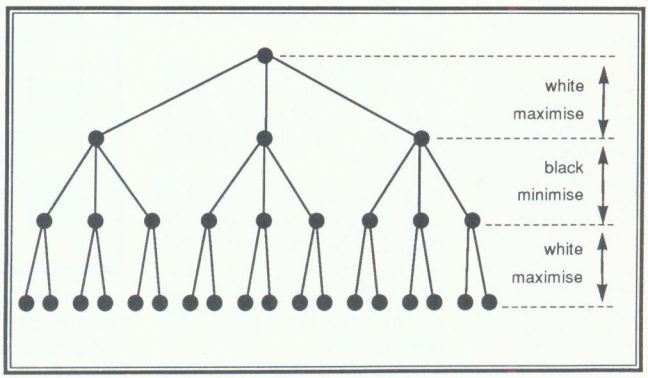


Figure 3

TREE SEARCHING

You might be beginning to wonder if we are ever going to deal with the actual problem in hand - namely choosing the best move to make. Well, we are going to right now. Remember what we said last month about looking ahead. Our aim is to choose a move that will hopefully lead to the best possible position at some point later in the game. The further we look ahead, the better 'feel' we can get for the game, and hence the more likely our chosen move is to succeed. In terms of our game tree, looking ahead simply means looking at the possible positions at one particular level in the tree. If we look ahead just one ply, then we examine the first level in the tree, two ply and it's the second level, and so on.

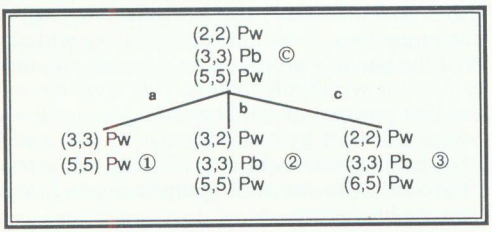


Figure 4

By using the static evaluator to generate a score for each node at our chosen depth, we can choose the best move. To start with, consider a one ply lookahead applied to the position in figure 1. This means that we will be looking at the first level of the game tree (nodes 1, 2 and 3 in figure 2). To simplify matters, we will assume that our static evaluator ignores the playing position, and only considers a player's material, with a pawn scoring one point.

As discussed last month, white's scores are positive and black's are negative. We can evaluate the score for the three positions in the first level and store them along with the actual position. Remember, we stated earlier that each node could hold a board position and a score. The scores for positions '1', '2' and '3' will be +2, +1 and +1 respectively. This is shown in figure 4. As we want to maximise the score, we need to take the move that leads to the position with the highest score. Therefore, we take the move (2,2) to (3,3) taking black's pawn and leaving white two points ahead.

THE MINIMAX TECHNIQUE

So, we now know how to deal with a single ply lookahead, but what about going one stage further - in other words a two ply lookahead. This means that we will consider the position after black makes a move in reply to whatever move we choose to make. This is shown in figure 5.

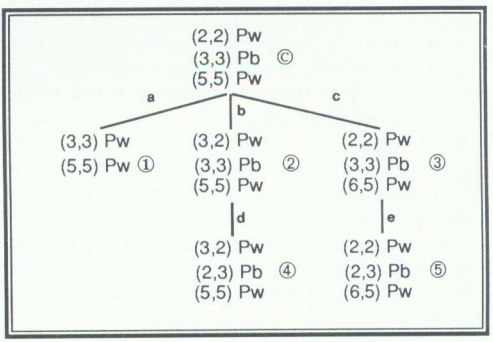


Figure 5

Remember that in the case of a single ply, we choose the move that maximises our score. However, when our opponent chooses his move, he will want to try and minimise the score, as the lower the score the better his position. Our very simple position in figure 5 can't be used to adequately describe this, so look instead at figure 6, which shows two levels of a game tree with three moves from the current position, and three possible moves from the new position. Therefore, white's move and black's reply can lead to any of nine positions. The numbers on the leaves of the tree show the scores for each of the nine positions (ignore the other numbers for the moment).

The score for the right-most position is +12, clearly the best score for white. So, you might expect white to play move 'b', heading towards the +12. But what happens if white does play this move? Black is quite likely to play the move that minimises the score - in this case move 'c', achieving a score of -4. Therefore, white has effectively played into his opponent's hands and allowed him to get ahead.

So, it is not simply a question of playing the move that 'heads towards' the best score.

Instead, we need to develop a new technique, known affectionately as *Minimax*. We have already said that black will try and minimise the score. Knowing this, for the three nodes in the first level of figure 6, we can determine the score that black will get from that position. For example, from the left-most node, black can choose moves leading to scores of +4, +2 and +6. As he is trying to minimise the score he is likely to choose the +2 move. Therefore, white knows that if he moves to that particular position, black could force a score of +2. Similarly, from the second node black could force a score of -9, and from the third node -4. These scores are shown on the first level of figure 6. We can now ignore the second level of the tree, as we know the scores that will be attained. This just leaves white with the choice between moves leading to scores of +2, -9 or -4. As white wants to maximise the score, he will choose the move leading to +2 (marked 'a' in figure 6).

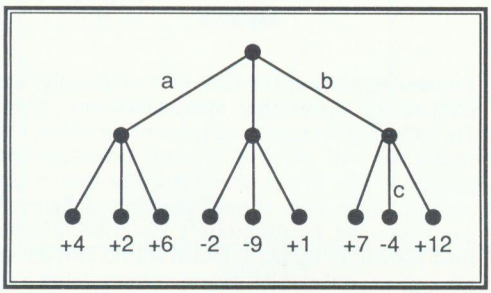


Figure 6

Consider the steps that were taken. Starting at the penultimate level of the tree (level one in this case), we took each node in turn, and looked at the positions that could be achieved from that node. Because it was black's turn, we took the minimum score that could be achieved from any of the moves, and assigned that to the particular node. Having done this, we moved up the tree by one level, and repeated the process. However, as it was now white that had to choose, we took the maximum score rather than the minimum. In our example this was the end of the story, but had our tree had more levels, we would have just continued upwards, alternating between the maximum and minimum scores at each level, hence the term Minimax. This process of working backwards up the tree is called, not surprisingly, *Backing*. If you are not sure about this I suggest you draw out a few trees of various depths, assigning arbitrary scores to each leaf, and then try the Minimax procedure on each one.

Going back to the position in figure 5, if white takes black's pawn then black cannot move, while if we ignore the capture, black has a single move at his disposal. This doesn't alter the score at all. Therefore the best move for white to make is still to take black's pawn.

Now consider a three ply lookahead, taking in all of figure 2. At first sight, it might seem that all the positions at the third level of the tree have a score of +1. However, look closely at position 'D'. White has managed to advance a pawn to the far rank, which means it is eligible for promotion. For argument's sake we will say that the pawn is promoted to a queen, and that a queen is worth nine points. This gives a score for this position of +9. Therefore, on this basis white will start by making move 'c', black will then be forced to make move 'e', and white will finish off with move 'i', gaining a pawn and hence a high score.

PRACTICAL TREE SEARCHING

So far I have given the impression that a game tree is created somehow in memory, and then searched subsequently. However, this is not practical in a real situation. Consider a three level tree for a game of chess. The average number of nodes will be:

$37^3+37^2+37^1+37^0$, or 52060

If each node takes 120 bytes of memory (a fairly realistic guess), then the entire game tree will occupy just under 6Mb of memory - not very useful on a model B!

The solution to this problem is to develop a routine to build up the tree as it searches it, discarding branches that are no longer needed. We will tackle this next month. We will also look at a very clever method for drastically cutting down the number of positions in the game tree which need to be examined.

The BEEBUG Draughts game in this issue puts the theory of the Minimax technique into practice.

File Handling For All (Part 8)

by Mike Williams and David Spencer

We continue the discussion on the use of pointers with more practical examples.

Let's first conclude the discourse that we started last time on the use of internal pointers, by considering a more complex, and to some extent, more realistic approach. Think back to the order processing application which we described previously. When an order is placed it is likely to contain several different items. For each item we would need to record some reference to the item itself (maybe a code number), the quantity ordered and possibly other information such as a discount. A difficulty arises because of the variable number of items which might appear on any one order.

One solution to this would be to define a record which would cater for a predetermined maximum number of items (5 say). If fewer than five items were ordered then space would be wasted. However, by associating a pointer with each record, a series of similar *continuation* records could be chained together to cater for more than five items per order. It is obviously a matter of judgement as to how big to make each unit record (the larger it is, the more space is potentially wasted), whereas using smaller sized records may seem more efficient, but will carry a greater overhead in processing time because of the many more links which have to be followed.

In fact, it might be desirable to have two pointers per record, one to point to the next complete order, and one to point to the next continuation record for this order. It all depends on what is known about the orders likely to be received, and what is required from such a system. In such a case, the orders within the file would be chained together, as described last month, but each order might itself consist of a set of linked records. The situation is illustrated in figure 1.

When we implemented pointers last month we chose to make them transparent, that is they were not identifiable as pointer fields within a record, but existed as as a separate entity tagged on to the end of each record. We will follow the same approach here so that each record will have two pointers appended to it.

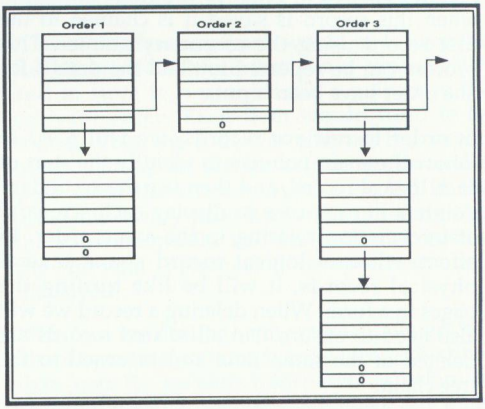


Figure 1. Structure of orders file

In dealing with this example further, it is worth looking at the difference in concept between logical and physical records. For convenience and economy in the use of storage we have decided to define a file which consists of physical records each of which can hold the order details for a specific number of items. A logical order corresponds to the order which has been received from the customer, containing a varying number of items. Each logical order will translate into one or more physical records in the file.

With this concept we can see how our first pointer effectively chains together logical records, while the second pointer chains together physical records.

Let's now consider what will happen in practice. Suppose we are going to process an order and add the details to our order file. The program will locate the first free record in the file and display the corresponding fields for input on the screen. We enter the details as required, and the only difference from previous practice is that we will need to indicate whether we have finished processing that order, or whether we have more details to enter. If all the

details have been completed then that record can be saved by chaining it to the existing records and we can proceed to process the next order.

If more details remain to be entered, then the first record is saved as before, and a new free record obtained in order to continue. However, when this record is saved it is chained to the first record using the secondary pointer. This process can be repeated until all the details for one order have been input.

In order to retrieve records, we will need to follow the main pointers to identify the start of each logical record, and then use the secondary pointers in each case to display each screenful of information relating to the same order. In effect, where a logical record spans several physical records, it will be like turning the pages in a form. When deleting a record we will also need to ensure that all related records are deleted at the same time and returned to the free chain.

To see how this affects any coding, let us look again at the procedure PROCchain introduced last month, modified here to implement our two-pointer system:

```
1600 DEF PROCchain(buffer%)
1610 R%=FNfree(RF%):IF NOT R% ENDPROC
1620 PROCfind_pos(key%,RL%)
1630 PROCwrite_record(R%,buffer%)
1640 BPUT#F%,R2% MOD 256
1650 BPUT#F%,R2% DIV 256
1660 PTR#F%=start%+(RS%+4)*R1%-4
1670 BPUT#F%,R% MOD 256
1680 BPUT#F%,R% DIV 256
1690 ENDPROC
```

Note that line 1660 has been modified to reflect the fact that each physical record is now followed by two pointers each using two bytes, a total of four. Since RS% is the record size, each record with its pointers uses (RS%+4) bytes. Note that last month's code had a minor error at this point, and should have read (RS%+2) and not (RS%+1). This line moves to the end of the specified record (including pointers), and then back the appropriate number of bytes ready to write a new pointer.

The PROCchain procedure locates a free record using FNfree. It then determines the position in the existing chain of records for the new record which is then saved together with relevant pointers. The variable R% indicates the new record, and R1% and R2% the immediately preceding and succeeding records in the chain. This is illustrated in figure 2, which shows the main pointers after the record has been added. The secondary pointers have been omitted as their status is irrelevant at this stage.

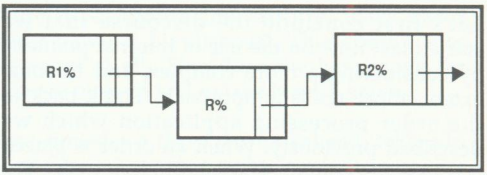


Figure 2. Inserting a new record (R%) in the main chain

The variable key% indicates which field acts as the key to each record. This will still serve in the new context to locate and save the first record corresponding to any order.

To cope with continuation records we will need to add a new variable link% and a new procedure called PROClink. The purpose of link% is to indicate (presumably from the user's response to a question) whether the current record is to be continued or not. If it is, PROClink will find and link in the record as follows:

```
1900 DEF PROClink(buffer%)
1910 R1%=R%:R2%=0
1920 R%=FNfree(RF%):IF NOT R% ENDPROC
1930 PROCwrite_record(R%,buffer%)
1940 BPUT#F%,R2% MOD 256
1950 BPUT#F%,R2% DIV 256
1960 PTR#F%=start%+(RS%+4)*R1%-2
1970 BPUT#F%,R% MOD 256
1980 BPUT#F%,R% DIV 256
1990 ENDPROC
```

This turns out to be very similar to PROCchain. Lines 1940 and 1950 set the main pointer of the continuation record to zero (it's not part of the main chain), while lines 1960 to 1980 locate and set the secondary pointer of the preceding record (by moving back two instead of four bytes as in PROCchain) to point to the continuation record. Repeated applications of PROClink will chain as many continuation records as required. The result of adding a continuation record is shown in figure 3.

The only other point to make is that the last physical record in a continuation chain, whichever it may be, will need to have its secondary pointer set to zero to indicate the end of a secondary chain. The easiest way to achieve this is to ensure that secondary pointers are always set to zero both initially and whenever a record is returned to the free chain. The last continuation record will then always have a zero secondary pointer without any further action on our part.

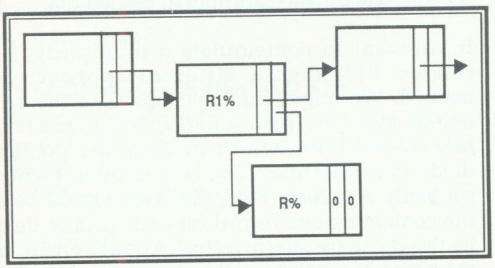


Figure 3. Adding a continuation record (R%)

As far as accessing and displaying records is concerned, again only a small modification is needed. The procedure PROCread_record (first given in BEEBUG Vol.7 No.4) will now need to read both the main (pointer%) and secondary pointers (link%) by adding lines:

```
2555 pointer%=BGET%#F%+256*BGET#F%
2556 link%=BGET#F%+256*BGET#F%
```

and the routine would take the form:

```
R%=RL%
REPEAT
PROCdisplay_record(R%,buffer%)
IF link% THEN
REPEAT:
PROCdisplay_record(link%,buffer%)
UNTIL link%=0
R%=pointer%
UNTIL pointer%=0
```

Compare this with the outline code given last month for the display of simply linked records. Routines for deleting and updating records can be coded similarly.

EXTERNAL POINTERS

Our discussion of pointers so far has concerned their use internally within a file for the purpose of ordering and linking records. Pointers can serve quite a different function when they are used to point to data in another file. As an illustration we can consider again our order processing example. Although it may be desirable and useful to display or print a description of each item ordered, it would be quite wasteful of storage space to include the full description in every order record where that item occurs.

A more efficient approach would be to keep the full descriptions of all items in a separate file, and to store in our order file a pointer to the relevant entry. Each item would need to be coded. When the code was entered, this would be looked up in the descriptions file and a pointer to the record found inserted in the order file. Every time that order is referred to, the software would use the pointer to ensure the relevant information was retrieved and displayed.

When we are dealing with external pointers it is almost essential to introduce *pointer fields* rather than keep the pointers transparent as before. In effect an external pointer field is like any other data field except that for convenience the data itself is located in another file.

All the time, although it may not be explicitly mentioned, we are weighing up a balance between the efficient use of storage on one hand and the extra time spent on more complex accessing on the other. It is often surprising how wasteful of storage commercial systems can be in the interests of fast processing.

It is also perfectly possible to combine the use of internal pointers for linking records together, with the use of external pointers for other purposes. An almost classic example of this is in the maintenance of computerised bibliographies designed for keyword access.

What we could do is to maintain a file of references giving details of title, authors, publisher and so on. This file could be simply in the order in which the records are entered. Each reference is classified according to a set of appropriate keywords, and the keywords would exist in a second file either chained together or ordered in some other way which provides speedy access to a given keyword. Each keyword points to a chain of single-field records where each field is an external pointer to a corresponding reference in the first file.

When a new entry is to be made in the bibliography, the details are entered in the main reference file, and the appropriate keywords entered. Each keyword will be located in turn in the keyword file, the related chain of pointers followed to the end, and a new pointer (to the latest entry) added to the end of the chain.

When it comes to accessing information we simply enter one or more keywords, obtaining in each case a set of pointers. If we enter several keywords then we would usually have the choice of ORing or ANDing the corresponding pointer sets. Finally we (or at least the program we are using) could follow each pointer to extract the corresponding reference.

IMPLEMENTING EXTERNAL POINTERS

Let's now spend the remainder of this month's article examining the practicalities of using external pointers. In all our previous discussions we have stored all our data as character strings, using knowledge of the field type, when appropriate, to convert between the way in which the data is stored in the file and the way in which it is manipulated within the program. We will do the same with pointer fields.

A pointer is just a record number so we will assume that record numbers are in the range 0 to 65535 (that's the largest number that can be stored in two bytes). If we store record numbers as they are we will need a field of 5 characters. However, if we can treat them as two bytes we need only two characters. To do that we require two functions, one to code a record number as two characters, and one to decode two characters as a record number: Here are two such functions:

```
DEF FNcode(R%)=CHR$(R%MOD256)+
   CHR$(R%DIV256)
DEF FNdecode(R$)=ASC(R$)+
   256*ASC(MID$(R$,2))
```

where FNcode returns a two-byte string corresponding to a record number R%, and FNdecode returns a record number corresponding to a two-byte string R$.

Right, so we have decided how an external pointer is to be stored, but how will the program know to which file an external pointer is related, and which field in a linked record is required? There are two answers to this. If we restrict the number of related files sufficiently (and maybe just one related file will suffice for most needs), then this information can be stored in the FDR. When the file is first opened, the contents of the FDR will specify which field is an external pointer field and the name of the related file and field, which is opened in turn. It is in just this situation that we need more than one record buffer in memory, one for the main file and one for each additional related file.

If we want to contemplate a multiplicity of related files then a different approach is needed. We might find that one data file has one or more external pointers, but in addition any related files may themselves use pointer fields to further files. The best solution then is probably a pointer field file. This would be a file containing one record for each pointer field in the database. Each record would consist of the name of a pointer field, the name of the file being pointed to, and the name of the related field.

I fear we are beginning to get well beyond the bounds of reasonable capability for any BBC micro. For a start, no more than five files may be open at any one time on the DFS. Also the complexity of processing may also become such that even if it does all work, it works so slowly as to be well nigh impractical. That is not to say that our efforts have been wasted, but that we must be realistic in our aims.

Pointer fields can be very useful in the kinds of example we have quoted. We suggestyou use our first suggestion of keeping the details of related files in the FDR to be accessed when a file is first opened, and restrict the number of related files to no more than one or two.

This month's magazine disc/tape contains a revised version of the program given last month which now implements dual internal pointers and external pointers as described above. It is accompanied by a short data file based on our order processing example.

That concludes our examination of file pointers. Next month, in our series on file handling, we will take a look at something quite different.

B

BEEBUG Draughts

To complement the game playing theory of the current Workshop series, David Spencer implements a fully working draughts game.

The program given here is a complete game of Draughts in which the computer plays an opponent. There are three playing levels, ranging from easy to very hard. The higher the level, the longer taken for each move (from less than a second on level 1 to a minute for some moves on level 3).

ENTERING THE PROGRAM

The program should be entered exactly as printed, paying special attention to the assembly language section at the end. As a precaution, save the program before running it.

BEEBUG Draughts will work on all machines. If the value of PAGE is greater than &1900 then the program is automatically downloaded to &1900.

Game in progress

PLAYING THE GAME

The computer plays with the red counters, and the player uses the yellow ones. The computer always moves first. To enter a move you simply type the co-ordinates of the 'from' and 'to' squares. For example, entering D2E3, would move your piece from square D2 to square E3. You must press Return to confirm the move, and Delete can be used to clear the move currently being entered. In the case of jumps, you enter the co-ordinates of the counter doing the jumping, and the co-ordinates of the square where it will end up. If the move is a multiple jump, and it is ambiguous as to which route to take, the program prompts for the co-ordinates of the intermediate square. For example, a double jump from C3 to C7 might, pieces permitting, pass over square A5 or E5.

The computer can play at three different levels. It starts on level two, but this can be changed by pressing the cursor up and down keys when it is the player's move. Level three is the hardest.

When either player loses all his pieces, or cannot move, then his opponent is declared the winner and pressing any key will start a new game. Escape can be used at any time to terminate the program.

Note that *BEEBUG Draughts* doesn't play the so-called 'Huffing rule', in which pieces can be removed if they fail to make a jump when one is possible. Therefore, in this version of the game jumping is not compulsory.

HOW IT WORKS

Armed with the listing given here, and the theory explained in the last two Workshops, you should be able to follow the operation of the program. In order to achieve a decent playing speed, the *Legal Move Generator* and the *Static Evaluator* (as well as a routine to copy a board) have been written in machine code. The tree search routine (FNminimax) is speeded up using a technique called *Alpha-Beta* pruning which will be covered in next month's Workshop. One point worth noting is the way that the player's move is validated by comparing it with the list produced by the *Legal Move Generator*.

```
   10 REM Program DRAUGHTS
   20 REM Version B1.00
   30 REM Author  David Spencer
   40 REM BEEBUG  Jan/Feb 1989
   50 REM Program subject to copyright
   60 :
  100 IF PAGE>&1900 THEN OSCLI"KEY 0 FOR
 F%=0 TO TOP-PAGE STEP 4:F%!&1900=F%!PAG
E:NEXT:PAGE=&1900:END|MRUN|M":OSCLI"FX13
8 0 128":END
  110 maxdepth%=3:depth%=2
  120 A=RND(-TIME):*FX4 1
  130 MODE 5:VDU 23,224,0,0,&A5,&A5,&5A,
&3C,&7E,0
  140 ON ERROR GOTO 350
  150 MOVE 160,1020:VDU 5:PRINT"BEEBUG D
RAUGHTS":VDU4
  160 DIM scratch% 100,mlist% 100,bstack
% 64*maxdepth%,mstack% 100*maxdepth%,bte
mp% 63
  170 PROCassemble
  180 REPEAT
  190 PROCbinit
  200 PROCsinit
  210 REPEAT
  220 PROCcmove
  230 over=FNwin(255)
  240 IF over=0 THEN PROCpmove
  250 IF over=0 THEN over=FNwin(0)
  260 UNTIL over
  270 PRINTTAB(0,29);SPC(19)
  280 IF black>0 AND over=-1 THEN PRINTT
AB(0,29);"Computer cannot move"
  290 IF white>0 AND over=1 THEN PRINTTA
B(0,29);"Player cannot move"
  300 PRINTTAB(0,31);:IF over=1 THEN PRI
NT"Computer wins"; ELSE PRINT"Player win
s";
  310 *FX15 1
  320 A=GET:UNTIL FALSE
  330 END
  340 :
  350 IF ERR=17 THEN OSCLI"FX4":MODE 7:E
ND
  360 MODE 7:OSCLI"FX4":REPORT:PRINT" at
 line ";ERL:END
  370 :
 1000 DEF PROCsinit
 1010 GCOL 0,3
 1020 MOVE 156,188:PLOT 1,964,0
 1030 PLOT 1,0,772:PLOT 1,-964,0
 1040 PLOT 1,0,-772
 1050 FORR%=0TO7
 1060 FORF%=0TO7
 1070 GCOL 0,3*(R% AND 1 EOR F% AND 1)
 1080 MOVE 160+120*F%,192+96*R%
 1090 PLOT 0,116,0:PLOT 81,0,92
 1100 PLOT 0,-116,0:PLOT 81,0,-92
 1110 NEXT
 1120 NEXT
 1130 FORF%=0TO63
 1140 IF cboard%?F%<>0 PROCcounter(F%,1-
(cboard%?F%>127),0)
 1150 NEXT:VDU 29,0;0;:GCOL 0,3
 1160 VDU5:FOR F%=0 TO 7
 1170 MOVE 72,256+96*F%:VDU F%+49
 1180 MOVE 204+120*F%,172:VDU F%+65
 1190 NEXT
 1200 VDU4,23,1;0;0;0;0
 1210 PROClevel
 1220 ENDPROC
 1230 :
 1240 DEF PROClevel
 1250 LOCAL pos,ypos
 1260 pos=POS:ypos=VPOS
 1270 PRINT TAB(0,31);"Level ";depth%;TA
B(pos,ypos);
 1280 ENDPROC
 1290 :
 1300 DEF PROCbinit
 1310 DIM cboard% 63
 1320 FORF%=0TO63:cboard%?F%=0:NEXT
 1330 FORR%=0TO2
 1340 FORP%=0TO3
 1350 cboard%?(R%*8+(P%*2-(R%=1)))=255
 1360 cboard%?((7-R%)*8+(7-P%*2+(R%=1)))
=1
 1370 NEXT
 1380 NEXT
 1390 ENDPROC
 1400 :
 1410 DEF PROCcounter(P,C,K)
 1420 LOCAL I%,X,Y
 1430 X=220+120*(P MOD 8):Y=240+96*(P DI
V 8)
 1440 VDU 29,X;Y;:GCOL 0,C
 1450 FORI%=0 TO 10
 1460 X=SQR(2304-23.04*I%*I%)
 1470 MOVE-X,I%*4:DRAW X,I%*4
 1480 MOVE-X,-I%*4:DRAW X,-I%*4
 1490 NEXT
 1500 IF K THEN GCOL 0,0:MOVE -32,16:VDU
5,224,4,23,1;0;0;0;0
 1510 ENDPROC
 1520 :
 1530 DEF PROCsmove(move%,board%,scr%)
 1540 LOCAL t%,new%,old%:new%=?move%
 1550 t%=board%?new%:board%?new%=0
 1560 IF scr% PROCcounter(new%,0,0)
 1570 REPEAT move%=move%+1
```

```
 1580 old%=new%:new%=?move% AND &7F
 1590 IF ABS(new%-old%)>=14 THEN board%?
(old%+(new%-old%) DIV 2)=0:IF scr% PROCc
ounter(old%+(new%-old%) DIV 2,0,0):IF ?m
ove%<128 THEN PROCcounter(new%,1-(t%>127
),1-(t% AND 1)):PROCcounter(new%,0,0)
 1600 UNTIL?move%>127
 1610 IF (new%<8 OR new%>55) AND (t% AND
 1) THEN t%=t%-1-2*(t%=1)
 1620 board%?new%=t%:IF scr% PROCcounter
(new%,1-(t%>127),1-(t% AND 1))
 1630 ENDPROC
 1640 :
 1650 DEF PROClegalmoves(board%,buff%,si
de%)
 1660 !&70=board%:!&72=buff%:?&74=side%
 1670 CALL lmoves
 1680 ENDPROC
 1690 :
 1700 DEF PROCpmove
 1710 LOCAL from,to,valid,sq%
 1720 REPEAT
 1730 PRINTTAB(0,29);"Enter Move:      "
;STRING$(5,CHR$8);
 1740 from=FNgcoord:IF from=-1 THEN 1730
 1750 PRINT"-";:to=FNgcoord
 1760 IF to=-1 THEN 1730
 1770 REPEAT addr=GET
 1780 UNTIL addr=13 OR addr=127
 1790 IF addr=127 THEN 1730
 1800 PROClegalmoves(cboard%,scratch%,25
5)
 1810 ?mlist%=from:sq%=1
 1820 REPEAT valid=FNvalmove(scratch%,to
,mlist%,sq%)
 1830 IF valid<>-1 THEN 1890
 1840 PRINTTAB(0,29);:IF sq%=1 PRINT"Via
:"; ELSE PRINT"and:";
 1850 PRINTSPC(13);STRING$(13,CHR$8);
 1860 REPEAT from=FNgcoord
 1870 UNTIL from<>-1
 1880 mlist%?sq%=from:sq%=sq%+1
 1890 UNTIL valid<>-1
 1900 IF valid=0 THEN SOUND 1,-12,0,5
 1910 UNTILvalid>0
 1920 PROCsmove(valid,cboard%,1)
 1930 ENDPROC
 1940 :
 1950 DEF FNvalmove(list%,to%,move%,coor
ds%)
 1960 LOCAL co%,valid%,good%,addr%,addr2
%
 1970 REPEAT:co%=0:valid%=TRUE
 1980 REPEAT
 1990 IF list%?co%<>move%?co% THEN valid
%=FALSE
 2000 co%=co%+1
 2010 UNTIL co%=coords% OR NOTvalid%
 2020 addr%=list%
 2030 REPEAT list%=list%+1
 2040 UNTIL ?list%>128
 2050 IF ?list%-128=to% AND valid% THEN
good%=good%+1:addr2%=addr%
 2060 list%=list%+1
 2070 UNTIL ?list%=255
 2080 IF good%=1 THEN =addr2%
 2090 IF good%=0 THEN =0
 2100 =-1
 2110 :
 2120 DEF FNgcoord
 2130 LOCAL key,pos,del
 2140 del=FALSE
 2150 REPEAT key=GET:IF key=127 THEN del
=TRUE
 2160 IF key=138 AND depth%>1 THEN depth
%=depth%-1
 2170 IF key=139 AND depth%<maxdepth%  T
HEN depth%=depth%+1
 2180 PROClevel
 2190 UNTILINSTR("ABCDEFGH",CHR$key) OR
del
 2200 IF del THEN =-1
 2210 pos=INSTR("ABCDEFGH",CHR$key)-1:VD
U key
 2220 REPEAT key=GET:IF key=127 THEN del
=TRUE
 2230 IF key=138 AND depth%>1 THEN depth
%=depth%-1
 2240 IF key=139 AND depth%<maxdepth%  T
HEN depth%=depth%+1
 2250 PROClevel
 2260 UNTIL (key>&30 AND key<&39) OR del
 2270 IF del THEN =-1
 2280 VDU key:=(key-49)*8+pos
 2290 :
 2300 DEF FNstaticevaluate(!&70)
 2310 LOCAL t%
 2320 t%=USR seval AND &FF
 2330 IF t%>127 THEN =t% OR &FFFFFF00 EL
SE =t%
 2340 :
 2350 DEF FNwin(side)
 2360 LOCAL pos
 2370 black=0:white=0
 2380 FOR pos=0 TO 63
 2390 IF cboard%?pos<>0 AND cboard%?pos<
128 THEN black=black+1
 2400 IF cboard%?pos<>0 AND cboard%?pos>
127 THEN white=white+1
 2410 NEXT
```

```
 2420 PROClegalmoves(cboard%,scratch%,si
de)
 2430 IF black=0 OR (side=0 AND ?scratch
%=255) THEN =-1
 2440 IF white=0 OR (side=255 AND ?scrat
ch%=255) THEN =1
 2450 =0
 2460 :
 2470 DEF PROCcmove
 2480 LOCAL off%,score%,max%,ptr%,poss%
 2490 max%=&80000000:ptr%=mlist%:poss%=0
 2500 PRINTTAB(0,29);"Thinking          "
 2510 PROClegalmoves(cboard%,mstack%,0)
 2520 off%=mstack%:REPEAT
 2530 !&75=cboard%:!&77=btemp%
 2540 CALL copbrd
 2550 PROCsmove(off%,btemp%,0)
 2560 score%=FNminimax(btemp%,bstack%,ms
tack%+100,0,1,max%)
 2570 IFscore%<max% THEN 2650
 2580 IFscore%>max% THEN ptr%=mlist%:pos
s%=0
 2590 REPEAT ?ptr%=?off%
 2600 off%=off%+1:ptr%=ptr%+1
 2610 UNTILoff%?-1>127
 2620 poss%=poss%+1
 2630 max%=score%
 2640 GOTO2680
 2650 REPEAT off%=off%+1
 2660 UNTIL?off%>127
 2670 off%=off%+1
 2680 UNTIL?off%=255
 2690 poss%=RND(poss%)-1:off%=mlist%
 2700 IF poss%=0 THEN 2760
 2710 REPEAT
 2720 REPEAT off%=off%+1
 2730 UNTIL?off%>127
 2740 off%=off%+1:poss%=poss%-1
 2750 UNTILposs%=0
 2760 PROCsmove(off%,cboard%,1)
 2770 ENDPROC
 2780 :
 2790 DEF FNminimax(board%,save%,ml%,min
max%,l%,ab%)
 2800 LOCAL sc%,score%,off%
 2810 IF minmax% THEN score%=&80000000 E
LSE score%=&7FFFFFFF
 2820 IF l%=depth% THEN =FNstaticevaluat
e(board%)
 2830 PROClegalmoves(board%,ml%,-1+2*min
max%)
 2840 off%=ml%:IF ?off%=255 THEN 2980
 2850 REPEAT
 2860 !&75=board%:!&77=save%
 2870 CALL copbrd
 2880 PROCsmove(off%,save%,0)
 2890 sc%=FNminimax(save%,save%+64,ml%+1
00,NOT minmax%,l%+1,score%)
 2900 IF NOTminmax% AND sc%<score% THEN
score%=sc%
 2910 IF minmax% AND sc%>score% THEN sco
re%=sc%
 2920 cut%=FALSE
 2930 IF NOTminmax% AND score%<ab% THEN
cut%=TRUE
 2940 IF minmax% AND score%>ab% THEN cut
%=TRUE
 2950 REPEAT off%=off%+1
 2960 UNTIL ?off%>127:off%=off%+1
 2970 UNTIL ?off%=255 OR cut%
 2980 =score%
 2990 :
 3000 DEF PROCassemble
 3010 boff=&80:yt=&81:kflag=&82
 3020 piece=&83:score=&84:sc=&85
 3030 side=&86:joff=&87:yt2=&88
 3040 temp=&89
 3050 ms=&AC0
 3060 FOR pass%=0 TO 2 STEP 2
 3070 P%=&900
 3080 [OPT pass%
 3090 .lmoves LDY #0:STY boff
 3100 .lmoves2 LDA (&70),Y
 3110 BEQ lmoves3:EOR &74
 3120 BMI lmoves3:JSR lm
 3130 .lmoves3 INY:CPY #64
 3140 BNE lmoves2:LDA #&FF
 3150 LDY boff:STA (&72),Y:RTS
 3160 .lm STY yt:LDA (&70),Y
 3170 STA piece:AND #1:STA kflag
 3180 CPY #56:BCS noup:TYA:AND #7
 3190 BEQ nol:LDA #7:JSR tmove
 3200 .nol LDA yt:AND #7:CMP #7
 3210 BEQ noup:LDA #9:JSR tmove
 3220 .noup LDY yt:CPY #8:BCC jmp
 3230 TYA:AND #7:BEQ nol2:LDA #&F7
 3240 JSR tmove:.nol2 LDA yt:AND #7
 3250 CMP #7:BEQ jmp:LDA #&F9:JSR tmove
 3260 .jmp LDA #0:STA joff
 3270 .jmp2 LDA yt:CMP #48:BCS noju
 3280 AND #7:CMP #2:BCC nojl:LDA #7
 3290 JSR tj:.nojl LDA yt:AND #7
 3300 CMP #6:BCS noju:LDA #9:JSR tj
 3310 .noju LDA yt:CMP #16:BCC nojd
 3320 AND #7:CMP #2:BCC nojl2:LDA #&F7
 3330 JSR tj:.nojl2 LDA yt:AND #7
 3340 CMP #6:BCS nojd:LDA #&F9:JSR tj
 3350 .nojd LDY yt:RTS
 3360 .tmove PHA:LDA kflag:BEQ ki
```

Continued on page 60

POSTBAG

MOVEMENT IN THE SHADOWS

I read with interest your article in BEEBUG Vol.7 No.2 on shadow memory, and felt like experimenting with the commands. So I took the 3D Rotating House program from Vol.6 No.9 and modified it to use the shadow memory technique, thus achieving smooth animation. As this only requires the addition of 5 lines, it is very easy to do, and quite effective.

The lines are:

```
 70 MODE129
 80 X%=2:Y%=0
145 A%=112:CALL &FFF4
420 A%=113:CALL &FFF4
425 IF X%=1 X%=2 ELSE X%=1
```

I hope these additions will be of interest to other Master-owning members.

Mark Etherington

Our thanks to Mr Etherington for supplying this information.

THE ERROR OF ITS WAYS

I should be glad of your help with regard to the article on error handling in BEEBUG Vol.5 No.9. The example program for dealing with errors is all right, but I would like to use it as a procedure to be called when required. However, when I attempt to do this, all works fine, but then I get an error "No Procedure at line ...". I would be grateful for your help.

W.H.Jones

Although the error trapping provided within BBC Basic initially looks to be quite useful, there is a fundamental flaw. When any error occurs, Basic's stack is re-initialised. That means that if the program is inside a FOR-NEXT or REPEAT-UNTIL loop, or within a procedure or function, Basic loses all 'knowledge' of that state.

Thus if a procedure to carry out some task includes any error trapping code relating to that task (divide by zero or Escape for example), then Basic 'forgets' that it is in a procedure. The error handling code will be executed correctly, but when Basic reaches the ENDPROC terminating the procedure definition, it appears as a mistake, as Basic has forgotten that it ever entered a procedure in the first place. Thus the error message referred to by Mr Jones' results.

So, error trapping should normally be placed outside any loop, or procedure or function definition. If you want the program to continue, the only possibility is to GOTO a line near the start of the program which is outside any loop, or procedure or function definition.

Readers may be interested to know that Basic V on the Archimedes provides additional error handling facilities which overcome most of these problems.

LOCATING HALLEY'S COMET

The following few lines will add Halley's Comet to the excellent 'Solar System' program published in BEEBUG Vol.7 No.6:

```
 250FOR Q=0 TO 9
 280FOR Q=0 TO 9
 290PRINTTAB((QMOD 3)*11,
Q DIV 3);Q;"=";P$(Q);:NEXT
 540N(Q)=fact*(d/P(Q))
 550nu(Q)=FNkepler(RAD(N(Q)
+ep(Q)-W(Q)),CY(Q))
 5601(Q)=nu(Q)+W(Q)
 760R=400/(AM(outer%)*(1+CY
(outer%)))
 765IF Q=0 THEN GCOL 0,3
 990DATA Halley's Comet
 992DATA 76.1,0.9673,112
 994DATA 17.9,58,162.2
 996DATA 57.78
 1660DEF FNkepler(M,e)
 1670LOCAL E,F
 1680REPEAT
 1690F=E:E=M+e*SINE
 1700UNTIL ABS(E-F)<1E-3
 1710=DEG(2*ATN(SQR((1+e)/
 (1-e))*TAN(E/2)
```

An iterative computation is necessary to keep the display sane if the eccentricity is near unity. There are also perturbations in its orbit which cannot be easily allowed for here, but which would affect the accuracy, particularly as one goes away in time from the epoch. However, interested readers will find a table of computed elements for all apparitions going back to 1404BC in Yeomans & Kiang, Mon. Not. R.Astr.Soc (1981) 197, 633-646.

V.W.Patterson

Our thanks to Mr Patterson for this interesting addition to a popular program. B

```
3370 PLA:PHA:EOR piece:BMI ki:PLA:RTS
3380 .ki PLA:CLC:ADC yt:TAY:LDA (&70),Y
3390 BNE tmove2:TYA:PHA:LDA yt:LDY boff
3400 STA (&72),Y:INY:PLA:ORA #&80
3410 STA (&72),Y:INY:STY boff
3420 .tmove2 RTS
3430 .tj PHA:LDA kflag:BEQ ki2
3440 PLA:PHA:EOR piece:BMI ki2
3450 .tjo PLA:.tjo2 RTS
3460 .ki2 PLA:PHA:CLC:ADC yt:TAY
3470 LDA (&70),Y:BEQ tjo
3480 EOR piece:BPL tjo:STY temp
3490 PLA:CLC:ADC temp:TAY:LDA (&70),Y
3500 BNE tjo2:STY yt2:LDA yt:LDY joff
3510 STA ms,Y:INC joff:LDY boff:LDX #0
3520 .pulo LDA ms,X:STA (&72),Y:INY
3530 INX:CPX joff:BNE pulo:LDA yt2
3540 ORA #&80:STA (&72),Y:INY
3550 STY boff:LDA yt:PHA:TAY
3560 LDA #0:STA (&70),Y:LDA temp:PHA
3570 TAY:LDA (&70),Y:PHA
3580 LDA #0:STA (&70),Y
3590 LDY yt2:LDA piece:STA (&70),Y
3600 STY yt
3610 JSR jmp2:LDA #0:LDY yt:STA (&70),Y
3620 PLA:STA temp:PLA:TAY:LDA temp
3630 STA (&70),Y:PLA:STA yt:TAY
```

```
3640 LDA piece:STA (&70),Y:DEC joff:RTS
3650 :
3660 .seval LDA #255:JSR seval2
3670 STA score
3680 LDA #0:JSR seval2:SEC:SBC score
3690 RTS
3700 .seval2 STA side:LDA #0:STA sc
3710 LDY #63
3720 .seval3 LDA (&70),Y:BEQ seval4
3730 EOR side:BMI seval4:INC sc
3740 .seval36 LDA (&70),Y:LSR A
3750 BCS seval4
3760 INC sc:INC sc:.seval4 DEY
3770 BPL seval3:LDA side:STA &74
3780 LDA #scratch% MOD &100:STA &72
3790 LDA #scratch% DIV &100:STA &73
3800 JSR lmoves:LDY #0
3810 .seval5 LDA scratch%,Y
3820 CMP #&FF:BNE seval6:LDA sc:RTS
3830 .seval6 INC sc:INY:LDA scratch%,Y
3840 BPL seval6:INY:BNE seval5
3850 :
3860 .copbrd LDY #63
3870 .copbrd2 LDA (&75),Y:STA (&77),Y
3880 DEY:BPL copbrd2:RTS
3890 ]NEXT
3900 ENDPROC
```

Personal Ads

BEEBUG members may advertise unwanted computer hardware and software through personal ads (including 'wants') in BEEBUG. These are completely free of charge but please keep your ad as short as possible. Although we will try to include all ads received, we reserve the right to edit or reject any if necessary. Any ads which cannot be accommodated in one issue will be held over to the next, so please advise us if you do not wish us to do this. We will accept adverts for software, but prospective purchasers should ensure that they always receive original copies including documentation to avoid any abuse of this facility.

We also accept members' Business Ads at the rate of 30p per word (inclusive of VAT) and these will be featured separately. Please send us all ads (personal and business) to MEMBERS' ADS, BEEBUG, Dolphin Place, Holywell Hill, St. Albans, Herts AL1 1EX. The normal copy date for receipt of all ads will be the 15th of each month.

Master 128, RGB colour monitor, Cumana 40/80 S/S disc drive, teletext adaptor. ROMs; Master ROM, InterWord, Spellmaster, Dumpmaster, ATS teletext ROMs. All books and manuals. Games; Revs +4 Tracks, Aviator, Grand Prix, Elite. £400. Tel. (0923) 671371.

Two double sided 40/80T disc drives cased £50 each, Music 5000 synthesiser £80, Miracle WS4000 V21/23 modem with battery backup £90, Prism 1000 V23 (1200/75) modem £25, Homebrew Maplin V21 (300/300) modem £25. Tel. (04027) 59309 eves.

Master Compact, View, Logo, etc. View manual, spare discs, Printer cable £285. Tel. (0753) 884360.

WANTED: Disc drive for the BBC B, complete with utilities disc and all documentation Tel. (0332) 556381.

Master 128 £250. D/D 40T SS with power supply £60. AMX Mouse + Superart and Pagemaker £55. Oxford Pascal for Master £30. ISO Pascal £30. Tel. (0245) 321285.

Archimedes A310 Colour: Arthur 1.2, User Guide, Programmer's Reference Guide (pt 1 & 2), ARM Assembly Language Programming (Cock-erell), BEEBUG serial link, 1st WordPlus, PC emulator, RISC User and discs, mouse mat, printer lead £900. Tel. (0442) 49378.

Nightingale Modem with Comstar ROM and documentation £25. BEEBUG magazine from issue 1 to date, all in binders. Best offer within a month of publication of this issue of mag. Tel. 01-363 7866.

BBC Master with TURBO Co-processor, 2 Peartree RAM/ROM cartridges, ACP 256k RAM, AC Panel, ACP ADFS, Pagemaker, Masterfile, Elite, Revs, Enthar 7, etc. Plus several manuals and books £340. Tel (0454) 778503.

Epson LQ-800 Printer with cut-sheet feeder. Excellent condition, only one year old. Selling in order to upgrade system £400 o.n.o. Tel. (0468) 61321.

PMS E2P-6502 Second Processor for Acorn Electron Plus One, 6 months old with full documentation and utility disc, will accept £50 o.n.o. Tel (0706) 812815.

Dual drive Graduate, teletext add-on. Written offers (from charities/ education preferably) to K&C, Pluto Moutern School, Hornsey Road, N7 7QT. Enquiries Tel. 01-263 0510.

WANTED: BBC model B, must be in working order. Tel. (0745) 825036.

BBC B 32k issue 7 with DFS and ATPL ROM board. Immaculate condition with dust cover and User Guide £280. ROMS; Wordwise Plus £20, Spellcheck II £15, Help II £15, Toolkit Plus £20, Watford Transfer £18, InterWord £30, all complete with manuals and key strips Tel. (0827) 58119.

BBC B issue 7 40/80T disc drive. Multiprom EPROM programmer. Programs any size of 28 pin EPROM. Unipac EPROM eraser + software £320. Tel. 041- 641 1200.

Daisywheel printer - Olympia RO. Serial and parallel ports; bidirectional; tractor feed, BBC/IBM compatible. Brand new and boxed Tel. (0376) 40649 after 6pm.

BBC B with OPUS DDOS, as new in original box with instruction manuals leads and software. £220 o.n.o Tel. 01-441 1019.

Miracle Modem V2123 excellent condition. Upgradeable £199 o.n.o Tel. (0908) 665806.

BBC Acorn Archimedes 310 colour as new in original packaging, can be used as an IBM clone only £945. Sanyo 14" RGB colour data display for BBC computer £102 o.n.o Tel. (0234) 267067.

BBC B games on 5.25" discs; Elite, Philosoper's Quest, Maze, Rocket Raiders, Snooker, Very little used. The lot £10. Tel. (09323) 44999.

Virtually new Archimedes 310M colour, all manuals magazines £750. 65C102 turbo board £65. 80186 Co-processor £60. Watford Co-pro adaptor £20. Used KAGA Vision III colour £100. Viglen 20MB Winchester £150. Twin 40/80 floppy drive PS £80. Tel. (040 22) 29912.

InterBase (latest, boxed) £30. Replay £20. Watford 32k Shadow RAM board £25. HR Electronics Eprom Programmer £55. Exmon II £15. Dixons data recorder (auto stop/ start) £15. Tel. (0245) 469266.

ROMS; Watford Electronics Dump-out 3 and BEEBUG's Command communication ROM £15 each; InterWord £25 o.n.o all with manuals Tel. (0453) 844624 after 9pm.

WANTED; BBC Master 128, preferably area of Ipswich and Colchester. Tel (0473) 310487.

Continued on page 65

ASTAAD

This keystrip is designed to be used with the Astaad on pages 27 to 32 of this issue. Either cut out the keystrip or photo-copy this page and place under the plastic strip on your micro when using this program.

Key f0	Key f1	Key f2	Key f3	Key f4	Key f5	Key f6	Key f7	Key f8	Key f9
Print screen	Vector/ rect	Solid or dotted	Margins	Mirror image	Set ECF pattern	Scale	Link STEAMS	Save screen	Load screen
Soft ASTAAD	Copy or move	Line or arrow	Circle or arc	Polygon ellipse	Infill outline	Colour reverse	Line fix or trans	2 or 16 steps	Origin Rel/Abs
ASTAAD	Area move	Line	Circle	Polygon	Repeat f2, f3, f4	Move	Draw	Rubout	Delete area

STEAMS

This keystrip is designed to be used with the Astaad on pages 27 to 32 of this issue. Either cut out the keystrip or photo-copy this page and place under the plastic strip on your micro when using this program.

Key f0	Key f1	Key f2	Key f3	Key f4	Key f5	Key f6	Key f7	Key f8	Key f9
Print screen			Margins				Link ASTAAD	Save screen	Load screen
	Copy or move	Overwrt overlay							Origin Rel/Abs
Switch screens	Area move	Mix screens	Shrink screen					Delete borders	Delete box

This month's hints and tips have been collected together by Mike Williams. Remember that we pay five pounds for each hint published, and fifteen pounds for the star hint.

*** * * STAR HINT * * ***

STRING SPACE

by Laurence Cox

When assigning any character string to a string variable (or element of a string array), the amount of space allocated is equal to the length of the string if it is 7 characters or less, and equal to the length of the string plus 8 bytes if it is 8 characters or more. This is particularly serious for string arrays where the string length is just over 8 bytes as it can, in principle, lead to a space requirement of up to twice the size really needed. If space is really critical, then using the string indirection operator ($) will get round this.

OVERPRINTING

by F.Duerden

Some members may have had the problem of overprinting - i.e. of 'dropping' a section of 'local' text accurately into a pre-printed layout. The following does this very well:

```
250 *WINDOW
260 PROCoverprint
... .......

290 DEF PROCoverprint
300 REPEAT
310 VDU2,1,27,1,56,3
320 X=GET
330 VDU2,1,27,1,65,1,36
340 FOR N=1 TO 4:VDU1,10:NEXT
350 VDU2,1,27,1,65,1,6
360 FOR N=1 TO 4:VDU1,10:NEXT
370 X=GET
380 *GDUMP 0 1 2 2 40
390 UNTIL FALSE
400 ENDPROC
```

The required text is assumed to have been set up on the screen bywhatever means you choose (BEEBUG's Hershey Characters or any other text/graphics package). The heart of the printing is the use of VDU1,10 at lines 340 and 360 to avoid mis-placed text arising from the use of PRINT. Line 330 sets line spacing to a massive 0.5 inch, and line 340 gives a fast run of 14 such steps, followed by 4 delicate steps (lines 350 and 360). These should be adjusted to suit your own positioning needs.

Lines 250 and 380 are as needed with Computer Concepts' Printmaster - substitute the commands for your own screen dump here. Lines 300 and 390 allow you to print as many copies as needed, while line 370 can be omitted altogether after a satisfactory trial run.

RECOVERING WORDWISE

by Al Harwood

In the Hints & Tips for BEEBUG Vol.7 No.7 there was a routine to recover text from View after pressing Break. In Wordwise you cannot lose text by pressing Break, unless you are previewing or printing text.

In these circumstances the following routine can be used to recover the text. To do this you should type *BASIC (or Ctrl-Break), and once in Basic type CHAIN"RECOVER". After a short wait you will be asked for a file name. Enter one and your text will be saved in ASCII format. The routine then re-enters Wordwise and automatically loads the text file in for you.

```
 10HIMEM=PAGE+&300:DIM osc 50
 20PRINT"'"Wordwise Text Recover"
 30PRINT"By Al Harwood"
 40PRINT"Please wait";
 50A%=PAGE+&4C4:REPEAT:A%=A%+1:UNTIL$A%=
CHR$129+"End"
 60INPUT'"Filename: "fn$
 70OSCLI("S."+fn$+""+STR$~(PAGE+&4C5)+"
" +STR$~(A%-1))
 80OSCLI("K.0*W.|M4"+fn$+"|M")
 90*FX138,0,128
```

Save this as the program RECOVER. In line 90 you may need to expand the abbreviation 'W.' to 'WORDW.' if you also have View installed, to avoid conflict. B

PLAY IT AGAIN SAM 7

NOW ALSO AVAILABLE FOR THE ELECTRON

FIRETRACK
Electric Dreams' Smash Hit
Micro User gave Firetrack a maximum ..... rating and enthused that "Firetrack is for the player who thrives on danger, a pilot with nerves of steel and lightning reflexes". This superbly addictive game, with beautifully detailed graphics, is now available again for the BBC Micro and is available for the first time ever for the Electron.

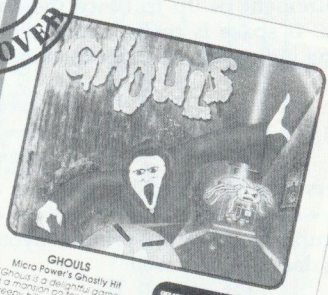
A Superior Hit

BONECRUNCHER
Superior's Humorous Soap Opera
Play the part of Fang, a green dragon, who collects skeletons for the spiders, ghosts, monsters and ghosts!
With smooth four-directional scrolling and often fiendish puzzles, Bonecruncher will delight game players of all ages. "Another Turbo hit" Micro User

SUPERIOR APPROVED

SNAPPER
Acornsoft's Arcade Classic
"Snapper is an excellent reproduction of the popular arcade game. Acornsoft have really excelled themselves yet again with Snapper. The maze is clear and the fruit are all very realistic. Summary: Totally excellent classic game. Brilliant!" A & B Computing

SUPERIOR APPROVED



GHOULS
Micro Power's Ghostly Hit
"Ghouls is a delightful game set in a mansion on top of a creepy hill. The graphics of Ghouls is impressive. The sound in Ghouls is superb. Some of the tunes are creepy and would be well placed in a horror film. Ghouls is a great game." Acorn User

PLAY IT AGAIN SAM 7 for the BBC Micro and Acorn Electron

Superior Software has combined three classic hits from other software houses (one new to the Electron) with one of its own top hits. The result is a top quality four-game compilation that combines variety with great value for money.

BBC Micro Cassette	£9.95	Acorn Electron Cassette	£9.95
BBC Micro 5¼" Disc	£11.95	BBC Master Compact 3½" Disc	£14.95

(Compatible with the BBC B, B+ and Master Series computers)

Please make cheques payable to "Superior Software Ltd".

(The screen pictures show the BBC Micro versions of the games.)

PLAY IT AGAIN SAM COMPETITION

Each copy of Play It Again Sam 7 contains a PLAY IT AGAIN SAM COMPETITION card.

Answer some straightforward questions on the card about previous Play It Again Sam compilations, and suggest possible games for future compilations, and you may be one of ten prizewinners.

Each prizewinner will receive all the previous Play It Again Sam compilations, or any other six games of his/her choice from the Superior Software BBC/Electron catalogue, and in addition will receive all future Play It Again Sam compilations on the day of release.

SUPERIOR SOFTWARE Limited

ACORNSOFT

(Acornsoft is a registered trademark of Acorn Computers Ltd. Superior Software Ltd is a registered user.)

Dept 7PS5, Regent House, Skinner Lane, Leeds LS7 1AX. Telephone:(0532) 459453

Available from WHSMITH and all major dealers

24 HOUR TELEPHONE ANSWERING SERVICE FOR ORDERS

VISA

OUR GUARANTEE

- All mail orders are despatched within 24 hours by first-class post.
- Postage and packing is free.
- Faulty cassettes and discs will be replaced immediately. (This does not affect your statutory rights)

BBC B series 7 with solidisk 256k/4meg expansion board fitted and working. Solidisk DFDC 8271/1770 controller giving ADFS and DFS; Zif socket. Wordwise Plus and Spellcheck ROMS. Cumana 80T disc drive; ROMS available Sleuth, Watford DFS, Toolkit, Catkit, Diary ROM; BEEBUG magazines vols 1 to 6 complete, BEEBUG magazine discs vols 5 and 6 almost complete; 100 disc storage box, various disc based progs including Dumpmaster, Studio Eight, Masterfile II, BEEBUG Filer, Wordease, Comp-u-Cater, Glider Pilot and Notebook, all with documentation. All very good condition, will split £450 o.n.o Tel. (0454) 416841.

Printer NEC 8023 (dot matrix) with spare ribbon. In original packaging with cable for BBC £80 o.n.o BEEBUG Dumpmaster ROM £10, 40T Shugart D/D £40 o.n.o Tel. (0258) 480100.

HCR External fully cased ROM/RAM unit. Capacity 15 ROMs. Includes 4x16k SWRAM fitted. Own PSU, mint £40. Also Acorn 6502 2nd Processor £50. Tel. (0483) 38285.

Vieglen dual DS 40/80T drive £110. Kaga 12" green monitor £25. Buyer collects BBC issue 4 with DFS £130. Wordwise Plus £20. Masterfile II £10. BEEBUG vol 1 no 5 to vol 7 no 7 £4 per vol. Tel. (0703) 891606.

BBC B issue 4, DNFS, BASIC 2. Teac 40/80 D/S D/D. Microvitec 1431 colour monitor. M. Tally MT80 printer, complete system £450. (might split) Tel. (0827) 715170.

Racal Modem comlink IV with documentation V24/RS232C with optional V10 interface. Offers! WANTED: Original or copy of Enigma Disc Imager ROM manual. Tel. 01-429 1868.

Master 128 plus ViewStore, printer controller & BEEBUG Master ROMS, utility discs, Acorn Master reference manuals, Dabs Press & AcornView, ViewSheet & ViewStore manuals, cumana 40/80T DS (400k) drive & Microvitec Cub colour monitor. All excellent condition £495. Tel. (0460) 40991.

ATPL ROM board £25. Mini Office II (Master) disc plus Dabhand guide & disc £20. Mini Office driver disc £12. Hershey Font £9. Tel. (0203) 465208.

BBC B issue 3 board with Acorn DFS some software and books, Voltmace Joystick and data recorder £195 o.n.o Tel. (0773) 769819 after 6.30pm or (0602) 785011 during the day.

Master Compact entry system, hardly used includes TV modulator, welcome manual & disc plus 9 copies of The Micro User £230 o.n.o Tel. (0382) 76128 after 7pm.

12 programs for BBC B £15 or swap for AMX Art or AMX Super Art. Tel. (0924) 274442.

WANTED: 2 books "Get More from BBC Micro Machine Code" by A P Stephenson/Granada. "Machine Code Programming" by Ian Birnbaum. Your price paid or would exchange new Cumana 40TS/S uncased disc drive for both books. Tel. (0481) 45866 9am until noon.

Acorn Electron with Plus 1, Plus 3, Slogger Turbo, View, ViewSheet, Database and games. All with manuals and boxed £150. Watford Electronic RAM ROMboard with 32k dynamic RAM £30. Tel. (0209) 860284.

Epson LQ800 printer (upgraded to laser) £150. Kaga Vision II RGB monitor £75. Psion Organiser IIXP £75. Interbase ROM (latest version unused) £35. All as new. Tel. (0536) 200094.

Tandy DMP105 dot matrix printer with tractor feeder attachment, instructions, lead for BBC B, boxed VGC £75. Tel. (0344) 51308.

WANTED: Back issues of BEEBUG magazine vols 1 to 5. Tel. (0373) 65675.

Solidisk system & keyboard cases (steel) with keyboard cable, to house computer and 2 drives, plus expansion boards. For Model B £20. Tel. (044282) 4543.

Master 128 with Acorn Tube 656c102 Co-processor and manuals. Watford Electronics DP35-800S Bridge Unit Disc Drives 5.5 6 3.25 Watford Mouse. Single 3.5 D/D 800k. Two quad cartridge holders. Microvitec Cub colour monitor (commercial model in metal case) complete £850 o.n.o. Tel. (09277) 63689.

BBC Master 512, Microvitec colour monitor, Pace D/D + PSU, lots of disc and ROM software, Habitat computer desk. Tel. 041-956 6106.

Master 128, with additional Viewstore and Prestel, Philips mono monitor, Cumana 800S 40/80 DD drive, Cassette player, Prestel modem + all leads, manuals and reference manuals 1&2, 30 discs. £450. Tel. (0491) 571338.

WANTED: W.H.Smith data recorder model CPD 8300. Must be in good condition. Tel. (0626) 864120.

Apollo Modem and software £50. Cannon DS single 40/80T D/D £95. Disc doctor £15. Advanced disc investigator £20. MuROM £10. ROMit £15. All boxed with manuals. Tel. (0904) 31313.

BBC B series 7 with leads, manual and welcome cassette, hardly used £200. Tel. (0246) 590546.

AMX Mouse £12. Stop Press £25. Extra Extra £10. All for use with the Master 128. Tel. (0474) 363503.

Epson XX88 Printer plus tractor unit, excellent condition £160. Tel. (09274) 23167.

BBC Master 128, Master colour monitor, twin D/D DDFS prints mounted. BBC teletext adaptor. Epson LX80 with sheet feeder, Demon modem and AMX mouse + joystick. ROMS, ROM cartridges + discs (SpellMaster and Master ROM included) total cost over £2,000. £950 o.n.o. Tel. (0780) 87784.

65C102 2nd processor (turbo) fitted into Acorn Universal 2nd processor unit. Boxed and unused. £145. Tel. (0709) 375135.

BEEBUG MEMBERSHIP

Send applications for membership renewals, membership queries and orders for back issues to the address below. All membership fees, including overseas, should be in pounds sterling drawn (for cheques) on a UK bank. Members may also subscribe to RISC User at a special reduced rate.

BEEBUG SUBSCRIPTION RATES

		BEEBUG & RISC USER
£ 7.50	6 months (5 issues) UK only	£23.00
£14.50	1 year (10 issues) UK, BFPO, Ch.I	£33.00
£20.00	Rest of Europe & Eire	£40.00
£25.00	Middle East	£44.00
£27.00	Americas & Africa	£48.00
£29.00	Elsewhere	

BACK ISSUE PRICES (per issue)

Volume	Magazine	Tape	5"Disc	3.5"Disc
1	£0.40	£1.00	-	-
2	£0.50	£1.00	-	-
3	£0.70	£1.50	£3.50	-
4	£0.90	£2.00	£4.00	-
5	£1.20	£2.50	£4.50	£4.50
6	£1.30	£3.00	£4.75	£4.75
7	£1.30	£3.50	£4.75	£4.75

All overseas items are sent airmail. We will accept official UK orders for subscriptions and back issues, but please note that there will be a £1 handling charge for orders under £10 which require an invoice. Note that there is no VAT in magazines.

Destination	First Item	Second Item
UK, BFPO + Ch.I	60p	30p
Europe + Eire	£1	50p
Elsewhere	£2	£1

POST AND PACKING
Please add the cost of p&p as shown opposite.

BEEBUG
Dolphin Place, Holywell Hill, St.Albans, Herts AL1 1EX
Tel. St.Albans (0727) 40303, FAX: (0727) 60263
Manned Mon-Fri 9am-5pm
(24hr Answerphone for Connect/Access/Visa orders and subscriptions)

BEEBUG MAGAZINE is produced by BEEBUG Ltd.

Editor: Mike Williams
Assistant Editor: Kristina Lucas
Technical Editor: David Spencer
Advertising: Sarah Shrive
Production Assistant: Sheila Stoneman
Membership secretary: Mandy Mileham
Editorial Consultant : Lee Calcraft
Managing Editor: Sheridan Williams

CONTRIBUTING TO BEEBUG PROGRAMS AND ARTICLES

We are always seeking good quality articles and programs for publication in BEEBUG. All contributions used are paid for at up to £50 per page, but please give us warning of anything substantial that you intend to write. A leaflet 'Notes to Contributors' is available on receipt of an A5 (or larger) SAE.

Please submit your contributions on disc or cassette in machine readable form, using "View", "Wordwise" or other means, but please ensure an adequate written description is also included. If you use cassette, please include a backup copy at 300 baud.

In all communication, please quote your membership number.

Printed by Head Office Design (0782) 717161 ISSN - 0263 - 7561

Magazine Disc/Cassette

JAN/FEB 1989 DISC/CASSETTE CONTENTS

CENTRES OF GRAVITY - use this program to create two dimensional objects and determine their centre of gravity.

ADFS DIRECTORY DELETER - delete ADFS directories and all sub-directories and files with this powerful utility.

AUDIO TAPE INLAYS - edit and print correct size contents lists for inserting in your audio cassettes.

VERSATILE ROM MANAGER - implements four new star commands to allow all users to manage the ROMs installed in their machines.

REAL-TIME CLOCK FOR ALL - this program implements a continuously updated clock and alarm system for permanent display on the screen.

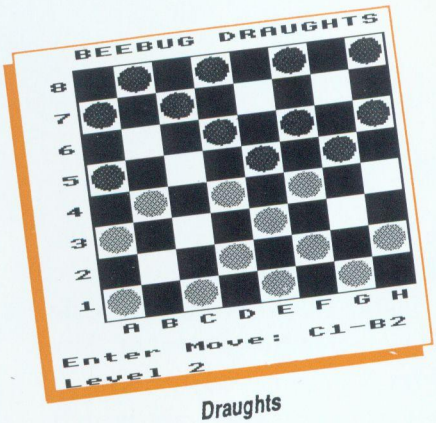
STEAMING AHEAD WITH ASTAAD - a further supporting program to manipulate graphics objects produced with our CAD program ASTAAD.

MODE 7 GRAPHICS CHARACTERS - a short utility allowing all the teletext graphics characters to be displayed or printed in a logical visual order for easy reference.

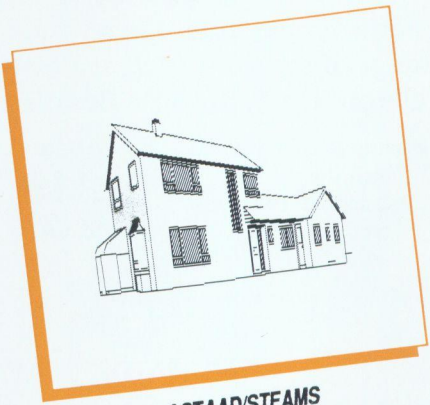
FILE HANDLING FOR ALL (Part 8) - a fully working example of a database system using both internal and external pointers.

DRAUGHTS - a clever implementation of this board game based on the principles described in our current workshop series.

MAGSCAN - bibliography for this issue (Vol.7 No.8).



Draughts



ASTAAD/STEAMS

All this for £3. 50 (cassette), £4.75 (5" & 3.5" disc) + 60p p&p (30p for each additional item).
Back issues (5.25" disc since Vol.3 No.1, 3.5" disc since Vol.5 No.1, tapes since Vol.1 No.10) available at the same prices.

SUBSCRIPTION RATES	UK ONLY			OVERSEAS		
	5" Disc	3.5" Disc	Cassette	5" Disc	3.5" Disc	Cassette
6 months (5 issues)	£25.50	£25.50	£17.00	£30.00	£30.00	£20.00
12 months (10 issues)	£50.00	£50.00	£33.00	£56.00	£56.00	£39.00

Prices are inclusive of VAT and postage as applicable. Sterling only please.

Cassette subscriptions can be commuted to a 5.25" or 3.5" disc subscription on receipt of £1.70 per issue of the subscription left to run. All subscriptions and individual orders to:

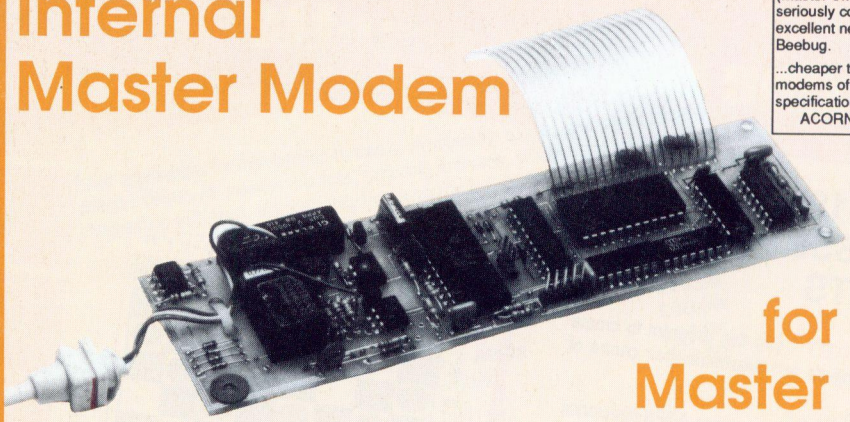
BEEBUG, Dolphin Place, Holywell Hill, St.Albans, Herts. AL1 1EX.

Internal Master Modem

(Master-owners)...should seriously consider this excellent new product from Beebug.

...cheaper than many other modems of a similar specification...

ACORN USER OCT 88



for the Master 128

The Beebug Master Modem is a high quality internal modem for the Master 128. It is easy to install and requires no soldering. The Master Modem is supplied with the Command software which allows easy access to a whole range of services, such as Prestel, Telecom Gold, Bulletin Boards etc.

Features include

- fits internally in Master 128
- just plugs in - no soldering reqd.
- 300 baud full duplex (V21)
- 1200/75 baud half duplex (V23)
- auto-dial facility
- V25 auto-answer facility
- line monitoring through computer loudspeaker
- tinkle suppression
- supplied with Beebug's highly acclaimed Command software

Beebug Master Modem including fitting instructions, software and user guide

price £119 inc VAT

APPROVED for connection to telecommunication systems specified in the instructions for use subject to the conditions set out in them.

S/3113/3/H/501018

Certain early versions of the Master 128 had their battery packs located in the modem fitting position. If your battery pack is NOT located to the left of the keyboard, it will need replacing with the new style battery pack (price £3.68).

To claim Beebug members price it is essential to quote your membership number

Please supply ___ Master Modem(s) at £119 ❑ or £113.05 memb. price ❑ Carriage £4

Please supply ___ New style battery pack(s) at £3.68 ❑ or £3.50 memb. price ❑ Carriage £1

Name: _____ **Membership No.** _____

Address: _____

_____ **Postcode:** _____

Access/Visa No. | | | | | | | | | | | | | | | | |

ACCESS/VISA Expiry Date | | |

Beebug Ltd. Dolphin Place, Holywell Hill, St. Albans, Herts., AL1 1EX Tel (0727) 40303