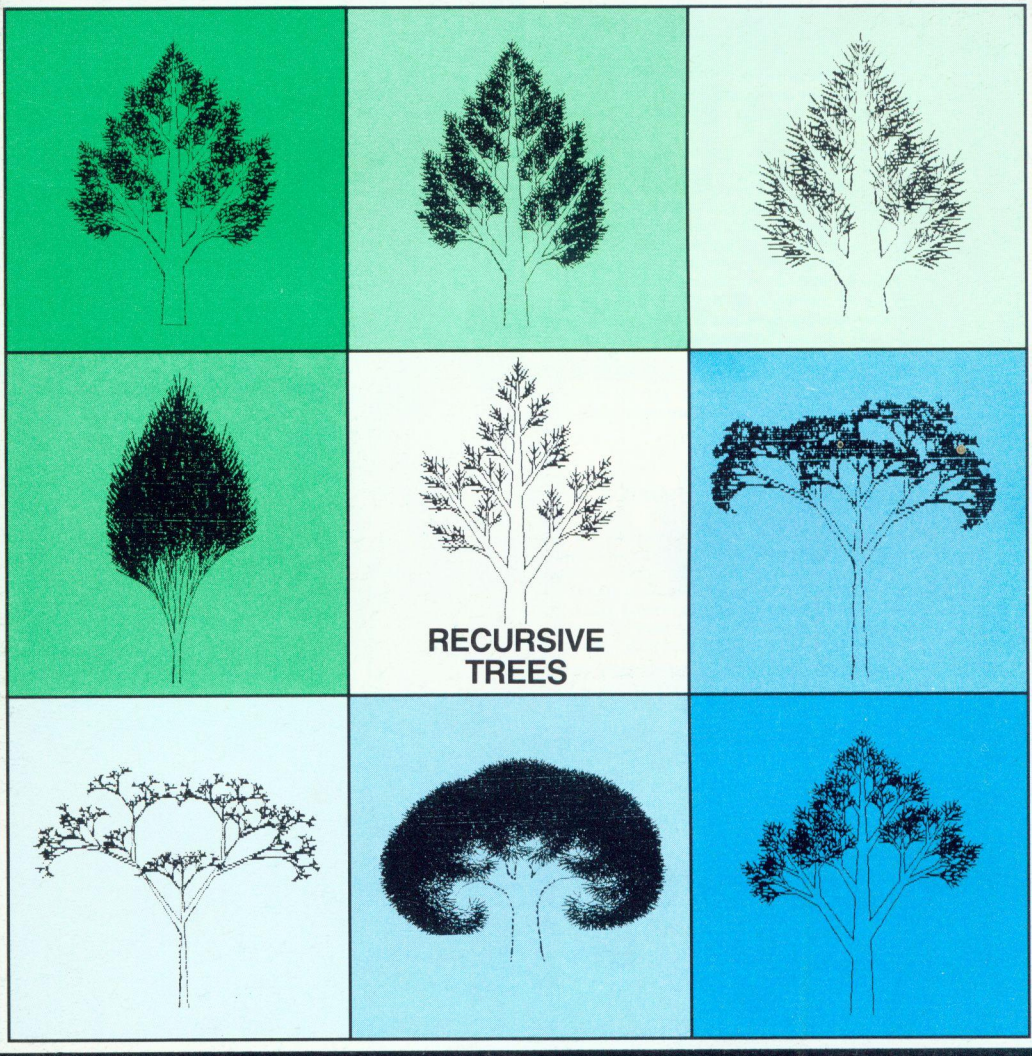
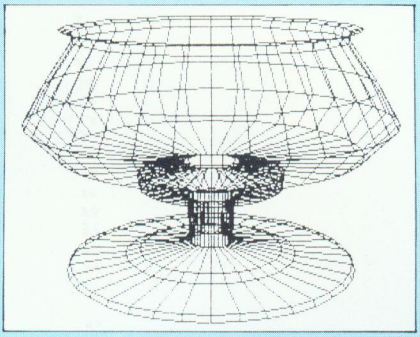
VOLUME 4 NUMBER 5 OCTOBER 1985 PRICE £1.20

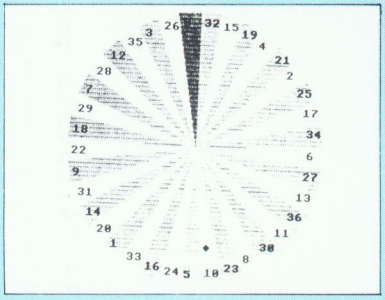
BEEBUG

FOR THE BBC MICRO

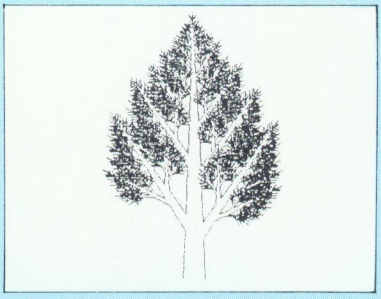




Glentop's 3D Graphics System

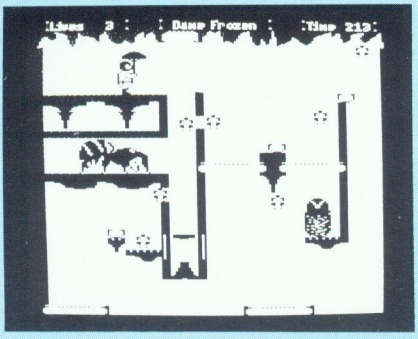


Roulette



Recursive Trees

Boffin



BEEBUG

VOLUME 4 NUMBER 5
OCTOBER 1985

GENERAL CONTENTS

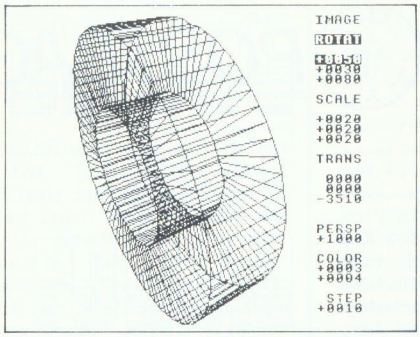
3 Editorial Jottings
4 Postbag
5 BEEBUGSOFT Forum
6 News
7 Recursive Trees
10 Glentop's 3D Graphics System
12 Dynamic Memory Window
15 Boffin – Review and Competition
16 Programming with Wordwise Plus (Part 1)
19 Hints for Wordwise Plus Users
19 Wordwise Help and Prize Competition
20 First Course
Print Formatting (Part 1)
22 Points Arising
23 EPROM Programmer Project (Part 2)
27 Loan Repayments
28 Double Density Disc Filing Systems Compared
32 BEEBUG Workshop
Text Compression
34 Adventure Games
35 Advanced Disc User Guide Reviewed
36 Further Disc Menu Extensions
38 Data Structures (Part 2)
42 Interfacing the Beeb
Eight Books Reviewed
45 Roulette

PROGRAMS

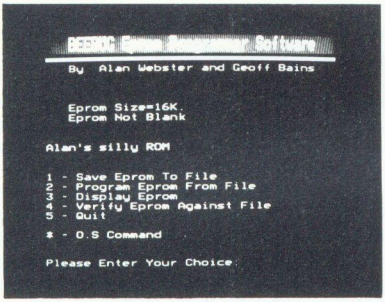
7 Recursive Trees
12 Dynamic Memory Window
16 Wordwise Plus Segment Programs
23 EPROM Programmer Driver
27 Loan Repayments
32 Workshop Procedures
36 Disc Menu Extensions
38 Data Structures – examples
45 Roulette Game

HINTS, TIPS & INFO

14 Music System Line Feeds
14 Z80 Filenames
35 Wordwise Plus Hints
VDU Codes
Sound
Filename Printing
Label Duplication
49 Local Points of Interest
49 Trouble with *FX3
49 Quick Random



Glentop's 3D Graphics System



EPROM Programmer

Further Disc Menu Extensions



EPROM Programmer Project



EDITORIAL JOTTINGS

Last month saw the first article in our new regular series for newcomers to computing on the Beeb – First Course. Despite Acorn's much-publicized troubles, new BBC micro owners are still to be found in as large numbers as ever, and BEEBUG intends to continue to support all Beeb owners, regardless of experience, as earnestly as we have in the past.

New and established members alike will be interested in another new magazine feature starting this month. The software arm of BEEBUG, has been given a voice in the form of the BEEBUGSOFT Forum. A page of the magazine has been devoted to bringing you news of new BEEBUGSOFT products and help with the old ones.

The recent Acorn User show was well attended and provided an opportunity for members to see all these BEEBUGSOFT products demonstrated and to try them out for themselves. There is a similar opportunity to view the software and meet the people behind BEEBUG at the Micro User Show at the Horticultural Halls in London between November 14th and 17th.

This month's magazine cassette/disc contains a couple of extra items. The full disc benchmark program is there for members interested in how the results published in the Double Density Disc Systems review, in this issue, were obtained. The program runs on single density systems too, so you may like to test out your own DFS if you have one.

The second extra on the cassette/disc is a bit unusual, for BEEBUG at any rate. We have included one of the demonstration programs from Glentop's 3D Graphics Development System. This self-contained program will show you just the kind of sophisticated three dimensional animation that this package is capable of. See the back cover for the full details of all the programs on the magazine cassette/disc.

PROGRAM CLASSIFICATION

All programs in the magazine, and on the magazine cassette/disc, are marked with the symbols shown below. An uncrossed symbol indicates full working, a single line through a symbol shows partial working (normally some modifications will be required), and a cross through a symbol indicates a program that will not work on that system.

Basic I	I	Electron	
Basic II	II	Disc	
Tube		Cassette	
Model B+	+		

POSTBAG

POSTBAG

Logically speaking

Thanks to Peter Lewis for some interesting tutorials on the use of logic (see BEEBUG Vol.4 Nos.2 & 3). Here is a quirk that puzzled me:

```
FOR J=Ø TO 5:PRINT J,(J MOD 2):NEXT
```

This worked OK but not this extension:

```
FOR J=Ø TO 5:PRINT J,(J MOD 2);:IF (J MOD2) PRINT " odd" ELSE PRINT " even":NEXT
```

I changed the IF statement to:

```
IF NOT(J MOD 2) PRINT " even" ELSE PRINT " odd":NEXT
```

and then to:

```
IF (J MOD 2) PRINT " odd":NEXT ELSE PRINT " even":NEXT
```

With this, I finally discovered that the IF must include its own NEXT as well as the ELSE.

More tutorials please.

Dennis Kemp

As Dennis Kemp found out, you can easily run into problems if you have NEXT in an IF-THEN-ELSE statement. The same can happen with REPEAT-UNTIL. It is best to avoid this situation altogether by using a different programming technique.

For example, in the above set type$(Ø)=" even" and type$(1)=" odd". The IF statement that caused all the trouble can then be replaced by:

```
PRINT type$(J MOD 2):NEXT
```

A Point of View

Users of View 1.4 may be interested to know that they can obtain View 2.1 in exchange for £1Ø and the 1.4 ROM from Acornsoft at Cambridge Technopark, 645 Newmarket Rd, Cambridge CB5 8PD. The two revised manuals "Into View" and "View Guide" are £2.5Ø each. However, View 1.4 users already have two good manuals. All we need is a note of the differences. The details (with thanks to BEEBUG) are as follows:

PRINT, SHEETS and SCREEN may be used on text in memory without saving it first. Just issue the command without a file name.

A new stored command, LJ (left justify), works in a similar fashion to RJ (right justify).

A new command FOLD can be used with the CHANGE command. With FOLD on (FOLD 1) characters being changed retain their original case. With FOLD off (FOLD Ø) characters to be changed take on the case supplied by the user (e.g. you can now change 'BASIC' to 'Basic', impossible before.

Joyce Diment

Clarss Distinction

The new Computer Concepts Speech ROM is a beautiful chip. Within minutes of installing it, I was able to produce quite a long sentence which was absolutely intelligible and with "expression". But then I thought I had found two glaring omissions, the sound for "AW" as in "awful" or "law", and the sound for "AH" as in "task". They give for these, respectively, "OR" and "AR", and actually give examples "sor" for "saw" and "clarss" for "class".

I am not especially for clarss distinction but this is orful! However, I find on closer listening, that they do indeed pronounce "AR" as "AH", leaving one obliged to use an "AH" sound for "are". One can go too fah!

Regarding clarity, I realised that as I knew what I intended, that was what I heard when listening to the result. So I made a recording of some longer sentences and asked a group of blind, or nearly blind, people to listen. The general verdict was that they needed to hear it at least three times before the meaning was clear. I was rather crestfallen as I thought it was pretty good. Maybe the speed is just a little too fast.

M.B.Dearlove

We are hoping to review this new Computer Concepts ROM in the near future. The point about knowing what to expect is a good one which we will take up in our review.

BEEBUG SOFT FORUM

Welcome to the first BEEBUGSOFT Forum. We hope to make this page a regular feature in BEEBUG magazine to give us, at BEEBUGSOFT, a chance to keep you, the members, up to date on software developments.

We will also use this page to answer some of the questions on BEEBUGSOFT products that we are most frequently asked by members, and to give some hints and tips about using BEEBUGSOFT products. If you wish to raise any points concerning our products, ask any questions, or make any contributions that you feel would be of interest to other members, please write to BEEBUGSOFT Forum at the Editorial address.

We are always pleased to receive suggestions and ideas, and as you will have seen from last month's price list, we are happy to evaluate members programs and consider them for inclusion in the next BEEBUGSOFT catalogue.

WHO ARE BEEBUGSOFT?

BEEBUG Publications and BEEBUGSOFT are entirely separate entities, with separate staff. We are however, located in the same offices and often work closely together. Our aims are similar: to produce, respectively, the magazine and the software that will be of interest to you, the members.

WORDEASE

At the Acorn User exhibition we launched our new WordEase program, a utility to assist with, and add to, Wordwise Plus. If you have any queries on WordEase, or suggestions for any further extensions to Wordwise Plus, we'd be pleased to hear from you.

PAINTBOX II OR PAINTMASTER?

Our popular drawing package, Paintbox II, has been renamed 'Paintmaster'. A company called Quantel has a prior registration of the name Paintbox, and has requested that we rename our program to avoid possible confusion. Paintmaster is identical in every way, except name, to Paintbox II.

NEW BEEBUG DISCS

We have just increased our range of BEEBUG blank discs. We can now offer 4 types of disc to meet your exact requirements:

ss dd 48TPI ds dd 48TPI
ss dd 96TPI ds dd 96TPI

Unfortunately we have also had to pass on a small increase in price. Full details are on the inside back cover of BEEBUG.

EXMON II HINT

Exmon II is able to assemble directly from disc using files created on a wordprocessor such as Wordwise, giving a number of advantages, including greater available memory. This useful facility is covered in the Exmon manual (see page 26) but many users seem unaware of it.

QUITTING MASTERFILE

When using Masterfile II it is essential that you use option "P" to quit the program, as described in the manual. If you press the Break key you may cause damage to your data because files will not have been updated and closed in a correct manner.

LICENCES

All BEEBUGSOFT products are subject to copyright. Unauthorised copying, apart from the production of one backup copy only for use by the purchaser, is strictly prohibited. However, we operate a licensing scheme to allow schools and colleges to have access to, and to produce, multiple copies of our programs at special rates. Further details are available from the Software Manager at the St. Albans address.

COMING SOON

We will shortly be increasing the amount of software and hardware available from BEEBUGSOFT. Watch this space for further information.

NEW TELEPHONE NUMBER

Please note our new telephone number for order queries and for technical enquiries is:

St. Albans (0727) 40303.

All telephone orders should still be placed on 049481 6666.

News News News News News News.....

Aries the Ram

Aries Computers, the first to produce a shadow RAM board for the Beeb, has now released a new version offering 32K extra RAM. This is 20K screen memory and 12K sideways RAM, along the same lines as the B+, and includes a sideways ROM socket to take the controlling ROM. The B-32 board costs £92 from Aries on 0223-862614.

Music Countdown

Hybrid Technology, designer of the Music 500, has previewed further models to extend the range, for a November release. The Music 400 is the long-awaited keyboard, the Music 200 is an interface to the industry standard MIDI specification and the Music 100 is a monitor amplifier offering 4 watts per channel. The music control language, Ample, to look after all of these, is to appear in new clothing as the 'Ample Nucleus' in ROM. Reviews of all coming soon. Hybrid also has an audio tape of the Music 500 in action for potential users. The tape costs £1.95 from Hybrid on 0223-316910.

Keynotes

Two new numeric keypads have arrived on the Beeb scene. Keypad from Commercial products (0293-30174) costs £44.95 and connects in parallel with the main keyboard and so is totally compatible with all software. It offers the numbers 0-9, decimal point and Return. The Kenpad costs £60 from K. Blanchard (04446-41290), one of the authors of Acornsoft's Termulator, and is designed for use with that ROM. It plugs into the User Port and offers 0-9, point, Return, and four special function keys.

Multi-Tasking

Multi-Basic is a sideways ROM that adds multi-tasking ability to your Beeb. Up to 8 background tasks, written in standard BBC Basic, can be operative at one time and triggered by time or logical conditions. The ROM also includes several I/O control commands for interfacing enthusiasts. Multi-Basic costs £34.95 from CMS on 0371-5666.

The Disc Drive's Friend

BS-DOS is the latest DFS for the Beeb from CUC. BS-DOS costs £39.50 and works alongside any Acorn-type DFS to provide several enhancements. Double sided disc drives are now configured as single drives, across the two surfaces, with up to 256 files. BS-DOS is compatible with all second processors (including Torch) and supports all the normal OS calls. Further details from CUC on 01-311 2555.

Space Bits

A self-contained, simple to use satellite data receiving and decoding package is available for the BBC micro. Astrid will receive, decode, store and display data from UoSAT 1 and 2. Astrid costs £149 from MM Microwave on 0751-75455.

Bilbo II

The infamous Hobbit adventure from Melbourne House (01-940 6064) is now available in a much expanded form on disc for the Beeb. The new Hobbit has more locations and over 50 graphic screens and costs £17.95.

More Teletext

Morley Electronics has released a teletext adaptor for the Beeb. The Morley adaptor connects to the User Port and, along with the inevitable sideways ROM, enables your Beeb to receive all teletext information and to down load programs. The tuning of the receiver is entirely under software control. The adaptor with ROM and power supply costs £100.

The Case for the Beeb

Oak Computers has launched a new and impressive looking cabinet for your BBC micro. Moulded in light grey plastic, the 'Personal Computer' range has a detached keyboard and a main cabinet which can house the main circuit board, power supply, disc drives, second processor, modem, etc. Prices start from £100. Further details from Oak on 0274-614167.

Recursive Trees

William Godwin uses the technique of recursion to produce a veritable forest of delights.

In the world of home micros the BBC micro stands out as having one of the most powerful recursion facilities. This program exploits this technique to the full to produce a variety of tree designs.

To create a tree, the various parameters that define the tree's features are first edited. These are explained in full in the program notes section, but briefly they comprise the tree's height width and density of branches. To select a parameter, you move the flashing cursor across the screen using the left and right cursor keys. Then you can either type in a new value for the parameter by first pressing the space bar, or use the up and down cursor keys to change the current digit of the value displayed.

For example, to change the number 100 to the number 200, you could place the cursor anywhere underneath the number, press the space bar and enter the new value, 200. Alternatively, you could place the cursor under the digit 1 and press the cursor up key to alter the digit 1 to a 2.

Each time you alter a parameter, the basic shape of the tree is drawn. This is not an exact copy of how the tree will finally look; it will be much more complex than this. It is just a rough guide.

When you are satisfied with your parameters, pressing the Return key will then start the drawing of the tree. If you get fed up with waiting for a certain tree to be drawn, as you will with one that involves a high level of recursion, pressing the Delete key will halt the drawing of the tree and allow you to return to the 'edit mode'.

The trees can take a long time to be drawn. The time taken is largely dependant on the value for 'TWIG'. Small values, such as 8, give very detailed trees that take a long time. Try larger values until you are satisfied with the general shape of the tree.

The trees produced by the program can be dumped to any Epson-compatible printer by pressing the Tab key. This initiates the screen dump routine in PROCgemini. If you have a printer that is not Epson-compatible you can insert your own screen dump routine in the procedure, PROCgemini and call it appropriately from line 2180. Alternatively you can call one of the various screen dumps in ROM from this line (*GDUMP for Printmaster, for example) and leave out PROCgemini (and line 130) altogether.

RECURSION

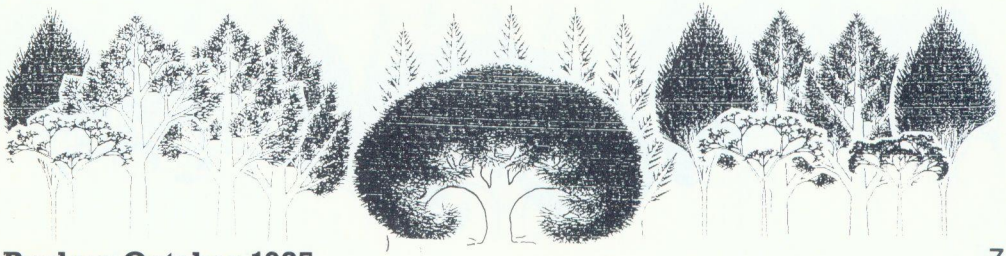
A recursive routine is one that calls itself. Obviously such a state of affairs could go on forever in an endless loop. The usual implementation of recursion in BBC Basic, therefore, is a function or a procedure that calls itself only if a particular condition is met. The condition in the case of this program is the length of the current branch being drawn. The drawing process - splitting the current branch recursively - is repeated until the branch is smaller than TWIG.

PROGRAM NOTES

The size and shape of the tree is determined by six parameters:

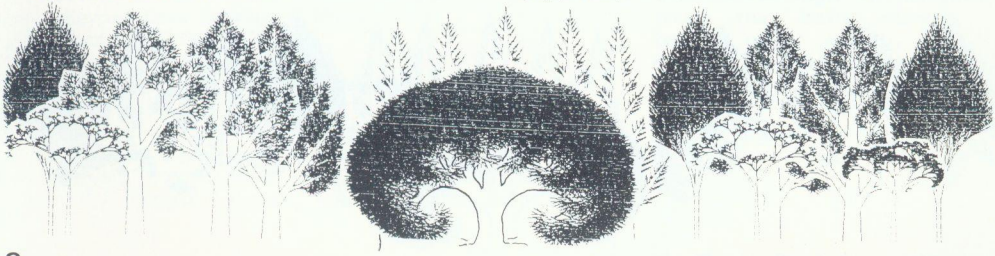
TWIG This gives the size of the smallest detail. The tree is drawn by drawing a smaller branch on each branch until the twig size is reached

TREE This gives the height of the main tree trunk

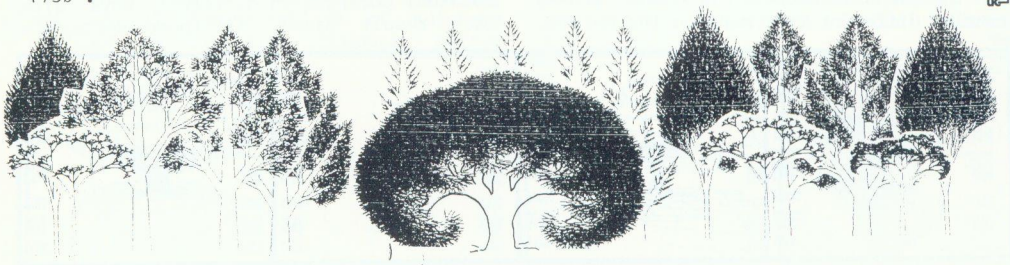


ANGLE	This is the angle at which new branches grow. A new branch would grow at right angles to the previous one if an angle of 90 degrees was stated
LIMB	This value governs the thickness of a limb (branch)
HIGH	This determines approximately how high up a new branch will grow
LONG	This gives the relative length of a limb. The maximum value is 999, but high values usually cause too much recursion

```
  10 REM PROGRAM TREE
  20 REM VERSION B0.2
  30 REM AUTHOR  W.GODWIN
  40 REM BEEBUG  OCTOBER 1985
  50 REM PROGRAM SUBJECT TO COPYRIGHT
  60 :
 100 ON ERROR GOTO210
 110 MODE1:PROCintro
 120 N%=5:DIM param(N%),code%&86
 130 IF code%?&85<>&60 PROCgemini
 140 F%=6:@%=F%
 150 FOR J%=0 TON%:READ param(J%):NEXT
 160 J%=0:REPEAT PROCedit
 170 PROCtree(tall,twig,640,8,trunk,H,u
p,down)
 180 *FX15 1
 190 IFGET=9 PROCprint
 200 UNTIL FALSE
 210 ON ERROR OFF:*FX4
 220 MODE7:REPORT:PRINT" at line ";ERL
 230 END
 240 :
1000 DEF PROCedit:*FX4 1
1010 K%=1:Y%=F%-1:VDU22,1:IF FNtest THE
N210
1020 PROCdisplay
1030 REPEAT PRINTTAB(F%*J%+Y%,31);
1040 A%=GET
1050 IFA%=32PRINTTAB(F%*J%+1,31)">>";:I
NPUTTAB(0,29)"VALUE",K:PROCalter(K-param
(J%))
1060 IFA%=138PROCalter(-K%)
1070 IFA%=139PROCalter(K%)
1080 IFA%=136PROCleft
1090 IFA%=137PROCright
1100 UNTILA%=13:VDU22,0:ENDPROC
1110 :
1120 DEF PROCalter(X%)param(J%)=param(J
%)+X%
1125 IFFNtest param(J%)=param(J%)-X%:EN
DPROC
1130 CLS:PROCdisplay:ENDPROC
1140 :
1150 DEF PROCleft
1160 IFK%<1000K%=K%*10:Y%=Y%-1:ENDPROC
1170 IFJ%>0J%=J%-1:K%=1:Y%=F%-1
1180 ENDPROC
1190 :
1200 DEF PROCright
1210 IFK%>1K%=K%DIV10:Y%=Y%+1:ENDPROC
1220 J%=(J%+1)MOD(N%+1):K%=1:Y%=F%-1:EN
DPROC
1230 :
1240 DEF PROCdisplay
1250 PROCtree(tall,tall*tall-1,640,8,tr
unk,H,up,down)
1260 PRINTTAB(0,30)"  TWIG   TREE ANGLE
 LIMB  HIGH  LONG"
1270 FORI%=0TON%:COLOURI%MOD2+1:PRINTpa
ram(I%),;:NEXT
1280 ENDPROC
1290 :
1300 DEF FNtest
1310 IFparam(0)<2THEN=TRUE ELSEtwig=par
am(0)*param(0)
1320 theta=PI-RADparam(2):phi=PI-RADpar
am(3)-theta:CT=COStheta:ST=SINtheta:CP=C
OSphi:SP=SINphi
1330 H=param(4)*.001:up=param(5)*.001:K
=up*ST
1340 down=K/SP:D=K*(CT/ST+CP/SP):E=1-H-
D
1350 tall=param(1):trunk=tall/TAN((thet
a+phi)*.5)
1360 IFtall<0 OR E<0 OR D<0 OR H<0 THEN
=TRUE
1370 IF50*LNup>LN(twig/tall/tall)THEN=T
RUE
1380 IF50*LNdown>LN(twig/tall/tall)THEN
=TRUE
1390 =FALSE
1400 :
1410 DEF PROCtree(tall,twig,P,Q,trunk,H
,up,down)
```



```
 1420 P=P-trunk:MOVEP,Q:PROCuptree(trunk
,tall):PROCdowntree(trunk,-tall)
 1430 ENDPROC
 1440 :
 1450 DEF PROCuptree(X,Y):LOCALU,V,K,F
 1460 K=H-RND(1)*(H*(H<E)+E*(E<H)):F=1-D
-K
 1470 IFX*X+Y*Y<twig PROCline(X,Y):ENDPR
OC
 1480 PROCline(K*X,K*Y):PROCturn(CT,ST):
PROCuptree(up*U,up*V)
 1490 PROCturn(CP,-SP):PROCdowntree(down
*U,down*V):PROCuptree(F*X,F*Y)
 1500 ENDPROC
 1510 :
 1520 DEF PROCdowntree(X,Y):LOCALU,V,K,F
 1530 K=H-RND(1)*(H*(H<E)+E*(E<H)):F=1-D
-K
 1540 IFX*X+Y*Y<twig PROCline(X,Y):ENDPR
OC
 1550 PROCdowntree(F*X,F*Y):PROCturn(CP,
SP):PROCuptree(down*U,down*V)
 1560 PROCturn(CT,-ST):PROCdowntree(up*U
,up*V):PROCline(K*X,K*Y)
 1570 ENDPROC
 1580 :
 1590 DEF PROCturn(C,S):U=X*C-Y*S:V=X*S+
Y*C:ENDPROC
 1600 :
 1610 DEF PROCline(X,Y)
 1620 IFINKEY(-90)twig=1E6
 1630 P=P+X:Q=Q+Y:DRAWP,Q:ENDPROC
 1640 DEF PROCintro
 1650 COLOUR2
 1660 PRINT'"       TREE DESIGNER"
 1670 PRINT'"Use the ARROW KEYS to alter
 parameters"
 1680 PRINT'"    or SPACE  to enter a va
lue;"
 1690 PRINT'" Press RETURN to draw the T
REE"
 1700 PRINT'" (then DELETE to escape)"
 1710 PRINT'"    or TAB    to send to pr
inter."
 1720 PRINT'" Press SPACE to continue."'
:IFGET
 1730 ENDPROC
 1740 DATA90,900,150,4,100,650
 1750 :
 1760 DEF PROCgemini
 1770 COL=&70:ROWS=&71:LOC=&72:STO=&74:W
R=&FFEE:FX=&FFF4
 1780 FORI%=0TO2STEP2:P%=code%:[OPTI%
 1790 LDA#2:JSR WR
 1800 LDA#27:JSR PR:LDA#65:JSR PR:LDA#8:
JSR PR
 1810 LDA#&84:JSR FX:STX LOC:STY LOC+1
 1820 LDA#32:STA ROWS
 1830 .BEGIN
 1840 LDA#80:STA COL
 1850 LDA#27:JSR PR:LDA#76:JSR PR:LDA#12
8:JSR PR:LDA#2:JSR PR
 1860 .BLOCK
 1870 LDY#7
 1880 .FETCH
 1890 LDA(LOC),Y:STA STO,Y
 1900 DEY:BPL FETCH
 1910 LDY#8
 1920 .SEND
 1930 LDX#7
 1940 .PINS
 1950 ASL STO,X:ROR A
 1960 DEX:BPL PINS
 1970 JSR PR
 1980 DEY:BNE SEND
 1990 LDA LOC:CLC:ADC#8:STA LOC
 2000 BCC NOC
 2010 INC LOC+1
 2020 .NOC
 2030 DEC COL:BNE BLOCK
 2040 LDA#13:JSR PR
 2050 DEC ROWS:BNE BEGIN
 2060 LDA#27:JSR PR:LDA#48:JSR PR
 2070 LDA#3:JSR WR
 2080 RTS
 2090 .PR
 2100 PHA:LDA#1:JSR WR
 2110 PLA:JSR WR:RTS
 2120 ]NEXTI%
 2130 PRINT~P%:ENDPROC
 2140 :
 2150 DEF PROCprint:*FX3 10
 2160 FORI%=0TON%:PRINTparam(I%),;:NEXT:
PRINT
 2170 *FX3
 2180 CALLcode%:ENDPROC
```



Glentop's 3D Graphic System

Geoff Bains takes a look at a new versatile graphics development system that promises to get more from your Beeb's screen than you ever thought possible.

Product : Graphics Development System
Supplier: Glentop Publishers,
Freepost, Barnet,
Herts.
01-441 4130
Price : £25.00

There are many graphics systems on the market for the BBC micro. These will allow you to create pictures of varying complexity on your Beeb's screen and display them. What all these packages lack, however, is any real application for the resulting pictures. All you can do is to save them to cassette or disc and load them in again at a later date.

The Graphics Development System from Glentop Publishers, however, is a different story. This unique package allows you to create wireframe representations of 3D objects on the screen, manipulate them in a variety of ways and then to use the resulting images in Basic or assembler programs to produce some stunning animated images.

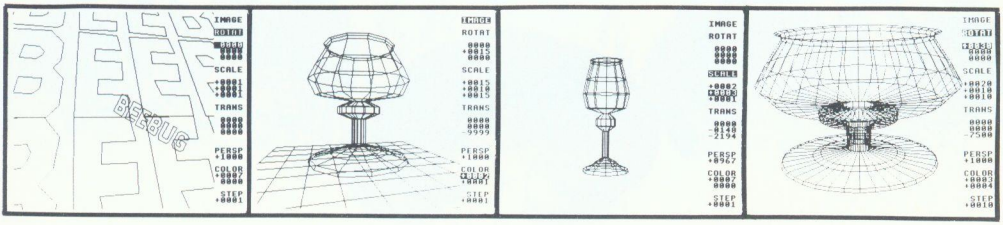
The graphics creation programs are a little different from many on the market. This package is exclusively concerned with wireframe models. There is no block filling, shading, or even much in the way of colour. The models are defined as a series of three-dimensional co-ordinates, representing each point of the wireframe, entered into an editor. As well as the co-ordinates each point is also defined as a draw or move operation with the colour and type (solid or dotted) of line used.

The data tables produced by the editor can be saved onto disc and reloaded for further editing. Unfortunately there is no option to insert extra points into the middle of a data table; these can only go onto the end. This does mean that alterations can take up unnecessary extra table entries.

Once an object's data is entered it can be viewed on the screen. The program displays a perspective projection of the wireframe and allows you to alter the viewpoint and perspective, and to perform rotations, scalings, and translations in three dimensions. The various parameters are displayed alongside the image and are altered using the cursor keys.

That's all very well but it doesn't go very much further than the 3D Rotation program in BEEBUG Vol.1 No.10. However, the Graphics Development System doesn't stop there. Shapes can be rotated about any of the three axes to form surfaces. The program to do this is very versatile, allowing either the lines parallel or perpendicular to the axis of rotation only to be created, or both, and the angle through which the rotation is performed and the step size to be altered at will. The creation of such 'profiles' is, like all the operations in the Graphics Development System, extremely fast.

Any wireframe model can be used as a 'macro' and stretched, translated, rotated, enlarged in different ways with each result saved. All these new models



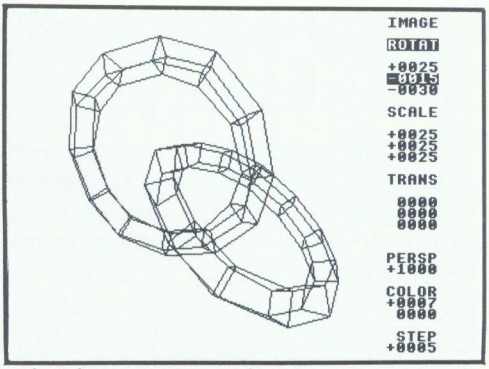
can be then linked together to form a new creation. In this way complex models can be built up with the minimum of data entry. The interlocking rings shown here, for example, were created by manipulating a simple diamond shape (five data entries only!). This was turned about one axis to give a profile and the resulting ring used as a macro. This was translated and rotated to form the two rings which were then linked together to form the whole picture.

The only limit to this model creation is the Beeb's memory. The Graphics Development System operates perfectly with the 6502 second processor, however, so larger models are available with this connected. The package also has a disc and printer dump option allowing you to save your masterpieces in a more immediate form.

The model-creation section of the Graphics Development System is very effective but it is somewhat eclipsed by the other half of the package - the programs to help you make some use of the models you have created. These consist of two types. Firstly there are the machine code routines that can operate on the data that represents the 3D co-ordinates of an object. These are all bundled together in one group in a variety of formats for use with different systems - cassette, disc, or second processor - and in different memory positions.

To access these routines there is provided the beginnings of a Basic program with procedures already defined to access the machine code. To display the image of an object, then, you just load the machine code, load the data, and run the Basic program suitably amended to include calls to the procedures required (PROCgcol, PROCtrans, PROCrot, PROCproject, etc.).

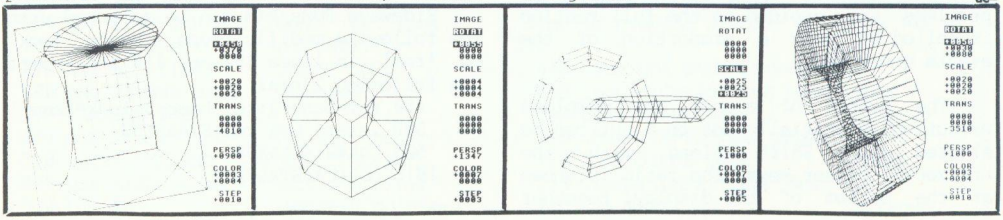
Objects can be animated in this way and several demonstrations of this are provided on the disc. However, better



animation can be achieved by resorting to assembler to access the 3D routines rather than Basic. This again is fairly simple as it is really just a case of calling up the routines when needed. However, really fast animation can be achieved by using another feature of the Graphics Development System.

The Beeb's OS can plot lines on the screen at the rate of about 9,000 pixels per second. The Graphics Development System contains its own line drawing routines that will draw at a rate of 30,000 pixels a second. Although it has some restrictions (pixel co-ordinates, GCOL0 type plotting only, no windows or redefinable origin, and it will not work with second processor) the speed difference can be staggering. A marvellous Elite-type space craft performs a complex turning manoeuvre without a suggestion of flicker in one of the demonstrations. (We have included this demo on this month's magazine cassette/disc so you can see the effect to the full).

All this takes some getting to grips with. The manual is adequate but better tutorials could be given. However, if you are looking for a package that will really take your Beeb to the limits of graphics and animation for a reasonable price, then the Graphics Development System delivers the goods.



Dynamic memory display

The BBC micro's event system provides Sebastian Lazareno with a dynamic and fast moving window onto the Beeb's inner workings.

Many readers will appreciate the value of a static memory dump, giving a snapshot of the micro's memory, but such a display does not show changes in RAM as they occur. This program provides a dynamic memory window, allowing you to observe the Beeb's memory in real time while using the keyboard, running other programs (including ROMs), or just 'resting'. The program can also display the contents of any sideways ROM, and is particularly useful for monitoring sideways RAM.

PROGRAM ENTRY

Save the program before you run it, since an error may cause the computer to hang. Cassette users should alter the value of 'code' in line 160 to &C00.

USING THE PROGRAM

When you RUN the Basic program you will be asked for the start address of the display (i.e. to display zero page, enter 0). The default display start is at PAGE.

The main routine extends through page 9 without intruding into page 10 (Toolkit workspace). It displays the contents of 48 bytes of memory in 'dump' format at the top of a mode 7 screen. You can scroll through memory using the vertical cursor keys with Shift (slow) or Ctrl-Shift (fast). The routine is called 50 times a second by the 'start of vertical sync' event. According to the Advanced User Guide (p.288) such routines should not last more than about 2 milliseconds. This one takes about 5.2 milliseconds, but problems are avoided as the full routine is called by only a proportion of the events generated.

The number of events per complete execution is initially set to 5 but can be altered using Shift along with the horizontal cursor keys. The ratio is shown at the bottom of the display, together with the number of events since the last execution and the display start address. The whole routine can be turned off using function key f7 and turned on using f8. This disables and enables the event generation.

POSSIBLE USES

1.) ROMS. Look at locations starting at &8000 and type *HELP: by stopping the display with Shift+Ctrl you can explore any of your active ROMs. With a small modification to the program (see below) the contents of any sideways ROM or RAM can be continuously displayed.

2.) Entering a line of Basic. Look at locations &700 onward (the Basic keyboard buffer), enter a line of Basic and see the line being tokenized when you press Return.

3.) Keyboard entry. Still looking at &700 onward, use the Copy key to enter a few characters. Note that the characters entering the buffer are the ones you copied, and this is still so if Shift or Ctrl are pressed with Copy. Now look at &3E0 onward (the keyboard buffer). Note that when you Copy a character, the code entering the buffer is that for the Copy key, rather than the character copied, and that pressing Shift or Ctrl together with Copy causes a different code to enter the buffer. Now look at locations &BC and &ED (which contain 'current keys pressed' information) and see that pressing Copy produces yet another code (the 'INKEY' code), which is not affected by pressing Shift and Ctrl. These observations provide some clues as to how the Beeb processes keyboard input, and suggest further questions.

PROGRAM MODIFICATIONS

The program displays both Teletext character sets, i.e. ASCII 32-126 and 160-255, so many displayed characters are ambiguous. To display only the normal set delete line 670.

Sideways ROMs/RAM can be selected with the following modifications. Change 'lock' to 'rom' in lines 390 and 410 and insert the following lines:

```
 150 rom=base+2:key=base+3:lock=base-1
 500 .start LDA rom:STA&FE30
 505 LDA#0:STA&308
1015 LDA&F4:STA&FE30
```

To inspect your RAM (or ROM), set the display to start at &8000 and press Shift with the horizontal cursor keys to select the required ROM.

As it stands, the routine monopolizes the event vector. The program could be modified to save the previous contents of the vector and exit via a JMP so that any other user routines using this event can take their turn.

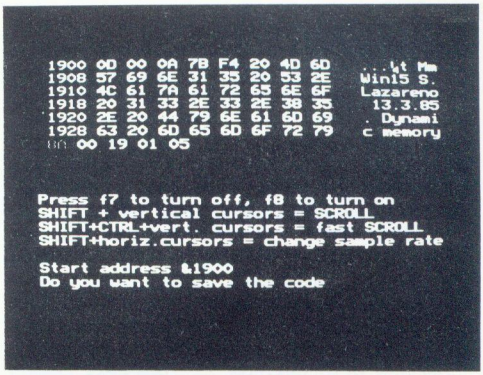
PROGRAM NOTES

Lines 220-800: checks number of event, keys pressed, sets up pointers, text window.

Lines 810-1120: the display routine.

Lines 1130-1220: sets up soft keys. Clears buffer if necessary using equivalent of ON ERROR to detect full soft key buffer. Starts routine by invoking f8.

```
  10 REM PROGRAM MEMWIN
  20 REM VERSION B0.7
  30 REM AUTHOR  S.LAZARENO
  40 REM BEEBUG  OCT 85
  50 REM PROGRAM SUBJECT TO COPYRIGHT
  60 :
 100 *FX13,4
 110 MODE7
 120 PRINT'"Press f7 to turn off, f8 to
 turn on"'"SHIFT + vertical cursors = SC
ROLL"'"SHIFT+CTRL+vert. cursors = fast S
CROLL"'"SHIFT+horiz.cursors = change sam
ple rate"
 130 ON ERROR GOTO 1600
 140 base=&8A:ad=base:INPUT"Start addre
ss &"S$:!ad=EVAL("&"+S$)
 150 key=base+2:lock=base+3
 160 line=base+4:code=&8C0:?key=0:?lock
=5
 170 REM EVENTV points to routine
 180 ?&220=code MOD &100:?&221=code DIV
 &100
 190 FORI%=0TO2STEP2
 200 P%=code
 210 [OPT I%:PHP:PHA:TXA:PHA:TYA:PHA
 220 LDX key:INX        \Increment
 230 STX key:CPX lock   \key,
 240 BEQ scrollcheck
 250 LDX #240:JMP end   \exit if < lock
 260 .scrollcheck       \is SHIFT or
 270 LDA &25A:TAX       \SHIFT LOCK
 280 AND #8:BNE ctrl    \pressed?
 290 TXA:AND #32
 300 BNE start          \If so, check
 310 .ctrl TXA:AND #&40\for CTRL,set
 320 LSR A:BNE scan     \acc. to alter
 330 LDA #8             \address pointer
```




```
 340 .scan LDX &EC      \Check whether
 350 BEQ skip           \a cursor
 360 PHA:SEC:TXA        \key has been
 370 SBC #128:TAX:PLA   \pressed,
 380 CPX #25:BNE right \if so,
 390 DEC lock:JMP start\take
 400 .right CPX #121    \appropriate
 410 BNE up:INC lock    \action.
 420 JMP start
 430 .up CPX #57:BNE down
 440 CLC:ADC ad:STA ad
 450 BCC skip:INC ad+1
 460 .skip JMP start
 470 .down CPX #41:BNE start
 480 EOR #&FF:ADC ad::STA ad
 490 BCS start:DEC ad+1
 500 .start LDA #0:STA &308
 510 LDA #24:STA &309   \Equivalent
 520 LDA #39:STA &30A   \to
 530 LDA #8:STA &30B    \VDU28,0,24,39,8
 540 ORA &D0:STA &D0    \Text window set
 550 LDA #1:STA line
 560 LDY #0:LDX #0:STX key\Print addr.
 570 .loopad:JSR printad  \and contents
 580 .loop:LDA (ad),Y     \of next 8
 590 JSR printacc         \locations,
 600 LDA #32:JSR prinx    \first in
 610 INY:ASL line:BCC loop\hex,
 620 ROL line:LDA #131:JSRprinx:INX
 630 TYA:SEC:SBC #8:TAY
 640 .loop2:LDA (ad),Y    \then ASCII
 650 CMP #32:BCC dot      \if possible.
 660 CMP #127:BCC pound
 670 CMP #160:BCS next    \ Remove this
 line to prevent display of chars 160 -
255
 680 .dot LDA #ASC".":BNE next
 690 .pound:CMP#96:BNE hash \MODE 7
 700 LDA#35:BNE next        \display
 710 .hash CMP#35:BNE under \of
 720 LDA#95:BNE next        \these
 730 .under:CMP#95:BNE next \is
 740 LDA #96                \strange
```

```
  750 .next JSR prinx
  760 INY:ASL line          \Display 8
  770 BCC loop2:ROL line    \locations
  780 CPX #240:BCS end      \per line,
  790 JMP loopad            \6 lines.
  800 \
  810 .printhex:AND #&F     \Print
  820 CLC:ADC #&30:CMP #&3A\digit in
  830 BCC prinx:ADC #6      \hex
  840 .prinx:STA &7C00,X    \increment
  850 INX:RTS               \screen point
er
  860 \
  870 .printacc:PHA         \Print
  880 ROR A:ROR A           \contents of
  890 ROR A:ROR A           \accumulator
  900 JSR printhex:PLA      \in hex.
  910 JSR printhex:RTS
  920 \
  930 .printad:LDA #134:JSR prinx
  940 TYA:CLC:ADC ad:PHA    \Print
  950 LDA ad+1:ADC #0       \2 byte
  960 JSR printacc          \address,
  970 PLA:JSR printacc      \increment
  980 LDA #135:JSR prinx    \screen
  990 RTS                   \pointer.
 1000 \
 1010 .end
 1020 LDA #133:JSR prinx    \This routine
 1030 LDA #base             \executed
 1040 JSR printacc          \at every
 1050 LDA #135:JSR prinx    \call: prints
 1060 LDY #0                \the start
 1070 .loop:LDA base,Y      \address
 1080 JSR printacc          \of the
 1090 LDA #32:JSR prinx     \display, and
 1100 INY:CPY #4:BNE loop   \contents of
 1110 PLA:TAY:PLA:TAX       \lock & key
 1120 PLA:PLP:RTS           \then exits.
 1130 .invoke
 1140 OPT FNonerror
 1150 OPT FNturnoff         \Set f7
 1160 OPT FNturnon          \Set f8
 1170 OPT FNbreak           \Set f10
 1180 LDA line:STA&202      \ON ERROR
 1190 LDAline+1:STA&203     \OFF
 1200 LDA#138:LDX#0         \Invoke f8 to
 1210 LDY#136:JSR&FFF4      \turn on rout
ine
 1220 RTS:]
 1230 NEXT
 1240 CALLinvoke:END
 1250 DEF FNonerror:pass=P%
 1260 FORopt=0TO2STEP2:P%=pass
 1270 [OPT opt
 1280 LDA&202:STA line
 1290 LDA&203:STA line+1
 1300 LDA #(onerr MOD 256):STA &202
 1310 LDA #(onerr DIV 256):STA &203
 1320 BNE setup
 1330 .onerr:LDA#18:JSR&FFF4
 1340 .setup:]
 1350 NEXT:=I%
 1360 :
 1370 DEFFNturnoff:str$="K.7*FX13,4 M":=
FNoscli
 1380 :
 1390 DEFFNturnon:str$="K.8MO.7:V.31,0,8
:*FX14,4 M":=FNoscli
 1400 :
 1410 DEFFNbreak:str$="K.10O. M?&220=&"+
STR$~(code MOD256)+":?&221=&"+STR$~(code
 DIV256)+" M":=FNoscli
 1420 :
 1430 DEFFNoscli
 1440 pass=P%:FORopt=0 TO2 STEP2
 1450 P%=pass
 1460 [OPT opt
 1470 LDX#(string MOD256)
 1480 LDY#(string DIV256)
 1490 BNEoscli
 1500 .string
 1510 ]
 1520 $P%=str$
 1530 P%=P%+LEN$P%+1
 1540 [OPT opt
 1550 .oscli JSR&FFF7
 1560 ]
 1570 NEXT
 1580 =I%
 1590 :
 1600 ON ERROR OFF:MODE 7
 1610 IF ERR=17 END
 1620 REPORT:PRINT" at line ";ERL
 1630 END
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS

MUSIC SYSTEM LINE FEEDS - Allan Woods

Island Logic's Music System was said in our review (BEEBUG Vol.4 No.1) to produce an extra linefeed at each line on printing out music (if the printer was configured in the normal way). The @ key switches off this effect but is not documented.

Z80 FILENAMES - Michael Colley

Filenames starting with character 152 (conceal display in mode 7) operate perfectly on the unadorned Beeb. They are not recognised with a Z80 second processor.

BOFFIN

Boffin is an exciting new game from Addictive Games. Geoff Bains, BEEBUG's resident 'boffin', dons his white coat and reports on the game and on an exclusive BEEBUG competition.

Boffin is the second release for the BBC micro from Addictive Games and seems set to follow 'Football Manager' as a chart topper. The boffin of the title is a cute little mad professor armed only with an umbrella and an insatiable desire to wipe out horseshoes and collect pieces of laboratory apparatus. The horseshoes and the bits of glassware are scattered around the inevitable caverns. You must guide your boffin around the caverns, destroying the unlucky horseshoes and picking up bonuses in the form of the apparatus.

The caverns are filled with the usual array of platforms, barbed wire, perilous drops and trampolines. The latter are especially effective, giving a beautiful boing noise and actually flexing when they're jumped on. The umbrella is not just carried for show. It will stop the professor's fall and can even be used to pick up the objects.

Also in the caverns, however, is a superb assortment of suitably evil creatures - gigantic spiders, giant puffers, manta bats, etc. The movement of the spider is excellent, given the size of it, and the game is better than most when considering playability. The collision detection is very good too. You don't die when you are still inches away from your foe as happens in many other games. This is especially important with the spider as you have to get pretty close to that, at times, to win.

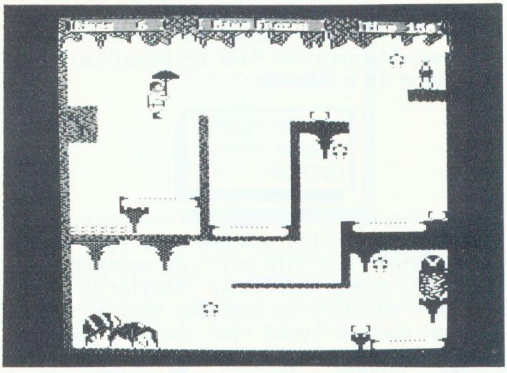
There is a high score table which records the number of the game currently being played and the game comes in two versions. Boffin I is Beeb only (B and B+) and boasts 45 caverns. Boffin II has only (only!) 25 caverns but it is compatible with both the Beeb and Electron.

Overall, the graphics are good, fast and smooth and the whole theme makes it very enjoyable. The mad professor with his umbrella is a great idea, and as an example of the Manic Miner type of caverns game, Boffin shows a wonderful sense of humour and is strongly recommended.

COMPETITION COMPETITION COMPETITION COMPETITION COMPETITIO

To celebrate the launch of Boffin, Addictive Games are running a competition exclusively for BEEBUG members. Each month Addictive are giving £50 to the highest scorer in Boffin II. To make things even easier for members there is a special price on the game as well. For BEEBUG members the excellent Boffin costs only £8.95, that's £1 off the normal price. Send your high scores to Addictive at:

BEEBUG Competition,
Addictive Games,
7a, Richmond Hill,
Bournemouth,
BH2 6HE.



You must quote your BEEBUG membership number and include a statement from a witness. Should you be the top scorer, you will be expected to reproduce a similar score at Addictive's offices.

Each month, the winner of the Boffin High Scores competition will be invited to Addictive Games to challenge Paul O'Mally, the author of Boffin, at his own game. The first reader skilful enough to beat him will win £250 worth of peripherals and add-ons, of his choice, for his computer. The results of the competition will be published in future issues of BEEBUG.

Wordwise Plus

In the first of three articles, Stephen Ibbs, an enthusiast for the new Wordwise Plus, introduces the whole idea of writing Wordwise Plus programs.

There are now over 20,000 Wordwise Plus ROMs in use. Perhaps the most exciting aspect of this ROM is the special programming language built into it. This enables routines to be written to control the word processor. In addition to the main text area there are ten 'segment' areas each of which can store text, or routines which are called by pressing Shift and one of the red function keys. The excellent reference manual gives a few examples but more complex routines can be easily written once you have a basic grasp of the language. This short series of three articles is intended to provide that initial understanding. Some useful hints will be given, and the text will be illustrated with worked examples.

It is first necessary to understand that the commands of the language can either be written as a program, in a segment, or typed directly from the main menu preceded by a colon. Thus if we wanted to move the cursor to the bottom of the text it could be done by executing the following in a segment:

```
SELECT TEXT
CURSOR BOTTOM
DISPLAY
```

or by typing :CURSOR BOTTOM <Return> from the menu. Any colon command must end with a <Return> but this will not normally be shown in these articles. Similarly you will not normally be reminded that a colon command must be typed from the menu page.

The ability to issue commands from the menu is very useful when you are writing routines. You have absolute control and can check the effect of each command as it is issued.

HINT: Try issuing commands (except REPEAT loops etc) one at a time from the menu if you are having trouble fault-finding.

Probably the most often-used command is CURSOR, followed by another instruction. Place some sample text into the main area, go back to the menu and type:

```
:CURSOR BOTTOM
:CURSOR UP 2
:CURSOR LEFT 5
```

Go back to the text, and you will see just how precisely the cursor position can be controlled. When moving it to a particular position on a line, the CURSOR AT command is used, thus CURSOR AT 10 will place the flashing cursor underneath the tenth character. The only, but important, problem is with CURSOR AT 39. Because of the 'wrap-around' effect that stops words being split over lines, a blank area may be left on the right hand edge of a line on the screen. CURSOR AT 39 places the cursor under the last actual character (which may of course be a space or a Return character) and not necessarily on the rightmost character position on the screen.

HINT: CURSOR AT 39 places the flashing cursor under the last character, whether at position 39 or not.

As a very simple first example, let us count how many characters a particular line has. To do this we will use the GCT$ command (Get Character from Text). The character 'got' will be the one above the cursor, and the cursor will then move one position to the right. Never forget that final part. This, at different times, can be both invaluable and a nuisance.

Enter segment 0 and type in the following lines (ending each with Return):

```
SELECT TEXT
CURSOR AT 0
A%=0
REPEAT
A$=GCT$
A%=A%+1
UNTIL A$=CHR$13
DISPLAY
```

Go back to the main text area and clear it. Type in a line of 34 characters then press Return and then a few more characters. Place the cursor anywhere on the upper line, and press Shift-f0 to run the segment program. After a brief pause you will see the cursor move to the start of the 2nd line. If you now (from the menu) type :PRINT A% the answer 35 (the count includes the Return at the end of the line) will be displayed.

What has happened? The cursor has started from the beginning of the line and repeatedly placed a GCT$ character into the variable, A$, and increased A% by 1. You can't tell it to stop the loop when it finds a Return character, as such, but by finding Return's ASCII code of 13. Hence the line, UNTIL A$=CHR$13. If you want to test this, type :PRINT ASC(A$) from the menu and you will see the number 13 displayed. Similarly f1 (green embedded start) is CHR$2 and f2 (white embedded end) is CHR$7. So if you want to count characters until the cursor encounters the start of an embedded command, it can be easily done by changing the line to:

UNTIL A$=CHR$2.

HINT: In situations where you can't use |R for the Return try CHR$13. Similarly use CHR$2 for f1 instead of |G and CHR$7 for f2 instead of |W

When the routine finds the Return character it adds one more to A% (giving 35) then stops. Finally the DISPLAY puts us back into edit mode.

The technique of collecting characters using REPEAT-UNTIL loops is relatively slow, but is used time and again in routines. Try inserting a Z into the upper line of text, and altering the appropriate line to UNTIL A$="Z" (note firstly the quotation marks, and secondly that if you use a lower case z it will not work. To trap either you would put UNTIL A$="Z" OR A$="z").

Whilst the above example worked, it would be extremely tedious to have to type a Return at the end of every line to be counted. Fortunately there is a little known memory location that can spring to our aid. ?&7E holds the current horizontal cursor position. Place the cursor anywhere on a text line, then type (from the menu):

```
:CURSOR AT 39
:PRINT ?&7E
```

and the amount returned will be the actual final character position, which may not necessarily be 39. This memory location is extremely useful and is used time and again. However, it is worth noting that this is not a very 'legal' method. There is no guarantee that this location will have the same function in future issues of Wordwise Plus.

HINT: Calling ?&7E memory location will return a value equivalent to the horizontal position of the cursor.

If you want to see the routine running, insert the command DISPLAY after the A$=GCT$ line. You will see the cursor moving along as it goes through the REPEAT loop, stopping when it finds the Return. To slow it down even more add the line IF GET THEN just above the A%=A%+1. This will 'single-step' the routine. Press the space bar each time the screen freezes.

HINT: when developing routines, the lines
DISPLAY
IF GET THEN
will allow examination of the cursor position and the routine's effect so far, before continuing.

The simple routine discussed so far could be used to transfer text very slowly from the main area to, say, segment 7 by adding a couple more lines so that the routine looks like this:

```
SELECT TEXT
REPEAT
A$=GCT$
SELECT SEGMENT 7
TYPE A$
SELECT TEXT
UNTIL EOT
CURSOR TOP
DISPLAY
```

Note the EOT command - 'End of Text', to end the REPEAT loop - and the CURSOR TOP command, which restores you to the top of the text in the main area.

There is, however, another command associated with getting characters from text, GLT$ (Get Line from Text). This will

place as many characters as possible into a variable up to either the first Return encountered, or until the variable has 255 characters in it. This means that the transfer of text mentioned above can be achieved in a fraction of the time using the following routine:

```
SELECT TEXT
REPEAT
A$=GLT$
SELECT SEGMENT 7
TYPE A$
SELECT TEXT
UNTIL EOT
DISPLAY
END
```

If you run this routine, you will find the resulting text in segment 7 is a little strange - all the Returns from the original are missing from the copy. This is because GLT$ stops at a Return (or after 255 characters). We can account for this by inserting the following after the TYPE A$ line:

```
IF LEN(A$)<255 THEN TYPE "|R"
```

If GLT$ stops before 255 it is because of a Return. A Return is inserted in the copy (with the |R) if the length of A$ is less than 255. The only case where this might not be accurate would be the very final line of the text. If this was important the following could be inserted after the UNTIL EOT line:

```
SELECT SEGMENT 7
IF LEN(A$)<255 THEN DELETE LEFT
SELECT TEXT
```

HINT: When using GLT$, remember that it does not collect the Return character which may need to be inserted separately.

As it stands, the transfer will commence from the current cursor position. If you want the entire text copied, insert as the 2nd line:

```
CURSOR TOP
```

so now the routine would look like this:

```
SELECT TEXT
CURSOR TOP
REPEAT
A$=GLT$
SELECT SEGMENT 7
TYPE A$
IF LEN(A$)<255 THEN TYPE "|R"
SELECT TEXT
UNTIL EOT
SELECT SEGMENT 7
IF LEN(A$)<255 THEN DELETE LEFT
SELECT TEXT
CURSOR TOP
A$=""
DISPLAY
END
```

The A$="" line illustrates good practice - to empty any string variables used. This is important, particularly with the GLT$ command. There are only just over 600 bytes of memory available for variables, and this amount can be very quickly filled, in which case the GLT$ command won't work, and the error message NO $ ROOM will appear. If you experience this, then you must clear the offending variables by typing, say, :D$="" from the menu.

HINT: Always try to clear variables, particularly $ variables, after use, to keep memory space empty.

Finally, for this month, let's use the ?&7E location information to delete a complete line, with the following routine:

```
SELECT TEXT
CURSOR AT 39
DELETE LEFT ?&7E
DELETE AT
DISPLAY
```

The cursor is placed on the rightmost character and ?&7E returns a value of, say, 34 so then 34 characters are deleted to the left of the cursor. The character above the cursor is still there, so this is removed with DELETE AT.

As an exercise for next month, you may like to alter this routine a little. How could the routine be developed so that, after deleting the text, the routine checks to ensure that the two lines now joined (above and below the line deleted) are separated by one space? Without this modification you will find that sometimes the lines are joined together with either no space, or two spaces. Think about that one for now. I will give you the answer in the next issue.

— HINTS — for Wordwise Plus users

During the course of development of BEEBUGSOFT's Wordease, many useful hints and tips were discovered. We bring some of these together here, for expert and beginner alike.

VDU CODES

Wordwise Plus accepts VDU codes in a very similar way to Basic. They may be strung together in the normal way and used to print out characters on the screen that could not otherwise be produced. Try the following segment program.

```
CLS
DOTHIS
VDU 129,157,131,141
PRINT "Double height in colour."
TIMES 2
IF GET THEN END
```

This will clear the screen and print a double height banner in yellow on a red background.

VDU 31 can be used to very good effect in a similar manner to position text on the screen. VDU31 is equivalent to the Basic TAB(x,y). For example:

```
CLS
X%=1
REPEAT
VDU31,X%*3,X%*2,134
PRINT "hello"
X%=X%+2
UNTIL X%>10
IF GET THEN END
```

However, be very careful not to execute a VDU 22 (select screen mode) command from Wordwise Plus. This can cause all your text and programs to be lost.

SOUND

Sounds can be produced from within Wordwise Plus using the VDU7 (beep) command. The pitch and duration of the note can be changed using *FX commands, 213 and 214. For example, to produce a one-second high-pitched note use:

```
*FX213,200
*FX214,20
VDU7
```

To control the pitch or the duration with a variable, OSCLI must be used:

```
N%=0
REPEAT
OSCLI"FX213,"+str$(N%)
*FX214,2
VDU7
N%=N%+5
UNTIL N%>250
```

The voice and the amplitude/envelope of the note can also be changed using *FX211 and *FX212, respectively. However, note that you cannot alter envelopes from within Wordwise Plus so if you select an envelope (using *FX212) then the envelope must be defined from Basic before entering Wordwise Plus.

FILENAME PRINTING

To automatically print the last-used filename within your text, use the following embedded command:

<f1> PS F$ <f2>

Where <f1> is the green code produced by pressing function key 1. This will print the filename when the text is either previewed or printed.

LABEL DUPLICATION

It is easy to duplicate label or procedure names when writing a long segment program, with disastrous results. To avoid a clash, test a proposed name before using as follows:

Press Escape to leave you in the menu of the segment containing your development program

Type :GOTO newname

If the error message, 'No such label' is produced then the name is okay. Otherwise it has already been used.

WORDWISE HELP AND PRIZE COMPETITION

The Wordwise Plus language is a fascinating one to use, and we hope that these hints and the series of articles that we are currently running tempt you to experiment with it.

To assist you in your explorations we have set up a panel of 'experts' who will try to answer any problems you have with Wordwise Plus. We will feature a selection of your questions (with answers!) in coming issues, and we are offering a prize of £50 for the most novel Wordwise Plus program submitted before November 30th.

1st course

Print Formatting (Part 1)

Correctly formatting numbers and text on the screen often causes problems for newcomers to programming. In the first of two articles on this subject, J. Pike looks at the help provided by BBC Basic.

Whether it is on the screen or to a printer, the correct formatting of data can often seem to be a hit or miss affair. The solution would seem to be the use of the special variable @% but a first reading of the User Guide about this can be most off-putting. However, the use of @% has much to offer when properly understood.

The explanation of format control using @% on pages 70 and 325 of The User Guide gives a good introduction to the facilities available, but it is incomplete. You will be surprised by some of the unexpected, and usually undesirable, formats which crop up. In addition determining the @% value to give a particular format can be very difficult.

USING @%

The format control variable @% is a special variable that controls the printing (or display) of numbers. @% is a four byte integer similar to the 'resident' integers A% to Z%. Its value remains unchanged by running a program or editing, but it is reset to a default value of &90A when Break is pressed. (In Basic I this default value of @% is &A0A. However, this acts the same as @%=&90A, the first A being interpreted as 9 because it is not in the permitted range of 1 to 9).

Byte 4 (B4) STR$	Byte 3 (B3) Type	Byte 2 (B2) Digits	Byte 1 (B1) Field width
00 Default	00 General	01-09 Digits	00-FF Width
01 @%	01 Exponential	01-09 Decimals +1	
	02 Fixed	00-09 Decimals	

Each of the four bytes that make up @% controls a particular function of print formatting. These are shown in the table and will be considered individually.

BYTE 1

The B1 byte determines the field width for printing a number. The print line is divided into fields B1 characters wide into which successive numbers are printed. Numbers and strings have different rules governing where they are printed.

Numbers are printed right justified in the next unused field, whereas strings ignore the fields and are printed consecutively. For example, with a field width of 5 (@%=&5):

```
PRINT 1 2 3
PRINT "1" "2" "3"
    |    |    |
    1    2    3
123
```

(| denotes the (invisible) field boundaries).

Numbers can be made to print like strings by using a semi-colon as a separator. However, the first number is still printed to be right justified in the first field. A semi-colon has no effect on the strings.

```
PRINT 1;2;3
PRINT "1";"2";"3"
    |    |    |
    123
123
```

Similarly, strings can be forced into the next free field with a comma as a separator, though they are still left justified. This has no effect on the numbers.

```
PRINT 1,2,3
PRINT "1","2","3"
    |    |    |
    1    2    3
1    2    3
```

All the numbers can be made to print as do strings by preceding the whole line with a semi-colon. Any other separator following (like a comma) will cancel this

effect. Again, the semi-colon has no effect on the strings but the comma forces the last string into the next field.

```
PRINT ;1 2 3,4
PRINT ;"1" "2" "3","4"
   |   |   |
123     4
123  4
```

A comma preceding all the items to be printed is ignored by both numbers and strings.

```
PRINT ,1 2 3
PRINT ,"1" "2" "3"
   |   |   |
   1   2   3
123
```

Problems arise when a number uses more characters in its representation than there are in the field width. Then the field is completely filled and the extra characters overflow into the next field. Thus even one too many characters causes all the subsequent printing to be one field further right. This can cause havoc when setting out numbers in tables.

One way of preventing this is to use the TAB command. TAB(X) positions the cursor X characters along the current line and accounts for the Return when the line exceeds the screen width. Numbers following TAB(X) are printed starting at position X but can be made to print in the next field by following TAB(X) with a comma. Problems can arise with TAB(X) when X<POS. It is better then to use TAB(X,Y) which moves directly to position X,Y. It then re-allocates the print fields starting at the current position and, unlike TAB(X), prints the next number right justified in the first field.

BYTE 4

The B4 byte is of no interest unless the STR$ function is to be used. STR$ converts a number into an equivalent string of characters. If the B4 byte of @% is set (B4=1), then the string created will follow the format currently defined by the other bytes of @%. If B4 is zero then the string created will follow the default format regardless of the current value of the other @% bytes.

BYTE 3

B3 controls the format in which a number is printed. A General, Exponential or Fixed format is used depending on whether B3 is 0, 1 or 2. Values of B3 greater than 2 give a general format.



GENERAL E.g.: 10.

The general format aims to give a reasonable representation of any number. This is the default format. Numbers below 0.1 are printed in exponential form.

EXPONENTIAL E.g.: 1E1.

The number is expressed as a multiple of a power of ten. The first number (the 'mantissa') is the multiple. The number following the E (the 'exponent') is the power of ten. 2.3E-1 is therefore equivalent to 0.23.

FIXED E.g.: 10.00.

The number is printed with a specified number of decimal places and with the decimal point in a fixed position.

BYTE 2

The B2 byte controls the accuracy to which the number is printed. In General and Exponential formats, this byte controls the maximum number of significant figures to which a number is printed. In General format the number of significant figures is the same as the total number of digits in the number. In Exponential format B2 is similarly equal to the number of digits in the mantissa. In General or Exponential format values outside the range 1 to 9 default to 9.

Although B2 influences the length of the number, it is with the accuracy that this byte is primarily concerned. When B2 is larger than the number of digits needed to represent the number then less digits are printed. For example, for all values of B2 greater than 2, the number 12.6 is printed as 12.6 in General format and 1.26E1 in Exponential format. When B2 is less than the number of significant figures the number is truncated. If B2=2, then 12.6 is printed as 13 or 1.3E1, with the least significant figure being rounded as appropriate. If B2=1 then it is not possible to represent 12.6 in General format, and it is then printed as 1E1 in General format and 1.E1 in Exponential format.

For numbers that are either very large or very small, the General format reverts to Exponential. This can cause the number of characters used to represent the number

1st course

to be larger than anticipated. For example, with the default format (@%=&90A), we might expect a maximum of 11 characters for any number (9 digits plus the decimal point and negative sign). However, a very small negative number will revert to Exponential format and be printed with many more (9 significant figures, a negative sign, the 'E', the exponent and the sign of the exponent!). It can easily take up 15 characters.

This can have serious implications when controlling the positioning of numbers using the B1 field width byte in, say, a table. Numbers can easily overflow their fields.

To cause, say, 0.01 to be printed as 0.01 and not as 1E-2 we use the Fixed format. In Fixed format, B2 controls the number of decimal places. Unlike General and Exponential formats, however, trailing zeros are not suppressed in Fixed format. With B2=2 the number 1 is printed as 1.00. If the number requires more than 9 decimal places the number of decimal places reduces until there are 9 digits, and the number prints in the default format. As for the General format, very large numbers revert to Exponential format, using a maximum of 14 characters. The range of B2 in Fixed format is 0 to 9. B2=0 causes numbers to be followed by a decimal point but no decimal digits!

The correct use of @% will enable you to deal with just about any situation requiring the tidy presentation of numbers. However, there is a need for a printing format which is more flexible, easier to specify, and which cannot overflow its allotted space. In part 2, we shall look at a format with these and other useful properties, similar to the PRINT USING command to be found in other versions of Basic.

POINTS ARISING POINTS ARISING POINTS ARISING POINTS

VIEW PRINTER DRIVER GENERATOR (BEEBUG Vol. 4 No.3)

As a result of renumbering, the line numbers of the '£' patch printed in the magazine are out by 10 and should be numbered from 1202 to 1208 and not as printed. The coding itself is quite correct.

PRETTY LIST (BEEBUG Vol.3 No.9)

Mr.N.Smith of Stoke-on-Trent has found that when this utility is used with assembler programs containing comments following a "/", the text drifts to the right of the page or screen. This apparently results when the utility finds a character which corresponds to a Basic token. Mr.Smith offers the following to prevent this:

```
  205 IF C%=&5C THEN PROCbslash
    :
 1670 DEF PROCbslash
 1680 REPEAT:C%=?I%
 1690 IF C%=&5C AND ?(I%-4)<>&D PRINT'SPC5;:FOR J%=1 TO S%:PRINT SPC1;:NEXT J%
 1700 IF C%<&80 PRINT CHR$(C%); ELSE A%=C%:CALL token
 1710 I%=I%+1:UNTIL C%=&D
 1720 ENDPROC
```

ASSEMBLER ARITHMETIC (BEEBUG Vol.3 No.8)

John Bank of Ang's Hotel, Brunei (yes, BEEBUG goes everywhere) has discovered that this program does not correctly handle a zero result if the destination variable is real, nor does it detect division by zero. The problems occur in the two sign routines sig1 and sig2, which do not recognise the special format of a floating point zero (five zero bytes). It can be corrected with the following changes:

```
2080 .sig1:LDX#4:LDA#0:.detzero:ORAfac1,X:DEX:BPLdetzero:TAX:BEQzero
2085 LDA#80:EORfac1-2:AND#&80:EORfac1+1:STAfac1+1:.zero:RTS
2090 .sig2:LDX#4:LDA#0:.detz2:ORAfac2,X:DEX:BPLdetz2:TAX:BEQzero
2095 LDA#&80:EORfac2-2:AND#&80:EORfac2+1:STAfac2+1:RTS
```

Eprom Programmer Project
(Part 2)

Alan Webster and Geoff Bains continue our project on constructing an EPROM programmer by describing the essential software to program your own EPROMs.

When you have finished building the EPROM programmer hardware (described in the previous issue of BEEBUG) you will need the following program to use it to program your own EPROMs. If you have any sideways software (that you have been using in sideways RAM, for example) then you can 'blow' an EPROM with this straight away. Otherwise you can use the programmer to copy or study existing ROMs. However, note that all commercial ROMs are copyright and should not be copied. We will be looking at writing your own sideways software in the next issue.

As the program is fairly lengthy and it requires 16k of memory space to store an EPROM in RAM, disc users will need to set PAGE to &1200 for this program. Type PAGE=&1200 before you CHAIN the program.

The program is extremely user friendly. However, make sure that the programmer is plugged in correctly before you start. Accompanying the main menu, there will be a 'test state' printed. The 'test state' will inform you of an EPROM's presence in the programmer, if it is blank or not, and the size and name of any non-blank EPROM present. This test state is continuously updated as long as the menu is displayed.

The main menu provides six options:

1. Move the present EPROM contents into memory and save it to cassette or disc under the specified filename.
2. Load the specified file into memory and blow it into EPROM. This takes a little over 7 minutes for an 8k and 14.5 minutes for a 16k EPROM.
3. Display the EPROM in a 'dump' format. Press the Shift Key to continue displaying the EPROM's contents. To exit: press Escape.
4. Verify the present EPROM against a file on cassette or disc. This is most useful to check a programmed EPROM or a saved EPROM file.
5. Leave the EPROM programmer software.

*. Execute a command line (any * command).

To copy an existing ROM or EPROM, then, all you have to do is to insert the ROM to be copied, select option 1 to save the contents to a cassette or disc file, insert a blank EPROM of the correct size, and select option 2, entering the filename used, when prompted. You may like to verify the resulting EPROM against the file saved. Use option 4 for this, entering the filename when asked.

To inspect a ROM or EPROM, insert it carefully in the programmer, select option 3 and study the dump on the screen at leisure. Shift displays the next screenful of the ROM and Escape will return you to the main menu.

That completes the facilities to produce your own EPROMs and to study others. In the next issue of BEEBUG we will look at how to write the sideways ROM programs to fill them with.

PROGRAM NOTES

Lines 140 to 160 form the main program loop. The main procedures are described below in order of appearance.

PROCinit	Set up variables and define break key to de-select 21 volts.
PROCmenu	De-select 21 volts, print menu and wait for input.
PROCtest	Test size and state of EPROM. 0=not present, 1=blank and 2=not blank.
PROCstar	Issue * command.
PROCquit	Select mode 7 and end.
PROCassemble	Set up the machine code (see later for a full description).
PROCsave	Save EPROM to file.
PROCersize	No EPROM to save.
PROCdispsave(x)	If x=0 then move EPROM into memory and execute PROCsave. If x=1 then display EPROM contents.



PROCprogram	Program EPROM from file (see later for full explanation).
PROCverify	Verify EPROM against file.
PROCromt	Display title of EPROM.

ASSEMBLER NOTES

PROCassemble sets up the following variables:

DDRA/DDRB	Data Direction Registers at &FE63 / &FE62.
PCR	Peripheral Control Register at &FE6C.
IORA/IORB	Input / Output Registers at &FE61 and &FE60.
bytes/bytes2	Test State of EPROM buffer.
LO8/HI6	14 bit address stores.

The Input / Output Registers are the two locations (for ports A and B of the User VIA) through which data is passed as input or output, from or to, the programmer. The Data Direction Registers control which direction this data is to go. The Peripheral Control Register is a general register which has just four states which concern us here, as follows:

CA2 hi,CB2 lo	1 1 0 0 1 1 1 0	= &CE
CA2 hi,CB2 hi	1 1 1 0 1 1 1 0	= &EE
CA2 lo,CB2 hi	1 1 1 0 1 1 0 0	= &EC
CA2 lo,CB2 lo	1 1 0 0 1 1 0 0	= &CC

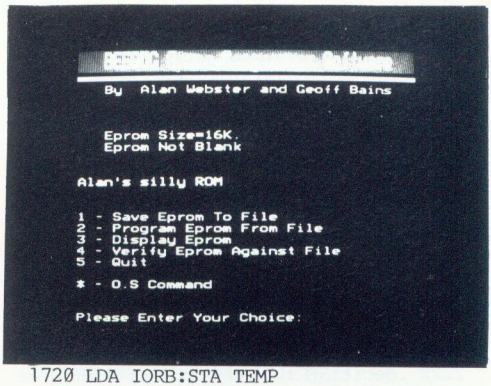
CA2 is connected to the EPROM's NOT program line and CB2 to the latch of the latch chip (see Aug/Sept BEEBUG for more information on these two lines).

The assembler code can be broken down into the following subroutines:

1650-1730	Read one byte of data.
1740-1780	Read title of EPROM.
1790-1820	Read bytes 0-3 and &2000-&2003. Store in 'bytes' and 'bytes2'. bytes=0 EPROM not present. bytes=&FFFFFFFF EPROM blank bytes=bytes2 8k EPROM. bytes<>bytes2 16k EPROM.
1830-2090	Display/Move EPROM to memory. ?&76=1 EPROM is read and displayed. ?&76=0 EPROM is read and moved down into memory.
2250-2430	Program EPROM from the data in memory.
2440-2450	Wait routine. Provides a delay of 50ms to program EPROM.

```
  10 REM PROGRAM EPROM PROGRAMMER
  20 REM VERSION B1.7
  30 REM AUTHOR  Alan Webster
  40 REM BEEBUG  OCTOBER 1985
  50 REM PROGRAM SUBJECT TO COPYRIGHT
  60 :
 100 MODE7
 110 PROCassemble
 120 PROCinit
 130 ON ERROR GOTO 1410
 140 REPEAT
 150 PROCmenu
 160 UNTIL 0
 170 END
 180 :
1000 DEFPROCinit
1010 OSCOM=&7B00:V%=0
1020 *K.10 O.|M?&FE61=1|M
1030 ENDPROC
1040 :
1050 DEFPROCmenu:VDU23,1,0;0;0;0;26
1060 ?&FE61=255:H%=21:*FX21
1070 PROCtitle:PROCtest:PRINT':*FX200
1080 PRINTCHR$131"1 - Save Eprom To Fil
e"'CHR$131"2 - Program Eprom From File"'
CHR$131"3 - Display Eprom"
1090 PRINTCHR$131"4 - Verify Eprom Agai
nst File"'CHR$131"5 - Quit"
1100 PRINT'CHR$134"* - O.S Command"''
1110 PRINTCHR$133"Please Enter Your Cho
ice:";
1120 G$=INKEY$(5):H%=H%+1
1130 IF G$="" PROCtest:GOTO1120
1140 G=ASC(G$)-48:IFG=-6 G=6
1150 IF G<1ORG>6 PROCtest:GOTO1120
1160 VDU31,26,16:PRINTG$'
1170 ON G GOTO 1180,1190,1200,1210,1220
,1230
1180 PROCdispsave(0):ENDPROC
1190 PROCprogram:ENDPROC
1200 PROCdispsave(1):ENDPROC
1210 PROCverify:ENDPROC
1220 PROCquit:ENDPROC
1230 PROCstar:ENDPROC
```

```
 1240 :
 1250 DEFPROCtest
 1260 CALL mctest:IF !bytes=-1 test=1 EL
SE IF !bytes=0 test=0 ELSE test=2
 1270 size=0:IF !bytes=-1 OR !bytes=0 TH
EN 1290
 1280 IF !bytes2=!bytes size=&20 ELSE si
ze=&40
 1290 PRINTTAB(3,2)CHR$134"Eprom Size=";
size/4;"K. "'TAB(3,3)CHR$134"Eprom ";
 1300 IF test=0 PRINT"Not Present";ELSE
IF test=1 PRINT"Blank";ELSE IF test=2 PR
INT"Not Blank";
 1310 PRINTSPC(10);:IF H%>20 H%=0:PROCro
mt
 1320 ENDPROC
 1330 :
 1340 DEFPROCstar
 1350 VDU42:INPUTLINE""$OSCOM:X%=OSCOM M
OD 256
 1360 Y%=OSCOM DIV 256:CALL &FFF7
 1370 PROCkey:*FX200 1
 1380 IFHY=ASC("*") PRINT:GOTO1350
 1390 ENDPROC
 1400 :
 1410 ON ERROR OFF
 1420 CLOSE#0:IF ERR=17 THEN 120
 1430 MODE7:*FX200
 1440 REPORT:PRINT" at line ";ERL
 1450 END
 1460 :
 1470 DEFPROCtitle:CLS
 1480 FORF%=1TO2:PRINTTAB(0,F%)CHR$129CH
R$157CHR$141CHR$131"BEEBUG Eprom Program
mer Software   "CHR$156:NEXT:PRINTTAB(0,
3)CHR$147;STRING$(38,CHR$172)
 1490 PRINTTAB(3)CHR$133"By  Alan Webste
r and Geoff Bains"
 1500 VDU28,0,24,39,6
 1510 ENDPROC
 1520 :
 1530 DEFPROCquit
 1540 VDU22,7:END
 1550 :
 1560 DEFPROCassemble
 1570 DDRA=&FE63:DDRB=&FE62
 1580 PCR=&FE6C:bytes=&80:bytes2=&84
 1590 IORB=&FE60:IORA=&FE61
 1600 LO8=&70:HI6=&71
 1610 TEMP=&73
 1620 DISPWIDTH=32:Esize=&88
 1630 FOR PASS=0 TO 2 STEP2
 1640 P%=&7100:[OPT PASS
 1650 .read:LDA #&FF
 1660 STA DDRA:LDA #&CE
 1670 STA PCR:LDA#255:STA DDRB
 1680 LDA#192:STA IORA:LDA LO8:STA IORB
 1690 LDA #&EE:STA PCR
 1700 LDA HI6:AND #&3F:ORA #&40
 1710 STA IORA:LDA #0:STA DDRB
```




```
 1720 LDA IORB:STA TEMP
 1730 RTS
 1740 .title:LDA#0:STA HI6:LDA#9:STA LO8
 1750 LDY#38:LDA#32:.ylp:JSR&FFEE:DEY:CP
Y#0:BNE ylp
 1760 LDA#31:JSR&FFEE:LDA#1:JSR&FFEE:LDA
#5:JSR&FFEE
 1770 LDY#0:.rdlp3:JSR read:BEQ endt:JSR
&FFEE:INC LO8:INY:CPY#38:BNE rdlp3
 1780 .endt:RTS
 1790 .mctest:LDA#0:STA LO8:STA HI6
 1800 LDY#0:.rdlp:JSR read:STA bytes,Y:I
NC LO8
 1810 INY:CPY#4:BNE rdlp:LDA#0:STA LO8:L
DA#&20:STA HI6
 1820 LDY#0:.rdlp2:JSR read:STA bytes2,Y
:INC LO8:INY:CPY#4:BNE rdlp2:RTS
 1830 .getdata:LDA #0
 1840 STA &8C:STA &8D:LDA#DISPWIDTH:STA&
8B
 1850 LDA &76:BEQ getlp
 1860 LDA#12
 1870 JSR &FFEE:LDA#14:JSR &FFEE
 1880 .getlp:CLC:LDA Esize:CMP &8D:BEQ r
eadend
 1890 LDA &76:BEQ over
 1900 LDA&8B:CMP#DISPWIDTH:BEQ num
 1910 .bck:JSR decit
 1920 .over
 1930 LDA &8C:STA LO8
 1940 LDA &8D:STA HI6
 1950 JSR read
 1960 LDA &76:BNE dsp
 1970 LDY #0:CLC:LDA &8D:ADC #&30:STA &7
8
 1980 LDA &8C:STA &77:LDA TEMP:STA (&77)
,Y
 1990 JMP ok2
 2000 .dsp:LDA TEMP
 2010 CMP #127:BCS dot
 2020 CMP #32:BCS ok
 2030 .dot:LDA #46
 2040 .ok:JSR &FFEE:.ok2
 2050 BIT&FF:BMI escp:CLC
```

```
2060 LDA &8C:ADC #1:STA &8C
2070 LDA &8D:ADC #0:STA &8D
2080 CLC:CMP #&40:BNE getlp
2090 .readend:LDA#15:JSR&FFEE:RTS
2100 .num:JSR&FFE7:LDA #130:JSR&FFEE
2110 LDA #38:JSR &FFEE
2120 LDA &8D:JSR shift
2130 LDA &8D:AND#&F:JSR disp
2140 LDA &8C:JSR shift
2150 LDA &8C:AND#&F:JSR disp
2160 LDA #134:JSR &FFEE:JMP bck
2170 .shift:AND#&F0:LSRA:LSRA
2180 LSRA:LSRA
2190 .disp:CLC:CMP#10:BCC nmb
2200 CLC:ADC#7:.nmb:ADC#48
2210 JSR &FFEE:.escp:RTS
2220 .decit:SEC:LDA&8B:SBC#1:STA&8B
2230 CMP#0:BNE ret:LDA#DISPWIDTH:STA&8B
2240 .ret:RTS
2250 .program:LDA #0:STA &8C:STA &8D
2260 LDA #255:STA DDRA:STA DDRB
2270 LDA #128:STA IORA:LDA#50:STA&74:ST
A&75:JSR wait
2280 .prglp:LDA #&CE:STA PCR:LDA#134:JS
R&FFEE
2290 LDA #38:JSR &FFEE:LDA &8D:JSR shif
t:LDA &8D:AND#&F:JSR disp:LDA &8C:JSR sh
ift:LDA &8C:AND#&F:JSR disp:LDA#13:JSR&F
FEE
2300 LDA &8C:STA IORB
2310 LDA #&EE:STA PCR
2320 LDA &8D:AND #&3F:ORA #&80
2330 STA IORA:CLC
2340 LDA &8D:ADC #&30
2350 STA &78:LDA &8C:STA &77
2360 LDA #0:STA &74:LDA #49:STA &75
2370 LDY #0:LDA (&77),Y:STA IORB
2380 LDA #&EC:STA PCR:JSR wait
2390 LDA #&EE:STA PCR:CLC
2400 LDA &8C:ADC #1:STA &8C
2410 LDA &8D:ADC #0:STA &8D
2420 CMP Esize:BNE prglp
2430 RTS
2440 .wait:DEC &74:BNE wait
2450 DEC &75:BNE wait:RTS
2460 ]:NEXT:ENDPROC
2470 :
2480 DEFPROCsave:IF size=0 PROCersize:E
NDPROC
2490 INPUT'"Filename: "F$
2500 A$="save "+F$+" 3000 +"
2510 G=&100*size:A$=A$+STR$~G
2520 $OSCOM=A$:X%=OSCOM MOD 256
2530 Y%=OSCOM DIV 256:CALL &FFF7
2540 ENDPROC
2550 :
2560 DEFPROCersize
2570 PRINT"?? No Eprom To Save ??":VDU7
:PROCkey
2580 ENDPROC
2590 :
2600 DEFPROCdispsave(L%)
2610 ?&76=L%
2620 ?Esize=size:CALL getdata
2630 IF L%=0 AND V%=1 ENDPROC
2640 IF L%=0 PROCsave:ENDPROC
2650 PROCkey
2660 ENDPROC
2670 :
2680 DEFPROCprogram
2690 CALL mctest:IF !bytes<>-1 VDU7:END
PROC
2700 VDU12:PRINT'CHR$131"Filename :";:I
NPUT""F$:PRINT'':*FX200 1
2710 A%=5:X%=0:Y%=&73
2720 F%=&7200:$F%=F$
2730 !&7300=F%:CALL &FFDD
2740 lngth=!&730A AND &FFFF
2750 IF lngth>&4000 PRINT"File too long
";:PROCkey:ENDPROC
2760 IF lngth>&2000 size=&40 ELSE size=
&20
2770 IF lngth=0 PRINT"File Empty";:PROC
key:ENDPROC
2780 D$="LOAD "+F$+" 3000"
2790 $OSCOM=D$:X%=0:Y%=OSCOM DIV 256
2800 CALL &FFF7:?Esize=size
2810 PRINTCHR$130"Ready To Go (Y/N):"'
2820 REPEAT:S=(GET AND 223)
2830 UNTILS=89 OR S=78
2840 IF S=78 ENDPROC
2850 CALL program
2860 ENDPROC
2870 :
2880 DEFPROCkey
2890 PRINT''CHR$131"Press Any Key To Re
turn To Menu";:HY=GET
2900 ENDPROC
2910 :
2920 DEFPROCverify
2930 V%=1:CLS:PROCdispsave(0)
2940 PRINT'CHR$130"Filename:";:INPUT""F
$
2950 B%=OPENIN(F$):E%=0
2960 FORS%=1TO (size*&100):PRINTCHR$13;
"&";~(S%-1);
2970 IF BGET#B%<> S%?&2FFF E%=E%+1:PRIN
TCHR$133"Verify Error at &";~(S%-1)
2980 NEXT:CLOSE#0:V%=0
2990 IF E%=0 PRINT''CHR$134"No Errors."
ELSE PRINT''CHR$129;E%;" Error(s)."
3000 PROCkey
3010 ENDPROC
3020 :
3030 DEFPROCromt
3040 IF test<>2 PRINTTAB(1,5)STRING$(38
,CHR$32);:ENDPROC
3050 VDU31,1,5
3060 CALL title
3070 ENDPROC
```

Loan repayments

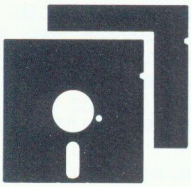
Most of us have to borrow money from time to time. In the short program listed here, Sheridan Williams offers a flexible and easy alternative to the usually tortuous calculations.

Many of us, from time to time, find the need to apply for a loan of some kind. In many cases we may not be sure how much to borrow, or what the repayments will be, or indeed over what period of time the loan should be repaid. This program takes all the hard work out of the calculations involved yet provides great flexibility.

When you run the program, answer the questions with the figures you have in mind, or if you're not sure just press Return. The program will calculate the missing information, and also display a table of repayments if requested.

Two types of loan can be considered: 'variable' where the interest is based on the amount still outstanding, and 'fixed' where the interest is based on the initial sum borrowed, as with many personal loans. In both cases, the program will answer all those 'what if...' questions, and very quickly too. However, the figures are only intended for guidance, though they are reasonably accurate in most instances.

```
   10 REM Program REPAY
   20 REM Version 1.0B
   30 REM Author Sheridan Williams
   40 REM BEEBUG October 1985
   50 REM Program subject to copyright
   60 :
  100 MODE 7:@%=&2020A:VDU14
  110 FOR I=1TO2:PRINTTAB(6,I)CHR$131;CH
R$141"Interest repayments":NEXT
  120 VDU28,0,24,39,4,12
  130 A=FNinput("Amount borrowed",TRUE,T
RUE)
  140 R=FNinput("Repayments",A,TRUE)
  150 IF A=0 AND R=0 THEN 140
  160 REPEAT
  170 I=FNinput("Annual interest rate (%
)",FALSE,FALSE)
  180 UNTIL I>=0
  190 y=FNinput("Number of years loan to
 run",NOT(A=0 OR R=0),FALSE)
  200 IF (A=0 OR R=0) AND y=0 THEN 190
  210 REPEAT
  220 n=FNinput("Number of repayments pe
r year",FALSE,FALSE)
  230 UNTIL n>0
  240 PRINT"Variable or Fixed: ";
  250 REPEAT:type%=GET:UNTIL INSTR("VvFf
",CHR$(type%))
  260 PRINT CHR$(type%AND223):fix%=((typ
e%AND16)=0)
  270 nr=y*n:i=I/(100*n):il=i+1
  280 IF i=0 i2=y*n ELSE i2=(il^nr-1)/(i
*il^nr)
  290 IF fix% i2=nr/(1+I*y/100)
  300 PRINT''
  310 IF A=0 A=FNamount:PRINT"Amount bor
rowed = £ ";A
  320 IF R=0 R=FNrepayment:PRINT"Repayme
nt = £ ";R'"Total repaid = £ ";nr*R
  330 IF y=0 y=FNnumberofyears:nr=y*n:PR
INT"Number of years = ";y
  340 PRINT'"Table of repayments (Y or N
)? ";:table=(GET AND 223)=89
  350 IF NOT table THEN 120
  360 VDU12,26:PRINTTAB(4,24)CHR$130"Pre
ss Shift to continue";:VDU28,0,22,39,4
  370 PRINT"Amount borrowed ";A:IF fix%
A=nr*R
  380 PRINT SPC4"Payment"SPC2"To pay"SPC
4"Repayment"
  390 FOR x=1 TO nr
  400 IF fix% N=A-R ELSE N=A*il-R
  410 PRINT x,N,A-N
  420 IF x MOD n=0 PRINT"_______ End of
year ";x DIV n;" _______"'
  430 A=N:NEXT x:PRINT':REPEAT UNTIL INK
EY(-1):GOTO120
  440 END
 1000 DEF FNamount=R*i2
 1010 DEF FNrepayment=A/i2
 1020 DEF FNnumberofyears
 1030 IF R-A*i<0 THEN PRINT"NEVER REPAID
":=10000
 1040 IF fix% THEN =A/(R-i*A)/n
 1050 =LOG(R/(R-A*i))/(n*LOG(il))
 1060 DEF FNinput(text$,flag,£%)
 1070 PRINT text$;:IF flag PRINT" (or Re
turn)";
 1080 IF £% PRINT" £"; ELSE PRINT" :";
 1090 INPUT " " reply
 1100 =reply
```



Double Density

Disc Filing Systems Compared

Much has changed since we first looked at double density disc filing systems back in January 1984 (BEEBUG Vol. 2 No. 8). We asked David Janda to take a fresh look at the whole issue.

Readers who use the standard Acorn DFS (or one of its single density rivals) will be well aware of the limitations that these DFSs have. There are limited file numbers, limited total storage capacity and a lack of speed. A DDFS can, to an extent, overcome most of these problems.

The most annoying problem of the existing Acorn DFS is its limit of only 31 files on one side of a disc. DDFSs can overcome this problem by providing some form of extended catalogue, offering 62 files, or more. 40 track drives have a storage capacity of only 100K per disc surface, and 80 track only 200K. This can also be limiting, especially when data files are used regularly. A DDFS can provide 60% to 80% more storage per side, but 'double' density is a bit misleading!

The speed improvement offered by DDFSs is due to the data being packed more densely on each track. Whereas an ordinary DFS stores data at 10 sectors per track, a DDFS uses 16 or 18 sectors per track. As it is not possible to control the speed at which the drive spins, it is necessary to read/write information that much quicker.

However, readers looking only for a speed improvement need not buy a DDFS. The new Acorn DNFS 1.20 provides a dramatic speed improvement over its predecessor. Similarly Watford's 62-file catalogue may be adequate for your needs.

The level of difficulty of installation of a DDFS depends on what issue board your Beeb has. The DDFS kits all comprise a disc controller board (DCB), an EPROM containing the DDFS itself, and various other TTL chips and header plugs. Fitting a DDFS is similar to fitting a standard DFS. If you own an issue three, or earlier, Beeb it may be necessary to do some soldering.

For the purpose of the review each DDFS was tested on a BBC model B (issue four) with OS 1.2, Basic II, and a sideways ROM board. A 20K Ram card was also fitted to see if there were any problems as far as space is involved.

WATFORD DDFS 1.53

Existing Watford DFS users will have no trouble adapting to Watford's DDFS, as the two products are almost identical in operation. The lack of any new commands for the double density version does not degrade the DDFS in any way - it's still an excellent product.

The disc controller used is the 1770 (as in the B+) which is mounted on the disc controller board (DCB) with three other chips and a few discrete components. Also supplied in the kit are four other chips and a 16K EPROM containing the DDFS.

Documentation supplied with the kit included a 112 page spiral bound DFS/DDFS manual, four page addendum sheet and an A4 sheet of fitting instructions. The DFS/DDFS manual is written for users of Watford's DFS and DDFS, and clearly points out any differences where applicable.

Installing the kit is easy enough if you know what you're doing. The fitting instructions lack detail, and there is no advice on what to do if things go wrong. That said, the DCB is very well made. The extended 40 pins which insert into socket 78 are rigidly supported, and a 40 pin sacrificial socket at the end prevents damage to the socket.

The DDFS provides 80% more storage capacity when used in double density mode. Depending on the number of files selected (31 or 62), 0.5 or 1K will be used up by the directory. The extended file option as well as the density are prompted for when formatting a disc.

As well as offering the standard set of DFS commands, the Watford DDFS provides a number of enhancements and additional commands. Wildcards can be used in *LOAD and *RENAME. With *LOAD the first file in the catalogue will be loaded. A very handy feature is retention of information on the drive, library, directory, density and others over a soft Break.

Extra star commands include MLOAD and MRUN. These enable machine code files to be loaded over the disc work space. *HELP FILES displays information on any opened files, and *HELP SPACE displays free disc space together with the sizes of any 'gaps' on the disc.

In use, Watford's DDFS proved to be very reliable and no compatibility problems were encountered. The reason for this (claims Watford) is an extensive 8271 emulator within the DDFS EPROM which simulates the single density controller. In practice this would appear to work well, as all the protected software ran without trouble.

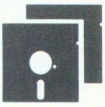
UDM DDFS 3.10

United Disc Memories has had its DDFS on the market for quite some time. The product has recently been improved and offers a number of features not found in the early versions.

UDM has opted to use the 2793 disc controller chip and unlike most other DDFSs only sixteen sectors per track are used. This means that the double density storage capacity for a 40 track disc is 160K, and 320K for 80 track.

The DCB is a low profile board which is supplied with a 40 pin sacrificial socket. Other parts in the kit include six chips, a header plug and the DDFS EPROM. Also supplied is an eight way DIP switch to set timings for disc access and so on.

UDM also supplied its 100% board for review. This item is mounted in socket 78 and includes two sockets: one for the standard 8271 controller, and the other for the DCB. A flying lead with two-way switch is attached to the 100% board. The board enables switching between the standard Acorn DFS and UDM's DDFS thus giving 100% compatibility.



Installing the DCB and supporting chips is easy enough, but the DIP switches require soldering. One header plug has no directional notch, and needs some guesswork as to which way it should be inserted!

The DDFS can be used in one of three modes: single density, double density or double density with multiple catalogues. The format command requires the drive, density, track and title to be specified, and if double density is chosen the user is asked if extended catalogues are required.

The effect of selecting the extended catalogue mode is to create four catalogues on each side of the disc. Each catalogue is assigned 63.5K and can store 30 files, thus giving a total of 120 possible files per disc. A special locked file called &.&&&&&&& is automatically saved to each catalogue when formatting, and this must not be erased at any cost.

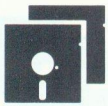
A catalogue is selected using the VOLUME command which is followed by the drive and volume letter (A to D). A nice feature is the ability to automatically save to another catalogue if the current catalogue is full. Loading is facilitated by searching through each catalogue until the correct file is found.

LVL DOS 1.01

The LVL DDFS has also been around for some time. Version 1.01 now includes a formatter and verifier within the DDFS.

	Watford	UDM	LVL	Cumana	Opus
Price	£97.75	£95	£124.95	£69.95	£99.95
Sectors	10/18	10/16	10/18	10/18	10/18
Maximum files	31/62	31/120	31/62	31	31/248
Max file size	disc	255K	255K	255K	252K
Width (mm)	55	60	61	51	45
Length (mm)	79	66	83	81	80
Height (mm)	34	22	27	19	21
Offset (mm)	33	22	42	7	5
Fit with ATPL ROM board	Y	N	Y	Y	Y
Software Compatibility	9	9	3	8	6

Note: Maximum files is the number per side of a disc, not per directory.
The offset is the distance the DCB protrudes to the right of the righthand edge of socket 78.
Nine protected packages were tried on each DDFS. These were Banjax, Castle Quest, Elite (early version), Jet Power Jack, Labyrinth, Magic Mushrooms, Rubble Trouble, Sorcery, and The Mine. The figure given is the number of packages that successfully ran.



When used in its double density mode, the LVL DDFS provides 179K and 359K of usable storage for 40 and 80 track drives respectively.

The kit contains nine chips, the DCB and the 16K DDFS EPROM. Documentation was in the form of a 34 page manual with a four sided addendum sheet.

Fitting the kit is a little tricky. The DCB is rather large, and inserting it involves bending (very gently) one of the power leads. The pins on the DCB are very weak, and are prone to bending if uneven pressure is placed on the board when inserting it. However, finding the right place to put everything is made easier by a very clear diagram in the packaging, which not only shows where each chip should be inserted, but has the chip itself stuck to that position - you can't go wrong!

The DDFS itself does not incorporate any extra commands (except format and verify), but does get round the 32 file problem in a unique way. Instead of providing 179 or 359K of continuous storage on one side of the disc, each surface (or side) is divided into two drives. This is best explained by an example. A single double-sided disc drive has drive numbers assigned like this:

Disc side	single density	double density
Top	0	0 4
Bottom	2	2 6

For dual 80 track drives in double density, eight drives are provided (numbered 0 to 7). This is very strange but it does mean that there are no extra commands or extensions for the user to learn. However, in use the LVL DDFS proved to be the slowest; and, worst of all, a lot of software would not run.

CUMANA QFS 2.0

Cumana - well known for their disc drives - have recently released the new version of their DDFS, or QFS as they call it. Unlike the other DDFSs reviewed, Cumana's DDFS does not offer any extra commands, nor does it allow extra files to be stored, yet it does have its advantages.

The QFS provides 80% extra storage capacity less 0.5K for the directory when used in double density mode. The only extra command included in the QFS is a formatter which automatically verifies.

The kit is supplied with six chips as well as the QFS EPROM. A seventh chip is attached to the DCB via a flying cable. The DCB is a low profile board which does have its disadvantages, as it's very hard to see if the pins are inserted into the socket correctly.

The formatter used in the QFS is the most comprehensive and user friendly I have ever used. The user is prompted for the drive to be formatted, the number of tracks (80, 40, and 35) and the title and number of the disc. There are options to format both sides of the disc and to format further discs without going through the instructions again. After formatting, the QFS is able to automatically detect the density and number of tracks.

In use, Cumana's QFS proved to be very reliable. It was also one of the fastest, and practically all the test software ran without any trouble.

The simplicity of the whole package makes it very nice to use, yet it could be improved by allowing for a greater number of files.

OPUS DDFS 3.45

The DDFS from Opus can best be described as 'feature packed'. Not only does it allow for up to 248 files per disc surface, but it also includes other useful utilities as well.

The disc controller chip used is the 1770, and 355K of user storage is provided. The kit consists of two chips, two header plugs, the DCB, and a 16K DDFS EPROM.

The DCB fits neatly into socket 78 and a reinforcing socket ensures rigid connection. Unlike the others, the Opus DCB is small and compact. Also worthy of note is that the majority of the board faces to the left, so the DCB shouldn't get in the way of other items (ROM boards, etc.) fitted in your Beeb.

To achieve the high number of files possible on one side, the first track is reserved by the DDFS for catalogue and

file information. No catalogue can exceed 31 files, so the Opus DDFS divides the disc into five lettered volumes when the disc is formatted in double density mode.

Up to eight volumes may be assigned, and this is done with the VOLGEN command. VOLGEN will display the current volume settings and allow the user to assign or de-assign any of the eight. However, it is necessary to *ENABLE before amending volume as this is a destructive command.

Two commands which I believe are essential in any DDFS are included: these are *40Ø and *DENSITY. The *40Ø command allows 40 track discs to be used in an 80 track drive. The command can be set to on, off or auto. Unfortunately the default is off. The *DENSITY command sets the density that the DDFS software will default to. It is usually automatically selected (which is handy), but many protected games require the density to be set to single, as this option provides.

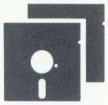
To actually 'attach' to a particular volume, the *DRIVE command is used. *DRIVE ØC, for example, would set the catalogue to volume C on drive Ø. The method of saving/loading to a particular catalogue is also simple. *LOAD :1G.$.name would load a file called 'name' from volume G on drive 1.

There are, however, a couple of problems with Opus' DDFS. It is not possible to format a disc in double density mode without the default volumes, and, more importantly, a number of packages failed to operate under this DDFS.

THE SYSTEMS COMPARED

The table of software tested will give you some idea as to software compatibility of the various systems.

As far as file capacity is concerned Opus wins hands down with an excellent implementation of the volume handling. Unlike the UDM DDFS, the extra file information is 'hidden' within the first track on the disc, and there is no way that a user can normally corrupt this data. However, UDM has managed to maintain a higher degree of compatibility with the standard Acorn DFS, even without the 1ØØ% board.



Cumana offers a much simpler, faster and cheaper system with no frills, yet a high degree of compatibility and speed. LVL's method of implementing double density is very strange, and its price is high. I don't think it will appeal to many users.

However, my own choice is Watford's DDFS. Even though it does not offer any DDFS like commands and it is only of average speed, it is reasonably priced and is packed with features that have been developed over a long period of time. The extended catalogue option of 62 files would fulfil most requirements, and together with the ability of saving files of any length (up to disc capacity) is a big plus in its favour.

THE BENCHMARKS

To put the various DDFSs through their paces, they were subjected to 13 separate benchmarks. The 13 benchmarks are a measure of the time taken to, respectively, *SAVE and *LOAD 16K, PRINT# and INPUT# 1ØØ integers, PRINT# and INPUT# 1ØØ reals, PRINT# and INPUT# 1ØØ strings, PRINT# and INPUT# 1ØØ records, randomly BPUT# and BGET# 1ØØ bytes, and reverse the order of the contents of a file.

The mean of all 13 benchmarks is given in the table for each of the DDFSs, along with that for Acorn's DFS and DNFS and Watford's DFS. A full table of all the benchmark results is available from the Editorial address on receipt of an SAE. The benchmark program is included in this month's magazine cassette/disc.

MEAN BENCHMARK RESULTS

Acorn 0.90		Acorn DNFS		Watford 1.43	Watford DDFS 1.53		UDM DDFS 3.10	
SD		SD		SD	SD	DD	SD	DD
32.19		16.40		18.66	18.27	16.75	12.99	14.27

LVL DOS 1.01		Cumana QFS 2.0		Opus DDOS 3.05	
SD	DD	SD	DD	SD	DD
19.46	18.88	13.62	14.55	15.18	14.27

Storing large quantities of text rapidly eats up the Beeb's limited memory. Surac outlines a technique that reduces the space needed by nearly half.

This month the Workshop looks at one way round the Beeb Adventurer's problem of insufficient memory. If you want to store a lot of text, the 25K or so that is normally available soon gets used up. There is, however, a way of packing text so that you can get up to 50% more in any given space.

First, though, a quick resume of how characters are stored in the Beeb and most other small computers. Each character is allocated one of 256 so-called ASCII codes in the range 0-255, and each code fits neatly into an 8-bit byte. There is also a variable overhead, of at least 5 bytes, for each string in the Beeb.

This is rather wasteful. If you're only displaying text, you don't need 256 characters and can get by with around 60: A-Z, 0-9, plus some punctuation marks. The exact mixture depends on just what text you have. Now, 5 bits can store 32 different character codes and there are 3 groups of 5 bits in two bytes.

However, if we limit ourselves to only 30 characters we can pack three into every two bytes. Unfortunately, 30 is a bit too limiting, so use 60, split them into two groups, and allocate a special code to switch between the groups. Once this is done, all that's left is to find a way of packing three 5-bit codes into two bytes. That is exactly what this first program will do.

```
1000 DEF PROCpackinit
1010 DIM Key$(1),cc%(3)
1020 DIM txtbuf 10000
1030 Key$(0)=
     " ABCDEFGHIJKLMNOPQRSTUVWXYZ.,"
     +CHR$(13)
1040 Key$(1)=
     " 0123456789*;/-=<>()'?$£_;:.,"
     +CHR$(13)
1050 ENDPROC
2000 DEF FNcode(str$,locn%)
2010 LOCAL bitptr%,chcode%,chptr%,
     code%,kptr%,lenstr%,ptr%
2020 chptr%=1
2030 lenstr%=LEN(str$)
2040 REPEAT
2050   chcode%=FNcd(MID$(str$,chptr%,1))
2060  code%=code% OR
       chcode%*2^(10-bitptr%*5)
2070   bitptr%=(bitptr%+1) MOD 3
2080   chptr%=chptr%+1
2090   IF bitptr%=0 THEN PROCpack
2100   UNTIL chptr%=lenstr%+2
2110 IF bitptr% THEN PROCpack
2120 =locn%+ptr%
3000 DEF FNcd(ch$)
3010 LOCAL cc%
3020 IF chptr%=lenstr%+1 THEN =0
3030 cc%=INSTR(Key$(kptr%),ch$)
3040 IF cc% THEN =cc%
3050 kptr%=(kptr%+1) MOD 2
3060 IF INSTR(Key$(kptr%),ch$) THEN
     chptr%=chptr%-1:cc%=31 ELSE
     kptr%=(kptr%+1) MOD 2:cc%=1
3070 =cc%
4000 DEF PROCpack
4010 locn%?ptr%=code% AND &FF
4020 locn%?(ptr%+1)=code% DIV &100
4030 ptr%=ptr%+2
4040 code%=0
4050 ENDPROC
```

PROCpackinit sets up the system, reserving a buffer area to hold the compressed data - in this example 10000 bytes, but that's up to you. It also sets

up coding strings in Key$(0) and Key$(1). The two strings here are good general-purpose ones, but you can easily change them to suit your own needs.

Very common punctuation, e.g. space, comma and Return, appears in each. This can help the compression, as there is less need to switch between coding sets. The common characters are in the same place in each string. Also, the two strings each hold a maximum of 30 characters. Although we can code 32 different values into 5 bits, zero is reserved to mark the end of a string while 31 is the shift code.

FNcode() packs str$ into the buffer starting at location locn%. Line 2060 packs the characters, in threes, into code% with FNcd() extracting the code of each in turn. If the character is not in the currently selected coding string, FNcd checks the other; if it finds it, it sends back the shift code (31), otherwise it sets a space. When it reaches the end of the string, FNcd returns a zero.

As every three characters are coded, PROCpack puts their coded values into the buffer. When it reaches the end of str$, any outstanding codes are also stored. On exit, FNcode returns the next free address in the buffer. This suits the best way of setting up a set of compressed strings, as in the following pseudo-Basic:

```
1 PROCpackinit
2 P%=address of start of buffer
3 REPEAT
4   READ a string to code
5   P%=FNpack(string,P%)
6   UNTIL last string coded
7 *SAVE buffer to tape/disc
8 END
```

Don't try to run that! You could read the strings to be packed from DATA statements, the keyboard, or from a tape or disc file; the last is probably the most likely. At the end of the process, your backing store will hold a copy of the compressed text all ready to be unpacked by whatever program is going to use it. If you're writing lots of programs, you'll probably have a standard compression program, to use every time you need it.

The unpacking routines will normally be part of a completely different program from the one which compressed the data.

Before you run the procedure you must first have run PROCpackinit and loaded the compressed data into the buffer. Then, by calling PROCdecode() with the start address of the packed string, the data will be expanded and displayed on the screen. I leave you to choose the best way of identifying the start of the particular code you need.



```
5000 DEF PROCdecode(locn%)
5010 LOCAL cn%,code%,i%,kptr%,ptr%
5020 REPEAT
5030   PROCunpack
5040   FOR i%=1 TO 3
5050     cn%=cc%(i%)
5060     IF cn%>0 AND cn%<31 THEN
         PRINT MID$(Key$(kptr%),cn%,1);
5070     IF cn%=30 THEN VDU10
5080     IF cn%=31 THEN
         kptr%=(kptr%+1) MOD 2
5090     NEXT
5100   UNTIL cc%(1)=0 OR cc%(2=0) OR
       cc%(3)=0
5110 ENDPROC
6000 DEF PROCunpack
6010 code%=locn%!ptr% AND &FFFF
6020 cc%(1)=code% DIV &400
6030 cc%(2)=(code% AND &3E0) DIV &20
6040 cc%(3)=code% AND &1F
6050 ptr%=ptr%+2
6060 ENDPROC
```

PROCdecode makes repeated calls to PROCunpack, which reads the next two bytes from the buffer and splits their contents into the corresponding 5-bit codes in cc%(). The codes are then used to extract data from Key$(), shifting between the two character sets every time the code 31 is encountered. As soon as the procedure reaches a zero code it exits as that is the end-of-string marker used.

The decoding program must have exactly the same strings in Key$() as the original packing program. If the strings are different, the most amazing gibberish may appear. There is little point in using these routines unless you need to store a lot of text. They can never quite reach the 50% improvement mark, but get very near if you use a long string which needs as few shift codes as possible.

A complete demonstration of these routines is contained on this month's magazine cassette/disc.



by Mitch

ADVENTURE GAMES ADVENTURE GAMES

This month, the Dungeon Master answers a few enquiries from lost and trapped adventurers.

In the game 'Twin Valley Kingdom' the instructions hint that 'an old, wise man may look upon the secret of life that lies nearby'. The object of the game is to finish with as much wealth as possible then solve the last problem - what to do when you have the maximum amount of points? I have found the Secret of Life and now have the maximum score but I can't find an old, wise man to get him to look at it. Please can you tell me if there is such a man and where I can find him.

Dominic Surman-Wells

To have solved all the problems of this fiendish game and to have found the Secret of Life must have taken a long time and means that by now YOU must be pretty old and wise. It seems likely, therefore, that no better old, wise man should LOOK AT THE SECRET than yourself.

ADVENTURE GAMES ADVENTURE GAMES

I can spare a few moments before I return to feed the dragon. Can you help me with Sphinx Adventure by Acornsoft.
(a) How do you remove the jack from the pit?
(b) How do you cross the Glacier bridge?
(c) How do you get out of the coloured maze and iron passages?

May your sword never go rusty,

David Barnes

Foolish Mortal, the answer to (a) and (b) is close at hand - in fact on your hand, within the Mithril Ring! Only he who wears the ring may cross the bridge or solve the problem of the pit. Remember, waving and rubbing objects is always a good idea.

'But wait!', I hear you moan, 'Where is this accursed ring?' Did your wise old mother never tell you that only fairies wear rings. Go seek the ring in the Fairy King's Grotto, and remember to give him a nice WAVE.

ADVENTURE GAMES ADVENTURE GAMES

I am blind and play text-only adventures via a speech synthesizer (Votrax 'Type and Talk'). I heartily agree with your sentiments regarding mazes! Fearful things, and I am stuck in the Black Maze in 'Castle of Riddles' by Acornsoft. Please give me a clue on what to do. I am sad to hear that more and more adventures are being written with graphics to give location descriptions. It means that I will not be able to play them. I hope that you find it of interest that blind players enjoy adventures as much as anyone.

David Calderwood

Happily the 'Castle' mazes all have clever solutions to their twisty paths. In your case the answer lies with the Metal Rod. Like its predecessor in 'Colossal', this object must be waved. You'll soon have a solution to your problem which like all good fairy stories lies somewhere just by rainbow's end. I should mention that this game cannot be solved without cheating! It appears to be impossible to use the rod for solving the problem of the maze and the Bear's Sitting Room in the same game. Unless a clever troll knows differently...

During my travels I have met a number of blind programmers who have found that the micro has enabled them to compete at work and play in a way I never believed possible.

ADVENTURE GAMES ADVENTURE GAMES

In Bug-Byte's 'Twin Kingdom Valley' how do you get the master key off the Dragon in the North Tower of the Castle?

Philip Macilroy

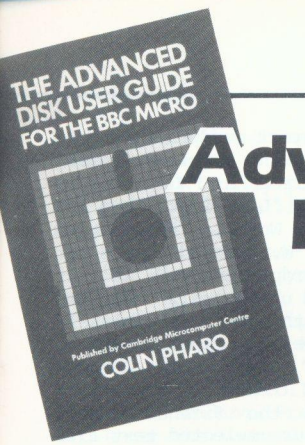
Same way as I get my slippers back from my Dragon - Smash him with a wooden staff.

ADVENTURE GAMES ADVENTURE GAMES

In Program Power's 'ADVENTURE', how do you kill the rat? We've tried everything!

R.G. Petersen

I don't really give a hoot about this problem, but I suggest that you do. Get yourself near the rat and give a loud HOOT. This foolishness will call a hungry owl from the nearby forest to knock the stuffing out of the rodent.



Advanced Disc User Guide

The Advanced Disc User Guide is the latest in a series of highly acclaimed reference books from the Cambridge Micro Computer Centre. James Fletcher has been taking a look at the latest offering.

The Advanced Disc User Guide, Cambridge Microcomputer Centre, at £14.95.

It is rare to find a technical tome that is written in such an enjoyably readable style as this 446-page Advanced Disc User Guide. I found the first few pages so interesting that I just had to keep on reading. The introduction claims that the book is meant to be read from start to finish, rather than just dipped into as a reference work. Although you would need to be very dedicated to read the whole thing at once, it does make sense to initially skim right through it so as to get some idea of the wealth of 'goodies' that it contains.

The word 'Advanced' in the title is most definitely justified, as some of the concepts used would only be of use to the experienced machine code programmer. However, the book contains such clear and straightforward explanations of the workings of the BBC micro's disc system that it would be well worth a place on the bookshelf of any disc user who wants to know more about the system than the rather meagre amount of information he is offered in Acorn's Disc System User Guide.

The author has succeeded in explaining the workings of the fairly complex 8271 floppy disc controller chip in a remarkably clear manner; although much of the information can be found on the 8271 data sheets it is far from easy to work out what is happening with just the manufacturer's information, and the book uses simple but useful operational flowcharts to help the reader's understanding.

The lengthy but very detailed DFS memory map will appeal to those who just have to know what goes where, and the introduction to the filing system routines and vectors is cleverly done, with each description of a low-level DFS call being accompanied by a fully-documented machine-code example program. Although commands such as OSBGET (read a single byte from a specific file) and OSBPUT (write ditto) are primarily for internal use by the DFS and the machine operating system, you can learn a great deal by using these functions for yourself.

An excellent example of how to build up a random access filing system is given, and the detailed description of a stamp collection catalogue is ideally suited to the beginner, with each step being clearly explained. It is useful to have details of all the various Acorn DFS error messages, although I usually find these less than helpful when something really goes wrong, and the book gives good advice on error prevention and data protection.

Some of the disc utilities provided in the book are both interesting and useful, and I particularly liked the combined 40/80 track formatting program (see also BEEBUG Vol.4 No.4), and the examples of disc overlay techniques which apparently give your computer more memory. The pros and cons of the Acorn DFS are fully discussed, and some of the snags of choosing a non-Acorn DFS are pointed out. Although these can give significant advantages, such as unrestricted numbers of files on each disc, the advice 'regard alternative filing systems as a risk' ties in with my own experiences of some of these products.

All in all then, this is a most useful and interesting book. Perhaps not for the absolute beginner, but for anyone who wants to learn about any aspect of the BEEB's disc system this book is a must. If you can raise the £14.95 - buy it !

This book is available on special offer to BEEBUG members - see price list for full details.

Further Disc Menu Extensions

David Andrews' extended disc catalogue program, described in the June issue of BEEBUG (Vol. 4 No. 2), has provoked considerable feedback. We have put the best of these ideas together to extend this useful utility even further.

The extended disc catalogue program in BEEBUG Vol.4 No.2 provides a menu of your Basic programs on a disc along with a short description of each obtained from a REM statement at the start of each program. Among the many letters of praise for this program that we have received are a number giving further extensions to it. We have taken the best four ideas and present them here as separate additions to the previous program.

New members who have missed the original program can buy a back issue (see page 50) or you can obtain a photocopy of the original article and program by sending an SAE and 30p to the Editorial address. Please mark your envelope 'DISCAT photocopy'.

This article provides three extensions to the original program:

1. To allow it to recognise Wordwise and View text files and to load these into the wordprocessors.
2. To recognise *SWAP (Disc Doctor) format discs.
3. To read 62 files on a Watford DFS.

As each part is separate, you need only choose the extensions that you require. Note that the line numbering of the extensions assumes that the numbering of the original program has been left unaltered. The Wordwise and View extensions are combined as one, and this is the extension that we describe first.

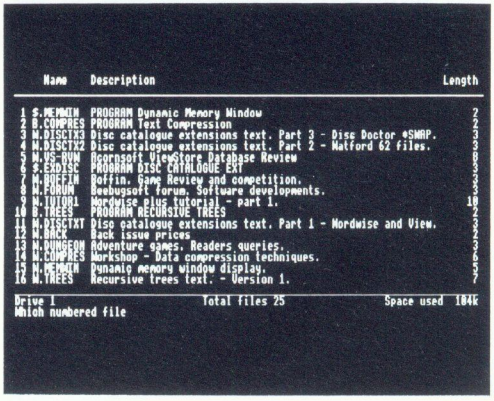
WORDWISE and VIEW

To load text into Wordwise and View, we have to set up a new procedure that will enter the correct wordprocessor and load the text file in the required manner. For View, this requires a *WORD to be executed, followed by 'L abcd' to Load the text file 'abcd'. For Wordwise the program needs to execute a *W. command then enter ':N.' into the wordprocessor, so if Wordwise Plus is in use we will clear the text from memory (although it is unlikely that there will be any there) and in Wordwise this will answer 'N' to the question 'Old Text (Y/N)'. We then enter '2' (Load text) and the filename. This will then load your selected text file into Wordwise.

The following lines are the amendments and additions required.

```
 240 PTR#N%=4:F%=BGET#N%:IFF%=244 OR
     F%=81 OR F%=124 THEN 250 ELSE 370
 250 IFF%=244 P%(T%)=1 ELSE IFF%=124
     P%(T%)=2 ELSE IF F%=81 P%(T%)=3
 260 @%=2:NAME$(T%)=NAME$(Z%)
 451 IFP%(f%-1)=2 PROCwwise(1)
 452 IFP%(f%-1)=3 PROCwwise(2)
1470 DEFPROCwwise(z)
1471 IF z=1 THEN 1480
1472 A$="*WORD"+CHR$13+"L"+NAME$(f%-1)
     +CHR$13:GOTO 1490
1480 A$="*W."+CHR$13+":N."+CHR$13+"2"
     +NAME$(f%-1)+CHR$13
1490 FORG%=1TOLENA$:Y%=ASC(MID$(A$,
     G%,1)):X%=0:A%=138:CALL&FFF4
1491 NEXT
1500 END
```

The text header that is to appear in the disc catalogue must conform to specific guidelines. For Wordwise and Wordwise Plus, the header must be the first item in the file and must start with



a green embedded command start character (function key 1), followed by two spaces, then *| and finally your title. This must all be on the same line and needs a Return at the end.

For View users, you need to start your text with a 'DM' (Define Macro) stored command. Enter this by using Shift + f8 to edit a command, then type DM followed by Return. Then type 'QQ' to define your macro as 'QQ' and follow that immediately by your heading. Press Return after this and enter the command 'EM' for End Macro. This now completes the extensions for the two Wordprocessors.

DISC DOCTOR *SWAP

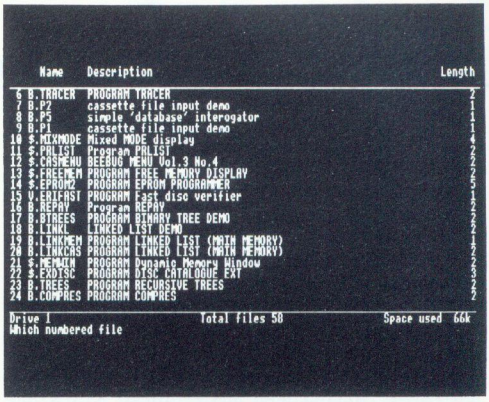
The extensions to allow the disc menu to work with *SWAP catalogues, created with Computer Concept's Disc Doctor, are given below. A single disc surface is treated as two separate discs. After the program prompts you for the disc to be catalogued, it will also ask if this is a *SWAP format disc. If it is then successive catalogues of the disc will alternately produce the two *SWAP catalogues on the disc.

```
 225 IFNAME$(Z%)="!.!!!!!!!" THEN
     V%=V%+512:GOTO370
 385 IF ?BUF%=42 U%=BUF%?1 ELSE U%=32
 400 PRINT"Drive ";D%;" ";CHR$U%;:
     IFU%<>32 PRINT" (*SWAP format)";
1020 PRINTTAB(27,8)"Catalogue of
     Programs"TAB(27,9)STRING$(21," ")
     TAB(28,11)"Current Drive";SPC(3);
     D%;" ";CHR$U%:Y$=""
1030 IFU%<>32 PRINTTAB(28,13)"Swap
     Files (Y/N)";SPC(2);:Y$=GET$:
     IFY$="Y" PRINT"Yes"
1035 IFY$="Y" PROCoscli("SWAP"):ENDPROC
     :ELSE PRINTTAB(28,13)"New drive"
     SPC(7);:drive%=GET-48
```

To run this program on *SWAP discs, you must title (using *TITLE) one 'half' catalogue with a name starting with '*A' and the other 'half' with one starting with '*B'. The extended program must be saved on both 'halves' of the *SWAP format disc, drive 0. With this amendment, the program will run with both *SWAP and standard format discs.

WATFORD DFS 62 FILES

The appropriate amendments and additions to allow the menu to cope with a Watford DFS 62 file catalogue are as follows.




```
 220 FORZ%=0TOR%-1
 360 IFT%=18 OR T%=36 OR T%=54 VDU7:
     PRINT'".. Any key to continue ..":
     IFGET:VDU11:PRINTSPC(27):VDU11,11
 410 PRINTTAB(32)"Total files ";(R%+S%)
     TAB(63)"Space used ";(V%+500)
     DIV1000;"k"
1080 DIMBUF%&400,readcat&50
1160 ?P%=D%:P%!1=BUF%:P%?5=3:P%?6=&53:
     P%?7=0:P%?8=0:P%?9=&24:P%?10=0
1210 S%=0:R%=(BUF%?&105)/8:IF BUF%!&200
     =&AAAAAAAA S%=(BUF%?&305)/8
1220 IF R%<1 PRINT'''"No Suitable files.
     "'"Press any key.":G=GET:RUN
1230 DIMNAME$(R%+S%-1),P%(R%+S%-1)
1240 FORZ%=0TOR%-1
1310 NEXT:IF S%=0 THEN 1320
1311 BUF%=BUF%+&200:FORZ%=0TO S%-1
1312 FORY%=0TO6:Q%=R%+Z%
1313 NAME$(Q%)=NAME$(Q%)+CHR$(BUF%?
     (Y%+8*(Z%+1)))
1314 NEXT:G%=BUF%?(Y%+8*(Z%+1))
1315 IF G%>127 G%=G%AND&7F
1316 NAME$(Q%)=CHR$G%+"."+NAME$(Q%)
1317 NEXT
```

These extensions will allow the menu to read both 62 file and normal 31 file catalogues.

With all of these extensions in place, you now have one of the most comprehensive menu programs available for the BBC micro.

Many thanks are due to A.R.Webster, David Andrews, Alan Marshfield, P.J.Swan and David Graham for suggestions and help in compiling these additions.

Looking at Data structures

(Part 2)

Paul Ganney continues his look at data structures by describing how to implement linked lists, and goes on to discuss the principles of tree structures.

Last month we investigated the principles of linked lists and how they operate. Structures like linked lists are not available in languages such as Basic, and so they must be implemented in terms of arrays. The first program does this, and provides insertion and deletion facilities as well as the ability to display the linked list.

```
  10 REM LINKED LIST DEMO
  20 REM VERSION B0.1
  30 REM AUTHOR  P. GANNEY
  40 REM BEEBUG  OCTOBER 1985
  50 REM PROGRAM SUBJECT TO COPYRIGHT
  60 :
 100 MODE 7
 110 VDU23,1,0;0;0;0;
 120 DIM A(1,9)
 130 REM A(0, ) IS DATA
 140 REM A(1, ) IS POINTER
 150 REM NULL POINTER IS -1
 160 start%=-1
 170 free%=0
 180 A(1,9)=-1
 190 FORI%=0TO8:A(1,I%)=I%+1:NEXT
 200 :
 210 REPEAT
 220 PROCmenu
 230 UNTIL FALSE
 240 :
1000 DEF PROCmenu
1010 PROCheader(1,2,"LINKED LIST")
1020 PRINT'''CHR$133;"Input choice:"''CHR$133;"1. Insert an item"'CHR$133;"2. Delete an item"'CHR$133;"3. List data"'CHR$133;"4. End"
1030 REPEAT:A%=GET-48:UNTILA%>0AND A%<5
1040 IFA%=1 THEN PROCinsert ELSE IF A%=2 THEN PROCdelete ELSE IF A%=3 THEN PROC list ELSE CALL!-4
1050 ENDPROC
1060 :
1070 DEF PROCheader(front%,back%,TEXT$)
1080 LOCAL spaces
1090 VDU26,12
1100 IF LEN TEXT$>33 THEN spaces=0 ELSE spaces=(33-LEN(TEXT$))DIV 2
1110 FOR I%=0 TO 1:PRINTTAB(spaces,I%);CHR$(128+back%);CHR$157;CHR$(128+front%);CHR$141;TEXT$;"    ";CHR$156:NEXT
1120 VDU28,0,24,39,3
1130 ENDPROC
1140 :
1150 DEF PROCdelay:TIME=0:REPEAT:UNTIL TIME>100:ENDPROC
1160 :
1170 DEF PROCinsert
1180 PROCheader(3,4,"INSERT")
1190 IF free%=-1 THEN VDU7,129:PRINT"LIST FULL":PROCdelay:ENDPROC
1200 INPUT"Data item",A(0,free%)
1210 F%=A(1,free%)
1220 IF start%=-1 THEN start%=free%:A(1,start%)=-1:GOTO1310 ELSE IF A(0,free%)<= A(0,start%) THEN A(1,free%)=start%:start%=free%:GOTO1310
1230 J%=start%
1240 K%=FALSE
1250 REPEAT
1260 I%=A(1,J%)
1270 IFI%=-1 THEN K%=TRUE ELSE IF A(0,free%)<=A(0,I%) THEN K%=TRUE ELSE J%=I%
1280 UNTILK%
1290 A(1,free%)=A(1,J%)
1300 A(1,J%)=free%
1310 free%=F%
1320 ENDPROC
1330 :
1340 DEF PROCdelete
1350 PROCheader(5,6,"DELETE")
1360 IF start%=-1 THEN VDU7,129:PRINT"EMPTY LIST - NOTHING TO DELETE":PROCdelay:GOTO1450
1370 INPUT"Data item",D%
1380 I%=start%
1390 J%=-1
1400 K%=FALSE
1410 REPEAT
1420 IF I%=-1 K%=TRUE ELSE IF A(0,I%)=D% THEN K%=TRUE ELSE J%=I%:I%=A(1,I%)
1430 UNTILK%
1440 IF I%=-1 THEN VDU7,129:PRINT"ITEM DOES NOT EXIST":PROCdelay ELSE IF J%=-1 THEN S%=A(1,start%):A(1,start%)=free%:free%=start%:start%=S% ELSE A(1,J%)=A(1,I%):A(1,I%)=free%:free%=I%
1450 ENDPROC
1460 :
1470 DEF PROClist
1480 PROCheader(2,3,"LIST")
1490 IF start%=-1 THEN PRINTCHR$130;"EMPTY LIST":GOTO1550
1500 I%=start%
1510 REPEAT
```

```
1520 PRINTA(0,I%)
1530 I%=A(1,I%)
1540 UNTILI%=-1
1550 PRINT''"PRESS SPACE":REPEAT:UNTILG
ET=32:ENDPROC
```

PROGRAM NOTES

160-190 set up the free space list (at this stage the entire array).

1170-1320. Insertion. Three 'special cases' are possible. These are tested for first and acted upon should one of them have occurred. Line 1190 checks for free space. Line 1200 inputs the new item into the first free space. Line 1220 tests if the list is currently empty (start%=-1), and if the new item should be placed at the head of the list. Lines 1250-1280 trace through the list until either the correct place in the list is found or the end of the list is reached. Lines 1290-1310 adjust the pointers as was described in the first article.

1340-1450. Deletion. The list is first tested for anything to delete (1360) and is then traced through (1380-1430) until either the item is found, or the end of the list is reached. If the item has been found, then pointers are adjusted so that the item before the one to be deleted now points to the one after it, and the deleted item is added to the front of the free space list, its pointer being adjusted to point to the previous head of this list (line 1440).

1470-1550. Listing. Simply traces through the list. I% is the index (place in the array) of the item being listed. After printing the data item (A(0,I%)), I% is set to the pointer from this item (A(1,I%)) - the index of the next item. This continues until I% = -1, the terminator.

This version of a linked list, implemented using a two dimensional array, will only work for numeric data. For string data we could use separate arrays for the data and the pointers.

USING MAIN MEMORY

One disadvantage of large arrays is that pressing the Break key automatically clears the array. A better method is to store the data in main memory (adjusting HIMEM so that the O.S. doesn't overwrite it) using indirection operators (see User Guide, p.409, BEEBUG Vol.3 Nos.7 and 8). The resident integer variables (also unaffected by Break) can also assist further as the pointer variables.

For the purpose of an example, let us assume that we wish to set up a catalogue, each record consisting of an eight character identifier (name), and an integer number. This gives us fifteen bytes per record (8 plus 1 (end-of-string character) for the identifier, 4 for the integer, and 2 for the pointer). Thus, to hold 100 records, we require 1500 (&5DC) bytes and hence HIMEM must be set at 30244 (&7624) in mode 7. This section of memory will then be partitioned into the 15-byte blocks by setting all the pointers so as to form the free space list. S% and F% will be used for the start and free space pointers, respectively.

Note that, although !&7BFE (the pointer from the last item) is initially set to -1 (the terminator), all subsequent tests for terminators are for 255, due to the nature of the indirection operators. Note also that the Break key must be programmed to reset HIMEM whenever pressed, in order to preserve the data.

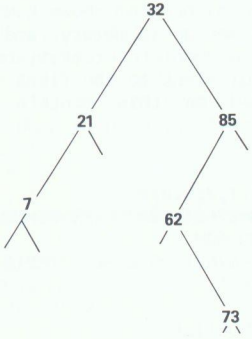
The second program shows how a linked list can be set up in memory and listed. This isn't a complete program, but a full routine, equivalent to the first program, is included on this month's magazine cassette/disc.

```
 100 MODE7
 110 VDU23,1,0;0;0;0;
 120 HIMEM=&7624:S%=255:F%=HIMEM:*K.10O
.MH.=&7624MG.60M
 130 FORI%=&7631 TO &7BEF STEP15:!I%=I%
+2:NEXT:!I%=-1
 140 :
1540 DEF PROClist
1550 PROCheader(2,3,"LIST")
1560 LOCAL P%
1570 P%=S%
1580 VDU14
1590 PRINTCHR$130;"Press";CHR$136;"Shif
t";CHR$137;"For next part of list"'
1600 IF P%=255 THEN PRINTCHR$133;"LIST
EMPTY" ELSE REPEAT:PRINT$P%,P%!9:P%=P%?1
4*256+P%?13:UNTIL P%=255
1610 VDU15
1620 PRINT''"PRESS SPACE":REPEAT:UNTIL
GET=32
1630 ENDPROC
```

BINARY TREES

Another linked data structure is the binary tree. The great advantage of the binary tree is that the data structure intrinsically reflects the ordering of the data stored in it. The binary tree has two pointers per element (or node), known respectively as the left and right pointers. The basic principle is that there exists a central node, called the root, which is (of course) at the top of the binary tree. Confused? No need to be, as the ideas are simple enough, it's just the terminology muddling matters as usual.

From the root, the binary tree spreads in two directions, left and right. Those nodes to the left are ordered before the root, those to the right after it. Moving down the binary tree to another node presents the same picture of two pointers. It is this feature that makes binary trees so useful. At any node (position in the binary tree) the structure below this point is also a binary tree (a 'sub-tree'). A node with no sub-trees is called a terminal node. The absence of a sub-tree is recorded by a terminator in the appropriate pointer, in much the same way as the end of a linked list is recorded.



At this point, an example would be useful. Consider the insertion of the number 57 into the purely numeric binary tree in the figure. The process is as follows:

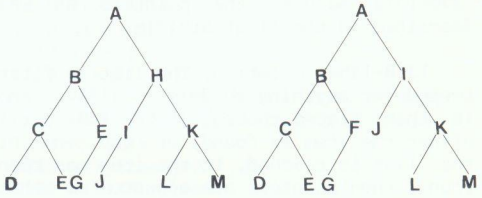
1. 57 is greater than 32, so branch right.
2. 57 is less than 85 so branch left.
3. 57 is less than 62 so branch left.
4. A terminator is reached, so place the data item 57 here, to the left of 62.

Again, you may like to try creating a binary tree from the following items (the first item inserted automatically becomes the root):

32,85,62,21,73,7.

You should end up with the original binary tree.

Having established the principles of inserting an item into a binary tree, what of deletion? This is actually a thornier problem than at first may seem to be the case. The pointer to the deleted node must be made to point to the left sub-tree of the deleted node. The right sub-tree of the deleted node becomes the rightmost element of the left sub-tree of the deleted node. If this sounds confusing (and it is!) follow the process through on the diagram.



DELETE H

POINTER TO H NOW POINTS TO I
I HAS NO RIGHT POINTER, SO THIS POINTS TO SUB-TREE, K

Listing the data in a binary tree (often called traversing the binary tree) is a simple recursive algorithm:

Traverse left sub-tree, if there is one.
Print node.
Traverse right sub-tree, if there is one.

As BBC Basic supports recursion, there is no problem implementing this routine.

Searching for an item in the binary tree is simply an application of the insertion principles, testing as we go for the required item, outputting an 'Item not found' message if a terminator is reached.

All these features are implemented in the following demonstration program:

```
10 REM PROGRAM BINARY TREE DEMO
20 REM VERSION B0.1
30 REM AUTHOR  P. GANNEY
40 REM BEEBUG  OCTOBER 1985
50 REM PROGRAM SUBJECT TO COPYRIGHT
```

```
  60 :
 100 MODE3
 110 ON ERROR REPORT:END
 120 PROCsetup
 130 REPEAT
 140 option%=FNmenu
 150 IF option%=1 THEN PROCinsert
 160 IF option%=2 THEN PROCdelete
 170 IF option%=3 THEN PROCsearch
 180 IF option%=4 THEN PROClist
 190 UNTIL option%=5
 200 MODE 3:END
1000 DEF PROCsetup
1010 LOCAL i%
1020 DIM data$(100),left%(100),right%(1
00)
1030 root%=-1:free%=0
1040 FOR i%=0 TO 100
1050 left%(i%)=-1:right%(i%)=-1
1060 NEXT i%
1070 PRINTTAB(31,2)"BINARY TREE DEMO"
1080 VDU28,0,24,79,4
1090 ENDPROC
1100 :
1110 DEF FNmenu
1120 LOCAL option%:CLS
1130 PRINTTAB(10,2)"1. Insert new item"
1140 PRINTTAB(10,4)"2. Delete item"
1150 PRINTTAB(10,6)"3. Search for item"
1160 PRINTTAB(10,8)"4. List all items"
1170 PRINTTAB(10,10)"5. Exit"
1180 PRINTTAB(10,14)"Enter your choice"
;
1190 REPEAT:option%=GET:UNTIL INSTR("12
345",CHR$(option%))
1200 =option%-48
1210 :
1220 DEF PROCinsert
1230 LOCAL D$:CLS
1240 IF free%>100 THEN PRINT'"Tree full
.":GOTO 1330
1250 INPUT''"Enter new data string (max
 6 characters): " D$
1260 IF root%=-1 THEN root%=free%:GOTO
1300
1270 found%=FNsearch(D$)
1280 IF found%>-1 THEN PRINT'D$;" alrea
dy exists in position ";found%:GOTO 1320
1290 IF turn$="L" THEN left%(lasturn%)=
free% ELSE right%(lasturn%)=free%
1300 data$(free%)=D$:free%=free%+1
1310 PRINT'D$;" now entered"
1320 PRINT'"Press any key to continue":
key%=GET
1330 ENDPROC
1340 :
1350 DEF PROCdelete
1360 LOCAL found%,P%:P%=root%:CLS
1370 IF P%=-1 THEN PRINT'"Tree empty":G
OTO 1490
1380 INPUT''"Enter data string: " D$
1390 found%=FNsearch(D$)
1400 IF found%=-1 THEN PRINT'"Not found
":GOTO 1490
1410 IF found%=root% THEN root%=left%(r
oot%):GOTO 1430
1420 IF turn$="L" THEN left%(lasturn%)=
left%(found%) ELSE right%(lasturn%)=left
%(found%)
1430 P%=left%(found%)
1440 IF root%=-1 THEN root%=right%(foun
d%):GOTO 1480
1450 IF P%=-1 THEN left%(lasturn%)=righ
t%(found%):GOTO 1480
1460 REPEAT:last%=P%:P%=right%(P%):UNTI
L P%=-1
1470 right%(last%)=right%(found%)
1480 PRINT'D$;" now deleted"
1490 PRINT'"Press any key to continue":
key%=GET
1500 ENDPROC
1510 :
1520 DEF PROCsearch
1530 LOCAL found%,P%:P%=root%:CLS
1540 IF P%=-1 THEN PRINT'"Tree empty":G
OTO1580
1550 INPUT''"Enter search string: " D$
1560 found%=FNsearch(D$):PRINT' D$;
1570 IF found%>-1 THEN PRINT" found in
position ";found% ELSE PRINT " not found
"
1580 PRINT'"Press any key to continue":
key%=GET
1590 ENDPROC
1600 :
1610 DEF FNsearch(D$)
1620 LOCAL found%,P%:found%=FALSE:P%=ro
ot%
1630 REPEAT
1640 IF D$=data$(P%) THEN found%=TRUE E
LSE lasturn%=P%:IF D$<data$(P%) THEN P%=
left%(P%):turn$="L" ELSE P%=right%(P%):t
urn$="R"
1650 UNTIL P%=-1 OR found%
1660 =P%
1670 :
1680 DEF PROClist
1690 CLS:IF root%=-1 THEN PRINT"Tree em
pty" ELSE PROClistl(root%)
1700 PRINT'"Press any key to continue":
key%=GET
1710 ENDPROC
1720 :
1730 DEF PROClistl(start%)
1740 IF left%(start%)<>-1 THEN PROClist
l(left%(start%))
1750 PRINT data$(start%)
1760 IF right%(start%)<>-1 THEN PROClis
tl(right%(start%))
1770 ENDPROC
```

Interfacing the BEEB

Eight books reviewed

With our EPROM Programmer Project we thought it timely to take a look at some of the many books now available on interfacing the Beeb. Geoff Bains has been doing all the reading.

When the BBC Micro was first launched it was much vaunted as having great potential for laboratory work and computer control. The same is true today. Some of that potential has been fulfilled by commercial devices but most is left up to the user with an interest in electronics and skill with a soldering iron.

To help out would-be DIY enthusiasts to construct their dreams, several books have come onto the market. All of these comprise three broad subjects: theory, constructional details, and supporting software. However, the relative proportions of these varies enormously, as do the topics for construction, the depth of discussion, and the skill levels assumed of the reader.

Sensing and Control Projects for the BBC micro by Thomas Nunns. £5.95 from Micro Books.

This book is the simplest of all those reviewed here. The book is divided between projects for the analogue port and those for the user port. The most complex (and appealing) of the ten projects covered is a slot car controller using simple relays and driver chips. The other projects are the like of LED traffic lights and measuring light levels.

Although the projects are, to say the least, uninspiring, the strength of this book is its constructional details. No soldering is required. Everything either screws or clips together. Large lettering, large diagrams and many sections on 'what to do if it doesn't work' mean that this book is ideal for older children or those who have never seen a piece of wire, let alone a chip, before.

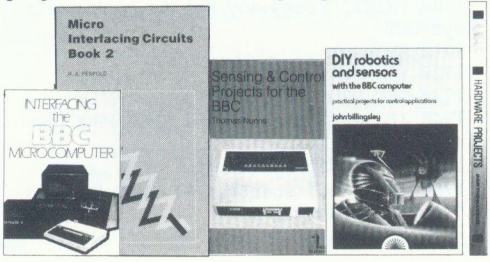
Micro Interfacing Circuits Book 1 and 2 by R.A.Penfold. £2.25 from Babani.

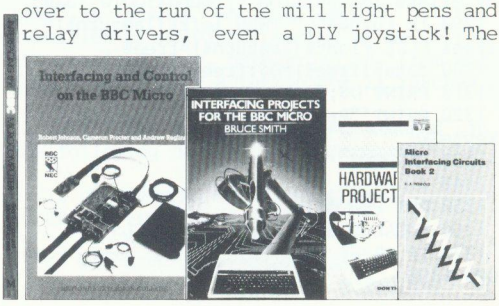
These two slim volumes are the opposite extreme from the previous book. There is next to no constructional details here, nor supporting software either. However, the books cover a wide range of topics - everything from powering LEDs to audio digital filters - and they are meant for other micros as well as the Beeb.

Only the circuits are provided along with a very brief explanation of their workings and pinouts of the various chips and transistors used. Babani books are a well known sight in component shops and their style is pretty uniform, so if you have never come across a Babani electronics book before, the chances are that these are not going to be introductory enough for you anyway.

DIY Robotics and Sensors with the BBC micro by John Billingsley. £6.95 from Sunshine Books.

This book is one of the more specialist tomes. The first half is given over to the run of the mill light pens and relay drivers, even a DIY joystick! The





second half is where the rather grandiose title takes over. It is mainly devoted to the control of stepper motors.

I get the impression that this book was written in a rush. There is a lot of theoretical background given and even 'experiments' to prove the fundamental operation of stepper motors. The result is an averaged sized book with an awful lot that you don't actually want. The diagrams, too, are just rough sketches. If you are a dedicated electronics fan with a desire to get into robotics, then this book could provide a starting point. However, don't expect too much in the way of in-depth constructional details. This book really just maps out the terrain.

Interfacing and Control on the BBC micro by R. Johnson, C. Procter, and A. Reglinski. £9.95 from NEC.

This book is essentially a text book for followers of the relevant NEC course. However, it is sold in 'normal' bookshops and so could well tempt the would-be interfacer. Although the book is fairly comprehensive it would not be recommended for most readers.

For a start all the projects are not DIY boards but based around a ready-made circuit board marketed by the NEC. In addition, the book IS a text book and the style does not encourage enthusiasm for the subject. It is also filled with questions and answers which are not really applicable outside the course. That said, the book does offer a reasonable grounding in the use of the Beeb's analogue and VIA ports. However, at nearly £10 the price alone is going to put many off.

Interfacing the BBC microcomputer by B. Bannister and M. Whitehead. £7.50 from Macmillan.

Macmillan's Beeb interfacing book can best be described as one for those well familiar with building their own electronics gadgets but have not yet tried this in the computer field.

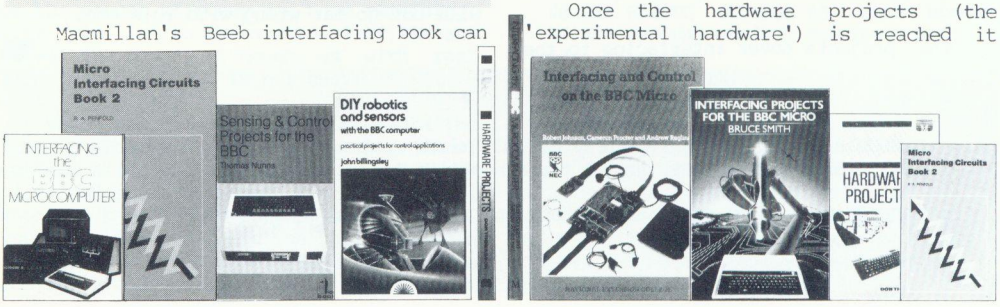
The book starts off with a detailed description of the facilities available. There is a chapter on the user port and one on the 1MHz bus, and one on the handling of analogue data. It is only by page 78 that 'some applications' are discussed. These are not so much applications as an extension of the theory with mention of some specific components. No real constructional details are given; it is assumed the reader knows all about that kind of stuff already.

By the time page 103 comes along the applications are all finished and we are back with the theory in the form of the appendices. These mainly give pages and pages of data sheets from various chip manufacturers. For a near-expert in electronics, unfamiliar with the operation and architecture of a microcomputer, this book would no doubt go down well. However, if you intend to actually build something after reading this, and you have not wielded a soldering iron that much before, let alone designed your own circuit boards, you are likely to be disappointed.

Interfacing the BBC microcomputer by Colin Opie. £8.95 from McGraw-Hill.

Colin Opie's book follows Messrs Bannister's and Whitehead's in more than just title. It too has large sections at the front (about two thirds of the 200 page book) devoted to the explanation of the detailed operation of the user port, 1MHz bus, and the Beeb in general. However, this time the whole is written in a manner that, for me, was a lot more acceptable and encouraging.

Once the hardware projects (the 'experimental hardware') is reached it



transpires that it is all based around circuit boards that can be purchased, ready made or in kit form, separately (from Watford Electronics). Granted the printed circuit board layouts are given at the end of the book, but it is unlikely that most potential buyers of this book will have the skill or the inclination to start messing around, making their own.

The hardware is advisedly called 'experimental'. There is little in the way of applications here. The boards are potentially very useful but nothing is really said on what and how they could be usefully applied.

Again this is a book for those already knowledgeable in the field. As a reference work for the electronics hobbyist the friendly style and excellent explanations of the theory of the Beeb's ports it is recommended, but the lack of real application and home appeal spoils the book's chances for most of us.

BBC Hardware Projects by Don Thommasson. £8.95 from Melbourne House.

Keeping the best for last, this is one of the two books in this bunch that can really start you off on an interesting and enlightening foray into the world of electronics.

The book starts off with a theory section. This only lasts for a quarter of the book's 200 or so pages but manages to reveal enough of the workings of the Beeb to keep you going for the rest of the book.

Then there is the constructional section. The range of projects is not very inspired but all are laid out on stripboard and so are eminently attemptable by the average Beeb owner. You don't have to be an electrical engineer to manage these.

The projects cover interfacing to the analogue and user ports. There is a light-detector, voltmeter, relay-driver, mains power controller, numeric keypad, and others. Although these may not seem to be great advances in the field of computer control, you do at least leave this book with a feeling that these things are actually possible; even you could achieve this.

The book does, however, fall down a little on the meagre supporting software. This is a pity as the combination of the excellent Beeb operating system and its interfacing abilities make the system ideal for this kind of thing. Only give half the story and you only have half the fun.

Interfacing Projects for the BBC micro by Bruce Smith. £6.95 from Addison-Wesley.

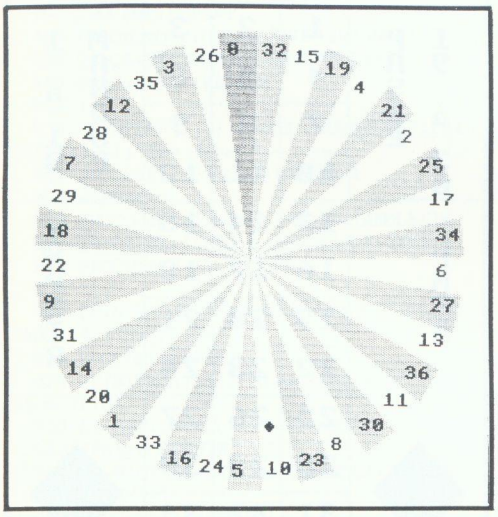
Like the previous book this one is a practical work. Some theory is given at the start (including a classic padding chapter on hex and binary!) but you soon get into the meat of the projects. These cover a good range - from light pen to EPROM Programmer (yes, other's have done it too!). Each project is accompanied by an adequate description of how the circuit works, a clear stripboard layout diagram, reasonable constructional details, and plentiful supporting software.

It is difficult to know why this book pleases as it does. The presentation is not very good. It is short (only 130 pages), the diagrams are more in the line of sketches and the program listings faint in places. However, please it does. The style is light enough to communicate the author's enthusiasm for the subject and yet it still remains very informative. There is little here for the absolute beginner, but then the book is not aimed at them. If you have a little experience with a soldering iron (or even a lot) you will not go far wrong with this one.

BEEBUG now keeps a wide variety of BBC micro related books available for mail order. Please send an SAE for a booklist to:

BEEBUG BOOK LIST, P.O.BOX 50, HOLYWELL HILL, ST ALBANS, HERTS AL1 1EX.

Roulette



Gambling into the night on the tables of Monte Carlo is no more than a dream (or nightmare) for most of us. Richard Pearce brings it all to life with his accurate and colourful implementation.

Can you break the bank at Monte Carlo? Can you turn £500 into £5,000,000 or will you lose your shirt on the world's most famous gambling table?

Roulette is a one-player game, that runs in mode 1, giving a good display of the betting table and roulette wheel. At the start of the game you are given £500 to gamble, and you can place one or more bets with each bet not exceeding £100,000 or the amount of money you have remaining. Your aim is to amass a total of £5M and 'break the bank', although as with real life roulette, this is not easy!

When run, the program presents the betting table and a list of options. These options allow you to place bets, and both view and spin the wheel. Please note that when you have completed your bet or bets, you should then select the option to spin the wheel. A brief description of the bets and their odds now follows.

Name	Description	Odds
(the following bets can include Ø)		
En Plein	Single Number	35-1
A Cheval	Two Numbers (adjacent) on the table, with vertical or horizontal. Zero can be coupled with 1, 2 or 3.	17-1
Trans-versale Plein	Horizontal row of 3 numbers with Ø being combined with any two of 1,2 or 3	11-1
Carre	Block of four numbers	8-1
(The following cannot include Ø)		
Sixaine	Two adjacent horizontal rows	5-1
Colonne	Vertical column of twelve	2-1
Douxaine	Either the first, second or third dozen numbers, i.e. 1-12, 13-24 or 25-36	2-1
Deux Douxaine	Two dozen. Either numbers 1-24 or 13-36. You must bet an even amount.	1-2

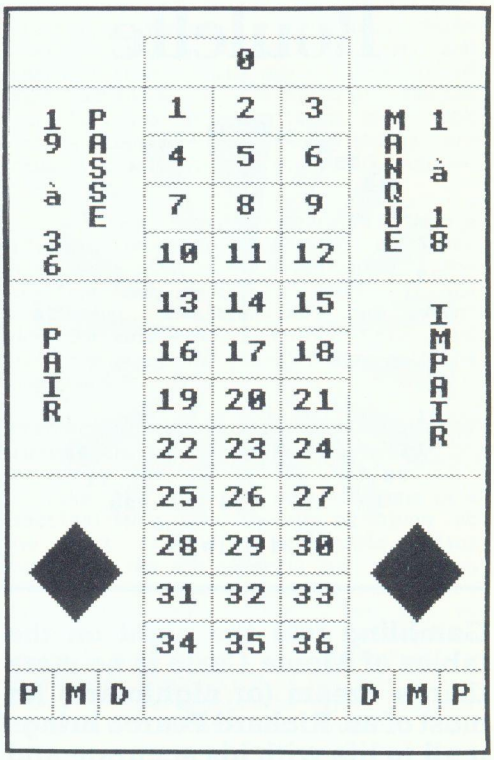
Because of the length of this program and the fact that it runs in mode 1, disc users or anyone who has page set to greater than &EØØ will have to move the program down before running. This can be achieved by using the move down routine published in BEEBUG Vol.3 No.5 or by adding the following lines to your program:

```
1 IF PA.<&E01 THEN 10
2 *K.0 *T|MF.A%=0TO(TOP-PA.)S.4
:A%!&E00=A%|PA.:N.|MPA.=&E00|M
0.|MDEL.0,4|MRUN|M
3 *FX138,0,128
4 END
```

Please take care when typing the program into your computer that no extra spaces are inserted because the program could run out of memory as a result.

```
  10 REM PROGRAM ROULETTE
  20 REM VERSION B0.1
  30 REM AUTHOR  R.PEARCE
  40 REM BEEBUG  OCTOBER 85
  50 REM PROGRAM SUBJECT TO COPYRIGHT
  60 :
  70 ONERRORGOTO190
  80 MODE1:PROCsetup
  90 REPEAT:CLS
 100 VDU19,2,2,0,0,0
 110 VDU23,1,0;0;0;0;
 120 PROCgame:VDU4,17,3
 130 CLS:IFK%=0GOTO150ELSESOUND1,2,4,50
:PRINTTAB(5,10)"YOU HAVE BROKEN THE BANK
!"TAB(9,16)"Pretty clever-huh!"TAB(9,18)
"How about lending me"
```

```
  140 PRINT TAB(9,20)"ten quid till Tues
day?":GOTO170
  150 PRINTTAB(7,11)"YOU HAVE NO MONEY L
EFT!":FORI%=30TO0STEP-1:SOUND1,4,I%,2:NE
XT:PRINTTAB(4,14)"You're a pretty awful
 gambler"TAB(4,16)"aren't you? I wouldn't
 advise"
  160 PRINTTAB(4,18)"taking your life sa
vings to":COLOUR2:PRINTTAB(4,20)"Monte C
arlo!"
  170 COLOUR1:PRINTTAB(6,24)"Do you want
 another game?":REPEAT:F%=GET:UNTILF%=78
ORF%=89
  180 UNTILF%=78:COLOUR3:END
  190 MODE6:REPORT:PRINT" at line ";ERL
  200 END
  210 :
  220 PRINTTAB(5,3);"EN PLEIN":PRINTTAB(
2,4);"Single Number"
  230 PROCbet:VDU28,23,31,39,10
  240 REPEAT:CLS
  250 INPUTTAB(1,1)"What number?"L%:UNTI
LL%<37
  260 Q%=Q%-B%:IFL%=N%H%=1
  270 PROCwin(35):RETURN
  280 :
  290 PRINTTAB(5,3);"A CHEVAL":PRINTTAB(
4,4);"Two numbers"
  300 PROCbet
  310 PROCnumber
  320 VDU28,23,31,39,16
  330 IFL%=0GOTO360
  340 REPEAT:CLS
  350 INPUTTAB(1,1)"What is the      sec
ond number?"Hi%:UNTILHi%=L%+1ORHi%=L%+3:
GOTO370
  360 REPEAT:CLS:INPUTTAB(1,1)"What is t
he      second number?"Hi%:UNTILHi%>0AND
Hi%<4:GOTO380
  370 IFL%MOD3=0ANDHi%=L%+1GOTO310
  380 Q%=Q%-B%:IFL%=N%ORHi%=N%H%=1
  390 PROCwin(17):RETURN
  400 :
  410 PRINTTAB(3,3);"TRANSVERSALE
 PLEIN":PRINTTAB(2,5);"Horizontal row"
  420 PROCbet
  430 PROCnumber
  440 IFL%=0GOTO480
  450 IFL%MOD3<>1ORL%>34GOTO430
  460 Q%=Q%-B%:IFN%>L%-1ANDN%<L%+3H%=1
  470 PROCwin(11):RETURN
  480 VDU28,23,31,39,15:REPEAT:CLS
  490 INPUTTAB(1,2)"What is the 2nd  num
ber?"L1%:UNTILL1%<3
  500 VDU28,23,31,39,19:REPEAT:CLS
  510 INPUTTAB(1,2)"What is the 3rd  num
ber?"L2%
  520 UNTILL2%<4ANDL2%>L1%
  530 Q%=Q%-B%:IFL%=N%ORL1%=N%ORL2%=N%H%
=1
```




```
  540 PROCwin(11):RETURN
  550 :
  560 PRINTTAB(6,3);"CARRE":PRINTTAB(2,4
);"Block of four"
  570 PROCbet
  580 PROCnumber:IFL%=0GOTO620
  590 IFL%MOD3=0ORL%>32GOTO580
  600 Q%=Q%-B%:IFL%=N%ORL%+1=N%ORL%+3=N%
ORL%+4=N%H%=1
  610 PROCwin(8):RETURN
  620 Q%=Q%-B%:IFN%<4H%=1
  630 PROCwin(8):RETURN
  640 :
  650 PRINTTAB(5,3);"SIXAINE":PRINTTAB(1
,4);"Two horiz. rows"
  660 PROCbet:REPEAT
  670 PROCnumber:UNTILL%MOD3=1ANDL%<32
  680 Q%=Q%-B%:IFN%>L%-1ANDN%<L%+6 H%=1
  690 PROCwin(5):RETURN
  700 :
  710 PRINTTAB(5,3);"COLONNE":PRINTTAB(1
,4);"Vertical column"
  720 PROCbet:REPEAT
  730 PROCnumber:UNTILL%>0ANDL%<4
  740 Q%=Q%-B%:FORI%=0TO11:IFL%+(3*I%)=N
% H%=1
```

```
  750 NEXT:PROCwin(2):RETURN
  760 :
  770 PRINTTAB(5,3);"DOUXAINE":PRINTTAB(
6,4);"Dozen"
  780 PROCbet:REPEAT
  790 PROCnumber:UNTILL%MOD12=1ANDL%<26
  800 Q%=Q%-B%:IFN%>L%-1ANDN%<L%+12 H%=1
  810 PROCwin(2):RETURN
  820 :
  830 PRINTTAB(1,3);"DEUX DOUXAINES":PRI
NTTAB(3,4);"Two dozen"
  840 PROCbet:IFB%MOD2=1B%=B%-1
  850 REPEAT
  860 PROCnumber:UNTILL%MOD12=1ANDL%<14
  870 Q%=Q%-B%:IFN%>L%-1ANDN%<L%+24H%=1
  880 PROCwin(0.5):RETURN
  890 :
  900 PRINTTAB(3,3);"PASSE/MANQUE":PRINT
TAB(4,4);"High/low"
  910 PROCbet:REPEAT
  920 PRINTTAB(1,6);"Press key for    ch
oice (P/M)"
  930 L%=GET:UNTILL%=77ORL%=80
  940 IFN%=0GOTO970
  950 H%=0:Q%=Q%-B%:IFN%<19ANDL%=77ORN%>
18ANDL%=80H%=1
  960 PROCwin(1):RETURN
  970 PROC0:IFN1%=0:RETURN
  980 H%=0:Q%=Q%-B%:IFN1%<19ANDL%=77ORN1
%>18ANDL%=80H%=1
  990 PROCwin(0):RETURN
 1000 :
 1010 PRINTTAB(2,3);"PAIR/IMPAIR":PRINTT
AB(3,4);"Even/odd"
 1020 PROCbet:REPEAT
 1030 PRINTTAB(1,6);"Press key for    ch
oice (P/I)"
 1040 L%=GET:UNTILL%=73ORL%=80
 1050 IFN%=0GOTO1080
 1060 H%=0:Q%=Q%-B%:IFL%MOD2=N%MOD2H%=1
 1070 PROCwin(1):RETURN
 1080 PROC0:IFN1%=0:RETURN
 1090 H%=0:Q%=Q%-B%:IFL%MOD2=N1%MOD2H%=1
 1100 PROCwin(0):RETURN
 1110 :
 1120 PRINTTAB(4,3);"NOIR/ROUGE":PRINTTA
B(4,4);"Black/red"
 1130 PROCbet:REPEAT
 1140 PRINTTAB(1,6);"Press key for    ch
oice (N/R)"
 1150 L%=GET:UNTILL%=78ORL%=82
 1160 IFN%=0GOTO1190
 1170 H%=0:Q%=Q%-B%:RESTORE1830:FORI%=1T
ON%:READE%:NEXT:IFE%=0ANDL%=78ORE%=1ANDL
%=82H%=1
 1180 PROCwin(1):RETURN
 1190 PROC0:IFN1%=0:RETURN
 1200 H%=0:Q%=Q%-B%:RESTORE1830:FORI%=1T
ON1%:READE%:NEXT:IFE%=0ANDL%=78ORE%=1AND
L%=82H%=1
 1210 PROCwin(0):RETURN
 1220 :
 1230 PROCwheel
 1240 MOVE835,100:PRINT"PRESS SPACE TO":
MOVE835,65:PRINT"RETURN TO MENU":IFGET
 1250 PROCtable:PROClist
  850 REPEAT
  860 PROCnumber:UNTILL%MOD12=1ANDL%<14
  870 Q%=Q%-B%:IFN%>L%-1ANDN%<L%+24H%=1
  880 PROCwin(0.5):RETURN
  890 :
  900 PRINTTAB(3,3);"PASSE/MANQUE":PRINT
TAB(4,4);"High/low"
  910 PROCbet:REPEAT
  920 PRINTTAB(1,6);"Press key for    ch
oice (P/M)"
  930 L%=GET:UNTILL%=77ORL%=80
  940 IFN%=0GOTO970
  950 H%=0:Q%=Q%-B%:IFN%<19ANDL%=77ORN%>
18ANDL%=80H%=1
  960 PROCwin(1):RETURN
  970 PROC0:IFN1%=0:RETURN
  980 H%=0:Q%=Q%-B%:IFN1%<19ANDL%=77ORN1
%>18ANDL%=80H%=1
  990 PROCwin(0):RETURN
 1000 :
 1010 PRINTTAB(2,3);"PAIR/IMPAIR":PRINTT
AB(3,4);"Even/odd"
 1020 PROCbet:REPEAT
 1030 PRINTTAB(1,6);"Press key for    ch
oice (P/I)"
 1040 L%=GET:UNTILL%=73ORL%=80
 1050 IFN%=0GOTO1080
 1060 H%=0:Q%=Q%-B%:IFL%MOD2=N%MOD2H%=1
 1070 PROCwin(1):RETURN
 1080 PROC0:IFN1%=0:RETURN
 1090 H%=0:Q%=Q%-B%:IFL%MOD2=N1%MOD2H%=1
 1100 PROCwin(0):RETURN
 1110 :
 1120 PRINTTAB(4,3);"NOIR/ROUGE":PRINTTA
B(4,4);"Black/red"
 1130 PROCbet:REPEAT
 1140 PRINTTAB(1,6);"Press key for    ch
oice (N/R)"
 1150 L%=GET:UNTILL%=78ORL%=82
 1160 IFN%=0GOTO1190
 1170 H%=0:Q%=Q%-B%:RESTORE1830:FORI%=1T
ON%:READE%:NEXT:IFE%=0ANDL%=78ORE%=1ANDL
%=82H%=1
 1180 PROCwin(1):RETURN
 1190 PROC0:IFN1%=0:RETURN
 1200 H%=0:Q%=Q%-B%:RESTORE1830:FORI%=1T
ON1%:READE%:NEXT:IFE%=0ANDL%=78ORE%=1AND
L%=82H%=1
 1210 PROCwin(0):RETURN
 1220 :
 1230 PROCwheel
 1240 MOVE835,100:PRINT"PRESS SPACE TO":
MOVE835,65:PRINT"RETURN TO MENU":IFGET
 1250 PROCtable:PROClist
```

```
1260 RETURN
1270 :
1280 DEFPROCwheel
1290 VDU5:VDU26:CLS:MOVE500,500
1300 FORF%=0TO36:MOVES1%(F%),C1%(F%)
1310 IFF%=0GCOL0,1
1320 IFF%MOD2=0GCOL0,1ELSEGCOL0,0
1330 IFF%=36GCOL0,2:F%=-1
1340 PLOT85,S1%(F%+1),C1%(F%+1)
1350 IFF%=-1F%=36
1360 MOVE500,500:NEXT
1370 RESTORE1870:FORF%=0TO36
1380 MOVES2%(F%),C2%(F%):READA$:GCOL0,3
:PRINTA$:NEXT
1390 ENDPROC
1400 :
1410 DEFPROCtable
1420 CLS:CLG:VDU5:GCOL0,2
1430 MOVE0,0:MOVE0,1024:PLOT85,736,1024
:MOVE736,0:PLOT85,0,0
1440 GCOL0,3
1450 FORI%=0TO3:MOVE32,928-(I%*256):DRA
W704,928-(I%*256):NEXT
1460 FORI%=0TO1:MOVE32+(I%*672),96:DRAW
32+(I%*672),928:NEXT:FORI%=0TO1:MOVE224+
(I%*288),96:DRAW224+(I%*288),992:NEXT
1470 FORI%=0TO12:MOVE224,992-(I%*64):DR
AW512,992-(I%*64):NEXT
1480 MOVE32,96:DRAW704,96
1490 FORI%=0TO1:MOVE320+(I%*96),160:DRA
W320+(I%*96),928:NEXT:FORI%=0TO1:FORJ%=0
TO1:MOVE96+(I%*64)+(J%*480),96:DRAW96+(I
%*64)+(J%*480),160:NEXT:NEXT
1500 FORI%=0TO1:MOVE128+(I%*480),224:MO
VE128+(I%*480),352:GCOL0,I%:PLOT85,64+(I
%*480),288:NEXT:FORI%=0TO1:MOVE128+(I%*4
80),224:MOVE128+(I%*480),352:GCOL0,I%:PL
OT85,192+(I%*480),288:NEXT
1510 MOVE352,976:GCOL0,0:PRINT"0"
1520 D%=1:RESTORE1830
1530 FORI%=0TO2:FORJ%=0TO2:D$=STR$(D%):
MOVE256+(J%*96),912-(I%*64):READE%:GCOL0
,E%:PRINTD$:D%=D%+1:NEXT:NEXT
1540 FORI%=3TO11:FORJ%=0TO2:D$=STR$(D%)
:MOVE240+(J%*96),912-(I%*64):READE%:GCOL
0,E%:PRINTD$:D%=D%+1:NEXT:NEXT
1550 RESTORE1820:FORI%=0TO2:MOVE48+(I%*
64),140:GCOL0,0:READB$:PRINTB$:MOVE656-(
I%*64),140:PRINTB$:NEXT
1560 RESTORE1840:FORI%=0TO13:MOVE80,896
-(I%*32):GCOL0,0:READC%:PRINTCHR$(C%):MO
VE624,896-(I%*32):READC%:PRINTCHR$(C%):N
EXT
1570 RESTORE1850:FORI%=0TO5:MOVE144,896
-(I%*32):GCOL0,0:READD%:PRINTCHR$(D%):MO
VE560,896-(I%*32):READD%:PRINTCHR$(D%):N
EXT
1580 ENDPROC
1590 :
1600 DEFPROCspin
1610 J%=RND(3)
1620 FORI%=1TORND(5):FORF%=0TO36:Po%=PO
INT(S%(F%),C%(F%)):GCOL0,3:MOVES%(F%),C%
(F%):VDU255:SOUND0,1,6,1:GCOL0,Po%:MOVES
%(F%),C%(F%):VDU255:NEXT:NEXT
1630 FORI%=1TOJ%:FORF%=0TO36:Po%=POINT(
S%(F%),C%(F%)):GCOL0,3:MOVES%(F%),C%(F%)
:VDU255:SOUND0,1,6,1:TIME=0:REPEAT:UNTIL
TIME=I%^2:GCOL0,Po%:MOVES%(F%),C%(F%):VD
U255:NEXT:NEXT
1640 C%=-2:RESTORE1870:REPEAT:READH%:C%
=C%+1:UNTILH%=N%
1650 FORF%=0TOC%:Po%=POINT(S%(F%),C%(F%
)):GCOL0,3:MOVES%(F%),C%(F%):VDU255:SOUN
D0,1,6,1:TIME=0:REPEAT:UNTILTIME=I%^2:GC
OL0,Po%:MOVES%(F%),C%(F%):VDU255:NEXT
1660 GCOL0,3:MOVES%(F%),C%(F%):VDU255
1670 ENDPROC
1680 :
1690 DEFPROCball
1700 PROCspin
1710 IFN%>0ORN%=0ANDZ%=0GOTO1760
1720 N%=N1%
1730 MOVE835,100:PRINT"PRESS SPACE TO":
MOVE935,65:PRINT"RESPIN":REPEAT:UNTILGET
=32:GCOL0,2:MOVES%(F%),C%(F%):VDU255
1740 VDU28,26,31,39,28:VDU4:CLS:VDU5
1750 PROCspin
1760 PROCwinlose:ENDPROC
1770 :
1780 DEFPROCwinlose
1790 IFX%>K%MOVE850,900:PRINT"YOU HAVE
LOST":MOVE950,850:PRINT"£";X%-K%:SOUND1,
4,20,10:SOUND1,4,5,20:ELSE MOVE900,900:P
RINT"YOU HAVE WON":MOVE950,850:PRINT"£";
K%-X%:SOUND1,4,95,2:SOUND1,4,115,2:SOUND
1,4,130,2:SOUND1,4,150,30
1800 MOVE835,100:PRINT"PRESS SPACE TO":
MOVE935,65:PRINT"CONTINUE":REPEAT:UNTILI
NKEY(1)=32:ENDPROC
1810 :
1820 DATA P,M,D
1830 DATA 1,0,1,0,1,0,1,0,1,0,0,1,0,1,0
,1,0,1,1,0,1,0,1,0,1,0,1,0,0,1,0,1,0,1,0
,1
1840 DATA 49,49,57,32,32,220,220,32,32,
49,51,56,54,32,32,32,32,73,80,77,65,80,7
3,65,82,73,32,82
1850 DATA 80,77,65,65,83,78,83,81,69,85
,32,69
1860 DATA SINGLE NUMBER,TWO NUMBERS,HOR
IZ. ROW,SQUARE,TWO ROWS,COLUMN,DOZEN,TWO
 DOZEN,HIGH/LOW,EVEN/ODD,BLACK/RED,SEE W
HEEL,SPIN BALL
1870 DATA 32,15,19,4,21,2,25,17,34,6,27
,13,36,11,30,8,23,10,5,24,16,33,1,20,14,
31,9,22,18,29,7,28,12,35,3,26,0
1880 :
1890 DEFPROClist
1900 VDU28,23,31,39,0:VDU4:CLS
```

```
 1910 COLOUR2:PRINTTAB(2,0);"CHOOSE YOUR
 BET":COLOUR1:PRINTTAB(1,2);"KEY   BET"
 1920 COLOUR3:FORI%=1TO13:PRINTTAB(1,I%+
3)CHR$(I%+64):NEXT
 1930 RESTORE1860:FORI%=1TO13:READG$:PRI
NTTAB(4,I%+3);G$:NEXT
 1940 PRINTTAB(1,20);"Money remaining
  £";Q%
 1950 ENDPROC
 1960 :
 1970 DEFPROCmenu:PROClist
 1980 IFQ%=0ENDPROC
 1990 REPEAT:A%=GET-64:UNTILA%>0ANDA%<14
 2000 CLS:COLOUR1
 2010 ON A% GOSUB220,290,410,560,650,710
,770,830,900,1010,1120,1230,2030
 2020 IFSpin%=1ENDPROC ELSEGOTO1980
 2030 Spin%=1:RETURN
 2040 :
 2050 DEFPROCbet
 2060 VDU28,23,31,39,6:CLS
 2070 COLOUR3:REPEAT:CLS
 2080 PRINTTAB(1,13);"Money remaining
  £";Q%
 2090 INPUTTAB(1,1)"How much do you  wan
t to bet?       £"B%:UNTILB%<100001ANDQ%
-B%>-1
 2100 ENDPROC
 2110 :
 2120 DEFPROCnumber
 2130 VDU28,23,31,39,10:CLS
 2140 REPEAT:CLS
 2150 INPUTTAB(1,1)"What is the       low
est number    your bet is to   cover",L%
:UNTILL%<36
 2160 H%=0:ENDPROC
 2170 :
 2180 DEFPROCgame
 2190 K%=500:REPEAT:Q%=K%:X%=K%:Z%=0
 2200 Spin%=0:N%=RND(37)-1
 2210 PROCtable:PROCmenu
 2220 SOUND1,3,100,25
 2230 PROCwheel:PROCball
 2240 UNTILK%>5*10^6ORK%=0
 2250 ENDPROC
 2260 :
 2270 DEFPROC0
 2280 IFZ%=1ENDPROC
 2290 N1%=RND(37)-1
 2300 IFN1%=0K%=K%-B%:Q%=Q%-B%:PROClist
 2310 Z%=1
 2320 ENDPROC
 2330 :
 2340 DEFPROCwin(W)
 2350 IFH%=1K%=K%+(W*B%)ELSEK%=K%-B%
 2360 PROClist:ENDPROC
 2370 :
 2380 DEFPROCsetup
 2390 VDU23,255,48,120,252,252,120,48,0;
 2400 VDU23,220,16,8,0,60,6,62,102,62
 2410 ENVELOPE1,1,0,0,0,6,3,3,127,-5,-5,
-5,100,60
 2420 ENVELOPE2,2,6,0,0,255,0,0,126,0,0,
-126,126,126
 2430 ENVELOPE3,5,16,12,8,2,1,1,10,-10,0
,-10,200,100
 2440 ENVELOPE4,3,0,0,0,0,0,0,127,-10,-5
,-2,120,120
 2450 PRINTTAB(8,11)"PLEASE WAIT A FEW S
ECONDS"
 2460 DIMS%(36),C%(36),S1%(36),C1%(36),S
2%(36),C2%(36)
 2470 F%=0:FORA=0TO360STEP360/37
 2480 FORA=0TO360STEP360/37
 2490 S%(F%)=370*SIN(RAD(A+5))+484:S1%(F
%)=500*SIN(RAD(A))+500:S2%(F%)=460*SIN(R
AD(A+5))+473
 2500 C%(F%)=370*COS(RAD(A+5))+516:C1%(F
%)=500*COS(RAD(A))+500:C2%(F%)=460*COS(R
AD(A+4))+515
 2510 F%=F%+1:NEXT
 2520 ENDPROC
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS

LOCAL POINTS OF INTEREST

The keyword LOCAL, followed by one or more variable names, inside a procedure will create local versions of those variables, only accessible inside that procedure. However, if another procedure is called from within the first procedure then the local definitions are good for that second procedure too.

TROUBLE WITH *FX3 - Martin Abernethy

If *FX3,8 is used to select output stream to a printer then VDU1 should not be used to send control codes to the printer as all the 1's are sent as well. If, after a *FX3,8, VDU2 (or Ctrl-B) is also used to enable the printer then a VDU1 not only sends the 1 but the following code twice!

QUICK RANDOM - Roger Burg

RND(X) is comparatively slow in BBC Basic. About twice the speed is the pseudo random function TIME MOD X.

BEEBUG MAGAZINE is produced by BEEBUG Publications Ltd.
Editor: Mike Williams
Assistant Editor: Geoff Bains
Production Editor: Phyllida Vanstone
Technical Assistant: Alan Webster
Secretary: Debbie Sinfield
Managing Editor: Lee Calcraft
Additional thanks are due to Sheridan Williams, Adrian Calcraft, John Yale and Tim Powys-Lybbe.

All rights reserved. No part of this publication may be reproduced without prior written permission of the Publisher. The Publisher cannot accept any responsibility, whatsoever for errors in articles, programs, or advertisements published. The opinions expressed on the pages of this journal are those of the authors and do not necessarily represent those of the Publisher, BEEBUG Publications Limited.
BEEBUG Publications Ltd (c) 1985

Editorial Address
BEEBUG
PO BOX 50,
Hollywell Hill,
St. Albans AL1 1EX

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 £40 per page, but please give us warning of anything substantial that you intend to write. A leaflet, 'Notes of Guidance for Contributors' is available on receipt of an A5 (or larger) SAE.

In the case of material longer than a page, we would prefer this to be submitted on cassette or disc in machine readable form using "Wordwise", "View", or other means, but please ensure an adequate written description of your contribution is also included. If you use cassette, please include a backup copy at 300 baud.

HINTS

There are prizes of £5 and £10 for the best hints each month, plus one of £15 for a hint or tip deemed to be exceptionally good.

Please send all editorial material to the editorial address below. If you require a reply it is essential to quote your membership number and enclose an SAE.

SUBSCRIPTIONS

Send all applications for membership, subscription renewals, subscription queries and orders for back issues to the subscriptions address.

MEMBERSHIP SUBSCRIPTION RATES

£ 6.40 6 months (5 issues) UK ONLY
£11.90 UK - 1 year (10 issues)
£18 Europe, £21 Middle East
£23 Americas & Africa, £25 Elsewhere

BACK ISSUES
(Members only)

Vol	Single issues	Volume sets (10 issues)
1	90p	£8
2	£1	£9
3	£1.20	£11
4	£1.20	–

Please add the cost of post and packing as shown:

DESTINATION	First issue	Each subsequent issue
UK	30p	10p
Europe	70p	20p
Elsewhere	£1.50	50p

All overseas items are sent airmail (please send a sterling cheque). We will accept official UK orders but please note that there will be a £1 handling charge for orders under £10 that require an invoice. Note that there is no VAT on magazines.

Back issues are for members only, so it is ESSENTIAL to quote your membership number with your order. Please note that the BEEBUG Reference Card and BEEBUG supplements are not supplied with back issues.

Subscriptions, Back Issues & Software Address

BEEBUG
PO BOX 109
St. Johns Road
High Wycombe HP10 8NP

Hotline for queries and software orders

St. Albans (0727) 40303
Manned Mon-Fri 9am-4.30pm

24hr Answerphone Service for Access and Barclaycard orders, and subscriptions
Penn (049481) 6666

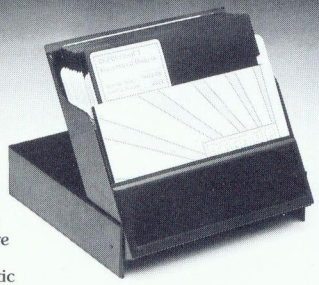
If you require members' discount on software it is essential to quote your membership number and claim the discount when ordering.

DYNAMIC DISCS

TESTED BY BEEBUG

BEEBUG, the largest independent computer user group in the UK, offer 100% tested discs supplied by one of Britain's leading disc manufacturers.

10 FREE LIBRARY CASE



Orders for 10 discs are sent in black plastic library cases.

25 FREE STORAGE BOX



Orders for 25 are delivered in strong plastic Storage box with 4 dividers.

50 FREE STORAGE BOX



Orders for 50 are delivered in strong plastic Storage box with 4 dividers.

COMPARE PRICES

4 Types of Disc To Meet Your Exact Requirement

	48 TPI	DOUBLE DENSITY			
10	S/S D/D	£14.90	10	D/S D/D	£20.50
25	S/S D/D	£34.90	25	D/S D/D	£46.20
50	S/S D/D	£59.30	50	D/S D/D	£82.40

	96 TPI	DOUBLE DENSITY			
10	S/S D/D	£20.50	10	D/S D/D	£21.90
25	S/S D/D	£46.20	25	D/S D/D	£49.90
50	S/S D/D	£82.40	50	D/S D/D	£93.50

All prices include Storage Box, VAT and delivery to your door (UK)

Suitable for BBC Micro and all other computers using 5¼ inch discs including Atari and Commodore.

Fully Guaranteed – Not only by Beebug but by one of the UK's top disc manufacturers.

We regret that we have had to pass on a slight increase in the price of our discs, but we are now able to offer a wider range to meet your exact requirements.

These discs are the best. Please use the enclosed order form and order from our usual address.
BEEBUG PO BOX 109
St. Johns Road
High Wycombe HP10 8NP.

Official orders are welcome.

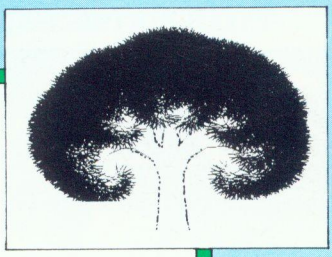
Barclaycard and Access telephone 0494 81 6666
Further information telephone 0727 60263

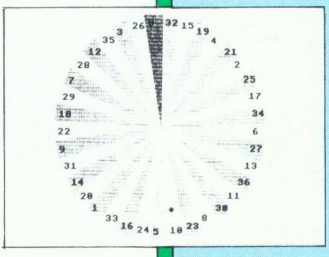
BEEBUG SOFT

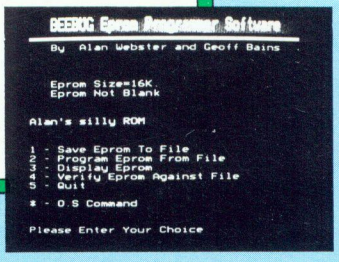
Magazine Cassette/Disc

OCTOBER 1985 CASSETTE DISC CONTENTS

RECURSIVE TREES – displays for the armchair gardener
DYNAMIC MEMORY WINDOW – an eye into your Beeb as it runs your programs
WORDWISE PLUS EXAMPLES – segment programs from the series
LOAN REPAYMENT – calculate how much you owe
EPROM PROGRAMMER DRIVER – blow your own
WORKSHOP PROCEDURES – text compression
DISC MENU EXTENSIONS – the complete disc menu program with all the extensions
DATA STRUCTURES – linked lists and binary trees
ROULETTE – spin the wheel and break the bank
EXTRA FEATURES THIS MONTH
DISC BENCHMARKS – test out your drives with the program used for the DDFS review
3D GRAPHICS SYSTEM DEMO – a fast moving display from Glentop's graphics package







All this for £3.00 (cass) £4.75 (disc) +50p p&p.
Back issues (disc since Vol. 3 No. 1, cass since Vol. 1 No. 10) available at the same prices.

Subscription rates	DISC UK	CASS UK	DISC O'seas	CASS O'seas
6 months (5 issues)	£25	£17	£30	£20
12 months (10 issues)	£50	£33	£56	£39

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

Cassette subscriptions can be commuted to disc subscription on receipt of £1.70 per issue of the subscription left to run.

All subscription and individual orders to
BEEBUG, PO BOX 109, St. Johns Road, High Wycombe HP10 8NP

Printed in England. DCL 10532