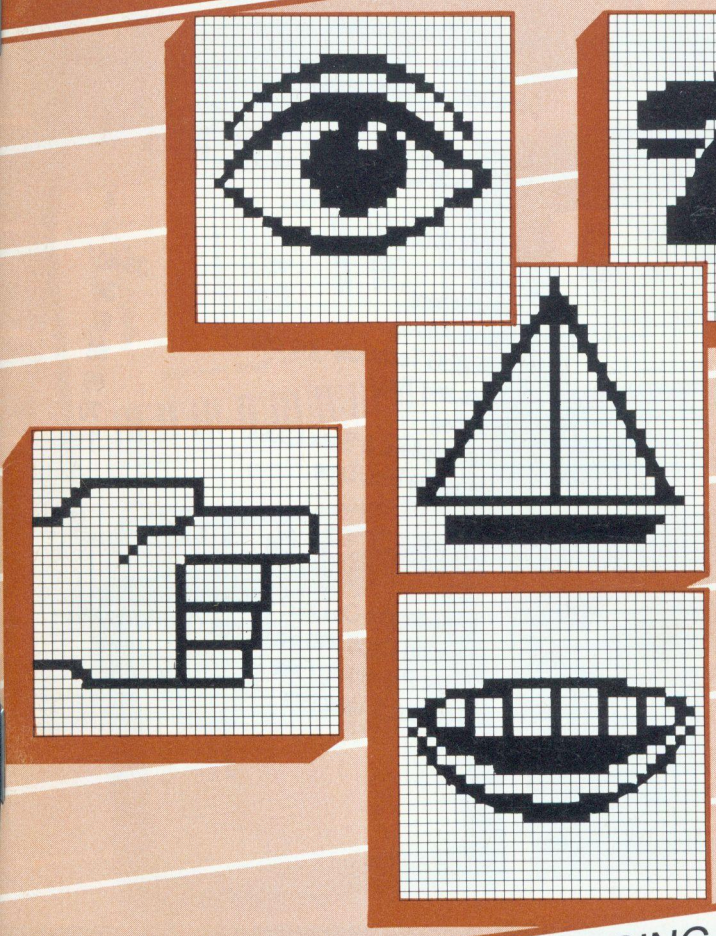
Vol.7 No.2 June 1988

BEEBUG

FOR THE BBC MICRO & MASTER SERIES



Beebug ICON Designer

- RENUMBER UTILITY
- USING SPREADSHEETS
- CHORDS AND KEYBOARDS
- KNIGHT'S TOUR

BEEBUG Vol.7 No.2 June 1988

FEATURES

REVIEWS

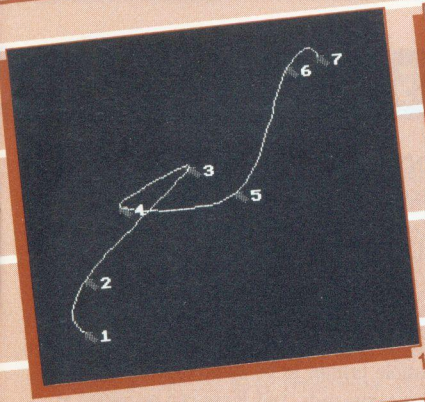
REGULAR ITEMS

HINTS & TIPS

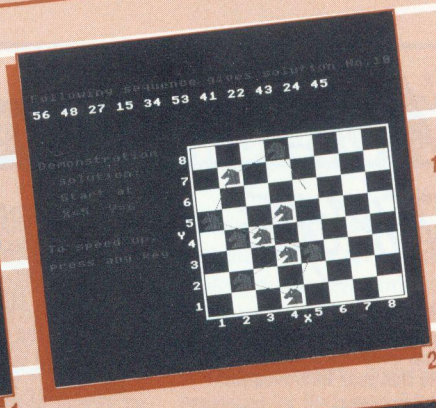
PROGRAM INFORMATION

All programs listed in BEEBUG magazine are produced direct from working programs. They are listed in LISTO1 format with a line length of 40. However, you do not need to enter the space after the line number when typing in programs, as this is only included to aid readability. The line length of 40 will help in checking programs listed on a 40 column screen.

Programs are checked against all standard Acorn systems (model B, B+, Master, Compact and Electron; Basic I and Basic II; ADFS, DFS and Cassette filing systems; and the Tube). We hope that the classification symbols for programs, and also reviews, will clarify matters with regard to compatibility. The complete set of icons is given



1.

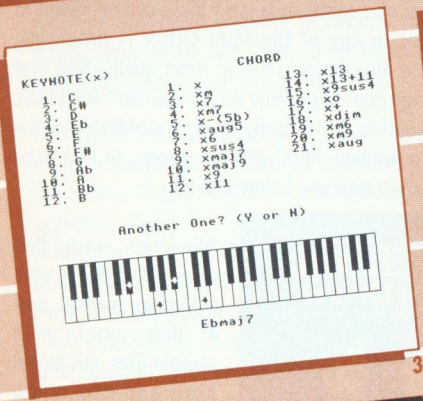


2.

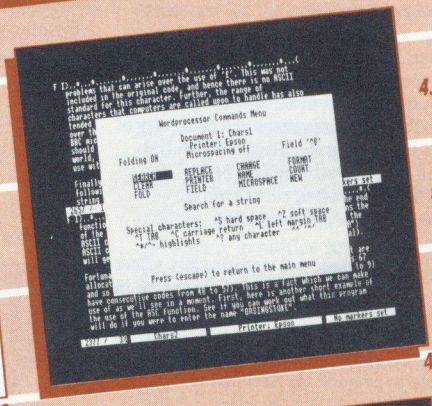
1. Curve Drawing

2. Knight's Tour

3. Chords & Keyboards



3.

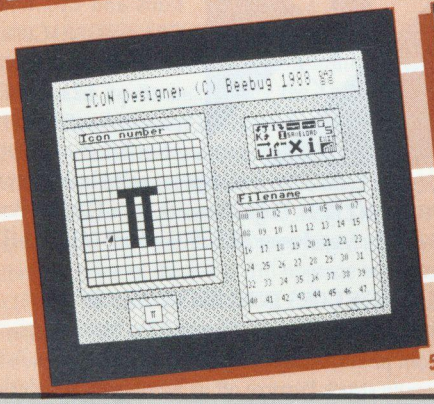


4.

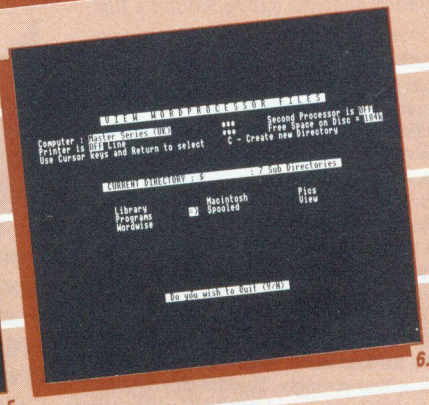
4. Double View Review

5. Icon Designer

6. ADFS View Menu



5.



6.

below. These show clearly the valid combinations of machine (version of Basic) and filing system for each item, and Tube compatibility. A single line through a symbol indicates partial working (normally just a few changes will be needed); a cross shows total incompatibility. Reviews do not distinguish between Basic I and II.

Computer System		Filing System	
Master (Basic IV)	M	ADFS	A
Compact (Basic VI)	C	DFS	D
Model B (Basic II)	II	Cassette	
Model B (Basic I)	I	**Tube Compatibility**	
Electron	E	Tube	

Editor's Jottings

NEWS OF ARCHIMEDES

The Archimedes continues to sell quite well, and we understand that Acorn has achieved a much more profitable first quarter to this year following their reported loss for the previous year.

Unfortunately, Acorn has also confirmed the forecast price increases for most of the Archimedes range with effect from 1st July, though the price of the 305 remains the same as before. The price increases are generally of the order of 9%, but increasing the price differential between the 305 and 310 has resulted in a price increase of some 67% in the cost of the 0.5 Mbyte memory upgrade to convert a 305 into a 310.

We certainly hope that this does not significantly affect sales, stemming as it does from a worldwide increase in chip prices. BEEBUG will also continue to supply the Archimedes at the old prices while stocks last, and we would remind BEEBUG members of the credit facilities that we are now able to provide for Archimedes purchasers.

RISC User, BEEBUG's magazine and support group for Archimedes users is still growing steadily, and we are more than pleased with the response so far. Many BEEBUG members have already taken advantage of the opportunity to subscribe to RISC User at the special members' rate, and this offers a substantial saving on the normal price. Likewise, RISC User members are entitled to subscribe to BEEBUG at the same advantageous rates. Further details are contained in the supplement pages.

Despite all the activity surrounding the Archimedes, Acorn recently announced firm plans for the production of a further 40,000 Master series computers which are still selling strongly. There is clearly plenty of life yet in the BBC micro, and BEEBUG will continue to offer the maximum level of support for all users.

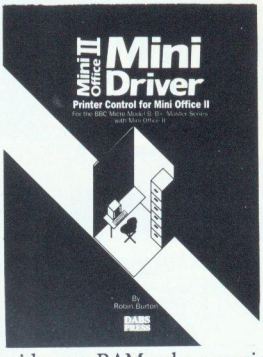
FUTURE ISSUES OF BEEBUG

We have received a good many letters in recent months regarding our coverage of C, and other topics as well. If you have any views on the content of future issues then we would be delighted to hear from you. We also welcome contributions for potential publication.

News News News Nev

OFFICE PRINTING

The latest piece of software to emerge from Dabs Press is *MiniDriver*, a very comprehensive printer driver for Database's Mini Office II system. The printer driver is based on the already successful *HyperDriver* from Dabs Press, and will work with all the various versions of Mini Office II. *MiniDriver* allows all the effects of any Epson-compatible printer to be controlled by easy-to-remember star commands, and these can be issued at any point in any of the Mini Office II programs. *MiniDriver* also contains a near letter quality font that allows high quality printing on printers that don't have an NLQ option as standard. Another nice feature is the ability to preview highlights on screen, so that page layout can be finalised before printing.



MiniDriver is supplied on a 16K EPROM with a 40 page manual, and a disc containing sample files, additional hints and tips, and a sideways RAM image of the program, at a cost of £24.95. For people wanting to run *MiniDriver* from sideways RAM only, a version is available without the EPROM at £19.95. Dabs Press are at 76 Gardner Road, Prestwich, Manchester M25 7HU, phone 061-773 2413.

PEARTREE GOES INTO RECEIVERSHIP

Peartree Computers, the Huntingdon based supplier of Acorn products, has just gone into receivership. Acorn recently struck Peartree off its list of approved dealers, and it is a combination of this and a disastrous attempt to break into the PC market that has lead to the company's problems. Peartree has recently been criticised by the Advertising Standards Authority for continuing to advertise its *Music 87* system, even

though there was no sign of it becoming available in the near future. The company is still trading but it is thought that a number of staff, including the manager Varten Mundigian, have been laid off. It is hoped that a buyer will be found for the company, but in the meantime an official receiver has been appointed to handle business. Everybody who is owed money by Peartree should have received a letter sent by the receiver. Anybody wishing to contact the receiver should write to Mr. M.J.Scott, Grant Thornton Ltd., 49 Mill Street, Bedford, or ring (0234) 211521.

ON SCREEN PUBLISHING

PMS has launched a Desktop Publishing package for the model B and Master series. Unlike other such systems, *Publisher*, as it is called, uses a page description language (PDL). All the effects can be used from within View, Wordwise or Inter-Word, and will work with any Epson-compatible printer. The package costs £44.85 from PMS, 38 Mt. Cameron Drive, East Kilbride G74 2ES, or phone (03552) 32796.

ARCHIMEDES PRICE RISES

As speculated in BEEBUG last month, Acorn has announced a price increase for the Archimedes range. The rises, which were announced to dealers at Acorn's first national dealer meeting, mean that the price of an A310 with colour monitor is now £1213.25. The price of the A305 remains unchanged, and the increased differential between the two machines is reflected in the new price of the 305 to 310 upgrade which is now £171.35, an increase of almost £90. The price of the Master series machines also remains unchanged. The new prices are effective from July 1st.

SHOW NEWS

Once again the Micro User show held in London last month attracted many visitors. However, many exhibitors said that they were disappointed with the show compared to previous ones, and many said that their takings were well below what they had expected. Meanwhile, the saga of the Acorn User show continues. When pressed on the subject at a recent dealer's meeting, Acorn's managing director Harvey Coleman would not comment. However, it is widely speculated that the up-market show wanted by Acorn will take place some time this autumn, and will still be organised by Redwood Publishing. We will of course bring you details as soon as they are available.

ARCHIMEDES LOGO

Logotron, the company that produced the version of Logo supplied with the Master Compact, has just launched an Archimedes Logo. The new version is written to take full advantage of the Archimedes advanced graphics and high speed. Despite this, *Archimedes Logo* is fully compatible with the BBC version from Logotron, and any extension modules written on the BBC can be transferred across. Logo is supplied on a 3.5" disc with keystrip, reference manual and a tutorial guide, although because of production problems the first few are being sent out without the tutorial guide, which will be sent later. *Archimedes Logo* costs £71.87 inclusive, and a network version with site licence is available at £345. Logotron are at Dales Brewery, Gwydir Street, Cambridge CB1 2LJ, or phone (0223) 323656.

ACTING ON IMPULSE

Computer concepts have announced that they are working on a new operating system for the Archimedes. The company claim that neither the current version of Arthur supplied with the machine, nor the new Arthur 2 that Acorn are developing, will satisfy the needs of the advanced software that they plan to release. The new operating system is called *Impulse* and it is expected that it will be available both as a ROM set to totally replace Arthur, and as a series of disc-based modules to co-exist with Arthur. It is thought that prototypes of *Impulse* will be displayed at the PCW show in September. B

The Knight's Tour

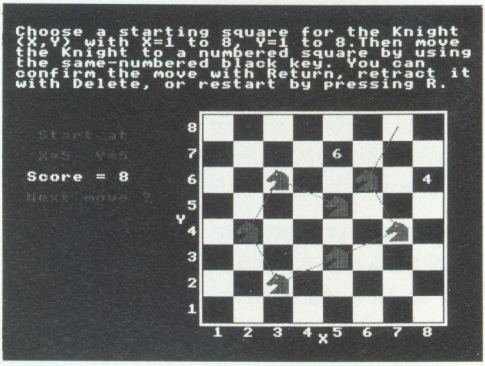
As an alternative to the more usual arcade-style of game, Eric Bramley presents a highly visual implementation of one of the oldest chess problems.

The curious move of the knight in chess is well known, even by non chess players. It can move to any vacant square on the chessboard which is one row and two columns, or two rows and one column, away from the square it occupies. It is an interesting and well known problem to see if a series of moves can be found which will take the knight in turn to each square on the board, but with the important restriction that no square must be visited more than once. If the journey can be made so as to arrive back at the original starting square (a so-called re-entrant path), so much the better.

You can try your hand at this venerable problem with the program given here, which allows you to keep track of the route followed, and at each step shows which squares are available for the next move. Any square may be chosen from which to start. There are many different solutions available for any starting square, and the program includes the option of seeing a number of these demonstrated. This can be done before you start trying to find your own solution, or after you have tired of finding a solution before reaching the magic score of 64 (This being the number of locations visited in the tour).

Type the program in as listed, and save to tape or disc. When the program is run, you are given the option of seeing a complete tour demonstrated, or trying it for yourself. In either case the starting square is chosen by entering its horizontal (X) and vertical (Y) co-ordinates as marked on the displayed chessboard. If a demonstration has been chosen, this then proceeds automatically with any one of the four possible routes being selected at random. The sequence of squares visited is shown, while the route of the knight is traced on the board.

If the option to 'play' has been selected, a flashing knight is shown on the starting square, together with up to 8 numbered squares which are available for the knight's next move. When one of these is chosen, by entering the appropriate number, the move may be confirmed with Return, or retracted with Delete. The process is continued until you run out of further available squares or reach the target score of 64. In either case you will have the option of trying again, or of seeing a solution demonstrated.



PROGRAM NOTES

The constant D in line 140 is the width in graphics units of a square on the chessboard, the value of 84 having been chosen to give the best representation of squares on a Microvitec 1451 monitor. This value may need adjustment for other display units. The board is formed in lines 1410-1500 by setting a succession of 32 graphics windows which are each cleared to white.

The demonstration solution is effected in PROCdemo, with four possible sequences of squares listed as DATA at the end of the program. The chosen starting square is first found in the list, which is then followed to the end, and re-read until the starting square is again reached. Note that RESTORE is used to locate the data for each route so the program must not be renumbered.

```
  10 REM Program  Knight's Tour
  20 REM Version  B1.03
  30 REM Author   Eric Bramley
  40 REM BEEBUG   June 1988
  50 REM Program  subject to copyright
  60 :
 100 ON ERROR MODE7:PROCerror:END
```

```
  110 MODE1
  120 DIM F%(8,8),x(8),y(8),H(8)
  130 x(1)=1:x(2)=2:x(3)=2:x(4)=1
  140 x(5)=-1:x(6)=-2:x(7)=-2:x(8)=-1
  150 y(1)=2:y(2)=1:y(3)=-1:y(4)=-2
  160 y(5)=-2:y(6)=-1:y(7)=1:y(8)=2
  170 D=84
  180 PROCintro
  190 REPEAT:G=GET:UNTIL G=68 OR G=80
  200 IF G=68 PROCdemo ELSE PROCplay
  210 REPEAT
  220 VDU28,0,31,12,18:CLS
  230 COLOUR1:PRINT TAB(0,1)"What now?"'
'"Press key :"'''"P to play"'''"D for a"'"
demonstrationsolution"'''"Q to quit"
  240 REPEAT:G=GET:UNTIL G=68 OR G=80 OR
 G=81
  250 IF G=81 MODE 7:END
  260 IF G=68 PROCdemo ELSE PROCplay
  270 UNTIL FALSE
  280 :
 1000 DEFPROCintro
 1010 PRINT TAB(11,4)"THE KNIGHT'S TOUR"
TAB(11,5)STRING$(17,"=")
 1020 PRINTTAB(11,7) "OF THE CHESS BOARD
"TAB(11,8)STRING$(18,"=")
 1030 PRINT'"The object of this puzzle
is to make a  complete tour of the chess
 board with   the Knight, visiting each
square once   and once only.Choose any s
quare on whichto start, and then move th
e Knight by"
 1040 PRINT"pressing one of the black ke
ys numbered 1 to 8, according to the dir
ection in   which you want to move."
 1050 PRINT'"The program includes four
different     solutions, which can be ca
lled at randomfor demonstration."
 1060 VDU 23,240,3,3,7,14,31,63,127,255
 1070 VDU 23,241,255,112,1,3,7,15,31,63
 1080 VDU 23,242,0,224,248,252,254,255,2
55,255
 1090 VDU 23,243,255,255,255,255,255,255
,255,255
 1100 COLOUR2:VDU19,2,10;0;
 1110 FOR K=5 TO 33 STEP28:PRINTTAB(K,3)
CHR$240 TAB(K,4)CHR$241 TAB(K+1,3)CHR$24
2 TAB(K+1,4)CHR$243:NEXT
 1120 PRINT TAB(0,29)"Press P to play, o
r D for demonstration"
 1130 ENDPROC
 1140 :
 1150 DEF PROCerror
 1160 REPORT
 1170 PRINT" at line ";ERL
 1180 ENDPROC
 1190 :
 1200 DEFPROCdemo
 1210 PROCboard
 1220 VDU28,0,31,12,9
 1230 COLOUR1:PRINT TAB(0,0)"Demonstrati
on"TAB(2,2)"solution:"
 1240 PROCstart
 1250 X=X1:X$=STR$X:Y=Y1:Y$=STR$Y
 1260 PRINT TAB(0,10)"To speed up,"'''"pr
ess any key"
 1270 Z$=X$+Y$
 1280 M=RND(4):M=M*10
 1290 VDU26:PRINT'"Following sequence gi
ves solution No.";M
 1300 PROCknight(2):VDU7
 1310 COLOUR3:PRINT
 1320 RESTORE (2410+M):REPEAT:READ A$:UN
TIL A$=Z$:PRINTA$" ";:MOVE 528+D*(X-.5),
80*(Y+.5):IF Z$="32" GOTO 1340
 1330 REPEAT:PROCstep:UNTIL A$="32"
 1340 RESTORE (2410+M):REPEAT:PROCstep:U
NTIL A$=Z$:PROCknight(2)
 1350 VDU28,0,31,12,16:CLS:COLOUR2
 1360 PRINT TAB(0,1)"That's it!"
 1370 ENDPROC
 1380 :
 1390 DEFPROCboard
 1400 VDU26:GCOL0,128:CLS
 1410 FOR K=0 TO 8 STEP4
 1420 IF K=0 GCOL0,0 ELSE GCOL0,3
 1430 MOVE 524-K,76-K:DRAW 528+8*D+K,76-
K:DRAW 528+8*D+K,720+K
 1440 DRAW 524-K,720+K:DRAW 524-K,76-K:N
EXT
 1450 GCOL0,131
 1460 FOR K=0 TO 1
 1470 FOR P=528+D*K TO 528+D*(K+6) STEP
2*D
 1480 FOR Q=160-80*K TO 640-80*K STEP 16
0
 1490 VDU24,P;Q;P+D-4;Q+76;:CLG
 1500 NEXT:NEXT:NEXT
 1510 GCOL0,128
 1520 VDU26,5
 1530 FOR X=1 TO 8
 1540 MOVE 512+D*(X-.5),60:PRINT;X:NEXT
 1550 FOR Y=1 TO 8
 1560 MOVE 480,80*Y+52:PRINT;Y:NEXT
 1570 MOVE 512+4*D,40:PRINT"X":MOVE 448,
408:PRINT"Y"
 1580 VDU4
 1590 ENDPROC
 1600 :
 1610 DEFPROCstart
 1620 COLOUR1:PRINT TAB(2,4)"Start at"TA
B(2,6)"X=?"
 1630 REPEAT:G=GET:UNTIL ABS(G-52.5)<4
 1640 X1=G-48:PRINT TAB(4,6);X1 TAB(7,6)
"Y=?"
 1650 REPEAT:G=GET:UNTIL ABS(G-52.5)<4
```

```
 1660 Y1=G-48:PRINT TAB(9,6);Y1
 1670 ENDPROC
 1680 :
 1690 DEFPROCknight(n)
 1700 VDU5
 1710 GCOL0,n:IF (X+Y)MOD2=1 GCOL0,131 E
LSE GCOL0,128
 1720 MOVE 496+D*(X-.5),68+80*Y:VDU240
 1730 MOVE 496+D*(X-.5),36+80*Y:VDU241
 1740 MOVE 528+D*(X-.5),68+80*Y:VDU242
 1750 MOVE 528+D*(X-.5),36+80*Y:VDU243
 1760 VDU4
 1770 ENDPROC
 1780 :
 1790 DEFPROCstep
 1800 I=INKEY(100):READ A$:
 1810 IF POS>37 VDU13,10
 1820 PRINT A$" ";
 1830 X=VAL(LEFT$(A$,1))
 1840 Y=VAL(RIGHT$(A$,1))
 1850 GCOL0,1:DRAW 528+D*(X-.5),40+80*Y
 1860 PROCknight(1)
 1870 MOVE 528+D*(X-.5),40+80*Y:VDU7
 1880 ENDPROC
 1890 :
 1900 DEFPROCplay
 1910 PROCboard
 1920 FOR J=1 TO 8
 1930 FOR K=1 TO 8
 1940 F%(J,K)=0
 1950 NEXT:NEXT
 1960 COLOUR3:PRINT'"Choose a starting s
quare for the Knight (X,Y) with X=1 to 8
, Y=1 to 8.Then move the Knight to a num
bered square by usingthe same-numbered b
lack key. You can    confirm the move wi
th Return, retract it";
 1970 PRINT"with Delete, or restart by p
ressing R."
 1980 VDU28,0,31,12,7
 1990 PROCstart
 2000 F%(X1,Y1)=1:S=1:VDU7
 2010 X=X1:Y=Y1:PROCknight(2)
 2020 VDU28,0,31,12,15
 2030 COLOUR3:PRINT TAB(1,0)"Score = ";S
 2040 IF S=64 PROCmax:ENDPROC
 2050 COLOUR1
 2060 PRINT TAB(0,2)" Next move ?  "
 2070 C=0
 2080 FOR K=1 TO 8:H(K)=0:X=X1+x(K):Y=Y1
+y(K):IF ABS(X-4.5)>4 OR ABS(Y-4.5)>4 GO
TO 2100
 2090 IF F%(X,Y)=0 C=C+1:PROCnumber(2):H
(K)=1
 2100 NEXT
 2110 VDU4
 2120 IF C=0 COLOUR2:PRINT TAB(0,2)"You'
re stuck!":ENDPROC
```

```
 2130 REPEAT:G=GET:UNTIL ABS(G-52.5)<4 O
R G=82:IF G=82 GOTO 1910
 2140 N=G-48:IF H(N)=0 GOTO 2130
 2150 FOR K=1 TO 8:IF H(K)=1 X=X1+x(K):Y
=Y1+y(K):PROCnumber(4-(X+Y)MOD2)
 2160 NEXT:VDU4
 2170 X=X1:Y=Y1:PROCknight(1)
 2180 X=X1+x(N):Y=Y1+y(N):PROCknight(2)
 2190 PRINT TAB(0,2)"   Return   "'"'" to
 confirm"TAB(5,6)"or"'"'"   Delete"
 2200 REPEAT:G=GET
 2210 UNTIL G=13 OR G=82 OR G=127
 2220 IF G=82 GOTO 1910
 2230 VDU28,0,31,12,17:CLS
 2240 IF G=127 CLS:PROCknight(4-(X+Y)MOD
2):GOTO 2010
 2250 S=S+1:F%(X,Y)=1:VDU7
 2260 MOVE 528+D*(X-.5),40+80*Y:GCOL0,1:
PLOT1,-D*x(N),-80*y(N)
 2270 X1=X:Y1=Y:GOTO 2020
 2280 ENDPROC
 2290 :
 2300 DEFPROCmax
 2310 VDU28,0,31,12,17:CLS
 2320 COLOUR2:PRINT " Well done!"
 2330 VDU7,26
 2340 ENDPROC
 2350 :
 2360 DEFPROCnumber(m)
 2370 VDU5
 2380 MOVE 512+D*(X-.5),52+80*Y
 2390 GCOL0,m:PRINT;K
 2400 ENDPROC
 2410 :
 2420 DATA 11,23,31,12,33,14,26,18,37,16
,28,36,17,25,13,21,42,54,75,87,68,47,35,
56,48,27,15,34,53,41,22,43,24,45,57,38,4
6,58,77,65,84,76,88,67,86,78,66,85,64,72
,51,63,55,74,82,61,73,81,62,83,71,52,44,
32
 2430 DATA 13,21,42,34,15,27,48,36,44,56
,75,87,68,76,88,67,86,78,57,65,84,72,51,
63,55,43,24,45,53,74,82,61,73,81,62,83,7
1,52,64,85,77,58,66,54,46,25,17,38,26,18
,37,16,28,47,35,14,22,41,33,12,31,23,11,
32
 2440 DATA 51,43,24,12,31,52,71,83,64,72
,84,63,82,74,53,41,22,14,33,54,62,81,73,
61,42,21,13,34,55,76,88,67,48,56,75,87,6
8,47,28,16,35,27,15,36,17,25,46,58,77,85
,66,45,37,18,26,38,57,78,86,65,44,23,11,
32
 2450 DATA 11,23,15,27,48,36,57,78,86,65
,53,74,82,61,42,21,13,34,46,25,17,38,26,
18,37,45,66,58,77,85,73,81,62,54,33,41,2
2,14,35,16,28,47,68,87,75,56,64,83,71,52
,31,12,24,43,55,67,88,76,84,72,51,63,44,
32
```

curve drawing with splines

Drawing curves through random sets of points is no trivial task. Roger Burg explains what it's all about.

A common problem in graphics is to have a very short list of points which define a shape, and no routine which can join them into one continuous curved line. The routine described here is an approach which has certain advantages over other methods. It can take any number of points greater than two, it produces a natural looking outline, and it can also produce straight runs and corners. And the name 'spline' - this is simply the name given by mathematicians to any curve made to fit a set of points.

The listing accompanying this article contains a complete procedure for drawing splines, and this is incorporated in a graphical demonstration of the results which may be readily obtained with this technique.

UNDERSTANDING THE PRINCIPLES

The method uses four ideas. Positions are defined repeatedly at a fraction, 'n', of their distance between two points. The basic shape is a simple bow, defined between three points as in Figure 1. Each such bow has an 'apex' which lies beyond the middle of its three points. And the position of the final continuous curve is defined by the same fractional method, to blend the series of bows into one line.

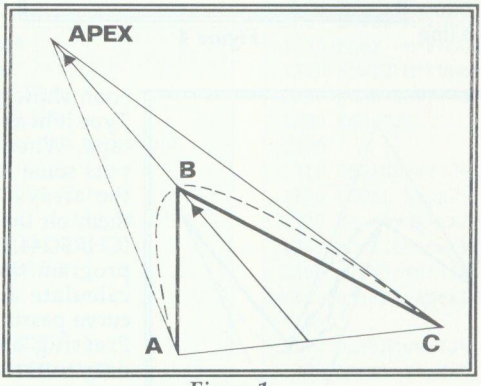


Figure 1

I thought I could produce a curve based on three points A, B and C, by drawing a series of lines. The first would start a fifth of the way from A to B, and end a fifth of the way from B to C. The second would begin two fifths further on, and so on. Figure 2 shows the idea. It seemed obvious that joining the mid-points of each line would produce a smooth curve from A to C. Of course, it doesn't work. It produces a straight line.

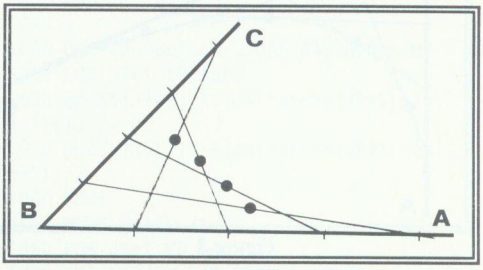


Figure 2

But in Figure 3, the first line (which began one fifth of the way from A to B) is marked one fifth of the way along it, then the second line (which was two fifths further on, was marked at a point two-fifths along it, and so on. This produces a nice freehand curve. It is not very useful, because the curve misses point B, and only touches at points A and C. But if we forget point B, and use an apex beyond it, the curve can still pass through point B.

The point needed is as shown in Figure 1. It's at the same distance from B, and in the opposite direction as the point halfway between A and C, so it can be determined easily. These points are calculated in the procedure PROCspline and stored in the arrays apeX%() and apeY%().

The variable 'n' holds the fraction of the distance along any line which the routine is processing at the time. It is always a fraction between zero and one. So in PROCspline the program uses FOR loops to call PROCsimplebow and draw the first half and the second half of this hyperbolic shape. One begins the long curve and the other finishes it.

Once I'd got this basic idea, I found the only tricky bit was keeping track of 'n', the fractional value. It's always measured forwards, in the direction the bow is drawn. It defines points

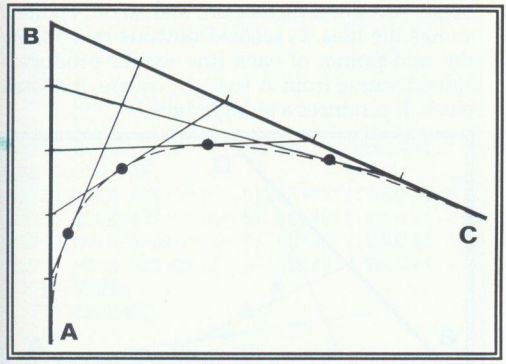


Figure 3

along A-to-apex and apex-to-C as well as the positions on the bow itself.

But you don't get a smooth curve from a series of arches, alternately leap-frogging over groups of three points (see Figure 4). The 'n' idea has to be used again. Figure 5 shows that we only need to calculate a series of points snaking alternately between these bows, to produce a continuous curve.

The resulting curve must stick to the centre of each bow, but the further it goes from the middle of the bow, the more closely it must approach the line of the next bow.

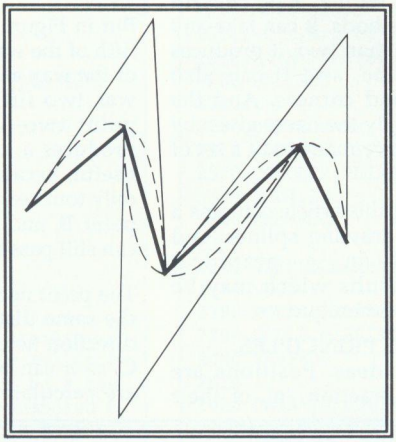


Figure 4

Since the bows overlap each other, we need only concentrate on the first half of each bow. If the routine calculates points between the first half of each bow, and in the same routine, the second half of the one before, then it will process every half-bow and thus work through the entire curve.

The variable 'm' is used to define the fraction of the distance of a point half way along each bow. 'm' is set equal to one at the centre of a bow, or zero at its start. Essentially it's 'n' multiplied by two.

So any positions on the final curve can be specified from the previous point J%, plus 'm', the fraction of its distance from J% to J%+1. From this, we can calculate the xy-position of the final curve anywhere along its length.

Take the positions of the two half bows at J% plus an 'm'th. They are always on each side of the final smooth curve. Add both pairs of co-ordinates in proportion to the current distance from J% to J%+1. The fraction of each to make up the final values is defined by 'm'.

So the position of the final smooth curve is always 'm' times the xy-values one 'm'th of the way along the current half-bow, plus '1-m' times the xy-values of the corresponding part of the previous half-bow (in the program 'M' holds the value '1-m'). But try it with a pencil and some scrap paper. It's clearer!

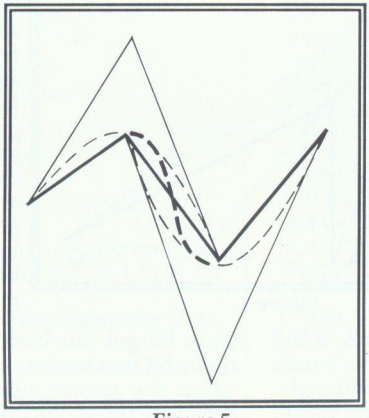


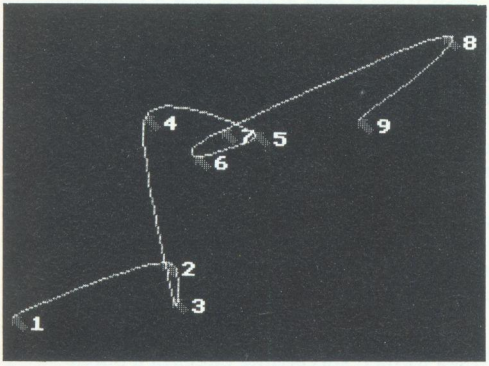
Figure 5

USING THE PROGRAM

The theory may sound complicated but running the demo program will soon show you what it's all about. Type it in as listed and save to disc or tape. When run the program first puts some fairly random points into the arrays X%() and Y%(), marks them on the screen with red arrows (CHR$(244)) and numbers them. The program then calls PROCspline to calculate and draw a continuous curve passing through all the points. Pressing any key will repeat the demonstration with a different set of random points.

If you want to incorporate PROCspline (and its associated procedures and functions) in your own programs simply allocate the co-

ordinates of the relevant points to the arrays X%() and Y%(), and the number of points used to the variable Len%. Calling PROCspline will then draw a spline through those points.



The STEP of the two 'FOR n' loops, in lines 1070 and 1120, determines the fraction of each bow at which a new point is calculated. As usual, you can trade speed for accuracy. Values of 1/4, 1/8, 1/16 etc work best.

PROCsimplebow draws the points of a simple bow, and PROChalfbow combines the positions of two consecutive half-bows to produce the final curve as it leaves the path of the first bow to follow the second. They all need FNbowX() and FNbowY() to give the X and Y values of a point on a simple bow.

```
  10 REM Program Spline
  20 REM Version B 1.0
  30 REM Author  Roger Burg
  40 REM BEEBUG  June 1988
  50 REM Program subject to copyright
  60 :
 100 MODE 1:ON ERROR GOTO 250
 110 VDU23,244,&F0F8;-8,-4,190,31,14,4
 120 DIM X%(20),Y%(20)
 130 DIM apeX%(20),apeY%(20)
 140 REPEAT:CLS
 150 Len%=RND(7)+2
 160 FOR J%=1 TO Len%:GCOL0,1
 170 X%(J%)=(J%*70)+RND(450)+X%(J%-1)/4
 180 Y%(J%)=(J%*50)+RND(400)+Y%(J%-1)/4
 190 MOVE X%(J%),Y%(J%)
 200 VDU5,244,20,J%+48,4
 210 NEXT J%
 220 GCOL0,2:PROCspline
 230 UNTIL GET=0
 240 :
 250 ON ERROR OFF:MODE 3
 260 REPORT:PRINT" at line ";ERL
 270 END
 280 :
1000 DEF PROCspline:LOCAL J%, n,N,m,M
1010 FOR J%=1 TO Len%
1020 apeX%(J%)=((X%(J%)*4)-X%(J%+1)-X%(
J%-1))/2
1030 apeY%(J%)=((Y%(J%)*4)-Y%(J%+1)-Y%(
J%-1))/2
1040 NEXT
1050 MOVE X%(1),Y%(1)
1060 FOR J%=1 TO Len%-2
1070 FOR n=0 TO 0.5 STEP 0.03125
1080 N=1-n:m=n*2:M=1-m
1090 IF J%=1 PROCsimplebow(J%)
1100 IF J%>1 PROChalfbow
1110 NEXT:NEXT
1120 FOR n=0.5 TO 1 STEP 0.03125
1130 PROCsimplebow(J%-1)
1140 NEXT n
1150 DRAW X%(J%+1),Y%(J%+1)
1160 ENDPROC
1170 :
1180 DEF PROCsimplebow(J%)
1190 DRAW FNbowX(J%,n),FNbowY(J%,n)
1200 ENDPROC
1210 :
1220 DEFPROChalfbow
1230 LOCAL BowX,BowX2,BowY,BowY2
1240 BowX =m*FNbowX(J%,n)
1250 BowX2=M*FNbowX(J%-1,n+.5)
1260 BowY =m*FNbowY(J%,n)
1270 BowY2=M*FNbowY(J%-1,n+.5)
1280 DRAW BowX+BowX2,BowY+BowY2
1290 ENDPROC
1300 :
1310 DEFFNbowX(J%,n)
1320 LOCAL X%,X2%,apeX%
1330 X%=X%(J%):X2%=X%(J%+2)
1340 apeX1%=apeX%(J%+1)
1350 =((X%+(n*(apeX1%-X%)))*(1-n))+((ap
eX1%+(n*(X2%-apeX1%)))*n)
1360 :
1370 DEFFNbowY(J%,n)
1380 LOCAL Y%,Y2%,apeY1%
1390 Y%=Y%(J%):Y2%=Y%(J%+2)
1400 apeY1%=apeY%(J%+1)
1410 =((Y%+(n*(apeY1%-Y%)))*(1-n))+((ap
eY1%+(n*(Y2%-apeY1%)))*n)
```

B

Spreadsheets In Practice

Alan Prior explains how spreadsheets can be applied to the small-scale bookkeeping needs of many clubs, societies and similar organisations.

Treasurers of small clubs and societies often keep day-to-day records using a method known as Analysis Cash Bookkeeping. In this system, transactions are entered into a book, with income and expenditure items recorded on separate pages, usually facing each other. Each transaction is identified by date, detail and often a voucher, or receipt number, and also whether it is a cash or bank transaction. A second entry on the same line analyses transactions by type, and the amount is repeated under the relevant column. In the simple example here no immediate indication of cash or bank balance is provided.

These details were compiled for a small choir which has regular rehearsals and gives a few concerts each year. Members pay subscriptions and income is derived from other activities. Each week a hall is hired and a conductor and accompanist are paid.

This type of presentation is well suited for implementation using a spreadsheet, which is effectively a large squared grid onto which data can be entered or manipulated, and in fact the example shows how the data might be laid out in a spreadsheet. There are many articles and books about spreadsheets available (BEEBUG Vol.3 No.3 discusses Ultracalc and Viewsheet), and a basic knowledge of spreadsheet operation is assumed in the rest of this article. Note that the Income and Expenditure pages shown may be placed adjacent in the sheet as if they formed succeeding pages of a book.

The expressions Csh>Bnk and Bnk>Csh used in the example need explanation. Csh>Bnk is used to show the payment of *cash* into a *bank* account, and appears on both Income and Expenditure sides of the book since the activity is both a *cash* expenditure and a *bank* income. In addition it is analysed on the Income side of the book. In a similar way, Bnk>Csh is used to

[*******	********	********	I N C O M E		********	********	*******]
INCOME DATE	INCOME DETAIL	INCOME CASH	INCOME BANK	INCOME CSH>BNK	INCOME SUBS	INCOME CONCERTS	INCOME OTHER
04JAN88	Open.Bal	28.30	325.70				354.00
11JAN88	Subs	41.00	163.50		204.50		
12JAN88	Tickets	18.00	42.00			60.00	
12JAN88	Csh>Bnk	100.00	100.00				
19JAN88	Subs	15.00	43.00		58.00		
19JAN88	TeaFund	8.34					8.34
19JAN88	Bnk>Csh	35.00					

[*******	********	********	E X P E N D I T U R E **		********	********	*******]
EXPEND. DATE	EXPEND. DETAIL	EXPEND. CASH	EXPEND. BANK	EXPEND. BNK>CSH	EXPEND. REHEARS.	EXPEND. CONCERTS	EXPEND. ADMIN.
06JAN88	Hall	10.00			10.00		
06JAN88	Pro.Fees		30.00		30.00		
07JAN88	Postage	3.68					3.68
12JAN88	Csh>Bnk	100.00					
12JAN88	Church		200.00			200.00	
19JAN88	Bnk>Csh		35.00	35.00			

show when a cheque is drawn on the bank for *cash:* again entries are made on both sides of the book and analysis takes place on the Expenditure side.

By using commands such as SUM(C4,C10) or the equivalent for the spreadsheet in use, it is possible to calculate the total of entries in individual columns. With this information cash and bank balances can be obtained by subtracting the expenditure total from the income total, as in the second example. This also contains a continuous check on the balance of the book to minimise the risk of mistakes being made in entries. This has proved very useful in practice, and is similar to the test an auditor would make when examining the book at the end of the financial year.

both sides a single checking slot can be produced; this value will always be zero after an entry is made correctly.

The presentation of the screen display can be made simpler by setting screen windows so that the column titles are not lost during vertical scrolling. Similarly the cash, bank and total balances can be fixed at the foot of the display, together with the overall check indication.

If the spreadsheet being used has the facility to lock screen windows together during sideways scrolling, this should be done with the title and data windows, so that when scrolling from Income to Expenditure the column titles remain correct.

```
[******* ******** ********     I N C O M E     ******** ******** *******]
INCOME   INCOME   INCOME   INCOME   INCOME    INCOME   INCOME   INCOME
DATE     DETAIL   CASH     BANK     CSH>BNK   SUBS     CONCERTS OTHER

         Totals   145.64   674.20   100.00    262.50    60.00   362.34
                  Cash     Bank     Total
         Balance   31.96   409.20   441.16

    Cash + Bank  =  819.84    Analysis Total  =  684.84
Csh+Bnk -Contra  =  684.84    Check =    0.00

[******* ******** ********  E X P E N D I T U R E  ** ******** *******]
EXPEND.  EXPEND.  EXPEND.   EXPEND.  EXPEND.   EXPEND.  EXPEND.  EXPEND.
DATE     DETAIL   CASH     BANK     BNK>CSH   REHEARS. CONCERTS ADMIN.

         Totals   113.68   265.00    35.00     40.00   200.00     3.68

    Cash + Bank  =  378.68    Analysis Total  =  243.68
Csh +Bnk -Contra =  243.68    Check =    0.00   Overall Check =      0.00
```

The check consists of making sure that the total Income (Cash + Bank) less the Contra (Cash to Bank and Bank to Cash) entries equals the total of the Income Analysis columns. A similar check is made for the Expenditure side. The beauty of using a spreadsheet, of course, is that this can be entirely automatic once set up.

This checking process will ensure that the analysis columns are correctly completed since any error will show up as a difference between the relevant figures on the same side of the book. The process can be further improved by subtracting one value from the other and looking for a zero result, indicating all is correct. By adding the results of the checks on

Either side of the book may be extended to provide further columns and it is convenient to scroll sideways one screen at a time from Income to Expenditure and then Expenditure Continued, for example. In this case it may help to repeat the date column on each page.

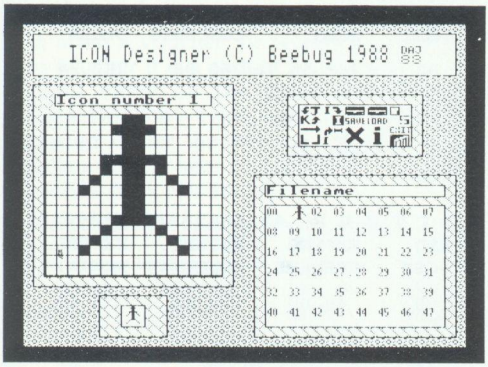
A common spreadsheet for use on the Beeb is Acornsoft's ViewSheet, and the print out of slot contents given here for the example sheets refers to this. The convention C4C15 for column addition means add the contents of slots C4 to C15. A column width of 8 has been used, with Format D2RM, (decimal places shown, aligned right, minus sign when negative).

Continued on page 18

THE BEEBUG MINI WIMP Part 2

In the second part of our MiniWimp series, David James presents a window-based icon designer.

The utility listed here is an icon designer that can be used to design icons for use with the MiniWimp ROM given last month. The icon designer uses MiniWimp for managing its own window displays, but we will be discussing the use of the MiniWimp itself in more detail next time. The MiniWimp code must be installed in sideways RAM before running the icon designer (see last month for details of this).



When you run the icon designer (after typing in and saving the program), you will be asked whether there are any icons in memory. Press 'Y' if there are some icons loaded that are to be edited. If you press 'N', you will be further prompted for confirmation, and if you answer 'Y' the icon memory will be cleared. When in the designer, the left side of the screen shows a 16 by 16 grid used to enter the icon, with the icon being edited shown actual size below the grid. To the right of the screen there are two rows of five icons that are used to control the designer, and below them a window showing the icons numbered 0 to 47. The icon designer can't be used to edit icons 48 to 63 because it uses those itself. If necessary, these must be edited as different-numbered icons and then loaded in as numbers 48 upwards.

An icon is edited by moving the pointer around the grid and clicking (with Copy or the Fire button) to invert the colour of the square pointed to. To store the contents of the grid as one of the icons, move the pointer to the icon to be updated in the lower right hand window and click. The ten icons in the control window can be clicked on to perform certain operations. From left to right, top row first, the first control switches between keyboard and joystick control (the designer cannot be used with a mouse). If you select the joystick option when no joystick is connected you will have to press Escape to recover control. The second control inverts the edit window (black becomes white and vice versa). To save a range of icons, click on the third control, then on the start and end of the range to be saved, and finally enter the filename. Similarly, to load a group of icons click on the fourth control and then on the start of the range, and enter the filename. The last control on the top row allows an operating system command to be entered. The first control on the bottom row mirrors the left-hand half of the edit window onto the right-hand half, while the seventh control rotates the edit window through 90 degrees clockwise, and the eighth control clears the grid. To edit an existing icon click on the ninth control followed by the icon to be edited, and it will be copied into the grid. Finally, the last control is used to quit from the designer.

Next month we conclude the BEEBUG MiniWimp with some further examples of its use.

```
   10 REM Program  ICON DESIGNER
   20 REM Version  B1.0
   30 REM Author   David James
   40 REM BEEBUG   June 1988
   50 REM Program  subject to copyright
   60 :
  100 MODE4:HIMEM=&4E00:O%=&5760:*FX4,1
  110 ENVELOPE 1,1,0,0,0,1,1,1,126,0,0,-
1,126,0
  120 PROCassemble:PROCdefineicons
  130 ON ERROR PROCerrorhandle:GOTO160
  140 CLS:PROCcursor(FALSE):*MWSETUP
  150 OSCLI"MWKEY":PROCsetupscreen
  160 PROCwindow(22,29,37,18)
  170 PROCwindow(9,29,10,28)
  180 PROCfill(2,24,17,9,CHR$154)
  190 PROCdrawoptions:PROCclearbank
  200 FOR V%=0 TO 31:O%?V%=0:NEXT
  210 joy%=0:new%=0:load%=0:save%=0
```

```
  220 OSCLI"MWCOORD 2,2":*FX 202,48
  230 REPEAT *MWPOINTER
  240 IF FNarea(25,11,34,8) PROCoption
  250 IFFNarea(2,24,17,9)PROCtogglepixel
  260 IF FNarea(22,29,37,18)PROCiconbank
  270 UNTIL FALSE
  280 :
 1000 DEF PROCdefineicons
 1010 FOR V%=0TO79:READ V%?ntable%:NEXT
 1020 FOR I%=48 TO 58:FOR J%=0 TO 1
 1030 READ A$:FOR K%=0 TO 15
 1040 ?(&5000+32*I%+16*J%+K%)=EVAL("&"+M
ID$(A$,1+K%*2,2)):NEXT K%,J%,I%:ENDPROC
 1050 :
 1060 DEF PROCassemble
 1070 oswrch=&FFEE:osword=&FFF1
 1080 FOR I%=0 TO 2 STEP 2
 1090 P%=&4E00:[OPT I%
 1100 .background
 1110 LDA #0:STA &73:LDA #&58:STA &74
 1120 .b1 LDY #0:.b2 LDA block%,Y
 1130 STA (&73),Y:INY:CPY #8:BNE b2
 1140 TYA:CLC:ADC &73:STA &73
 1150 LDA #0:ADC &74:STA &74
 1160 CMP #&80:BNE b1:RTS
 1170 .pr LDX #0:.P1 LDA #31:JSR oswrch
 1180 TXA:AND #8:CLC:ADC #2:JSR oswrch
 1190 TXA:AND #16:CLC:ROR A:CLC:ADC #9
 1200 STA &73:TXA:AND #7:CLC:ADC &73
 1210 JSRoswrch:LDA O%,X:STA &73:TXA:PHA
 1220 LDX #8:.P2 CLC:ROL &73:PHP:PLA
 1230 AND #1:CLC:ADC #154:JSR &FFEE
 1240 DEX:BNE P2:PLA:TAX
 1250 INX:CPX #32:BNE P1:RTS
 1260 .rot LDA #(O% MOD 256):STA &73
 1270 LDA #(O% DIV 256):STA &74
 1280 JSR rot8bytes:JSR rot8bytes
 1290 JSR rot8bytes:JSR rot8bytes
 1300 LDY #0:.R5 LDA O%,Y:PHA
 1310 LDA O%+16,Y:STA O%,Y
 1320 LDA O%+24,Y:STA O%+16,Y
 1330 LDA O%+8,Y:STA O%+24,Y
 1340 PLA:STA O%+8,Y
 1350 INY:CPY #8:BNE R5:RTS
 1360 .rot8bytes
 1370 LDA &73:STA R2+1:LDA &74:STA R2+2
 1380 LDY #0:.R1 LDA #0:LDX #0:.R2
 1390 ASL &FFFF,X:ROR A:INX:CPX#8:BNE R2
 1400 STA block%,Y:INY:CPY #8:BNE R1
 1410 LDY #0:.R3 LDA block%,Y:STA(&73),Y
 1420 INY:CPY #8:BNE R3
 1430 LDA &73:CLC:ADC #8:STA &73
 1440 LDA &74:ADC#0:STA &74:RTS
 1450 .invert LDX &76:.I1 LDY #0
 1460 .I2 LDA (&73),Y:EOR #&FF
 1470 STA (&73),Y:INY:CPY &75:BNE I2
 1480 LDA &73:CLC:ADC #&40:STA &73
 1490 LDA &74:ADC #1:STA &74
 1500 DEX:BNE I1:RTS
```

```
 1510 .do STA block%:LDX #block% AND &FF
 1520 LDY#block%/256:LDA #10:JSR osword
 1530 LDX #0:LDY #0:.D1
 1540 LDA #23:JSR oswrch:TXA:CLC
 1550 ADC #158:JSR oswrch:.D2
 1560 LDA block%+1,Y:ROR A
 1570 AND block%+1,Y:JSR oswrch
 1580 JSR oswrch:INY:TYA:AND #3
 1590 BNE D2:INX:CPX #2:BNE D1:RTS
 1600 .block%EQUD&B57EE7DA:EQUD&5BE77EAD
 1610 .ntable%:]:NEXT
 1620 ENDPROC
 1630 :
 1640 DEF PROCtogglepixel
 1650 IF (new% OR load% OR save%) SOUND
1,-15,99,5:GOTO 1700
 1660 X1%=X%-2:Y1%=Y%-9
 1670 B%=16*(Y1% DIV 8)+8*(X1% DIV 8)+Y1
% MOD 8
 1680 M%=2^(7-X1%MOD8):O%?B%=O%?B%EOR M%
 1690 VDU31,X%,Y%,154-((O%?B% ANDM%)<>0)
 1700 PROCdelay(.1)
 1710 PROCcurrent:ENDPROC
 1720 :
 1730 DEF PROCiconbank
 1740 IF new% PROCcopyfrombank:ENDPROC
 1750 IF load% PROCloadicons:ENDPROC
 1760 IF save% PROCsaveicon:ENDPROC
 1770 IN%=FNiconnu
 1780 X%=2*INT(X%/2):Y%=2*INT(Y%/2)
 1790 ADR%=&5000+IN%*32
 1800 FOR V%=0TO31:ADR%?V%=O%?V%:NEXT
 1810 VDU 31,X%,Y%:*MWICON 59
 1820 PRINTTAB(15,7)"  ";TAB(15,7);IN%
 1830 SOUND 1,1,200,3:ENDPROC
 1840 :
 1850 DEF PROCcurrent:VDU 31,9,28
 1860 OSCLI"MWICON 59":ENDPROC
 1870 :
 1880 DEF PROCcopyfrombank
 1890 SOUND 1,1,180,1
 1900 IN%=FNiconnu:ADR%=&5000+32*IN%
 1910 FOR V%=0 TO 28 STEP 4
 1920 V%!O%=V%!ADR%:NEXT:CALL pr
 1930 PROCinvopt:PRINTTAB(15,7);IN%
 1940 PROCcurrent:new%=0:ENDPROC
 1950 :
 1960 DEF PROCclearbank
 1970 PROCwindow(24,22,35,20)
 1980 PRINTTAB(25,21)"OLD ICONS?"
 1990 PROCwindow(24,27,28,25)
 2000 PRINTTAB(25,26)"YES"
 2010 PROCwindow(31,27,35,25)
 2020 PRINTTAB(32,26)"NO":*FX 15,1
 2030 REPEAT G%=GET AND &DF
 2040 YN%=INSTR("NY",CHR$G%):UNTIL YN%
 2050 no%=YN%-2:IFno% no%=no% AND FNsure
 2060 IF no% PROCinvscr(31,27,35,25):SOU
ND 1,1,200,3:FOR V%=&5000 TO &55FC STEP
```

```
4:!V%=0:NEXT V%:ELSE PROCinvscr(24,27,28
,25):SOUND 1,1,100,3
 2070 PROCwindow(22,29,37,18)
 2080 PROCprintbank:ENDPROC
 2090 :
 2100 DEF PROCprintbank
 2110 FOR G%=0 TO 47
 2120 VDU 31,22+2*(G% MOD 8),18+2*(G% DI
V 8)
 2130 ADR%=&5000+G%*32:S%=0
 2140 FOR G1%=0 TO 28 STEP 4
 2150 S%=S% OR G1%!ADR%:NEXT G1%
 2160 IF S%=0 PROCprintnumber(G%) ELSE O
SCLI"MWICON "+STR$G%
 2170 NEXT G%:ENDPROC
 2180 :
 2190 DEF PROCoption:X1%=X%-25:Y1%=Y%-8
 2200 IF (new% OR load% OR save%) THEN S
OUND 1,-15,99,5:GOTO 2330
 2210 OPT%=5*(Y1% DIV 2)+(X1% DIV 2)
 2220 SOUND 1,1,120+10*OPT%,3
 2230 IF OPT%=0 PROCjoykey
 2240 IF OPT%=1 PROCinvert
 2250 IF OPT%=2 save%=1:PROCinvopt
 2260 IF OPT%=3 load%=1:PROCinvopt
 2270 IF OPT%=4 PROCopsyst
 2280 IF OPT%=5 PROCreflect
 2290 IF OPT%=6 PROCrotate
 2300 IF OPT%=7 PROCwipe
 2310 IF OPT%=8 PROCnewicon
 2320 IF OPT%=9 PROCexit
 2330 PROCdelay(.1)
 2340 ENDPROC
 2350 :
 2360 DEF PROCjoykey:joy%=NOT joy%
 2370 IF joy% THEN *MWSTICK ELSE *MWKEY
 2380 ENDPROC
 2390 :
 2400 DEF PROCinvert
 2410 FOR V%=0 TO 28 STEP 4
 2420 O%!V%=NOT O%!V%:NEXT
 2430 CALL pr:PROCcurrent:ENDPROC
 2440 :
 2450 DEF PROCsaveicon
 2460 IF save%=2 GOTO 2500
 2470 save%=2:from%=FNiconnu
 2480 SOUND 1,1,190,3:PROCdelay(.2)
 2490 ENDPROC
 2500 to%=FNiconnu
 2510 IF to%<from% SOUND 1,-15,99,5:ENDP
ROC
 2520 SOUND 1,1,190,3
 2530 save%=FALSE:F$=FNfilename
 2540 OSCLI"MWOPEN 20,25,30,20":PRINT"Fi
lename:-"'F$:PROCcursor(TRUE)
 2550 OSCLI "MWSAVE "+F$+" "+STR$from%+"
 "+STR$to%
 2560 PROCcursor(0):PROCinvopt:*MWSHUT
 2570 ENDPROC
 2580 :
 2590 DEF PROCloadicons:SOUND 1,1,210,3
 2600 IN%=FNiconnu:F$=FNfilename
 2610 OSCLI"MWOPEN 20,25,30,20":PRINT"Fi
lename:-"'F$:PROCcursor(TRUE)
 2620 OSCLI "MWLOAD "+F$+" "+STR$IN%
 2630 PROCcursor(FALSE):*MWSHUT
 2640 load%=0:PROCwindow(22,29,37,18)
 2650 PROCprintbank:PROCinvopt:ENDPROC
 2660 :
 2670 DEF PROCopsyst:PROCinvopt
 2680 *MWOPEN 3,30,22,5
 2690 PROCcursor(TRUE):*FX 15,1
 2700 PRINT "*";:INPUT "" OS$
 2710 PROCcursor(0):OSCLIOS$
 2720 PRINT'"<Press SPACE>":*FX 15,1
 2730 REPEAT UNTIL GET=32:*MWSHUT
 2740 PROCinvopt:ENDPROC
 2750 :
 2760 DEF PROCreflect
 2770 IF NOT FNsure ENDPROC
 2780 PROCinvopt:FOR J%=0TO16 STEP16
 2790 FOR J1%=0 TO 7:M%=O%?(J%+J1%)
 2800 M%=-128*((M%AND1)<>0)-64*((M%AND2)
<>0)-32*((M%AND4)<>0)-16*((M%AND8)<>0)-8
*((M%AND16)<>0)-4*((M%AND32)<>0)-2*((M%A
ND64)<>0)-((M%AND128)<>0)
 2810 O%?(J%+J1%+8)=M%:NEXT,
 2820 CALL pr:PROCcurrent:PROCinvopt
 2830 ENDPROC
 2840 :
 2850 DEF PROCrotate CALLrot
 2860 CALLpr:PROCcurrent
 2870 ENDPROC
 2880 :
 2890 DEF PROCwipe IF NOT FNsure ENDPROC
 2900 FOR V%=0 TO 28 STEP 4
 2910 O%!V%=0:NEXT V%
 2920 CALL pr:PROCcurrent:ENDPROC
 2930 :
 2940 DEF PROCnewicon
 2950 IF NOT FNsure ENDPROC
 2960 PROCinvopt:PRINT TAB(15,7)"  "
 2970 FOR V%=0 TO 28 STEP 4:O%!V%=0
 2980 NEXT V%:CALL pr:new%=-1:ENDPROC
 2990 :
 3000 DEF PROCexit
 3010 IF NOT FNsure ENDPROC
 3020 COLOUR 128:COLOUR 1:*FX 15,1
 3030 CLS:*FX 4
 3040 *FX 202,32
 3050 VDU20:END
 3060 :
 3070 DEF PROCsetupscreen
 3080 PROCcursor(FALSE)
 3090 CALL background
 3100 VDU 23,154,128,128,128,128,128,128
,128,255,23,155,255,255,255,255,255,255,
255,255,23,156,170,85,170,85,170,85,170,
```

```
85,23,157,136,80,32,16,8,4,2,5
 3110 PROCwindow(1,4,38,1)
 3120 VDU 31,4,2:PROCdoub("ICON Designer
 (C) Beebug 1988  ")
 3130 VDU 31,34,2:*MWICON 58
 3140 PROCfill(1,25,18,6,CHR$157)
 3150 PROCwindow(3,7,16,7):VDU 31,3,7:PR
INT "Icon number   "
 3160 PROCfill(21,30,38,15,CHR$157)
 3170 PROCwindow(22,16,37,16):VDU 31,22,
16
 3180 PRINT"Filename        "
 3190 PROCfill(24,12,35,7,CHR$157)
 3200 PROCwindow(25,11,34,8)
 3210 PROCfill(7,30,12,27,CHR$157)
 3220 ENDPROC
 3230 :
 3240 DEF PROCbox(X1%,Y1%,X2%,Y2%)
 3250 x1%=32*X1%-4:x2%=32*X2%+32
 3260 y1%=((31-Y1%)*32)-4:y2%=((31-Y2%)*
32)+32
 3270 GCOL 0,0:MOVE x1%,y1%:DRAW x1%,y2%
 3280 DRAW x2%,y2%:DRAW x2%,y1%
 3290 DRAW x1%,y1%:ENDPROC
 3300 :
 3310 DEF PROCwindow(X1%,Y1%,X2%,Y2%)
 3320 PROCbox(X1%,Y1%,X2%,Y2%)
 3330 VDU 28,X1%,Y1%,X2%,Y2%
 3340 COLOUR 129:COLOUR 0:CLS:VDU26
 3350 ENDPROC
 3360 :
 3370 DEFPROCfill(X1%,Y1%,X2%,Y2%,fill$)
 3380 PROCbox(X1%,Y1%,X2%,Y2%)
 3390 fill$=STRING$(X2%-X1%+1,fill$)
 3400 FOR F%=Y2% TO Y1%
 3410 PRINT TAB(X1%,F%)fill$
 3420 NEXT:ENDPROC
 3430 :
 3440 DEF PROCdoub(A$)
 3450 FOR a%=1 TO LEN A$
 3460 A%=ASC(MID$(A$,a%,1)):CALL do
 3470 VDU 158,10,8,159,11:NEXT:ENDPROC
 3480 :
 3490 DEF PROCcursor(C%)
 3500 VDU 23,1,ABS(C%);0;0;0;
 3510 ENDPROC
 3520 :
 3530 DEF PROCprintnumber(N%)
 3540 D%=N% DIV 10:U%=N% MOD 10
 3550 VDU 23,153
 3560 adrd%=ntable%+8*D%
 3570 adru%=ntable%+8*U%
 3580 FOR V%=0 TO 7
 3590 VDU 16*(adrd%?V%)+(adru%?V%)
 3600 NEXT V%:VDU 153:ENDPROC
 3610 :
 3620 DEFPROCinvscr(X1%,Y1%,X2%,Y2%)
 3630 !&73=&5800+&140*Y2%+8*X1%
 3640 ?&75=8*(X2%-X1%+1)
 3650 ?&76=Y1%-Y2%+1:CALLinvert:ENDPROC
 3660 :
 3670 DEF PROCinvopt
 3680 X1%=25+2*(OPT% MOD 5)
 3690 Y1%=8+2*(OPT% DIV 5)
 3700 PROCinvscr(X1%,Y1%+1,X1%+1,Y1%)
 3710 ENDPROC
 3720 :
 3730 DEF PROCdelay(T)
 3740 TIME=0:REPEAT UNTIL TIME>T*100
 3750 ENDPROC
 3760 :
 3770 DEF PROCerrorhandle
 3780 *MWKEY
 3790 *MWOPEN 8,17,30,15
 3800 SOUND 1,1,100,3:*FX15,1
 3810 PRINT"Error:-";:REPORT
 3820 PRINT ;".":PRINT"<Press SPACE>";
 3830 REPEAT UNTIL GET=32
 3840 FOR N%=1 TO ?&87:*MWSHUT
 3850 NEXT:ENDPROC
 3860 :
 3870 DEF PROCdrawoptions
 3880 FOR N%=0 TO 9
 3890 VDU 31,25+2*(N% MOD 5),8+2*(N% DIV
 5):OSCLI"MWICON "+STR$(N%+48)
 3900 NEXT N%:ENDPROC
 3910 :
 3920 DEF FNarea(X1%,Y1%,X2%,Y2%)
 3930 =(X%>=X1% AND X%<=X2% AND Y%<=Y1%
AND Y%>=Y2%)
 3940 :
 3950 DEF FNiconnu
 3960 X%=2*INT(X%/2):Y%=2*INT(Y%/2)
 3970 X1%=X%-22:Y1%=Y%-18
 3980 =8*(Y1% DIV 2)+(X1% DIV 2)
 3990 :
 4000 DEF FNsure
 4010 X1%=10+RND(10):Y1%=10+RND(5)
 4020 OSCLI "MWOPEN "+STR$X1%+" "+STR$(Y
1%+5)+" "+STR$(X1%+14)+" "+STR$(Y1%)
 4030 VDU 31,1,2:PROCdoub("Are you sure?
")
 4040 *FX 15,1
 4050 REPEAT G%=GET AND &DF
 4060 S%=INSTR("NY",CHR$G%):UNTIL S%
 4070 *MWSHUT
 4080 =1-S%
 4090 :
 4100 DEF FNfilename
 4110 A%=202:X%=0:Y%=&FF:X%=(USR&FFF4 AN
D &FF00)DIV&100:*FX 202,32
 4120 VDU 31,31,16:PRINT "        "
 4130 VDU 31,31,16:PROCcursor(TRUE):*FX
15,1
 4140 A$="":REPEAT:REPEAT K%=INKEY(5)
 4150 UNTIL(LENA$<7AND(K%>32ANDK%<126))O
R(K%=13ORK%=127AND(LENA$>0ANDLENA$<8))
 4160 IF K%=127 VDU 8,32,8:A$=LEFT$(A$,L
```

```
ENA$-1) ELSE IF NOT(K%=13) VDU K%:A$=A$+
CHR$K%
 4170 UNTIL K%=13:Y%=0:CALL&FFF4
 4180 PROCcursor(FALSE):=A$
 4190 :
 4200 DATA 0,2,5,5,5,5,5,2,0,2,6,2,2,2,2
,7,0,6,1,1,1,2,4,7,0,6,1,1,2,1,1,6,0,1,3
,5,5,7,1,1,0,7,4,4,7,1,1,6,0,1,2,4,6,5,5
,2,0,7,1,1,1,2,2,2,0,2,5,5,2,5,5,2,0,2,5
,5,3,1,1,6
 4210 DATA 000E1E187E3C180000FEFE1818189
870,666C78786C66630000183C7E18787000
 4220 DATA 003C181818183C0000F0F8187E3C1
800,00000000000000000FFC3E7E7E7E7C3FF
 4230 DATA 007F7F7F407E7F7F00FEFEFE027EF
AFE,00007745771575000000565456542600
```

```
 4240 DATA 007F7F7F407E7F7F00FEFEFE027EF
AFE,00004E4A4A4A6E000000ECAAEAAAAC00
 4250 DATA 007F41414343437F0000000000000
000,00000000000000000FE8080FE0606FE00
 4260 DATA 00007F7F000000060080CFEFE0C080
006,60606060607E7E0006060606067E7E00
 4270 DATA 0004061F3F3634300000427E7E420
000,00783030303078000000000000000000
 4280 DATA 0010387C3E1F0F0700081C3E7CF8F
0E0,070F1F3E7C381000E0F0F87C3E1C0800
 4290 DATA 0003030300070703000C0C0C000C0C
0C0,03030303030707000C0C0C0C0C0E0E000
 4300 DATA 007545624575007F0077222222720
0FE,7F7F787176747474E20A3AAAAAAAAAAA
 4310 DATA 00714A4A4B4A7200009E4242C2524
C00,3C42423C42423C003C42423C42423C00
```

SPREADSHEETS IN PRACTICE *(continued from page 13)*

Slot	Contents	Slot	Contents	Slot	Contents	Slot	Contents
A1	[*******	I2	EXPEND.	B16	Totals	F20	F16+G16+H16
B1	********	J2	EXPEND.	C16	C4C15	I20	Cash
C1	********	K2	EXPEND.	D16	D4D15	J20	+ Bank =
D1	I N C	L2	EXPEND.	E16	E4E15	K20	K16+L16
E1	O M E	M2	EXPEND.	F16	F4F15	L20	Analysis
F1	********	N2	EXPEND.	G16	G4G15	M20	Total =
G1	********	O2	EXPEND.	H16	H4H15	N20	N16+O16+P16
H1	*******]	P2	EXPEND.	J16	Totals	A21	Csh+Bnk
I1	[*******	A3	DATE	K16	K4K15	B21	-Contra=
J1	********	B3	DETAIL	L16	L4L15	C21	C20-E16-M16
K1	********	C3	CASH	M16	M4M15	I21	Csh +Bnk
L1	E X P E	D3	BANK	N16	N4N15	J21	-Contra=
M1	N D I T	E3	CSH>BNK	O16	O4O15	K21	K20-E16-M16
N1	U R E **	F3	SUBS	P16	P4P15	E17	Total
O1	********	G3	CONCERTS	C17	Cash	E18	C18+D18
P1	*******]	H3	OTHER	D17	Bank	D21	Check =
A2	INCOME	I3	DATE	B18	Balance	E21	F20-C21
B2	INCOME	J3	DETAIL	C18	C16-K16	L21	Check =
C2	INCOME	K3	CASH	D18	D16-L16	M21	N20-K21
D2	INCOME	L3	BANK	A20	Cash +	N21	Overall
E2	INCOME	M3	BNK>CSH	B20	Bank =	O21	Check =
F2	INCOME	N3	REHEARS.	C20	C16+D16	P21	E21-M21
G2	INCOME	O3	CONCERTS	D20	Analysis		
H2	INCOME	P3	ADMIN.	E20	Total =		

In general the size of spreadsheet available does not form any limitation on the size of book which can be kept using the method described. More sophisticated programs are readily available for the larger applications. However, two factors, storage limitation in the computer and the time taken to move around a full sheet when putting in entries may give problems. In this case remember that the spreadsheet can be used for periods of less than a year, and the final accounts divided into, say, three sections each covering four months' activities. This application of spreadsheets is intended for use by clubs, societies and similar organisations where only a relatively small number of entries are made each year, and the treasurer has perhaps a limited knowledge of accounting and accounting techniques. From personal experience it works, and the provision of continuous balances and the self check facility certainly save time.

MULTI-USER DUNGEONS

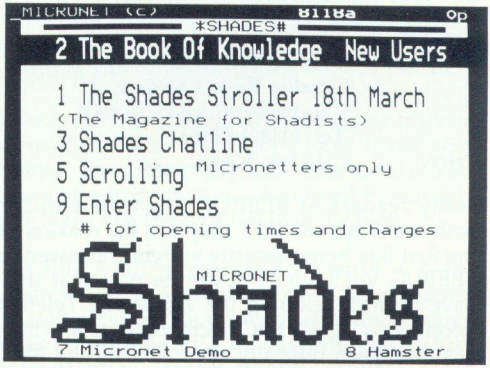
Richard Brunton, Quixote the poetical Monsignor in other worlds, introduces the uninitiated to the devious and addictive world of MUD - Multi-User Dungeons.

A Multi-User Dungeon is basically a vast text-only adventure game with several players (adventurers) participating at any one time. Adventurers choose personas (identities) for themselves and use these personas whenever they play the game. As they gain more points and increase their skill in the game, they become more powerful, gaining magical spells, and often the approbation and respect of other players.

The ultimate goal is not to quell evil, but to reach Immortal Status and become either a Witch or a Wizard. The road to this goal may be long and exhausting (especially on your pocket), depending on the level of your enthusiasm and addiction. But it provides ample reward in the feeling of achievement at having successfully fought many gruelling battles, something I can certainly vouch for having been one of the few to become Immortal.

In the adventure, the artificial landscape is littered with hidden traps, red herrings, powerful weapons and fabulous treasures. After solving complex puzzles, and avoiding blood-thirsty players and computer generated characters (Mobiles), you finally deposit the treasure collected along the way to increment your score. At timed intervals all Adventurers find that their connection to the Host Computer has been terminated and the treasure is restored, either to the original or randomly selected locations, forcing Adventurers to begin the quest once more.

At present, one of the most popular Dungeons is Micronet's "Shades" (Micronet is a Closed User Group on Prestel). Shades is a vast world of intrigue, danger and excitement, and to the players it is not just a game but a way of life. During the game you may collect treasure; take part in a quiz set by Immortals with treasure as the prize; make alliances with other adventurers; take part in a duel; fight with other adventurers; collect the treasure to increase your skill and power; chat with friends; celebrate Christmas, New Year, Easter, etc; you may even marry other adventurers! As part of the adventure, you may join the Shades Athletic Association and take part in a game of football, or in a great range of other athletic events. The list of activities is only hampered by your own imagination.



To interact with other people a wide range of commands is available. For instance you may wish to *shout, yell, tell* someone something, *sob, giggle, scream,* - the list is endless. To convey a feeling or action not described by the general commands a special command ***EMOTE*** is available. If I typed ***EMOTE*** *eats his hat and leaps up and down* it would appear to other adventurers who are at the same location in the adventure as *Quixote eats his hat and leaps up and down.* This provides an easy way to relay feelings or moods and allow you creativity in your movements and actions.

Multi-User Dungeons supply endless interactive entertainment with other REAL people, and this brief description is a mere drop in the ocean of their limitless fantasy world.

Unfortunately there is a price to pay, and addiction can be a soul-destroying experience when the quarterly account is received. Self control is vital, and a stop-watch is a worth-while asset. I look forward to meeting you in Shades in the future.

To enter Shades key *81181# or *81188# on Micronet. Micronet may be contacted (verbally) on 01-278 3143.

The GIS Teletext Adaptor

Some four years after the original Acorn teletext adaptor was released, GIS has launched the official replacement. Dave Somers takes a look at the new offering.

Product	**Advanced Teletext Receiver**
Supplier	**General Information Systems Ltd Croxton Park, Croxton, Cambridgeshire PE19 4SY. Tel. (0480) 87464**
Price	**£149 inc. VAT**

Many readers of BEEBUG might be familiar with the name of GIS - they are the makers of the Red Box home security system (reviewed in BEEBUG Vol.5 No.7). Over the past year they have been developing the Advanced Teletext Receiver (ATR) for BBC microcomputers, which has now been officially approved by the BBC, replacing the old Acorn system.

GETTING STARTED

The ATR consists of a cream-coloured injection moulded case measuring 8.5 by 4.25 by 1.25 inches, and for some reason a wall mounting bracket is supplied. Emerging from the base is a thick one metre cable with two plugs on the end: one attaches to the User Port on a model B or Master 128, the other to the auxiliary power supply output. The only other connector is a socket for an aerial lead at one side. If you are already using the power-out socket on the computer, a splitter will be needed. Such a device can be supplied by BEEBUG.

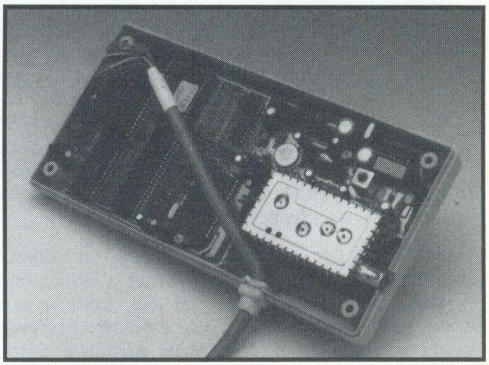
ATS+ ROM

Software to control the unit is supplied in the form of the Advanced Teletext System Plus (ATS+) ROM. This is a version of the ATS ROM produced by BBC Soft, which is now the standard software for all teletext adaptors on BBC computers. The ROM can either be fitted internally, put in a cartridge on a Master 128, or transferred to disc and loaded into sideways RAM when needed. Full instructions are supplied with the system.

TUNING IN

Just like a television set, the ATR needs to be tuned into your local transmitter. The ATR uses digital synthesised tuning which is completely automated - no more twiddling tuning wheels as with the old Acorn system. The software scans through all the TV channels, stopping when a recognisable signal is received. Details of this channel can then be stored and the next one programmed, if you so wish.

Details of the frequencies are stored in an EEPROM (Electrically Erasable Programmable Read-Only Memory) device in the ATR. Thus, once you have tuned all the channels the details are remembered, even if the power is disconnected.



Inside the GIS Teletext Adaptor

TERMINAL MODE

Teletext Terminal mode is entered by typing the command *TELETEXT, which makes the computer behave just like a Teletext Television. The top line shows the name of the Teletext Service (i.e. CEEFAX or ORACLE); the page number indicator of each page number as it is updated; and the date and time. To select a page, its number is entered, and when that page is transmitted it will be loaded into the computer and displayed. The ATS+ software uses the function keys to perform tasks such as: tuning the ATR, selecting the channel to view, hold, reveal, load page, and save page.

LINKED PAGES

At the bottom of the screen there may be a series of coloured boxes followed by page numbers or 'keywords'. These are called links, and are designed to allow fast access to pages

related to the current one. By pressing the appropriate function key, the page pointed to by a link will be loaded automatically. This is analogous to the routing of pages on Prestel, and is a feature not normally found on teletext TVs, except the new 'FastText' systems.

PAGE GRABBING

Because of the serial nature of teletext transmission, it can take up to a minute for the page you request to be received and displayed. To speed things up, the ATS+ software has a page grabber that keeps a list of up to twenty pages, and loads them as soon as they are transmitted. If you try and access a page that has been grabbed it will be displayed instantly. The software normally decides for you which pages to grab, on the basis of the last few pages looked at, and the links from the current page. It is however possible to override this, and decide for yourself which pages you want to catch as they come around. By pressing Return from within Terminal Mode the Terminal Status page can be displayed. This displays details of the current page, 'links' from it, and the pages in the Page Grabber. Alongside the numbers of the pages in the Grabber are a series of flags showing their individual status: page not yet found; found; updating; revealed; held; and kept. It is quite fascinating to watch as the flags are updated as the pages are transmitted.

LOW LEVEL CONTROL

For those who wish to control the ATR directly from within application languages this can be achieved through a number of star commands and low-level OSWORD routines.

For example, if you want to display the contents of Page 700 on BBC2, here is a simple Basic program to achieve this.

```
10 MODE 7
20 *TTXON
30 *BBC2
40 *PAGE 700
50 *TRANSFER 7000
60 *DISPLAY 7000
70 *TTXOFF
```

More useful programs could be written, for example, to read the pages containing the FT share prices and tell you how much your portfolio is worth. Full details of all the commands are given in the manual.

TELESOFTWARE

One of the main advantages of a teletext adaptor connected to a computer over a teletext TV is the ability to download data and programs. The ATS+ system makes downloading very easy; all that is needed is to select the channel and press f5. The ATS+ then searches for and displays a special page giving details of all the programs that may be currently downloaded. It is then a simple matter of moving a highlight over the file required and pressing Return.

Due to the way teletext works, the time taken to download a file can be quite long. It has been known for some really BIG files to take over one-and-a-half hours to download! Fortunately this is the exception rather than the rule.

On BBC2, all pages starting with a 7, for example 701, are for telesoftware and computer related services: computing news; forthcoming events; and details of current and future telesoftware. Many useful programs are available to download including ones to make use of the ATS software, such as an interactive personal portfolio manager and a frame grabber and printing utility. There are also downloadable teacher's notes for many schools programs, and of course everything is free.

FINAL THOUGHTS

In use the ATR behaved impeccably. Compared to the original Teletext Receiver, the ATR is a worthy successor. It took me under five minutes to set it up and begin receiving data - the automatic tuning was very impressive compared with other receivers, and the Page Grabber facility was most useful.

Documentation came in the form of a 118 page A5 manual. As it was the first issue, I spotted a number of mistakes and (minor) omissions, but these should be tidied up for the second issue. Nevertheless, it is still perfectly adequate for the average user.

All in all a most impressive piece of kit which I would have no hesitation in recommending. B

Smart Renumber

David Spencer presents a smart renumber utility to tidy up your Basic programs.

Program listings always look much nicer, and are easier to follow, if the various sections are split up, with different number ranges for each part. This is a style that we at BEEBUG have always adopted in our program listing. In general, programs listed in BEEBUG follow a particular standard, namely:

Lines 10 onwards are REM statements
Lines 100 onwards are the main program
Lines 1000 onwards are the function and procedure definitions, and data.

The problem with this convention is renumbering the program when any changes are made. The RENUMBER command in Basic will only renumber the entire program with a fixed increment between lines. Several utility ROMs offer a partial renumber, which allows only a section of the program to be renumbered, but this still means handling each part separately. However, the program presented here solves all that by renumbering a Basic program in sections with one simple star command.

To use the renumber command, the program listed here should be typed in, saved and run. This will save a machine code routine under the name 'RENUMBER'. Make sure that you don't call the Basic source code 'RENUMBER' as well, because it will then get overwritten by the machine code. If the name RENUMBER clashes with any star commands already in use, change it to an alternative of your choice.

In its simplest form, the renumber command is used by typing *RENUMBER with no parameters. This will renumber the program currently in memory to BEEBUG style, as described earlier. If the command is followed by one numeric parameter, for example *RENUMBER 2000, then the program will be renumbered as before, but instead of procedures, functions and data starting at line 1000, they will start at the given line number.

The most complex form of the command is when two parameters are used, such as *RENUMBER 2000,1000. In this form, the renumbering will be done as before, with procedures etc. starting at the line specified by the first parameter, but this time, each procedure, function or block of data is renumbered separately. The second parameter specifies the gap between blocks. In the example above, the first procedure would start at line 2000, the second at line 3000, the third at 4000 and so on.

The renumber command contains a 'safety' feature to prevent lines getting out of order. If at the start of a block (procedure, function etc.), the current line number is greater than the new line number, then the renumber just carries on without using the new number. For example, with *RENUMBER 1000,1000 if the first procedure finished after line 2000, then the second procedure would be numbered from where the first one finished, rather than going back to line 2000, in which case more than one line would have the same line number.

*RENUMBER automatically deals with line number references in GOTO, GOSUB, RESTORE etc. in exactly the same way as the normal Basic RENUMBER. In other words, any simple line reference can be changed, such as GOTO 1234, but calculated references, for example GOSUB4*A%, cannot be handled. The 'Failed at ...' warning is generated if a reference is made to a non-existent line.

Because the assembled RENUMBER program is over &300 bytes long, it cannot load into the serial buffers at &900, which is the most common place for transient programs such as this. Instead, RENUMBER loads in at &404, which is part of Basic's workspace. This means that the resident integer variables (A%-Z%) are lost when *RENUMBER is used. The reason for starting at &404 and not &400 is so that the

variable @% isn't lost, as this would upset the printing of numbers. Another consideration is the length of the program. The Basic workspace is only 1K (&400) bytes long. Therefore the program must not end past &7FF. As it stands, the *RENUMBER command ends at &709, so there is plenty of room to make any further additions.

If you want the first line to have a number other than 10, or for the main program to start at a line other than 100, then this can be done with a few simple changes to the program. The first line number is set in line 590 (LDA #10), and the first line after the header is determined by lines 700 and 720 (CMP #100 and LDA #100), and both instructions must be changed. The increment between lines is set in line 960 (ADC #10). Any of these values can be changed to a number in the range 0 to 255.

HOW IT WORKS

It is very easy to write a simple renumber routine for BBC Basic, because each line of the program stored in memory starts with the line number in binary. Therefore, to renumber a program all that you need do is go through each line in turn and change the line number stored in memory. Each line also contains a length byte so that you can jump straight to the start of the next line without scanning through all the intervening bytes of code. The end of the program is identified by a line number that is greater than 32767. To renumber in increments of 10, starting at line 10, you merely need to store a value of 10, use that value as the new line number, and then add 10 to it before doing the next line.

The *RENUMBER command differs from a simple renumber as outlined above in three respects, namely:

1) It is a star command, which needs special routines to decode the parameters etc.

2) Lines are renumbered according to a certain standard, rather than with a constant increment.

3) References to line numbers in the program are adjusted as necessary.

Going through the listing line by line:

Lines 240-410 decode the parameters given after the *RENUMBER command.

Lines 420-570 store the line number of each line in memory starting immediately after the program. These numbers are needed to handle line references. If there is not enough space between TOP and HIMEM in which to store the line numbers, a 'RENUMBER space' error is generated.

Lines 580-890 perform the actual renumbering. Each line is examined to see what the next line number should be.

Lines 910-980 move the pointer to the program onto the next line, and add 10 to the current line number.

Lines 1000-1010 set up a pointer to the start of the Basic program in memory.

Lines 1030-1050 add the appropriate value to the line number of the last procedure to get the number for the next procedure. This is used when *RENUMBER is given with two parameters.

Lines 1070-1100 decode line references within a program. Rather than storing line numbers in GOTOs etc. as binary, BBC Basic stores them as a byte containing &8D, followed by the line number in a three byte form. This routine decodes the value to a sixteen bit binary number.

Lines 1120-1230 scan through the program after it has been renumbered, looking for line references. Each time one is found, the next routine is called to alter it.

Lines 1250-1680 are the heart of the line reference handler. The principle used is to decode the line reference into a binary number (see above), and then search through the list of old line numbers made at the start until that number is found. By counting the position of the number in the table, it is possible to work out the new line number by looking through

the program. This new number is then converted back into the special form and put in the program. If the number can't be found in the table then a 'Failed at ..' message is given, followed by the current line number in decimal.

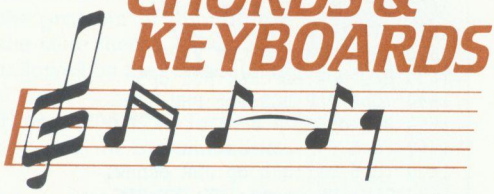
Lines 1700-1720 form a subroutine to skip spaces in the command line.

Lines 1740-1940 read a decimal number from the command line.

```
  10 REM Program Smart Renumber
  20 REM Version B1.0
  30 REM Author  David Spencer
  40 REM BEEBUG  June 1988
  50 REM Program subject to copyright
  60 :
 100 DIM code &400
 110 bassp=6:top=&12:page=&18
 120 base=&70:inc=&72:dflag=&74
 130 ptr=&75:line=&77:datf=&79
 140 ptr2=&7A:lno=&7C:ptr3=&7E
 150 temp=&80:curl=&81:dflag2=&83
 160 ytemp=&84
 170 osargs = &FFDA
 180 osnewl = &FFE7
 190 oswrch = &FFEE
 200 :
 210 FOR pass=0 TO 3 STEP 3
 220 P%=&404:O%=code
 230 [OPT pass+4
 240 LDA #1:LDY #0:STY dflag
 250 LDX #ptr:JSR osargs
 260 LDA #1000 MOD 256:STA base
 270 LDA #1000 DIV 256:STA base+1
 280 LDA #&FF:STA dflag2:JSR sskp
 290 BCC renum:JSR nget:BCC bok
 300 .syntax
 310 BRK:EQUB &DC
 320 EQUS "Syntax: RENUMBER [<start>,[<
increment>]]"
 330 EQUB 0
 340 :
 350 .bok STA base:STX base+1
 360 JSR sskp:CMP #ASC",":BNE bok2
 370 INY:JSR sskp
 380 .bok2 CMP #ASC" ":BCC renum
 390 DEC dflag:JSR nget:BCS syntax
 400 STA inc:STX inc+1:JSR sskp
 410 BCS syntax
 420 .renum:JSR setptr:LDA top
 430 STA ptr2:LDA top+1:STA ptr2+1
 440 .noloop
 450 LDY #0:LDA (ptr),Y:STA (ptr2),Y
 460 PHA:INY:LDA (ptr),Y:STA (ptr2),Y
 470 PLA:BMI main:INY:LDA (ptr),Y
 480 CLC:ADC ptr:STA ptr:BCC noup
 490 INC ptr+1
 500 .noup LDA ptr2:CLC:ADC #2
 510 STA ptr2:BCC noup2:INC ptr2+1
 520 .noup2 LDA ptr2+1:CMP bassp+1
 530 BCC noloop:BEQ chklo
 540 .err BRK:EQUB 0
 550 EQUS "RENUMBER space":EQUB 0
 560 .chklo LDA ptr2:CMP bassp
 570 BCC noloop:BCS err
 580 .main LDA #&FF:STA datf
 590 JSR setptr:LDA #10:STA line
 600 LDA #0:STA line+1
 610 .loop LDY #3
 620 .loop2 LDA (ptr),Y:INY
 630 CMP #ASC" ":BEQ loop2
 640 CMP #&F4:BEQ remok
 650 CMP #ASC":":BNE endofrem
 660 .remok LDY #0:LDA line+1
 670 STA (ptr),Y:INY:LDA line
 680 STA (ptr),Y:INY:LDA (ptr),Y
 690 JSR update:BCC loop:JMP exit
 700 .endofrem LDA line:CMP #100
 710 BCS loop3:LDA line+1:BNE loop3
 720 LDA #100:STA line
 730 .loop3 LDY #3
 740 .loop4 LDA (ptr),Y:INY
 750 CMP #ASC" ":BEQ loop4
 760 CMP #&DC:BCC nodef:CMP #&DE
 770 BCS nodef:CMP #&DD:ROR datf
 780 BIT dflag2:BPL nodef:LDA datf
 790 AND #&C0:BEQ nodef:LDA dflag
 800 STA dflag2
 810 .upnok LDA line:CMP base
 820 LDA line+1:SBC base+1:BCC upok
 830 JSR lineadd:BCC upnok
 840 .upok LDA base:STA line
 850 LDA base+1:STA line+1:JSR lineadd
 860 .nodef LDY #0:LDA line+1
 870 STA (ptr),Y:INY:LDA line
 880 STA (ptr),Y:INY:LDA (ptr),Y
 890 JSR update:BCC loop3:BCS exit
 900 :
 910 .update CLC:ADC ptr:STA ptr
 920 BCC update2:INC ptr+1
 930 .update2 LDY #0:LDA (ptr),Y
 940 BPL update3:SEC:RTS
 950 .update3 LDA line:CLC
 960 ADC #10:STA line:BCC update4
 970 INC line+1
 980 .update4 CLC:RTS
 990 :
1000 .setptr LDA page:STA ptr+1
1010 LDA #1:STA ptr:RTS
1020 :
```

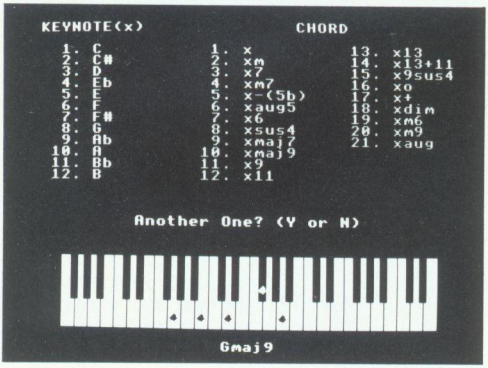
```
1030 .lineadd CLC:LDA base
1040 ADC inc:STA base:LDA base+1
1050 ADC inc+1:STA base+1:RTS
1060 :
1070 .dec LDA (ptr),Y:ASL A:ASL A
1080 TAX:AND #&C0:INY:EOR (ptr),Y
1090 STA lno:INY:TXA:ASL A:ASL A
1100 EOR (ptr),Y:STA lno+1:RTS
1110 :
1120 .exit JSR setptr:.eloop
1130 LDY #0:LDA (ptr),Y:BPL eloop1
1140 RTS
1150 .eloop1 STA curl+1:INY
1160 LDA (ptr),Y:STA curl:INY
1170 .eloop2 INY:LDA (ptr),Y
1180 CMP #13:BEQ exnxt:CMP #&8D
1190 BNE eloop2:JSR change
1200 JMP eloop2
1210 .exnxt SEC:TYA:ADC ptr
1220 STA ptr:BCC eloop:INC ptr+1
1230 BCS eloop
1240 :
1250 .change INY:STY ytemp
1260 JSR dec:LDA top:STA ptr3
1270 LDA top+1:STA ptr3+1:LDA page
1280 STA ptr2+1:LDA #1:STA ptr2
1290 .hunt LDY #0:LDA (ptr2),Y
1300 BMI nof:LDA lno+1:CMP (ptr3),Y
1310 BCC nof:BNE hnext:LDA lno
1320 INY:CMP (ptr3),Y:BCC nof
1330 BNE hnext:LDA (ptr2),Y
1340 LDY ytemp:INY:PHA:AND #&3F
1350 ORA #&40:STA (ptr),Y
1360 STY ytemp+1:LDY #0
1370 LDA (ptr2),Y:LDY ytemp+1
1380 PHA:AND #&3F:ORA #&40
1390 INY:STA (ptr),Y:DEY:DEY
1400 PLA:AND #&C0:LSR A:LSR A
1410 LSR A:LSR A:STA temp:PLA
1420 AND #&C0:LSR A:LSR A
1430 ORA temp:EOR #&54:STA (ptr),Y
1440 INY:INY:RTS
1450 .hnext LDA ptr3:CLC:ADC #2
1460 STA ptr3:BCC hnext2
1470 INC ptr3+1
1480 .hnext2 LDY #2:LDA (ptr2),Y
1490 CLC:ADC ptr2:STA ptr2
1500 BCC hunt:INC ptr2+1
1510 BCS hunt
1520 .nof LDX #9
1530 .nof2 LDA fmess,X:JSR oswrch
1540 DEX:BPL nof2:LDA curl
1550 STA temp:LDA curl+1:STA temp+1
1560 CLC:PHP:LDY #4
1570 .dp LDX #&30
1580 .dp2 SEC:LDA temp:SBC tenlo,Y
1590 PHA:LDA temp+1:SBC tenhi,Y
1600 BCC dp3:STA temp+1:PLA
1610 STA temp:INX:BCS dp2
1620 .dp3 PLA:TXA:CMP #&30
1630 BEQ dp4:PLP:SEC:BCS dp5
1640 .dp4 PLP:BCS dp5:PHP
1650 BCC dp6
1660 .dp5 PHP:JSR oswrch
1670 .dp6 DEY:BPL dp:JSR osnewl
1680 PLP:LDY ytemp:INY:INY:RTS
1690 :
1700 .sskp DEY
1710 .sskp2 INY:LDA (ptr),Y
1720 CMP #ASC" ":BEQ sskp2:RTS
1730 :
1740 .nget LDA #0:STA ptr2+1
1750 LDA (ptr),Y:JSR tenchk
1760 BCC nget2:RTS
1770 .nget2 STA ptr2
1780 .nget3 INY:LDA (ptr),Y
1790 JSR tenchk:BCC nget4:CLC
1800 LDA ptr2:LDX ptr2+1:RTS
1810 .nget4 PHA:LDA ptr2+1:PHA
1820 LDA ptr2:ASL A:ROL ptr2+1
1830 ASL A:ROL ptr2+1:ADC ptr2
1840 STA ptr2:PLA:ADC ptr2+1
1850 ASL ptr2:ROL A:STA ptr2+1
1860 PLA:ADC ptr2:STA ptr2
1870 BCC nget5:INC ptr2+1
1880 .nget5 JMP nget2
1890 :
1900 .tenchk CMP #ASC"9"+1
1910 BCS tenchk1:CMP #ASC"0"
1920 BCC tenchk2:CLC:AND #&F
1930 .tenchk1 RTS
1940 .tenchk2 SEC:RTS
1950 :
1960 .fmess EQUS " ta deliaF"
1970 :
1980 .tenhi
1990 EQUB 1 DIV 256
2000 EQUB 10 DIV 256
2010 EQUB 100 DIV 256
2020 EQUB 1000 DIV 256
2030 EQUB 10000 DIV 256
2040 .tenlo
2050 EQUB 1 MOD 256
2060 EQUB 10 MOD 256
2070 EQUB 100 MOD 256
2080 EQUB 1000 MOD 256
2090 EQUB 10000 MOD 256
2100 :
2110 ]NEXT
2120 OSCLI("SAVE RENUMBER "+STR$~code+"
"+STR$~O%+" 404 404")
```

CHORDS & KEYBOARDS



For the more musically minded Beeb owner Jean Barnard describes a short program for converting chords to keyboard notes.

Much modern music is written in the form of a melody line, together with chord symbols for the guitar or keyboard such as C, A7, Bbm, Ebmaj7, etc. It is often useful to convert chord notation into keyboard sequences, but information on this is not readily available. This program enables the user to find most of the common chords for the keyboard.



Type the program in and save it as usual. When run the program displays a keynote menu and a chord menu, together with a blank keyboard. Suppose you now want to find the Ebmaj7 chord. The keynote (x) is Eb (No.4 on the Keynote Menu) and the chord is a maj7 (xmaj7, No.9 on the Chord Menu). If these values are entered in response to the prompts, the correct keys for playing the chord will be shown on the keyboard, and the notes of the chord will sound.

We have supplied a further program called 'Blanks' on the magazine disc/tape this month. This program prints out blank keyboards which can be used to note down the chords found above. The chord shapes can then be retained and kept with the music for which they are used. Line 150 may need to be altered depending upon the printer dump ROM or program that is to be used. The one used here will work with BEEBUG's Dumpmaster ROM.

```
   10 REM Program  CHORDS
   20 REM Version  B1.2
   30 REM Author   Jean Barnard
   40 REM BEEBUG   June 1988
   50 REM Program  subject to copyright
   60 :
  100 MODE1
  110 ON ERROR GOTO 1210
  120 ENVELOPE1,1,0,0,0,0,0,0,121,-10,-5
,-2,120,120
  130 VDU19,1,6,0,0,0
  140 A=1:D=1
  150 DIM A$(12),B$(24)
  160 DIM C(48),D(48)
  170 FOR n=1 TO 12:READ A$(n):NEXT n
  180 FOR n=2 TO 21:READ B$(n):NEXT n
  190 FOR n=1 TO 48:READ C(n):NEXT n
  200 FOR n=1 TO 48:READ D(n):NEXT n
  210 PROCmenu
  220 PROCkeyboard
  230 PROCselect
  240 PROCmarkandplay
  250 PROCmore
  260 END
  270 :
  280 DEF PROCkeyboard
  290 GCOL0,3
  300 MOVE100,100:MOVE1220,100 :PLOT85,1
220,300
  310 MOVE100,300:PLOT85,100,100
  320 PROCblacknotes(130,210)
  330 PROCblacknotes(290,330)
  340 PROCblacknotes(410,490)
  350 PROCblacknotes(570,610)
  360 PROCblacknotes(690,770)
  370 PROCblacknotes(850,890)
  380 PROCblacknotes(970,1050)
  390 PROCblacknotes(1130,1170)
  400 FOR n=140 TO 1180 STEP 40
  410 MOVEn,300:DRAWn,100
  420 NEXT n
  430 ENDPROC
  440 :
  450 DEF PROCblacknotes(s,f)
  460 FOR n=s TO f STEP 40
  470 GCOL0,0
  480 MOVE n,180:MOVE n+20,180 :PLOT85,n
+20,300
  490 MOVE n,300:PLOT85,n,180
  500 NEXT n
  510 ENDPROC
  520 :
  530 DEF PROCmenu
  540 COLOUR2
```

```
  550 PRINT TAB(1,2);"KEYNOTE(x)"
  560 FOR n=1 TO 12
  570 IF n<10 PRINT TAB(3,n+3);n;". ";A$
(n) ELSE PRINT TAB(2,n+3);n;". ";A$(n)
  580 NEXT n
  590 COLOUR1
  600 PRINT TAB(25,2);"CHORD"
  610 FORn=1 TO 12
  620 IF n<10 PRINT TAB(17,n+3);n;". x";
B$(n) ELSE PRINT TAB(16,n+3);n;". x";B$(
n)
  630 NEXT n
  640 FOR n=13 TO 21
  650 PRINT TAB(30,n-9);n;". x";B$(n)
  660 NEXT n
  670 COLOUR3
  680 ENDPROC
  690 :
  700 DEF PROCselect
  710 PRINT TAB(6,19);"Which Keynote? (E
nter number)"
  720 INPUT k:IF(k<1 OR k>12)PRINTTAB(1,
20);SPC(10):GOTO710
  730 PRINT TAB(18,30);A$(k):PRINT TAB(0
,20);SPC(3)
  740 PRINT TAB(6,19);"Which Chord? (Ent
er number)";SPC(3)
  750 INPUT c:IF(c<1 OR c>21):PRINTTAB(1
,20);SPC(10):GOTO740
  760 IF LEN(A$(k))<2 THEN PRINT TAB(19,
30);B$(c) ELSE PRINT TAB(20,30);B$(c)
  770  PRINT TAB(0,20);SPC(3)
  780  PRINT TAB(6,19);SPC(27)
  790  ENDPROC
  800 :
  810 DEF PROCmarkandplay
  820 IFc=17 OR c=21 THEN c=6
  830 IFc=18 THEN c=16
  840 PROCspot(k+7):SOUND 1,A,49+k*4,D
  850 IF c=2 OR c=4 OR c=16 OR c=19 OR c
=20 PROCspot(k+10):SOUND 1,A,61+k*4,D
  860 IF c=8 OR c=15 OR c=19 OR c=20 GOT
O 880
  870 IF NOT(c=2 OR c=4 OR c=16) PROCspo
t(k+11):SOUND 1,A,65+k*4,D
  880 IF c=8 OR c=15 PROCspot(k+12):SOUN
D 1,A,69+k*4,D
  890 IF c=5 OR c=16 PROCspot(k+13):SOUN
D 1,A,73+k*4,D
  900 IF NOT(c=5 OR c=6 OR c=16) PROCspo
t(k+14):SOUND 1,A,77+k*4,D
  910 IF c=6 PROCspot(k+15):SOUND 1,A,81
+k*4,D
  920 IF c=7 OR c=16 OR c=19 PROCspot(k+
16):SOUND 1,A,85+k*4,D
  930 IF c=3 OR c=4 OR c=11 OR c=12 OR c
=13 OR c=14 OR c=15 OR c=20 PROCspot(k+1
7):SOUND 1,A,89+k*4,D
  940 IF c=9 OR c=10 PROCspot(k+18):SOUN
D 1,A,93+k*4,D
  950 IF c=1 OR c=2 OR c=5 OR c=6 OR c=7
 OR c=8 OR c=19 PROCspot(k+19):SOUND 1,A
,97+k*4,D
  960 IF c=10 OR c=11 OR c=12 OR c=13 OR
 c=14 OR c=15 OR c=20 PROCspot(k+21):SOU
ND 1,A,105+k*4,D
  970 IF c=12 OR c=13 PROCspot(k+24):SOU
ND 1,A,117+k*4,D
  980 IF c=14 PROCspot(k+25):SOUND 1,A,1
21+k*4,D
  990 IF c=13 OR c=14 PROCspot(k+28):SOU
ND 1,A,133+k*4,D
 1000  ENDPROC
 1010 :
 1020 DEF PROCspot(s)
 1030 IF D(s)<200 GCOL0,0 ELSE GCOL0,3
 1040 MOVEC(s)+10,D(s)
 1050 FOR theta=90 TO 360 STEP 90
 1060 x=10*COS(RAD(theta))
 1070 y=10*SIN(RAD(theta))
 1080 x=C(s)+x:y=D(s)+y
 1090 MOVEC(s),D(s):PLOT85,x,y
 1100 NEXT theta
 1110 ENDPROC
 1120 :
 1130 DEF PROCmore
 1140 PRINT TAB(10,19);"Another One? (Y
or N)"
 1150 Z$=GET$
 1160 IF NOT (Z$="Y" OR Z$="y" OR Z$="N"
 OR Z$="n")GOTO1140
 1170 IF(Z$="Y" OR Z$="y") THEN PRINT TA
B(18,30);SPC(9):GOTO 220
 1180 IF(Z$="N" OR Z$="n") ENDPROC
 1190 ENDPROC
 1200 :
 1210 MODE7
 1220 ON ERROR OFF
 1230 REPORT:PRINT" at line ";ERL
 1240 END
 1250 :
 1260 DATA C,C#,D,Eb,E,F,F#,G,Ab,A,Bb,B
 1270 DATA m,7,m7,-(5b),aug5,6,sus4,maj7
 1280 DATA maj9,9,11,13,13+11,9sus4,o
 1290 DATA +,dim,m6,m9,aug
 1300 DATA 120,140,160,180,200,220,240
 1310 DATA 280,300,320,340,360
 1320 DATA 400,420,440,460,480,500,520
 1330 DATA 560,580,600,620,640
 1340 DATA 680,700,720,740,760,780,800
 1350 DATA 840,860,880,900,920
 1360 DATA 960,980,1000,1020,1040,1060,1
080
 1370 DATA 1120,1140,1160,1180,1200
 1380 DATA 130,210,130,210,130,210,130
 1390 DATA 130,210,130,210,130
 1400 DATA 130,210,130,210,130,210,130
 1410 DATA 130,210,130,210,130
 1420 DATA 130,210,130,210,130,210,130
 1430 DATA 130,210,130,210,130
 1440 DATA 130,210,130,210,130,210,130
 1450 DATA 130,210,130,210,130
```

THE COMMS SPOT

Peter Rochford provides a run-down of some of the technical jargon used in Comms.

In an earlier Comms Spot I promised to provide a number of articles that dealt with the 'basics' for those who are new to computer communications. In this first article I have compiled a list of some of the technical terms used, with a short explanation of each.

ASCII (pronounced 'askey')
This is a standard method of representing characters with numeric codes. Most computers, from micros to mainframes, represent characters with the same ASCII codes. ASCII characters are transmitted and received using a 7 bit code so there are 128 possible characters. Some computer systems (including the Master and Compact) provide an extended character set by making use of the top (eighth) bit, but there is no standard for this.

AUTO-ANSWER
This is a facility which allows modems to detect when the line 'rings', and enables them to answer the call. When used with the appropriate software, the whole system can be left unattended, receiving and sending information automatically. This is the basis of most on-line systems like bulletin boards.

AUTO-DIAL
A common facility on many modems nowadays is a facility whereby the modem can dial a number instead of the user having to do this manually using a telephone. Some modems store telephone numbers in their own memory and are instructed by the computer as to which one to use, otherwise the number may be selected by a control panel on the modem.

BAUD RATE
This is the rate at which data is exchanged when two computers are communicating via a modem and telephone line (often quoted in bits per second - bps). There are two signal levels, high and low, which represent the 0 and 1 values of each 'bit'.

BIT
The word BIT stands for Binary Digit and is exactly that. It is the simplest piece of information that a computer can hold (0 or 1). A modem sending data down a telephone line sends high and low tones to represent the two 'bit' states.

BYTE
A byte consists of eight 'bits'. When a byte is transferred inside a computer, it is sent along eight parallel data paths all at the same time using one path for each 'bit'. When a byte of information is sent down a telephone line, the 'bits' are sent in sequence (or serially).

CARRIER SIGNAL
This is generated by the modem and used to 'carry' data (digital) signals down the telephone line. This can be heard audibly when you dial another computer through your telephone.

CONTROL CODE
These are special non-printable characters used by a computer (represented by the ASCII codes 0 to 31), and cause particular functions to occur. For example, when on-line to a scrolling text bulletin board, the received information scrolls continuously up the screen as it is received from the host computer. At high baud rates this may not give you time to read it, but by pressing the Ctrl key and the 'S' key (ASCII 19) on your keyboard together, you send a control code to tell the host computer to halt its output. Pressing Ctrl and the Q key (ASCII 17) will tell the host computer to restart.

DATABASE
With reference to computer communications, this means any on-line information service that can be accessed by a computer and a modem.

DOWNLOAD
This is the facility available on most bulletin boards and Viewdata systems whereby you may receive computer programs or text files. These can then be stored in your computer's memory or saved to a filing system for later use.

ECHO
When you are on-line to a host system, every key you press causes that character to be sent to the host computer, which usually sends back (echoes) the same character to your computer where it is displayed on the screen. This allows you to check that the character has been correctly received.

ELECTRONIC MAIL (EMAIL)
This is used on systems like Telecom Gold and Prestel. A user may send text to another user with access to the same system, and this is stored until the recipient logs-on and elects to read it. He may then store the text on the system for further reference, or erase it.

FULL DUPLEX
This is the term used to describe the transmission of data in two directions simultaneously.

HALF-DUPLEX
This term also describes the transmission of data, but in this case, transmission may not take place simultaneously in both directions.

HOST
This term refers to any computer system connected to a telephone line via a modem that allows access by anyone with a suitable computer and modem, and perhaps the necessary passwords.

LOG ON AND OFF
These terms simply mean to connect or disconnect yourself from a host computer using the necessary commands.

MODEM
British Telecom's telephone lines were originally designed to handle only voice communication over a very limited bandwidth. To send digital information along a telephone line you need a hardware device called a modem. This converts digital signals to analogue signals within the audio band that can be sent along the public telephone system. The modem also converts the incoming analogue signal back to digital. These two processes are called modulation and demodulation. This is the origin of the term modem (i.e. modulator/demodulator).

MODEM SPEED
The following list gives some common modem speeds and their corresponding names.

NAME	SPEED(baud)	DUPLEX
V21	300/300	Full
V23	1200/75	Full
V22	1200/1200	Full
V22bis	2400/2400	Full

PRESTEL
This is a viewdata type online system operated by BT. It is open to anyone who wishes to become a subscriber. There are masses of information to look at on all sorts of subjects. Within the main system are many so-called Closed User Groups (CUGS). These are areas often of specialist interest that need a separate subscription payable to the information provider (IP) before you may gain access. One of the best known of these is Micronet which is devoted to computer users and encompasses BEEBUG's own database.

SCROLLING TEXT
When applied to comms, this refers to those types of online systems, like Telecom Gold for example, that send their information in a continuous stream, line by line, unlike viewdata systems, where the information is sent page by page.

TELECOM GOLD
This is a scrolling text system operated by BT. Again, this is open to anyone who wishes to subscribe, and mainly functions as a very sophisticated EMAIL system, but has many other facilities. Finding your way around the system and getting to grips with the large number of commands can be quite daunting to newcomers. It is rather expensive to use, and its main customers are commercial rather than domestic.

VIEWDATA
These are systems such as Prestel, where the information is sent to the user in the form of pages or frames, along with colour and graphics characters. The screen display looks much the same as you would get on a normal teletext type TV.

BEEBUG WORKSHOP Understanding Recursion

David Spencer takes control of the Workshop, and kicks off with a look at recursion.

One of the more advanced programming techniques is that of recursion, but as we will show in this workshop, the topic is not as difficult as many people think, and if you keep a clear head and employ some lateral thinking it can be very easy to understand.

So what is recursion? To explain this properly we really need to look at what it isn't.

Everybody who is used to programming in BBC Basic, or a language such as C or Pascal, will be familiar with the concept of a main program which calls various functions and procedures, which in turn can call other functions and procedures at a lower level. This leads to a hierarchy of procedures and functions within a program, with the main program calling procedures to do the major tasks, and these in turn calling other procedures to do more specialised tasks. If you have a program that is structured in this way, you can draw a diagram to show how all the separate blocks are related. Such a diagram for a simple spelling checker might look like that shown in figure 1.

Looking at this diagram you will notice that the links between various blocks always go in the same direction. At no point does a procedure link back to a higher level. If this did happen then an endless loop could easily result. For example, if procedure 1 calls procedure 2 at the next level down, and procedure 2 then calls procedure 1 again, you get an endless loop. A similar thing will happen if only one procedure is involved, and simply calls itself. This may appear to be a useless situation, but it is in fact the basis of recursion.

So, recursion is where the definition of a procedure or function calls itself. This can be done directly, which is the most common type of recursion, or indirectly, such as in the above example where two procedures are involved. Figure 2 shows the difference between these two types of recursion.

Using the above definition of recursion, about the simplest recursive procedure that can be written in BBC Basic is of the form:

```
10 DEF PROCrecurse
20 PROCrecurse
30 ENDPROC
```

In this rather pointless procedure definition, whenever the procedure is called it immediately calls itself again, which then calls itself again, and so on. Theoretically this will continue for ever, and the ENDPROC in line 30 will never be reached. However, if you actually type the definition in and start it off with PROCrecurse, then Basic will eventually give a 'No room' error. This is because each time you call a function or procedure, Basic has to store a certain amount of information, such as where the procedure was called from. This information is 'pushed' by Basic in the form of a 'frame' onto a stack which Basic maintains in memory. Each time the procedure is called in line 20, another frame is pushed onto the stack and eventually the stack will become full.

Clearly then, for a recursive procedure to be of any use, it must avoid calling itself too many times. The point at which recursion is terminated depends entirely on what that procedure does. In a simple case, it might be that a procedure will not call itself when the recursion has reached a certain depth (in other

words after the procedure has called itself a certain number of times). An example of a procedure that terminates in this way is given below.

```
10 PRINT "START"
20 A%=0
30 PROCrecurse
40 PRINT "END"
50 END
60 DEF PROCrecurse
70 A%=A%+1:PRINT A%
80 IF A%<500 THEN PROCrecurse
90 ENDPROC
```

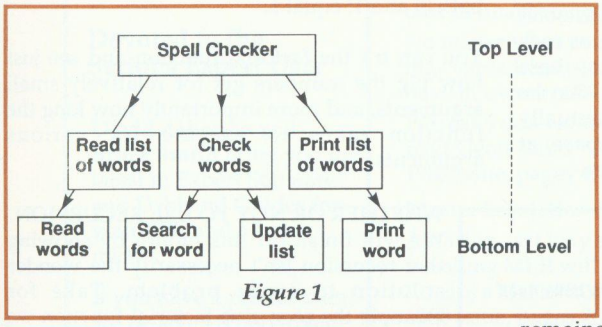


Figure 1

This will print START, followed by the numbers 1,2,3 etc. right up to 500, and will then, after a short pause, print END and exit. It should be fairly easy to see what is happening here. When the procedure is first called, A% is zero, and this is incremented and the new value printed. Then, because A% is less than 500, the procedure calls itself in line 80, and once again A% is incremented and printed. This goes on until A%=500 at which point there will be 500 frames of return information on the stack. What happens next can be a awkward to understand initially. After the condition fails in line 80 the program executes the ENDPROC in line 90. This causes one of the frames to be pulled off the stack, and execution to continue immediately after the procedure call. This means that another ENDPROC will be executed immediately, and the whole process is repeated. This goes on until there is only one frame left on the stack. When that ENDPROC is executed, control returns to the main program, and the END gets printed. To see this 'un-recursing' exit in progress add the line:

```
85 PRINT A%:A%=A%-1
```

Now the numbers cycle up as the recursion gets deeper, and then go down again as it gets shallower.

LOCAL VARIABLES

One of the most troublesome areas, as far as recursion is concerned, is the use of local variables. Normally in BBC Basic, when you assign a value to a particular variable, that variable can be accessed by all parts of the program. This means that if you set up a variable at the highest level of a program, it can still be accessed by any procedures called anywhere in the program. Further, if these procedures change the value of any variables, these changes will be carried back up to the top level. For example:

```
10 A%=1
20 PROCvar
30 PRINT A%:END
40 DEF PROCvar
50 A%=2
60 ENDPROC
```

If you run this program, the value printed out in line 30 will be 2. This is because the procedure changes the value of A% in line 50, and this change remains when you leave the procedure and print out the current value.

To show the principle of a local variable add the following line to the program:

```
45 LOCAL A%
```

and then run it. This time the value 1 is printed out. This is because the LOCAL command tells Basic to create a local version of A% for the duration of that procedure. You can use LOCAL within any procedure or function, and it should be followed by a list of variable names separated by commas. Basic takes all the variables in the list one by one, saves the current value of that variable (if any) on the Basic stack (the one used for the procedure call frames), and then creates a new version of the variable, with the value 0 or "". Therefore, if you change a variable that has been declared as local within a procedure, the changes will not go any higher than that procedure.

Local variables can be particularly important when you write recursive procedures. In the first example, A% is used as a global variable which is incremented each time the procedure

moves down a level. However, in most cases each time a procedure is entered recursively you want it to appear as a self-contained unit. This means that any variables used in the procedure must be declared as local, otherwise the values of those variables at all the higher levels will be corrupted.

Incidentally, any variables used to indicate parameters in a procedure heading, for example 'time' and 'date' in DEF PROCstamp(time,date) are automatically treated as local variables.

You should keep the number of local variables to a minimum in a recursive procedure, as each recursive call leads to further copies of these variables being pushed onto the stack. As the stack descends in memory, it will eventually reach the end of the program's workspace, at which point Basic gives a 'No room' error.

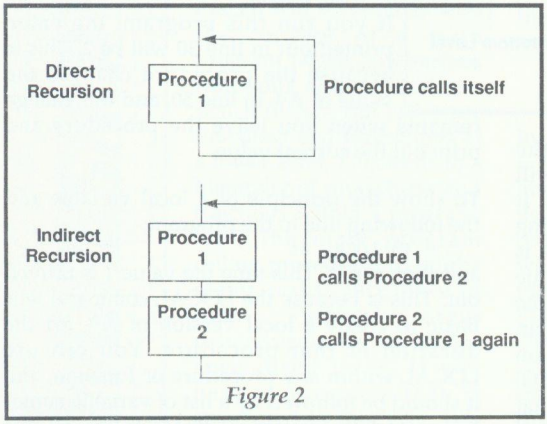


Figure 2

RECURSIVE FUNCTIONS

So far we have only talked about recursively defined procedures, but the technique is equally applicable to functions. A good example where a recursive function can be used in the solution of a problem is in calculating the factorial of a number. The factorial of a number 'n', which is written n!, is defined as the result of multiplying together all the integers from 1 to 'n'. By definition, the value of 0! is 1, and the factorial of negative or fractional numbers is undefined. A very short recursive function to calculate a factorial could be defined as:

```
10 DEF FNfact(n)
20 IF n<2 THEN =1 ELSE =n*FNfact(n-1)
```

This works on the basis that if n! is n*(n-1)*(n-2)*...2*1, then since (n-1)! is (n-1)*(n-2)*....2*1, n! is the same as n*(n-1)!. You can therefore work out the factorial of a number by multiplying that number by the factorial of the number less one. The ELSE section of line 20 does just this, by multiplying 'n' and the result of a recursive call to itself with the argument n-1. To prevent the endless loop situation occurring, line 20 returns the result 1 if the argument to the function is less than 2. This is correct because both 0! and 1! equal 1.

You can try the factorial function and see just how big the numbers get for relatively small arguments, and more importantly how long the function takes to execute for various arguments.

WHETHER OR NOT TO USE RECURSION

We will finish off this month by showing that recursion isn't necessarily the wonder solution to every problem. Take for example the recursive factorial function we have just given. This may be a very elegant looking solution to the problem, but is it the best? Consider the following alternative:

```
10 DEF FNfact(n)
20 LOCAL n2,i
30 IF n<2 THEN =1
40 n2=1
50 FOR i=2 TO n
60 n2=n2*i
70 NEXT
80 =n2
```

This longer function first of all sorts out the special case of n=0 or n=1 in line 30, returning the result 1 if necessary. Otherwise, the value of 'n2' is set to 1 and then multiplied by all the values between 2 and the argument 'n'. This function uses a loop, rather than recursion, but the result is the same.

If you compare the speed between the two factorial functions you might be surprised by the result. The longer loop-based function is faster than the short recursive method. The moral of this is always consider all the alternatives before deciding on a particular technique to use. B

DABS PRESS

Dabhand User News

Devoted to the Serious Expert User

David Atherton and **Bruce Smith** bring you the latest in Expert Software and Dabhand Guides for your BBC and Master micros.

If you have a printer then we believe HyperDriver will be one of the most significant purchases you can ever make.

Here's what **Beebug** said about our products in the March 1988 issue:

HyperDriver: *"...an ingenious blessing.. a million other good design features..."*

MOS Plus: *"..an excellent product."*

Master Emulation ROM: *"...the whole system FEELS just like a Master...the most impressive implementation is an almost complete emulation of the temporary filing system."*

Send or phone for our free 32 page catalogue.

HyperDriver: Printer Power

Possibly the most significant purchase you can make

HyperDriver isn't just another printer ROM - it's the *ultimate* one. It's absurdly easy to use and provides you with so many of the facilities missing from your current printer orientated software including: on-screen preview, CRT graphics, NLQ font, VIEW printer driver and user definable macros to name but a few.

No matter what you use your printer for, word processing, spreadsheets, databases, programming, you will have in excess of 80 * commands instantly available.

We really don't know how to cram all the facts in here, so read Geoff Bains review of HyperDriver in the March 1988 issue, pages 47/48, or ask for our latest catalogue.

Master Emulation ROM

Amazing as it may seem, if you own a BBC B or B+ then installing MER will give it virtually all the software features of a Master 128! See the Beebug review in the March 1988 issue, pages 28/29.

MOS Plus

MOS Plus is a must for all Master 128 owners not least because it fixes those irritating bugs in the OS and adds many essential extras such as *SRLOAD, *FORMAT and *VERIFY in ROM. See Beebug March 1988 pages 44/45.

Prices

Beebug members can obtain their normal 5% discount on these Dabs Press products simply by quoting their membership number (prices in brackets).

HyperDriver: ROM £29.95(£28.52), SWR £24.95 (£23.76)

MOS Plus: ROM £12.95 (£12.33), SWR £7.95 (£7.57)

MER: ROM £19.95 (£19), SWR £14.95 (£14.24)

Orders

Send cheques, POs, official orders to the address below, or quote your Access/Visa card number and expiry date. Credit card orders accepted by phone or mailbox. P&P free in UK. Elsewhere add £2 or £10 airmail. 3.5" ADFS disc £2 extra please state if required.

Dabs Press, 76 Gardner Road,
Prestwich, Manchester, M25 7HU
Phone: 061-773-2413
Prestel: 942876210
BT Gold: 72:MAG11596

DABS PRESS

Personal Ads

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

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

Wordprocessing package for BBC B. Very little used Kaga Taxan KP 810 dot matrix printer with spare ribbon. Epson compatible. Also Wordwise plus and Wordease ROMs. All with manuals. £99 for the lot. No offers, won't split. Tel. (0628) 26076.

Peripherals: Cumuna DS/DD 400K disc drive with own PSU £85. Microvitec Cub colour monitor £150. HFC external expansion ROM box - expand to 28 ROMs £50 or no reasonable offer refused. ROMs: Acornsoft Pascal with stand alone generator £45. Acornsoft LOGO £35. AMX Super Art £35. Peartree Artist £20. InterWord £30. InterSheet £20. InterChart £15. GXR £5. Watford Transferom £5. Watford Dumpout £5. All as good as new with original documentation. Games: Repton 3, White Knight Mk 12, Codename Droid, Bonecruncher, Crazee Rider, on 5" discs, with original documentation £6.50 each. Tel. 01-737 6179.

BBC B 32K, DFS, ATPL ROM/RAM board, Watford 32K RAM board £255. Cumana 40/80T DS disc drive (own PSU) £60. Microvitec 1451 TTL colour monitor £70. Lots more software. Keen prices. Tel 01-643 1186.

AMX Stop Press £20. AMX Extra Extra £10. Solidisk 2 Meg 128 sideways RAM £30. Masterfile II £8. Vucalc Spreadsheet £5. Clares Replica III £5. View Printer Driver Generator £6. Cumuna QFS Disc Interface £10. Tel. (0858) 34032.

BBC B, Issue 4, DNFS, O.S.1.2, Viglen console, five cartridges, dual 40/80T, Microvitec 1431, Interword £425. Tel. (021) 329 2058 (evenings).
Watford ROM Manager £12. Disc Doctor ROM £10. STARdataBASE ROM/Disc £15, ATPL sideways ROM board £15. 6502 Second Processor £60. Watford's NLQ ROM for Epson FX/RX80 £10. Beebugsoft Toolkit ROM £10. All with original packing. Tel. (0233) 32787 (after 5 pm).

ROMs and mint instruction manuals for BBC B micro: Sleuth £12, ROM Manager £8, TD Tape-to-Disc ROM £15, Disc Doctor £10, Dumpout 3 £10, Beebmon £10, Watford DFS £12. Also excellent external ROM expansion plus 32K SW RAM unit with own power supply £65. Tel. (0483) 38285 (eves).

Solidisk free Eprom erasure - boxed, never used £15. Tel. (0454) 613300.

Catalogue your software, store your whole software collection, no memory limits using the program linker system. Program offers full search and print facilities. All data entry is done in Basic £5. Also, programs wanted for future marketing. Tel (0734) 576587 (eves) for negotiations or contact Matthew Reardon, 43 Lesford Road, Coley Park, Reading, Berks, RG1 6DX.

Master 128 including 2 Acorn ROM cartridges. Ref manuals 1 and 2 and View and ViewSheet manuals £310. Acorn 512 second processor inc Gem software, mouse, etc £130. Cumuna CD 800 twin 40/80T disc drives £150. Philips monochrome monitor £50. Acorn ViewStore ROM £25. InterSheet ROM £25. InterChart ROM £15. Acornsoft FORTH disc inc. 2 books £10. View Printer Driver Generator £5. Beyond Basic Assembler course + book and cassette £5. Model B Advanced User Guide £7. More books - ring for details. Masterfile II £5. Beebug Toolkit ROM £7. Disc Doctor ROM £5. Tel. (0883) 842691 (eves).

Lots of games and software on disc and cassette for the BBC B. Tel. (0895) 31163 or write to Mr S J Cooke, 19 Thomson Road, Uxbridge, Middlesex.

Cumana CD 358 twin 3.5" disc drive with formatting disc and cables. Little used and in original packing £135. Viglen Console unit for Master (no packing) £25. Accelerator ROMs plus manuals and packing (fitted in ACP cartridge for Master) works with DFS only £20. Fifty unused Tandy universal discs (5.25") in Rexel lockable box £45. Ten 3M discs (5.25") 80TK DS unused and in Rexel lockable box £10. Tel. 01-572 2917.

Software for Beeb, Master and Electron. Some old, some new, all originals. From 50p to £9. Ring (061) 626 9170 (eves) or send SAE to 16 Oakworth Croft, Moorside, Oldham, OL4 2LE for list.

Toolkit Plus, boxed as new £20. 16K sideways RAM £15. Grand Prix Construction disc £7. Ravenskull disc £5. Castle Quest disc £5. Write to R Hogben, 29 Otley Road, Killinghall, Harrogate, HG3 2DN.

NEW

WORDWISE PLUS II

"The feel of InterWord yet the style of WORDWISE, making a great combination".
BEEBUG Dec 87

Preview in 80 columns
With a long document in memory, there is normally insufficient room to preview it in 80 column mode. WORDWIDE PLUS II allows you to preview your text in 80 columns, irrespective of how full memory is. A menu-driven facility permits several files to be printed out as if they were one long document.

Easy file selection
WORDWISE PLUS II makes memorising filenames, drives and directories a thing of the past. A single key press shows all the files in the current directory. Select one with the cursor key or enter it by name. Works with all properly written DFS's. E.g. Acorn DFS ADFS, Watford 62-file systems. Also Solidisk's "unlimited catalogue entry" DFS.

Multiple documents
WORDWISE PLUS II provides a text area and ten 'segments'. This feature allows up to eleven separate documents to be in memory at once. Just a single key press now enables you to select the text or any segment, from both the menu and edit mode.

Improved search and replace
Search and Replace operations are now even easier to use. Both the search and replace strings are stored for instant editing or re-use. Works directly from edit mode.

Check saved file
Disc drives are not 100% reliable. For peace of mind, WORDWISE PLUS II can check any part of your text against any file. This way you can be sure that your text has been correctly saved.

Easy to use
WORDWISE and WORDWISE PLUS received consistently excellent reviews for convenience and ease of use. WORDWISE PLUS II still retains the familiar WORDWISE PLUS environment, but offers even greater flexibility.

Extra CTRL keys
CTRL keys are the short-cuts in WORDWISE word processing. WORDWISE PLUS II provides over a dozen extra CTRL keys to solve common requirements. E.g. mark word at cursor, mark paragraph, delete current line, GOTO any string. Operations on marked sections of text can now be chosen from a menu.

Supplied on one chip
WORDWISE PLUS II is a complete word processor in its own right. It is 32K long (twice the length of WORDWISE PLUS) but behaves like an ordinary ROM. WORDWISE PLUS II can replace your existing word processing chip.

Compatibility
WORDWISE PLUS II is 100% compatible with existing WORDWISE and WORDWISE PLUS files. It runs WORDWISE PLUS segment programs and works with **Spell Master**. Works with all filing systems (Tape, DFS, ADFS, Network etc.)

Additional embedded command
A new embedded command is present in WORDWISE PLUS II. **PD** (Print Date) means you never have to remember the date again! (This requires a Master or Model B with Solidisk Real Time Clock to operate).

Free utilities
WORDWISE PLUS II is supplied with a disc containing several useful programs. The mail-merge routine lets you prepare a circular letter. A multiple column utility prints your text in up to 5 columns. A label printer is also included.

Printer buffer
Why wait around while your printer churns out a long document? With the printer buffer supplied your computer becomes available for use again in a fraction of the time. (NB This requires sideways RAM to work. We can supply a suitable module — see below).

No limit on document size

Users of WORDWISE on a normal model B computer encounter two main problems. First, you cannot preview the text in 80 columns when memory is nearly full. Second, there simply isn't enough memory to write even quite a short manual, let alone a book. Second Processors can help to solve these difficulties, but they are not cheap.

Our Continuous Processing (CP) utility overcomes both problems, and also offers additional facilities. The CP utility allows documents to be split up into as many files as required. There is virtually no limit on the overall size of a document, since it can span several discs if needed. Search and replace operations can be performed on all or just part of the text, and you have all the benefits of 80 column previewing.

Fast data sorting

Looking for a fast and flexible way of sorting data? Look no further. WORDWISE PLUS 2's sorting routine is very flexible and easy to use. It is also extremely quick, typically 5 times faster than WORD-EX. Our routine operates from sideways RAM, so you can load it as and when required. You don't permanently tie up yet another ROM socket.

Typical time to sort 100 addresses:

WW+2: 0.9 secs **WORD-EX:** 5.0 secs

SPECIAL OFFER!!

For a limited period, we are giving away the continuous processing program and the fast sorting routine absolutely FREE with every copy of WORDWISE PLUS 2.

MONEY-BACK GUARANTEE

If you are in any way dissatisfied with WORDWISE PLUS 2, just return it in good condition for a no-quibble refund! (Applies only when purchased direct from IFEL)

WW+2 is available exclusively from;
IFEL
36, Upland Drive
Derriford
Plymouth PL6 6BD
(07555) 7286

Please allow £1 for postage

WORDWISE PLUS II	£65.95
Upgrade from **WORDWISE PLUS**	£31.95
Spell Master (£50.00 when purchased with WW+2)	£51.00
16K sideways RAM (Requires no soldering)	£15.50

Personal Ads (continued)

Printer Fastext 80 Smith Corona, new, still in box, bargain at £90, not to be missed. BBC Computer Basic with data recorder and lots of software, excellent condition, kept in box £150 or offers. Ideal for a beginner. Tel. Slough (0753) 75141 (eves and weekends).

Solidisk 128K Sideways RAM (old style) and software - RAM disc, printer buffer, page E00 DFS, etc £13.50. 4 colour printer plotter pens, paper, and cable for BBC £19.50. Tel. (04023) 77138.

WANTED donations scrap machines for school, BBC's or even Atoms! Write to Miss Allison, Patchway High School, Hempton Lane, Almondbury, Bristol.

PMS NTQ ROMs and utilities disc complete with documentation, boxed £18. WANTED: InterBase ROM for BBC B. Tel (0332) 556381.

WANTED: Single DS DD 40T or 40/80 drive with or without PSU. Tel (0734) 776836 (eves). Price: £50 - £80.

BBC B Issue 7, 16K s/w RAM Opus Challenger 3, 512K RAM disc, some cassette software inc Elite, some books. Excellent condition £320. Tel. (0934) 514818.

Juki 5510 dot matrix printer, 160 cps draft, 30 cps NLQ, tractor/friction feed £150. View 2.1 and Printer Driver Generator £25. Spellcheck III £20. Sleuth £15. Microman Plotter ROM £15. Oxford Pascal £20. Wordwise Plus £20. Interword £25. Dumpout 3 £15. Watford Electronics Print ROM £10. Gemini DataGem £30. BEEBUG C and Stand Alone Generator £40. Watford Electronics Transferom £10. Acorn Speech Upgrade £20. Computer Concepts Speech £12. Acorn BCPL £30. BCPL Stand Alone Generator £20. BCPL Calculations Package £10. The Music System (80 track) and utilities disc £15. All items in original packing with complete manuals. Tel. (051) 336 3765 (eves).

ViewSpell, boxed with manuals £15. Slidemaster photographic database program on 80 track disc £8. Centronics microprinter P1 £10. Acorn NFS 3.6 £5. Tel. (0705) 594845.

HELP WANTED - instruction for Chalice Scythe. Contact Charles, 47 Clumber Drive, Gomersal, Clecicheaton, BD19 4RP.

Canon Typestar 5r (portable type writer/serial printer) + cable to BBC, 4 new ribbons. Hardly used £110 ono. Four black spare pens for Silver Reed EB 50/JB 10 plotter £1. Wordwise Plus (1.4E), Watford WordAid, BEEBUG Dumpmaster II - above ROMs £12 each. Gemini Office Mate + Office Master, ViewChart Disc, ViewIndex Disc £5 each. Unused parallel printer cable £3. Tel. (0532) 651614.

Enigma Disc Imager ROM BBC B £20. InterWord ROM and all manuals, etc £20. InterBase ROM and all manuals, etc £30. Mini Office II 80T disc for BBC B and B+ £8. Printwise 80T for BBC B or Master £10. Upgrade Mk II ROM (BBC B 8271 only) £10. Advanced Computer Products' Advanced 1770 OFS (Master only) £10. Tel. 01-494 1365 (office hours only).

Points Arising....Points Arising....Points Arising....Points Arising....

BOXED IN THE CARPARK (Vol.7 No.1)

The following lines of data (part of the solution) were omitted from the end of the program:

```
1980 DATA A,137,A,137,C,137,C,137,I,138
,J,138,B,138,D,136,F,136,E,139,E,139,J,1
37,B,138,B,138,D,138,D,138,F,136,E,136,H
,136,G,139,G,139,J,137,B,137,B,139,I,139
,C,136,C,136,A,136,A,136,J,138
1990 DATA B,137,B,137,D,137,D,137,I,139
,A,139,A,136,J,136,J,138,X
```

BEEBDOS-Version 2.01
from MicroBoss

SOFTWARE FOR BBC AND IBM FILE INTERCHANGE

Runs on IBM PC XT AT or compatibles and supports DOS 2.0 or later Supplied on 360k IBM diskette with manual and plastic wallet, the software includes:

BCOPY: Copy files BBC - IBM, IBM - BBC, BBC-BBC (wild cards allowed)
BCONV: Flexible IBM file translate facility enabling text to be passed between VIEW, WORDSTAR and Displaywrite 3 amongst other uses.
BGRAPH: Display BBC mode 0,1,4,5 screen format (AMX Art included) files on IBM standard colour (CGA) , enhanced colour (EGA) or Hercules screens for use in IBM PC presentation, DTP, drawing and other graphic applications.

BeebDOS is almost a complete BBC DFS/ADFS for the IBM PC with utilities to catalogue, format, backup, delete, lock, unlock compact, rename, alter title and options, map free space, change and create directories of BBC diskette/files.

In fact BeebDOS is probably the fastest and most powerful BBC/IBM PC file transfer method available and no cables or serial connections are required.

£46
Inclusive of VAT

BBC double density diskette formats supported are Acorn ADFS, Watford, Solidisk, OPUS DDOS and UDM. Acorn single density DFS is only supported when running BeebDOS on an IBM AT or XT286 with a 360k disk drive . Please send or phone for an information sheet

Cheque or postal order Trade/Education enquiries welcome

MICROBOSS LTD, 3 Hadleigh Road, Frinton, Essex CO13 9HG.
Telephone: (0255) 671095

C & F Associates

Quality Software at Competitive Price

Comp	Cass	Disc	
CDS			
Colossus Chess 4	7.75	12.95	-
DATABASE			
Mini Office 2	12.45	14.95	19.50
Mini Office 2 (Master 80T)	-	17.50	-
French On The Run	7.75	-	-
DESIGN PEOPLE			
Sink The Bismark	-	9.95	-
Tanks!	-	10.95	-
Battle of Britain	7.75	10.95	-
Midway	-	11.95	-
INCENTIVE			
Graphic Adventure Creator	19.95	22.95	-
LARSOFT			
Puppet Man	3.95	-	-
Nine Dancers	3.95	-	-
Wychwood	3.95	-	-
Rising of Salandra	4.95	-	-
Hex	3.95	-	-

Comp	Cass	Disc	
MANDARIN			
Icarus	7.95	10.95	-
Time and Magic (80T)	-	12.95	-
ROBICO			
The Hunt	7.95	10.95	-
Rick Hanson Trilogy	-	19.95	-
Village of Lost Souls	7.95	10.95	-
Beeline	2.95	4.95	-
SUPERIOR			
Barbarian	7.95	10.95	12.95
Spycat	7.95	10.95	12.95
Quest	7.95	10.95	12.95
Repton Thru Time	5.95	6.95	7.75
Play It Again Sam 2	7.95	10.95	12.95
Play It Again Sam	7.75	10.95	12.95
Spellbinder	7.75	10.95	12.95
Elixir	7.75	10.95	12.95
Bone Cruncher	7.75	10.95	12.95
Life of Repton	5.95	6.95	7.75
Palace of Magic	7.75	10.95	12.95
Crazy Rider	7.75	10.95	12.95
Around the World 40 Screens	5.95	6.95	7.75

Comp	Cass	Disc	
SUPERIOR continued			
Grand Prix Construction	7.75	10.95	12.95
Speech	7.50	10.95	12.95
Elite	10.95	12.95	17.50
Revs	10.95	12.95	17.50
TYNESOFT			
Saigon	7.95	12.95	-
Indoor Sports	7.95	11.95	-
Boulder Dash	7.95	11.95	-
Winter Olympiad 88	7.75	12.95	-
Spy Vs Spy	7.75	12.95	-
SPECIAL OFFERS			
ROBICO DISCS ON SPECIAL OFFER ARE 40T ONLY			
Rick Hanson	2.95	5.95	-
Project Thesius	2.95	5.95	-
Myorem	-	5.95	-
Enthar 7	-	8.95	-
Island of Xaan	2.95	-	-
Banjax	1.99	-	-
Ric Hanson Trilogy	10.95	-	-

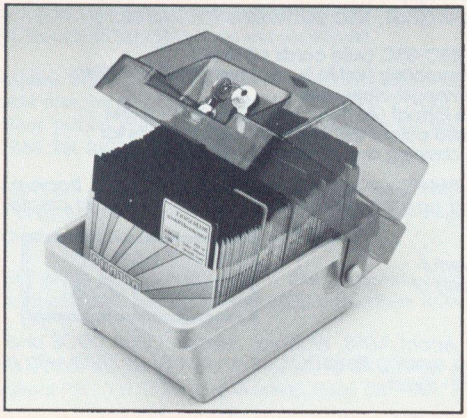
ALL PRICES INCLUDE VAT AND POSTAGE AND PACKING
(Overseas Orders add £1.00 per order or Airmail £2.00 per Item) Cheques/PO Payable to:

C & F ASSOCIATES

and send to: **C & F Associates (BB) 1 Moorend, Wembworthy, Devon EX18 7SE**
Telephone: Exeter (0392) 413418

Orders normally despatched within 24 hours of receipt but allow maximum of 7 days. For our comprehensive range of BBC and Electron Programmes please send for list.

BEEBUG Discs - The Ultimate in Quality & Reliability



50 discs with free lockable storage bock

50 Single Sided 40T Double Density Discs — **Only £35.60**

50 Double Sided 80T Quad Density Discs — **Only £39.90**

Prices shown are members prices and include VAT.

BEEBUG discs are manufactured to the highest specifications and are fully guaranteed.

Our Guarantee

We confidently offer a lifetime data guarantee and will replace any disc with which you encounter problems. We have found that the standard of quality control at the factory makes this necessity very rare.

40 Track Single Sided Double Density

	Price	Members Price	Order Code
10	£9.37	£8.90	0657
25	£23.00	£21.85	0661
50	£37.50	£35.60	0665

80 Track Double Sided Quad Density

	Price	Members Price	Order Code
10	£10.42	£9.90	0660
25	£26.20	£24.90	0664
50	£42.00	£39.90	0668

Please send me _____ (qty) _____ (stock code) at £_____ (unit price)

UK post 10 £1, 25/50 £3.75. Overseas send same price inc. UK post & VAT

I enclose a cheque for £_____ /Please debit my Access/Visa card £_____

Expiry date: ☐☐

Name _____ Memb No. _____

Address _____

_____

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

☎ (0727) 40303

RISC USER

The Archimedes Support Group

Our new Risc User magazine has now had seven successful issues and is enjoying the largest circulation of any magazine devoted to the Archimedes. The list of members seeking support from the Risc User group is growing rapidly and at present we believe that it includes over half of the Archimedes owners.

Existing Beebug members, interested in the new range of Acorn micros, may either transfer their membership to the new magazine or extend their subscription to include both magazines. A joint subscription will enable you to keep completely up-to-date with all innovations and the latest information from Acorn and other suppliers on the complete range of BBC micros. RISC User has a massive amount to offer, particularly at this time while documentation on the Archimedes is still limited.

Here are some of the topics covered in the first seven issues of RISC User:

3D GRAPHICS
A series of basic programs creating and manipulating 3D objects.

RISC USER TOOLBOX
A major new series which is aimed to develop a comprehensive Toolbox module for the Archimedes.

SIGMASHEET
Spreadsheet Review.

A series on **BASIC V**

A WINDOW ON THE ARCHIMEDES
An introduction to the Archimedes WIMP environment for Basic programmers.

INTELLIGENT AUTO-CONFIGURE
A program which reconfigures your machine to suit any particular application.

FANCY FONT SCROLLER
A routine for displaying messages.

ARCHIMEDES VISUALS
Dual Screen Cross Fader
The Mandelbrot Set
And a lot more.

INTRODUCING ARM ASSEMBLER
A series dedicated to ARM machine code.

GETTING THE ARCHIMEDES ON-LINE
A survey of comms packages.

OFF-THE-SHELF
A look at some of the products available now for the Archimedes.

ARCHIMEDES INTO PC
A look at the PC emulator.

SOUNDSYNTH
Review of the first of a series of music modules.

SCREEN MANAGER
Real-time screen and colour manager.

Don't delay - Phone your instructions now on (0727) 40303

As a member of BEEBUG you may extend your subscription to include RISC User for only £8.50 (overseas see below).

Name:.....
Memb No:.....
Address: .....
.....
.....
.....

SUBSCRIPTION DETAILS

Destination	Additional Cost
UK,BFPO &Ch Is	£ 8.50
Rest of Europe and Eire	£13.00
Middle East	£15.00
Americas and Africa	£17.00
Elsewhere	£19.00

I wish to receive both BEEBUG and RISC User. *I enclose a cheque for £ ..... or alternatively*

I authorise you to debit myACCESS/Visa/Connect account: _____/ _____/ _____/ _____/

Signed: ..... *Expiry Date:* ____/____/

Send to: RISC User, Dolphin Place, Holywell Hill, St Albans, Herts AL1 1EX, or telephone (0727) 40303

A full implementation producing the most compact code in the fastest compilation time, for the BBC Micro and Master series.

"...the Beebug implementation is superior in so many respects..."
"The Beebug C system is far superior to that provided by Acorn"
"...the Beebug version has to be the best buy"
A&B COMPUTING FEB 88

"...Beebug C is about five times faster at compilation and linking than the Acornsoft version and produces code at a fraction of the length."
MICRO USER NOV 87

The Language C is now available from Beebug for all users of the BBC Micro and Master. Beebug C conforms to, and extends beyond the Kernighan and Ritchie standard, producing fast, compact code and supporting full floating point maths. A comprehensive set of library functions are supplied on disc, conforming to the proposed ISO standard.

Features Include:

- Runs on a standard 32K BBC model B, B+ or Master 128
- 40/80Track DFS and ADFS compatible
- Support for Acorn operating system (vdu, osbyte, mode etc.)
- Powerful command line interpreter with over 20 commands & qualifiers
- Expandable run-time library on disc containing nearly 100 functions & macros
- Full macro-handling facilities
- Debugging facilities and helpful error messages

Technical Summary

Beebug C is a full implementation of the Kernighan & Ritchie standard. The following is a summary of the full specification.

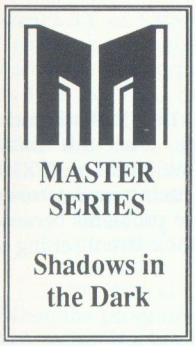
Expressions:	*, &, -, !, ~, ++, --, sizeof, ->, *, /, %, +, -, >>, <<, <, >, <=, >=, &, ^, |, &&, ||, ?:
Assignments:	=, +=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=
Declarations:	char (8 bits), int (16 bits), short (8 bits), long (32 bits), float (32 bits), double (32 bits), unsigned, void, pointer, auto, static, extern, typedef
Statements:	if, while, do, for, switch, case, default, break, continue, return, goto, struct, union
Preprocessors:	#define, #undef, #redef, #include, #if, #ifdef, #ifndef, #else, #endif, #line, #pragma
Library:	nearly 100 library functions, plus a full range of header files - h.stdio, h.stdlib, h.string, h.ctype etc.

BEEBUG C is supplied on two 16K ROMs & a disc. ***(Specify 40/80T)***

Members Price
£44.25

Non-Members Price
£59.00

To write C programs you will need a text editor or word-processor such as View, Wordwise, InterWord etc. Beebug C is supplied with a detailed user guide, however this does not teach C, and a basic knowledge of the C programming language is assumed. The definitive book on C is **The C Programming Language** by Kernighan & Ritchie available from BEEBUG.



David Spencer throws some light on the shadow memory of the Master.

One of the most common enquires we receive at BEEBUG about the Master 128 and Compact is "How do I use shadow memory?", or "What do all these shadow memory FX calls do?". This article will explain how shadow memory is controlled from the hardware level, up to the highest level of software the operating system provides.

Firstly, a brief history. In the original model B, and indeed the Electron, the screen memory was always taken out of the main memory used for the user's programs and data. When some modes use up to 20K, this left very little space out of the 32K available for programs. Acorn recognised this problem and introduced shadow RAM on the model B+. When enabled, the B+ shadow RAM is used for screen memory, leaving the user with memory up to &8000 regardless of the screen mode in use. While the contents of shadow memory can be displayed, and written to by the VDU code in the operating system, it is not possible to access it directly by poking to a particular address. In order to maintain compatibility, the B+ also supports the old non-shadow system to maintain compatibility with the model B.

On the Master series, shadow RAM is present just as in the B+, and can be used in much the same way, but there are also many new commands to provide more flexible access. It is the use of these commands that have caused a certain amount of confusion.

*SHADOW AND MODE CHANGES

At the highest level, the use of shadow memory is controlled by the Basic statement MODE, and the operating system command *SHADOW. After a hard reset, MODE n will select a non-shadow mode if n is in the range 0-7, or a shadow mode if n is in the range 128-135. In other words, to get a shadow mode just add 128 to the mode number. Unless you directly read or write screen memory, the only difference between the shadow and non-shadow modes is that HIMEM will be &8000 in a shadow mode regardless of mode, whereas in non-shadow modes HIMEM will be at a value dependent of the mode in use. This system means that a piece of software written for a computer without shadow memory will select a non-shadow mode when it changes mode.

The *SHADOW command can be used to force modes 0 to 7 to use shadow memory as well. After issuing *SHADOW or *SHADOW 0, modes 0 to 7 become shadow modes, and it is impossible to enter a non-shadow mode. *SHADOW n, with n anywhere in the range 1 to 255, restores the default state described above.

There is also an OSBYTE call that is equivalent to *SHADOW. This is OSBYTE 114 (&72) and should be called with the X register containing the *SHADOW argument, and the Y register containing 0. So for example:

```
A%=114:X%=1:Y%=0:CALL&FFF4
```

is the same as *SHADOW 1. Of course you can also use *FX to call the routine, so *FX 114,1 is yet another form of *SHADOW 1.

Finally on the *SHADOW command, there is another OSBYTE call to *read* the current *SHADOW state. This is OSBYTE 239 (&EF), and should be called with the X register set to zero, and Y set to 255, the result being returned in X. So for example:

```
A%=239:X%=0:Y%=255:shad=(USR &FFF4
AND &FF00) DIV 256
```

would set the variable 'shad' to the current *SHADOW setting, (that is, the value given in the last *SHADOW command).

NEW COMMANDS IN THE MASTER SERIES

What we have described so far is equally applicable to the Master series and the B+, but as stated earlier, the Master offers many more controls over shadow RAM, and it is these which will now be described. The first thing to note before continuing is that there are three different forms of access to the memory that contains the screen image. The first is when the contents of screen memory are actually displayed, at which point the video controller chip is accessing the memory. The second form

of access is when the VDU driver software in the operating system accesses the screen image in memory, for example when a character is printed on the screen. The final form of access is when a program other than the VDU driver accesses the memory used for the screen, for example if a Basic program uses ? and ! to poke to the screen. The hardware inside the computer can detect the difference between all three types of access. It is well worth making sure that you understand the difference between the three types of screen access, because this is the key to doing clever things with shadow memory.

Each form of screen access has its own set of OSBYTE calls to control whether main memory or shadow memory will be accessed. Normally the operating system chooses which bank of memory should be accessed, according to the mode in use, but by changing this, special effects such as smooth animation can be achieved.

PROGRAM ACCESS

The memory bank accessed by any program other than the VDU driver software is controlled by a single OSBYTE call, OSBYTE 108 (&6C). Executing *FX108,0 (or just *FX108) will page main memory into the area &3000-&7FFF, while *FX108,1 will page shadow memory into that area. Normally the operating system always keeps main memory paged in regardless of the mode. This means that in a non-shadow mode you can access the screen directly, and in a shadow mode you can't.

However, by using *FX108,1 you can directly access the shadow memory. To demonstrate this go into mode 128 and type !&5000=-1. Nothing appears on the screen, because the ! writes to main memory, but the computer is displaying shadow RAM. If you now type *FX108,1 and then !&5000=-1 you will see a blob appear on the screen, because the ! now writes to shadow memory. One obvious point to bear in mind is that as soon as *FX108,1 is issued, the main memory from &3000-&7FFF is paged out. If your program happens to be running in this area then it will appear to vanish, and probably cause the machine to crash. To show this, type the following.

```
PAGE=&3000
NEW
10 PRINT "HELLO"
20 *FX 108,1
30 PRINT "GOODBYE"
RUN
```

This will print HELLO, but then crash because the program disappears and the Basic interpreter can't find it. A final point on *FX108 is that the current value is maintained across a mode change. This can cause problems because there is no way of reading the current setting of this FX call.

VDU DRIVER ACCESS

The VDU driver is the piece of software stored in the operating system ROM between addresses &C000 and &DFFF. A clever piece of hardware inside one of the custom chips (the memory controller) monitors the address of each instruction as it is executed, and uses this to tell the difference between VDU driver access to the screen and program access.

The bank of memory accessed by the VDU driver is determined by *FX112. If you issue *FX112,0 (or just *FX112), the bank appropriate to the current mode is selected. In other words, if the computer is in a shadow mode the VDU driver will access shadow memory, otherwise it will access main memory. *FX112,1 will force main memory to be accessed by the VDU driver regardless of the mode, while *FX112,2 will force access to shadow memory. For example, try the following:

```
10 MODE 128
20 PRINT "LINE 20"
30 *FX 112,1
40 PRINT "LINE 40"
50 *FX 112,2
60 PRINT "LINE 60"
```

This will print the 'LINE 20' and 'LINE 60', but produce a blank gap where the 'LINE 40' should be. The reason is that 'LINE 40' is written to main memory, while shadow memory is being displayed. You get a blank line because the cursor movement isn't affected by any of the shadow commands.

The current *FX112 setting can be read using OSBYTE 250 (&FA). The line:

```
A%=250:X%=0:Y%=255:
bank=(USR &FFF4 AND &FF00) DIV 256
```

will return the current setting in the variable 'bank'.

DISPLAY ACCESS

The bank of memory accessed by the VDU hardware is determined in much the same way as that for VDU driver access, except the call used is *FX113, rather than *FX112. This call takes the same arguments as *FX112. As an example, add the following lines to the previous program:

```
 70 A=GET
 80 *FX 113,1
 90 A=GET
100 *FX 113,2
```

When the program is run, the display will be as before, but if a key is then pressed the 'LINE 40' message will be displayed, possibly surrounded by garbage, and a further key press will revert to the previous display. The reason for this is that the program switches to displaying main memory, thus showing the text that was written to main memory, and then switches back to displaying shadow RAM again.

As with *FX112, the current *FX113 state can be read. The call to do this is OSBYTE 251 (&FB). To use the line given above to read the displayed bank simply change the A%=250 to A%=251.

ANIMATION

By using *FX112 and *FX113 it is possible to produce smooth animation. The principle is to alternate the banks of memory accessed by the display and the VDU driver, so that the program can be drawing to one bank while the other is displayed. When the drawing is finished, the two banks are 'flipped over', so that the new image is displayed and the old image can be rubbed out and replaced by the next one in the sequence. By using this technique the movement appears smooth because the redraws are not visible. The listing oposite is a program that rotates a square on the screen using bank switching to get a smooth effect. To see the program without bank switching remove all the lines containing *FX112 and *FX113.

HARDWARE CONTROL

Finally, we will take a brief look at how the hardware is programmed to allow for the various accessing arrangements. It should never be necessary to access the hardware directly though, because the FX calls are suitably fast.

The three different forms of access to screen memory are controlled by the bottom 3 bits of the register at address &FE34. The three bits correspond to the three forms of access in the following way:

```
Bit 0 Display access
Bit 1 VDU driver access
Bit 2 Program access
```

In each case, if the bit is zero, main memory is accessed, and if it is set shadow memory is accessed. The top five bits of the register must be preserved, so to change any of the lower bits the register must be read and masked off. For example,

```
?&FE34=(?&FE34 AND &FE)
```

will reset bit 0, while

```
?&FE34=(?&FE34 AND &FB) OR 4
```

will set bit 2.

```
  10 REM Program SQUARE
  20 REM Version B1.0
  30 REM Author  David Spencer
  40 REM BEEBUG  June 1988
  50 REM Program subject to copyright
  60 :
 100 MODE128:MODE0
 110 ON ERROR GOTO 270
 120 VDU29,640;512;:DIM pts(3,1)
 130 FOR F%=0 TO 3
 140 READ pts(F%,0),pts(F%,1)
 150 NEXT:PROCdraw
 160 sin=SIN(RAD 1):cos=COS(RAD 1)
 170 bank=1:REPEAT
 180 FOR F%=0 TO 3
 190 A%=112:X%=bank+1:Y%=0:CALL&FFF4
 200 x=pts(F%,0)*cos-pts(F%,1)*sin
 210 y=pts(F%,0)*sin+pts(F%,1)*cos
 220 pts(F%,0)=x:pts(F%,1)=y
 230 NEXT:PROCdraw
 240 A%=113:X%=bank+1:Y%=0:CALL&FFF4
 250 bank=(NOT bank) AND 1
 260 UNTIL FALSE
 270 *FX 113
 280 *FX 112
 290 END
 300 :
1000 DEF PROCdraw
1010 CLS
1020 MOVE pts(0,0),pts(0,1)
1030 FOR F%=0 TO 3
1040 G%=F%+1:IF G%=4 G%=0
1050 DRAW pts(G%,0),pts(G%,1)
1060 NEXT
1070 ENDPROC
1080 :
1090 DATA-200,-200, 200,-200
1100 DATA 200, 200,-200, 200
```

BASIC II TO BASIC I CONVERSION

By Lance Allison

In the past we have always endeavoured to make programs compatible with Basic I as well as Basic II. If it has not proved possible to adapt the program to work under Basic I, then this has been reflected in the compatibility icons above the article.

In future we shall be relaxing our policy a little as Basic II ROMs are now readily available, and we will no longer take steps in every case to ensure a program's compatibility with Basic I. The purpose of this short article is to present a set of short procedures and functions which may be used in the future to convert programs to run under Basic I when necessary.

PRINCIPAL DIFFERENCES

Besides the fact that Basic II fixed many of the bugs in Basic I, there were a number of extra features included in Basic II. These additions are the principal source of incompatibility between the two versions.

One such is the OSCLI command. This provides a simple means for a program to issue star commands in a flexible way - for example, to allow a *SAVE command to refer to a filename held in a variable. Thus:

```
OSCLI("SAVE "+F$+" 3000 7FFF")
```

would save a mode 0, 1 or 2 screen with the filename stored in the variable F$. This cannot be achieved with a star command directly. In Basic I, OSCLI may be simply replaced by PROCoscli, but you must append the corresponding procedure definition (as listed) at the end of your program. If you have a program that uses this command simply add the procedure PROCoscli to the end, insert the statement DIM buf%(50) somewhere near the beginning, and replace the OSCLI command with a PROCoscli call.

Another major difference is in the Basic assembler. Four new commands (known as directives) are included in Basic II. These allow text and numbers to be easily inserted into assembly language programs. If you use Basic I, simply add the appropriate function to the end of the program and replace the EQU command with the equivalent function. For instance, if you have a line:

```
100 EQUS "This is a demonstration"
```

it would be replaced with:

```
100 OPT FNEQUS "This is a demonstration"
```

The variable PASS% must be used as the pass variable in the assembly language loop for this to work. Note also, the equally essential use of OPT.

Where file handling is concerned Basic II has a new command OPENUP. This command opens a file for both reading and writing. In general, replace this with the OPENIN command in Basic I, as these commands have the same token in the two Basics.

A good many programs written to run under Basic II incorporate semi-colons in INPUT statements. Basic I does not allow this, but the problem is easily overcome by replacing the semi-colon with a comma.

There are some other differences (such as split mode assembly), but these tend to be quite complicated and difficult to implement on Basic I. In any case, the routines here should allow you to adapt the majority of programs for Basic I quite successfully.

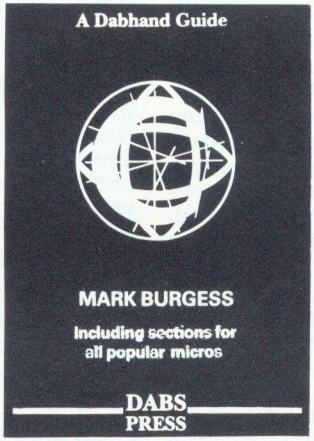
```
 5000 DEF FNEQUB(byte1):LOCAL byte,I$:by
te=byte1
 5010 ?P%=byte:IFPASS%<>3 P%=P%+1:=PASS%
 5020 IF P%<&1000 PRINT;"0";~P%; ELSE PR
INT;~P%;
 5030 I$="":IF?P%<&10I$="0"
 5040 I$=" "+I$+stng$~?P%:PRINTI$:P%=P%+
1:=PASS%
 5050 :
 5060 DEF FNEQUW(word1):LOCAL word,I$,I1
$:word=word1
 5070 ?P%=word MOD 256:P%?1=word DIV 256
:IF PASS%<>3 P%=P%+2:=PASS%
```

Continued on page 66

BECOME A DABHAND AT C

Dec McSweeney takes a look at the latest Dabhand Guide from Dabs Press, on the C programming language.

Since last year, several different implementations of the C language have become available for the Acorn family of computers, and a good tutorial guide to the language is well overdue. Now, Dabs Press have stepped in to fill this gap.



C - A Dabhand Guide by Mark Burgess, published by Dabs Press at £14.95.

The author states in the introduction, "This is a tutorial. Its aim is to teach C to the beginner, in a way which will not be outgrown as the years go by", a tall order, but one which, I think, is to an extent fulfilled. C is a difficult language to master, the more so for those with little or no programming experience. Many underlying concepts are difficult to grasp, and much must be learned before "useful" programs can be written. The standard work on the subject, *The C programming language* by Kernighan and Ritchie (reviewed in BEEBUG Vol.6 No.9) contains a much-praised tutorial section, but the authors are writing for computer professionals rather than the hobbyist, and so even the tutorial is hard going for beginners.

C - A Dabhand Guide attempts to bridge the gap, assuming almost no knowledge of computers. Having said this we are thankfully not treated to yet another first chapter called "What is a computer?" but all the basic concepts are fully explained. The introductory chapters move somewhat slowly for those with any exposure to languages other than Basic, but a cursory read at least is recommended, as they do contain information valuable to everybody.

Topics covered in the first few chapters include: structured programming; the use of C program headers (stdio.h and so on); variable types and declarations; and the C pre-processor. Each chapter ends with a few review questions, to allow the reader to check his command of the subject matter. The author also includes hints on detecting and avoiding bugs, which can be hard to spot in a C program. These appear throughout the text, and there is a separate chapter on errors and debugging.

Once the basic building blocks have been introduced, example programs are listed to illustrate the points covered. For those without access to a C compiler the results of many programs are also listed - a useful feature.

The book is divided into thirty-seven easily-digested chapters, the last five dealing with hardware-specific considerations for each of the following machines: Amiga, Atari ST, Archimedes, IBM-style PCs and the BBC model B and Master (this last chapter dealing with both BEEBUG and Acornsoft C). These sections contain the sort of information that you will find in the appropriate manual, and I feel this makes them rather superfluous.

The appendices cover variations between implementations (the original Kernighan & Ritchie language has been extended for the proposed ANSI standard, and there is now a C++!); an ASCII character conversion chart; answers to the review questions and further notes on programming style - the book being full of good advice about program design and layout.

Two sizeable application programs are included - a statistical data handler and a variable cross-referencer. There is also a version of the famous "Game of life" and many "toolkit" routines which, as well as being useful in their own right, provide good examples of C

Continued to page 66

This month Mike Williams and David Spencer explain how to amend and delete records in serial files, and discuss some of the disadvantages of this appealingly simple system.

AMENDING AND DELETING RECORDS

Last month in our new series on file handling we presented three straightforward programs for creating, displaying and updating simple serial files. We will first of all complete our set of programs by providing the means of both deleting records and amending records in our sample file.

Both of these tasks present problems. If we can identify a way of removing a record from the existing file we will be left with a 'hole' which will surely cause problems later. The obvious solution somehow to close up the remaining records so that the hole disappears. The problem with amending an existing record is that the revised version is very unlikely to occupy exactly the same space as the original, and will either be shorter, again creating a small gap in the file, or it will be too long to fit.

The way to handle both of these situations consistent with our existing programs is to copy the records one by one from the original file to a new file, deleting the original file at the end of this process and renaming the new file with the name of the original. A record to be deleted simply doesn't get copied across. Amended records just take as much (or as little space) as they need. With some additional refinements this is basically the method which we propose to use. Do note though, that with this method the longer the file the longer any updating will take.

However, that is unfortunately not the end of our troubles. We have to identify some means by which the user of our program can specify which records are to be deleted or amended. Whatever we do, we have to read each record in turn from the original file, so the simplest solution is to display each record in turn on the screen and ask if it is to be deleted, amended or to stay the same. You may feel that this is time consuming, and to some extent it is, but the kind of file that you are likely to use with these programs, like our birthdays example, is unlikely to be that long. And if it is so much longer then we would not recommend this method anyway.

Before presenting the program to do the job, we will discuss this matter a little further. One way of speeding up processing would be to specify the name of the person whose record is to be amended or deleted. The disadvantage of this is that once you have gone past a record in the file, in the course of making one file amendment or deletion, you cannot go backwards to locate an earlier record. You would just have to continue to the end of the file to start again. Further, any mistake in specifying the person's name would have the same consequences.

Here now is the complete program to amend or delete records in our 'DATES' file.

```
Program: Amend
  100 MODE 3
  110 ON ERROR GOTO 390
  120 Escape$=CHR$(27):Exit%=FALSE
  130 A$="ACD"+Escape$:*FX229,1
  140 PRINTTAB(10,1)"Amending DATES file
"
  150 VDU28,0,24,79,3
  160 F1=OPENIN("OLDATES")
  170 IF F1>0 THEN CLOSE#F1:*DELETE OLDA
TES
  180 *RENAME DATES OLDATES
  190 F1=OPENIN("OLDATES")
  200 F2=OPENOUT("DATES")
  210 REPEAT:CLS
  220 INPUT#F1,Name$,Address$,Phone$,Bda
te$
  230 PRINT'"Name: ";Name$
  240 PRINT"Address: ";Address$
  250 PRINT"Phone number: ";Phone$
  260 PRINT"Birthday: ";Bdate$
  270 PRINT''"Amend, Copy or Delete - Es
cape to finish"
```

```
  280 REPEAT:G$=CHR$(GET AND &DF):UNTIL
INSTR(A$,G$)>0
  290 IF G$="A" THEN PROCamend
  300 IF G$<>"D" THEN PRINT#F2,Name$,Add
ress$,Phone$,Bdate$
  310 IF G$=Escape$ THEN PROCcopy:Exit%=
TRUE
  320 UNTIL EOF#F1 OR Exit%
  330 PRINT'"File amended OK"
  340 CLOSE#0:VDU26,31,0,20:*FX229
  350 *DELETE OLDATES
  360 END
  370 :
  380 MODE 3:REPORT:PRINT" at line ";ERL
  390 CLOSE#0:VDU26,31,0,20:*FX229
  400 END
  410 :
 1000 DEF PROCamend
 1010 INPUTLINE'"Name: " Name$
 1020 INPUTLINE"Address: " Address$
 1030 INPUTLINE"Phone number: " Phone$
 1040 INPUTLINE"Birthday: " Bdate$
 1050 ENDPROC
 1070 :
 1080 DEF PROCcopy
 1090 REPEAT
 1100 INPUT#F1,Name$,Address$,Phone$,Bda
te$
 1110 PRINT#F2,Name$,Address$,Phone$,Bda
te$
 1120 UNTIL EOF#F1
 1130 ENDPROC
```

THE MAIN FEATURES DESCRIBED

There are quite a number of points that need to be discussed in relation to this program. Let's start by looking at the main essentials, and leave the finer details for later. We start by renaming the existing file (it's given the name OLDATES at line 180). It is easier to do this at the outset than it is to create a new file with a different name, and then at the finish get rid of the original file and change the name of the new file to that of the original.

The program then proceeds to open the renamed (original) file for input, and opens a new file with the same name as the original for output (lines 190, 200). The main loop of the program then runs from line 210 down to line 320. To start with, the program reads in the data for one record (line 220) and displays this on the screen in the same way as last month's Display program.

This time the program offers three choices; to amend, copy or delete the record displayed (there is a fourth option to which we will return shortly). Line 280 contains a short REPEAT-UNTIL loop which cycles until an 'A', 'C' or 'D' is input (the AND &DF ensures that the result is in upper case regardless of the case of the letter entered). If the user selects amend then a new record is prompted for which replaces the original (the coding is very similar here to that used for record input last month). Provided that the option selected was not 'D' (for Delete) the current record in memory is then written out to the new file being created.

Once all the records in the original file have been processed in this way, a confirming message is displayed, the original (renamed) file is deleted and the program terminates. Try it out in conjunction with last month's programs to prove this to yourself.

Although the program leaves us in the state in which we started, with the one file called DATES, it would probably be safer to omit line 350 so that the original version of the data file remains as a backup. It is always wise to be cautious with data files, and you never know, you might just delete one or more records by mistake. If you have a backup, it is an easy matter to discard the newly created file (using *DELETE) and rename the backup with the proper file name.

REFINEMENTS

Now we will deal with some of the lesser points. If the program is going to rename a file (as at line 180) this will fail with a standard error message if a file already exists with the name that you are trying to use (or if a file is locked). Clearly the program must ensure that any such file is deleted (or renamed) beforehand. To check whether a file exists, simply open the file for input and check the resulting channel number (see lines 160, 170). If the channel number is non-zero then a file of that name already exists. This file is then closed before being deleted. If you try to delete the file without closing it first then another error message would be generated.

To be sure of catering for the possibility of a locked file, a program would need always to

execute a *ACCESS command first to ensure that the file was unlocked before any further action was taken with that file.

USING ESCAPE

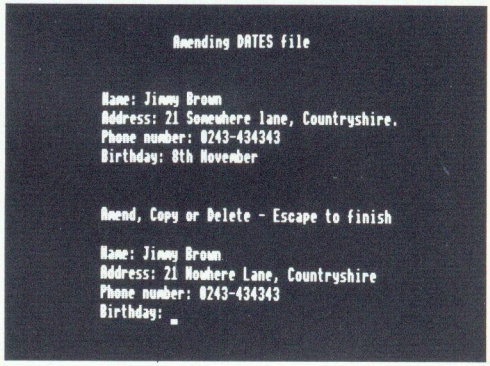
The other refinement which has been included in the program is a means of terminating the program before we have examined every single record in order to speed up processing. The most obvious way to do this is to use Escape so that when this is pressed, all the remaining records are copied, without further reference to the user, from the input file to the output file. But there is a problem if we attempt to use Escape as normal and trap it with an ON-ERROR statement.

It is well nigh impossible to time the pressing of Escape so that its action occurs at precisely the right point in the execution of the main loop. In practice you would likely find that either a record got copied twice, or not at all, or even part of one record got mixed up with part of the next. However, Escape seems the natural choice in the circumstances (we used Escape in our programs last month where appropriate to terminate execution).

What we have done therefore is to execute *FX229,1 near the start of the program to disable the normal action of the Escape. From then on pressing Escape generates an ASCII code (ASCII 27) just like any other key on the keyboard, instead of interrupting the Basic program and invoking error trapping. The Escape key will now be detected at line 280 when the prompt for user input appears. If Escape is detected, the last record input is still copied to the output file, and the procedure PROCcopy is then called to transfer all the remaining records from the input file to the output file. A flag is also set (Exit%), and this is used to terminate the REPEAT-UNTIL loop in line 320 as an alternative to the check on 'end-of-file'.

One further point remains to be mentioned regarding this program. When any program executes various VDU or FX calls to change the normal status of the computer, it is highly desirable that it should restore the default state on termination, and not only as a result of a normal exit, but also when an error exit is made as well. Hence the coding at lines 340 and 390. Note also the use of CLOSE#0 to ensure that both files being used by the program are properly closed.

We have discussed the above program at some length because we believe that it is important to have a full understanding of all the points involved before we can proceed further. Understanding the functions of the various file handling instructions in Basic is really quite simple; understanding how to use these for efficient and effective file handling is another matter altogether. You should also begin to appreciate that in most file handling programs there is more potential for errors to occur and more loose ends to clear up at the end than is the case with many programs.



THE LIMITATIONS OF THE SIMPLE APPROACH

The three programs presented last month together with this month's listing provide a complete suite for simple serial file handling, and by altering file and field names (and prompts) may suffice for many simple applications. However, there are many inherent limitations, particularly if we wish to deal with much larger files, while the lack of generality is a further and major drawback.

We need to explore some of these constraints and to examine file handling in more detail before we can begin to make further progress. One of the major drawbacks with our file handling so far is that each piece of data is stored in the file at whatever length it happens to be. As a result every record is likely to be different in length to every other record - remember the problem of how to amend an

existing record in situ which suffered from this very problem.

If every record were the same size then a number of benefits would result. Any record could be updated where it was, instead of having to copy the entire file. And we could locate any particular record by simply multiplying the length of a record by the number (or position) of that particular record within the file. Of course, we would also need to have some means of moving the file pointer to a specified position, but that exists in the form of the Basic keyword PTR# of which more anon.

STORING DATA IN FILES

Let's go back to this question of field and record lengths. Data that is stored in a file may be one of three types, integer, real or string. All explicit integers are allocated four bytes, plus an initial *indicator* byte (value &40), making five bytes in all. Reals are allocated five bytes plus an initial indicator byte of &FF making six in all. Lastly strings are stored as a zero byte (&00) followed by a byte count of the number of characters in the string followed by the string itself. Thus strings require two more bytes of storage than the length of the string itself, and the maximum length of a string (stored in this way) is limited to the maximum value which may be stored in a single byte, namely 255.

Applying this approach to the file used in our previous programs, we might design our record structure, using fixed length fields, as follows:

Field Name	Field Type	Data Length	Total Length
Name	string	20	22
Address	string	60	62
Phone	string	12	14
Birthday	string	14	16
Record Length			114

It so happens that in this example all the fields are string fields (it hardly makes sense to treat a phone number as numeric). As you can see, following this fixed field length, fixed record length approach, each record would use exactly 114 bytes. Generalising, we can deduce a simple formula. If the record length is 'r' (in bytes) and the number (position) of a record in the file is 'n', then that record will start in position r*(n-1), since the first record always starts in position zero. We could then use PTR# to move the file pointer to the start of a particular record by writing:

```
PTR#F=r*(n-1)
```

where F is the file handle, and r and n are as defined above.

Of course, the actual data that we enter into such a fixed length file will not fit exactly into the (arbitrary) field lengths specified. Each piece of data will need to be padded out (or justified) so that it does then fit. If you want to check out this process (which is quite common in file handling) then refer to the First Course article in BEEBUG Vol.6 No.10.

All numbers, real or integer, are always allocated the number of bytes stated (5 or 6) and justification is never needed. However, it is often easier to store all data in files in string format, as this gives complete uniformity with regard to the file handling, converting numbers to and from string format as required. Numbers converted to strings will also then need to be justified within their respective fields.

There is one further peculiarity with the way in which strings are stored in files in BBC Basic - the characters are stored in reverse order. Why this should be so is lost in Acorn's dim and distant past, but presumably there was thought to be a good reason for it. Generally speaking you can ignore the fact - just let the system sort it all out for you, but there are instances when knowledge of this is important. For example, the *DUMP command is useful for displaying the contents of a file on screen when debugging file handling programs. If you apply this to the DATES file you will see what we mean by the reversal of character strings. You should also be able to see the zero and character count bytes.

Next month we will combine the ideas on fixed record lengths and the use of PTR# to show how we can achieve direct access to records in a file for more efficient file handling. In addition, if you have any specific topics you would like to see covered in this series of articles then please drop us a line. B

1st COURSE

Making Better Use of Filing Systems

Mike Williams departs from the world of Basic to explain how to make better use of your disc filing system.

Most of the articles published under our First Course banner are concerned with exploring the elements of Basic. This time I propose to deal with something a little different by taking a look at the disc filing systems used on the BBC micro. This month we will be concentrating on the simpler DFS, preceded by a little history.

Disc drives are now so cheap (I recently saw brand new 100K 40 track drives on sale for just £49.95) that there is really no sense in not using a disc filing system, and I have little hesitation in ignoring cassette systems (discs are much easier and far less error prone as well). As for the ROM Filing System (RFS), this is much more specialised and not really a topic which I feel we should be attempting to cover at this level.

ACORN DISC FILING SYSTEMS

There are two quite different disc filing systems used on BBC micros, the DFS (Disc Filing System) and the ADFS (Advanced Disc Filing System). Despite their similarity of name these two systems are quite distinct, and discs formatted for one system cannot be used with the other. The DFS was the only system supplied for use with the original model B. The Master 128 is fitted with both DFS and ADFS, while the Compact is effectively the same as the Master except that the DFS ROM image has to be loaded from the Welcome disc supplied with the machine before it may be used.

The DFS was originally written around the 8271 disc controller chip, while the ADFS uses the more advanced 1770 chip fitted to the Master and Compact. The DFS used by these later machines is a different version to that used on earlier BBC micros as it (like the ADFS) uses the 1770 chip, but the majority of users will notice no difference between the two.

Finally in this tale of Acorn disc filing systems, the 1770 chip can now be fitted as an upgrade to a model B in order that the ADFS may also be used on that machine. Readers contemplating such an upgrade should be aware that this can impose severe limitations on the model B, as the value of PAGE is raised to &1A00 even with the DFS disabled to provide adequate disc work space, and for this reason some programs which work perfectly well on a model B using the DFS may not function at all on the same machine using the ADFS.

ALTERNATIVE DISC FILING SYSTEMS

During the life of the BBC micro other companies have also released disc filing systems. The majority of these (but not all) are based on the Acorn DFS, offering the same set of commands and claiming full compatibility with the Acorn system. To my knowledge, only one such system is still available and to be recommended, and that is the Watford DFS (supplied by Watford Electronics). Some early versions did have problems, but version 1.43 or later should prove error free.

The Watford DFS offers some additional commands, and a special 62 file catalogue system as an optional alternative to the Acorn-compatible 31 file system. This means that up to 62 files may be stored on one surface of a disc instead of the standard (and more limiting) 31 files. Users often find that DFS format discs run out of catalogue space well before the disc itself is full. However, 31 file and 62 file catalogue systems are not interchangeable, so take note.

Watford, as other companies have done in the past, also sells a DDFS (Double-density Disc Filing System). This is another system based on the 1770 chip which offers increased disc

capacity whilst otherwise maintaining the structure of the Acorn DFS. However, discs formatted for use under this system are not interchangeable with those formatted with the standard Acorn DFS.

Despite the claims of the suppliers, there always remains some small doubt over the strict compatibility between alternative DFS systems and the Acorn standard. Thus unless you have some particular need for the additional features of a non-Acorn product, I would strongly recommend you stick with the Acorn standard.

UNDERSTANDING THE ACORN DFS

The surface of each disc has 40 or 80 tracks with each track divided into sections called sectors (10 per track). Each sector in turn stores 256 bytes. The first track on a disc (perversely called track 0) always contains the disc catalogue. This contains space for details of each possible file stored on the disc (maximum 31). The catalogue serves two basic purposes; it tells the user what files are stored on the disc, and it tells the DFS where on the disc each file is located. The catalogue also stores other information about each file as we shall see in due course.

Because track 0 is in precisely the same position on both 40 track and 80 track discs, it is normally possible to read the catalogue of any DFS disc in any drive (though clearly a single-sided drive cannot read both sides of a double-sided disc - you can't just turn it over). However, if you then attempt to load any file listed on a drive which is incompatible with the disc (40 track disc in an 80 track drive for example), the attempt will fail with a message like:

```
Disk fault 18 at
```

followed by a reference to the drive, track, and sector numbers in use at that time. This can be confusing unless you are aware of this possible situation, as the ability to read the catalogue tends to suggest that all is well.

When you catalogue a disc the DFS, most conveniently, lists the files in alphabetical order irrespective of the true order of the files on the disc. The files in the currently selected directory are always listed first, followed by the files in any other directories. Directories on a DFS system are not really true directories in the sense that the word is used on other disc systems. On the whole, a directory letter is best used to indicate the type of file (e.g. B for Basic, V for View, M for machine code etc.), and the whole of the catalogue treated as one big 'directory'.

Generally speaking, DFS commands are not sensitive to the case of letters used in the file name. For example, if you have a file called DATA then this may be deleted by entering:

```
*DELETE DATA
```

or `*DELETE data`

or any other similar variation. On the other hand the command:

```
*RENAME DATA Data
```

will change the name of the file (called DATA, data or whatever) specifically to the name 'Data'.

When the DFS is asked to save a file to disc it checks on the size of the file to be saved, and then searches through the catalogue to locate the first space large enough for that file. At first, the only space available will be at the end of the files already saved. Once the space is identified, the file is transferred to the disc, and the catalogue then updated with the details of the file just saved. As time passes, however, your disc space is likely to become fragmented as files are deleted or updated. In essence, files no longer fit the the spaces they are allocated.

If this persists, then ultimately you are likely to encounter the message 'Disk full' even though you know there is sufficient storage capacity left on the disc for the file in question. The problem is that all this remaining space will have been fragmented throughout the disc surface, and no one 'hole' will be big enough for the file you wish to save.

The answer is to use the *COMPACT command. This will shuffle all the files on your disc without changing their order so that all the free space is collected together after all the files. If any of your discs is particularly active (i.e. you frequently save and delete files) it can be a good idea to compact the disc at the end of each

session. It also makes sense to ensure that you always have at least one spare disc handy with a good deal of free space on it. Why? I'll explain.

*COMPACT makes use of memory to read in sections of the disc and write them back out to a different location. Any program or file in memory is likely to be corrupted, so you cannot use the *COMPACT command to solve a 'disk full' problem without destroying the very file you want to save. That's why you need the spare disc.

DFS UTILITIES

In past issues of BEEBUG we have published a good many useful utilities to help DFS users. One which I would particularly recommend is the Disc Manager (for the model B and B+ only) by Bernard Hill first published in BEEBUG Vol.5 No.4. For convenience a copy of this program has been included on this month's magazine disc (or the original magazine is still available as a back copy for £1.20 plus 40p p & p). This utility provides a range of useful functions including some of the standard DFS commands, but all selectable from a menu. It also allows several files to be flagged for action together, such as deletion, renaming etc. This is a very useful utility to keep on all your main working discs, and certainly proved very popular when it was first published.

But in particular, for this First Course article, the Disc Manager provides a command which will let you see graphically the degree of fragmentation of any DFS disc. This is not only useful as a means of determining whether a disc needs to be compacted or not, but also provides a fascinating picture of how files are spread out on a disc.

DFS MEMORY IMPLICATIONS

The DFS needs memory for use in buffering file transfers, and for use as work space. On the model B (and B+) this is taken from user memory, which is why the value of PAGE goes up from &E00 on a cassette system to &1900 on a DFS system. The Master and Compact are able to make use of private RAM for this purpose, and the value of PAGE remains at &E00 on these machines.

The use of buffering means that when data is to be transferred to disc, it goes initially into a 256 byte area of memory (the buffer). Once this is full the entire contents of the buffer are transferred to disc in one continuous operation. The same process is applied in reverse on loading data from disc to memory. This provides a 'cushion' between the much faster internal operation of the computer and the relatively slower functioning of the disc drive. It also avoids the disc drive switching rapidly on and off if data would otherwise be transferred just a few bytes at a time.

The DFS provides for up to 5 files to be open at the same time, each with its own input and output buffer. If less than five files will be open at the same time then the value of PAGE may be reduced on a model B to gain more memory space. If a program needs no data files of its own then PAGE may be reduced as low as &1200, and still permit programs themselves to be loaded and saved. Running a program with PAGE set too low will usually result in a 'hung' machine. Programs can also be reduced in length by using a *compacting* routine as in many utility ROMs (BEEBUG's Toolkit for example).

```
1 *K.0 *T.|MFORA%=0TO(TOP-PAGE)STEP4:
  A%!&E00=A%!PAGE:NEXT|MPAGE=&E00|MOLD|M
  DELETE1,3|MRUN|M
2 *FX138,0,128
3 END
```

If a program is still too long to run in the available memory there is a problem as the program has to be loaded from disc to start with. However, even this can be overcome by moving the program down in memory after it has been loaded, and you will have seen this referred to in connection with some of the programs published in BEEBUG. If you need to gain the last ounce of space then add the above routine at the start of any program. Once loaded, the program will move itself down in memory before deleting the actual instructions that performed this task. It may sound like magic, but it can sometimes be a life saver. More on disc filing systems next month.

B

ADFS VIEW MENU UPDATE

Compiled by Lance Allison

The ADFS menu for View that we published in BEEBUG Vol.6 No.10 has proved to be very popular indeed. We have received many suggestions and improvements for the program, so we have decided to bring these together on this page.

Firstly there appears to be a shortage of memory when the program is run on a model B with the ADFS fitted. For the menu to run, the default value of PAGE must be no higher than &1D00. To achieve this when both ADFS and DFS are fitted, the DFS must be disabled. This is done either by removing the DFS ROM or disabling it by software (using the BEEBUG Master ROM for example).

The second problem occurs when a new directory is created from within the menu, and the %HEADER file is copied across. This corrupts the memory immediately above the default value of PAGE. To prevent this, the value of PAGE should be raised by &100 before loading the program. This can be done with the line:

```
PAGE=PAGE+&100
```

Provided that the DFS has been disabled if necessary, running the menu with PAGE=&1E00 will work in all cases. This can be done automatically by adding the following line and ensuring that the program is saved with the filename $.VMENU.

```
90 IF PAGE<>&1E00 PAGE=&1E00:CH."$.VMENU"
```

In the original program it is in fact possible to create a new directory without any warning that a file or directory already exists of the same name. The following alteration will prevent the existing file or directory from being overwritten:

```
2680 FOR X=0 TO P%
2701 FOR X=0 TO M-1
2702 IF newname$=DIR$(X) exist=TRUE
2703 NEXT
```

EXTRA INFORMATION

The following enhancement to the menu will automatically read and display the currently selected directory at all times together with the number of available sub-directories.

```
 1305 PROCrd
 1540 PROCwind(2,TRUE):PRINT':PROCh(" CU
RRENT DIRECTORY : "+crd$+" : "+STR$(M-1)
+" Sub Directories "):PROCwind(3,TRUE)
 2030 PROCwind(2,TRUE):PROCh(" CURRENT D
IRECTORY : "+crd$+"    "+STR$(E%)+" file
s "):PRINT:PRINTTAB(3)"Name";SPC(22);"De
scription";SPC(33);"k bytes":PROCwind(3,
TRUE)
 2170 IFE%=0 PRINTAB(3)" ";:PROCcol(1):P
RINT"No files":PROCcol(2):Z%=1:ENDPROC
 3470 DEFPROCrd:crd$="":A%=&6:X%=blk%MOD
256:Y%=blk%DIV256
 3480 ?blk%=0:blk%!1=buf%:CALL &FFD1
 3490 FORY%=3TO 12:crd$=crd$+CHR$(buf%?(
Y%)):NEXT:ENDPROC
```

Because this modification will increase the length of the program considerably, it is essential that all additional spaces and REM statements be removed from the original program if you have an unexpanded model B. If you have Toolkit Plus or a similar utility ROM the program may be compacted. If you have a Master or Compact it may be simpler to lower PAGE to &F00 by changing references from &1E00 to &F00 in line 90. This will still leave enough memory to prevent corruption problems when creating new directories.

RE-LOADING THE VIEW MENU

Regular users of the menu will find the following modification extremely handy. Once the menu has been booted up, and a View file selected for editing, the menu can be re-loaded simply by pressing Crtl-Shift-f8. This will save your current document and then re-enter the View ADFS Menu.

```
2870 PROCoscli("KEY8 SAVE|M*BASIC|M
     *EXEC $.!BOOT|M")
```

With all of these modifications we believe that you have a very powerful and easy to use menu system. A copy of the revised menu program is included on this month's magazine disc/tape. Thank you to those members, including the original author Alan Mothersole, who have supplied these enhancements and alterations. B

A series for beginners
to machine code by Lee Calcraft

Part 11: Stacks and Subroutines

Like most microprocessors, the 6502 is provided with a so-called *stack* which can be used for the temporary storage of addresses and data. If you have written almost any machine code routine for the 6502, you will have made use of the stack; though you may not have been aware of the fact. Every time that a subroutine call is made, using JSR (Jump to SubRoutine), the current position in the program is stored on the stack, and is retrieved when an RTS (ReTurn from Subroutine) is encountered; so enabling the program to continue from where it left off.

If subroutines are nested (i.e. with one calling another), then a whole sequence of return addresses is stored on the stack. The stack is also used whenever the CALL statement is issued from Basic. In this case, the Basic interpreter's current position (which will be an address in ROM) is stacked. Again it is restored after an RTS, thus permitting an orderly return to Basic.

It is in fact possible to store data of any kind on the stack, and the designers of the 6502 have provided a set of instructions with which to do so. We will take a look at just two of these, PHA and PLA, before examining the precise structure of the 6502 stack. The first instruction, PHA, allows the user to *Push* the contents of the accumulator onto the stack. As with the instruction STA, which stores the contents of the accumulator in memory, the state of the accumulator is left unaltered by the process. To retrieve the data, PLA may be used. This *Pulls* the byte from the stack and places it in the accumulator, setting the Z and N flags according to the result. Thus after the sequence:

```
LDA #&FF
PHA
LDA #0
PLA
```

the accumulator will contain the value &FF. This was pushed onto the stack at the start, and pulled from it after the intervening (and pointless) load instruction LDA #0. Now we will take a look at how the stack performs such an operation.

HOW THE STACK WORKS

The 6502 stack is simply a page of RAM at &100 to &1FF set aside by the processor for storing data and addresses. It is a so-called *last in first out* (LIFO) stack in that data is retrieved from the stack in the reverse order from which it was saved. This is the opposite of a data queue which operates on a *first in first out* basis. A *queue* structure is appropriate to such applications as a printer (or other) buffer, where data entering the buffer first *must* be the first data out.

When stacking the return addresses of subroutines it is vital that they are returned on an LIFO basis. Suppose subroutine 1 calls subroutine 2, which in turn calls subroutine 3. When subroutine 3 terminates, the stack must supply the very *latest* stacked address so that control returns to subroutine 2, rather than to some earlier stacked address. This is all arranged using a so-called *stack pointer*. The stack pointer is held in a special register in the 6502's register set, and always points to the next *vacant* position in the stack. When PHA is used to push a byte onto the stack, it is pushed at the address pointed to by the stack pointer, which is then decremented, so as to point once again to the next free location. When a PLA instruction is executed, the stack pointer is first incremented, and then the byte to which it points is copied into the accumulator.

To illustrate this, take the following sequence of code:

```
LDA #&AA:PHA
LDA #&BB:PHA
```

Figure 1(a-c) illustrates the state of the stack during this process. It is assumed that at the start the stack is empty (though in practice it is likely to contain a return address to Basic, at the very least). Since it is empty, the stack pointer contains the value &FF. In fact when the Beeb powers up, the stack pointer is loaded with &FF. The top of the stack is located at &100 + &FF (i.e. the base of the stack plus the value in the pointer).

After the first pair of instructions the stack will appear as in figure 1b. The top location of the stack now contains &AA (the contents of the accumulator), and the pointer has been decremented to &FE, and so points to the next empty location. After the second pair of instructions the stack will appear as in figure 1c. The value

concerned, the only change is that the pointer is incremented with each PLA instruction. The items on the stack are not actually *removed*, they are simply copied into the accumulator, and the pointer incremented accordingly.

STACKING RETURN ADDRESSES

A similar process occurs every time that a JSR instruction is encountered, except that now, since the return address of a subroutine must be two bytes wide, two bytes are placed on the stack for each JSR instruction (they are pushed high byte first), and the stack pointer is decremented twice. As the stack is just one page in size this allows subroutines to be nested to around 126 levels deep (providing that the stack contains no other data). In practice, the way in which the 6502 handles return addresses is slightly unusual, in that the actual address stored is not the address of the next instruction, as one might have expected, but one less than this.

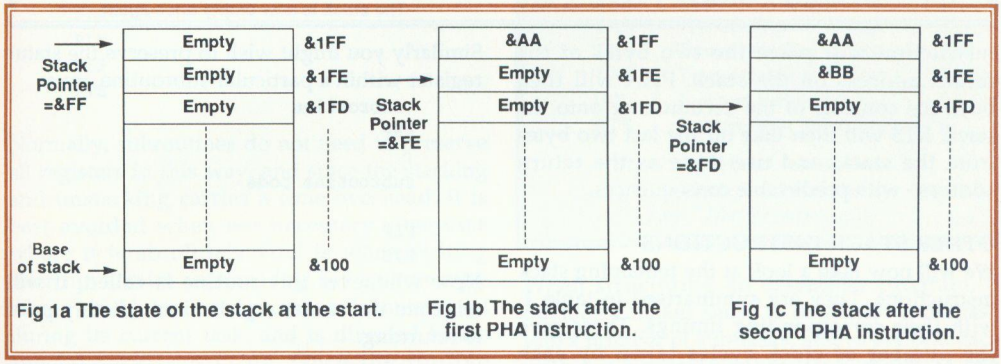


Fig 1a The state of the stack at the start.

Fig 1b The stack after the first PHA instruction.

Fig 1c The stack after the second PHA instruction.

&BB has been added to the stack at the next empty location. Note that although we refer to the process as *pushing*, the first item on the stack has not been *pushed* downwards at all. That would be far too inefficient. It is much easier to leave it where it is and to alter the pointer.

Figures 2a and 2b show the effect of the following two instructions on our stack:

```
PLA
PLA
```

As far as the appearance of the stack is

This has very little consequence for the user, since the programmer is never normally aware of stacked addresses, nor indeed of the value of the stack pointer; though the instruction pair TSX, TXS (Transfer Stack pointer to X register, and Transfer contents of X register to Stack pointer) does permit the user to read and write to the stack pointer if he so wishes.

Having seen how the stack operates, one extremely important proviso to its use is apparent. If you are using the stack for the temporary storage of data using PHA or

whatever, you must *always* unstack the data before trying to access stack return addresses. For example, the following will completely crash your machine:

```
JSR subroutine
:
.subroutine
LDA #&FF
PHA
RTS
```

The reason for the fatal error is that the JSR instruction will place the two bytes of the return address on the stack. PHA will then push the contents of the accumulator onto the stack. RTS will then take off the last two bytes from the stack, and use these as the return address - with predictable consequences.

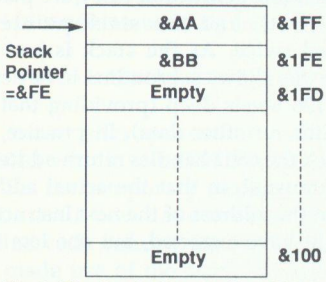


Fig 2a The Stack after the first PLA instruction.

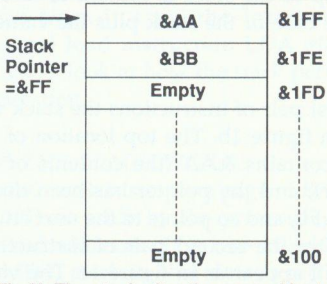


Fig 2b The stack after the second instruction. The stack is now empty once again

OTHER STACK INSTRUCTIONS

We will now take a look at the remaining stack instructions. They are summarised in table 1, with their corresponding timings. These give the number of clock cycles taken by each instruction, and we shall return to them in a moment. After PHA and PLA, the next pair of instructions in our table are PHP and PLP. These are used to Push and Pull the processor status register to and from the stack.

The status register holds the 6502's seven status flags (including the zero flag, the carry flag and so on). It can often be very useful to save these at certain stages in a program, and particularly when entering a subroutine which may corrupt them in various ways. For example, you may wish to stack the status register before performing an operating system call. Almost all operating system calls alter the status register. In the following piece of code, the status register is preserved before OSWRCH is called to output a query ("?"):

```
PHP
LDA #ASC("?")
JSR oswrch
PLP
```

At the end of this sequence all flags will be in exactly the same state as they were at the start.

Similarly you might wish to preserve the status register within a particular subroutine, thus:

```
.subroutine
PHP
:
subroutine code
:
PLP
RTS
```

Now whenever this routine is called, it will reinstate the status register immediately prior to returning.

You can extend the principle so as to preserve the contents of the accumulator:

```
.subroutine
PHP
PHA
:
subroutine code
:
PLA
PLP
RTS
```

It is worth a moment's pause to notice the order of pushing and pulling. We pushed the status register *first* so we must pull it *last*. This is

because of the stack's LIFO structure. If we get the wrong order then the contents of the status register and of the accumulator will be switched.

If it is necessary for a subroutine to preserve *all* registers, then you can use PHX and PHY if you have a Master Series machine. If not, you will have to use TXA and TYA to transfer the contents of the X and Y registers to the accumulator before pushing, and the reverse (using TAX and TAY) after pulling. Here you have to be even more careful to get the order correct:

```
.subroutine
PHP
PHA
TXA:PHA
TYA:PHA
:
subroutine code
:
PLA:TAY
PLA:TAX
PLA
PLP
RTS
```

Normally, subroutines do not need to preserve all registers in this way, and since the stacking and unstacking carries a time overhead, it is best avoided when not necessary. One case where it is absolutely vital is when writing interrupt service routines (e.g. for an interrupt-driven clock). Here, the processor is interrupted during its current task, and is diverted to the special service routine. Once the routine releases the processor to continue its previous tasks *all* registers must be reinstated or the system will fail.

TIMING AND STRUCTURE

I have included in table 1 the relative timings of the various instructions. These are supplied in terms of clock cycles, and PHA, as you can see, takes 3 cycles. If the processor is running at 2 MHz, then this instruction will take 1.5 micro-seconds to execute. The timing overhead for saving and subsequently restoring *all* registers to the stack on a Model B is a considerable 36 clock cycles. This works out at 18 micro-seconds with a 2MHz clock. And although not massive in itself, soon adds up if the routine is carried out many times. Table 1 also gives the timings for JSR and RTS. These are both 6 clock cycles in duration, and represent relatively time-intensive instructions. This has implications for program structure.

Table 1: 6502 Stack Instructions

PHA	Push Accumulator	3
PLA	Pull Accumulator	4
PHP	Push Processor Status	3
PLP	Pull Processor Status	4
TSX	Transfer Stack Pntr to X	2
TXS	Transfer X to Stack Pntr	2
PHX	Push X Register *	3
PLX	Pull X Register *	4
PHY	Push Y Register *	3
PLY	Pull Y Register *	4

Associated Instructions

TXA	Transfer X Reg to Acc	2
TAX	Transfer Acc to X Reg	2
TYA	Transfer Y Reg to Acc	2
TAY	Transfer Acc to Y Reg	2
JSR	Jump to Subroutine	6
RTS	Return from Subroutine	6

Stack and related instructions, giving timings in clock cycles
Key: * Master series only

When structuring an assembler program, it obviously makes sense to break down the task into a number of sub-tasks, and to perform each within a separate subroutine, much as one might in Basic using procedures and functions. Each sub-task may itself be further sudivided, and each of its sections performed within a separate subroutine. This all makes for more legible code, and reduces the likelihood of programming errors. However, notwithstanding this, you should take care that any time-intensive sections of a program do not contain too many subroutines, especially in cases where code is repeatedly called.

Next month we will take a look at machine code graphics.

DOUBLE VIEW

Mike Williams experiences an attack of double vision in trying out this new adjunct for Acorn's View word processor.

Product	DoubleView
Supplier	Tubelink P.O.Box 641, London NW9 8TF. Tel. 01-205 9393
Price	£39.95 (5.25" or 3.5" disc) £49.95 (two 16K ROMs)

DoubleView is a ROM-based piece of software from Tubelink designed to work with Acorn's View word processor, and providing both additional features such as split-screen dual-file editing, and enhanced versions of existing ones. The software is in the form of two 16K ROMs, or in ROM image format for loading into sideways RAM. It is accompanied by a 32 page manual. You must also have the View word processor installed in your machine, and version 3.0 or later at that.

DoubleView also needs one 16K bank of sideways RAM for use as workspace, and the manual also recommends the use of shadow RAM if you want to be able to store reasonable quantities of text in memory.

Once installed on your system, DoubleView is called up with *DBVIEW. This takes you directly into View's edit mode, but with an additional status line (in inverse video) at the foot of the screen. This information, which is continually updated, shows the name of the current document and printer driver. It also shows the number of free bytes, which markers are set,and your position in the current document as a percentage. 0% is displayed when the cursor is at the start of the document, 100% when at the end and other values appropriately in between.

One point to note is that DoubleView will only work in modes 0 and 3 (most dedicated View users seem to prefer mode 3), but for extensive use in these modes a high resolution monochrome monitor is much kinder on the eyes than the typical medium resolution colour monitor. And in these modes, DoubleView can also provide a 106 column screen as an alternative to the more usual 80 columns if required.

CONTROL KEY FUNCTIONS

Within edit mode, text may be entered and processed as usual with View - nothing has been changed in this respect. However, a range of Control key functions has been added which implement a good many of DoubleView's extra features. Ctrl-f9 provides a *Delete Word* function, while Ctrl-T and Ctrl-E will move the cursor to the top or bottom of the current screen.

One of the most important improvements is the option to handle two separate documents together, which gives the package its name. Once the first document has been loaded, Ctrl-X will toggle the alternative screen window, and a second text file may then be loaded. In addition, Ctrl-F also allows the user to switch between having just one editing window and two windows on the screen. Thus parts of both documents may be displayed simultaneously, and Ctrl-X then toggles between the two screens. Ctrl-W allows the relative sizes of the two windows to be adjusted, while Ctrl-U and Ctrl-D allow the text in the other window (i.e. not the one in which you are currently working) to be scrolled up and down. This saves unnecessary toggling between windows.

Another feature implemented via Control keys is a *clipboard*. The clipboard is a file on disc created and maintained by DoubleView. Any piece of text delineated by markers may be copied (Ctrl-L) or cut (Ctrl-K) to the clipboard, and the contents of the clipboard may be pasted (Ctrl-P) at the current cursor position.

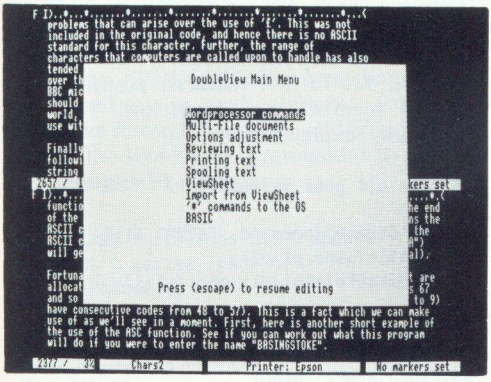
START-UP MODES

Unlike View, DoubleView allows the user to specify a wealth of start-up options and the names of up to two files for editing when called with the *DBVIEW command. This information is supplied as a set of parameters following the command itself.

The choice is quite comprehensive, and as such it would probably make more sense to put a command into a !BOOT file. However, DoubleView addresses this problem too, and a single 'O' option may be used to specify the name of an *options* file.

Using this facility you can select a one or two window screen on entry, screen mode (0 or 3), and the colours of text and background. Other options allow the user to determine the size of the upper window (and thus also the lower one), choice of normal or wide screen (106 columns), and the cursor flash rate.

There are some more interesting options too. Disc buffering, if selected, allows more memory per document when editing two documents together by saving the non-active document when a switch is made from one window to the other. Disc buffering can use floppy disc (but this is quite slow), hard disc or a RAM disc if available. The result is that even when editing two documents, each document can use all available memory.



The 'Recovery Files' option implements a system whereby any document being edited is saved at regular intervals (determined by the user), for recovery in the event of some catastrophe. With dual file editing, only the active file is backed up in this way as the non-active file is saved anyway.

Another important option provides a 'Multi-File' facility. A *master* file may be set up with the names of View files. If the master file is specified on entry, the first View file named is loaded. Moving to the end of this file and pressing Ctrl-Cursor Down automatically saves the current document and loads in the next. You can also move back to a previous document in the same way. As a means of allowing much larger files to be edited this is a step in the right direction, but it would have been really good if this had been made totally transparent to the user.

THE MENUS

Pressing Escape in edit mode no longer takes you into View's normal command mode, but into DoubleView's main menu. This provides 10 menu options replacing and supplementing View's own commands. These options include:

Word Processor Commands to deal with View commands like Search, Replace, Fold etc.

Multi-File Documents to create and edit the master files referred to before.

Options Adjustment to set or change start-up options from within DoubleView.

There is also a facility to enter ViewSheet (Acorn's spreadsheet equivalent of View), and from within that package to use two further Control keys to return to DoubleView or save a spreadsheet to the clipboard. Tubelink promises a full *DoubleSheet* for the future.

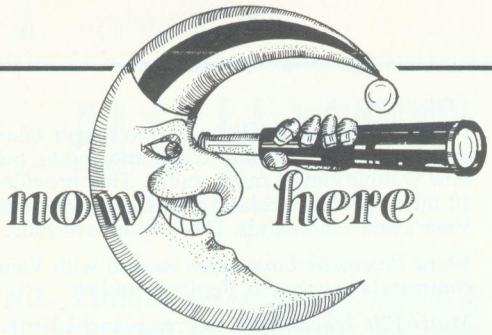
CONCLUSIONS

I have mixed feelings about the usefulness of DoubleView. As a longstanding proponent of View for word processing, I welcome any product which will enhance View's editing environment. The dual-file editing facility and split screen, together with the clipboard, are very welcome, though it would have been useful to be able to view the contents of the clipboard without having to load them into one of the two screen windows.

The automatic back-up at regular intervals would certainly have been a life-saver for me in the past, and I suspect for many others. But I am less sure about the multi-file facility for the reason stated.

The start-up functions are attractive, and the 106 column screen, on a high resolution mono monitor, is eminently readable for those wider documents. The manual explains everything well, though the order of the contents seems a little haphazard. And all Control key functions and start-up options are listed for reference at the back.

Provided you have both sideways and shadow RAM, then at just under £40 the price of this product seems reasonable. If you edit many long text files, and frequently need to cut and paste between documents then I am sure that you will rapidly find DoubleView to be invaluable. B



ANYONE FOR AN ARGUMENT?

Dec McSweeney continues his foray into the realms of C by adding some new features to the editor started last month.

Last month we discussed arrays, pointers and structures, illustrating the concepts with a line editor suitable for creating and editing C program sources (in particular). This month we'll start by adding a few extra facilities.

The editor is controlled from a main function which accepts a line from the keyboard and calls a function depending on the *verb* part of the line. To add any new verbs, we need to change **main** so they are recognised. This is achieved by inserting a new **if** statement in the **while** loop, of the form:

```
...
  if(!strncmp(command, "VERB",n)){
        /* preprocess */
        verb(arguments);
        }
...
```

The string "VERB" is obviously the new command, followed by "n", its length. If you recall, **strncmp(string1, string2, num)** compares the first **num** characters of **string1** and **string2**, returning FALSE if they match. The exclamation mark before the function call is C-speak for NOT, so if the function returns FALSE the **if** statement evaluates to TRUE. The **preprocess** comment field might for example be replaced by call(s) to **split** if the verb has more than one argument.

Astute readers, examining the code in **main()**, may have noticed that the command is tested against each valid value every time, which could have been avoided using an "if - else if - else ..." construction. An extended construction of this type can cause a stack overflow during compilation on the Beeb, so I have stuck to the simpler though less elegant approach. I don't believe the processing overhead is too serious, though.

As promised, we can quite easily incorporate a clone of the Basic AUTO command. The command has two arguments, **start-line** and **increment**, both optional with a default of 10. In Basic, the AUTO facility is switched off by pressing Escape. This editor, however, is a program and Escape causes it to terminate! Within a C program Escape could be used by disabling Escapes with a *FX call and testing each character input for ASCII code 27. This is neither portable nor convenient - **getstring** would have to be amended. I have therefore selected *Tab in first character position* as my AUTO terminating condition. The code (hopefully) explains itself:

```
/* this bit goes in main() */

   if(!strncmp(command, "AUTO",4)){
     split(line,p1,p2);
     editauto(atoi(p1),atoi(p2));
     }

/* AUTO function */
editauto(start, ainc)
int start;
int ainc;
{
char acommand[80];
if(start <= 0)
  start = 10;
if(ainc <= 0)
  ainc = inc;
printf("Enter <Tab><Return> to exit auto\n");
do{
  printf("%d ",start);
  getstring(acommand, 80);
  if(acommand[0] != '\t')
     insert(start, acommand);
  start += ainc;
  } while(acommand[0] != '\t');
}
```

The call to **editauto** is preceded by a call to **split**, dividing the remainder of the command line into two parts. The call to **editauto** converts

p1 and p2 to numeric with the library function atoi() - "ASCII to integer".

Within **editauto**, we test first for valid values in **start** and **ainc**, substituting defaults if necessary. **start** (a copy of p1) is displayed on the input line, and **getstring()** obtains the input. This is passed to **insert()** with the line number, **start** is incremented and the process is repeated. In the **do-while** loop we test for the terminating Tab (known in C as \t) after each line is input, performing an exit when appropriate.

Two other functions remain to be added if our C editor is to behave like its Basic ancestor; RENUMBER and DELETE. Both have two numeric arguments, like AUTO, though DELETE's are not optional. Unlike AUTO, they can be achieved by manipulating the control table, **info**. RENUMBER involves changing the line numbers in each element, following **ptr->next** through the table. DELETE involves linking the line before **argument-1** to the line after **argument-2**. Here goes:

```
/* C.DELREN - routines for C editor */
/* BEEBUG C series, part 4          */
/* by D.McSweeney 1988              */

/* preamble */
extern struct txtcontrol {
 char *ptext;
 struct txtcontrol *prev;
 int linenum;
 struct txtcontrol *next;
 };
extern int inc, firstline, lastline;
extern struct txtcontrol *firstpointer,
*lastpointer;
#define NULL 0
#define FALSE 0
/* The "#define"s included instead */
/* of #include <h.stdio>          */
/***** end of preamble ************/

/* The functions */
/* RENUMBER the whole program */
editrenum(start,rinc)
int start;
int rinc;
{
struct txtcontrol *ptr;
if(rinc <= 0)
   rinc = inc;
if(start <= 0)
   start = 10;
ptr = firstpointer;
firstline = start;
do{
  ptr->linenum = start;
  start += rinc;
  ptr = ptr->next;
  } while(ptr != NULL);
lastline = start;
}
/*  DELETE a range of lines */
editdel(start, finish)
int start;
int finish;
{
struct txtcontrol *ptr, *ptr1;
/* Validation */
printf("delete - %d,%d",start,finish);
if(start > finish || start > lastline ||
finish < firstline){
   printf("Error\n");
   return(FALSE);
   }
/* Search for first line to delete */
for(ptr = firstpointer; ptr->linenum <
start; ptr = ptr->next)
   /* do nothing - all the work is */
   /* done in the 'for' statement  */
    ;
ptr1 = ptr->prev;
/* search for the line after the */
/* deleted area                  */
for( ; ptr->linenum <= finish && ptr->next
!= NULL; ptr = ptr->next)
   /* do nothing again */
    ;
ptr->prev = ptr1;
ptr1->next = ptr;
}
```

If you're the proud owner of Beebug's C, you may like to enter the above into a file called C.DELREN. The purpose of the 'preamble' section will be explained in good time. Other C users can append the source starting at "The functions".

The '||' in the validation section of **editdel** means *logical OR* by the way. Note also the empty initialising segment of the second **for** in **delete()** - **ptr** is already set to the required value.

These routines may be added to the original source (using last month's version of the editor as it stands), or can be held in a separate file and invoked with a **#include** line in the main source. There is a third way of adding them - of which more later.

COMMAND LINE ARGUMENTS

To edit an existing file with this program you have to type:

```
*RUN edit <Return>
LOAD c.source <Return>
```

Dullsville! Wouldn't it be great if you could run the editor by typing:

```
RUN edit c.source <Return>
```

Guess what! You can! A simple modification to the main function is all it takes. The facility is called a *command line argument* and is based on the fact that **main()** actually has two arguments:

1. An integer, containing a count of the number of *words* on the command line.
2. A pointer to an array of character strings - the words themselves.

Conventionally, these are referred to as **argc** and **argv**. They appear thus in the program:

```
main(argc, argv)
int argc;
char *argv[];
{ /* program starts here */
```

The first argument pointed to, **argv[0]** is by convention the program name. The count, **argc**, is therefore always at least 1. To tweak our editor to accept a filename as a command line argument, modify the start of **main()** as above, then insert the following:

```
if(argc == 2)
   progload(*argv[1]);
```

and Bob's your uncle. To be really flash, having entered Beebug C (by typing *C), you can type:

```
COMMAND 2
```

which causes C to try and run any unrecognised commands. Then all you need to type is:

```
EDIT c.source <Return>
```

Command line arguments are most useful in programs where you want to reduce the amount of interaction between the user and the program. Consider the small file copying program in last month's article. The program copies a file (MYFILE) onto the end of another called OUTFILE. To deal with any other files, we would have to rename them or recompile the program with the new filenames. With command line arguments we can generalise the program, allowing the user to type, for example:

```
fappend newdata master
```

and let the program do the work. Here's the revised listing.

```
/** FAPPEND - command line argument **/
/**          version               **/
#include <h.stdio>
#include <h.string>
main(argc, argv)
int argc;
char *argv[];
{
FILE *in, *out;
int c;
char infile [25];
char outfile[25];
if(argc == 3){
   strncpy(infile, argv[1], 24);
   strncpy(outfile, argv[2], 24);
   /* open both files */
   if((in=fopen(infile,"r")) &&
(out=fopen(outfile,"a"))){
/** if successful, do the copy **/
     while((c=fgetc(in)) != EOF)
        fputc(c,out);
     fclose(in);
     fclose(out);
   }
   else
      printf("It's all gone wrong!");
}
else
   printf("Format: fappend <infile>
<outfile>");
/* the end */
}
```

Here we first test the value of **argc**. If this is not 3 (program + file + file) we drop through to print the "Format" message. Otherwise, the program copies the arguments into our arrays **infile** and **outfile** (using the library function **strncpy()**) and proceeds as before. The arrays are given a size of 25 to cope with the longer ADFS filenames (though DFS filenames should not exceed 13 characters including drive and directory), plus the terminating null character.

A truly helpful programmer would use **getstring()** to obtain the filenames if they hadn't been supplied, but you should be able to do that for yourself by now.

Command line arguments are user-friendly without being programmer-hostile, and more examples may be found in Kernighan and Ritchie, the Beebug C user guide and any self-respecting C tutor.

Now we return to the various methods of incorporating the new functions in our editor. There are several possibilities - adding them to the end of the source file or using a #include statement have already been mentioned. A third method involves compiling the new file C.DELREN separately, then linking it to the editor. This is the reason for the preamble, where various "global" variables are declared (though not defined) to stop the compilation falling over.

COMPILING AND LINKING C PROGRAMS

A C program such as C.SOURCE must first be compiled:

```
COMPILE SOURCE
```

This creates a file called O.SOURCE which contains "semi-compiled" code, full of unresolved references to "external functions" like **printf()**, **fopen()** etc. This must be "linked" to produce an executable program in E.SOURCE:

```
LINK SOURCE
```

The linker tidies up the mess by fetching the external functions from a library file. It can, at the same time, join two or more semi-compiled files together. In this example, we are going to compile our delete and renumber functions as a freestanding program, and join it to the main program at the link stage. The **main()** function will refer to the functions as described above, but they will not exist in the program. To complete the picture, we should also declare the functions as external:

```
/* insert this before "external */
/* variables" in C.EDIT         */
/*  external functions          */
int editrenum(int, int);
int editdel(int, int);
```

This will prevent the compiler getting confused - **printf()**, **scanf()** and so on are declared like this in **h.stdio**. Our external declarations in C.DELREN tell the compiler not to look for a variable called **firstpointer**, for example, until the linking stage.

The final sequence of events is:

- edit C.EDIT to include the declarations and calls to **editdel** and **editrenum**;
- create C.DELREN as above;
- COMPILE EDIT
- COMPILE DELREN
- LINK EDIT, DELREN

The last instruction will create a file called E.EDIT which contains all the necessary code. The beauty of this approach lies in ease of maintenance. If we wish to amend either **editrenum** or **editdel** we need only edit and compile C.DELREN, then repeat the LINK. With large programs and commonly used routines (think of **getstring**) this has immense benefits.

On this month's disc you will find all the necessary source files, plus an all-singing all-dancing version of EDIT to play with in a 6502 stand-alone compilation. Next month we'll tackle the creation of our own library.

THE C EDITOR

The complete editor, written in C, is supplied on the magazine disc in a stand-alone format. This means that it can be run on any BBC micro, even if you don't have a C compiler. The program is called editor, and may be called with:

***editor <Return>**
or *RUN editor <Return>

You may also specify a source file name as a parameter, for example:

***editor MYPROG <Return>**

The editor responds to the following commands:

AUTO <start>,<inc>
RENUM(BER) <start>,<inc>
DELETE <start>,<end>
LOAD <filename>
SAVE <filename>
LIST <start>,<end>
DIAG(NOSTIC) <start>,<end>
HELP
QUIT

These are intended to function similarly to their Basic counterparts. Apart from SAVE and LOAD, all other parameters are optional. Remember this is a line editor; any line may be entered at any time provided it is given a line number, and edited as per Basic. Remember, too, that all files edited in this way must be ASCII files.

BEEBDOS FROM MICROBOSS

If you need to transfer data between your BBC micro and a PC why not let BeebDos from Microboss do the work for you. David Somers powers up his PC and explains all.

Product	**BeebDos v2.01**
Supplier	**Microboss Limited, 3 Hadleigh Road, Frinton-On-Sea, Essex CO13 9HG.**
Price	**£46.00 inc VAT.**

I have always been a faithful follower of Acorn, my first computer being an Atom (remember that?), then a Model B, and finally a Master Turbo. However, I have since moved over to the PC camp and ended up buying an AT compatible.

Now, like many people, I am faced with the problem of transferring data between the two computers. This is where BeebDos comes to our aid for it allows a PC to read and write to BBC formatted discs.

The minimum system requirement is an IBM PC/XT/AT or compatible with at least 128 Kbytes of user memory and a 360 Kbyte floppy drive. This will allow you to read and write to double density 40 track DFS discs which were formatted with either the Acorn ADFS; Watford DDFS; Solidisk DDFS versions 1, 2, 2.1, or 2.2; UDM DDFS (Microware); Opus DDOS (Challenger); or Acorn 1770 DFS with an ACP DDFS ROM.

To read or write to Acorn SINGLE DENSITY formatted 40 track DFS discs you will need an AT or XT286 with a high capacity (1.2 Mbyte) floppy disc adaptor (FDA) and a 360 Kbyte floppy drive. On the 1.2 Mbyte drive you can also read or write to ADFS 'L' 'M', and double density 40 or 80 track discs.

TESTPACK

If you have ever used a PC system you will know that there are many different combinations of hardware. Unfortunately some combinations of hardware may not be compatible with BeebDos. To see if BeebDos will work with your particular system a Test Pack is available from Microboss for £5.75 inc. VAT. It consists of a wallet containing the necessary instructions and some test software on disc. If the tests succeed, and you place an order for BeebDos within 30 days, you are entitled to a £5.75 discount. Unfortunately, if the test fails, you get no refund.

USING BEEBDOS

BeebDos comes supplied in a plastic wallet containing a 36 page A5 instruction manual and a 5.25" 360 Kbyte floppy disc. It consists of a number of separate utility programs which are invoked by typing their name at the DOS prompt. If you have IBM's Fixed Disc Organizer (FDO) or Microsoft Windows then data files are supplied for you to access the utilities through these front-end systems.

When the utilities are executed you are prompted for various pieces of information. For ease of use, especially once you have become accustomed to the command's syntax, parameters can be placed after a utility's name on the command line. This is particularly useful as it allows you to call the utilities from within a batch file (which would supply the parameters). In this way it is quite possible to automate the transfer procedure.

COPYING FILES

When BeebDos is first used it needs to be informed which BBC filing system was used to format the disc to be read or written to. This is

necessary as the BeebDos routines bypass the BIOS (the PC's MOS) and use their own routines to handle disc I/O.

The main utility is BCOPY which copies files between PC and Beeb formatted discs, or vice versa. Wildcards can be specified to copy multiple files. When copying from Beeb format to PC discs you can specify what extension is to be used if any. Additionally, when copying from ADFS discs, filenames are truncated to their first eight characters.

When transferring text files, certain characters are not directly mapped between the two systems - the most noticeable being the "pound sign". A utility, BCONV, can be called upon to apply standard alterations to an IBM file. Details of the alterations are stored in a text file, whose name is supplied to BCONV along with the file to be converted. This utility is like using the global replace function in the Master's text editor EDIT. Several sample alteration files are supplied with the package.

FILING UTILITIES

Several other BeebDos utilities perform functions normally provided with the appropriate filing system on the Beeb. These are locking or unlocking a file (i.e. setting or clearing the L flag), renaming a file, backing up a disc, cataloguing a disc, compacting a disc, formatting a disc, renaming a file, setting the disc title and boot options, and deleting files (equivalent to *WIPE).

When working with ADFS discs you can create sub-directories, and select and enter a sub-directory. You can also obtain information on entries in the free space map and the amount of free disc space (equivalent to *MAP and *FREE).

LOOKING PRETTY

The final utility provided is the most aesthetic of them all, for it allows BBC screens to be viewed on a PC. To use this facility your PC must be equipped with one of the following graphics adaptors: standard colour graphics (CGA); enhanced colour graphics (EGA); hercules monochrome graphics (HGC).

BBC screen dump files for modes 0, 1, 4, or 5 can be viewed. Various options allow the width, top-left x and y co-ordinates, and vertical compression to be altered to suit your display. Also, the screen can be saved in a format suitable for displaying with BasicA's BLOAD command on the PC.

FINAL COMMENTS

BeebDos makes transferring programs and data between the different filing systems simplicity itself. The ultimate test has to be this article which was written on my AT. The text was spooled out of my word processor, passed through BCONV to correct minor differences between the PC's and Beeb's character set (namely the pound sign), then transferred with BCOPY to an ADFS formatted disc, which was then sent to BEEBUG for editing and typesetting (but see editorial comment).

Although BeebDos initially seems complicated to use, once you have familiarised yourself with the commands you soon realise just how powerful and useful it is. It certainly gets top marks from me. B

EDITORIAL NOTE

We can confirm this, and the original text was read into View for final editing. But do remember that although these utilities can transfer files between the two types of micro, it does not help you to use a file so transferred. 'Text only' (ASCII) files can normally be transported from one word processor to another with only small modifications. On the other hand, programs may well prove quite a problem, unless you are dealing with the source code for compiled languages such as C or Pascal in ASCII format, which may be altered in a text editor or similar. Basic programs may be converted to ASCII format, but there are many variations between different versions of this language.

BASIC II TO BASIC I CONVERSION (continued from page 44)

```
 5080 IF P%<&1000 PRINT;"0";~P%; ELSE PR
INT;~P%;
 5090 I$="":IF?P%<&10 I$="0"
 5100 I1$="":IF?(P%+1)<&10 I1$="0"
 5110 I$=" "+I$+stng$~?P%:I1$=" "+I1$+st
ng$~?(P%+1)
 5120 PRINTI$;I1$:P%=P%+2:=PASS%
 5130 :
 5140 DEF FNEQUD(dword1):LOCAL dword,I%,
I$,I1$:dword=dword1
 5150 !P%=dword:IFPASS%<>3P%=P%+4:=PASS%
 5160 IF P%<&1000 PRINT;"0";~P%; ELSE PR
INT;~P%;
 5170 I1$="":FORI%=P%TOP%+2:I$="":IF?I%<
&10 I$="0"
 5180 I1$=I1$+" "+I$+stng$~?I%:NEXT:PRIN
TI1$'"    ";
 5190 I$="":IF?(P%+3)<&10 I$="0"
 5200 I$=" "+I$+stng$~?(P%+3):PRINT I$:P
%=P%+4:=PASS%
 5210 :
 5220 DEF FNEQUS(stng$):LOCAL str$,Len,C
%,I%,I$:str$=stng$:Len=LENstr$
 5230 FOR C%=1 TO Len:?(P%+C%-1)=ASC MID
$(str$,C%,1):NEXT
 5240 C%=0:IF PASS%<>3 P%=P%+Len:=PASS%
 5250 IF P%<&1000 PRINT;"0";~P%; ELSE PR
INT;~P%;
 5260 I%=P%:REPEAT:I$="":IF?I%<&10I$="0"
 5270 I$=" "+I$+stng$~?I%:PRINTI$;:I%=I%
+1
 5280 C%=C%+1:IFC%MOD3=0ANDI%<>P%+Len C%
=0:PRINT'"    ";
 5290 UNTIL I%=P%+Len:PRINT:P%=P%+Len:=P
ASS%
 5300 :
 5310 DEF PROCoscli(line$)
 5320 $buf%=line$:X%=buf%:Y%=buf%DIV256:
CALL &FFF7
 5330 ENDPROC
```

BECOME A DABHAND AT C (continued from page 45)

own right, provide good examples of C programming style. Because the book is aimed at users of a variety of machines, no mention is made of sound or graphics outside the machine-specific chapters mentioned above.

As far as the style of the text goes, the tone is friendly and the explanations are full and easy to understand without being patronising. The drawback to this approach lies in the length of some explanations, which reduces the book's value as a reference work. As an example, an entire (two-page) chapter is devoted to the format and use of comments where a single sentence might have sufficed. Many topics, though, deserve the fullest possible treatment and it scores over other books on C in this respect.

The early chapters are illustrated with cartoons showing, for example, a car ("at the highest level, an engine is just a black box"), or an axe truncating a floating point number when it is moved to an integer variable. I found many of these confusing rather than illuminating. That said, the program structure diagrams which illustrate the larger programs are very helpful.

The 510 pages cover all the important aspects of C, and include a brief summary of the language and a 15-page index to help you find your way about. For readers who don't want to type in the long listings from the book, there is a programs disc available. There are different versions for all the machines covered, with the BBC version containing source code suitable for both BEEBUG and Acornsoft C. The BBC disc costs £9.95, and an order form is given in the book.

In conclusion, then, a very good, reasonably priced introduction to C for the non-specialist. I doubt, though, that it will become a standard reference work on the language. Perhaps more experienced users would find Kernighan & Ritchie a better buy, though it assumes a greater knowledge on the part of the reader, and at over £20 a greater commitment!

David Spencer presents some more technical hints for Cambridge Computer's Z88.

MAKING NOISES

The VDU drivers in the Z88 provide two basic ways of making a beeping sound. The simplest sound you can make is the default beep from within Basic using VDU 7. There is however, a more advanced way that allows you to make a repeated beep, rather like the Z88's Alarm. This extended beep instruction allows you to specify how many beeps there are, and both the length of the actual beep, and the time between beeps. The command is:

```
VDU 1,52,33,32+b,32+s,32+m
```

where 'b' is the number of beeps, 's' is the space period (the time between beeps), and 'm' is the mark period (the length of the beep). Both 's' and 'm' are given in 1/100s of a second. The sound is generated under the control of interrupts, which means that program execution continues while the sound is made. For example:

```
VDU 1,52,33,42,52,92
```

will generate ten long beeps in quick succession, while:

```
VDU 1,52,33,37,132,40
```

will sound a 'pip' every second for five seconds.

USER DEFINED CHARACTERS

One very powerful feature of the Z88 is the ability to display user-defined characters. This can partly make up for the lack of high resolution graphics, by defining appropriate graphics characters. On an unexpanded machine, (one which doesn't have a 128K RAM card in slot one), you can only have 16 user defined characters, while on a machine with 128K in slot 1 you can define up to 64 characters. The characters are defined on a 6x8 matrix (6 across by 8 deep). The character definitions are specified in much the same way as on a BBC computer, except that only bits 0 to 5 of each byte are used, because of the six bit width. In addition, bit 7 of each byte must also be set. The command to define a character is thus:

```
VDU 1,138,61,char,a,b,c,d,
e,f,g,h
```

Here char is the character number to be re-defined (in the range 64-127), and a-h are the eight bytes that define the character ('a' is the top row, and 'h' is the bottom). For example:

```
VDU 1,138,61,64,140,140,132,
191,140,140,146,161
```

will define character 64 to be a human like figure.

Unlike a BBC machine, user defined characters are not allocated ASCII codes as such. Instead they co-exist with the standard characters. To print a user defined character you would use the command:

```
VDU 1,50,63,char
```

where char is the character number in the range 64-127. To print twenty of our men across the screen, assuming the character is already defined, you could use:

```
10 FOR count =1 TO 20
20 VDU 1,50,63,64
30 NEXT
```

HARD AND SOFT RESETS

Many people seem confused about the difference between a hard and soft reset, and how to generate them. When a soft reset is issued, the Z88 will kill all the currently suspended tasks, along with the current task, and because no tasks are running it will drop you into the Index. This means for example, that any Basic program or Pipedream document you are working on will be lost, but all files in RAM will be retained. A hard reset on the other hand will do everything that a soft reset does, but also clear out all stored files. This makes a hard reset a very drastic last measure.

A soft reset is brought about either by going into the index, pressing the diamond key and typing PURGE, or by pressing the reset button (next to the power socket) when the cartridge flap is closed. To perform a hard reset, open the flap, press the reset button, and then close the flap again. It is also possible that the computer will perform an automatic hard reset if the batteries run right down, causing the contents of memory to be corrupted.

OVER TO YOU

If you have any hints on the Z88 that you feel would interest others, or if you have any technical questions on the Z88, we would like to hear from you. Write to Z88 Hints at the BEEBUG address given elsewhere in the magazine. B

IS THE PRINTER ON LINE (Revisited)

Neil Stephens

It is often very useful to determine whether a printer is on line or not. By doing this you can prompt the user to press the On-Line button before trying to print something, rather than just starting and then hanging up when the buffer gets full. The easiest way of doing this is to flush the printer buffer, print two null characters and then see how much space is in the buffer. If the buffer is empty then the printer is on-line and the two nulls have been printed (with no effect because they are nulls). If the buffer isn't empty at the end of the test, then the printer is not on-line. The reason for using *two* nulls is that the operating system will try to send the first one to the printer even if it isn't on-line. The following simple function returns TRUE if the printer is on-line, and FALSE if it isn't.

```
DEF FNonline:*FX21,3
VDU 2,1,0,1,0,3
=(ADVAL-4=63)
```

FORCING CAPS AND SHIFT LOCK ON (Revisited)

Dave Somers

For some programs it would be nice to set up the Caps and Shift Locks to a known state. For example, a word processor could turn Caps Lock off when in edit mode, or a game that used Caps Lock and Ctrl could force the Caps Lock on before terminating. This can easily be accomplished by using *FX202 to change the keyboard status flags. To force Caps Lock off you can use *FX202,16,239, and to put it back on - *FX202,0,239. The equivalents for Shift Lock are *FX202,32,223 and *FX202, 0,223. *FX202,48,207 will force both locks off.

WRITE PROTECTING FILES

Keith Lowe

When using the ADFS or a network, both of which allow files to be given the attribute Write (W), Read (R) and Locked (L), it is important to realise the exact effect of each attribute. In particular, when you are reading or writing a file byte by byte rather than loading or saving it, the L attribute will not stop the file being changed. In other words, even if the L attribute is set, a file can still be opened using OPENUP and its contents changed. The file cannot however be opened using OPENOUT because that would involve deleting the old file, something that the L attribute prevents. The way to prevent writing to the file is to make sure that the W attribute is not set. [BEEBUG member Mr Lowe discovered this problem while using Viewsheet, when he set the L attribute on some files to stop them being changed, and then discovered to his horror that the program had changed them all.]

TELETEXT CHARACTERS

Ben Kestner

As you may know, the teletext character set used for the BBC's mode 7 is slightly different from the ASCII character set used in other modes. Normally the VDU drivers in the operating system translate the teletext characters before they are displayed on the screen making them equivalent to ASCII characters. This can sometimes cause problems if you are writing a program which must have the true teletext characters, a prime example being viewdata terminal software. There is, however, an easy way to bypass the operating system's translations simply by ensuring that each character has bit 7 set before being displayed. This means each character should be ORed with 128 before being output, for example:

```
VDU(char OR 128)
```

This also works in the opposite direction, in that most viewdata terminals produce text with bit 7 set. If pages are to be spooled to a file, and then loaded into a word processor, it is first necessary to reset bit 7 of each character with AND 127 otherwise garbage will appear in place of the text. B

POSTBAG POSTBAG

READER SURVEYS

Other magazines regularly conduct reader surveys to find out what their readers think about the present content and format, and what readers would like to see included or omitted. I would suggest that popping the editorial question, "do you want more C language articles", although an easy option is not the best, since a minority of members (fervent C protagonists) may write in reply "yes", while the silent majority may not really want additional C coverage.

What would I like in BEEBUG? Certainly more articles involving the use of sideways RAM, some adaptations of disc menus and catalogues to double density disc systems, greater mention of Solidisk products, and finally an article which SIMPLY explains load, start and execute addresses.

C.P.Jennings

Mr Jennings may well be right in his assessment of the response we are likely to receive regarding C. However, there is a place for minority interests in the magazine, as well as articles of wider general interest. We already send questionnaires to random samples of members, and we intend to repeat these at regular intervals.

As to Mr Jennings' requests, I think recent issues prove that we are still covering the use of sideways RAM, although articles on double density disc systems are less likely because of the inherent variations between systems. We would be only too pleased to review Solidisk products if they would provide them for review. Lastly, the idea of a simple article on file addresses sounds an excellent one, and is already being pursued in our First Course series.

IMPROVED SPRITE EDITOR

I am writing with some suggested improvements to the Sprite Editor (BEEBUG Vol.6 No.8). When you want a background colour other than black, it becomes very tedious colouring each individual block. The following routine, added to the original program, allows the screen to be cleared (using Ctrl-f8) in the colour currently chosen for plotting.

```
 290 IF funckey=228 PROCclear
3082 DEF PROCclear
3083 PROCcursor
3084 GCOL0,128+curC%
3085 VDU24,0;0;(maxX%+1)*step
X%-dotsizeX%;(maxY%+1)*stepY%
-dotsizeY%;:CLG
3086 VDU24,chrX%;chrY%;chrX%+
((maxX%+1)*dotsizeX%)-1;chrY%
+((maxY%+1)*dotsizeY%)-1;:CLG
:VDU26
3087 PROCcursor
3088 ENDPROC
```

To give an indication of what the present colour is, insert the following lines:

```
 315 COLOUR128:COLOUR7:PRIN
TTAB(0,0)"Colour ";:COLOUR
curC%+128:PRINT;" "
```

Then change line 2630 to:

```
2630 chrY%=1000-dotsizeY%*Y
width%
```

Robert Alcock

Thanks to Mr Alcock for supplying these updates to this program. The revised program is included on this month's magazine disc/tape.

LACK OF CONTROL

I have noticed an error in your article on Character Control (First Course, BEEBUG Vol.6 No.9). You say that the null character is represented in programs by "". This actually represents a null string; the only way to represent the null character is CHR$(0). If you output the null character to the screen the effect may be the same as outputting a null string, but if output to a printer there will often be a difference.

The distinction can easily be demonstrated in immediate mode by typing:

```
A$="X"+""+"Y"
PRINT A$
PRINT LEN(A$)
B$="X"+CHR$(0)+"Y"
PRINT B$
PRINT LEN(B$)
```

G.Davies

I can only plead guilty - Mr Davies is quite right in what he says. I was trying to recall just this information at the time of writing, but signally failed to do so. B

BEEBUG MEMBERSHIP

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

BEEBUG SUBSCRIPTION RATES

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

BACK ISSUE PRICES

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

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

FURTHER DISCOUNTS

We will allow you a further discount:

Five or more:	deduct £0.50 from total
Ten or more:	deduct £1.50 from total
Twenty or more:	deduct £3.50 from total
Thirty or more:	deduct £5.00 from total
Forty or more:	deduct £7.00 from total

POST AND PACKING

Please add the cost of p&p:

Destination	First Item	Second Item
UK, BFPO + Ch.I	40p	20p
Europe + Eire	75p	45p
Elsewhere	£2	85p

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

BEEBUG MAGAZINE is produced by BEEBUG Ltd.

Editor: Mike Williams
Assistant Editor: Kristina Lucas
Technical Editor: David Spencer
Technical Assistant: Lance Allison
Production Assistant: Yolanda Turuelo
Membership secretary: Mandy Mileham
Editorial Consultant : Lee Calcraft
Managing Editor: Sheridan Williams

CONTRIBUTING TO BEEBUG PROGRAMS AND ARTICLES

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

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

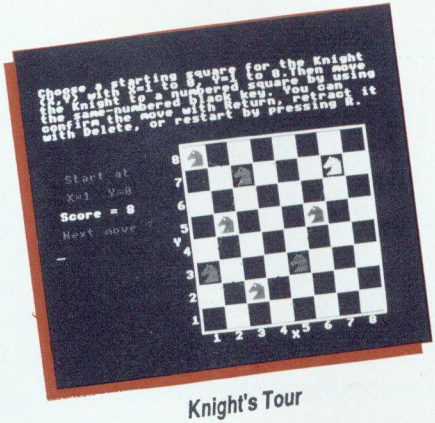
In all communication, please quote your membership number.

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

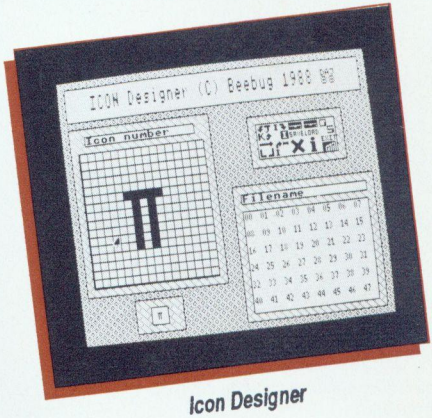
Magazine Disc/Cassette

JUNE 1988 DISC/CASSETTE CONTENTS

KNIGHT'S TOUR - a graphic computer approach to the age-old problem in chess.
CURVE DRAWING WITH SPLINES - how to draw smooth curves through random sets of points.
FILE HANDLING FOR ALL (Part 2) - a program to amend and delete records in serial files.
NOW C HERE (Part 4) - source code of editor and additional routines, plus a full stand-alone version of the editor for all to use.
BEEBUG MINI-WIMP (Part 2)
ICON DESIGNER - this month a sophisticated icon designer, including for convenience the MiniWimp ROM ready to load.
SMART RENUMBER - a comprehensive renumber utility for well-structured programs.
FIRST COURSE
DISC MANAGER - this comprehensive and efficient DFS utility is repeated from Vol.5 No.4.
THE MASTER SERIES
SHADOWS IN THE DARK - a demonstration of the use of shadow memory for animation.
CHORDS AND KEYBOARDS - convert modern chord notation into keyboard notes, plus a further utility for printing blank piano-style keyboards.
ADFS VIEW MENU UPDATED - an updated and improved version of this popular program.
BASIC II TO BASIC I CONVERSION - a complete set of procedures and functions to assist in converting Basic II programs to run under Basic I.
SPRITE EDITOR - an extended version of this program first published in BEEBUG Vol.6 No.8.
MAGSCAN - bibliography for this issue (Vol.7 No.2).



Knight's Tour



Icon Designer

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

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

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

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

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

The Best Deals on Archimedes From BEEBUG

The Archimedes Specialists

1 0% FINANCE

For a limited period we are able to offer 0% APR finance over 9 months on the purchase of any Archimedes. You pay no interest at all. This is a brand new scheme only available from BEEBUG. The deposit and repayments are shown below.

	Deposit	9 Payments		Deposit	9 Payments
A305 Base	£83.85	£80.00	A310 Base	£90.25	£89.00
A305 Mono	£87.35	£86.00	A310 Mono	£102.75	£94.00
A305 Colour	£106.85	£103.00	A310 Colour	£113.25	£112.00
A310M Base	£96.25	£96.00	A440 Base	£267.85	£264.00
A310M Mono	£108.75	£101.00	A440 Mono	£271.35	£270.00
A310M Colour	£119.25	£119.00	A440 Colour	£290.85	£287.00

2 TRADE IN YOUR OLD BBC, MASTER OR COMPACT FOR AN ARCHIMEDES

We will be pleased to accept your old computer (in working condition) as part exchange towards the purchase of an Archimedes. (If you use the finance scheme this will replace your initial deposit on a 305/310, so you pay nothing now). Allowances are as follows:

BBC Issue 4 No DFS	£125
BBC Issue 4 DFS	£175
BBC Issue 7 No DFS	£175
BBC Issue 7 DFS (Or B+)	£225
Master 128	£250
Compact Base System	£215

Please phone for allowances on other Compact and Master systems.

3 FREE DISCS & PC EMULATOR

Join RISC USER, the Archimedes magazine and support group, and purchase your Archimedes by Cheque, Access, Visa, Official Order or 11.5% finance and we will supply you, absolutely free, 10 3.5" discs, a lockable disc storage box, printer lead and the latest version of The PC Emulator from Acorn. Altogether you save more than £142.00.

Prices Including VAT

A305 Base	£803.85	Mono £861.35	Colour £1033.85
A310 Base	£891.25	Mono £948.75	Colour £1121.25
A440 Base	£2643.85	Mono £2701.35	Colour £2873.85

4 11.5% FINANCE OVER 12 TO 36 MONTHS

As a Licensed Credit Broker we are able to offer finance on the purchase of any equipment, including the Archimedes. You still benefit from the free PC Emulator, discs, disc box and printer lead. (Typical APR 23% on the purchase of a 310 Colour system over 36 months.
Deposit £126.25 36 payments of £37.36).

5 DISCOUNTS FOR EDUCATION

We are able to offer attractive discounts to Education Authorities, Schools, Colleges and Health Authorities. Please write with your requirements for a quotation.

TO FIND OUT MORE PHONE OR WRITE NOW. TEL: 0727 40303

We offer a complete service, including Advice, Technical Support, Showroom, Mail Order and Repairs. Our showroom in St. Albans stocks everything available for the Archimedes. Call in for a demonstration.

Please indicate your requirements below.
Subscription to Risc User (£14.50 UK) ☐ Information Pack and Catalogue ☐ 0% Finance Form for 305/310/310M/440 Base/Mono/Colour ☐ 12-36 Months Finance Form for 305/310/310M/440 Base/Mono/Colour ☐ Trade In BBC/Master/Compact ☐ Purchase 305/310/440 Base/Mono/Colour ☐
UK Courier Delivery £7.00. Overseas please ask for a quotation.

I enclose a cheque value £.....
Please debit my Access/Visa/Connect Card No
Expiry...../..... with £.....

Name .....
Address .....
.....
Signature .....

Beebug, Dolphin Place, Holywell Hill, St. Albans, Herts AL1 1EX Tel: 0727 40303
BEEBUG – The Archimedes Specialists