Vol.10 No.10 April 1992

BEEBUG

FOR THE BBC MICRO & MASTER SERIES

Musical Muskrats

- HIDDEN PERSUADERS
- STOREPRINT
- PARADOX
- CROSS REFERENCE LISTER

BEEBUG Vol.10 No.10 April 1992

FEATURES

The Hidden Persuaders 6
Musical Muskrats 8
Ray Tracing in 2D (2) 13
Wordwise User's Notebook 18
Cross Reference Lister 23
First Course: Pseudo-Variables (3) 29
Zeus II: The Final Conflict 34
512 Forum 37
BEEBUG Workshop: Finding a Route in a Network 40
Public Domain Software (3) 44
Mr Toad's Machine Code Corner 47
Storeprint 49
BEEBUG Function/ Procedure Library (10) 51

REVIEWS

Paradox 17

REGULAR ITEMS

Editor's Jottings 4
News 5
Hints and Tips 57
RISC User 58
Postbag 59
Personal Ads 60
Subscriptions & Back Issues 62
Magazine Disc 63

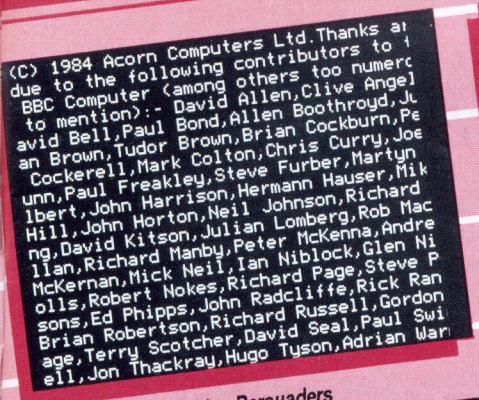
HINTS & TIPS

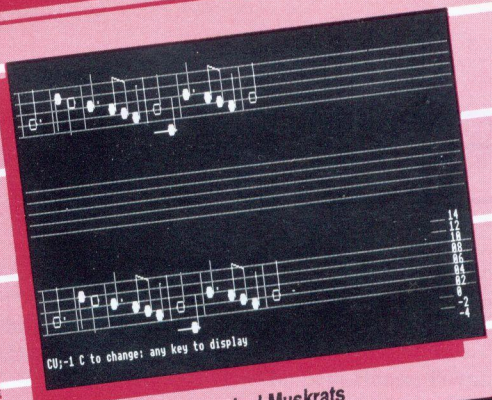
Programming the f-Keys for Mode 7
Next Best Thing
How Do You Spell ADFS

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 |.

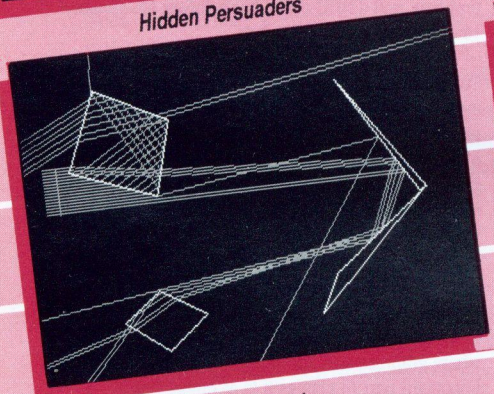
All programs in BEEBUG magazine will run on any BBC micro with Basic II or later, unless otherwise indicated. Members with Basic I are referred to the article on page 44 of BEEBUG Vol.7 No.2 (reprints

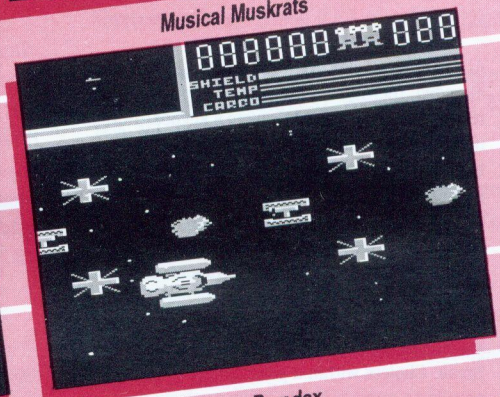




Musical Muskrats

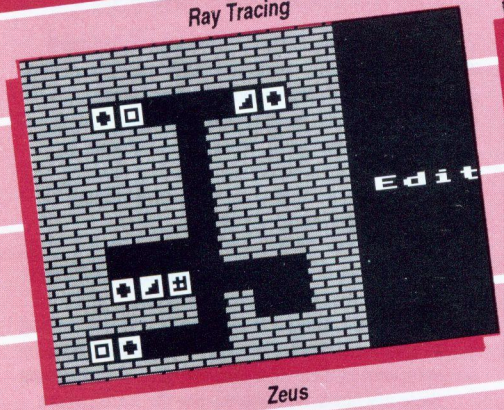
Hidden Persuaders

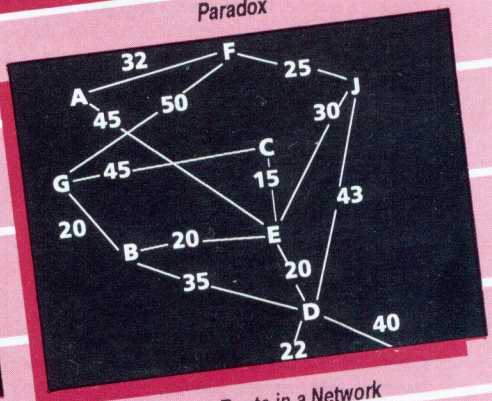




Paradox

Ray Tracing





Finding a Route in a Network

Zeus

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 needs at least one bank of sideways RAM.

Program is for Master 128 and Compact only.

Editor's Jottings

At a recent one day show for teachers which we attended, there was quite a lot of discussion of the relative merits of the BBC micro's keyboard-oriented user interface, and the more sophisticated Wimp interface (using a mouse to point and select) of the Archimedes.

When using computers with younger age groups within the primary school environment, there was a feeling that simplicity of use and display had a lot going for it. Typically, a program would be started by placing the disc in the drive and pressing Shift-Break. Choices within a program are often made by pressing a number key to make a selection. Familiarity can be very important in ensuring that both children and teachers are able to make the best possible use of expensive equipment.

The discussion arose from a dilemma which must face many schools, and indeed all users, from time to time. If your computer should break down, is it worth spending money on repair, or is it better to upgrade to a Wimp system like the A3000? This is by no means a simple question to answer, as it depends so much on individual circumstances.

Perhaps it is easier for the home user with a single system. He only has to decide for himself, and it is *his* money. My view would be that if you are entirely happy with the performance of your existing system, and the software you use on it, then there is little point in upgrading, an option which will nearly always cost more money anyway. However, like all things there ultimately will come a time when BBC micros will become just too expensive to repair (unless they attract an inflated value as a collector's item!), and the necessary spare parts will become increasingly difficult to find (in our last clearance sale 20 Beeb power units were snapped up in no time at all). At present there is not too much of a problem, and a BBC micro has a future life of many years yet as I have said before.

The position for schools is different. Many schools, even primary schools, will have several machines and there are clear advantages in maintaining sets of the same type - that supports repairing BBCs as necessary. However, the requirements within schools will not remain static, but will continue to grow, and whilst this may not demand intrinsically more sophisticated machines (i.e. Wimp-based systems), there is and will be a growing demand for systems with more power than the BBC micro can ever hope to muster. And this is as true of primary schools where graphics and animation can have an important part to play as it is at higher levels.

Schools are also likely to be working under much tighter budgetary constraints - any expenditure has to be made in competition with many other demands, and a wider variety of pressure groups will have to be considered. What I would venture to suggest is that every school, including primary schools, should at least try to evolve a strategy for its on-going computing provision.

An old principle, which still holds sway today in my view, says that any hardware provision should be driven by the software which you need to use. As long as your software needs can be met by a BBC micro, or even some of them, then there is value in maintaining these machines. Add systems like the A3000 where there is a clear educational need for software of a type which cannot be run on anything simpler.

And at the end of the day, remember that much of the simpler software which runs so well on the BBC micro can be used very often on an Archimedes, thereby getting the best of both worlds.
M.W.

BETT 1992

We conclude our summary of products and companies catering for the BBC micro which we have identified following the BETT show in January.

Micro-Aid provides software to compile family trees, and includes a Royal family sample, used in GCSE projects. *Microvitec* was showing two new touchscreens for use with the BBC model B and Master. *MJP-Geopacks* offers an automatic weather station, and a dual measurement *Flowmeter* for measuring wind and water velocity. *MUSE* is another organisation with software for the BBC micro, and runs supporting courses.

Northwest SEMERC launched the Oldham Overlay Keyboard, a touch sensitive A3 keypad for use with overlay keyboard software. *Storm Software* was showing new titles and enhanced versions of its role-play and adventure programs for the BBC and other systems. *Vision Software (Disney) Ltd.* has a range of subject specific educational software for use in schools, colleges and at home. Specialising in special needs and primary, *Wigit Software* also has products for the BBC micro range.

Micro-Aid, Kildonan Courtyard, Barrhill, South Ayrshire, Scotland KA26 0PS, tel./fax (0465) 82288.

Microvitec plc, Bolling Road, Bradford, West Yorkshire BD4 7TU, tel. (0274) 390011, fax (0274) 734944.

MJP-Geopacks, P.O.Box 23, St Just, Penzance, Cornwall TR19 7JS, tel. (0736) 787808, fax (0736) 787880.

MUSE, P.O.Box 43, Houghton on the Hill, Leicestershire LE7 9GX, tel. (0533) 433839.

Northwest SEMERC, Flitton Hill CDC, Rosary Road, Oldham, Lancashire OL8 2QE, tel. 061-627 4469, fax 061-627 2381.

Storm Software, Beth House, Poyntington, Sherborne, Dorset DT9 4LP, tel./fax (0963) 22469.

Vision Software (Disney) Ltd, 6 Pilkington Avenue, Sutton Coldfield, West Midlands B72 1LA, tel. 021-354 3981, fax 021-355 6929.

Wigit Software, 102 Radford Road, Leamington Spa CV31 1LP, tel. (0926) 885303, fax (0926) 825683.

ALL FORMAT FAIRS

Latest dates and venues for *All Formats Computer Fairs* are as follows:

Apr 12 Northumbria Centre, Washington, County Durham, A194(M).
Apr 26 National Motorcycle Museum, NEC, M42 J6.
May 16 Sandown Exhibition Centre, Esher, M25.
May 17 Brunel Centre, Temple Meads, Bristol.

For information and tickets contact John Riding on (0225) 868100. Note that the Sandown Exhibition Centre will be the venue for all future London fairs instead of the Horticultural Hall, Westminster.

EDINBURGH INTERNATIONAL SCIENCE FESTIVAL

Heriot-Watt University will be exhibiting various data-logging systems using BBC micros at this exhibition which takes place from April 11th to 25th. The systems were partly described in *Practical Electronics* for February 1991. For more more information contact R.W.Goulding, Dept. of Building, Heriot-Watt University, Riccarton, Edinburgh EH14 4AS. B

The Hidden Persuaders

David Holton investigates hidden features of the Master Operating System.

The BBC computer's various ROMs have all sorts of people's names scattered about in various places - programmers just can't resist leaving their signature on their work. If you have a memory editor that can handle sideways ROMs, just look at the last five bytes of the Master's Basic ROM; if not, type in this line of Basic:

```
FOR n%=0 TO 4:PRINT CHR$(n%?&BFFB);
:NEXT
```

DIG DOWN DEEP

That's only the tip of the iceberg! You may already be aware that hidden away in the Master's MOS ROM is a long list of names of people who worked on the production of the machine. It runs from &FC00 to &FEFF, and is normally paged out of the memory-map, as I/O devices are mapped to this area - pages FRED, JIM and SHEILA - but setting bit 6 of the access control register ACCON (&FE34) will select this part of the ROM and the names may be read. The program *Secret* is a short machine-code routine which does just this, copying the three pages of ROM into shadow screen memory; it then restores the old value in ACCON. Type it in and run it; it will assemble and call the machine-code routine.

Note that interrupts are disabled, as there is not quite time to do the copying between one interrupt and the next, and some interesting crashes occur if you try. The I/O area has to be mapped in during interrupts: if you fiddle around with this part of the code, make sure you've saved anything of value first!

My first thought was to copy the ROM to main memory, tidy up, then write a second loop to print out the data. I soon realised, however, that if we were forcing ACCON's bit 6, we might as well force bit 2 as well, which selects shadow memory &3000 to &7FFF, and copy the names to &7C00 onwards, placing them directly on the screen. This shortens the code a lot and, incidentally, is startlingly fast. You can do this in Teletext mode, in which the screen memory just holds the ASCII values of the characters and the Teletext chip outputs the video.

```
(C) 1984 Acorn Computers Ltd.Thanks a
due to the following contributors to
 BBC Computer (among others too numer
 to mention):- David Allen,Clive Angel
avid Bell,Paul Bond,Allen Boothroyd,Ju
an Brown,Tudor Brown,Brian Cockburn,Pe
 Cockerell,Mark Colton,Chris Curry,Joe
unn,Paul Freakley,Steve Furber,Martyn
lbert,John Harrison,Hermann Hauser,Mik
Hill,John Horton,Neil Johnson,Richard
ng,David Kitson,Julian Lomberg,Rob Mac
llan,Richard Manby,Peter McKenna,Andre
McKernan,Mick Neil,Ian Niblock,Glen Ni
olls,Robert Nokes,Richard Page,Steve P
sons,Ed Phipps,John Radcliffe,Rick Ran
Brian Robertson,Richard Russell,Gordon
age,Terry Scotcher,David Seal,Paul Swi
ell,Jon Thackray,Hugo Tyson,Adrian War
r,Jess Wills,Roger Wilson,Graham Winte
lood.
```

In modes 0 to 6 screen memory contains bit-maps of the display and this trick wouldn't work; you might as well call OSWRCH for all normal purposes. While we are at it, we might as well force bit 0 of ACCON as well - set shadow mode - just to be on the safe side. Two drawbacks to this direct poking of screen RAM: you can't send the data to the printer, and I had to write the equivalent of a PRINT TAB(statement to get the text out of the way of the cursor, since OSWRCH wasn't updating the cursor position.

The list contains some well-known names and the three pages are quite full; only the last 3 bytes are 'wasted', containing spaces. I also find it very interesting that you can write to ACCON to restore the I/O devices even though ACCON itself, being in SHEILA, has

mapped itself out. The ROM is read correctly - &FE34 contains the capital E in the name 'Ed Phipps' - but any write still goes to ACCON. It's a question of write-access and read-access not necessarily being the same. If it were not so, we'd be stuck with the list and a hung-up computer until power-off!

Incidentally, Acorn's reason for having text in this area of ROM appears to be that it's useful for testing the hardware. On page F.2-3 of the Master Reference Manual (Part 1), where each bit of ACCON is given a separate name, bit 6 is called TST and the book says it must be reset at all times.

GOING DEEPER

As I said, that particular list of names is quite well-known and is documented in at least one book. The next thing I'm going to show you isn't known at all: I think I am the only person to have discovered it, although one can never be quite sure of these things in computing. Again, a short routine will reveal something you didn't know was there. The text is rather well hidden - type in the *Reveal* listing, run it to assemble the machine code and then type:

```
CALL code
```

See what I mean? The text is clearly subject to some kind of protection. It can be got at, though, by a more devious stratagem: type:

```
CALL !&043C
```

and all will be revealed.

Interesting, isn't it!

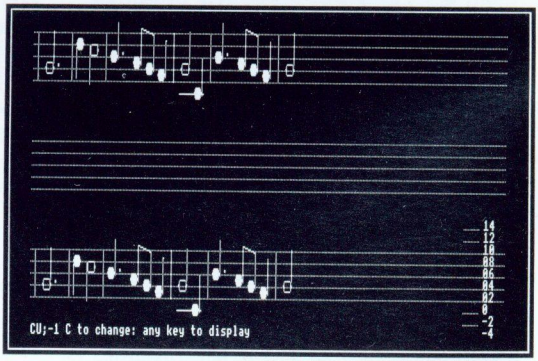
```
 10 REM Program Secret
 20 REM Author  David Holton
 30 REM Version B1.0
 40 REM BEEBUG  April 1992
 50 REM Program subject to copyright
 60 :
100 MODE 135
110 accon=&FE34
120 osasci=&FFEE
130 P%=&900
140 [ OPT 2
150 .code
160 LDA #&FC:STA &8D
170 LDA #&7C:STA &8F
180 STZ &8C:LDA #&50:STA &8E
190 LDX #3:LDY #0
200 LDA accon:PHA
210 LDA #&45:TSB accon:SEI
220 .loop
230 LDA (&8C),Y:STA (&8E),Y
240 INY:BNE loop
250 INC &8D:INC &8F
260 DEX:BNE loop:CLI
270 PLA:STA accon:RTS
280 ]
290 CALL code
```

```
 10 REM Program Reveal
 20 REM Version B1.0
 30 REM Author  David Holton
 40 REM BEEBUG  April 1992
 50 REM Program subject to copyright.
 60 :
100 osprint=&FFEE
110 acon=&FD34
120 key=&E530
130 P%=&7000:O%=P%
140 [ OPT 2
150 EQUD &18B900A0
160 EQUD &701159E2
170 EQUD &C8FFE320
180 EQUD &F2D012C0
190 EQUD &A8CEA360
200 EQUD &6CC3C9D6
210 EQUD &6180919A
220 EQUD &BE85B7EA
230 EQUD &00C7DB86
240 :
250 .code
260 LDA acon:ORA #&80
270 STA acon:LDX #0
280 .loop
290 LDA key,X:PHP:JSR osprint
300 INX:PLP:BNE loop
310 LDA acon:AND #&7F
320 STA acon:RTS
330 ]
340 REM First type <CALL code>
350 REM Then do <CALL !&043C>
```

Musical Muskrats

Philip Prior breathes new life into a BEEBUG classic.

Dr.C.A.A.Wass's music printing and transposition program *MUSTRAN* (BEEBUG Vol.6 No.8 January/February 1988) is a valuable aid to anyone producing musical scores. However, I found it useful to add facilities to save and load data files of the music previously entered and these additions form the basis of the amendments described in this article. Also included is a method by which the program can be terminated from the symbol entry section.



Let's start by looking back at the original program, the full listing of which is provided again this month.

MUSTRAN

MUSTRAN is a program to display and print music on normal five-line staves, with automatic transposition if selected. It will cope with all the different types of note and sharps, flats, naturals and rests as well other features. All input is coded and entered via the standard keyboard.

USING THE PROGRAM

Type the program in and save it. When the program is run the screen first prompts for transposition up (U) or down (D) and for the number of steps. If transposition is not required then either U or D may be pressed, followed by 0. Note that neither here, nor elsewhere on input, is the use of Return needed.

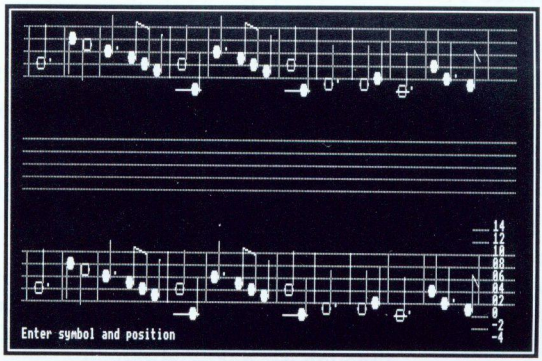
Either mode 4 or mode 0 may be selected for the screen display. Mode 4 may be used for demonstration and easier reading of prompts, but mode 0 will usually be needed to display a full line of music.

Three staves are displayed on the screen with the lines of the lowest stave numbered 01, 03, 05, 07, 09, and the spaces 02, 04, 06, 08. For clarity, only the even numbers are displayed. Above the stave the numbers continue with 10, 11, 12 etc., and below the stave the numbers descend, with 00 ,-1, -2, etc. Notes are entered using two-letter codes and displayed on the bottom stave, then transposed by the program on to the top stave. Once this line of music is complete further notes may be entered and transposed to the middle stave.

A flashing prompt appears at the bottom of the screen for the entry of a symbol and position. All the symbols are entered with one of the following codes:

SH — Sharp
NA — Natural
FL — Flat
WW — Semibreve ('W'=White; 'S' is used for semiquaver)
MU — Minim with stem up
MD — Minim with stem down

CU — Crotchet with stem up
CD — Crotchet with stem down
QU — Quaver with stem up
QD — Quaver with stem down
SU — Semiquaver with stem up
SD — Semiquaver with stem down
DT — Dot
RW — Semibreve rest
RM — Minim rest
RC — Crotchet rest
RQ — Quaver rest
RS — Semiquaver rest
BL — Barline
CS — Change stave
BQ — Single joining bar (quavers)
BS — Double bar (semiquavers)
E — Erase
P — Print



Enter the two letter code followed immediately by a TWO digit position number. Once these codes have been entered they will be displayed, and if they are correct then any key except C may be pressed. This will display the symbol in its correct place on the lower stave and also in its transposed position on the top stave. If C is pressed after the entries have been displayed, then they will be ignored, and you will be prompted for a new entry. C may also be pressed with the same effect after the entry of just the two-letter code.

If you need to display a pair of quavers or semiquavers with a bar or double bar joining their stems, then they should first be entered and displayed as crotchets. The bar can then be added by pressing BQ or BS as appropriate.

If any symbol that has just been displayed on the stave needs to be changed then key E may be used. This will draw a black vertical stripe on both staves, and repeated use will erase a rectangular patch. This is rather slow, but it is useful because the left-hand edge of the next symbol will appear at the position of the last black stripe. If more space is needed to the right of the last displayed music character, then entering any two-letter combination, other than those in the above list, will print an invisible character, and the position for the next character will be moved along by that amount.

This position can be found by pressing key E once, and the space can be reduced by repeated presses of E. The E key can also be used to reduce the space between symbols, for example to compact a key signature.

When the top line of music is complete, code CS should be entered. The bottom stave will then clear and entering symbols can continue. They will be displayed as before on the bottom stave but now the transposed version will appear on the middle stave. The next use of the CS command will clear all staves ready to re-start at the top.

When the two upper staves are full the screen display may be sent to a printer by typing P. As written the program calls Computer Concepts' Printmaster ROM

to produce a screen dump using *GDUMP 0 1 2 1 0 (see line 2490). This gives full-size music, ready for playing. This line could be changed to call other printer dump routines (e.g. BEEBUG's Dumpmaster), or a routine of your own. In fact, the print routine may be called at any time.

If the transposition involves a key change the key signature will not be changed automatically. However, the sharps or flats needed for the new key can be entered on the bottom stave and so positioned that they will appear in their correct positions on the upper staves after transposition. For a transposition from F major, with one flat, to G major, with one sharp, notes will be raised by two steps. If then a sharp is entered on line 07 on the bottom stave it will appear in its correct position on line 09 on the upper staves.

UPDATING THE ORIGINAL PROGRAM

In order to integrate the new facilities into *MUSTRAN*, and still allow execution of the program on BBC Micros without sideways RAM, it has been necessary to divide the program into two sections, one for creation of data files, and the other for the loading of data for printing. The two sections mentioned above correspond to two different programs. These start off as copies of the original program and I have called them *MUSKRAT* and *MUSKOX* respectively to create and load programs.

TECHNICAL NOTES

A new procedure, PROCfile, has been added to those in the original program. It prompts the user for a filename and then opens the file for use in the main parts of both programs.

Several new variables have been added:

EP% is a flag variable taking either a value of 1 or 0, used in the termination of the programs under the correct conditions.

G% contains the value of a GET variable in both programs.

PR% is another flag variable corresponding to the printing of the score after plotting is complete. It is set to either 1 or 0 depending on the response of the user to the question in line 110.

A$ holds the user's answer to the question in line 110 of MUSKOX.

X - This is the identifier of the files opened and closed in both programs.

TERMINATION OF PROGRAMS

In the original program, the only way to exit from the symbol entry part was by pressing Escape or Break. A facility has been added in *MUSKRAT* so that entering QQ as the response to the prompt for the next symbol closes the created file, and then terminates the program through the use of the variable *EP%*. This also appears in *MUSKOX*, although it is not so important.

USING MUSKRAT

The operating instructions for this program are basically the same as those for the original *MUSTRAN*, with a few small changes. On entering the program, the new procedure, PROCfile, asks for a filename of the file to be created. This then opens a file of that name. Type in the score of the music using either one or two staves. As you type the data in, or change it, the program sends it to the file. When you have finished entering all the correct data, type QQ as the response to

the prompt for the next symbol and this will close the file and terminate the program. You should now load *MUSKOX* to print your score out.

USING MUSKOX

Having previously created your file, you should now enter its filename when prompted. If you require a printed copy of the score then you should press Y as your response to the next question. If you just wish to view the contents of the data file, enter N as your answer. The file will be loaded and the score plotted exactly as displayed before. Printing, if chosen, will occur here. The program will terminate after this has finished or, if a printed copy was not chosen, will terminate after plotting the score.

MUSTRAN

The original program forms the basis for the two new ones. It should be stored with the others, in case you do not wish to save the score, using *MUSKRAT*. If this is so, enter the data into *MUSTRAN*, and then print out the score if you wish to. You may find it helpful to amend *MUSTRAN* to allow termination of the program, by the use of QQ, to occur, by amending the following lines.

```
100 MODE 7:PROCtitle:EP%=0:ONERROR GO
TO 250
220 REPEAT:PROCchoosesym:IF EP%=1 THE
N 30 ELSE UNTIL FALSE
1720I$=GET$:K$=J$+I$:IF K$="QQ" THEN
EP% =1:ENDPROC
```

IMPLEMENTING THE NEW FACILITIES

To create the new versions of the program first type in the MUSTRAN listing and save it. Make sure you test it thoroughly before going on to the next stage.

MUSKRAT

To create the MUSKRAT program load in your tested version of MUSTRAN and add the following lines.

```
105 PROCfile:EP%=0
2570 :
2580 DEFPROCfile:CLOSE#0
2590 INPUT'"Name of file to be create
d :"; FI$
2600 X=OPENOUT FI$
2610 ENDPROC
```

Now change these lines:

```
120 T=GET:PRINT#X,T:VDUT,58:NS=GET: P
RINT#X, NS:VDUNS,13,10
160 M=GET-48:PRINT#X,M:MODE M
220 REPEAT:PROCchoosesym:IF EP%=1 THE
N 230 ELSE UNTIL FALSE
1010 FOR I=1TO2:PRINTTAB(0,I)CHR$141C
HR $129 CHR$157CHR$131"MUSIC COMPOSITION
: CREATE FILE "CHR$156:NEXT I
1580 Y$=GET$:PRINT#X,Y$:IF Y$="C"=FAL
SE
1610 G%=GET:PRINT#X,G%:Y1=G%-48-10*(Y
$= "+")
1690 J$=GET$:PRINT#X,J$:PRINTTAB(0,30
) SPC 26
1720 I$=GET$:PRINT#X,I$:K$=J$+I$:IF K
$= "QQ" THEN CLOSE#X:EP%=1:ENDPROC
1780 VDU4:PRINTTAB(6,30)"C to change:
any key to display";:Q=GET:PRINT #X,Q
```

When these changes are complete save the new program as MUSKRAT.

MUSKOX

To create MUSKOX reload the MUSTRAN program and add the following lines.

```
105 PROCfile
115 IF INSTR("Yy",A$) THEN PR%=1
2570 :
2580 DEFPROCfile:CLOSE#0
2590 INPUT'"name of file to be read :
```

```
" ;FI$
  2600 X=OPENIN FI$
  2610 INPUT#X,T
  2620 ENDPROC
```

Delete lines 150, 1640, 1680, 1760, and 1790, and change the following lines.

```
  110 PRINT"Do you want to print the Mu
sic? ";:REPEAT:A$=GET$:UNTIL INSTR ("YyN
n",A$): PRINTA$
  120 VDUT,58:INPUT#X,NS:VDUNS,13,10
  160 INPUT#X,M:MODE M
  220 REPEAT:PROCchoosesym:IF EP%=1 THE
N 230 ELSE UNTIL FALSE
  1010 FOR I=1TO2:PRINTTAB(0,I)CHR$141C
HR$ 129CHR$157CHR$131"MUSIC COMPOSITION :
LOAD FILE "CHR$156:NEXTI
  1580 INPUT#X,Y$:IF Y$="C" =FALSE
  1610 INPUT#X,G%:Y1=G%-48-10*(Y$="+")
  1690 INPUT#X,J$
  1720 INPUT#X,I$:K$=J$+I$:IF K$="QQ" T
HEN CLOSE#X:PROCprint:EP%=1:ENDPROC
  1780 VDU4:INPUT#X,Q
```

Save the new program as MUSKOX.

These changes turn the original program into a much more powerful tool and add music processing to the wide range of tasks the BBC B can undertake.

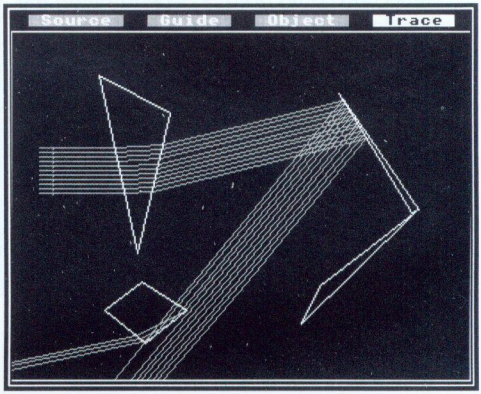
```
  10 REM Program MUSTRAN
  20 REM Version B1.1
  30 REM Author  C.A.A. Wass
  40 REM BEEBUG  Jan/Feb 1988, April 92
  50 REM Program subject to copyright
  60 :
 100 MODE 7:PROCtitle:ON ERROR GOTO 250
 110 PRINT'"Transpose music?"'"Enter 'U
' or 'D' and number of steps."
 120 T=GET:VDU T,58:NS=GET:VDU NS,13,10
 130 XST1=0:YST1=0:XST2=0:YST2=0:LS=0:L
F=0
 140 UD=NS-48+2*(NS-48)*(T=68)
 150 PRINT'"MODE -- 0 or 4 ?"
 160 M=GET-48:MODE M
 170 IF M=4 SP=8 ELSE SP=4
 180 XX=4:STNo=2:STS=288:KR$=""
 190 JU=STNo*STS+16*UD
 200 PROCstave(2)
 210 PROCvdu:PROCsym
 220 REPEAT:PROCchoosesym:UNTIL FALSE
 230 END
 240 :
 250 ON ERROR OFF:MODE 7:REPORT
 260 PRINT" at line ";ERL:END
 270 :
1000 DEF PROCtitle
1010 FOR I=1 TO 2:PRINTTAB(5,I)CHR$141C
HR$129CHR$157CHR$131"MUSIC COMPOSITION
"CHR$156:NEXT I
1020 ENDPROC
1030 :
1040 DEF PROCvdu
1050 VDU23,224,63,127,127,255,255,127,1
27,63
1060 VDU23,225,0,128,128,192,192,128,12
8,0
1070 VDU23,226,63,64,64,192,192,64,64,6
3
1080 VDU23,227,128,64,64,96,96,64,64,12
8
1090 VDU23,228,0,128,255,0,0,255,128,0
1100 VDU23,230,192,32,32,48,48,32,32,19
2
1110 VDU23,231,0,6,6,103,110,126,118,23
0
1120 VDU23,232,103,110,126,118,230,96,9
6,0
1130 VDU23,233,0,96,96,96,110,126,118,1
02
1140 VDU23,234,102,110,126,118,6,6,6,0
1150 VDU23,235,96,96,96,96,96,124,126,9
9
1160 VDU23,236,99,99,102,108,104,112,0,
0
1170 VDU23,237,0,0,0,0,255,0,0,0
1180 VDU23,238,64,32,16,8,14,28,56,56
1190 VDU23,239,24,30,48,48,48,16,14,0
1200 VDU23,240,0,130,254,30,4,4,6,8
1210 VDU23,241,24,24,24,24,24,4,0,0
```

Continued on page 54

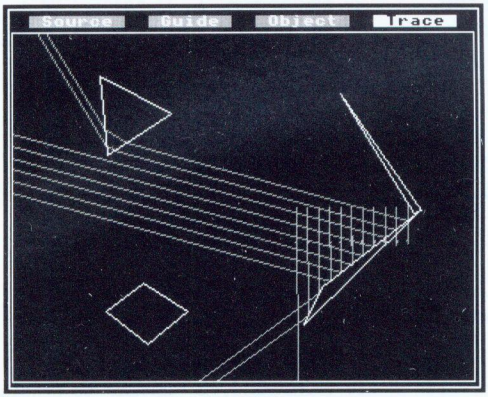
Ray Tracing in 2D (Part 2)

David Lowndes Williams trips the light fantastic.

In this issue we bring this package to its breathtaking conclusion as the paths of light rays are traced through the scenes that you created with Part 1.

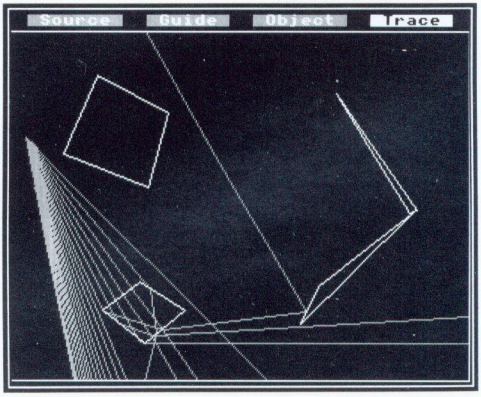


The instructions for use are contained in the previous issue but as this month's program performs the mathematics of ray tracing, I shall discuss some of the interesting techniques required. There is no need to understand any of the following, since it is all *transparent* to the user - but it is very interesting, and involves the marriage of computing and mathematics, and also the odd law of physics.



Lines are constructed as either parallel or radiating, and are stored as two points, the start and end to each vector. The program then tests for the intersection of this line with every line in the scene, using FNline_line_intersect. It picks the line of intersection closest to the vector's start, lying in front of the vector. If this intersection point lies on the screen border then the program has come to the end of the path for that ray, and it then traces the next ray. If the point of intersection lies on a mirror then PROCreflect is called and the ray emerging from the mirror is traced in an identical way through the scene.

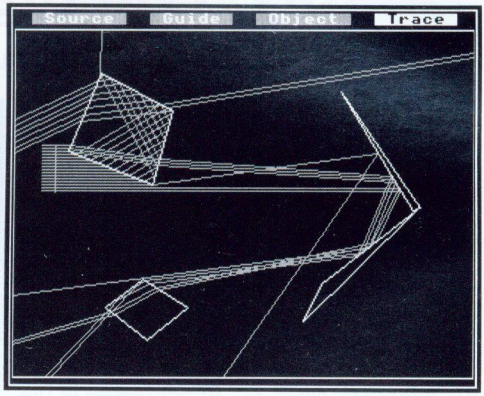
To use the program type in this month's listing and save it as *Tracer* on the same disc as the *RAYed* program from last month.



HOW IT WORKS

Each object is held as a ring of points in the arrays *sx%* and *sy%*. The sides of the object join these points. PROCclockwise ensures that all the points go in the same direction around the object. This is done by comparing each pair of adjacent lines making up the object. FNtest does this, by using the cross product.

The process is a little more complicated when an intersection occurs with a refracting surface, since refraction or reflection could occur - depending on the incidence angle and the refractive index of the material (dielectric). Using the ordered list of points, the program determines if the ray is emerging from or incident on the glass. If emergent it sets the refractive index to the reciprocal of the index for incident rays. It then calls PROCrefract. PROCrefract is rather clever - even if I do say so myself - making the ray of light obey Snell's law (if incident on the appropriate material, some crystals, including ice and calcite, have interesting properties which mean that Snell's law is not obeyed but this is not modelled by this program). PROCrefract constructs the emergent ray using the vector normal to the surface of intersection and a couple of right angled triangles.



SAVING SCREENS

The final display, when saved, is put under the filename *screen*, the name of the file may be changed, from immediate mode, using:

```
*RENAME screen MYFILE
```

MYFILE is the new name of the file. Any previously saved screen may be reloaded in the usual way, i.e. by typing:

```
MODE1
*LOAD screen
```

CONCLUSION

This brings this package to its conclusion and it demonstrates what the model B is still capable of. It will be of particular use in education since it demonstrates in a tangible way the validity of the expressions used to predict the behaviour of light. It may also be of use to people interested in gem stones, or in the design of some optical systems. It would be interesting to hear from you if you find any other uses for the program.

```
  10 REM Program TRACER
  20 REM Version B1.0
  30 REM Author  David Lowndes Williams
  40 REM BEEBUG  April 1992
  50 REM Program subject to copyright
  60 :
 100 ON ERROR GOTO 280
 110 MODE1
 120 M%=TRUE
 130 PROCinitScreen
 140 PROCload("wkfile")
 150 PROCclockwise
 160 GCOL0,3
 170 PROCobjects
 180 :
 190 PROCtrace
 200 GCOL0,3:PROCobjects
 210 PRINTTAB(1,31)"<SPACE> to continue
, <S>ave screen.";
 220 REPEAT:g%=GET
 230 UNTIL g%=32 OR g%=115 OR g%=83
 240 IF g%=115 OR g%=83 OSCLI"*SAVE scr
een 3000 8000"
 250 CHAIN"RAYed"
 260 END
 270 :
 280 IF ERR=17 AND INKEY-1 THEN 310
 290 IF ERR=17 CHAIN"RAYed"
 300 REPORT:PRINT" at line ";ERL
 310 *FX4,0
 320 END
 330 :
```

```
1000 DEF PROCinitScreen
1010 GCOL0,3:COLOUR128:CLS
1020 COLOUR128:VDU31,2,0
1030 COLOUR129:COLOUR3:PRINT;" Source "
;:COLOUR128:PRINTSPC2;:COLOUR129:PRINT;"
 Guide ";:COLOUR128:PRINTSPC2;:COLOUR129
:PRINT;" Object ";:COLOUR128:PRINTSPC2;:
COLOUR0:COLOUR131
1040 PRINT" Trace ":COLOUR128:COLOUR3
1050 ENDPROC
1060 :
1070 DEF PROCobjects
1080 FOR n%=5TO0 STEP-1
1090 IF sy%(0,n%)=0 GOTO 1150
1100 MOVE sx%(1,n%),sy%(1,n%)
1110 FOR nn%=1TO sx%(0,n%)
1120 DRAW sx%(nn%,n%),sy%(nn%,n%)
1130 NEXT nn%
1140 DRAW sx%(1,n%),sy%(1,n%)
1150 NEXT n%
1160 ENDPROC
1170 :
1180 DEF PROCdrawVector(a,b,c,d)
1190 LOCAL p,q
1200 p=.9:q=.025
1210 MOVEa,b:DRAW a+c,b+d
1220 MOVE a+c,b+d
1230 DRAW a+p*c-q*d,b+p*d+q*c
1240 MOVE a+c,b+d
1250 DRAW a+p*c+q*d,b+p*d-q*c
1260 ENDPROC
1270 :
1280 DEF FNvectorLength(a,b,s)
1290 LOCAL len
1300 len=SQR(a*a+b*b)
1310 =s/len
1320 :
1330 DEF PROCload(file$)
1340 y=OPENIN(file$)
1350 INPUT#y,Index,Number%,Parallel%
1360 INPUT#y,VSx,VSy,VVx,VVy,VPx,VPy
1370 INPUT#y,L%
1380 DIM ex%(L%):DIM ey%(L%)
1390 DIM sx%(L%,5),sy%(L%,5)
1400 FOR n%=0TO4
1410 FOR nn%=0TOL%
1420 INPUT#y,sx%(nn%,n%),sy%(nn%,n%)
1430 NEXT:NEXT
1440 CLOSE#y
1450 sx%(0,5)=4:sy%(0,5)=10
1460 sx%(1,5)=16:sy%(1,5)=976
1470 sx%(2,5)=1263:sy%(2,5)=976
1480 sx%(3,5)=1263:sy%(3,5)=48
1490 sx%(4,5)=16:sy%(4,5)=48
1500 M%=TRUE
1510 ENDPROC
1520 :
1530 DEF FNline_line_intersect(x1,y1,x2
,y2,x3,y3,x4,y4)
1540 LOCAL d1,d2,ax,ay,bx,by
1550 i%=TRUE
1560 A=-(y2-y1):B=x2-x1
1570 C=A*x1+B*y1
1580 vx=x4-x3:vy=y4-y3
1590 d1=A*vx+B*vy
1600 IF d1=0 i%=FALSE:GOTO1690
1610 t=(C-A*x3-B*y3)/d1
1620 IF t<0 OR t>1 i%=FALSE:GOTO1690
1630 X=x3+t*vx:Y=y3+t*vy
1640 d2=(x2-x1)^2+(y2-y1)^2
1650 k=((X-x1)*(x2-x1)+(Y-y1)*(y2-y1))/
d2
1660 ax=x2-x1:ay=y2-y1
1670 bx=x4-x3:by=y4-y3
1680 D%=SGN(ax*by-bx*ay)
1690 =i%
1700 :
1710 DEF PROCreflect(Rx,Ry,Sx,Sy)
1720 LOCAL Qx,Qy,k,Px,Py
1730 Px=-Rx:Py=-Ry
1740 k=(Px*Sx+Py*Sy)/(Sx*Sx+Sy*Sy)
1750 Qx=k*Sx:Qy=k*Sy
1760 Jx=2*Qx-Px:Jy=2*Qy-Py
1770 REM output (Jx,Jy)T
1780 ENDPROC
1790 :
1800 DEF PROCrefract(Rx,Ry,Nx,Ny,Indx)
1810 LOCAL k,a,sx,sy
1820 k=(Rx*Nx+Ry*Ny)/(Nx*Nx+Ny*Ny)
1830 Nx=Nx*k:Ny=Ny*k
1840 sx=(Rx-Nx)/Indx:sy=(Ry-Ny)/Indx
1850 a=((Rx*Rx+Ry*Ry)-sx*sx-sy*sy)/(Nx*
Nx+Ny*Ny)
1860 IF a<0 PROCreflect(Rx,Ry,Nx,Ny):EN
DPROC
1870 a=SQR(a)
```

```
1880 Jx=sx+a*Nx:Jy=sy+a*Ny
1890 ENDPROC
1900 Jx=ODx*k:Jy=ODy*k
1910 REM output (Jx,Jy)T
1920 ENDPROC
1930 :
1940 DEF FNVectorLength(a,b,s)
1950 LOCAL len
1960 len=SQR(a*a+b*b)
1970 IF len=0 =0
1980 =s/len
1990 :
2000 DEF PROCtrace
2010 GCOL0,1
2020 FOR s=0TO Number%
2030 Dax=VPx:Day=VPy
2040 Dbx=s*VVx/Number%+VSx
2050 Dby=s*VVy/Number%+VSy
2060 IF Parallel%=FALSE Ax=Dax:Ay=Day:B
x=Dbx:By=Dby ELSE Ax=Dbx:Ay=Dby:Bx=VVy/3
+Ax:By=-VVx/3+Ay
2070 PROCdrawVector(Ax,Ay,Bx-Ax,By-Ay)
2080 REPEAT
2090 ka=FNtest(Ax,Ay,Bx,By)
2100 GCOL0,1:MOVE Ax,Ay:DRAW XX,YY
2110 IF Dir%=TRUE Ind=Index ELSE Ind=1/
Index
2120 RRX=Bx-Ax:RRY=By-Ay
2130 j=FNVectorLength(RRX,RRY,100)
2140 RRX=RRX*j:RRY=RRY*j
2150 IF TYPE%=1 PROCrefract(RRX,RRY,DDX
,DDY,Ind)
2160 IF TYPE%=2 PROCreflect(RRX,RRY,DDX
,DDY)
2170 IF TYPE%=1 OR TYPE%=2 Ax=XX:Ay=YY:
Bx=Ax+Jx:By=Ay+Jy
2180 UNTIL TYPE%=10
2190 NEXTs
2200 ENDPROC
2210 :
2220 DEF FNtest(ax,ay,bx,by)
2230 kaa=1E32
2240 FOR n%=0TO5
2250 IF sy%(0,n%)=0GOTO 2340
2260 cx=sx%(sx%(0,n%),n%)
2270 cy=sy%(sx%(0,n%),n%)
2280 FOR nn%=1TOsx%(0,n%)
2290 dx=sx%(nn%,n%):dy=sy%(nn%,n%)
2300 type%=sy%(0,n%)
2310 a%=FNline_line_intersect(ax,ay,bx,
by,cx,cy,dx,dy)
2320 IF a%<>TRUE GOTO2340
2330 IF k>0.001 AND k<kaa THEN kaa=k:TY
PE%=type%:XX=X:YY=Y:Dir%=D%:DDX=(cy-dy):
DDY=-(cx-dx)
2340 cx=dx:cy=dy
2350 NEXT:NEXT
2360 GCOL0,1:MOVE ax,ay:DRAW XX,YY
2370 =TYPE%
2380 :
2390 DEF PROCclockwise
2400 FOR n%=0TO5
2410 IF sy%(0,n%)=0GOTO2470
2420 A=0
2430 FOR nn%=1TOsx%(0,n%)
2440 A=A+FNdirection(nn%,n%)
2450 NEXT
2460 IF A<0 PROCorder(n%)
2470 NEXT
2480 ENDPROC
2490 :
2500 DEF FNdirection(nn%,n%)
2510 LOCAL x1%,y1%,x2%,y2%,x3%,y3%,vax%
,vay%,vbx%,vby%
2520 nn1%=n%-1:IF nn1%<1 nn1%=sx%(0,n%)
2530 nn2%=n%
2540 nn3%=n%+1:IF nn3%>sx%(0,n%) nn3%=1
2550 x1%=sx%(nn1%,n%):y1%=sy%(nn1%,n%)
2560 x2%=sx%(nn2%,n%):y2%=sy%(nn2%,n%)
2570 x3%=sx%(nn3%,n%):y3%=sy%(nn3%,n%)
2580 vax%=x2%-x1%:vay%=y2%-y1%
2590 vbx%=x3%-x2%:vby%=y3%-y2%
2600 =SGN(vax%*vby%-vbx%*vay%)
2610 :
2620 DEF PROCorder(N%)
2630 t%=1
2640 FOR s%=sx%(0,N%)TO1 STEP-1
2650 ex%(t%)=sx%(s%,N%)
2660 ey%(t%)=sy%(s%,N%)
2670 t%=t%+1
2680 NEXT
2690 FOR s%=1TO sx%(0,N%)
2700 sx%(s%,N%)=ex%(s%):sy%(s%,N%)=ey%(
s%)
2710 NEXT
2720 ENDPROC
```

Paradox

Peter Rochford tries out a new game for the Beeb.

Product	**Paradox**
Developer	**The Really Good Software Company**
Supplier	**TCA 39 Carisbrooke Road, Harpenden, Herts AL5 5QS. tel: 0582 761395**
Price	**£14.95 on 5.25 disc for BBC Micro and Master £15.95 for 3.5 disc for Master Compact**

It makes a nice change to have something totally new to review gameswise for the Beeb, rather than the usual crop of re-releases from Superior's 'Play It Again Sam' series, that seem to go on forever.

This company, with the rather snappy title, is new to me, but has come up with a game called *Paradox* that claims to be not only new, but to offer something different to what we have seen before.

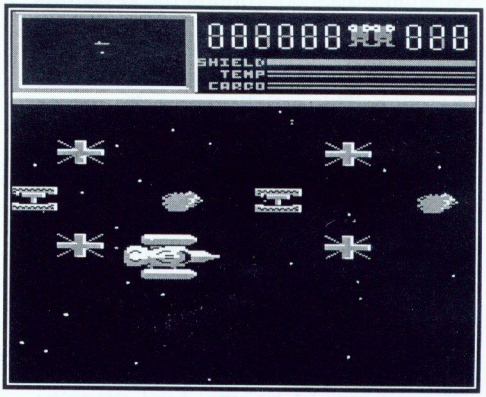
Certainly the storyline that sets the scene is rather different if not far fetched, and reads like something out of Star Trek, but I won't bore you with that. What is on offer here, is basically an arcade shoot-'em-up combined with a certain amount of strategy and puzzle solving.

Essentially, you are a time traveller lost in space and unable to get back to where you started, 500 years ago. You must guide your ship through space, and search for asteroids which you can heat up by firing your laser at them to release their energy to fuel your ship. Having fuelled your space craft, you can then power your way into a black hole and travel back ten years in time. This provides for a 50 levels of play, each of increasing difficulty and complexity.

As is to be expected, there are the usual bunch of nasties hanging around waiting to bring you to an early demise. But there are also various objects that can be collected or used to help you in your task.

The screen display in Paradox uses a four way scrolling and features a separate long range radar panel at the top, showing you the whereabouts of all objects in the playing area. Along with this are displays for time, score and energy levels. Keys are the usual 'Snapper' ones along with Return for fire. There are also keys allowing pause and control of sound.

Graphics don't really break any new ground in Paradox, and the sound I would describe as minimal and predictable. Nonetheless the game is well-presented and implemented with flicker free scrolling. Game play in the early levels is none too taxing and mercifully these can be skipped as you progress, via a password system.



Paradox in play

I wouldn't say I was absolutely bowled over by this game and tend to disagree with the publicity blurb that came with it describing it as 'revolutionary'. It does not really contain anything I haven't come across before in one form or another in terms of graphics and gameplay. Still, it is good to play and does provide a new challenge with a decent degree of longevity and of course, as I've said already, it is something new at a time when there is depressingly little being released on the leisure front for the BBC micro.

Wordwise User's Notebook

Colin Robertson shows how to create and use DATA statements in Wordwise segments.

INTRODUCTION

The DATA and READ statements in Basic are a familiar and useful way of configuring a program with items of data that can be read progressively as the program proceeds. A major practical advantage of this approach is that it is extremely easy to re-configure the program with new data simply by changing the few lines containing the DATA statements. If the data items had to be introduced at the points of use, re-configuration would be extremely difficult.

The Wordwise Plus programming language does not contain the keywords DATA and READ, yet quite often a segment program may need to be configured with data items to control its function. This article shows, by means of an example program, how batches of data may be supplied in segment programs and the data items read and processed as they are required.

THE SEGMENT

VARPRN1 is a segment program performing the simple task of extracting and printing batches of seven successive lines from a text file, in much the same way as would be required to print address labels. However, it has the additional feature that the spacing after each line is individually programmable. The original was written to print sets of labels for audio cassettes in batches comprising several different titles.

Standard cassette labels were used, each containing five peel-off strips to label one cassette with up to seven lines of information. As the label was not designed with machine printing in mind, the vertical intervals between the print positions were not uniform or exact multiples of a normal linefeed. Also, the peel-off sections themselves were sized to allow little room for error in the vertical direction. By making each line spacing independently configurable, it was possible to print successfully on batches of the labels held in a suitable sprocket driven carrier in the printer.

As listed here, the program is intended only to illustrate the programming techniques employed. It could be modified and extended to handle actual tasks, including more complex jobs, such as the entry of sets of particulars on pre-printed forms.

VARPRN1 reads the lines to be printed from a source text file named, by default, TXTFILE. This must be a pure ASCII file containing a series of lines each terminated with a carriage return. For test purposes, short lines of poetry are ideal. If you type the lines in Wordwise Plus edit mode, and supply the carriage returns from the keyboard, the saved file will be suitable as long as there are no green commands in it. Alternatively, you could edit an existing document file for the purpose. A ready-made source file is included on the monthly magazine disc, called *TXTFILE*. This file contains no blank lines, so that the printed format on blank paper can be more easily observed.

ASSIGNING DATA

The approach adopted in *VARPRN1* is to produce each inter-line space by the combination of one or more linefeeds of standard size (or any predetermined fixed size) with an appropriate variable number of additional small increments of paper movement to make up the

desired total spacing. These increments are produced by the Epson printer code ESC J *n* where *n* is the number of increments, each one-third of a printer dot in size. The distance corresponding to one third of a dot varies from one type of printer to another, typically 1/216 inch and 1/180 inch. When this code is issued, the paper is advanced once by the specified amount, with no effect on any subsequent movements.

Thus, any required spacing, as physically measured on a sample of the label, can be expressed by two integer parameters: a coarse component, *M%*, representing a number of linefeeds and a fine component, *N%*, representing a number of one-third dot increments. In the example, there are seven printable lines to each label, so seven values each of *M%* and *N%* are required to specify the spacings for a whole label, including the jump to the next label.

The required individual values of the parameters are assigned in two string variables, *M$* and *N$*, thus:

```
M$="Lfeeds,1,1,2,1,1,1,5,"
N$="Increments,15,15,3,0,4,0,20,"
```

The appearance of these strings, in which the items are separated by commas, is immediately reminiscent of Basic DATA statements, although two points of difference are worthy of note. Firstly, two separate strings have been assigned, one for each category of data, whereas in Basic items would normally be listed in a single DATA statement. The use of two strings is not essential, but is a convenient option in this case, making it easier to read the data items; also it makes the program listing easier to follow. Secondly, a comma is used not only between the items but also at the end of the string. This again is not fundamental. The spacing character could be any character not used in the data items themselves, and the final one is used only to support the particular method used to extract the individual data items.

This method of holding the data makes it easy to alter any items if required. Also, since Wordwise Plus string variables are language-resident and are not nulled automatically when a program is executed, it would be easy to assign the data externally to the program in which it is used, for example in another segment program executed previously.

Although the example strings are quite short, there is no limitation on their length other than the normal 255 character limit applying to any one string, and the limited total string space available in Wordwise Plus segment programs (437 characters). Very long data streams could be handled by reading from a file.

READING DATA

The data reading operation requires the means of extracting individual data items from the strings *M$* and *N$*. However, the dissection of strings is not a strong point in the standard Wordwise Plus programming language, which contains none of the string handling functions we find in Basic. However, remembering that Wordwise Plus is primarily a text editor, we can overcome this limitation if we first write the strings into a vacant segment and then use the powerful editing commands to locate and extract the items from the contents of the segment.

Referring to the listing, the work is done mainly by the procedure *.plusprint* from within which *.segput* selects and clears segment 8 and types *M$* and *N$* into it as separate lines of text. The required values of *M%* and *N%* are acquired by the

procedure *.params*. This in turn uses the procedure *.readdata* to access the lines in segment 8. It does this by finding the comma preceding the required item and reading the subsequent characters out into a string until the next comma is encountered. The value of *I%* controls which item is to be found; this is an index variable incremented before the next item of data is to be acquired.

Although the method implemented in *.readdata* is by no means the only one that could have been used, it does have certain general advantages. Firstly, since the required data line is found by recognising its first item (provided for this purpose as a "name"), the technique could be used equally well if there were considerably more than two data strings to be considered. Also, strings can be accessed regardless of their order in the holding segment. Secondly, the method allows data lists to wrap over the 40 column editing screen and occupy more than one screen line if necessary. Thirdly, since every access to the segment is absolute and not relative to a previous cursor position, the technique puts no limitation on the selection of other segments between data readings if the program should require this - Wordwise Plus stores the cursor position in a segment only while that segment remains the one currently selected.

It can be seen, therefore, that this is a method with wide fields of application, going far beyond the simple application mentioned above. A further advantage is that once the strings *M$* and *N$* have been deposited in segment 8, these variables can be nulled, as in .segput, to release their reservation of overall string space even before the data items have been extracted.

USING VARPRN1

When typing in the listing, you may choose to abbreviate the keywords, especially PRINT, which appears frequently and can be abbreviated to "P." Also you can ignore any leading spaces to the program lines, which are included only for clarity. You can also omit the spaces in the program lines themselves unless they are within quotes. If in doubt, copy the listing as given. Save the program as *VARPRN1*.

Before using *VARPRN1*, you must ensure that your source text file, named *TXTFILE*, is present in the current directory. You can load *VARPRN1* into any segment except segment 8 and execute with Shift-fn in the usual way - fn being the function key with the same number as the segment.

As it can be very wasteful of time and paper to test printing utilities repeatedly with real hard copy output, *VARPRN1* offers you the option at start-up to use a printer sink instead of a real printer. Otherwise, it is assumed that a parallel connected printer is in use and that this is Epson compatible.

Having checked that the program is operating normally, you can of course experiment with it, for example by changing the given values of the data items and observing the effect on the inter-line spacing of the printed output. It is unlikely that you will find *VARPRN1* useful without alteration. It is suggested that you retain it as an example and a guide to the writing of other programs that may need to use the same or similar programming techniques.

```
V$="VARPRN1"
REM Printing utility
REM with programmable line spacing.
REM BEEBUG April 1992
REM Author C W Robertson
REM No of lines per set:
L%=7
REM source filename:
```

```
K$="TXTFILE"
REM
REM Data strings:
M$="Lfeeds,1,1,2,1,1,1,5,"
N$="Increments,15,15,3,0,4,0,20,"
REM If Q%=73 it's in WW+II, else WW+
Q%=?&8017
CLS
VDU3
PROCheader
PROCprinterset
PROCprinterinit
Z%=OPENIN K$
IFZ%=0 THEN G.nofile
TIME=0
PROCplusprint
T%=TIME
GOTO finish
END

.movepaper
IFM%=0 THEN GOTO badparam
DOTHIS
 PRINT
TIMES M%
IFN%>0 THEN VDU1,27,1,74,1,N%
ENDPROC

.plusprint
PROCsegput
REPEAT
 I%=1
 REPEAT
  A$=GLF$#Z%
  PRINT A$;
  PROCparams
  PROCmovepaper
  I%=I%+1
 UNTIL I%>L% OR EOF#Z%
UNTIL EOF#Z%
DELETE TEXT
SELECT TEXT
ENDPROC
```

```
.params
T$="Lfeeds"
PROCreaddata
M%=P%
T$="Increments"
PROCreaddata
N%=P%
ENDPROC

.readdata
CURSOR TOP
FIND T$
DOTHIS
 FIND ","
 IF EOT THEN GOTO badparam
 CURSOR RIGHT
TIMES I%
P$=""
REPEAT
 C$=GCT$
 IF EOT THEN GOTO badparam
 IF ASC C$<>44 THEN P$=P$+C$
UNTIL ASC C$=44
P%=VAL P$
ENDPROC

.header
Y%=2
VDU12,31,14,Y%,134
PRINT V$
VDU 31,3,Y%+2,134
PRINT "Experimental label printer with"
VDU31,6,Y%+3,134,
PRINT "programmed line spacing"
VDU31,6,Y%+4,134
PRINT "(C) C W Robertson 1991"
A$="Wordwise Plus"
IFQ%=73 THEN A$=A$+" II"
VDU31,0,Y%+6,134
PRINT "Running in";
VDU131
PRINT A$
VDU31,0,Y%+8,134
PRINT "Programmed for";
VDU131
```

```
PRINT L%;
VDU134
PRINT "lines per label"
VDU134
PRINT "using";
VDU131
PRINT K$;
VDU134
PRINT "as source."
PRINT
VDU131
PRINT "Use Printer sink? (Y/N)";
VDU130
*FX15,1
A%=GET AND &DF
IF A%=89 THEN *FX5,0
IF A%=89 THEN PRINT "Yes"
IF A%<>89 THEN PRINT "No"
ENDPROC

.printerinit
CLS
VDU31,0,5,134
PRINT "Printing in progress"
VDU28,0,20,39,19
REM set condensed & L. margin of 10
VDU2,1,27,1,64,1,15
VDU1,27,1,108,1,10
ENDPROC

.printerset
IFA%=89 THEN GOTO sink
PRINT
VDU131
PRINT "Please set paper in position"
VDU131
PRINT "& ensure printer is on line"
.sink
PRINT
VDU131
PRINT "Ready to go? (Y/N)"
*FX15,1
A%=GET AND &DF
IF A%<>89 THEN GOTO abort
ENDPROC
```

```
.segput
SELECT SEGMENT 8
DELETE TEXT
TYPE M$+"|R"
TYPE N$+"|R"
M$=""
N$=""
ENDPROC

.abort
VDU3,26,12,31,0,5,129
PRINT "Abandoned!"
GOTO abandon

.nofile
VDU12,3,26,12,31,0,5,129
PRINT "Text file not found"
GOTO terminate

.badparam
CLOSE#Z%
VDU12,3,26,12,31,0,5,129
PRINT "Parameter missing or invalid"
GOTO terminate

.finish
CLOSE#Z%
U%=(T%MOD100)DIV10
T%=T%DIV100
VDU12,3,26,12,31,0,5,134
PRINT "Printing finished";
PRINT " in "+STR$T%+"."+STR$U%+" s"
GOTO terminate

.terminate
VDU2,1,27,1,64,3
.abandon
*FX5,1
VDU131
PRINT "Press any key"
*FX15,1
A%=GET
DISPLAY
END
```

Cross Reference Lister

by Ian Gooding

This month's utility (originally published in Vol.3 No.6) is a very useful program that provides a cross reference listing of a Basic program. It builds up lists of all the variables, procedures, functions, etc. used in the program, and the lines at which they are referenced. This program is a must for the serious developer of Basic software.

INTRODUCTION

The purpose of this program is to provide the user with a list of names and line numbers used within his Basic program to aid in debugging and documentation. The program searches for procedure and function names and the lines at which they are referenced; it deals similarly with variable names and lines which have GOTO or GOSUB statements referring to them. Support for printers is included within the program, and any or all of the search options mentioned above may be omitted. The total number of lines in the program is always displayed after the program has been run, as is an indication of the amount of remaining free memory.

USING THE CROSS REFERENCER

The cross referencer should be typed in and saved away for future use. The program to be analysed should be first saved onto disc.

The program as listed appears longer than it really is as many REM statements have been included to assist those who wish to examine the workings of the program. All REM statements may be omitted when you type the program in.

Once run, the cross referencer prompts for your choice of the available options. These include selecting output to the printer, and the checking for procedures, functions, variables, and GOTO and GOSUB references. There is an option to reference all the options without having to type in 'yes' to each of them. Having answered the questions mentioned above, the program prompts for the file name of the program to be analysed, and then proceeds to build up the cross reference list.

While the program is running, it displays on the screen the current line being scanned, the most recent procedure, function or variable name read, and the last line number referenced by a GOTO or GOSUB statement. No information is displayed for any option omitted (e.g. no reference to line numbers with GOTO or GOSUB) though the current line number is always displayed.

Once the analysis has been completed, the program lists out the information collected with coloured highlights, with the corresponding text sent to the printer if selected.

The program functions quite happily in a 6502 second processor (there is a significant speed improvement in fact). Once the run is finished, you can always obtain the list of references again by just typing PROCreport.

The necessary data is contained within the program in a fairly complex manner, and space does not permit here a detailed explanation of how this is arranged. For the more adventurous reader, an examination of the program should prove quite interesting as it is very well structured.

LIMITATIONS

A consequence of writing a utility like this in Basic, and keeping the program to a reasonable length, is that some simplifications have to be made. Any computed GOTO or GOSUB line number references are ignored (these are impossible to determine in advance anyway). DATA, REM and assembler comment blocks are ignored, and any * calls are treated as variables; e.g. *FX243,129 is treated as a reference to the variable FX243. The program will mark arrays such as A(..) as A(, but for arrays such as A%(..) or A$(..), then only A% or A$ will be displayed.

```
  10 REM Program Cross Referencer
  20 REM Author  Ian Gooding
  30 REM Version B1.00
  40 REM BEEBUG  April 1992
  50 REM Program Subject To Copyright
  60 :
 100 ON ERROR MODE7:PROCerror:END
 110 MODE 7:crlf$=CHR$13+CHR$10
 120 PROCassemble:PROCbanner:PROCOption
s:CLS:PROCbanner
 130 FOR I%=16 TO 23:PRINT TAB(0,I%);CH
R$(130);:NEXT
 140 REM If tape show tape movement, al
low error retries
 150 A%=0:Y%=0:T%=(USR&FFDA) AND 15:IF
T%=2 OR T%=3 THEN A%=139:X%=1:Y%=2:CALL&
FFF4:X%=2:Y%=2:CALL&FFF4
 160 VDU15:REM Scroll screen
 170 mline%=260:REM Space for current l
ine
 180 DIM line% mline%
 190 REM Set up chain heads
 200 DIM proc% 3:!proc%=-1
 210 DIM fn% 3:!fn%=-1
 220 DIM var% 3:!var%=-1
 230 DIM goto% 3:!goto%=-1
 240 DIM gosub% 3:!gosub%=-1
 250 REM Set up flags
 260 eof%=FALSE:ass%=FALSE:gos%=FALSE
 270 count%=0
 280 REM Open the program file
 290 f%=FNopenfile
 300 REM Loop reading each line
 310 REPEAT
 320 PROCreadline
 330 IF eof% THEN 360
 340 count%=count%+1
 350 PROCscanline:REM Store references
 360 UNTIL eof%
 370 CLOSE #0:MODE 7:PROCreport
 380 END
 390 :
1000 DEF PROCreport
1010 IF NOT print% THEN VDU 14:prlen%=0
ELSE PROCsetprinter
1020 PROCpdouble("Program "+prog$,CHR$(
134))
1030 PROCp("","")
1040 PROCp("Total program lines = "+STR
$(count%),CHR$130)
1050 PROCp("","):REM blank line
1060 DIM P% -1:PROCp("Memory used = "+S
TR$(INT(100-(100*(HIMEM-P%))/(HIMEM-TOP)
))+" %",CHR$(130))
1070 PROCp("","):REM blank line
1080 IF lproc% THEN PROCpvar(!proc%,"PR
OCEDURES","PROC")
1090 IF lfn% THEN PROCpvar(!fn%,"FUNCTI
ONS","FN")
1100 IF lvar% THEN PROCpvar(!var%,"VARI
ABLES","")
1110 IF lgoto% THEN PROCpgo(!goto%,"GOT
O LINES","GOTO")
1120 IF lgosub% THEN PROCpgo(!gosub%,"G
OSUB LINES","GOSUB")
1130 PRINT:VDU 15
1140 ENDPROC
1150 :
1160 DEF FNopenfile
1170 LOCAL f%:VDU 28,6,23,35,16
1180 REPEAT
1190 INPUT "Program name :"prog$
1200 f%=OPENUP(prog$)
1210 UNTIL f%<>0
1220 VDU 28,0,24,39,0:=f%
1230 :
1240 DEF PROCreadline
1250 LOCAL i%
1260 VDU 28,6,23,35,16:REM set screen s
ubset up
1270 eline%=line%-1:i%=BGET#f%
1280 IF i%<>&0D THEN PRINT "Bad file":E
```

```
ND
 1290 nline%=BGET#f%*256+BGET#f%:REM lin
e number in 2 bytes
 1300 IF (nline% AND &8000) <> 0 THEN CL
OSE#f%:eof%=TRUE:ENDPROC
 1310 i%=BGET#f%:REM character count
 1320 REPEAT
 1330 eline%=eline%+1
 1340 IF (eline%-line%)>mline% THEN PRIN
T "Line buffer overflow":STOP
 1350 ?eline%=BGET#f%
 1360 UNTIL (eline%-line%)=(i%-5)
 1370 ?(eline%+1)=&0D:REM terminator
 1380 VDU 28,0,24,39,0
 1390 ENDPROC
 1400 :
 1410 DEF PROCscanline
 1420 LOCAL i%,j%,k%,q%,quote%,proc$:pro
c$=""
 1430 def%=FALSE:fnc%=FALSE:quote%=FALSE
:gos%=FALSE
 1440 A%=POS:B%=VPOS:PRINTTAB(8,5);"Line
 number : ";RIGHT$("    "+STR$(nline%),6
);TAB(A%,B%);
 1450 FOR i%=line% TO eline%
 1460 REM &F4 => Keyword "REM", so skip
out
 1470 REM &DC = Keyword "DATA", ignore t
his too
 1480 IF ?i%=&F4 OR ?i%=&DC THEN i%=elin
e%:GOTO 1890
 1490 REM look for start of assembler
 1500 IF ?i%=ASC("[") AND NOT quote% THE
N ass%=TRUE:GOTO1890
 1510 IF ?i%=ASC("]") AND NOT quote% THE
N ass%=FALSE:GOTO1890
 1520 IF ass% THEN GOTO 1890
 1530 REM &DD => Keyword "DEF"
 1540 IF ?i%=&DD THEN def%=2
 1550 REM ":" is statement separator
 1560 IF ?i%=ASC("""") THEN quote%=NOT q
uote%:GOTO1890
 1570 IF ?i%=ASC(":") OR i%=ASC("(") THE
N def%=0:fnc%=0:proc$="":GOTO1890
 1580 REM Look for GOTO etc line numbers
 1590 REM &E4 = GOSUB    &E5 = GOTO
 1600 REM &F7 = RESTORE  &8C = THEN
 1610 IF ?i%=&E4 THEN gos%=TRUE ELSE IF
 ?i%=&F7 OR ?i%=&E5 OR ?i%=&8C THEN gos%=
FALSE
 1620 REM &8D = line number marker
 1630 IF ?i%=&8D THEN i%=FNscangoto(i%):
GOTO 1890
 1640 IF ?i%=&B8 AND ?(i%+1)=&50 THEN i%
=i%+1:GOTO 1890:REM bodge for TOP = TO+P
 1650 REM &F2 => Keyword "PROC"
 1660 REM &A4 => Keyword "FN"
 1670 IF (?i%<>&F2) AND (?i%<>&A4) THEN
GOTO 1780
 1680 fnc%=(?i%=&A4)
 1690 j%=i%+1
 1700 REPEAT
 1710 proc$=proc$+CHR$(?j%):j%=j%+1
 1720 UNTIL j%>eline% OR NOT (FNletter(?
j%) OR ?j%=ASC("£") OR ?j%=ASC("_") OR F
Ndigit(?j%))
 1730 i%=j%-1
 1740 IF lfn% AND fnc% THEN PROCline(FNd
ef(fn%,proc$)) ELSE IF (NOT fnc%) AND lp
roc% THEN PROCline(FNdef(proc%,proc$))
 1750 proc$="":fnc%=FALSE:def%=FALSE
 1760 GOTO 1890
 1770 REM hexadecimal number ?
 1780 IF ?i%<>ASC("&") THEN GOTO 1810
 1790 REPEAT:i%=i%+1:UNTIL i%>eline% OR
NOT FNhex(?i%)
 1800 i%=i%-1:GOTO1890
 1810 REM variable reference ?
 1820 IF quote% OR NOT (FNletter(?i%) AN
D ?i%<>ASC("£") AND ?i%<>ASC("_")) THEN
GOTO 1890
 1830 var$=""
 1840 REPEAT
 1850 var$=var$+CHR$(?i%):i%=i%+1
 1860 UNTIL i%>eline% OR NOT (FNletter(?
i%) OR FNdigit(?i%) OR ?i%=ASC("£") OR ?
i%=ASC("_"))
 1870 IF ?i%=ASC("(") OR ?i%=ASC("$") OR
 ?i%=ASC("%") THEN var$=var$+CHR$(?i%) E
LSE i%=i%-1
 1880 IF lvar% THEN PROCline(FNdef(var%,
var$))
 1890 NEXT i%
 1900 ENDPROC
 1910 :
 1920 DEF FNscangoto(i%)
 1930 LOCAL num%:num%=FNgnumber(i%):REM
extract line number
```

```
 1940 i%=i%+3:REM skip over it
 1950 IF lgosub% AND gos% THEN PROCline(
FNgo(gosub%,num%)) ELSE IF (NOT gos%) AN
D lgoto% THEN PROCline(FNgo(goto%,num%))
 1960 =i%
 1970 :
 1980 DEF FNgnumber(x%)
 1990 REM Turn 3 bytes from x% into line
 number
 2000 REM from internal GOTO format
 2010 ?&70=?(x%+1):?&71=?(x%+2):?&72=?(x
%+3)
 2020 CALL denumb:=256*?&73+?&74
 2030 :
 2040 DEF PROCassemble
 2050 DIM denumb 30
 2060 P%=denumb:[OPT 2
 2070 \ Decode GOTO line ref in &70,&71,
&72
 2080 \ to binary line number in &73,&74
 2090 \ Temporary storage in &75
 2100 LDA &70:ASL A:ASL A:STA &75
 2110 AND #&C0:EOR &71:STA &74
 2120 LDA &75:ASL A:ASL A:EOR &72:STA &7
3:RTS
 2130 ]
 2140 ENDPROC
 2150 :
 2160 DEF FNletter(x%)=((x%>=ASC("A") AN
D x%<=ASC("Z")) OR (x%>=ASC("a") AND x%<
=ASC("z")))
 2170 :
 2180 DEF FNdigit(x%)=(x%>=ASC("0") AND
x%<=ASC("9"))
 2190 :
 2200 DEF FNhex(x%)=((x%>=ASC("0") AND x
%<=ASC("9")) OR (x%>=ASC("A") AND x%<=AS
C("F")))
 2210 :
 2220 DEF FNdef(chain%,name$)
 2230 A%=POS:B%=VPOS:PRINTTAB(8,7);"Symb
ol :";name$;SPC(40-POS);TAB(A%,B%);
 2240 LOCAL end%,found%,new%,last%
 2250 end%=FALSE:found%=FALSE:last%=chai
n%:chain%=!chain%
 2260 REPEAT
 2270 IF chain%=-1 THEN end%=TRUE:GOTO 2
300
 2280 IF name$=$(chain%+11) THEN end%=TR
UE:found%=TRUE:GOTO 2300
 2290 IF name$>$(chain%+11) THEN last%=c
hain%:chain%=!chain% ELSE end%=TRUE
 2300 UNTIL end%
 2310 IF found% THEN GOTO 2380
 2320 DIM new% LEN(name$)+11
 2330 !last%=new%:!new%=chain%:chain%=ne
w%
 2340 $(chain%+11)=name$
 2350 ?(chain%+9)=0:?(chain%+10)=0
 2360 DIM new% 3:!new%=-1:!(chain%+4)=ne
w%
 2370 ?(chain%+8)=0
 2380 IF def% THEN ?(chain%+9)=nline%/25
6
 2390 IF def% THEN ?(chain%+10)=nline%
 2400 =chain%
 2410 :
 2420 DEF FNgo(chain%,line%)
 2430 LOCAL end%,found%,new%,last%
 2440 end%=FALSE:found%=FALSE:last%=chai
n%:chain%=!chain%
 2450 A%=POS:B%=VPOS:PRINTTAB(8,9);"Ref
Line    : ";RIGHT$("    "+STR$(line%),6)
;TAB(A%,B%);
 2460 REPEAT
 2470 IF chain%=-1 THEN end%=TRUE:GOTO 2
500
 2480 IF FNnumber(chain%+9)=line% THEN e
nd%=TRUE:found%=TRUE:GOTO 2500
 2490 IF FNnumber(chain%+9)<line% THEN l
ast%=chain%:chain%=!chain% ELSE end%=TRU
E
 2500 UNTIL end%
 2510 IF found% THEN =chain%
 2520 DIM new% 10:!last%=new%:!new%=chai
n%:chain%=new%
 2530 DIM new% 3:!new%=-1:!(chain%+4)=ne
w%
 2540 ?(chain%+8)=0:?(chain%+9)=line%/25
6
 2550 ?(chain%+10)=line%:=chain%
 2560 :
 2570 DEF PROCline(ref%)
 2580 LOCAL i%,p%,end%,new%,qu%,last%
 2590 end%=FALSE
 2600 last%=!(ref%+4):p%=!last%
 2610 REPEAT
 2620 IF p%=-1 THEN DIM new% 43:!last%=n
```

```
ew%:p%=new%:FOR i%=0 TO 40 STEP 4:!(i%+n
ew%)=-1:NEXT i%
 2630 i%=2
 2640 REPEAT
 2650 i%=i%+2:qu%=FALSE
 2660 IF i%=42 THEN qu%=TRUE ELSEIF FNnu
mber(p%+i%)=&FFFF OR FNnumber(p%+i%)=nli
ne% THEN qu%=TRUE
 2670 UNTIL qu%
 2680 IF i%<42 THEN end%=TRUE ELSE last%
=p%:p%=!p%
 2690 UNTIL end%
 2700 ?(p%+i%)=nline%/256
 2710 ?(p%+i%+1)=nline%
 2720 ENDPROC
 2730 :
 2740 DEF FNnumber(x%)=(?x%)*256+?(x%+1)
 2750 :
 2760 DEF PROCpvar(chain%,title$,type$)
 2770 LOCAL a$
 2780 PROCpdouble(title$,CHR$(131))
 2790 PROCp("","")
 2800 IF chain%=-1 THEN PROCp("    (NONE
)","):PROCp("","):ENDPROC
 2810 REPEAT
 2820 a$=type$+" "+$(chain%+11)
 2830 IF FNnumber(chain%+9)<>0 THEN a$=a
$+" ("+STR$(FNnumber(chain%+9))+")"
 2840 PROCp(a$,CHR$(134))
 2850 PROCplines(chain%)
 2860 chain%=!chain%
 2870 UNTIL chain%=-1
 2880 ENDPROC
 2890 :
 2900 DEF PROCpgo(chain%,title$,ti$)
 2910 PROCpdouble(title$,CHR$(130)):PROC
p("","")
 2920 IF chain%=-1 THEN PROCp("    (NONE
)","):PROCp("","):ENDPROC
 2930 REPEAT
 2940 PROCp(" "+ti$+" "+STR$(FNnumber(ch
ain%+9))+" From :",CHR$(134))
 2950 PROCplines(chain%)
 2960 chain%=!chain%
 2970 UNTIL chain%=-1
 2980 ENDPROC
 2990 :
 3000 DEF PROCplines(ref%)
 3010 LOCAL p%,i%,end%,a$:p%=!!(ref%+4):
a$=STRING$(6," ")
 3020 REPEAT
 3030 IF p%=-1 THEN GOTO 3110
 3040 i%=2:end%=FALSE
 3050 REPEAT
 3060 i%=i%+2
 3070 IF i%<42 AND FNnumber(i%+p%)<>&FFF
F THEN a$=a$+RIGHT$("    "+STR$(FNnumbe
r(i%+p%)),6) ELSE end%=TRUE
 3080 IF (print% AND LEN(a$)>prlen%-7) O
R (NOT print% AND LEN(a$)>33) THEN PROCp
(a$,""):a$="    "
 3090 UNTIL end%
 3100 p%=!p%
 3110 UNTIL p%=-1
 3120 PROCp(a$,""):PROCp("","")
 3130 ENDPROC
 3140 :
 3150 DEF PROCp(text$,control$)
 3160 REM print control$+text$ on screen
 3170 REM print text$ only on printer
 3180 REM if enabled
 3190 PRINT control$;
 3200 IF print% THEN VDU 2
 3210 PRINT text$:VDU 3
 3220 ENDPROC
 3230 :
 3240 DEF PROCpdouble(text$,control$)
 3250 REM print control$ AND text$ in
 3260 REM double height on screen,
 3270 REM print text$ on printer (once)
 3280 REM if enabled
 3290 PRINT CHR$(141);control$;text$
 3300 PRINT CHR$(141);control$;
 3310 IF print% THEN VDU 2
 3320 PRINT text$:VDU 3
 3330 PRINT SPC1;control$;
 3340 IF print% VDU2
 3350 PRINT STRING$(LEN text$,"=")
 3360 VDU3
 3370 ENDPROC
 3380 :
 3390 DEF PROCsetprinter
 3400 REM user to do whatever needed
 3410 prlen%=80:REM line length to use
 3420 REM send line feeds
 3430 REM *FX6,0 if necessary
 3440 ENDPROC
 3450 :
```

```
 3460 DEF PROCoptions
 3470 REM User inputs options
 3480 REM Is print output required ?
 3490 print%=FNyesno("Do you want to pri
nt the results"+crlf$+"as well as displa
y them")
 3500 IF NOT FNyesno("Do you want to lim
it the types of detailwhich are recorded
") THEN lvar%=TRUE:lproc%=TRUE:lfn%=TRUE
:lgoto%=TRUE:lgosub%=TRUE:ENDPROC
 3510 lproc%=FNyesno("Do you want PROCs"
)
 3520 lfn%=FNyesno("Do you want FNs")
 3530 lvar%=FNyesno("Do you want variabl
es")
 3540 lgoto%=FNyesno("Do you want GOTOs"
)
 3550 lgosub%=FNyesno("Do you want GOSUB
s")
 3560 ENDPROC
 3570 :
 3580 DEF PROCbanner
 3590 FOR I%=1 TO 2
 3600 PRINT CHR$(141);CHR$(134);SPC(3);C
HR$(157);CHR$(132);CHR$(139);"Cross Refe
rence Lister   ";CHR$(156)
 3610 NEXT:PRINT
 3620 ENDPROC
 3630 :
 3640 DEF FNyesno(prompt$)
 3650 LOCAL end%,ans$:end%=FALSE
 3660 PRINT prompt$;" (Y or N) ";
 3670 REPEAT
 3680 INPUT ans$
 3690 ans$=LEFT$(ans$,1)
 3700 IF ans$="Y" OR ans$="y" OR ans$="N
" OR ans$="n" THEN end%=TRUE ELSE VDU 7:
PRINT "Answer Y or N please ";
 3710 UNTIL end%
 3720 =(ans$="Y" OR ans$="y")
 3730 :
 3740 DEF PROCerror
 3750 ON ERROR OFF
 3760 IF ERR<>17 REPORT:PRINT" at line "
;ERL
 3770 CLOSE#0:END
```

Pseudo-Variables (3)

Alan Wrigley concludes his explanation of pseudo-variables by looking at those which relate to time.

In the last two issues I looked in some detail at Basic's file handling and memory management pseudo-variables. The final group is concerned with the management of time, and consists of just two variables: TIME and TIME$. In fact, unless you have a Master 128, then you are restricted to just one, since TIME$ requires the presence of the Master's battery-backed real-time clock. Basic on a model B is unaware of this variable, so any attempt to access TIME$ will produce an error (unless of course your program has declared a variable of that name itself), while on a Master Compact it will simply return a nonsensical value.

TIME

The pseudo-variable TIME provides a means of timing events and actions. Built into the computer is a timing device known as the system clock (which is quite separate from the real-time clock on the Master 128 which keeps a record of the actual date and time of day). The system clock is simply a timer which is reset to zero each time the computer is switched on or Ctrl-Break is pressed, and increments once every centisecond thereafter. The current value can be read at any time, and equally importantly, a value can be written to the system clock, whereupon the timer continues to increment from that value.

The system clock can be read and written by using operating system calls. Of more importance for the purposes of this article, the current value of the system clock can be read or written by using the pseudo-variable TIME. The variable is very simple to use; for example:

```
TIME=0
```

would reset the timer to zero, while:

```
PRINT TIME
```

would display the number of centiseconds since the timer was last reset to zero. Similarly:

```
elapsed%=TIME
```

would place that value into a variable *elapsed%*.

There are many uses for TIME. For example, it is often used in games where a finite amount of time might be allocated for a particular action. So the program would set the clock to zero at the start of the action, and then continually read the value until it was greater than the maximum time allowed. Another common use is in control applications, where you might be reading data from the analogue port at specific time intervals, say. Suppose you wanted to read the state of an input line every second. The system clock will increment 100 times for every second, so you need to know whenever the value of the clock crosses a 100-centisecond boundary. You can do this as follows:

```
TIME=0:oldtime%=0
REPEAT:REPEAT
newtime%=TIME DIV 100
UNTIL newtime%>oldtime%
oldtime%=newtime%
PROCreaddata
UNTIL FALSE
```

The timer is reset at the start, and also a variable *oldtime%* is set to zero (we can use integer variables here since the value returned by TIME will always be an integer). This variable will hold the number of complete seconds which have elapsed. The inner REPEAT-UNTIL loop keeps reading the timer until the value divided by 100 is greater than *oldtime%*.

At this point, another second will have passed and so *oldtime%* is incremented and the data is read.

I should perhaps mention that there is an alternative, and possibly better, method of achieving the same objective by using events, but that is really only of relevance to machine code programmers and is outside the scope of this article. For simple applications, using TIME in the way described here is a perfectly valid method.

If you are developing your own programs, it can often be very useful to find out exactly how long a particular section of code takes to execute, especially if the actions being performed are time-critical. Using TIME during program development can be a very quick and simple way of doing this. For example, suppose that you want to try out various ways of performing a certain action to see which is the fastest. This could be achieved by inserting a statement to write to the clock at the start of the process, and another to read the value at the end. By altering the code in between, you can compare the timer values to see which alternative executed more quickly. This can easily be done as follows:

```
TIME=0
.....
REM action takes place here
.....
PRINT TIME
```

Sometimes, however, the actions you are timing will take considerably less time than 1 centisecond to execute, and in these circumstances the system clock will obviously be useless since it cannot register a value of less than 1 centisecond. A useful trick if this is the case is to perform the actions a number of times within a loop, and time the entire loop, for example:

```
TIME=0
FOR I%=1 TO 1000
.....
REM action takes place here
.....
NEXT
PRINT TIME
```

The value returned will, of course, be 1000 times the value obtained from one individual occurrence of the action.

TIME$

As I mentioned earlier, TIME$ is used to read and write the value of the real-time clock, and hence is only of relevance to Master 128 owners. The real-time clock facility is provided by a special calendar/clock chip, which is powered by the battery in the computer when mains power is switched off. Operating system calls are available to write a value to the clock and to read it back in a number of formats. From Basic, however, the simplest way is to use TIME$. When reading, TIME$ returns a string of a fixed format; when writing, it takes a string of the same format as its argument. The format is as follows:

```
Day,dd Mon yyyy.hh:mm:ss
```

where *Day* and *Mon* are the first 3 letters of the current day and month respectively, and the other elements are numerical values representing the date, year, hours, minutes and seconds. So a typical string returned by TIME$ would be:

```
Wed,19 Feb 1992.13:32:45
```

In conjunction with Basic's string-slicing facilities, this can often be used to great effect in your own programs, as the following examples illustrate:

```
100 Day$=LEFT$(TIME$,3)
.....
200 PRINT "Today's date is ";MID$(TIME$
,5,11)
.....
300 current$=TIME$
310 PRINT "At the third stroke, it will
be ";MID$(current$,17,5);" and ";RIG
HT$(current$,2);" seconds"
```

By assigning a value to TIME$ you can set the real-time clock. You can specify either the date element or the time element or both, but in all cases the format must match exactly that described above, and punctuation is critical, as shown in the following examples:

```
TIME$="Fri,29 Nov 1991"
TIME$="17:23:45"
TIME$="Fri,29 Nov 1991.17:23:45"
```

It can be useful to do this if the real-time clock has got out of step with the actual time, as it tends to do over a period, or indeed if the battery has to be replaced in which case the clock will stop when the machine is switched off. You might also perhaps want to set a particular date or time to test a program you are writing - perhaps to generate an effect on 1st April or at midnight. Note that because the expression is a fixed-format string, it must be *exactly* the right length, with spaces inserted if necessary. For example, the following statement would be ignored:

```
TIME$="Fri,8 Nov 1991"
```

because the date only contains a single character ("8") instead of the two required, whereas either of the following is acceptable:

```
TIME$="Fri, 8 Nov 1991"
TIME$="Fri,08 Nov 1991"
```

Note that the operating system makes no attempt to check that the day of the week you supply matches the date in question (unlike the Archimedes, where the correct day will automatically be substituted).

TOKENS

Before we leave pseudo-variables, it is worth knowing that they possess one interesting additional property. As you will probably know, a Basic program in memory is tokenised; that is, each Basic keyword, such as PRINT for example, is not stored in memory as a string of ASCII codes, but as a single byte with a value greater than 128 (these are listed in the User Guide for reference). Each keyword is represented by a different byte, and this is known as its "token". When the program is run, the token is read and the appropriate routine in the Basic ROM is called. However, apart from EXT# each pseudo-variable keyword uses one of two tokens, one for when a value is assigned to the variable, and another for when it is read. This is because a different routine must be called in each case, and it is generally preferable for Basic to decide which one at the time of tokenising rather than when the program is run.

This property of pseudo-variables has important consequences for programmers. For example, on a Master the command LIST IF is very useful for finding all occurrences of a particular string or keyword. However, LIST IF will match keywords only to the first token it finds in the token list, so it will only find occurrences of a pseudo-variable where it is being read, and not where it is assigned. Also, while the keyword THEN is normally optional after an IF statement, it is imperative to include it if the first statement after the condition assigns a value to a pseudo-variable. In other words, a line such as:

```
IF A%=5 PTR#file%=123
```

will generate a syntax error, while:

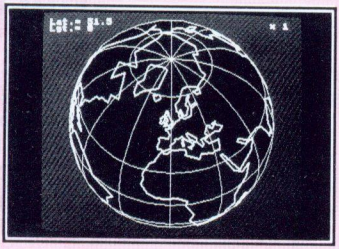
```
IF A%=5 THEN PTR#file%=123
```

is perfectly acceptable. It is useful to know this, since otherwise there is no obvious reason for the error.

I hope this in-depth look at pseudo-variables has been worthwhile. Next month we will move on to the subject of memory usage, and in particular why and how you might manipulate memory directly from within your programs.

Best of BEEBUG

Applications I Disc

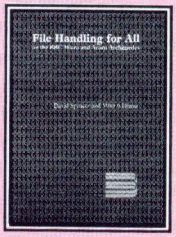


Business Graphics - for producing graphs, charts and diagrams
Video Catalogue - catalogue and print labels for your video cassettes
Phone Book - an on-screen telephone book which can be easily edited and updat
Personalised Letter-Headings - design a stylish logo for your letter heads
Appointments Diary - a computerised appointments diary
Mapping the British Isles - draw a map of the British Isles at any size
Selective Breeding - a superb graphical display of selective breeding of insects
Personalised Address Book - on-screen address and phone book
The Earth from Space - draw a picture of the Earth as seen from any point in space
Page Designer - a page-making package for Epson compatible printers
World by Night and Day - a display of the world showing night and day for any time and date of the year

File Handling for All

on the BBC Micro and Acorn Archimedes

by David Spencer and Mike Williams



Computers are often used for file handling applications yet this is a subject which computer users find difficult when it comes to developing their own programs. *File Handling for All* aims to change that by providing an extensive and comprehensive introduction to the writing of file handling programs with particular reference to Basic.

File Handling for All, written by highly experienced authors and programmers David Spencer and Mike Williams, offers 144 pages of text supported by many useful program listings. It is aimed at Basic programmers, beginners and advanced users, and anybody interested in File Handling and Databases on the Beeb and the Arc. However, all the file handling concepts discussed are relevant to most computer systems, making this a suitable introduction to file handling for all.

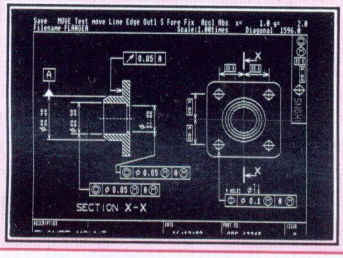
The book starts with an introduction to the basic principles of file handling, and in the following chapters develops an in-depth look at the handling of different types of files e.g. serial files, indexed files, direct access files, and searching and sorting. A separate chapter is devoted to hierarchical and relational database design, and the book concludes with a chapter of practical advice on how best to develop file handling programs.

The topics covered by the book include:

Card Index Files, Serial Files, File Headers, Disc and Record Buffering, Using Pointers, Indexing Files, Searching Techniques, Hashing Functions, Sorting Methods, Testing and Debugging, Networking Conflicts, File System Calls

The associated disc contains complete working programs based on the routines described in the book and a copy of Filer, a full-feature Database program originally published in BEEBUG magazine.

ASTAAD



Enhanced ASTAAD CAD program for the Master, offering the following features:

- full mouse and joystick control
- built-in printer dump
- speed improvement
- STEAMS image manipulator
- Keystrips for ASTAAD and STEAMS
- Comprehensive user guide
- Sample picture files

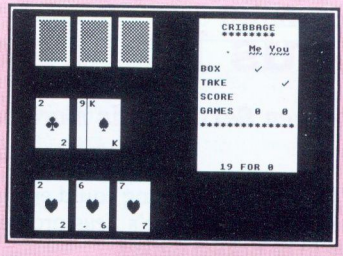
	Stock Code	Price		Stock Code	Price
ASTAAD (80 track DFS)	1407a	£ 5.95	**ASTAAD** (3.5" ADFS)	1408a	£ 5.95
EDIKIT (EPROM)	1451a	£ 7.75			
EDIKIT (40/80T DFS)	1450a	£ 5.75	**EDIKIT** (3.5" ADFS)	1452a	£ 5.75
Applications II (80 track DFS)	1411a	£ 4.00	**Applications II** (3.5" ADFS)	1412a	£ 4.00
Applications I Disc (40/80T DFS)	1404a	£ 4.00	**Applications I Disc** (3.5" ADFS)	1409a	£ 4.00
General Utilities Disc (40/80T DFS)	1405a	£ 4.00	**General Utilities Disc** (3.5" ADFS)	1413a	£ 4.00

Please add p&p

RISC Developments Ltd, 117 Hatfield Road, St Albans, Herts AL1 4JS.

Best of BEEBUG

Board Games



Solitaire - an elegant implementation of this ancient and fascinating one-player game, and a complete solution for those who are unable to find it for themselves.

Roll of Honour - Score as many points as possible by throwing the five dice in this on-screen version of 'Yahtze'.

Patience - a very addictive version of one of the oldest and most popular games of Patience.

Elevenses - another popular version of Patience - lay down cards on the table in three by three grid and start turning them over until they add up to eleven.

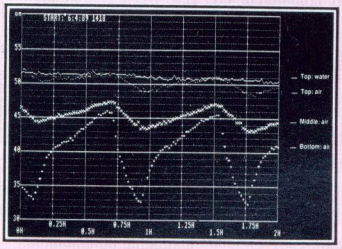
Cribbage - an authentic implementation of this very traditional card game for two, where the object is to score points for various combinations and sequences of cards.

Twiddle - a close relative of Sam Lloyd's sliding block puzzle and Rubik's cube, where you have to move numbers round a grid to match a pattern.

Chinese Chequers - a traditional board game for two players, where the object is to move your counters, following a pattern, and occupy the opponent's field.

Aces High - another addictive game of Patience, where the object is to remove the cards from the table and finish with the aces at the head of each column.

Applications II Disc



Crossword Editor - for designing, editing and solving crosswords

Monthly Desk Diary - a month-to-view calendar which can also be printed

3D Landscapes - generates three dimensional landscapes

Real Time Clock - a real time digital alarm clock displayed on the screen

Running Four Temperatures - calibrates and plots up to four temperatures

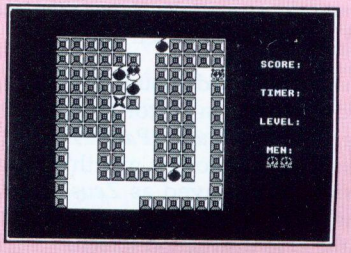
Julia Sets - fascinating extensions of the Mandelbrot set

Foreign Language Tester - foreign character definer and language tester

Label Processor - for designing and printing labels on Epson compatible printers

Share Investor - assists decision making when buying and selling shares.

Arcade Games



George and the Dragon - Rescue 'Hideous Hilda' from the flames of the dragon, but beware the flying arrows and the moving holes on the floor.

Ebony Castle - You, the leader of a secret band, have been captured and thrown in the dungeons of the infamous Ebony Castle. Can you escape back to the countryside, fighting off the deadly spiders on the way and collecting the keys necessary to unlock the coloured doors?

Knight Quest - You are a Knight on a quest to find the lost crown, hidden deep in the ruins of a weird castle inhabited by dangerous monsters and protected by a greedy guardian.

Pitfall Pete - Collect all the diamonds on the screen, but try not to trap yourself when you dislodge the many boulders on your way.

Builder Bob - Bob is trapped on the bottom of a building that's being demolished. Can you help him build his way out?

Minefield - Find your way through this grid and try to defuse the mines before they explode, but beware the monsters which increasingly hinder your progress.

Manic Mechanic - Try to collect all the spanners and reach the broken-down generator, before the factory freezes up.

Quad - You will have hours of entertainment trying to get all these different shapes to fit.

	Stock Code	Price
Arcade Games (40/80 track DFS)	PAG1a	£5.95
Board Games (40/80 track DFS)	PBG1a	£5.95
File Handling for All Book	BK02b	£9.95
File Handling for All Disc (40/80T DFS)	BK05a	£4.75
Joint Offer book and disc (40/80T DFS)	BK04b	**£11.95**

	Stock Code	Price
Arcade Games (3.5" ADFS)	PAG2a	£5.95
Board Games (3.5" ADFS)	PBG2a	£5.95
File Handling for All Disc (3.5" ADFS)	BK07a	£4.75
Joint Offer book and disc (3.5" ADFS)	BK06b	**£11.95**

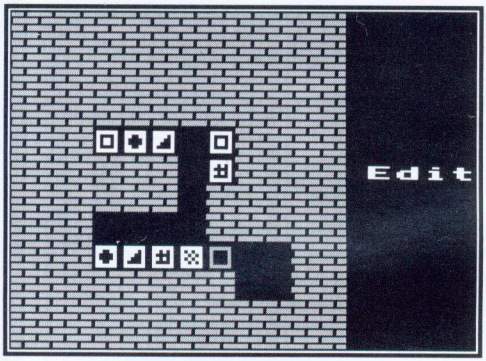
Please add p&p. UK: £1.00 first item (50p for every additional item), Europe and Eire: £1.60 first item (80p every additional item), Elswhere: £2.60 first item (£1.30 every additional item)

Tel. (0727) 40303 Fax. (0727) 860263

Zeus II - The Final Conflict

Stephen Sexton adds more mind-mangling levels to his game and shows you how to make your own.

If you thought that nothing could ever be as much fun as last issue's *Zeus* program prepare to be tickled pinker.



The first gem we bring you is a new set of levels. Type in the listing *BDATA*, save it then run it. This will produce a new set of levels for *Zeus*. If you want them to load automatically save the levels as *DEFAULT* - if not, save them as something else.

When you have had enough of that you can add the screen editor to the main program. The *Zeus2 (Part 2)* listing should be added to last month's *Zeus2* program and resaved as *Zeus2*. Now you will be able to access the editor from the *Zeus* instruction screen.

THE ZEUS EDITOR EXPLAINED

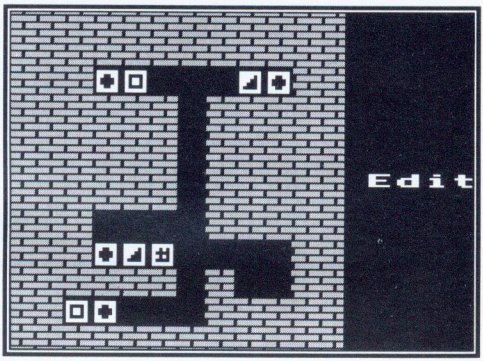
On entering the editor you are presented with the Edit screen. The cursor keys move the block cursor around, whilst keys 1 to 7 put a block of type 1-7 at the cursor. Blocks of type 1 score least on removal, whilst type 7 score most. Put the higher scoring blocks on the later levels.

Return puts a unit of wall at the cursor, whilst Space puts in a blank - for tunnels, shafts and chambers.

When the game is actually played, blocks only disappear when they are moved together, so check that you don't start with two similar blocks alongside each other.

All editing occurs on the scratch playfield. This is not accessed by the main game during play - so you can experiment to your heart's content without messing up the other levels.

Screens can by yanked from or shoved to the scratch playfield by using Y and S. Yanking a level copies the selected level to the scratch playfield, whilst shoving does the opposite. Both options require numeric input for the level to be yanked or shoved - there are 8 levels to bbcZEUS, so press 1 to 8.



Pressing P allows a playtest of the level currently in the scratch playfield. If the level is impossible (i.e. only one block of a type) the editor says so and continues editing. This also occurs when shoving a level. If the level is deemed clearable, it is played. The normal 2 lives are given, the score shown, and when clearance occurs,

or both lives are lost, the editor is re-entered with the scratch playfield still intact.

While the program can test for single blocks it is still possible to design an unsolvable screen so make sure you can finish it before adding it to the main levels.

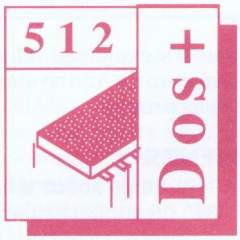
Q takes you back to the main text screen and you can save your new levels from here.

It's worth noting that if the game contains large numbers of the same block that are all cleared at once there will be a delay while *Zeus* sorts things out. You just need to wait until the block cursor reappears. Have fun!

```
   10 REM Program BDATA
   20 REM Author  S. Sexton
   30 REM Version A1.3
   40 REM BEEBUG  April 1992
   50 REM Program subject to copyright
   60 :
  100 MODE7
  110 INPUT"Filename:"file$
  120 out%=OPENOUT(LEFT$(file$,7))
  130 RESTORE 1000
  140 FOR A%=0 TO 4:PRINT#out%,"T H Pine
apple",300*(5-A%):NEXT
  150 FOR L%=1 TO 8
  160 T%=0:FOR A%=0 TO 7
  170 READ X%:T%=T%+X%:PRINT#out%,X%
  180 NEXT
  190 FOR B%=1 TO T%:READ X%,Y%
  200 PRINT#out%,X%,Y%:NEXT
  210 NEXT
  220 CLOSE#out%:END
  230 :
  240 REM Levels Data
 1000 DATA 9,2,2,3,0,0,0,0
 1010 DATA 4,5, 5,5, 6,5, 6,6, 7,6, 7,7,
 8,7, 7,8, 7,9
 1020 DATA 4,6, 8,9
 1030 DATA 5,6, 6,7
 1040 DATA 5,7, 6,8, 8,8
 1050 :
 1060 DATA 15,2,4,4,2,0,0,0
 1070 DATA 4,5, 5,5, 6,5, 7,5, 8,5, 4,6,
 5,6, 6,6, 7,6, 8,6, 5,7, 6,7, 8,7, 6,8,
 8,8
 1080 DATA 9,5, 6,9
 1090 DATA 9,6, 5,8, 7,8, 4,9
 1100 DATA 4,7, 9,7, 5,9, 8,9
 1110 DATA 7,7, 4,8
 1120 :
 1130 DATA 15,3,3,0,0,0,0,0
 1140 DATA 5,4, 6,4, 7,4, 8,4, 7,5, 8,5,
 5,6, 6,6, 7,6, 6,7, 7,7, 7,8, 8,8, 7,9,
 8,9
 1150 DATA 4,4, 8,6, 6,9
 1160 DATA 4,6, 9,6, 6,8
 1170 :
 1180 DATA 8,3,4,4,2,0,0,0
 1190 DATA 7,4, 8,4, 7,5, 8,5, 7,6, 7,7,
 7,8, 7,9
 1200 DATA 5,4, 6,5, 8,8
 1210 DATA 6,4, 5,6, 6,8, 8,9
 1220 DATA 5,5, 6,6, 8,7, 5,8
 1230 DATA 8,6, 5,9
 1240 :
 1250 DATA 14,3,3,6,0,0,0,0
 1260 DATA 6,5, 8,5, 6,6, 4,7, 5,7, 7,7,
 8,7, 5,8, 7,8, 8,8, 5,9, 6,9, 7,9, 6,10
 1270 DATA 5,3, 9,3, 9,5
 1280 DATA 7,3, 5,5, 7,5
 1290 DATA 5,4, 7,4, 9,4, 6,7, 4,8, 6,11
 1300 :
 1310 DATA 27,12,7,4,6,3,0,0
 1320 DATA 2,5, 2,6, 2,7, 2,8, 2,9, 3,9,
 4,9, 4,10, 5,2, 5,3, 5,4, 5,5, 5,6, 5,7
, 5,8, 5,9, 6,9, 7,9, 7,10, 8,2, 8,3, 8,
4, 8,5, 8,6, 8,7, 8,8, 8,9
 1330 DATA 2,2, 4,2, 3,3, 4,4, 7,4, 3,5,
```

```
 6,5, 4,6, 3,7, 6,10, 8,10, 2,11
 1340 DATA 6,3, 7,7, 2,10, 8,11
 1350 DATA 2,3, 4,3, 3,4, 7,5, 6,6, 4,7,
 3,11
 1360 DATA 7,3, 6,4, 4,5, 3,6, 3,10, 5,1
0
 1370 DATA 7,6, 6,7, 5,11
 1380 :
 1390 DATA 8,3,3,2,0,0,0,0
 1400 DATA 5,5, 7,5, 8,5, 5,6, 7,6, 5,7,
 7,7, 5,8
 1410 DATA 4,5, 6,5, 4,8
 1420 DATA 6,6, 8,6, 4,7
 1430 DATA 6,7, 8,7
 1440 :
 1450 DATA 10,3,3,2,0,0,0,0
 1460 DATA 7,4, 7,5, 8,5, 7,6, 7,7, 7,8,
 8,8, 5,6, 5,7, 5,8
 1470 DATA 6,4, 6,7, 6,9
 1480 DATA 6,5, 4,7, 6,8
 1490 DATA 4,6, 6,6
```

```
   10 REM Program Zeus2 (Part2)
   20 REM BEEBUG  April 1992
   30 :
  210 IF G%=101 OR G%=69 PROCedit
 2930 DEFPROCclearhi
 2940 FOR A%=0 TO 4:hi$(A%)="T H Pineapp
le":hi%(A%)=300*(5-A%):NEXT
 2950 ENDPROC
 2960 :
 2970 DEFPROCedit:PROCclearhi
 2980 PROCshow(9)
 2990 COLOUR2:PRINTTAB(15,12)"Edit"
 3000 GX%=1:GY%=1:REPEAT
 3010 PROCcursor(GX%,GY%,9):Z%=GET
 3020 PROCcursor(GX%,GY%,9)
 3030 IF (Z%=80 OR Z%=112) IF FNlevelok
PROCtest
 3040 IF Z%>48 AND Z%<56 PROCdoblock(GX%
,GY%,Z%-48)
 3050 IF Z%=32 PROCdoblock(GX%,GY%,0)
 3060 IF Z%=13 PROCdoblock(GX%,GY%,-1)
 3070 IF Z%=89 OR Z%=121 PROCcopy(FNsele
ct("Yank from:"),9):PROCshow(9):COLOUR2:
PRINTTAB(15,12)"Edit"
 3080 IF (Z%=83 OR Z%=115) IF FNlevelok
PROCcopy(9,FNselect("Shove to:"))
 3090 IF Z%=82 OR Z%=114 S%=FNselect("Sw
ap level:"):D%=FNselect("To level:"):PRO
Ccopy(S%,0):PROCcopy(D%,S%):PROCcopy(0,D
%)
 3100 IF Z%=136 GX%=GX%+(GX%>1)
 3110 IF Z%=137 GX%=GX%-(GX%<12)
 3120 IF Z%=138 GY%=GY%-(GY%<12)
 3130 IF Z%=139 GY%=GY%+(GY%>1)
 3140 UNTIL Z%=81 OR Z%=113:ENDPROC
 3150 :
 3160 DEFPROCdoblock(BX%,BY%,BT%)
 3170 IF playfield%(BX%,BY%,9)>=0 IF typ
e%(playfield%(BX%,BY%,9),9)>0 type%(play
field%(BX%,BY%,9),9)=type%(playfield%(BX
%,BY%,9),9)-1
 3180 playfield%(BX%,BY%,9)=BT%:IF BT%>=
0 type%(BT%,9)=type%(BT%,9)+1
 3190 COLOUR coldat%(BT%+2):PRINTTAB(BX%
+1,BY%*2);CHR$(240+BT%*2);TAB(BX%+1,BY%*
2+1);CHR$(241+BT%*2)
 3200 ENDPROC
 3210 :
 3220 DEF FNselect(A$):COLOUR3
 3230 PRINTTAB(9-LEN(A$)/2,27)A$
 3240 REPEATI%=GET:UNTIL I%>48 AND I%<57
 3250 PRINTTAB(9-LEN(A$)/2,27)SPC(LEN(A$
)):=I%-48
 3260 :
 3270 DEF FNlevelok:ok=TRUE
 3280 FOR T%=0 TO 7
 3290 IF type%(T%,9)=1 ok=FALSE
 3300 NEXT:IF NOT ok COLOUR3:PRINTTAB(2,
27)"Check blocks!":VDU7:PROCwait(40):PRI
NTTAB(2,27)SPC(13)
 3310 =ok
```



512 Forum

by Robin Burton

Over the last three issues we've taken a fairly detailed look at various elements of the GEM system supplied on the 512's issue discs.

We'll round off our investigation with a brief recap of the items covered, but there are also some overall considerations that will affect the choices you'll need to make.

MEMORY CONSTRAINTS

We looked at the various elements that go to make up the GEM system, so you know that in essence configuring the system only involves the changing and copying of a few files. In the case of the new black and white screen driver this is simply so that the program works at all, while for ASSIGN.SYS the purpose of the changes is to alter the range of fonts at your disposal for display on the screen. The only other thing to do is to copy the appropriate files to the working GEM disc.

However, there are other points to weigh up before you do any of this. The biggest limiting factor on how many font files you can include in your new version of ASSIGN.SYS, for example, will be the amount of free RAM in your 512. This is especially critical in unexpanded machines.

The major factor in this is the number and size of screen fonts to be included, because these are memory resident, even if you don't use them all. As an illustration, just adding the two 36 point font files to a standard system means that only one full size window can be opened in GEM Paint for example. By the way, printer fonts have no effect on memory, as these files are only loaded when they're needed during printing.

You can of course make savings in other areas, for example you can increase memory by discarding the desktop accessories, simply done by deleting their files. If these files are loaded, like fonts, they consume RAM even if you don't use the accessories. CALCLOCK.ACC contains the calculator and print spooler, while SNAPSHOT.ACC is the snapshot file. Removing these two files will save about 40K which GEM can use for other purposes.

On a 1Mb 512 you'll have fewer problems, but you need to make adjustments for GEM to use your extra memory. The line:

```
ADDMEM 333
```

in GEM.BAT needs changing, but there's a bit of a compromise here.

Not all GEM applications use allocated GEM memory in quite the same way. GEM Paint for example, stores screen images within the allocated memory, so purely for the purposes of Paint the larger the allocation the better. However, on the other side of the fence, GEM Write stores working documents in the memory remaining outside the specified allocation, so for this application the smaller the initial allocation the better, up to a point.

Of course one possibility is to have a version of GEM.BAT for Paint (perhaps called PAINT.BAT) and another for Write, each with different memory allocations. This does mean however, that importing a Paint file into a document could become very tedious if not impossible, requiring a reboot of GEM between applications.

A more sensible approach is to find a value that suits both applications in your system and, if necessary, stick to smaller documents with more of them. David Harper suggests that 'ADDMEM 500' is a reasonable compromise for expanded memory 512s.

HINTS, TIPS AND ODDMENTS

I'll use the rest of this Forum to cover a number of sundry points that, judging by the occasional but regular queries, still cause the occasional bit of head scratching.

First, DOS function keys, with a bit of background thrown in for general interest. There are ten standard function keys on PCs, so a DOS application can reasonably expect to to use all of them if it needs to.

As an aside, in fact there are more on most machines. Twelve is typical on AT-type PCs, but as many as twenty is normal on current models. However, for reasons of backward compatibility with older hardware such as XT-type PCs, only the first ten function keys return a key-code in DOS 'get character' function calls. To read the extra keys, programs must scan the keyboard at a lower level, but most applications don't do this because there are still a lot of ATs, XTs and earlier models in use.

All this means that most programs, except the latest that specify a 386 machine as the minimum, will work on the 512, at least as far as the ten function keys are concerned. The bit that can confuse 512 users is that the function keys are numbered from one to ten on a PC keyboard, while the BBC micro's are zero to nine. There's no problem though, as function key zero in the 512 acts as function key ten in a PC.

I know of one or two quite experienced 512 users who have been caught out by this bogus problem, although it might seem obvious when it's explained. Once again it's a case of Acorn not documenting the facts properly.

PC DEVELOPMENTS

I've been asked a few times about what exactly 'XT' and 'AT' mean, particularly with reference to potential software compatibility. Having mentioned both of these this month, here goes.

The first PCs actually used 16-bit processor chips, but memory handling in those machines was old 8-bit technology. The result was that they were slow, since a 'load processor register from memory' instruction required two 8-bit reads of memory. Of course machines improved, and along came the XT, short for 'eXtended- Technology'. These were still slow by today's PC standards, using the same processor chips as their predecessors, (usually an 8086) but at least they had a 16-bit data bus so they were quicker than the earlier models. Intel then started to develop a new generation of processors, of which the 80186 used in the 512 was the first attempt.

The 80186 didn't catch on for PCs, primarily because very shortly after it appeared the 80286 followed (and everyone knew it was on the way). The 286 had more efficient memory management which was quicker still, but which also allowed the processor to address 16Mb of memory as opposed to the 1Mb limit of the earlier processors.

The 80286 processor however, was still a 16-bit device, and retained software compatibility with XTs (and largely the 512 too). All machines using an 80286 chip became generally known as ATs, short for 'Advanced Technology', after the IBM machine range of that name. As can be seen, the 512 fits into the scheme of things somewhere between an XT and an AT, both in terms of its origins and its performance.

After this, hardware remained fairly static for a time, because although the 80386 chip (a 32-bit processor) appeared several years ago, the existing XT/AT software market was much too big to ignore or abandon. Of course DOS was occasionally revised and updated while 'Lotus-Intel-MicroSoft Expanded Memory System' (LIMS EMS for short) appeared during this period too.

Although the 80386 offered yet faster and better memory management, with the potential to address up to 2^32 bytes (4 gigabytes) with higher processor speeds, very little software was produced to exploit the new chip. There were simply too few 386s to make it worthwhile.

Until now that is. The current top of the range PC processor is now the 80486 DX33. This is a monster of a chip with built-in maths co-processor, on-board high-speed instruction cache with pre-fetch and pipelining, plus dedicated built-in fast memory management circuits. In this chip the processor neither knows nor cares what's happening in the outside world.

The fastest current 486 is 33MHz, the same clock speed as the fastest 386, but there the similarity ends. A 33MHz 486 machine is about four to six times faster than a 33MHz 386 PC and around ten times the speed of a 12/16MHz 286 AT machine. That's not all as a 50MHz 486 is about to appear, plus another add-on chip which doubles the speed of any 486 processor. There's also an 80586 lurking in the future. To put all this in context, remember that the current Acorn ARM3 runs at 25MHz.

Over the last year, since the appearance of 486 machines, the price of 386 machines has not only fallen, but plummeted. What's more, because of aggressive marketing in the backwash from this, so has the cost of a 486 PC. Basic 486 machines were expected to be in the £3,000 to £4,000 bracket upwards (large file-servers and serious CAD machines still are), but it's now possible, with a bit of shopping around, to get an SVGA 486 PC complete with screen and hard disc at around £1,500 and a VGA 386 machine at well under £1000 (for example, BEEBUG can supply their own 33MHz 486 PC for £1699, with SVGA, 60Mb hard disc, 4Mb RAM, a mouse, Windows 3 and MS-DOS 5). In fact these falling prices haven't quite bottomed-out yet.

THE FUTURE

What's this got to do with the 512? Well, the result of the past year's price wars, plus cessation of 286 chip production, is that both users and software suppliers alike increasingly see at least a 386 SX (a 16-bit version of the 386) as being the minimum entry-level to PC computing. The inevitable consequence of this change in perception is that software is now beginning to appear that simply won't run on a 286 (much less an XT) or if it does the machine is on its knees.

For example, MicroSoft Windows is current 'flavour of the month' (it will probably last longer than a month though) and Windows-only applications are naturally appearing too. It's generally accepted that a 2Mb 33MHz 386DX is the minimum system you should try to use to run this sort of software, but a 4Mb 486 would obviously be very much more suitable.

For the 512 user this unfortunately means that more and more new PC software, including updates to existing packages, can be expected not to run. It isn't a new problem, more of a natural progression, but it's certainly becoming very much more likely that you'll hit problems as a result of the changes of the last twelve months.

The problem isn't only the long known compatibility difficulties of the 512, nor is it the anticipated problems of 32-bit processor architecture. Much more simply

continued on page 48

Finding a Route in a Network

by Ian Palmer

Back in BEEBUG (Vol.9 No.3) there was a letter requesting a program for a 'shortest route' problem. I have not seen a response before now, so I hope this article will explain how this and related problems can be solved.

There are five listings that go with this article. The first three are programs proper, while the final two are just data which the programs can use. First you need to type in all the listings, separately, and save them as follows:

Listings 1 and 3 should be saved as programs called *Route1* and *Route3* respectively. Listings 2, 4 and 5 should be typed in as normal but then spooled out with filenames *Route2a*, *List1* and *List2* respectively.

Both the data files (List1 and List2) can be used with any of the three programs, and simply need to be *EXECed in after loading the relevant program. When typing in the first two listings (Route1 and Route3) line numbering is important as the second listing (Route2a) should be *EXECed in over the first listing and then resaved to produce the second program (Route2).

The programs work on networks of nodes. The networks can be anything, for example the first set of data refers to figure 1, where the nodes are A to J. The second set of data contains places in mainland Britain (the nodes are the places) and the links are road links.

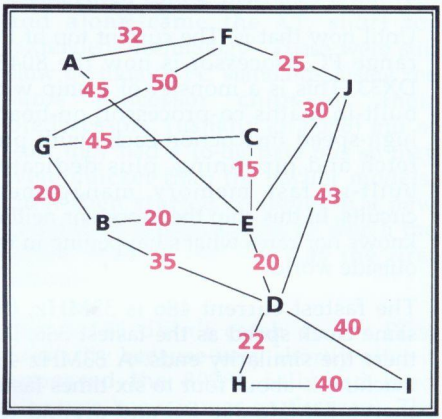


Figure 1

The first part of each set of data is simply a list of all the nodes, terminated by **END. The main part of the data then follows which is a list of all the links in the network. This data is of the following format:

Node, Node, Length of Link, Average Speed

All links are taken to be bi-directional.

The first program (Route1) will take a network and try to find the shortest route such that each node is visited once, and once only. With this program the

shortest route is that which is of shortest length, and in fact the average speed part of the data is ignored by this program.

The crux of the program is the procedure *PROCfind* which takes one parameter, the number of nodes in the current route found so far (thus it is initiated by the call PROCfind(0)).

This procedure simply cycles through all the nodes and tries to assign each to this next place in the route. It needs to check that this node has not already been used in the route and this is done by saving an array *v%()* which has one element for each node which holds TRUE if the node has been used, and FALSE if not. Also, the procedure needs to check that this node can be reached from the previous node in the route; this is simply looked up in the array *d()* which is built up from the data at the beginning of the program.

All being OK we simply assign this node to this position in the route and recursively call *PROCfind* for the next position. One other test is needed in *PROCfind* and that is the terminating condition. Here, this is when all nodes have been assigned (i.e. we have found a route). This is tested by seeing if the number of positions found is equal to the total number of nodes. If so, we print out the list and try to find the next route.

The list of positions is only printed out if the total length of the path is less than or equal to the length of the last path found. Thus the last list printed before the program terminates is the shortest route.

You may have noticed that *PROCfind* isn't a particularly well written procedure. For a start it has dreaded GOTOs in it. There is, however, a good reason for this. The GOTOs are trying to emulate a FOR-NEXT loop as you can only nest up to 10 FOR-NEXT loops in BBC Basic, and this would limit us to a network of only 10 nodes.

Even with the GOTOs the size of the network is still limited because recursion takes up quite a large amount of room on Basic's stack. Thus I have kept the number of local variables to a minimum to try to maximize the size of network allowed. Making sure you have PAGE set as low as you can will also help. On the first set of data, the program takes about one and a half minutes to execute and yields the result that the shortest route is AFJECGBDHI with length 264 units (or the reverse route).

With the second set of data there is no result, as there is no path which takes you around all the places without revisiting some. If you simply add the following line:

```
1515 DATA Dover,Sheffield,229,60
```

then the program will yield a 1380 mile route around Britain after about a seven minute pause.

It's all very well being able to find the shortest route around all the places, but what if you want to go, say, from Dover to Aberdeen in the shortest distance. Well the alterations in the second listing (Route2) will make your program do this. First the program will pause for a few seconds while it reads through the data. Then it will ask you for the start location and the destination.

The alterations basically limit the start location to the one you type in and change the terminating condition inside *PROCfind* to testing for the last position in the route being equal to the destination. This program executes in less time than the first program, and again the last printed route is the shortest (in length).

If, for either Route1 or Route2, you wish that the speed of each link should be taken into consideration, i.e. you wish to find the fastest route not just the shortest in terms of distance, then the following changes are necessary:

In line 160 change both occurrences of '=d' to '=d/t'

Change the word "dist" on line 1130 to "time"

This sort of problem is one which computers are often asked to perform. One place where this problem occurs is in computer communication. Most large networks are based on the mesh, similar to that in figure 1, as opposed to the ring or bus which both have no problems in routing.

Each node can simply test to see how long it takes to send a message to one of its immediate neighbours, and this fact is used in the following way. What is desirable is to be able to build up a table of best routes to use when sending messages to any node on the network. In fact all that is needed is to know to which of its neighbours it should send messages for each node on the network, as that node will know the next one, and so on. The third program (Route3) will build up just such a table in array *s%()* which is indexed by this node number and the destination node number, and which yields the node number of the neighbour to which to send the message.

First the program fills various arrays with information from the data, and this time the speed of the link is noted, thus fastest routes will be obtained. Then basically each node, in turn, tells each of its neighbours that it's there, by calling procedure *PROCinf*. This procedure then works out the time taken to get the message. This is worked out from the length and speed of the link. If a message has already been received via a quicker route then an exit is made from *PROCinf*, otherwise this node then sends this time to all its neighbours so that they can see if this route is faster than any they've seen before, and so on.

Effectively, information floods the network until the shortest routes are known by each node. Then that is all that is needed to allow any message to go via the fastest route simply by looking up the table. This method can be used for our purpose, although it takes longer for a single run than Route2 and produces the same results (if using Route2 where speed is also taken into account) the usefulness of this program becomes apparent when more than one query is asked.

With Route2 on List2 each query takes somewhere between 30 seconds and 1 minute to execute. With Route3 there is about one and a half minute pause while the table is set up, but then each query is instantly answered. The only problem with Route3 is that you are further limited to the size of network allowed because of the increased number of local variables required and the increased array space.

```
   10 REM Program Route1
   20 REM Version B2.00
   30 REM Author  Ian Palmer
   40 REM BEEBUG  April 1992
   50 REM Program subject to copyright
   60 :
  100 DIM d(25,25),N$(25),v%(25),n%(25)
  110 N%=-1:REPEAT:N%=N%+1:READ N$(N%):U
NTIL N$(N%)="**END"
  120 REPEAT:READ S$,F$,d,t
  130 FOR A%=0 TO (N%-1)
  140 IF S$=N$(A%) s%=A%:ELSE IF F$=N$(A
%) f%=A%
  150 NEXT
  160 IF S$<>"**END" d(s%,f%)=d:d(f%,s%)
=d
  170 UNTIL S$="**END"
  180 md=-1:d=0:PROCfind(0)
  190 END
  200 :
 1000 DEF PROCfind(A%)
 1010 LOCAL B%:B%=0
 1020 IF A%=N% PROCprint:ENDPROC
 1030 IF B%=N% ENDPROC
 1040 IF A%<>0 IF v%(B%) OR d(n%(A%-1),B
%)=0 B%=B%+1:GOTO1030
```

```
 1050 v%(B%)=TRUE:IF A%<>0 d=d+d(n%(A%-1
),B%)
 1060 n%(A%)=B%:PROCfind(A%+1):v%(B%)=FA
LSE:IF A%<>0 d=d-d(n%(A%-1),B%)
 1070 B%=B%+1:GOTO1030
 1080 :
 1090 DEF PROCprint
 1100 IF md<>-1 AND d>md ENDPROC
 1110 PRINT"Route : "
 1120 FOR C%=0 TO N%-1:PRINTN$(n%(C%));"
] ";:NEXT
 1130 PRINT"dist ";d:md=d:ENDPROC
```

```
  60 REM Program : Route2
  70 REM *EXEC over Route1
 171 INPUT'"Start : "S$:INPUT"Dest. : "
D$:FOR A%=0 TO N%-1:IF S$=N$(A%) s%=A%:E
LSE IF D$=N$(A%) f%=A%
 172 NEXT
 180 md=-1:d=0:v%(s%)=TRUE:n%(0)=s%:PRO
Cfind(1)
 1020 IF n%(A%-1)=f% PROCprint:ENDPROC
 1120 FORC%=0 TO A%-1:PRINTN$(n%(C%));"
] ";:NEXT
```

```
  10 REM Program Route3
  20 REM Version B1.60
  30 REM Author  Ian Palmer
  40 REM BEEBUG  April 1992
  50 REM Program subject to copyright
  60 :
 100 MODE135:VDU23;11,0;0;0;0;
 110 PRINTTAB(10,1);CHR$141;CHR$130;"Sh
ortest Route";TAB(10,2);CHR$141;CHR$130;
"Shortest Route"
 120 PRINT'"Setting up table - Please w
ait":DIM d(25,25),t(25,25),s%(25,25)
 130 DIM N$(25):FOR A%=0 TO 25:FOR B%=0
 TO 25:s%(A%,B%)=-1:NEXT:NEXT
 140 N%=-1:REPEAT:N%=N%+1:READ N$(N%):U
NTIL N$(N%)="**END"
 150 REPEAT:READ S$,F$,d,t
 160 FOR A%=0 TO (N%-1):IF S$=N$(A%) s%
=A%:ELSE IF F$=N$(A%) f%=A%
 170 NEXT
 180 IF S$<>"**END" d(s%,f%)=d:t(s%,f%)
=d/t:s%(s%,f%)=f%:s%(f%,s%)=s%:t(f%,s%)=
d/t:d(f%,s%)=d
 190 UNTIL S$="**END"
 200 FOR A%=0 TO (N%-1):FOR B%=0 TO (N%
-1):IF s%(A%,B%)=B% PROCinf(B%,A%,A%,0,0
```

```
):PROCinf(A%,B%,B%,0,0)
 210 NEXT:NEXT
 220 REPEAT
 230 INPUT'"Start : "S$:INPUT"Dest. : "
D$:s%=-1:f%=-1:FOR A%=0 TO N%-1:IF N$(A%
)=S$ s%=A%:ELSE IF N$(A%)=D$ f%=A%
 240 NEXT
 250 IF s%=-1 PRINT"Start not valid"
 260 IF f%=-1 PRINT"Dest. not valid"
 270 IF s%>=0 AND f%>=0 PROCshow
 280 UNTIL S$="EXIT"
 290 END
 300 :
1000 DEF PROCshow
1010 PRINT'"Travel from ";S$;" to ";D$
1020 PRINT"Distance : ";d(s%,f%)
1030 PRINT"Total time ";t(s%,f%)
1040 PRINT'"Route : ":P%=s%:REPEAT
1050 PRINT"From ";N$(P%);" to ";:P%=s%(
P%,f%):IF P%<>-1 PRINTN$(P%):ELSE PRINT"
eek - No route"
1060 UNTIL P%=f% OR P%=-1:ENDPROC
1070 :
1080 DEF PROCinf(me%,tm%,to%,d,t)
1090 IF me%=to% OR me%=tm% ENDPROC
1100 IF t(me%,to%)<>0 AND (t+t(tm%,me%)
)>t(me%,to%) ENDPROC
1110 LOCAL I%
1120 d(me%,to%)=d+d(tm%,me%):t(me%,to%)
=t+t(tm%,me%):s%(me%,to%)=tm%
1130 I%=0
1140 IF s%(me%,I%)=I% AND I%<>tm% PROCi
nf(I%,me%,to%,d(me%,to%),t(me%,to%))
1150 I%=I%+1:IF I%<N% GOTO1140
1160 ENDPROC
```

```
70REM data file : List1
1170:
1180REM List of nodes (end with **END)
1190DATA A,B,C,D,E,F,G,H,I,J,**END
1200:
1210REM Node Node Length (Speed)
1220DATA A,F,32,70
1230DATA F,J,25,70
1240DATA G,F,50,60
1250DATA A,E,45,40
1260DATA G,C,45,40
1270DATA E,C,15,30
1280DATA E,J,30,40
1290DATA G,B,20,30
```

Continued on page 46

Public Domain Software

Alan Blundell looks at the range of utility software available in the public domain.

As promised last month, I'll take a look at utility software this month. The public domain includes a growing amount of utility software; when I first got involved in PD software, I expected this to be by far the largest category of software, because of the ease with which Acorn computers can be programmed and the interest in the inner workings of the system which large numbers of BBC owners seem to have. It didn't turn out that way, at least so far, as games are available in larger quantity, but there is a fair selection of utilities available.

Radio amateurs seem to have a need for particular utilities as tools for their hobby, such as bearing and distance calculations, antenna efficiency calculations, etc. A small selection of utilities are available in this area, but although I profess no great understanding of the subject, from comments received they are less than accurate in their calculations and not particularly well presented.

A radio amateur logbook program sold commercially by Technical Software was at one time mistakenly distributed as PD, due to a bit of 'hacking' somewhere along the line and a consequently mistaken library submission (if we look at it charitably...). This is an example of the problems caused by pseudo-PD software, and illustrates why I see a need for PD libraries to exercise great caution when accepting software submissions. I still frequently see copyright software distributed as PD, without evidence of the copyright holder's permission. Although I accept that it can be very difficult to locate an author to gain permission, this does not mean that such permission can be assumed! Technical Software still sell their logbook program and I would be happy to pass on their address if anyone wants it. A distance and bearing utility written by Glynn Fowler has recently become available as PD and is understood to be of better quality than most PD utilities of this type.

There are a number of utility ROM images available which will be of wider interest. A.M.Flintham, whose games (such as Sorcerer's Domain, Exiz, Zedon) are popular, has also written a selection of ROM images which include a toolkit ROM and a set of sideways RAM notepad utilities. Steven Flintham (no relation, to the best of my knowledge) has also produced a range of utilities, including a Master series initialisation ROM, an ADFS utilities ROM, a text compression utility, an ADFS directory tree viewer and several others. Some of Steven's projects, for example the ADFS Utilities ROM, are early versions, which do not include a vast range of facilities. He has released them as 'freeware' (effectively PD software for which the author retains copyright) so that others can help him to add to the software by offering their own '*command' routines for example, enabling each to become a strong and genuinely useful piece of software over time.

One particularly good aspect of Steven's work deserves mention - his documentation. Every program comes with a text file containing comprehensive

documentation, which is genuinely useful with any software except perhaps simple games which are completely self-explanatory. The more we see of this standard of documentation, the better in my view. If you are writing software which you want to release to other people, remember that documentation takes little time compared with how long it takes you to complete your masterpiece, but can make all the difference to someone trying to make use of it.

The documentation provided for the small but well-produced range of PD software available from Lancaster University's National Public Domain Software Archive is also exemplary. NPDSA is a central resource maintained by the University which covers a range of micros. Large ranges of software for PCs, Amigas and STs are available, but there is a BBC area. It includes some less common utilities such as a file archive/unarchive system, a customisable toolkit ROM, 'HandiROM', which is useful as an aid to some disabled people in their use of their micro, a printer redirection utility and a function keystrip printer.

Lars Osterballe, who seems to have been mentioned more than once in this column, has produced several utilities, including a disc cache program which uses spare sideways RAM to buffer disc accesses. Although I haven't used it myself, I see this being most useful for byte level access to files, rather than straightforward load/save functions. Also of note is his 'Dynamics ROM', which is a system for collecting your favourite machine code utilities into one ROM image for use in sideways RAM: the 'dynamic' refers to the fact that these utilities may be added to or removed from the system at your convenience.

Specifically for Master users, Andrew Fiddaman has developed a Master series utility ROM with a wide range of commands which he has released as shareware.

As you might expect, there are quite a number of commonly used utilities such as character designers, font designers, function keystrip printers and sound envelope designers. Each tends to have its own strengths and weaknesses, but I think that most people probably have at least one or two such utilities by now, unless they have no use for them at all, as programs of this type have been published in quantity over the years in magazines, via telesoftware and by small commercial operations.

Sideways ROM/RAM utilities are another popular area with programmers, with programs such as ROM image load/save utilities, ROM managers, emulations of the Master's *UNPLUG command, and others for Model B owners. Andrew Pepperell has produced a memory utilities ROM for use on either range of micros, which includes the full source code on disc. In my opinion, this is an excellent practice; if you release your work as PD, what is the point of keeping the source code to yourself? Availability of source code helps less experienced programmers to develop their own skills and someone may even produce an improved version of your program which will be of benefit to you as well as to others. So long as everyone acts honourably and doesn't try to pass off your work as their own, everyone gains.

Allan Kelly has produced a range of utilities which were originally distributed by the BBC's Telesoftware service before it sadly ceased to operate.

As you will know from last month's column, telesoftware is not automatically PD, although large numbers of people may have received copies free of charge. However, Allan has re-released his programs as PD, including a memory 'snapshot' utility for sideways RAM, 'Eclipse' which is a SWR utilities system and macro language, 'Banners' - a utility for printing large text, and printer dump utilities.

Finally, several teletext editors have been released as PD. 'Teletext Editor' is not really a good name for these as several include screen editors, carousel display facilities and generally everything needed to set up a complete viewdata system. Systems written by Rafael Jay, Jonathan Harston and Alan Phillips (mentioned previously for his 6502 assembler system) are available; choice is probably a personal issue and all are well-produced, but my choice would be Alan Phillips' 'Fanfare' system for completeness and for documentation. It has the additional advantage of being in the form of a ROM image for convenience, although I accept that this is no advantage if you have no sideways RAM!

That's all there is room for this month, but I think PD utilities have now received a fair airing. I have received a fair bit of educational software recently, so it seems appropriate to turn our attention to that next month. I hope to cover Peter Davey's work (mentioned briefly last month) and a range of software recently re-released as shareware by John Lyons Educational Software, as well as an overview of what is generally available in the public domain for educational use.

BEEBUG Workshop (continued from page 43)

```
1300DATA B,E,20,30
1310DATA E,D,20,30
1320DATA B,D,35,30
1330DATA D,H,22,60
1340DATA H,I,40,60
1350DATA I,D,40,60
1360DATA D,J,43,70
1370DATA **END,**END,0,0
```

```
70REM data file : List2
1170:
1180REM List of nodes (end with **END)
1190:
1200DATA London,Birmingham,Leeds
1210DATA Glasgow,Manchester,Dover
1220DATA Aberdeen,Nottingham,Cardiff
1230DATA Oxford,Canterbury,Lincoln
1240DATA York,Sheffield,Blackpool
1250DATA Liverpool,Norwich,Hull
1260DATA **END
1270:
1280REM Node Node Length Speed
1290:
```

```
1300DATA Dover,Canterbury,15,70
1310DATA London,Canterbury,58,70
1320DATA London,Birmingham,111,70
1330DATA Birmingham,Nottingham,49,70
1340DATA Nottingham,Manchester,71,70
1350DATA Manchester,Leeds,41,70
1360DATA Leeds,Hull,56,70
1370DATA Lincoln,Hull,38,60
1380DATA Nottingham,Lincoln,35,70
1390DATA Liverpool,Manchester,34,70
1400DATA Liverpool,Blackpool,46,70
1410DATA Manchester,Blackpool,47,70
1420DATA Glasgow,Blackpool,189,70
1430DATA Glasgow,Aberdeen,145,70
1440DATA Norwich,Birmingham,161,70
1450DATA Norwich,London,114,70
1460DATA Oxford,London,56,70
1470DATA Oxford,Cardiff,104,70
1480DATA Cardiff,Birmingham,102,70
1490DATA Hull,York,39,70
1500DATA Leeds,York,24,70
1510DATA Sheffield,Leeds,34,70
1520DATA **END,**END,0,0
```

Mr Toad's Machine Code Corner

by David Holton

In the next few issues this column will be concentrating solely on assembly language programming, not only for the 'old hands', but also for beginners; we hope, in fact, that we may encourage those Basic programmers who are looking at machine code much as one looks at a cold swimming pool, wondering if the time is right to take the plunge - and yes, we all went through that stage. We don't intend to give a course, more a bit of everything; above all, we'd like some active response from you. We hope to give you some hints and tips and print any you may send in; there will be one or two puzzles and some discussions of assembly language techniques, until all the triodes and pentodes in the old Beeb finally burn out.

During the first World War, a certain crossroads on the Menin Road out of Ypres was nicknamed "Hell Fire Corner" because it was in full view of the Germans and was shelled day and night. Mr Toad has a feeling that his Machine Code Corner is going to be a bit like that when the letters start flowing in, pointing out all the mistakes and bugs and dodgy pokes.

Let's start with a bit of deep philosophy. A month or two ago, a reader passed on a useful address for poking in some value or other, but ended his letter by saying that direct manipulation of memory is not to be recommended. Mr Toad thinks that those days are past. As long as there was a prospect of new versions of the MOS coming out, it was in everyone's interest to stick to OSBYTEs and OSWORDs instead of going straight to the system variables. It is now clear that no more versions of the BBC will ever appear, and so long as we know that a certain address has the same function in all existing machines, we may as well cut corners. Writers of big pieces of commercial software - assuming that any such are still likely to be written, which is a big 'if' - might still need to take account of such things as BBC emulators on other computers, but most of the kind of stuff we type in from magazines such as BEEBUG will run on machines which will never again be altered.

Another important issue is Tube compatibility: many of the common pokes to variables in page zero, page two and so on cause chaos when the code is running on the far side of the Tube, because the whole organisation of memory is different and many areas of the I/O side have no counterparts there. Most of the OSBYTEs and documented routines take account of this, but how many people have 6502 co-processors and keep their machines configured to TUBE? I've got the Turbo board on one of my machines, and my own Toad ROM 90 contains a star command to turn it on and off easily, but I rarely use the Turbo board because a lot of the commercial software I use won't run on it. I suspect that this is the norm. More about the Tube in another issue, by the way.

Some examples of what we are talking about: OSBYTE &FC reads the number of the current language ROM, but why bother when we all know that the number lives in location &028C? The fact is well documented and applies to all versions. The currently active ROM, of course - which is not the same thing as the current language - has its number at &F4, and everybody uses that one. It's only a copy; as we all know, the address that really matters is ROMSEL at &FE30. Writing to that address actually switches ROMs, but &FE30 is

stated in all the books I've ever read to be a 'write-only' address, being in the memory select chip and not in RAM. It is the programmer's responsibility to update the copy in &F4 whenever he writes to ROMSEL. I always do it myself. Just turn on your machine now, though, wait for the steam pressure to build up, then type P.?&F4, then try P.?&FE30. See? It never fails, and Mr T knows of at least one MOS routine which shamelessly reads &FE30 directly.

Going back to OSBYTEs, it is well known and documented that in all machines, locations &236 to &28F contain the single bytes accessed by OSBYTEs &A6 TO &FF. Just add &190 (or 400, if you're one of *those* people) to the OSBYTE number and use the address directly. There are one or two common sense exceptions, that's all, but why set up two or three registers and do a JSR just to alter the attributes of the 'bleep' note? Don't get me wrong; there are a lot of OSBYTEs which are very useful, and some are essential, but many have lost their *raison d'etre*, which was simply to assure compatibility with future operating systems.

Right - that should have stirred you all up a bit. Write in to Mr Toad and tell him what a load of trash he has just spouted. Soon in this spot I intend to pass on some useful page zero addresses for various languages and filing systems; a little less well known, I hope, than gems like the high byte of PAGE being at &18. If you know any, send them in. That's it for this month, except to thank Jon Tottman for his super Toad logo. Next month, another lively discussion on whether magnetic-core memory will ever supplant our trusty mercury-filled glass tubes, and whether these new-fangled germanium diodes will ever catch on.

512 Forum (continued from page 39)

it's the amount of processor power needed to drive current applications. If it's any consolation at all, bear in mind that a vast number of XT and AT users are now increasingly finding themselves in much the same boat.

This means that shareware is even more important to 512 users now than before. It's true that 'Windows' shareware is appearing, but shareware is becoming the most likely if not the only place to find new DOS software suitable for trying in the 512. Bear in mind too that existing shareware catalogues will be maintained for the millions of XT and AT users. Conversely, commercial suppliers usually supply and support only the latest version of any package, so when a new one appears the old version is discarded.

There's no need to be depressed, nor to throw your system away and start again. Just like millions of XTs and ATs, the 512 still does a perfectly adequate job for a huge number of applications. I still use mine for writing books and programs. A 486 might be very much faster, but for this sort of job it wouldn't make much practical difference. The only operation in which I'd expect to see any material gain is spell checking, and that's hardly a continuous job.

Maybe your 512 does take a minute to recalculate a large spreadsheet and a fast 486 would do the same job in 2.5 seconds, but does it matter?

I can only paraphrase Mike Williams' words from Postbag in issue 8. Many users still receive excellent service from old and trusted equipment. If you're one of them, why worry about it so long as support continues. Just pity the poor AT owner!

Storeprint

Graham Nunn takes some of the pain out of setting up ViewStore, Acorn's database for the BBC micro.

THE PROBLEM

After using ViewStore for several years and building up a couple of fairly large and useful databases, I had found ViewStore lacking in one particular area. When I wanted information from a database without too much fuss, usually when the computer was not in use, I had to completely setup ViewStore and its associated utilities to access the required data.

This wasn't much of a problem if I only wanted to read the data from the screen, but if a printout was required then far more effort and key presses were needed; the *SELECT UTILITY* is required to choose the required data; you then wait while a *SELECT* file is created before you finally use the *REPORT UTILITY* to print our your data, assuming you have a suitable *REPORT* file already created.

I therefore decided to write a program to simplify all the operations I have described above and alleviate all that keyboard work.

THE SOLUTION

The Storeprint program works in mode 3 and, depending on the number of fields in the selected data file, it should work without the luxurious hardware of either shadow RAM or a second processor.

It reads the data file in the D directory, calculates how many fields there are in each record and then all fields are displayed and/or printed when a search match is found.

When run the program first asks whether a printout is required - all that is needed is a single key Y or N; no Return is necessary. If you want to use your printer you need to add any set-up codes in PROCprint_codes at line 1630.

Next you will be asked if your search is case dependent; again all that's required is a single key Y or N. If N is entered matches will be found for strings irrespective of case. When N is pressed the program will internally convert all search string and data string inputs to upper case (using the FNswap() function from the BEEBUG Function Library). The exception to this is when a numeric search string is entered, when case dependency automatically switches to Y.

The advantage of a case dependent search is that, because the program is not converting to upper case, it runs much faster.

The next input is the name of the ViewStore file, the program automatically adds "D." A heading will now appear to show the number of fields per record, name of data file and its length in bytes.

Now enter the string you are looking for; the program will search every field of each record for a match and each time a match is found the whole record will be printed out, either just to the screen or to screen and printer depending on your earlier choice. Blank fields appear as empty curly brackets.

Pressing Escape at any time during a search will stop the program reporting the number of matches found and records searched. The same will occur when the end of the data file is reached. You will be then given the opportunity to re-search the file using a different search string or quit the program, using either R or Q respectively.

The program should work with any ViewStore data file and without any of the ViewStore utilities or ROM being present.

```
   10 REM Program Storeprint
   20 REM Version B1.2
   30 REM Author  Graham Nunn
   40 REM BEEBUG  April 1992
   50 REM Program Subject to copyright
   60 :
  100 ON ERROR GOTO 1710
  110 MODE3:CLOSE#0
  120 PROCtitle
  130 PRINT"Do you want print out (Y/N)
";:REPEAT:Q$=GET$:Q$=FNswap(Q$,1):UNTIL
Q$="Y" OR Q$="N":PRINTQ$:IF Q$="Y" THEN
PROCprint_codes:print%=2 ELSEprint%=3
  140 PRINTSPC7"Case Dependant (Y/N) ";:
REPEAT:Q$=GET$:Q$=FNswap(Q$,1):UNTIL Q$=
"Y" OR Q$="N":PRINTQ$:IF Q$="Y" THEN cas
e%=FALSE ELSE case%=TRUE
  150 INPUTSPC9"Name of Datafile : "F$:I
FLENF$>10 THEN110
  160 F$="D."+F$:CLS
  170 PROCno_of_fields
  180 PROCsetup
  190 VDU12,print%:PRINT"No. of Fields =
 ";field%;SPC9;"Filename : ";F$;SPC7"Fil
e length = ";length%;" Bytes":VDUprint%:
PRINTTAB(0,1);STRING$(79,"-")
  200 VDU28,0,24,79,2:ON ERROR GOTO 1710
  210 INPUTLINE"Enter Keyword to search
for : "KW$
  220 K%=LEN(KW$)
  230 IF VAL(KW$)>0 THEN case%=FALSE
  240 PRINTSTRING$(79,"-")
  250 VDU28,0,24,79,4:VDU3
  260 IF case% THEN KW$=FNswap(KW$,1)
  270 PROCkeyword
  280 CLOSE#0
  290 PRINTSTRING$(79,"=")TAB(0);found%"
 Matches found"TAB(33)"END OF SEARCH"TAB
(60)record%;" Records searched"'STRING$(
79,"="):VDU3
  300 PRINT"Enter 'Q' to QUIT or 'R' to
RE-SEARCH again : ";
  310 REPEAT:Q$=GET$:Q$=FNswap(Q$,1):UNT
IL Q$="Q" OR Q$="R"
  320 IF Q$="R" THEN PRINT:record%=0:fou
nd%=0:FOR X%=1 TO field%:R$(X%)="":NEXT:
VDU26:GOTO190 ELSE END
  330 END
  340 :
 1000 DEF PROCsetup
 1010 record%=0:found%=0:size%=FALSE
 1020 DIM R$(field%),C$(field%)
 1030 ENDPROC
 1040 :
 1050 DEFPROCno_of_fields
 1060 F%=OPENUP(F$):CLOSE#F%
 1070 IF F%=0 CLS:PRINT"File does not ex
ist!":GOTO150
 1080 F%=OPENUP(F$):PTR#F%=0
 1090 length%=EXT#F%
 1100 REPEAT:B%=BGET#F%
 1110 IFB%=&09 THEN field%=field%+1
 1120 IFB%=&0D THEN GOTO1140
 1130 UNTILFALSE
 1140 CLOSE#0
 1150 ENDPROC
 1160 :
 1170 DEF PROCkeyword
 1180 record%=0:CLS
 1190 F%=OPENUP(F$):PTR#F%=0
 1200 REPEAT:MATCH=FALSE
 1210 FOR X%=1 TO field%
 1220 REPEAT
 1230 B%=BGET#F%
 1240 IF B%=&03 OR B%=&D THEN 1340
 1250 IF B%<>&09 THEN C$=CHR$B%
 1260 IF B%=&01 THEN 1340
 1270 IF B%<>&09 THEN R$(X%)=R$(X%)+C$:C
$(X%)=R$(X%)
 1280 UNTIL B%=&09
 1290 IF K%>=LEN(R$(X%)) THEN size%=FALS
E ELSE size%=TRUE
 1300 IF case% AND size% THEN C$(X%)=FNs
wap(C$(X%),1)
 1310 IF size% THEN Z%=INSTR(C$(X%),KW$)
:IF Z%>0 THEN MATCH=TRUE
 1320 NEXT:record%=record%+1
 1330 PROCprint_match
 1340 UNTIL B%=&01
 1350 ENDPROC
 1360 :
 1370 DEF PROCprint_match
 1380 IF MATCH=FALSE THEN 1440
 1390 FOR X%=1 TO field%
 1400 IFR$(X%)="" THEN VDUprint%:PRINT"{
}" ELSE VDUprint%:PRINTR$(X%)
 1410 NEXT:PRINTSTRING$(35,"-");"Record
No:";record%;
 1420 PRINT;STRING$((79-POS),"-")
 1430 found%=found%+1
 1440 FOR X%=1 TO field%
```

Continued on page 52

BEEBUG Function/Procedure Library (10)

by R.W.Smith

This month we present further functions and procedures from R.W. Smith. Some of these are needed by the procedures in last month's issue on printing. Here we have some useful snippets to help deal with dates and user numbers as well as an extension to the BBC's own error reporting system.

Next month we'll be rounding off this particular series with some very useful stuff to help you get the most out of sideways RAM.

THE FUNCTION/PROCEDURE LIBRARY (PART 10)

Routine 25. Date conversion routine.

Type: FUNCTION
Syntax: N%=FNjdat(K%)
Purpose: a) To reduce a numerical date into an integer of maximum size of 5375 for packing into two bytes.
b) To cause the date to have a numerical sequence for sorting purposes (e.g. birth dates).
Parameters: K% is a numerical date in the form of an integer expressed in decimal as DDMMYY.
Notes: This routine is not a 'number of days' routine. The routine is useful to hold dates infree bytes for passing from one program to another, such as an operational date.
Related: Routine 26 converts the output of this routine to DDMMYY form.

Example:

```
10 N% =FNjdat(120191):?&90=N% DIV 256:
?&91=N%
```

Routine 26: Condensed date conversion routine.

Type: FUNCTION
Syntax: K%=FNfjdat(N%)
Purpose: To convert a two digit condensed date into the accepted form of numerical date.
Notes: It can be used in conjunction with routine 19 FNpformat to printout a date in the form of DD/MM/YY
Related: The date must have been previously condensed by function 25.

Example:

```
10 N%=?&90 *256+ ?&91
20 K%=FNfjdat(N%)
30 PRINT FNpformat(N%,6,"D")
```

Routine 27: Retrieve date and user number from free memory.

Type: FUNCTION
Syntax: DAT=FNrdat
Purpose: To retain a user number and operational date in free memory and call this data in any program. Used for printing to establish operator identity
Notes: The data is placed into free memory by adding the User number (Maximum 6) * 10000 to the date, before using routine 25 FNjdat. Uses the variable USN% to place user number in.
Related: Uses FNfjdat. Used by PROCptrset

Example:

```
10 DAT=FNrdat
20 PRINT FNpformat(DAT,6,"D"),
FNpformat(USN%,3,"I")
```

Routine 28: Error routine with error line printout.

Type: PROCEDURE
Syntax: PROCerr
Purpose: To extend the BBC Basic Error Report to give the line in which the error has occurred as well.
Parameters: None.
Notes: When calling this procedure on an ONERROR line it should be followed by some other action like RUN, END etc.
Related: Uses PROCtell

Example: ONERROR PROCerr:END

Routine 29: Print out buffer message.

Type: PROCEDURE
Syntax: PROCtell(X%,N$)
Purpose: To insert a command into a buffer and to cause this to be executed.
Parameters: X% is the buffer number (0 is keyboard buffer). N$ is the command in string form.
Notes: None.
Related: Used by routine 28.

Example:

```
10 N$="SAVE FRED"
20 PROCtell(0,N$)
```

```
30230 REM Date Conversion Routine.
30240 :
30250 REM Convert to Condensed Form.
30260 :
30270 DEFFNjdat(_%):=(_%DIV10000)+((((_%
DIV100)MOD100)-1)*32)+(((_%MOD100)-80)*3
84)
30280 :
30290 REM Convert from Condensed Form.
30300 :
30310 DEFFNfjdat(_%):=_%DIV384+80+(((_%M
OD384)DIV32)+1)*100+(_%MOD32)*10000
30320 :
30330 REM Retrieve Date & User No. from
Memory Bytes &90 & &91.
30340 :
30350 DEFFNrdat:_%=?&90*256+?&91:USN%=_%
DIV10000:=FNfjdat(_%MOD10000)
30360 :
30370 REM Error Routine.
30380 :
30390 DEFPROCerr: CLS: ONERROROFF:REPORT
30400 _$="L."+STR$(ERL):PROCtell(0,_$):E
NDPROC
30410 :
30420 REM  Print Out Buffer Message.
30430 :
30440 DEFPROCtell(X% , _$)
30450 _$=_$+CHR$13
30460 FOR_%=1 TO LEN(_$) : A%=138 : Y%=A
SC(MID$(_$,_%,1))
30470 CALL&FFF4 : NEXT
30480 ENDPROC
30490 :
```

Storeprint (continued from page 50)

```
 1450 R$(X%)="":C$(X%)="":NEXT
 1460 ENDPROC
 1470 :
 1480 DEF FNswap(S$,M%)
 1490 IF S$="" =""
 1500 LOCAL A%,F%,H%,I%,L%:L%=LEN(S$):A$
="":
 1510 FOR I%=1 TO L%
 1520 H%=ASC(MID$(S$,I%,1))
 1530 IF H%<65 OR (H%>90 AND H%<97) OR H
%>122 THEN 1570
 1540 IF (H%>=65 AND H%<=90) F%=FALSE
 1550 IF (H%>=97 AND H%<=122) F%=TRUE
 1560 IF NOT(F%) AND (M%=0 OR M%=2) H%=H
%+32 ELSE IF F% AND (M%=1 OR M%=2) H%=H%
-32
 1570 A$=A$+CHR$H%
 1580 NEXT I%
 1590 =A$
```

```
 1600 :
 1610 DEF PROCprint_codes
 1620 VDU2
 1630 REM put print codes here **
 1640 VDU3
 1650 ENDPROC
 1660 :
 1670 DEF PROCtitle
 1680 PRINTTAB(29,0)"S T O R E - P R I N
T"'STRING$(80,"_")
 1690 ENDPROC
 1700 :
 1710 ON ERROR OFF
 1720 IF ERR=17 THEN GOTO280
 1730 PRINT"E R R O R"'':REPORT:PRINT" a
t line ";ERL
 1740 CLOSE#0
 1750 END
```

BEEBUG

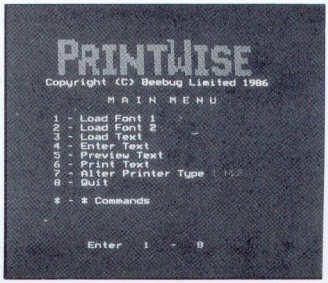
Software for the BBC Micro and Master Series

Beebug C Programming Language

Beebug C is the much acclaimed C programming language for the BBC Micro and Master 128. Although normally only available on more powerful computers, Beebug C is a full implementation of the Kernighan & Ritchie standard. Features include:

★ Runs on BBC B, B+ and M128.
★ Comprehensive set of ANSI functions.
★ OS functions vdu, osbyte, mode etc.
★ Command line interpreter.
★ Floating-point maths.
★ Linker for multi-source programs.
★ Expandable run-time library.
★ Optional stand alone generator.
★ Optional maths functions.
★ Full macro handling facilities.
★ Supplied on 2 ROMs & library disc

Normal price: ***£60.28*** inc VAT
Members price: ***£45.21*** inc VAT
Stock code: 0074 40T, 0075 80T
Please add £2.00 carriage.



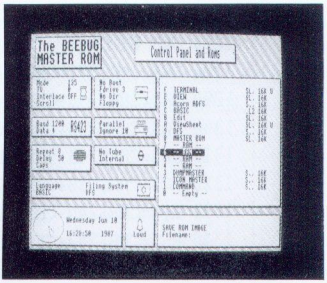
Printwise

Printwise is a low-cost publishing aid allowing you to create professional looking magazines, leaflets, posters etc - the possibilities are endless. Simply take your text file, use embedded commands to specify the font styles you require, and let Printwise do the rest.

★ 9 authentic fonts from 4pt to 40pt.
★ Font designer for creating new fonts.
★ Fonts may be used on the same line.
★ Italics, bold, condensed, reversed etc.
★ Proportional spacing.
★ Subscript and superscript.
★ Left, centre, right & full justification.
★ Suitable for Epson compatible printers.

Printwise is easy to use and requires no programming skills. It may be used with text files created on Wordwise, View, InterWord, Mini Office and almost any text editor.

Normal price: ***£30.66*** inc VAT
Members price: ***£22.99*** inc VAT
Stock code: 0085 40T, 0086 80T
Please add £2.00 carriage.



Master ROM

The Master ROM is a powerful 32K ROM packed with features to enhance the facilities of the Master 128.

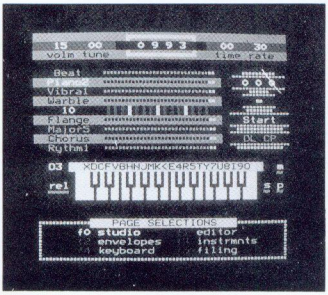
Disc Menu - A single command takes you to a full feature disc menu displaying all the items in the current directory. You can change directories or run, copy, delete, rename selected files.

Control panel - This displays all the computers status settings and the ROMs fitted. The cursor keys may be used to adjust any of the settings, which may be saved for future loading at any time.

Disc commands - A whole range of useful ADFS commands, including: *FIND, *FORMAT, *VERIFY, *BACKUP, *MERGE, *WIPE etc.

Plus: **16K-64K Printer buffer**
Simple 16K-64K RAM disc
Diary and automatic alarm

Normal price: ***£39.84*** inc VAT
Members price: ***£29.88*** inc VAT
Stock code: 0087 ROM
Please add £2.00 carriage.



Studio 8

Studio 8 is a real-time studio system with digital recorder, rhythm and drum machines which give hours of entertainment.

Studio - Allows playing and recording in real time with keyboard and sequencer.

Editor - A full-screen editor allowing the precise editing of notes.

Envelope editor - allows the definition of up to 16 amplitude and pitch envelopes.

Instrument definer - Create up to 32 instruments by combining pitch and amplitude envelopes, volume & sustain.

Plus many more features.

Normal price: ***£22.48*** inc VAT
Members price: ***£16.86*** inc VAT
Stock code: 0009 80T
Please add £2.00 carriage.

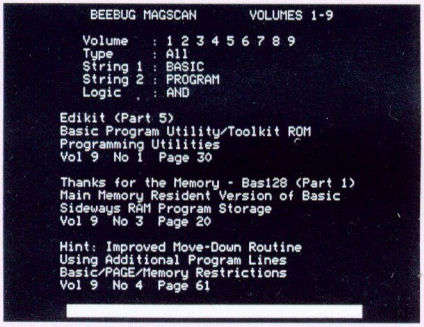
```
 1220 VDU23,242,0,130,254,30,132,252,24,
8
 1230 VDU23,243,255,255,255,0,0,0,0,0
 1240 VDU23,244,0,0,0,0,0,255,255,255
 1250 ENDPROC
 1260 :
 1270 DEF PROCsym
 1280 SH$=CHR$231+CHR$8+CHR$10+CHR$232
 1290 NA$=CHR$233+CHR$8+CHR$10+CHR$234
 1300 FL$=CHR$235+CHR$10+CHR$8+CHR$236
 1310 RW$=CHR$243:RM$=CHR$244
 1320 RC$=CHR$238+CHR$10+CHR$8+CHR$239
 1330 RQ$=CHR$240+CHR$10+CHR$8+CHR$241
 1340 RS$=CHR$242+CHR$10+CHR$8+CHR$241
 1350 WW$=CHR$226+CHR$230
 1360 BB$=CHR$224+CHR$225
 1370 DT$="."
 1380 ENDPROC
 1390 :
 1400 DEF PROCstave(SS)
 1410 VDU5:FOR S=0 TO SS
 1420 FOR L=0 TO 4
 1430 MOVE0,132+32*L+288*S:PLOT21,1276,1
32+32*L+288*S
 1440 NEXT L,S
 1450 FOR LiNo=-4 TO 14 STEP 2
 1460 Ypos=124+16*LiNo:Yval$=FNjust(LiNo
)
 1470 MOVE1220,Ypos:PRINT Yval$
 1480 IF Rflag MOVE1168,Ypos+8:PLOT21,12
08,Ypos+8
 1490 NEXT LiNo
 1500 ENDPROC
 1510 :
 1520 DEF FNjust(n):Rflag=FALSE
 1530 IF n>=0 AND n<10 n$="0"+STR$(n) EL
SE n$=STR$(n):IF n<13 Rflag=TRUE
 1540 =n$
 1550 :
 1560 DEF FNpos
 1570 IF J$="R" OR J$="B" OR K$="CS" YP=
0:=TRUE
 1580 Y$=GET$:IF Y$="C" =FALSE
 1590 IF INSTR("-10",Y$)=0 GOTO 1580
 1600 IF Y$=";" Y$="+"
 1610 Y1=GET-48-10*(Y$="+")
 1620 Y2$=Y$+STR$(Y1)
 1630 YP=VAL(Y2$):YY=132+16*YP
 1640 VDU4:PRINTTAB(3,30);Y2$
 1650 =TRUE
 1660 :
 1670 DEF PROCchoosesym
 1680 VDU4:PRINTTAB(0,30)"Enter symbol a
nd position"
 1690 J$=GET$:PRINTTAB(0,30)SPC26
 1700 IF J$="P" PROCprint:ENDPROC
 1710 IF J$="E" PROCerase:ENDPROC
 1720 I$=GET$:K$=J$+I$
 1730 IF K$="CS" PROCchst:ENDPROC
 1740 IF K$="CB" PROCbar:ENDPROC
 1750 IF K$="BQ" OR K$="BS" PROCbar:ENDP
ROC
 1760 VDU4:PRINTTAB(0,30)K$;";"
 1770 IF NOT FNpos ENDPROC
 1780 VDU4:PRINTTAB(6,30)"C to change: a
ny key to display";:Q=GET
 1790 PRINTTAB(0,30)SPC38;
 1800 IF Q<>67 PROCif
 1810 ENDPROC
 1820 :
 1830 DEF PROCif:VDU5:YA=228
 1840 FOR CL=1 TO 2
 1850 IF K$="BQ" OR K$="BS" PROCbar:ENDP
ROC
 1860 IF K$="SH" MOVE XX,YY+16:PRINT SH$
 1870 IF K$="NA" MOVE XX,YY+16:PRINT NA$
 1880 IF K$="FL" MOVE XX,YY+16:PRINT FL$
 1890 IF K$="RW" MOVE XX+2*SP,YA:PRINT R
W$:XX=XX+2*SP
 1900 IF K$="RM" MOVE XX,YA:PRINT RM$
 1910 IF K$="RC" MOVE XX,YA:PRINT RC$
 1920 IF K$="RQ" MOVE XX,YA:PRINT RQ$
 1930 IF K$="RS" MOVE XX,YA:PRINT RS$
 1940 IF K$="BL" MOVE XX+2*SP,YA+32:DRAW
 XX+2*SP,YA-96
 1950 IF K$="WW" MOVE XX+4*SP,YY:PRINT W
W$:LS=XX+3*SP:LF=XX+11*SP
 1960 IF (KR$="QU" OR KR$="QD" OR KR$="S
U" OR KR$="SD") XX=XX+5*SP:KR$=CHR$32
 1970 IF K$="MU" MOVE XX+2*SP,YY:PRINT W
W$:MOVE XX+7.5*SP,YY:DRAW XX+7.5*SP,YY+8
4:LS=XX+SP:LF=XX+9*SP
```

```
 1980 IF K$="DT" MOVE XX-8,YY+16:PRINT D
T$
 1990 IF K$="MD" MOVE XX+2*SP,YY:PRINTWW
$:MOVE XX+2*SP,YY:DRAW XX+2*SP,YY-104:LS
=XX+SP:LF=XX+9*SP
 2000 IF K$="CU" AND CL=1 XST2=XST1:YST2
=YST1:XST1=XX+5*SP:YST1=YY+78
 2010 IF K$="CU" MOVE XX,YY:PRINT BB$:MO
VE XX+5*SP,YY-20:DRAW XX+5*SP,YY+78:LS=X
X-SP:LF=XX+6*SP
 2020 IF K$="CD" AND CL=1 XST2=XST1:YST2
=YST1:XST1=XX:YST1=YY-102
 2030 IF K$="CD" MOVE XX,YY:PRINT BB$:MO
VE XX,YY-24:DRAW XX,YY-102:LS=XX-SP:LF=X
X+6*SP
 2040 IF K$="QU" OR K$="SU" MOVE XX,YY:P
RINT BB$:MOVE XX+5*SP,YY-20:DRAW XX+5*SP
,YY+78:DRAW XX+8.5*SP,YY+50:IF K$="SU" M
OVE XX+6*SP,YY+50:DRAW XX+8.5*SP,YY+30
 2050 IF K$="QU" OR K$="SU" LS=XX-SP:LF=
XX+6*SP
 2060 IF K$="QD" OR K$="SD" MOVE XX+3*SP
,YY:PRINT BB$:MOVE XX+3*SP,YY-24:DRAW XX
+3*SP,YY-102:DRAW XX+6.5*SP,YY-74:IF K$=
"SD" MOVE XX+3*SP,YY-82:DRAW XX+6.5*SP,Y
Y-54:LS=XX+2*SP
 2070 IF K$="QD" OR K$="SD" LS=XX+2*SP:L
F=XX+9*SP
 2080 YY=YY+STNo*STS+16*UD
 2090 YA=YA+STNo*STS
 2100 NEXT CL
 2110 IF YP<0 OR YP>10 OR (YP+UD)<0 OR (
YP+UD)>10 PROCleger
 2120 IF K$="CU" XX=XX+1.5*SP
 2130 IF K$="WW" XX=XX+9*SP
 2140 IF K$="MU" OR K$="MD" XX=XX+4*SP
 2150 IF K$="BL" XX=XX+5*SP ELSE XX=XX+7
*SP
 2160 KR$=K$:ENDPROC
 2170 :
 2180 DEF PROCleger
 2190 JL=STNo*STS
 2200 YP2=YP+UD
 2210 IF YP>10 MOVE LS,292:DRAW LF,292:I
F YP>12 MOVE LS,324:DRAW LF,324:IF YP>14
 MOVE LS,356:DRAW LF,356
 2220 IF YP<0 MOVE LS,100:DRAW LF,100:IF
 YP<-2 MOVE LS,68:DRAW LF,68
 2230 YP2=YP+UD
 2240 IF YP2>10 MOVE LS,292+JL:DRAW LF,2
92+JL:IF YP2>12 MOVE LS,324+JL:DRAW LF,3
24+JL:IF YP2>14 MOVE LS,356+JL:DRAW LF,3
56+JL
 2250 IF YP2<0 MOVE LS,100+JL:DRAW LF,10
0+JL:IF YP2<-2 MOVE LS,68+JL:DRAW LF,68+
JL
 2260 ENDPROC
 2270 :
 2280 DEF PROCchst
 2290 GCOL0,0
 2300 IF STNo=1 THEN STNo=2 ELSE STNo=1
 2310 Ypos=704*STNo-352
 2320 VDU24,0;0;1279;Ypos;:CLG:VDU26
 2330 JU=STNo*STS+16*UD
 2340 GCOL0,1:XX=0
 2350 PROCstave(STNo)
 2360 ENDPROC
 2370 :
 2380 DEF PROCbar
 2390 IF XST2=0 ENDPROC
 2400 MOVE XST1,YST1:DRAW XST2,YST2
 2410 MOVE XST1,YST1+JU:DRAW XST2,YST2+J
U
 2420 IF K$="BQ" GOTO 2450
 2430 MOVE XST1,YST1+12:DRAW XST2,YST2+1
2
 2440 MOVE XST1,YST1+12+JU:DRAW XST2,YST
2+12+JU
 2450 XST1=0:YST1=0:XST2=0:YST2=0
 2460 ENDPROC
 2470 :
 2480 DEF PROCprint
 2490 VDU2:*GDUMP 0 1 2 1
 2500 ENDPROC
 2510 :
 2520 DEF PROCerase
 2530 GCOL0,0:MOVE XX,340:DRAW XX-SP,340
:PLOT85,XX,0:PLOT85,XX-SP,0
 2540 GCOL0,0:MOVE XX,340+JU:DRAW XX-SP,
340+JU:PLOT85,XX,JU:PLOT85,XX-SP,JU
 2550 GCOL0,1:XX=XX-SP:PROCstave(2)
 2560 ENDPROC
```

Magscan

Comprehensive Magazine Database for the BBC Micro and the Master 128

An updated version of Magscan, which contains the complete indexes to all BEEBUG magazines from Volume 1 Issue 1 to Volume 10 Issue 5



Magscan with disc and manual **£9.95**+p&p

Stock codes: 0005a 5.25"disc 40 track DFS
0006a 5.25"disc 80 track DFS
1457a 3.5" ADFS disc

Magscan update **£4.75** +p&p

Stock codes: 0011a 5.25"disc 40 track DFS
0010a 5.25"disc 80 track DFS
1458a 3.5" ADFS disc

Magscan allows you to locate instantly all references to any chosen subject mentioned anywhere in the 95 issues of BEEBUG magazine to date.

Just type in one or two descriptive words (using AND/OR logic), and you can find any article or program you need, together with a brief description and reference to the volume, issue and page numbers. You can also perform a search by article type and/or volume number.

The Magscan database can be easily updated to include future magazines. Annual updates are available from BEEBUG for existing Magscan users.

Some of the features Magscan offers include:

- full access to all BEEBUG magazines
- rapid keyword search
- flexible search by volume number, article type and up to two keywords
- keyword entry with selectable AND/OR logic
- extensive on-screen help
- hard copy option
- easily updatable to include future magazines
- yearly updates available from BEEBUG

Special Offers to BEEBUG Members April 1992

1407a	ASTAAD3 - 5" Disc (DFS)	5.95
1408a	ASTAAD3 - 3.5" Disc (ADFS)	5.95
1404a	Beebug Applics I - 5" Disc	4.00
1409a	Beebug Applics I - 3.5"Disc	4.00
1411a	Beebug Applics II - 5" Disc	4.00
1412a	Beebug Applics II - 3.5" Disc	4.00
1405a	Beebug Utilities - 5" Disc	4.00
1413a	Beebug Utilities - 3.5" Disc	4.00
1450a	EdiKit 40/80 Track	5.75
1451a	EdiKit EPROM	7.75
1452a	EdiKit 3.5"	5.75
0005b	Magscan Vol.1 - 8 40 Track	9.95
0006b	Magscan Vol.1 - 8 80 Track	9.95
1457b	Magscan Vol.1 - 8 3.5" ADFS	9.95
0011a	Magscan Update 40 track	4.75
0010a	Magscan Update 80 track	4.75
1458a	Magscan Update 3.5" ADFS	4.75
PAG1a	Arcade Games (5.25" 40/80T)	5.95
PAG2a	Arcade Games (3.5")	5.95
PBG1a	Board Games (5.25" 40/80T)	5.95
PBG2a	Board Games (3.5")	5.95
1600a	Beebug magazine disc	4.75
0077b	C - Stand Alone Generator	14.56
0081b	Masterfile ADFS M128 80 T	16.86
0024b	Masterfile DFS 40 T	16.86
0025b	Masterfile DFS 80 T	16.86
0074b	Beebug C 40 Track	45.21
0075b	Beebug C 80 Track	45.21
0084b	Command	29.88
0073b	Command(Hayes compatible)	29.88
0053b	Dumpmaster II	23.76
0004b	Exmon II	24.52
0087b	Master ROM	29.88
1421b	Beebug Binder	4.20

Have you got your BEEBUG Binder for Volume 10?

Only £4.20

RISC Developments Ltd, 117 Hatfield Road, St Albans, Herts AL1 4JS. Tel (0727) 40303 Fax (0727) 860263

HINTS and tips HINTS and tips HINTS and tips HINTS and tips HINTS and tips

Please keep sending in your hints and tips, and don't forget we pay for all those we publish.

PROGRAMMING THE f-KEYS FOR MODE 7

Brian E. Lowe

In the Dec 1991 (Vol.10 No.7) issue of Beebug, D.N.Atkinson showed how to program the red function keys for use in the Teletext Mode. The following program offers an alternative:

```
100 *FX18
110 *KEY8"|!|L"
120 *KEY9 "|!|M"
130 *FX226,128
140 *FX227,144
150 *FX228,154
```

This program doesn't have the single-key press advantage for text colours, but instead, the alpha/graphic colours, steady and flash are available with their original key combinations as shown in the User Guide. The following additional key combinations are also provided:

f8 - normal height
f9 - double height
Ctrl-f8 - conceal display
Ctrl-f9 - contiguous graphics
Shift-Ctrl-f0 - separated graphics
Shift-Ctrl-f2 - black background
Shift-Ctrl-f3 - new background
Shift-Ctrl-f4 - hold graphics
Shift-Ctrl-f5 - release graphics
Shift-Ctrl-f6 - filled block (CHR$160)
Shift-Ctrl-f8 - quotation mark (CHR$162)

The last two key combinations were not previously available directly from the keyboard. The normal quotation mark character, obtained by Shift-2, has character code 34. This cannot be used for display purposes because it is used for starting and ending a string in a PRINT statement. Function keys f0/f7 are free for programming by the user in the normal way.

NEXT BEST THING

John Tupper

There is an undocumented way of terminating multiple FOR-NEXT loops that doesn't seem to be documented anywhere. You can replace the statements NEXT:NEXT by NEXT, to terminate two loops at the same time, and NEXT:NEXT:NEXT can be replaced by NEXT,, and so on. I personally feel that it is bad practice not to write out NEXT:NEXT in full, particularly in BEEBUG magazine where many members still enjoy typing in the monthly listings, because omitting the comma will cause an error that is hard to track down. However, this method can be very useful in short programs or when typing at the Basic prompt. (Ed: We first published this hint in BEEBUG Vol.1 No.8, Dec 1982, page 16!)

HOW DO YOU SPELL ADFS

Denis Atkinson

The BEEBUG program PreView (Vol.9 No.9) presented me with a new challenge - to apply the ViewSpell checker to a file saved to an ADFS disc. The ViewSpell instruction booklet touches on this briefly at the foot of page 13, but for some time I was unable to persuade ViewSpell to work. Eventually I succeeded with the following method: put the ADFS disc into drive 0 and *MOUNT it; put the ViewSpell disc into drive 1, change to DFS (by *DISC) and select drive 1 (by *DRIVE 1); then type the following:

```
*SPELL
PREFIX T -ADFS-:0.$.
PREFIX M ;1.
LOAD $.filename
```

where *filename* is the name of the file to be checked. While on the subject of ViewSpell and ADFS, note that ViewSpell limits the prefix to 13 characters and the filename to 10 characters. If, therefore, the file is in a subdirectory, then when ViewSpell loads it or sets the prefix, it may generate a "Too Long" error message. In such a case it is necessary to transfer the file to the root directory $.

RISC USER

The Archimedes Magazine & Support Group

RISC User continues to enjoy the largest circulation of any subscription magazine devoted solely to the Acorn Archimedes range of computers. Its in-depth, authorative approach appeals to all users, whatever their interest and level of expertise, and the lively mixture of articles, programs, reviews and news is carefully selected to cater for all Archimedes owners from beginner to expert.

Existing BEEBUG members, who want to find out more about the Archimedes range, may either transfer their existing subscription to RISC User (at no extra charge), or extend their subscription to include both magazines.

A joint subscription will enable you to keep completely up-to-date with all innovations, and the latest information from Acorn and other developers for both the BBC micro and the Archimedes range. RISC User is the magazine for enthusiasts and professionals at all levels.

The Archimedes Magazine & Support Group

Here are some articles and series published in the most recent issues of RISC User:

ALEPH ONE PC BOARD
A review of the first PC card for the Archimedes.

FLIGHT SIM TOOLKIT
A review of this package from SIMIS, the creators of all jet fighter simulations on the Arc, which enables you to design your own flight simulation world.

COLOUR PRINTING WITH THE DESKJET 500C
A look at what is being heralded as a revolution in low cost colour printing.

SQUIRREL
A review of this powerful database management system from Digital Services.

CREATING DRAW FILES IN BASIC
A two-part arfticle on how to use Basic to create Draw files.

ARMED AND READY FOR BATTLE
A look into the latest developments and the future of the ARM processor.

EXPLOITING THE WIMP
A series on some practical aspects of WIMP programming.

QUICK ON THE DRAW
A comparison review of three packages which aim to create Draw files from sprites.

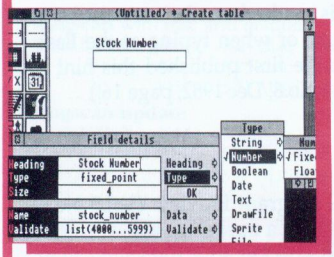
EXPLORING UNIX ON THE ARC
A look at this internationally accepted operating system which can be run on an Archimedes.

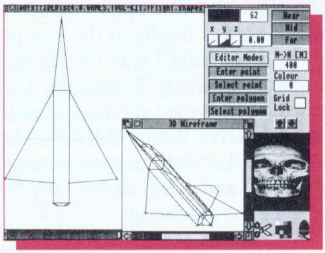
DESKTOP ANIMATOR
A tutorial mini-series on how to program moving images within a Desktop window.

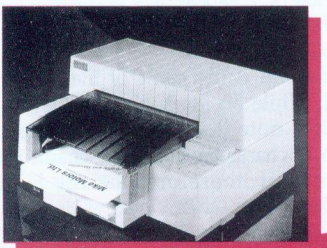
USING ANSI C
A series of articles on programming Desktop applications in C.

WP/DTP
A regular column on using different DTP and WP packages.

INTO THE ARC
A regular series for beginners currently treating the subject of using shared resources.







SUBSCRIPTION DETAILS

As a member of BEEBUG you may extend your subscription to include RISC User for only:

Destination	*Additional Cost*
UK,BFPO &Ch Is	£ 10.50
Rest of Europe and Eire	£ 15.40
Middle East	£ 19.60
Americas an d Africa	£ 21.90
Elsewhere	£ 33.00

UPDATING THE DISC ORGANISER

I find the *Disc Organiser for ADFS* (BEEBUG Vol.10 Nos.7 & 8) to be perhaps the most useful utility you've published in the 10 years I have been a member of BEEBUG.

However, I should like to offer a few enhancements. In order to prevent file names overwriting the control bar at the base of the screen, if there are more than 20 files in a directory, change as follows:

Lines 1910 & 2150 - change 22 to 20
Line 2530 - change PRINTTAB(0,24) to PRINTTAB(0,23)
Line 2390 - change VDU24 to VDU22 (twice).

One further enhancement is to add:

```
4925 PROCunlock
4945 PROCrelock
```

This allows you to move a file without having to access it first to change its attributes (i.e. to unlock it), and it allows you to move the file to another directory/disc even if it already exists in that directory/disc. After having moved the file, it automatically deletes the original file and relocks all files in that directory. The definitions of the two procedures are given below.

```
5650:
5660 DEF PROCunlock
5670 drivefrom$=MID$(from$,2,1)
5680 driveto$=MID$(to$,2,1)
5690 OSCLI("MOUNT "+drivefrom$)
5700 IF side$="left" THEN OSCLI("DIR "+
ldir$) ELSE OSCLI("DIR "+rdir$)
5710 OSCLI("ACCESS "+name$+" WR")
5720 OSCLI("MOUNT "+driveto$)
5730 IF side$="left" THEN OSCLI("DIR "+
rdir$) ELSE OSCLI("DIR "+ldir$)
5740 IF filecount%=0 THEN ENDPROC
5750 OSCLI("ACCESS * WR")
5760 ENDPROC
5770:
5780 DEF PROCrelock
5790 driveto$=MID$(to$,2,1)
5800 OSCLI("MOUNT "+driveto$)
5810 IF side$="left" THEN OSCLI("DIR "+
rdir$) ELSE OSCLI("DIR "+ldir$)
5820 OSCLI("Access * WR")
5830 ENDPROC
```

Ian Crawford

Note the use of lower case 'L' in the above (ldir).

PC DISCS ON A BEEB WITH WATFORD DDFS

The programs presented in BEEBUG Vol.10 No.4 do not work with the original version of the Watford DDFS disc controller, although this does use the 1770 chip. Limited but practical operation has been found possible by making the following changes. They apply to the original Watford DDFS controller with single 80 track drive on a model B.

MS-DOS TO BBC TRANSFER

```
140 D=0
1510 val=rst
1680 wd=&FE84:ctrl=&FE80:sel=2:dden=8:
rst=&20
1710 ?ctrl=rst:ENDPROC
```

BBC TO MS-DOS TRANSFER

```
160 D=0
1810 val=rst
1960 wd=&FE84:ctrl=&FE80:sel=2:dden=8:
rst=&20
1990 ?ctrl=rst:ENDPROC
```

PC FORMATTING

```
255 ENDPROC:REM Disables drive selecti
on
760 IF M%=251 OR M%<1 cmd%=&FE84:ctrl%
= &FE80:M%=FALSE:fc%(0)=32:fc%(1)=34
```

Can anyone add to these to cover multiple drives or the Mk 2 controller?

P.J.Lawrence

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 40p per word (inclusive of VAT) and these will be featured separately. Please send us all ads (personal and business) to MEMBERS' ADS, BEEBUG, 117 Hatfield Road, St. Albans, Herts AL1 4JS. The normal copy date for receipt of all ads will be the 5th of each month.

3D snooker, Tarzan and Hypersports for BBC B only £5 for all 3. PRES Advanced File Manager and Advanced Control Panel £20 each or £35 for both. Acornsoft Overview cartridge (includes Viewstore, Viewspell and rest of View family) £35, CC Accelerator (2 ROMs) £20. Bargain-buy the lot for £75. Tel. 061-226 1802 after 5pm.

BBC B with Watford 32k Shadow RAM board, Watford 128k ROM/RAM board (16k battery backed) and a Challenger 3 256k RAM disc and 40/80 D/D D/S disc drive, amongst the other items are an AMX mouse, Superart, Stop Press and Replay ROMs, a spare Power Supply unit and many manuals £350 o.n.o. Tel. (0532) 755756.

BBC issue 4, DFS, ATPL board with 16k sideways RAM, Watford shadow RAM board, View 3 + printer driver disc, Viewsheet, Viewstore + Dabs guide to Viewsheet and Viewstore, Viewchart disc, Viewspell, Hyperdrive ROM and disc and manual, excellent condition, sensible offers around £185. Ideal for word processing, in any mode, to include Watford shadow RAM manual and View suite manuals. Dual 40/80T disc drive in piggy back case with mains PSU £120. Tel. 071-494 1365 office hours only.

Econet Filestore E20 + 4 Master E.T. terminals. Offers? Tel. (0584) 872846 day or (0568) 85410 eves.

M128, 512 co-processor, Acorn reference manuals, Dabs 512 books and discs, Essential Software mouse, dual disc drives, Taxan amber monitor, case cartridges, various ROMs inc. 'B' ROMs, various Beeb books, magazines and BEEBUG discs, GIS teletext adaptor, ATPL ROM/RAM board, must go! £650 or might split. Tel. (0789) 266946.

WANTED: For BBC Master Compact, Mertec compact companion or a PearTree HC1 2MHz Bus interface, Morley teletext adaptor, RS232 interface IC's. Tel. (0452) 830146 eves or 305752 day.

BBC 512 co-processor in Watford co-pro adaptor, (DOS 2.1) complete with manuals, Gem software and mouse, Dabhand User and Technical guides with discs, Essential Softwares RAMDISC and TRNSLATE, Shibumi Problem Solver, Shareware etc. £120. Tel. 081-368 5535.

Competition Pro joystick £5, Voltmace twin joysticks £11, Care (Master) 4 way cartridge £7.50, 5 (27128-16k) EPROMs £10, Tapes of Knitwear Design £2.50, Mastermind £2.50, Mini Office II £3.50. Tel. (0752) 896077.

ATPL Sideways ROM board £15, BEEBUG Toolkit plus ROM £10, both with original instructions. Tel. (0628) 819394.

Volumes 3 to 8.5 of BEEBUG, perfectly clean, offers? Tel. (0923) 823659.

A3000 with Philips colour monitor 2Mb memory expansion includes monitor stand, all leads and manuals £600. Tel. (0865) 864182.

Archimedes A310, Philips 8833 colour monitor, manuals, boxes, some software £650. Tel. (0792) 404331 after 6pm.

Taxan Kaga KP810 DM printer, internal RAM chips plus Master cartridge and NLQ designer ROM and over 20 fonts for downloading to printer £75, EXMON II ROM £12, Play it again Sam 1 £5, Repton 3 £5, AMX mouse £5, Acorn Tape machine £5, bundle of 7 good BBC B tapes £10, best offers considered. Tel. (0283) 31403 anytime.

JUKI 6100 daisywheel printer many fonts £95, BBC Watford Video Digitiser with ROM £45, Z80 2nd Processor with CP/M software £80, Centronics 739 dot

matrix printer with Dump ROM £45, AMX mouse £15, Light Pen with Pen Pal software £9, 11x1987 Acorn User magazines (August missing) £11, 12 x1986/7 Micro User magazines in binder £12. Tel. (0252) 510486.

M128 with upgrade OS chip, twin Cumana 40/80T switchable DD with PSU, Master internal modem 14" colour TV/RGB monitor, Smart cartridge, Dumpmaster II (ROM), plus lots of games/utilities and reference books etc. £500. Tel. (0895) 672132 after 7.30pm.

A310M, DTP software, PC Emulator (1.6), Render Bender, Genesis and games, all for £540, will split, Old PRMs £9, various computer magazines, EMR sound sampler £60, Watford video digitiser £75, Acorn colour monitor £110. Tel. 061-973 0529 after 6pm.

M512, Acorn amber monitor, dual 5.25" & 3.5" drives, 6502 second processor, Bitstik, Citizen 120D printer £450. Tel. 081-698 3772.

M512, 40/80T plinthed DD, cartridge, mouse, joysticks, GEM software and books £500 + carriage. Philips TTL/RGB colour monitor £200 + carriage. Tel. 010 468 767 9295 daytime (Note: this is an overseas number).

BBC computer software and hardware, Viewsheet ROM, manuals £9, Interword ROM, manuals £10, Solidisc 2Mb 128k board and disc £20, Solidisc 128 sideways RAM & 13 discs £25. Tel. (0726) 814488.

Archimedes 440, 53Mb hard disc, Taxan 770+, multisync monitor, loads of software. Offers? Tel. 081-445 7875.

Epson LX800 printer with manuals, original packaging and 2 spare ribbons £100, Acorn Desktop Assembler SKB76 new unused £100. Tel. (0935) 77581 eves.

Archimedes A310, Philips 8833 colour monitor, manuals, boxes, some software £650. Tel. (0792) 404331. After 6pm.

BBC B 512k, 2nd processor and Watford co-processor box £100 o.n.o. Tel. (0277) 821620 anytime.

WANTED: Acorn ISO-Pascal and Acorn Micro-Prolog with documentation for a BBC B computer. Tel. (0705) 733281.

Has anyone use for several PETS and Apple IIe computers including manuals and software in working order? Tel. 081-395 2346.

Master Compact (no monitor) including manual in good condition £150. Tel. 081-395 2346.

BBC B issue 4 fitted with DFS and Econet £45. Tel. 081-318 5155.

I have a Penman plotter that has ceased to function, due, I believe to a problem with one of its location devices and its associated diffraction grating. I need someone to repair it for me since the company that made it has ceased to exist. Tel. (0526) 22381.

If you are interested in either IBM or Archimedes software and programming (particularly in Basic) or even in swopping ideas for programs. Please write to me at East House, Michaelhouse, Balgowan, 3275, South Africa.

A3000 with colour monitor as new £625, RICOH daisywheel printer as new £75, A3000 carrying case £15, Parallel printer cable £5, BEEBDOS £20, BEEBPC £20. **WANTED:** A3000 Disc Buffer and/or RAM upgrade. Tel. (0483) 480632.

WANTED: Official manual for BBC LISP, "LISP on the BBC Microcomputer" by Norman and Cattell. Tel. (0829) 270176 eves.

Archimedes A420/1, colour monitor, 2Mb RAM and 20Mb St502 hard drive, only 16 months old, very good condition, Panasonic KXP-1124 pin printer, will throw in some games, dust covers and other extras! The whole lot for £1,200! Ideal set-up for word processing. Tel. (0707) 323032 eves.

A3000 with 2Mb RAM upgrade, Philips CM8833 stereo colour monitor, Pres: monitor stand, system housing, 5.25" disc drive, disc buffer, 65Host DFS, podule case, latest PC emulator, covers, all manuals etc. As new, the perfect educational system £875. Tel. (0444) 454348 eves.

Potential purchasers are advised to insist that it is the seller's responsibility to ensure that goods arrive with the purchaser in a fit state as described. In the event of damage in transit, contact the seller before taking further action.

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

BEEBUG		BEEBUG & RISC USER
£18.40	1 year (10 issues) UK, BFPO, Ch.I	£28.90
£27.50	Rest of Europe & Eire	£42.90
£33.50	Middle East	£53.10
£36.50	Americas & Africa	£58.40
£39.50	Elsewhere	£62.50

BACK ISSUE PRICES (per issue) 1 July 1991

Volume	Magazine	5"Disc	3.5"Disc
6	£1.00	£3.00	£3.00
7	£1.10	£3.50	£3.50
8	£1.30	£4.00	£4.00
9	£1.60	£4.75	£4.75
10	£1.90	£4.75	£4.75
Binders	£4.20		

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.

POST AND PACKING

Please add the cost of p&p when ordering individual items. See table opposite.

Destination	First Item	Second Item
UK, BFPO + Ch.I	£1.00	£0.50
Europe + Eire	£1.60	£0.80
Elsewhere	£2.40	£1.20

BEEBUG
117 Hatfield Road, St.Albans, Herts AL1 4JS
Tel. St.Albans (0727) 40303, FAX: (0727) 860263
Manned Mon-Fri 9am-5pm (for orders only 9am-6pm and 9am-5pm Saturdays)
(24hr Answerphone for Connect/Access/Visa orders and subscriptions)

BEEBUG MAGAZINE is produced by RISC Developments Ltd.

Editor: Mike Williams
Assistant Editor: Kristina Lucas
Technical Assistant: Mark Moxon
Editorial Assistance: Marshall Anderson
Production Assistant: Sheila Stoneman
Advertising: Sarah Shrive
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 Arlon Printers (0923) 268328 ISSN - 0263 - 7561

Magazine Disc

April 1992
DISC CONTENTS

THE HIDDEN PERSUADERS - a couple of short programs which help to reveal some secret information hidden in your Master ROMs.

MUSICAL MUSKRATS - an enhanced version of the Mustran program, published in Vol.6 No.8, which is a valuable aid for producing and printing musical scores.

RAY TRACING IN 2D (Part 2) the concluding part of this 2D ray tracing package, which traces rays through the shapes created in part 1.

WORDWISE USER'S NOTEBOOK - an example program which demonstrates how batches of data may be supplied in segment programs and the data items read and processed as they are required.

CROSS REFERENCE LISTER - a very useful utility which provides a cross reference listing of a Basic program - a must for the serious developer of Basic software.

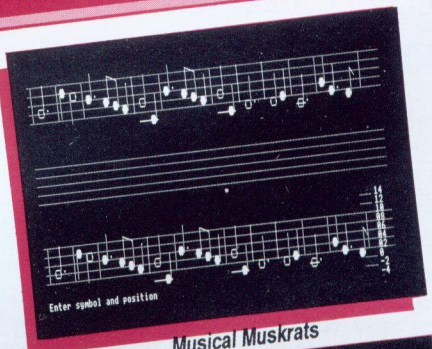
ZEUS II - THE FINAL CONFLICT (Part 2) - this month's two programs add more levels to the mind-bending puzzle game and provide an editor to create your own levels.

BEEBUG WORKSHOP: FINDING A ROUTE IN A NETWORK - five programs providing the routines and the data for resolving the 'shortest route' problem.

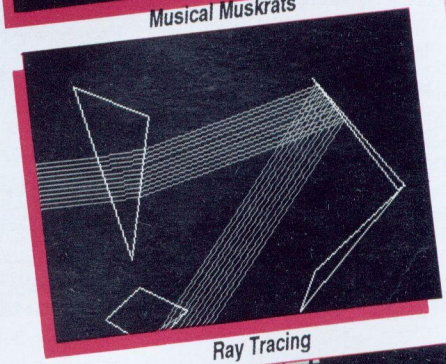
STOREPRINT - a program which helps you print out ViewStore files directly.

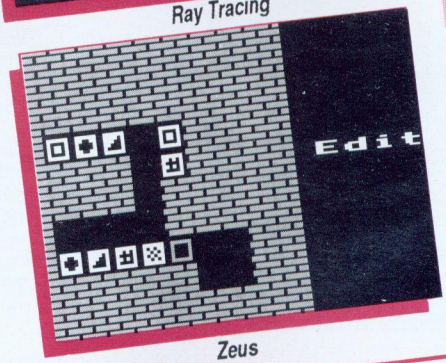
FUNCTION/PROCEDURE LIBRARY (10) - a new instalment of 5 routines some of which are needed when printing.

MAGSCAN DATA - bibliography for this issue.



Musical Muskrats


Ray Tracing



Zeus

ALL THIS FOR £4.75 (5.25" & 3.5" DISC) + £1 P&P (50P FOR EACH ADDITIONAL ITEM)

Back issues (5.25" and 3.5" discs from Vol.5 No.1) available at the same prices.

DISC (5.25" or 3.5") SUBSCRIPTION RATES	UK ONLY	OVERSEAS
6 months (5 issues)	£25.50	£30.00
12 months (10 issues)	£50.00	£56.00

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

RISC Developments, 117 Hatfield Road, St.Albans, Herts AL1 4JS

BEEBUG

The Archimedes Specialists

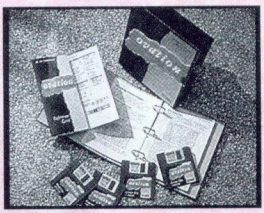
A3000 Hard Drive DTP System

If you have been thinking of getting an A3000 there has never been a better time.

This special offer provides an excellent system ready for immediate use. The hard drive, RAM and Ovation are all installed ready so you can simply turn on and start.

Ovation is the highly acclaimed package combining word processing and DTP. Widely used in education it offers a whole host of features and is powerful yet simple to use.

Our high speed IDE unit was designed especially for the A3000. It has an access time faster than SD506 or 8 bit SCSI, features auto-parking and sleep mode and is fitted in the internal expansion slot.



- Acorn A3000 Computer
- Genuine Acorn Colour Monitor
- Monitor Plinth
- 2 Mb RAM
- 20 Mbyte Internal Hard Drive
- Ovation DTP

Normal Price £1299 + VAT

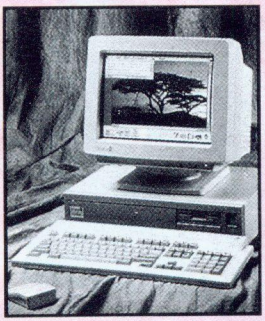
Save Over £300

Special Offer £999 + VAT

(£1173.83 inc. VAT)

The **A3000 Learning Curve** is also available if required.
This includes Pacmania game, Genesis Plus Database, 1st Word Plus, Acorn PC Emulator and a 120 min audioTraining Tape.
Just add £40 + VAT (£47.00 inc VAT).
Courier delivery please add £9.00.

The A5000 Learning Curve



The A5000 is now available from BEEBUG, either from our showroom or mail-order.

We are one of Acorns largest dealers and have been supporting the Archimedes range since its launch.

You can have total confidence in BEEBUG. Our technical team are always on-hand to provide any assistance and help that you may need with the A5000.

BEEBUG & RISC Developments also produce the magazine RISC User, dedicated to the Archimedes range.

BEEBUG - The Archimedes Specialists

A5000 Features

- RISC OS Version 3
- ARM 3 For Unbelievable Speed
- 1.6 Mb Format Floppy Drive
- 40 Mb IDE Hard Drive
- Acorn Multi-Scan Monitor

The New Learning Curve Pack

- New Multi-tasking PC Emulator
- Genesis plus Database
- 1st Word Plus Wordprocessor
- Acorn DTP
- Pacmania Game
- Audio Training Tape
- Optional 300 dpi Ink Jet Printer

The A5000 Learning Curve complete with Acorn Multi-scan Monitor Is now available for **£1799** inc. VAT.
Phone or write to reserve yours Now !
Courier delivery please add £9.00.

Educational Establishments
Please ask for our Educational A5000 price.

BEEBUG

Phone or write for an Information Pack
All products covered by 12 months full warranty
Access / Visa / Switch / Cheque / Official Orders Welcome
Showroom hours Mon to Sat 9 am - 6 pm (Thu until 8 pm)

BEEBUG Ltd, 117 Hatfield Road, St. Albans, Herts. AL1 4JS. Tel. 0727 40303 Fax. 0727 860263