Vol.9 No.10 April 1991

BEEBUG

FOR THE BBC MICRO & MASTER SERIES

```
BEEBUG MAGSCAN          VOLUMES 1-

Volume    : 1 2 3 4 5
Type      : All
String 1  : DATABASE
String 2  : SPREADSHEET
Logic     : OR

Filer Accounts Part 2
BEEBUG Database / Home Banking
Vol 5  No 5  Page 53

ROM Review - Database Publications
Instant Mini Office II - Communications
Wordprocessor, Database, Spreadsheet
Vol 5  No 8  Page 5

BEEBUG Education
ass Database Review - Newman Coll
abase for Schools, Colleges
  No 9  Page 14
```

● FAST FACTORISING FOR FUN

● COLLAPSING SCREENS

● COLOUR BLENDER

● MS-DOS AND DFS COMPARED

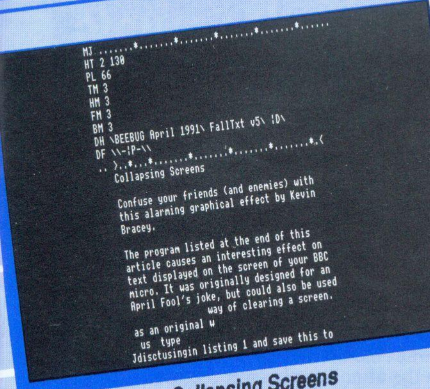
BEEBUG

Vol.9 No.10 April 1991

PROGRAM INFORMATION

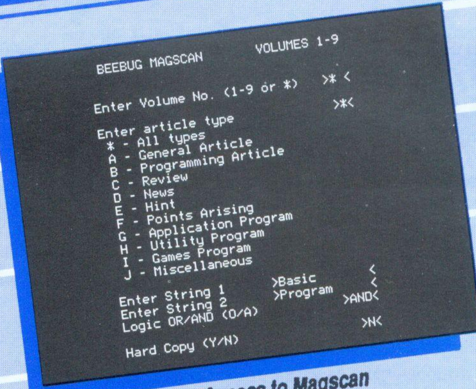
All listings published in BEEBUG magazine are produced directly from working programs. They are formatted using LISTO 1 and WIDTH 40. The space following the line number is to aid readability only, and may be omitted when the program is typed in. However, the rest of each line should be entered exactly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

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



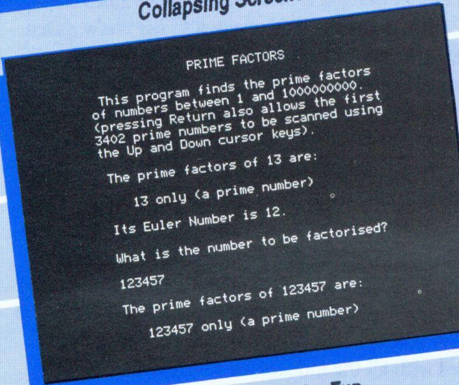
```
MJ .......*.......*.......*.......*.......*......
HT 2 130
PL 66
TM 3
HM 3
FM 3
BM 3
DH \BEEBUG April 1991\ FallTxt v5\ ¦D\
DF \\-¦P-\\  ..*.......*.......*.......*.<
.. >..*...*.......*
   Collapsing Screens

   Confuse your friends (and enemies) with
   this alarming graphical effect by Kevin
   Bracey.

   The program listed at the end of this
   article causes an interesting effect on
   text displayed on the screen of your BBC
   micro. It was originally designed for an
   April Fool's joke, but could also be used
               way of clearing a screen.

   as an original w
     us  type
   Jdisctusingin listing 1 and save this to
```

Collapsing Screens



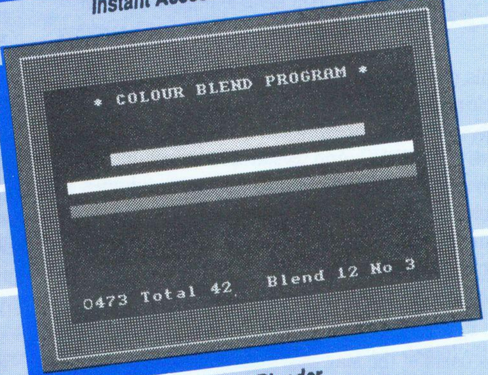
```
BEEBUG MAGSCAN            VOLUMES 1-9

Enter Volume No. (1-9 or *)      >* <
                                  >*<
Enter article type
 * - All types
 A - General Article
 B - Programming Article
 C - Review
 D - News
 E - Hint
 F - Points Arising
 G - Application Program
 H - Utility Program
 I - Games Program
 J - Miscellaneous

Enter String 1       >Basic       <
Enter String 2       >Program     <
Logic OR/AND (O/A)              >AND<

Hard Copy (Y/N)                   >N<
```

Instant Access to Magscan



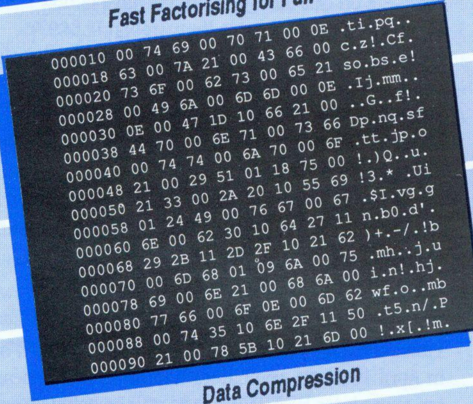
```
              PRIME FACTORS

  This program finds the prime factors
 of numbers between 1 and 1000000000.
 (pressing Return also allows the first
 3402 prime numbers to be scanned using
 the Up and Down cursor keys).

 The prime factors of 13 are:

     13 only (a prime number)

 Its Euler Number is 12.

 What is the number to be factorised?

 123457

 The prime factors of 123457 are:

     123457 only (a prime number)
```

Fast Factorising for Fun



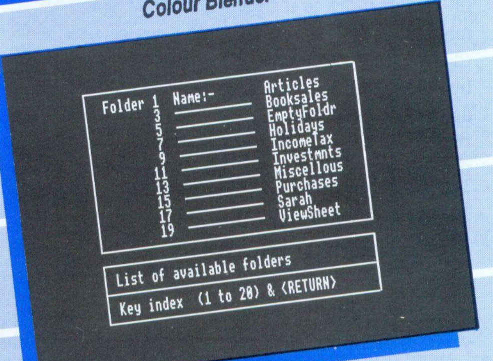
```
   * COLOUR BLEND PROGRAM *




 0473 Total 42     Blend 12 No 3
```

Colour Blender



```
000010 00 74 69 00 70 71 00 0E .ti.pq..
000018 63 00 7A 21 00 43 66 00 c.z!.Cf.
000020 73 6F 00 62 73 00 65 21 so.bs.e!
000028 00 49 6A 00 6D 6D 00 0E .Ij.mm..
000030 0E 00 47 1D 10 66 21 00 ..G..f!.
000038 44 70 00 6E 71 00 73 66 Dp.nq.sf
000040 00 74 74 00 6A 70 00 6F .tt.jp.o
000048 21 00 29 51 01 18 75 00 !.)Q..u.
000050 21 33 00 2A 20 10 55 69 !3.* .Ui
000058 01 24 49 00 76 67 00 67 .$I.vg.g
000060 6E 00 62 30 10 64 27 11 n.b0.d'.
000068 29 2B 11 2D 2F 10 21 62 )+.-/.!b
000070 00 6D 68 01 09 6A 00 75 .mh..j.u
000078 69 00 6E 21 00 68 6A 00 i.n!.hj.
000080 77 66 00 6F 0E 00 6D 62 wf.o..mb
000088 00 74 35 10 6E 2F 11 50 .t5.n/.P
000090 21 00 78 5B 10 21 6D 00 !.x[.!m.
```

Data Compression



```
Folder  1  Name:-  ________   Articles
        3          ________   Booksales
        5          ________   EmptyFoldr
        7          ________   Holidays
        9          ________   IncomeTax
       11          ________   Investmnts
       13          ________   Miscellous
       15          ________   Purchases
       17          ________   Sarah
       19          ________   ViewSheet

 List of available folders
 Key index  (1 to 20) & <RETURN>
```

ADFS/View Utility for the Master

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

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

Program will not function on a cassette-based system.

Program needs at least one bank of sideways RAM.

Program is for Master 128 and Compact only.

Editor's Jottings

MAGSCAN

In this month's issue of BEEBUG you will find an article and program related to BEEBUG's *Magscan*. Magscan is a computerised index to all the back issues of BEEBUG, and was first launched in 1985 when it was supplied with the indexes to volumes 1 to 3 inclusive. Subsequently, monthly indexes have been included on the magazine disc for each issue, and an annual Magscan update has been available after the completion of each volume.

Thus users of the Magscan system can maintain an up-to-date index to the contents of all their magazines, by typing in and creating new indexes themselves, by subscribing to the magazine discs, or by purchasing the annual update. As readers will appreciate, with nine complete volumes, that represents a wealth of information, yet Magscan provides an efficient and speedy way of locating a particular article, or finding the references to all articles on a particular subject, using the keyword system employed by Magscan.

In fact, Magscan has proved so successful over the years, that an equivalent program called *ArcScan* has been written for the Archimedes, and this is available to Archimedes users complete with indexes for both BEEBUG and RISC User, our magazine for Archimedes users (and, indeed, with indexes to all the Acorn manuals for the Archimedes). ArcScan uses an identical data format to Magscan, so all Magscan files are immediately accessible via ArcScan as well.

With the completion of volume 9 of BEEBUG, and the imminent start of volume 10, we feel that it is a good time to update Magscan, and more importantly draw Magscan to the attention of newer readers.

We have made some modifications to Magscan to improve flexibility. For example, 40 track users will find that it is essential to start using a second data disc once the index to volume 10 begins to be built up. The new version automates such considerations as far as possible.

Magscan is supplied on 40 track or 80 track (double-sided) 5.25" disc, or on 3.5" ADFS format disc (for the first time). It is accompanied by a manual and a set of release notes covering the latest modifications, and comes with all the indexes for volumes 1 to 9 complete on one disc. The full package is now available at the new price (to members) of £9.95. Existing Magscan users can obtain a full update for just £4.75 on returning their original disc or proof of purchase. Full details for ordering Magscan may be found elsewhere in this issue - please refer to this before placing an order.

As this issue of BEEBUG marks the completion of volume 9, all readers will receive with the first issue of volume 10 a full printed index to volume 9.

I am sure that when BEEBUG was first founded back in April 1982, its originators had no idea that nearly ten years later BEEBUG would still be going strong, or that the organisation would have grown to the size which it is today. Let us hope that the future is as interesting for all users of Acorn computers as the last ten years have been.

M.W.

News News News News News News

PANDERING TO BBC USERS

Proving that there is a viable market still for new products for the BBC micro, Panda Discs (perhaps best known for its music collections) has announced three new products for 1991.

Touch and *Learn* is a content free Concept Keyboard package which will appeal to teachers in primary and special schools, and it runs on all BBC micros and networks. The primary function of the software is to present questions on screen which can be answered by touching overlays on the Concept Keyboard. There is also a database mode, allowing multiple records to be linked to each overlay item.

The other two releases are both intended to enhance the use of Computer Concepts' word processor Wordwise Plus. *+Windows* enables green embedded commands to be obtained through a series of pull-down menus, making their selection or alteration much easier. This includes control of printing, and although *+Windows* is configured for Epson printer codes, it can be reconfigured by the user as required.

To complement +Windows Panda Discs has also produced *+Catalogue*, a versatile information storage system for Wordwise Plus. The software allows large amounts of information to be stored and searched using normal Wordwise Plus files. Subject to disc capacity, *+Catalogue* can search automatically up to 25 separate text files. It is claimed to be ideal for storing and searching record collections, sets of slides, names and addresses, video titles, books etc.

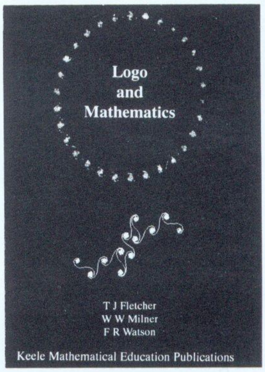
Each disc costs £9.95 including post and packing direct from Panda Discs, Four Seasons, Tinkers Lane, Brewood, Stafford ST19 9DE.

BEEBUG ON MICRONET

The BEEBUG database on Micronet has now moved from its original base node on page 800909 to a new one on page 800708. The keywords *BEEBUG and *DATABUS will still take you to the front page or the DATABUS menu. At the same time, the retail price lists have been removed from the database as it is proving impossible to keep them up to date. Instead, a free retail catalogue will be supplied to anyone who requests it, and there will be a new section on the database devoted to current special offers, including secondhand and shop-soiled items. It will still be possible to place orders and renew subscriptions via Prestel. An expansion of the editorial section is also planned. This will include more news and information about BEEBUG and all things Acorn. The hints and tips section may also be increased, and further areas may be added to enhance the database.

LOGO AND MATHEMATICS

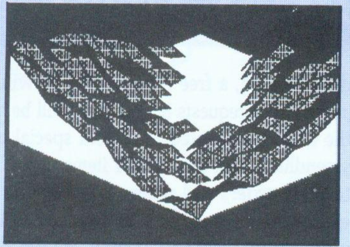
Logo and *Mathematics* is the title of a new book published by the Education Department of the University of Keele, and compiled by T.J.Fletcher, W.W.Milner & F.R.Watson, and running to just over 400 pages. It is a compendium of papers and articles written at different times, but all the examples of programming in Logo are based upon BBC Logo (available from Longman Logotron). Information is also given on any differences between that and Logo on the RML Nimbus and Acorn Archimedes, and a supporting disc of programs is available in any of these three formats.



The sections of the book discuss the application of Logo in the teaching of a variety of mathematical topics, for example: graphics, trigonometric functions, Fibonacci series, arrays and determinants, number theory, vectors, Turing machines and a great deal more. The book appears to represent an ideal source of ideas and applications based on the use of Logo and is available price £10.00 from Keele Mathematical Education Publications, Department of Education, University of Keele, Keele, Staffordshire ST5 5BG, tel. (0782) 621111. The supporting disc (state format) costs £9.00 (both prices are inclusive of post & packing).

Best of BEEBUG Software

Applications II Disc



Crossword Editor - for designing, editing and solving crosswords

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

3D Landscapes - generates three dimensional landscapes.

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

Running Four Temperatures - calibrates and plots up to four temperatures

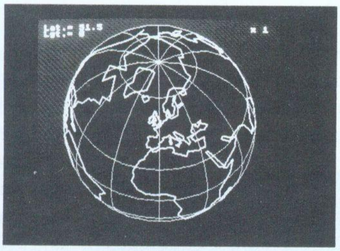
Julia Sets - fascinating extensions of the Mandelbrot set

Foreign Language Tester - foreign character definer and language tester

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

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

Applications I Disc



Business Graphics - for producing graphs, charts and diagrams

Video Cataloguer - catalogue and print labels for your video cassettes

World by Night and Day - a display of the world showing night and day for any time and date

Phone Book - an on-screen telephone book which can be easily edited and updated

Page Designer - a self contained page-making package for Epson compatible printers

Personalised Letter-Headings - design a stylish logo for your letter heads

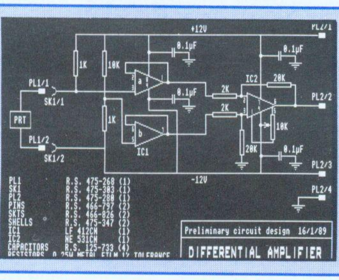
Appointments Diary - a computerised appointments diary

Mapping the British Isles - draw a map of the British Isles at any size

Selective Breeding - a superb graphical display of selective breeding of insects

The Earth from Space - draw a picture of the Earth as seen from any point in space

Personalised Address Book - on-screen address and phone book



ASTAAD

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

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

General Utilities Disc

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

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

EDIKIT ROM

An indispensible utility ROM for all Basic programmers, containing the following commands:

***FTEXT** (find text) ***FBASIC** (find Basic) ***FPROCFN** (find procedure/function)
***LPROC** (list procedure) ***LFN** (list function) ***LFROM** (list 8 lines of a program)
***RTEXT** (replace text) ***RBASIC** (replace Basic) ***SYSINF** (system information)
***VARLIST** (list program variables) ***FKDEFS** (function key definitions)

Incorporating the updated Basic Booster utilities:

SUPER SQUEEZE PARTIAL RENUMBER PROGRAM LISTER
RESEQUENCER SMART RENUMBER TEXTLOAD AND TEXTSAVE

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

Please add p&p - 60p for the first item and 30p for every additional item.

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

Collapsing Screens

Confuse your friends (and enemies) with this alarming graphical effect by Kevin Bracey.

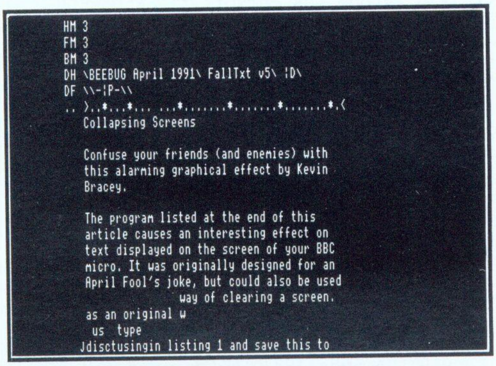
The program listed at the end of this article causes an interesting effect on text displayed on the screen of your BBC micro. It was originally designed for an April Fool's joke, but could also be used as an original way of clearing a screen.

Just type in listing 1 and save this to disc using any suitable name (but not Fall). Master users should check below for changes required for their machines. Run the program and the assembled machine code will be saved with the name Fall. To test this just type:

```
        *FALL
```

fill the screen with some text (by typing *CAT for example), press 'K' and enjoy the show as the letters fall into random heaps at the foot of the screen.

If you want to catch some friends out on April 1st, then simply load the machine code into their computer before they use it, press Break and wait!



```
HM 3
FM 3
BM 3
DH \BEEBUG April 1991\ FallTxt v5\ ¦D\
DF \\-¦P-\\
.. >..*...*... ...*.......*.......*.......*.<
   Collapsing Screens

   Confuse your friends (and enemies) with
   this alarming graphical effect by Kevin
   Bracey.

   The program listed at the end of this
   article causes an interesting effect on
   text displayed on the screen of your BBC
   micro. It was originally designed for an
   April Fool's joke, but could also be used
                 way of clearing a screen.
   as an original w
    us  type
  Jdisctusingin listing 1 and save this to
```

The text begins to fall

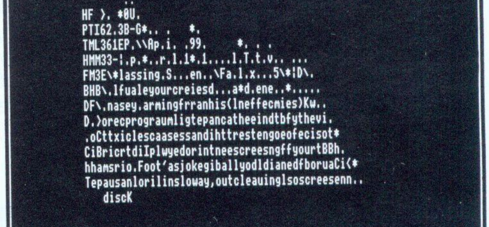
The assembled routine will remain in the machine through both Break and Ctrl-Break, and so can only be removed by switching the machine off. Pressing 'K' at any time after the code is loaded will initiate the routine, even within word processors, in the middle of listing a program, and while running a program. When the routine has completed its work, the computer will carry on with what it was doing unaware that anything untoward has happened. It can be quite entertaining to watch otherwise dignified programs come crashing to the ground.

SCREEN CLEARING

To use the routine as a novel way of clearing the screen in your own programs simply delete lines 180 to 380 of the assembly code program and change line 660 to read:

```
660 RTS
```

Run the program to assemble the revised code, and add the command *LOAD Fall at the start of your program. You will also need to append the procedure given in listing 2 to the end of your program. You can then use the procedure call PROCcls to clear the screen. Note that the higher the number of the screen mode you are using, the faster the routine will work.



```
       ..
HF >. *0U.
PTI62.3B-G*.. .   *.
TML361EP.\\Ap.i. .99.    *. . .
HMM33-¦.p.*..r.l.1*.1....l.T.t.v.. ...
FM3E\*lassing.S...en..\Fa.l.x...5\*¦D\.
BHB\.lfualeyourcreiesd...a*d.ene..*.....
DF\.nasey.armingfrranhis(lneffecmies)Kw..
D.>orecprograumligtepancatheeindtbfythevi.
.oCttxiclescaasessandihttrestengoeofecisot*
CiBricrtdiIplwyedorintneescreesngffyourtBBh.
hhamsrio.Foot'asjokegiballyodldianedfboruaCi<*
Tepausanlorilinsloway,outcleauinglsoscreesenn..
   discK
```

The final result

MASTER USERS

The routine normally loads at &1400, within an area of disc file work space normally unused on a model B. As this space is part of user memory on a Master, it is necessary to find somewhere else. If you are using shadow memory, then the best place is at &7B00. You must then set HIMEM=&7B00 before calling *Fall. Alter the program to reflect this in line 120, and don't forget to change line 30020 of PROCcls as well if you use it.

PROGRAM NOTES

The routine intercepts all 'Character entering buffer' events, and checks whether the triggering key has been pressed. This key can be altered in line 120 (but note that the routine

IS case sensitive). The number of characters which fall together can be altered for different effects in line 130. The variable MaxChars% should be set to a value in the range 2-255, this being the number of letters falling at a time. 255 is the most realistic value, but this is far too slow in any mode other than mode 7. The falling routine is 100% legal, accessing screen memory via documented OS routines. This means that it will work in any screen mode and shadow mode, and should even run on an Archimedes under the 6502 emulator.

The falling routine works by first scanning the screen for non-space characters, and storing the co-ordinates of any it finds in an array. When the array is full (its capacity being defined by the value of MaxChars%), it then checks the spaces below each character on the screen to see firstly if it can fall downwards, or secondly topple to one side. After every character has been moved, it checks to see how many characters actually moved. If none moved, then they have all fallen as far as they will go, and the routine goes on to make the next group of characters fall. Note, because of the algorithm used, the program may crash if it is activated when a large number of characters cannot fall, e.g. by pressing 'K' twice in succession. Have fun!

Listing 1

```
   10 REM Program Collapsing Screens
   20 REM Version B4.4
   30 REM Author  Kevin Bracey
   40 REM BEEBUG  April 1991
   50 REM Program subject to copyright
   60 :
  100 DIM code% &500
  110 OSWRCH=&FFEE:OSBYTE=&FFF4
  120 exec%=&1400:key%=ASC"K"
  130 MaxChars%=23
  140 FOR pass%=4 TO 7 STEP 3
  150 O%=code%:P%=exec%
  160 [
  170 OPT pass%
  180 .SetVec
  190 LDX &220:STX OldVec
  200 LDY &221:STY OldVec+1
  210 .SetVec2
  220 LDX #NewVec MOD 256:STX &220
  230 LDY #NewVec DIV 256:STY &221
  240 LDA #14:LDX #2:JSR OSBYTE
  250 LDA #247:LDX #&4C:LDY #0
  260 JSR OSBYTE
  270 LDA #248:LDX #SetVec2 MOD 256:LDY
#0
  280 JSR OSBYTE
  290 LDA #249:LDX #SetVec2 DIV 256:LDY
#0
  300 JSR OSBYTE
  310 RTS
  320 :
  330 .NewVec
  340 PHP:PHA:TXA:PHA:TYA:PHA
  350 CPY #key%:BEQ Fall
  360 .Leave
  370 PLA:TAY:PLA:TAX:PLA:PLP:JMP (OldVe
c)
  380 :
  390 .Fall
  400 LDA #134:JSR OSBYTE:STX pos:STY vp
os
  410 LDA #26:JSR OSWRCH
  420 LDA #17:JSR OSWRCH
  430 LDA #7:JSR OSWRCH
  440 LDA #17:JSR OSWRCH
  450 LDA #128:JSR OSWRCH
  460 LDX #0
  470 .VLoop:LDA cursoff,X:JSR OSWRCH
  480 INX:CPX #10:BNE VLoop
  490 LDA #2:STA &D0
  500 LDA #135:JSR OSBYTE
  510 LDA WTable,Y:STA Width
  520 LDA HTable,Y:STA Height:STA MaxY
  530 :
  540 .Floop
  550 JSR FindChars
  560 LDA Pointer:CMP #0:BEQ Finished
  570 JSR Drop:JMP Floop
  580 .Finished
  590 LDA#31:JSR OSWRCH
  600 LDA pos:JSR OSWRCH
  610 LDA vpos:JSR OSWRCH
  620 LDA#0:STA &D0:LDX#0
  630 .V2Loop
  640 LDA curson,X:JSR OSWRCH
  650 INX:CPX #10:BNE V2Loop
  660 JMP Leave
  670 :
  680 .FindChars
  690 LDA #0:STA Pointer
  700 LDA MaxY:STA Ypos
  710 .Yloop
  720 LDA #0:STA Xpos
  730 LDA #31:JSR OSWRCH
```

```
 740 LDA #0:JSR OSWRCH
 750 LDA Ypos:JSR OSWRCH
 760 .Xloop
 770 LDA #135:JSR OSBYTE
 780 CPX #0:BEQ FindNext
 790 CPX #32:BEQ FindNext
 800 LDY Pointer
 810 LDA Xpos:STA CharX,Y
 820 LDA Ypos:STA CharY,Y
 830 INC Pointer:LDA Pointer
 840 CMP #MaxChars%:BEQ Exit
 850 .FindNext
 860 INC Xpos:LDA Xpos
 870 CMP Width:BEQ Uprow
 880 LDA #9:JSR OSWRCH
 890 JMP Xloop
 900 .Uprow
 910 DEC Ypos:LDA Ypos:BPL Yloop
 920 LDA #0:STA Ypos
 930 .Exit
 940 LDA Ypos:STA MaxY
 950 RTS
 960 .Drop
 970 LDA Pointer:STA NotMoved
 980 LDA #0:STA DCount
 990 .DLoop
1000 LDY DCount
1010 LDA CharX,Y:STA Xpos
1020 LDA CharY,Y:STA Ypos
1030 CMP Height:BNE Continue
1040 .GotoNoFall
1050 JMP NoFall
1060 .Continue
1070 LDA #31:JSR OSWRCH
1080 LDA Xpos:JSR OSWRCH
1090 LDA Ypos:JSR OSWRCH
1100 LDA #135:JSR OSBYTE
1110 STX CChar
1120 LDA #10:JSR OSWRCH
1130 LDA #135:JSR OSBYTE
1140 CPX #32:BNE FallRight
1150 LDA CChar:JSR OSWRCH
1160 LDA #11:JSR OSWRCH
1170 LDA #8:JSR OSWRCH
1180 LDA #32:JSR OSWRCH
1190 LDX DCount:INC CharY,X
1200 JMP NextChar
1210 .FallRight
1220 LDA #31:JSR OSWRCH
1230 INC Xpos:LDA Xpos:JSR OSWRCH
1240 INC Ypos:LDA Ypos:JSR OSWRCH
1250 LDA #135:JSR OSBYTE
1260 CPX #32:BNE FallLeft
1270 LDA #11:JSR OSWRCH
1280 LDA #135:JSR OSBYTE
1290 CPX #32:BNE FallLeft
1300 LDA #10:JSR OSWRCH
1310 LDA CChar:JSR OSWRCH
1320 LDA #11:JSR OSWRCH
1330 LDA #8:JSR OSWRCH
1340 LDA #8:JSR OSWRCH
1350 LDA #32:JSR OSWRCH
1360 LDX DCount:INC CharX,X:INC CharY,X
1370 JMP NextChar
1380 .FallLeft
1390 LDA #31:JSR OSWRCH
1400 DEC Xpos:DEC Xpos
1410 LDA Xpos:JSR OSWRCH
1420 LDA Ypos:JSR OSWRCH
1430 LDA #135:JSR OSBYTE
1440 CPX #32:BNE NoFall
1450 LDA #11:JSR OSWRCH
1460 LDA #135:JSR OSBYTE
1470 CPX #32:BNE NoFall
1480 LDA #10:JSR OSWRCH
1490 LDA CChar:JSR OSWRCH
1500 LDA #11:JSR OSWRCH
1510 LDA #32:JSR OSWRCH
1520 LDX DCount:DEC CharX,X:INC CharY,X
1530 JMP NextChar
1540 .NoFall
1550 DEC NotMoved:BEQ NoneMoved
1560 .NextChar
1570 INC DCount:LDA DCount
1580 CMP Pointer:BNE StartAgain
1590 JMP Drop
1600 .StartAgain
1610 JMP DLoop
1620 .NoneMoved
1630 RTS
1640 :
1650 .WTable
1660 EQUB 80:EQUB 40:EQUB 20:EQUB 80
1670 EQUB 40:EQUB 20:EQUB 40:EQUB 40
1680 .HTable
1690 EQUB 31:EQUB 31:EQUB 31:EQUB 24
1700 EQUB 31:EQUB 31:EQUB 24:EQUB 24
1710 .Height:EQUB 0
1720 .cursoff
1730 EQUB 23:EQUB 1:EQUD 0:EQUD 0
1740 .curson
```

Continued on page 28

Magscan

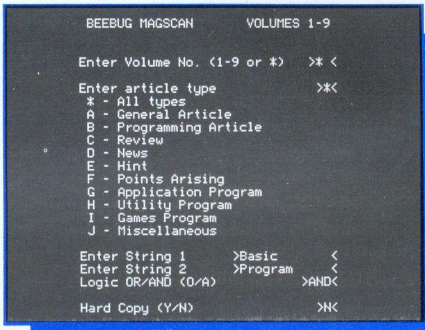
Comprehensive Magazine Database for the BBC Micro and the Master 128

An updated version of Magscan, which contains the complete indexes to all BEEBUG magazines from Volume 1 Issue 1 to the latest Volume 9 Issue 10

Magscan allows you to locate instantly all references to any chosen subject mentioned anywhere in the 90 issues of BEEBUG magazine.

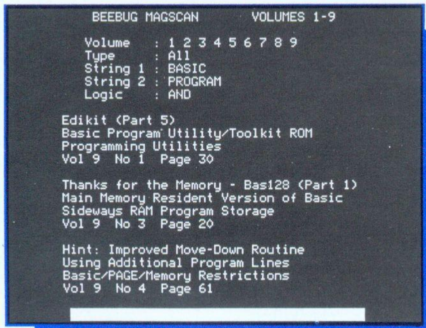
Just type in one or two descriptive words (using AND/OR logic), and you can find any article or program you need, together with a brief description and reference to the volume, issue and page numbers. You can also perform a search by article type and/or volume number.

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



```
           BEEBUG MAGSCAN        VOLUMES 1-9

         Enter Volume No. (1-9 or *)    >* <

         Enter article type              >*<
          * - All types
          A - General Article
          B - Programming Article
          C - Review
          D - News
          E - Hint
          F - Points Arising
          G - Application Program
          H - Utility Program
          I - Games Program
          J - Miscellaneous

         Enter String 1        >Basic        <
         Enter String 2        >Program      <
         Logic OR/AND (O/A)                >AND<

         Hard Copy (Y/N)                    >N<
```

Specifying a Magscan search



```
        BEEBUG MAGSCAN        VOLUMES 1-9

      Volume     : 1 2 3 4 5 6 7 8 9
      Type       : All
      String 1   : BASIC
      String 2   : PROGRAM
      Logic      : AND

    Edikit (Part 5)
    Basic Program Utility/Toolkit ROM
    Programming Utilities
    Vol 9  No 1  Page 30

    Thanks for the Memory - Bas128 (Part 1)
    Main Memory Resident Version of Basic
    Sideways RAM Program Storage
    Vol 9  No 3  Page 20

    Hint: Improved Move-Down Routine
    Using Additional Program Lines
    Basic/PAGE/Memory Restrictions
    Vol 9  No 4  Page 61
```

Entries retrieved from Magscan files

Some of the features Magscan offers include:

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

Phone your order now on (0727) 40303

or send your cheque/postal order to the address below. Please quote your name and membership number. When ordering by Access, Visa or Connect, please quote your card number and the expiry date.

Magscan complete pack, contains disc, manual and release notes: **£9.95**+p&p

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

Magscan update, contains disc and release notes: **£4.75** +p&p
(for update, please return old disc, label or evidence of purchase)

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

Postage: **a** - 60p UK, £1 Europe + Eire, £2.40 Elswhere **b** - £1.50 UK, £2.50 Europe + Eire, £4.80 Elswhere

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

Instant Access to Magscan on a Master

Ronald Smith shows how BEEBUG's popular computerised magazine index can be speeded up using the Master's sideways RAM.

Soon we shall be reading the tenth volume of BEEBUG and many readers, like myself will treasure every issue of the magazine which has guided us through the jungle of understanding of the BBC micro. It will always be a source of reference to those who are staying loyal to the eight bit wonder which entered our lives in 1981. I had been working with computers for twenty years previous to this and when I saw the BBC specification, I did not believe that anyone could make such a jump forward in computer technology. But Acorn did it and I still marvel at what has and what can still be achieved at such small cost.

To be able to refer to items in BEEBUG, the Beebug Bibliography *Magscan* has been available, and with the Magscan updates, has grown into a large index system. I have recently decided that my Master could provide me with a speedy method of look-up by bringing into use the four banks of sideways RAM. This involves creating a database from the Magscan index, and then loading this into RAM for quick memory searching. I will now take you through the steps needed to achieve this.

First, format a new disc into ADFS format as the larger capacity of this format will be needed. Next transfer all Magscan files which are named VOLxx from your DFS discs to the new disc. To do this place a Magscan disc in drive 1 (assuming dual drives) and enter DFS by typing *DISC. Then list the directory of the disc using *CAT 1. Make a note of all the VOLxx files. Put the ADFS disc in drive 0 and enter *ADFS. Then use:

```
*MOVE-DISC-:1.$.VOLxx -ADFS-:0.$.VOLxx
```

to do the transfer. It is necessary to do this file by file until you have all the Magscan files on the ADFS Disc. The Magscan DFS disc can now be removed.

Keep the list of all the VOLxx Magscan files you have on the new disc. When you have all the Magscan VOL files on your ADFS disc, it is necessary to enter *ACCESS VOL* WR to make them accessible.

The format of each item in a Magscan file is shown in figure 1.

```
Line 1 ^(up-arrow) followed by a classification letter
Line 2 Subject of magazine item
Line 3 Additional info on item
Line 4 \(backslash) followed by issue number and page number
```

Figure 1. Format of Magscan entries

Figure 2 shows the first few lines of VOL1a of Magscan. I decided to condense this into one line for the database. View was employed to do this using mode 131 (the Master's Edit program would be an equally good alternative - see *Mastering Edit* in this and previous issues). It is important to cancel formatting by using Ctrl-f2 whilst in the entry mode. Then enter command mode and load the Magscan file using READ (not LOAD) VOLxx. I used a suffix of E before the file name to denoted an edited Magscan file, so then entered NAME EVOLxx to be ready to save each file which will eventually constitute the database.

```
^C
Brief Review of the BBC Micro
Hardware / Software  OS 0.1 & OS 1.0
\1 5
^A
Machine Ordering and Supply
BBC Micro Deliveries
\1 8
^J
3D Surfaces
```

Figure 2. Original Magscan file format

As these are to be collected together, the first requirement is to insert into each item the volume number. To do this enter the command:

```
CHANGE/\/\BBx /
```

where 'x' is the volume number from the file, and I added BB purely for search purposes. Note the space after the 'BBx'. When this command is executed, the number of strings changed will be reported. If edit mode is entered, it will be seen that the last line of each item reads \BB followed by the volume number, the issue number and the page. To convert the item into one line means going back to command mode and making some devious changes.

The first is:

```
CHANGE/^C/^C:/
```

which places a colon at the beginning of each line. The second command:

```
CHANGE/:^^^?/^C/
```

replaces the first line with a Carriage Return as I did not require the classification letter. The next command:

```
CHANGE/^C:/,/
```

removes all the Carriage Returns and colons within an item and so converts to single lines. Next:

```
CHANGE/,\/ \/
```

removes the comma before the issue details. It is possible to cut down the size of the file by abbreviating some common words and phrases. For instance I used:

```
CHANGE/Points Arising/P.A./
```

which saved a lot of space. Figure 3 shows the single line conversion and some further items for abridging can be seen. All that remains is to go into edit mode and cancel some empty lines at the top and bottom of the text.

The number of free bytes is shown at the top of the screen in command mode, and this should be deducted from 28926 to give the number of bytes in the edited file. This should be recorded against the list of Magscan files for later use. Save the file and then proceed to edit further volumes.

The four banks of sideways RAM hold 16K bytes each, but the first &200 bytes will be used by the database program so to be safe, only 15K of information should be loaded into each bank.

View is used to read each of the edited VOL files in number order into memory. By adding up the number of bytes in each file from the list previously made, the files can be appended one after the other until the 15K maximum is reached. A terminator is entered as the last line of the file with | BBMSx(File Name).

If you have used the abbreviation facility you should be able to get nearly all the original files into three full new files, and the remainder in the last file. This leaves room for making additions as new issues of BEEBUG come along. I used the names - BBMSA, BBMSB, BBMSC and BBMSD for the Database files. (BBMS being BEEBUG MagScan). You have now a mass of information at your fingertips.

```
Brief Review of the BBC Micro,Hardware / Software  OS 0.1 & OS 1.0 \BB1 1 5
Machine Ordering and Supply,BBC Micro Deliveries \BB1 1 8
3D Surfaces,16k Graphics Program \BB1 1 9
Sound,Using Sound and Envelope \BB1 1 10
The 10 Most Asked Questions,Hardware / Software Problems Answered \BB1 1 13
Screen Scrolling,Screen Handling Hint - Page Mode \BB1 1 15
```

Figure 3. Converted Magscan file format for SWR

To obtain access to this database, you should enter the program given in listing 1 and save it with the name BBUGDB onto the disc with the other files. Run the program and it will make four further files under names: HDBBGA, HDBBGB, HDBBGC, HDBBGD.

The next task is to create a BOOT file for loading your sideways RAM. Enter *BUILD !BOOT and create a file with the lines:

```
 1. *SRLOAD HDBBGA 8000 4
 2. *SRLOAD HDBBGB 8000 5
 3. *SRLOAD HDBBGC 8000 6
 4. *SRLOAD HDBBGD 8000 7
 5. *SRLOAD BBMSA 8200 4
 6. *SRLOAD BBMSB 8200 5
 7. *SRLOAD BBMSC 8200 6
 8. *SRLOAD BBMSD 8200 7
 9. ?&2A5=130
10. ?&2A6=130
11. ?&2A7=130
12. ?&2A8=130
```

Set the disc to read the BOOT file by using *OPT4,3 To use the BEEBUG DATABASE, put your DATABASE disc in drive 0, press Shift-Break and the sideways RAM will be loaded and be available until the computer is switched off or the SWR is used for something else.

To look up an item in the Index, enter:

```
  *BBGA Screen Scrolling
```

as an example. Almost instantly you will see listed on the screen all the lines having *Screen Scrolling* in the first file of the index. If you don't find what you want repeat with:

```
  *BBGB Screen Scrolling
```

to look through the next file. You can go through the four files in this way. You may find it easier to put the screen in scroll mode (VDU 14).

There are some points to note. The command line *BBGx etc. will accept any case, and the search does not differentiate between upper and lower case. The search looks for a sequence of characters equalling the input string and so *BBGA Print would give all items having PRINT, Print, print, PRINTER, PRINTING and so on. To confine the search to a specific string, the '\' (backslash) should terminate. Thus *BBGA print\ will only give lines with PRINT, Print etc. in them.

The '?' (question mark) can be used for wild card characters thus *BBGA P?int would bring up POINT, PRINT, PAINT. The commas in a line are replaced with a Returns so the information is displayed as it would be in Magscan. Enclosing text between < and > will cause any commas in the text to be displayed as commas. Unfortunately there can only be one search argument whereas Magscan uses multiple arguments. However, the speed of the search makes up for this. It is useful to use the search as a look-up for some of the contents of BEEBUG. For instance - *BBGA BB1\ will list all titles contained in the first Volume of BEEBUG.

```
   10 REM Program BBUGData Base
   20 REM Version B1.0
   30 REM for BBC Master Only
   40 REM Author  Ronald Smith
   50 REM BEEBUG  April 1991
   60 REM Program subject to copyright
   70 :
  100 MODE135
  110 DATABBGA,BBGB,BBGC,BBGD,XXXX
  120 osasci=&FFE3:osnwl=&FFE7
  130 wkspace=&70:chars=&71
  140 key=&72:source=&74
  150 thisitem=&76:ch=&78
  160 work=&9F
  170 FOR opt = 4 TO 7 STEP 3
  180 P%=&8000:O%=&4000
  190 [OPT opt
  200 JMP back:JMP entry
  210 EQUB &82:EQUB copy:EQUB 0
  220 .name
  230 EQUS "DBBBGX":EQUB 13
  240 .copy
  250 EQUB 0:EQUS "(C) BEEBUG Magscan Da
tabase"
  260 EQUB 0:EQUB 13
  270 EQUS "Syntax:*BBGX etc"
  280 EQUB 13:EQUS "DB @&8200"
  290 EQUB 13:EQUS "=:":EQUB 0
  300 .done
  310 EQUS "E N D :BBGX":EQUB 13
  320 :
  330 .entry
  340 STA work:LDA &F3
  350 PHA:LDA &F2:PHA
  360 PHY:PHX:LDX  #8
  370 .savewkspace
  380 LDA wkspace,X
  390 PHA:DEX:BPL savewkspace
  400 LDAwork:CMP #3:BEQ help
  410 CMP #9:BEQ help
  420 CMP #4:BEQ star1
  430 .back LDX #0
  440 .restorews
  450 PLA:STA wkspace,X:INX:CPX #9
  460 BNE restorews:PLX:PLY
  470 PLA:STA &F2:PLA
  480 STA &F3:LDA work:RTS
  490 .star1 BRA scom
  500 .print
  510 PHY:STZ ch:LDY #0
  520 .ploop
  530 LDA(thisitem),Y
  540 CMP #44:BEQ nl
  550 CMP #60:BEQ misa:CMP #62:BEQ misb
  560 .p1
  570 JSR osasci:CMP #13:BEQ dun
```

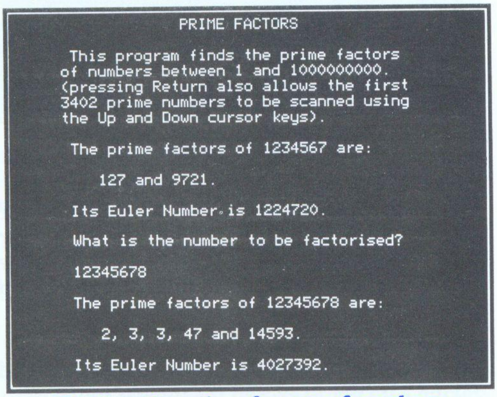
```
 580 CMP #0:BEQ dun
 590 .miss
 600 INC thisitem:BNE ploop
 610 INC thisitem+1:BNE ploop
 620 .dun
 630 INC thisitem
 640 BNE good:INC thisitem+1
 650 .good
 660 LDA thisitem
 670 STA source:LDA thisitem+1
 680 STA source+1:PLY:RTS
 690 .nl LDX ch:BNE nl1:LDA #13
 700 .nl1 JSR osasci:BRA miss
 710 .misa INC ch:BRA miss
 720 .misb STZ ch:BRA miss
 730 .help
 740 CMP #3:BEQ ack
 750 LDA (&F2),Y:CMP #13
 760 BEQ ack:JMP back
 770 .ack
 780 PHA:LDA #(name MOD 256)
 790 STA thisitem
 800 LDA #(name DIV 256)
 810 STA thisitem+1:JSR print
 820 PLA:CMP #3:BEQ nothelp
 830 LDX #6
 840 .more
 850 JSR print:DEX:BNE more
 860 .nothelp JSR osnwl:JMP back
 870 .command EQUS("BBGX")
 880 .scom LDX #0
 890 .comcheck
 900 LDA (&F2),Y:AND #223:CMP command,X
 910 BNE nogood:INX:INY
 920 LDA (&F2),Y:CMP #46
 930 BEQ dot:CPX #4:BEQ yes
 940 BNE comcheck
 950 .nogood JMP back
 960 .dot INY
 970 .yes
 980 .findtxt
 990 LDA (&F2),Y:CMP #32:BNE rest:INY
1000 CLC:BCC findtxt
1010 .rest
1020 CLC:TYA:ADC &F2
1030 STA key:LDA &F3:ADC# 0
1040 STA key+1:LDA #0:STA source
1050 LDA #&82:STA source+1:LDY #255
1060 .findchs
1070 INY:LDA (key),Y
1080 CMP #13:BNE findchs
1090 STY chars
1100 .another
1110 LDA source:STA thisitem
1120 LDA source+1:STA thisitem+1
1130 .compare
1140 LDY #0:LDA (source),Y:CMP #124
1150 BEQ nomore:CMP #97
1160 BMI cha:AND #223
1170 .cha
1180 STAch:LDA(key),Y
1190 CMP #97:BMI chb:AND #223
1200 .chb CMPch:BEQ right
1210 .nxchar
1220 CLC:INC source
1230 BNE nocar:INC source+1
1240 .nocar
1250 LDA (source),Y:CMP #13:BNE compare
1260 INC source:BNE another
1270 INC source+1:BNE another
1280 .right LDY chars:DEY
1290 .check
1300 LDA(source),Y
1310 CMP #97:BMIchc:AND #223
1320 .chc
1330 STA ch:LDA(key),Y
1340 CMP #63:BEQ che:CMP #92
1350 BNE chcc:LDA #32:BRA chd
1360 .chcc CMP #97:BMI chd:AND #223
1370 .chd
1380 CLC:CMP ch:BNE nxchar
1390 CPY #0:BEQ hit
1400 .che DEY:CLC:BCC check
1410 .hit JSR print:JMP compare
1420 .nomore
1430 LDA #(done MOD 256)
1440 STA thisitem:LDA #(done DIV256)
1450 STA thisitem+1:JSR print
1460 JSR osnwl:LDA #0:STA work
1470 JMP back
1480 ]
1490 NEXT
1500 REPEAT:READcurr$
1510 IF curr$="XXXX" THEN1570
1520 C%=ASC(MID$(curr$,4,1))
1530 ?&400E=C%:?&4031=C%
1540 ?&404E=C%:?&40FF=C%
1550 A$="SAVE HD"+curr$+" 4000 +208 800
0 8000"
1560 OSCLI A$
1570 UNTIL curr$="XXXX"
1580 END
```

Recreational Mathematics
Fast Factorising for Fun

In the first of an occasional series Michael Taylor, author of our recent article Continued Fractions, describes a program to find the prime factors of any number up to a billion, and fast.

INTRODUCTION TO RECREATIONAL MATHEMATICS

This is the first of a number of articles, with programs, which will explore what is known as Number Theory. You may have enjoyed the program *Continued Fractions* (BEEBUG Vol.9 No.7) which was on the same theme. Each article and program can be taken on its own or as part of a sequence - as the reader chooses.



```
                    PRIME FACTORS

   This program finds the prime factors
  of numbers between 1 and 1000000000.
  (pressing Return also allows the first
  3402 prime numbers to be scanned using
  the Up and Down cursor keys).

   The prime factors of 1234567 are:

      127 and 9721.

   Its Euler Number is 1224720.

   What is the number to be factorised?

   12345678

   The prime factors of 12345678 are:

      2, 3, 3, 47 and 14593.

   Its Euler Number is 4027392.
```

Finding the prime factors of numbers

On its own, each article is intended to provide mathematical entertainment. However, they will also form a collection which may be useful to students of elementary number theory. There will be just enough explanation for the reader to relate them to formal texts on the subject, but there is no intention to fill the pages of BEEBUG with what is competently explained in textbooks (see the references at the end of the article). The main purpose is just to give pleasure in using the micro to explore the world of numbers.

This month's program quickly finds the prime factors of any number up to 1,000,000,000 (and its Euler Number too).

Future articles currently planned will deal with Euclid's (very ingenious and quick) algorithm for finding the GCD and LCM of two numbers. It will do this much more quickly than can be done by selecting factors with this month's program. There will also be a short program to display addition, multiplication and power tables for modular (finite) arithmetics.

Another article centres round a program for doing modular arithmetic with relatively large moduli (up to 1,000,000,000). For example, after about 20 seconds it will show that:

```
123456789^999999936
```

is congruent to 1 (mod 999999937) (illustrating Fermat's 'little' theorem).

This will lead the way to a further article which will include a simulation of the method of encryption of Rivest, Shamir and Adleman. In a recent article on Cryptology (BEEBUG Vol.9 No.8), Bernard Hill outlined several ciphers including the RSA one. As he explained, the RSA cipher is a remarkable invention of the last decade which revolutionises ciphers by allowing the encoding algorithm to be made public while not even the world's fastest computers can find the deciphering algorithm. It all depends on the difficulty of factorising very large prime numbers. It will be possible to explore more fully the mathematics that makes the RSA cipher possible and also to offer a program that simulates it.

FAST FACTORISING

Fast factorising? On a BBC micro? For most of the numbers up to 1,000,000,000 this program only takes a second or two to find the prime factors. The slowest case is that of the number 999,999,937 - which (Wells, p.190) is the 50,847,534th prime number. The program takes just over a quarter of a minute on a model B to report that 999,999,937 is prime.

An early warning to the impatient: the program will not run after typing it in. On the first occasion it is necessary to generate a list of the first 3402 prime numbers by calling the procedure PROCGenerate and to save this list as the file STORE2.

There is something fascinating about prime numbers as part of the 'fundamental structure' of numbers. Any natural number from 2 onwards can be represented as the product of prime numbers (unless it is itself prime) and this can be done in only one way.

Prime numbers have recently gained a practical importance in the new 'trap-door' ciphers such as the RSA cipher mentioned above. They are also of central importance in number theory. This program can provide prime numbers for testing and illustrating the theory of whole numbers, and - as we shall see - for use in the RSA method of encryption.

The program listed here is easily typed in and saved, but before running it type:

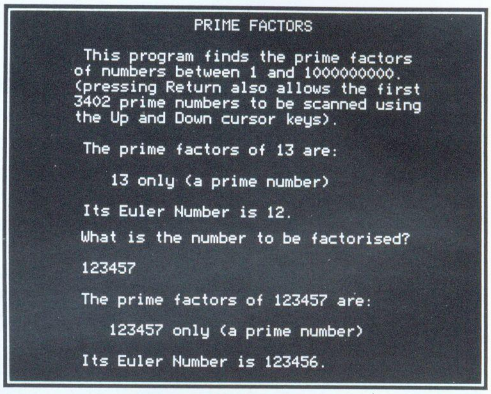
```
        PROCGenerate
```

and a two-column list will appear. The left-hand column contains the numbers from 1 to 3402, and opposite each number is the corresponding prime number. This procedure takes about 16 minutes to run on a model B. After a cup of coffee come back to press Return so that the list of primes is stored as STORE2 for the program's future use.

The main program, *Euler*, can now be run. Provided STORE2 is available, it will *LOAD it at &3000 and then ask for numbers to be factorised. Positive integers can be keyed in directly or as expressions like 2^19-1 (which happens to be a prime).

As mentioned above, it takes longest to show that 999,999,937 is prime. It takes almost as long to factorise a number which is the square of a prime and which is just less than 1,000,000,000 - the biggest one is 31607^2 = 999,002,449.

Each natural number has an 'Euler Number' which is important in number theory and in RSA encryption. It is the number of numbers less than and relatively prime to the given number. It is easily found once the prime factors of a number are known and so it is offered here - though the reader can always ignore it.



```
                 PRIME FACTORS

   This program finds the prime factors
  of numbers between 1 and 1000000000.
  (pressing Return also allows the first
  3402 prime numbers to be scanned using
  the Up and Down cursor keys).

   The prime factors of 13 are:

      13 only (a prime number)

   Its Euler Number is 12.

   What is the number to be factorised?

   123457

   The prime factors of 123457 are:

      123457 only (a prime number)

   Its Euler Number is 123456.
```

Locating prime numbers

For a prime p the Euler Number is just p-1. In the case of a composite number N with (non-repeating) prime factors p, q, r ... (so that N = p^a*q^b*r^c...), the Euler number can be found as:

```
        N*(1-1/p)*(1-1/q)*)1-1/r)
```

or, with the integer variables used here and taking care not to exceed the integer range, as:

```
        N/p*(p-1)/q*(q-1)/r*(r-1)...
```

Since the program has to make use of the first 3402 prime numbers, advantage is taken of their presence. They can be inspected with PROCScan. If this is called (in immediate mode if you need) further instructions are displayed on-screen. Once in use, pressing Return will exit from the procedure.

There are possible extensions. The program could be modified to search for prime numbers in a specified range or with chosen properties. The range could be extended to the maximum of 2^31-1 set by the BBC micro's 4 byte integers (PROCGenerate would have to be extended too, at least to include the last prime before SQR(2^31-1)=46340 which is 46337). If more storage space is needed, it is possible to modify the program to store the primes of STORE2 as two-byte integers - though it would take a little longer for the program to make use of them. PROCGenerate could stand alone or be incorporated into another program.

PROGRAM OUTLINE

To test for all the factors of a number it is only necessary to divide it by prime numbers, from 2 onwards - and only necessary to do so until the next prime number would be bigger than the number's square root. That is why it is only necessary to use the first 3401 primes (up to 31607) for factorisation up to 1,000,000,000 since the next prime, 31627, has a square of 1,000,267,129. 31627 is also included in the list of primes for ease of programming so that it can be detected as being greater than SQR(1,000,000,000).

The procedure PROCGenerate is only run on the first occasion to produce the file STORE2. It 'bootstraps' itself by storing primes as they are found and then using them to test for further primes. Of course, it only makes use of the ones it has found which are not more than SQR(31627).

When the main program is run it *LOADs STORE2 at &3000. PROCFactorise then takes the primes in turn, from 2 upwards - as far as 31627 if need be. If it finds one of them is a factor of the number, it prints it out, divides the number by it and then continues to try to divide the left-over factor by the same prime factor, or if that fails by higher prime factors. Once again, when a prime factor is found, it is divided into the left-over factor to leave an even smaller one. Once the next prime to be tested exceeds the square root of the left-over factor, then that remaining factor, too, must be prime.

Some useful references on number theory and the RSA cipher are:

1. *The Higher Arithmetic*, by H.Davenport (1952), Cambridge University Press (a classic in the field and used in the earlier program on *Continued Fractions* in BEEBUG Vol.9, No.7.)

2. *Number Theory* and Its History, by Oystein Ore (1948), Dover Publications Inc. (very readable).

3. *Think of a Number*, by Malcolm E. Lines (1990), Adam Hilger (discusses modular arithmetic, the RSA system and much else of entertainment).

4. *Numbers, Groups and Codes*, by J.F.Humphreys and M.Y.Post (1989), Cambridge University Press (this includes an account of the RSA cipher system. It would also make excellent reading for a 6th former thinking of mathematics or computing at university).

5. There is an article on two trap-door ciphers, one of them the RSA one: *The Mathematics of Public-Key Cryptography*, by Martin E. Hellman, in Scientific American, Vol.241, No.2, August 1979, pp.130 to 139 (the usual lucidity of a Scientific American article).

6. *Cryptology* by Bernard Hill, in BEEBUG (January/February 1991) Vol.9 No.8. (introduces both traditional ciphers and the new RSA system).

7. *The Penguin Dictionary of Curious and Interesting Numbers*, by David Wells (1986), Penguin Books (fun to browse through).

```
   10 REM Program Euler
   20 REM Version B1.0
   30 REM Author  Michael Taylor
   40 REM BEEBUG  April 1991
   50 REM Program subject to copyright
   60 REM Execute PROCGenerate in immedi
ate before running this program.
   70 :
  100 ON ERROR PROCError:END
  110 HIMEM=&3000:P%=HIMEM:*FX4,0
  120 MODE7:REM Only Mode 7 possible wit
hout sideways RAM.
  130 DIM F%(32): DIM G%(32)
  140 ON ERROR PROCFileError:END
  150 *LOAD STORE2 3000
  160 REM: The previous line should have
 loaded 'STORE2' which has the first 340
2 primes, from 2 to 31627.
  170 ON ERROR PROCError:END
  180 CLS:PRINTSPC13;"PRIME FACTORS"
  190 PRINT'" This program finds the pri
```

```
me factors"'"of numbers between 1 and 10
00000000."'"(pressing Return also allows
 the first"'"3402 prime numbers to be sc
anned using"'"the Up and Down cursor key
s)."
  200 VDU28,0,24,39,8
  210 REPEAT
  220 PROCInput
  230 PROCFactorise
  240 REM PROCGenerate is not called by
the main program.
  250 UNTIL FALSE
  260 END
  270 :
 1000 DEF PROCInput
 1010 D%=1:F%=0:G%=0:Old%=0
 1020 REPEAT
 1030 REPEAT
 1040 PRINT''" What is the number to be
factorised?"'
 1050 VDU10,10,11,11
 1060 INPUT" "X$:X%=FNTest(X$,1000000000
)
 1070 IFX%=-1 PROCScan:*FX4,0
 1080 UNTIL X%<>-1
 1090 IF X%=0 PROCClear:VDU 11,11,11,11
 1100 UNTIL X%<>0
 1110 ENDPROC
 1120 :
 1130 DEF PROCFactorise
 1140 IF X%=1 PRINT'" Number 1 has no pr
oper factors, and is"'" not usually cons
idered as prime."'" Its Euler number is
 defined to be 1.":ENDPROC
 1150 IF X%=2 PRINT'" Number 2 is the fi
rst prime number and"'" the only even on
e."'" Its Euler Number is 1.":ENDPROC
 1160 C%=P%:Y%=X%:R%=INT(SQR(X%))
 1170 PRINT'" The prime factors of ";X%"
 are: "''SPC4;
 1180 REPEAT:REM The next line limits sp
eed
 1190 REPEAT:C%=C%+4:D%=!C%:UNTIL Y%MODD
%=0 OR D%>R%
 1200 IF D%<=R%PROCFound
 1210 UNTIL D%>R%
 1220 IF D%>R% F%=F%+1:F%(F%)=Y%:IF D%<>
Y% G%=G%+1:G%(G%)=Y%
 1230 IF Y%<>X% VDU8,8:PRINT" and ";STR$
(Y%);"." ELSE PRINT;STR$(Y%);" only (a p
rime number)"
 1240 E%=X%
 1250 FOR Z%=1 TO G%
 1260 E%=E%/G%(Z%):E%=E%*(G%(Z%)-1)
 1270 NEXT
 1280 PRINT'" Its Euler Number is ";E%".
"
 1290 ENDPROC
 1300 :
 1310 DEF PROCFound
 1320 C%=C%-4:PRINT STR$(D%);", ";
 1330 IF POS>35 VDU13:PRINT'SPC4;
 1340 F%=F%+1:F%(F%)=D%:REM Array of fac
tors with repeats in case it is useful
 1350 IF D%<>Old% Old%=D%:G%=G%+1:G%(G%)
=D%:REM Array of factors without repeats
 for finding the Euler Number.
 1360 IF Y%<>D% Y%=Y%/D%:R%=INT(SQR(Y%))
 1370 ENDPROC
 1380 :
 1390 DEF PROCScan
 1400 PRINT'" Use the Up and Down cursor
 keys to"'" scan prime numbers from 2 to
 31627."'" Use Shift and the cursor keys
 for the"'" beginning or ending of the l
ist."''" Use Return for main program aga
in."
 1410 *FX4,1
 1420 PRINT
 1430 FOR Z%=1 TO 17
 1440 M%=P%+4*Z%:PRINTTAB(8)!M%
 1450 NEXT
 1460 REPEAT
 1470 H%=FALSE:K%=GET
 1480 IF K%=138 AND INKEY-1 M%=P%+&3528:
CLS:PRINTTAB(8,24):H%=TRUE
 1490 IF K%=139 AND INKEY-1 M%=P%+4:CLS:
H%=TRUE
 1500 IF K%=138 AND M%<P%+&3528 M%=M%+4:
H%=TRUE
 1510 IF K%=139 AND M%>P%+4 VDU11,11:M%=
M%-4:H%=TRUE
 1520 IF H%=TRUE PRINTTAB(8)!M%;
 1530 *FX21,0
 1540 UNTIL K%=&0D:CLS
 1550 ENDPROC
 1560 :
 1570 DEF PROCClear:VDU11,13:PRINTSPC79:
```

Continued on page 25

New Products From Essential

Bernard Hill assesses the latest products for 512 users from Essential Software.

Product	**Fastboot**
Price	**£10.95 on disc**
	£14.95 on EPROM
(upgrade cost £4.00, return your disc)	

Product	**CPFS**
Price	**£24.95 on EPROM with disc**
Supplier	**Essential Software**
	PO Box 5,
	Groby, Leicester LE6 0ZB.

Essential Software, the software house run by Robin Burton, author of BEEBUG's 512 Forum, has recently released two important software packages for 512 co-processor owners. These are *Fastboot* and *CPFS* which are described separately below.

FASTBOOT

It always seems that a 512 system takes an eternity to boot up DOS as it reads from the ADFS-formatted boot disc. The procedure involves loading a number of files including the XIOS (the equivalent of the PC's BIOS), DOS itself (DOSPLUS.SYS), the code which resides in the 6502 system (6502.SYS), to say nothing of AUTOEXEC.BAT and any commands you've put in there. Fastboot is a very simple idea which keeps some of this code in sideways RAM (or ROM) ready for instant use.

The package consists of just a disc containing a ROM image of the system. In common with all Essential Software packages the documentation is in printable form on disc. For my money this is the best way: after initial perusal I tend to put manuals away in the cupboard and rarely refer to them. Not having to bother with printing and distribution costs enables Essential to keep their prices low and their value high, probably chopping about £10 off the cost of the package.

Once you have loaded your RAM image (or blown an EPROM - Essential give you permission to make one 8K EPROM for your own use), you can create an 800K format disc with DOSPLUS.SYS on it, and Fastboot will use that to boot from. This combination of using ROM and the fastest disc format speeds up the boot process quite considerably. On my model B (without an AUTOEXEC.BAT) it reduced the boot time from 31 seconds to 14. With an AUTOEXEC the improvement will be greater because of the faster disc access for each operation in your AUTOEXEC.

Whether it's worth it to you to pay £11 for a saving of just a few seconds must depend on how often you boot up and how much spare cash you have!

CPFS

This is a much more exciting product which has been overdue for some time. The existence of a half-megabyte of RAM (in my 512) sitting alongside my Beeb and usually switched off has always struck me as a complete waste. Essential Software's Co-Processor Filing System turns this 512K into a complete functioning RAM disc for use with your model B, B+ or Master when working in BBC mode. It comes supplied this time on a 16K EPROM and (again) with the 24-page manual on disc ready for printing or browsing through on screen.

It's important of course after installing the chip (and it must go above the other filing systems on a model B) to be able to start your machine with the Tube off. On a Master you can *CONFIGURE NOTUBE but on the model B the chip provides the equivalent function *NOTUBE to be followed by a Break. Then the filing system can be started with the command *CPFS.

The filing system allow up to 5 files open at a time (as do the DFS and ADFS) and raises the value of PAGE by &100. It allows file names up to 10 characters in length and has a directory structure similar to DFS's, i.e. one-character directories followed by a full stop. There is room for 127 files in the CPFS totalling 503,808 bytes and the lack of a complete hierarchical filing system like ADFS should be no loss in what is after all a temporary system as it loses the files when you power off.

How to review a filing system? There is really no great excitement to what you see, the benefit comes in the speed of access and capacity of the system. The facilities available are very much like those found in the DFS. The commands with an identical purpose and syntax are:

```
*ACCESS       *BUILD       *CAT        *DELETE
*DESTROY      *DIR         *DUMP       *INFO
*LIB          *LIST        *OPT        *PRINT
*RENAME       *TYPE        *WIPE
```

There is also *MTYPE to perform *TYPE in Master-style, i.e. with an ASCII display of control characters.

Some commands are borrowed from the ADFS:

```
*EX           *FREE
```

However, *COMPACT has been modified. Since CPFS performs automatic compaction when required by the system (no more "Can't Extend"s) this command merely forces early compaction when issued. It is also possible to force a file to be last on the disc for faster extension, with:

```
        *COMPACT <filename>
```

Some commands are recognised by CPFS but do nothing because they are not needed. These are:

```
*BACK         *CDIR        *COPY       *DISMOUNT
*DRIVE        *ENABLE      *MAP        *MOUNT
*TITLE
```

Commands omitted are:

```
*BACKUP       *CLOSE
```

I suspect the latter is an omission easily rectified but I wish the former had been included (see below).

The CPFS will also work as a temporary filing system with the Master (and Master Emulation ROM for the model B) with all the advantages that gives.

There is one new command in the system: *TRANSFER, which will copy files or groups of files between filing systems. It requires answers to five questions: the target and source filing system, the target and source directory and the file specification. Wildcards are allowed (although '*' as a directory caused confusion). It can even transfer files between DFS and ADFS when CPFS is not active!

Its most common use must be, of course, the transfer of a complete (A)DFS disc to CPFS at the start of a session, and the transfer back at the end. Beware, however, that CPFS holds more than two complete DFS discs, or more than one ADFS disc if you have a 512K expansion fitted to your 512! Also, I found *TRANSFER rather slow, presumably because it uses byte access.

It was also annoying that having transferred a program to CPFS and modified it, it caused a "Can't Extend" message from the DFS as it tried to overwrite its shorter version on disc. *TRANSFER also won't replace locked files on the target filing system, though I can understand the reason for this, but the obvious way to use this product is with a fast transfer of all files from DFS to CPFS at the start of a session, and an easy transfer of files back at the end. I am finding I have to use a second blank floppy disc for the target at the end of a work period.

The product works very well and fills a much-needed gap if you find floppy discs too slow or too small. In operation it behaves perfectly and gives a lightning response - saving a 64K file with:

```
        *SAVE TEST 0 10000
```

took 8 seconds on a DFS floppy disc, 0.01 seconds on CPFS!). I have already started to use it for all development work as it is blindingly fast and I can forget the dreaded "Can't Extend" message. It's also ideal for a time-critical data-logging application I have where the floppy disc's slow BPUT makes for much difficulty.

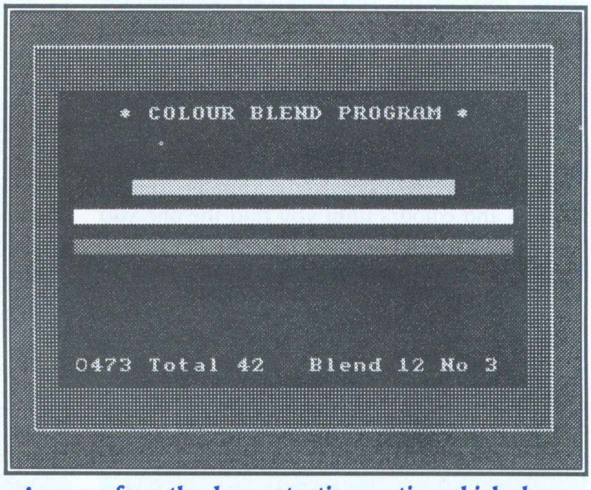
I also like the 'O' flag alongside the 'Locked' flag in catalogues to indicate files which are open and I guess it's a good practice anyway to make me use a blank floppy disc to save onto at the end of my sessions.

A must for every 512 owner who uses his Beeb as a Beeb, and £25 very well spent.

Colour Blender and Screen Design Aid

by Peter Reis

This program was originally written as an aid to blending colours for the annual Trinidad Carnival festival. A further development allows the user to design unique screens which can be used to introduce other programs. An extensive and fascinating array of colour combinations and effects is available to the adventurous. The program works by plotting alternate pixels from two or more of the standard colours to create up to 420 new colours.



```
* COLOUR BLEND PROGRAM *

0473 Total 42    Blend 12 No 3
```

A screen from the demonstration routine which shows 4 standard and 3 blended colours

USING THE PROGRAM

Type in the listing and save it as *Blender*. On the model B, PAGE must be set to &1200 for the program to operate. When run, the program starts off with a demonstration of the colours available. You can cycle through all the various combinations either manually, or automatically with the colours changing approximately every 3 seconds. An initial screen prompts you for a choice between these two modes, and also asks whether you want 6 or 7 colours on the screen at the same time.

The demonstration itself displays four of the standard colours in the centre of the screen, in the form of horizontal bars. The top and bottom bars are then mixed to form a Base screen around the edge of the display. All four colours are mixed to produce a smaller rectangle inside this called the Mid screen, and if you have chosen the 7 colour mode, the two lower bars are mixed to produce a Fore screen inside this. The standard colours are then altered one at a time, cycling through all the combinations of the 7 non-flashing colours available in the standard palette. At the bottom left of the centre screen a four-digit number shows you which of these colours are currently being used.

To quit the demonstration, you can either wait for it to finish its natural course, or press Escape at any point. You now have four options. Press Ctrl-Q to quit the program, Space to run the demonstration again, Tab to input a code which will display a screen in the colours of your choice, and S to alter the co-ordinates of the Mid or Fore screens. These latter two options will now be explained.

CREATING SCREENS

Pressing Tab displays an input screen with a prompt to enter a colour/screen code. This should contain 7 characters, the first 4 of which are the colour codes as shown at the bottom of the demonstration sequence screens, followed by 3 characters which should be either 'Y' or 'N' to select or deselect the Back, Mid and Fore screens respectively. For example:

> 1027YYY would use the colours 1 (red), 0 (black), 2 (green) and 7 (white), and would create all three screens.
>
> 1027YNY would use the same colours but would not create the Mid screen.

The screen chosen will now be displayed, whereupon pressing Space will show you the actual programming lines needed to add the display to your own program.

Pressing S from the mini-menu described above allows you to alter the co-ordinates of the Mid and/or Fore screens. The default values are shown, and you may type in new values. The values used for the Mid screen are standard OS graphics co-ordinates, and are given in the usual order, i.e. x-min, y-min, x-max, y-max. For the Fore screen, however, you must specify text character positions in the same order, not graphics co-ordinates.

You can now try out various combinations of screens and colours until you find the one you need. Note that in extreme cases of screen size selection, the text which prompts "Space bar to continue" may be somewhat displaced but the screens represented should not be affected.

```
   10 REM         >Blender
   20 REM Program Colour blender
   30 REM Version B1.0
   40 REM Author  Peter Reis
   50 REM BEEBUG  April 1991
   60 REM Program subject to copyright
   70 :
  100 ON ERROR PROCerr
  110 VDU23,250,170,85,170,85,170,85,170
,85:inpt=0:inpg=0
  120 MODE1:PROCsetup:CLS:PROCinput
  130 PROCinit:PROCstart:PROCendscr:END
  140 :
 1000 DEF PROCblend:IF N>1 ENDPROC
 1010 VDU26,12:?&D0=2:FOR V=1 TO 10:PRIN
TSTRING$(128,CHR$B1);:NEXT:?&D0=0
 1020 base=base+1
 1030 ENDPROC
 1040 :
 1050 DEF PROCbias:VDU19,0,c0;0;19,1,c1;
0;19,2,c2;0;19,3,c3;0;:ENDPROC
 1060 :
 1070 DEF PROCendscr:VDU30:a=0:b=1:c=3:d
=7
 1080 VDU19,0,a;0;19,1,b;0;19,2,c;0;19,3
,d;0;:w=7:w1=3:w2=1:w3=0
 1090 PROCinsert:c0=a:c1=b:c2=c:c3=d:PRO
Cann:G=GET
 1100 ENDPROC
 1110 :
 1120 DEF PROCann:PROCcontrast
 1130 COLOUR 128:COLOUR k:PRINTTAB(1,18)
;c0;c1;c2;c3;
 1140 PRINTTAB(5,18);" Total ";T;" "
 1150 PRINTTAB(26,18);"No ";base;:VDU23,
1,0;0;0;0;
 1160 PRINTTAB(17,18);"Blend";" ";v%;" "
 1170 ENDPROC
 1180 :
 1190 DEF PROCinsert:VDU24,75;100;1205;9
30;:PROCcol
 1200 IF N=1 VDU28,4,25,35,6:IF cls=0 CL
S:PROCforecol
 1210 MOVE 75,100:GCOL 0,3:DRAW 1205,100
:DRAW 1205,930:DRAW 75,930:DRAW 75,100
 1220 COLOUR 3:PRINTTAB(4,1);"* COLOUR B
LEND PROGRAM *"
 1230 COLOUR 131:COLOUR w3:PRINTTAB(5,6)
;" Developed By P.Reis. "
 1240 COLOUR 130:COLOUR w2:PRINTTAB(1,8)
;"Allows 28 Hybrid base colours."
 1250 COLOUR 129:COLOUR w1:PRINTTAB(1,10
);"  420  Colour combinations !  "
 1260 COLOUR 128:COLOUR w:PRINTTAB(5,12)
;" ";n%;" colours  in mode 1 "
 1270 ENDPROC
 1280 :
 1290 DEF PROCcol:IF N>1 ENDPROC
 1300 GCOL 26,cl:CLG:GCOL 27,cl:CLG:GCOL
 28,cl:CLG:GCOL 26,cl:CLG
 1310 ENDPROC
 1320 :
 1330 DEF PROCinp
 1340 S%=1:B1=250:cl=129:N=1:PROCplotsc
 1350 COLOUR 3:PRINTTAB(2,5);" Colour/Sc
reen code ->";:PROCcls
 1360 COLOUR 2:INPUT" "CS$:COLOUR 3:IF L
ENCS$<>7 PROCerror:GOTO1340
 1370 IF VAL(LEFT$(CS$,4))<1 OR VAL(RIGH
T$(CS$,3))>0 PROCerror:GOTO1340
 1380 c0=VAL(MID$(CS$,1,1)):c1=VAL(MID$(
CS$,2,1))
 1390 c2=VAL(MID$(CS$,3,1)):c3=VAL(MID$(
CS$,4,1))
 1400 IF MID$(CS$,5,1)="Y" OR MID$(CS$,5
,1)="y" S1=1 ELSE S1=0
 1410 IF MID$(CS$,6,1)="Y" OR MID$(CS$,6
,1)="y" S2=1 ELSE S2=0
 1420 IF MID$(CS$,7,1)="Y" OR MID$(CS$,7
,1)="y" S3=1 ELSE S3=0
 1430 ENDPROC
 1440 :
 1450 DEF PROCscr:VDU26
 1460 VDU19,0,c0;0;19,1,c1;0;19,2,c2;0;1
9,3,c3;0;:PROCoff:CLS
 1470 IF inpg=1 a=a%:b=b%:c=c%:d=d% ELSE
 a=75:b=100:c=1205:d=930
 1480 IF inpt=1 p=p%:q=q%:r=r%:s=s% ELSE
 p=4:q=25:r=35:s=6
 1490 IF S1=1 N=1:PROCblend:N=0
 1500 IF S2=1 N=1:VDU24,a;b;c;d;:PROCcol
:N=0
 1510 IF S3=1 N=1:VDU28,p,q,r,s:CLS:PROC
forecol2:N=0
 1520 IF S2 MOVE a,b:GCOL 0,3:DRAW c,b:D
RAW c,d:DRAW a,d:DRAW a,b
 1530 MOVE (a+205),(b+150):VDU5:GCOL 0,2
 1540 PRINT" Spacebar to continue ":VDU4
:PROCoff
 1550 COLOUR 3:PRINTTAB(5,5);c0;c1;c2;c3
:inpt=0:inpg=0
 1560 REPEAT UNTIL GET=32
 1570 PROCdata
 1580 ENDPROC
 1590 :
 1600 DEF PROCfind
 1610 IF c0 B$=LEFT$(A$,c0)+MID$(A$,(c0+
2),(c3-(c0+1)))+MID$(A$,(c3+2),(8-c3+2))
 ELSE B$=MID$(A$,(c0+2),(c3+2-(c0+3)))+M
ID$(A$,(c3+2),(8-(c3+1)))
 1620 ENDPROC
 1630 :
 1640 DEF PROCvariations
 1650 FOR x%=1 TO 5:c1=VAL(MID$(B$,x%,1)
)
 1660 FOR y%=x%+1 TO 6:c2=VAL(MID$(B$,y%
,1)):N=2:T=T+1:v%=v%+1
 1670 PROCbias:PROCinsert:PROCann
 1680 IF aut% REPEAT UNTIL GET=32 ELSE Z
=INKEY(250)
 1690 NEXT y%:NEXT x%:base=base+1:v%=0
 1700 ENDPROC
 1710 :
 1720 DEF PROCstart
 1730 FOR z%=num TO 7:c0=VAL(MID$(A$,z%,
1))
 1740 FOR j%=z%+1 TO 8:c3=VAL(MID$(A$,j%
,1)):N=N+1
 1750 PROCblend:PROCinsert:PROCfind:PROC
variations
 1760 NEXT j%:NEXT z%
 1770 ENDPROC
 1780 :
 1790 DEF PROCinit
 1800 cl=129:B1=250:A$="01234567":v%=0:b
ase=0:T=0:N=0:num=1
 1810 w=0:w1=1:w2=2:w3=3:VDU23,1,0;0;0;0
;
 1820 ENDPROC
 1830 :
 1840 DEF PROCcontrast
 1850 k=3:IF c0=3 AND c3=7 OR c0=2 AND c
3=6 OR c0=1 AND c3=5 OR c0=6 AND c3=7 k=
k-2
 1860 IF c0=0 AND c3=4 OR c0=2 AND c3=3
k=k-2
 1870 ENDPROC
 1880 :
 1890 DEF PROCinput:VDU23,1,0;0;0;0;
 1900 cl=129:B1=250:N=1:PROCblend:PROCin
troscr:VDU23,1,0;0;0;0;
 1910 REPEAT UNTIL GET=32
 1920 VDU20:ENDPROC
 1930 :
 1940 DEF PROCintroscr
 1950 VDU19,0,0;0;19,1,1;0;19,2,2;0;19,3
,7;0;:w=7:w1=2:w2=1:w3=0:PROCinsert
 1960 MOVE 280,250:VDU5:GCOL 0,2:PRINT"
Spacebar to continue ":VDU4
 1970 ENDPROC
 1980 :
 1990 DEF PROCsetup:VDU23,1,0;0;0;0;
 2000 VDU19,0,4;0;19,1,0;0;19,2,1;0;19,3
,7;0;
 2010 VDU26,12:?&D0=2:FOR V=1 TO 10:PRIN
TSTRING$(128,CHR$250);:NEXT:?&D0=0
 2020 VDU24,55;70;1230;960;
 2030 GCOL 26,129:CLG:GCOL 27,129:CLG:GC
OL 28,129:CLG:GCOL 26,129:CLG
 2040 MOVE 55,70:GCOL 0,3:DRAW 1230,70:D
RAW 1230,960:DRAW 55,960:DRAW 55,70
 2050 VDU28,3,27,36,4:CLS
 2060 COLOUR 1:?&D0=2:FOR X1=1 TO 8:PRIN
T STRING$(102,CHR$250);:NEXT:?&D0=0
 2070 PROCsetuptext
 2080 ENDPROC
 2090 :
 2100 DEF PROCforecol
 2110 COLOUR 1:?&D0=2:FOR X1=1 TO 10:PRI
NT STRING$(64,CHR$B1);:NEXT:?&D0=0
 2120 ENDPROC
 2130 :
 2140 DEF PROCerr:IF ERR<>17 CLS:REPORT:
PRINT" at line ";ERL:END
 2150 VDU20,22,3,14
 2160 REPEAT PROCselect:G=GET
 2170 IF G=9 VDU20,22,1,14:PROCinp:PROCs
cr
 2180 IF G=32 RUN
 2190 IF G=83 VDU22,3:PROCscreenchg:IF G
<>113 VDU22,1:PROCinp:PROCscr
 2200 UNTIL G=17
 2210 CLS:VDU22,3,14:PROCoff:PRINTTAB(36
,15);"E X I T":Z=INKEY(100):CLS:VDU23,1,
1;0;0;0;:END
 2220 ENDPROC
 2230 :
 2240 DEF PROCdata:VDU26,22,3:CLS:VDU23,
1,0;0;0;0;
 2250 PRINT'"    MODE1:VDU23,250,170,85,1
70,85,170,85,170,85"
 2260 PRINT"    VDU19,0,";c0";0;19,1,";c1
";0;19,2,";c2";0;19,3,";c3";0;"
 2270 IF S1=1 PRINT'"Base Screen:"
 2280 IF S1=1 PRINT"    VDU26,12:?&D0=2:F
OR V=1 TO 10:PRINTSTRING$(128,CHR$250);:
NEXT:?&D0=0"
 2290 IF S2=1 PRINT'"Mid  Screen:"
 2300 IF S2=1 PRINT"    VDU24,";a";";b";"
;c";";d";"
 2310 IF S2=1 PRINT"    GCOL 26,129:CLG:G
COL 27,129:CLG:GCOL 28,129:CLG:GCOL 26,1
29:CLG"
 2320 IF S2=1 PRINT"    MOVE";a",";b":GCO
L 0,3:DRAW";c",";b":DRAW";c",";d":DRAW";a"
,";d":DRAW";a",";b
 2330 IF S3=1 PRINT'"Fore Screen:"
 2340 IF S3=1 PRINT"    VDU28,";p",";q","
;r",";s
 2350 IF S3=1 PRINT"    COLOUR 1:?&D0=2:F
OR X1=1 TO ";B%;:PRINT":PRINT STRING$(";
A%;",CHR$250);:NEXT:?&D0=0"
 2360 PRINT'"Text"
 2370 PRINT"    MOVE 280,250:VDU5:GCOL 0,
2:PRINT' Spacebar to continue ':VDU4"
 2380 PRINT"    COLOUR 128:COLOUR 1:PRINT
TAB(18,22);c0;c1;c2;c3"
 2390 S1=0:S2=0:S3=0
 2400 ENDPROC
 2410 :
 2420 DEF PROCselect
 2430 PRINT'TAB(3);:COLOUR 131:COLOUR 0:
PRINT" PRESS CTRL (Q) TO QUIT ";:VDU20
 2440 PRINTTAB(33);:COLOUR 131:COLOUR 0:
PRINT" PRESS SPACE BAR TO RUN ":VDU20
 2450 PRINTTAB(3);:COLOUR 131:COLOUR 0:P
RINT" PRESS <TAB>  NEW INPUT ";:VDU20
 2460 PRINTTAB(33);:COLOUR 131:COLOUR 0:
PRINT" PRESS (S) ALTER SCREEN ":VDU20
 2470 COLOUR 128:COLOUR 131:ENDPROC
 2480 :
 2490 DEF PROCerror
 2500 CLS:PROCoff:PRINTTAB(11,12);"INPUT
 SELECTION ERROR !":Z=INKEY(150):CLS
 2510 PROCplotsc:ENDPROC
 2520 :
 2530 DEF PROCplotsc:c0=0:c1=1:c2=3:c3=7
:a=75:b=350:c=1205:d=675:p=2:q=21:r=37
 2540 VDU19,0,c0;0;19,1,c1;0;19,2,c2;0;1
9,3,c3;0;:s=10
 2550 VDU28,p,q,r,s:CLS:PROCforecol
 2560 MOVE a,b:GCOL 0,3:DRAW c,b:DRAW c,
d:DRAW a,d:DRAW a,b
 2570 MOVE (a+105),(b-100)::VDU5:GCOL 0,
2:PRINT" Enter Code  --  Press <RET>":VD
U4:PROCoff:ENDPROC
 2580 :
 2590 DEF PROCcls
 2600 FOR x%=24 TO 32:PRINTTAB(x%,5);" "
;:NEXT:PRINT"<":FOR y%=1 TO 12:VDU8:NEXT
 2610 ENDPROC
 2620 :
 2630 DEF PROCforecol2
 2640 ax=(39-p)-(39-(r+1)):ay=(31-s)-(31
-(q+1)):k1=ax*ay
 2650 X=10:REPEAT A=k1 MOD X
 2660 IF A=0 A%=k1 DIV X:B%=X ELSE B%=ay
:A%=ax
 2670 X=X-1:UNTIL A=0 OR X=5
 2680 COLOUR 1:?&D0=2:FOR X1=1 TO B%:PRI
NTSTRING$(A%,CHR$250);:NEXT:?&D0=0
 2690 ENDPROC
 2700 :
 2710 DEF PROCscreenchg:inpt=0:inpg=0
 2720 REPEAT CLS:PRINTTAB(19);"WHICH SCR
EEN/S DO YOU WISH TO ALTER ?"
 2730 PRINT'"PRESS (M) FOR MIDSCREEN";TA
B(31);"Default -->  75,100,1205,930"
 2740 PRINT"PRESS (F) FOR FORESCREEN";TA
B(31);"Default -->  4,25,35,6"
 2750 PRINT"PRESS (B) FOR BOTH"
 2760 PRINT"PRESS (Q) TO QUIT OPERATION"
;
 2770 G=GET OR 32:UNTIL G=98 OR G=102 OR
 G=109 OR G=113
 2780 PRINT"    (";CHR$G;")"
 2790 IF G=98 OR G=102 INPUT'TAB(19);"FO
RESCREEN  (a,b,c,d) >  "p%,q%,r%,s%:inpt
=1
 2800 IF G=98 OR G=109 INPUT'TAB(19);"MI
DSCREEN   (a,b,c,d) >  "a%,b%,c%,d%:inpg
=1
 2810 IF G<>113 PROCtab
 2820 ENDPROC
 2830 :
 2840 DEF PROCsetuptext
 2850 MOVE81,843:VDU5:GCOL 0,1:PRINT"  -
 *  COLOUR BLEND PROGRAM  * -  "
 2860 MOVE90,850:VDU5:GCOL 0,3:PRINT"  -
 *  COLOUR BLEND PROGRAM  * -  "
 2870 MOVE93,853:VDU5:GCOL 0,2:PRINT"  -
 *  COLOUR BLEND PROGRAM  * -  "
 2880 MOVE 220,700:VDU5:GCOL 0,3:PRINT"S
elect mode of operation :"
 2890 MOVE 320,630:GCOL 0,2:PRINT"Manual
 (1) >  "
 2900 MOVE 320,580:GCOL 0,2:PRINT"Auto
 (0) >  "
 2910 MOVE 220,415:GCOL 0,3:PRINT"Select
 colour blend mode :"
 2920 MOVE 320,345:GCOL 0,2:PRINT"(6) or
 (7) >  ":VDU4
 2930 COLOUR 130:COLOUR 3:PRINTTAB(25,10
)" <":VDU8
 2940 VDU31,25,10:COLOUR 3:INPUT" "aut%
 2950 PRINTTAB(25,17)" <":VDU8
 2960 VDU31,25,17:INPUT" "cls
 2970 IF cls=6 cls=1:n%=6:ELSE cls=0:n%=
7
 2980 IF aut%<0 OR aut%>1 aut%=0
 2990 VDU20,26,12:ENDPROC
 3000 :
 3010 DEF PROCtab:PRINTTAB(35,20);"PRESS
 TAB":ENDPROC
 3020 :
 3030 DEF PROCoff:VDU23,1,0;0;0;0;:ENDPR
OC
```

Recreational Mathematics (continued from page 18)

```
VDU11,13
 1580 ENDPROC
 1590 :
 1600 DEF PROCError
 1610 ON ERROR OFF:VDU26,12:REPORT:PRINT
" at Line ";ERL;"."
 1620 *FX4,0
 1630 ENDPROC
 1640 :
 1650 DEF PROCFileError
 1660 VDU26,12:ON ERROR PROCError
 1670 PRINT'" The file 'STORE2' cannot b
e found."''" Type 'PROCGenerate' and a l
ist of"'" the first 3402 prime numbers c
an be"'" saved as 'STORE2'."
 1680 ENDPROC
 1690 :
 1700 DEF FNTest(T$,U%)
 1710 IF T$="": =-1
 1720 IF VALT$=0:PROCClear:=0 ELSE T=EVA
LT$
 1730 IF T<0 OR T>U%: =0
 1740 IF T<>INTT: =0
 1750 =T
 1760 :
 1770 :REM*******************************
 1780 DEF PROCGenerate:REM Not called by
 the main program
 1790 VDU15
 1800 P%=&3000:REM (Not redundant, since
 PROCGenerate must run alone.)
 1810 B%=P%+4:!B%=2
 1820 PRINT'SPC9;"N";SPC3;"Nth Prime"
 1830 PRINT'SPC9;"1.";SPC6;"2"
 1840 Z%=1
 1850 REPEAT
 1860 Z%=Z%+2:S%=INT(SQRZ%):N%=P%
 1870 REPEAT
 1880 N%=N%+4:T%=!N%
 1890 UNTIL Z% MOD T%=0 OR T%>=S%
 1900 IF Z% MOD T%<>0 B%=B%+4:!B%=Z%:PRI
NT(B%-P%)/4;".";SPC6;Z%
 1910 UNTIL Z%>=31627
 1920 PRINT" Press Return to save the pr
ime numbers as 'STORE2'."
 1930 REPEAT:UNTIL GET = &0D:*SAVE STORE
2 3000+3530
 1940 ENDPROC
```

MS-DOS and DFS Compared

Kai S. Ng provides a useful comparison between the commands used by the PC operating system MS-DOS and the Beeb's DFS filing system.

Very often, BBC microcomputer users are having to use PCs at their workplaces. I hope the following cross-reference tables will assist anyone who is experiencing the inconvenience of using the different filing systems implemented on the two very different machines.

Table 1 lists those commands for which there is a DFS equivalent, Table 2 lists DOS commands for which there is no DFS equivalent, and Table 3 lists the DFS commands which do not exist in MS-DOS with suggested alternatives. The 'i' or 'e' in the leftmost column indicates whether the DOS command is internal or external (i.e. requiring access to disc).

	MS-DOS (3.3)	DFS (2.26)	Purpose
i	a: b: ...	*DRIVE	changes the disc drive
e	append	*LIB	sets a search path for data files
e	attribute	*ACCESS	displays or changes the attributes of selected files
i	chdir (cd)	*DIR	changes the currently selected directory
e	chkdsk	*VERIFY	scans the disc to check for legibility and errors
i	cls		CLS (from BASIC) clears the console screen
i	copy	*COPY	copies a file from one disc to another
i	del (erase)	*DELETE	erase a single file
i	del (erase)	*WIPE	erase multiple files with individual prompts
i	del (erase)	*DESTROY	erase a group of files with a single prompt
i	dir/w	*CAT (*OPT1,0)	displays the catalogue of current directory or disc
i	dir	*EX or *INFO	reports detailed information on (*OPT1,1) files
i	dir	*FREE	displays the number of files and amount of free memory on disc
e	diskcopy	*FORMAT then *BACKUP	identically reproduce a normal disc
e	format	*FORM	formats a disc
e	label	*TITLE	changes the volume title label of a disc
e	more	Ctrl-N to set up	sends output to console one screen,Shift to scroll
i	path	*LIB	sets a search path for command files
i	rename	*RENAME	changes the name of a file
e	replace	*LOAD then *SAVE	updates a previous version of a file
i	time (date)	*TIME	displays or changes the time and date (on Master only)
i	type	*TYPE or *LIST	displays a text file
i	ver	*HELP	reports the system version number
i	vol	*CAT	reports the disc's volume title

Table 1. DOS commands for which there is a DFS equivalent

	MSDOS (3.3)	Purpose
e	assign	divert filing operations on drives x,y.. to z
e	backup	produce a backup selected files on another disc, which can only be recovered using the "restore" command
e	break	sets CTRL-C check for halting a program
i	chcp	changes the current (international) code page for the command processor "command.com"
e	command	starts the "command.com" file
e	comp	compares the content of 2 files
i	ctty	changes the device from which you issue commands
e	diskcomp	compares the content of two discs
e	exe2bin	converts .exe executable files to binary format
i	exit	terminates the current "command.com" program and returns to a previous level, if one exits
e	fastopen	assigns the number of frequently used files to track
e	fc	compares two sets of files and displays the differences between them
e	fdisk	configures the hard disc partition for use with MSDOS
e	find	searches for a specific string of text in a file or files
e	graftabl	enables an extended character set when using graphics display adaptors
e	graphics	prints a graphics display screen on a printer
e	join	joins a disc drive to a specified path
e	keyb	loads a keyboard program
i	mkdir	makes a new directory
e	mode	sets operation modes for peripheral ports
e	nlsfunc	loads country-specific information
e	print	prints a text file in background mode
i	prompt	changes the MSDOS command prompt
e	recover	attempts to recover a file or disc containing bad section
i	rmdir	removes an empty directory
e	select	formats and installs MSDOS on a new floppy along with country-specific information
i	set	sets one string of characters in the environment to be translated as another string
e	share	installs file sharing and locking over a network
e	sort	reads and sorts a source data file then writes the output to the screen, file or peripheral
e	subst	substitutes a path with a drive letter
e	sys	updates the system files on a disc
e	tree	displays the path of each directory and sub-directory
i	verify	turns the data verify switch on and off when writing to a disc
e	xcopy	copies files and lower level directories

Table 2. DOS commands for which there is no DFS equivalent

DFS (2.26)	What to do in MSDOS
*BUILD	use the line editor program "edlin.com" to create text files
*COMPACT	a similar facility is only available from utilities such as PCTools
*DUMP	use the D command of debug.com
*EXEC	this is identical to executing a .bat batch file
*LOAD	simply loading a program into the memory is not available at MSDOS command level, and it is rather hazardous too since the PC is a "virtual" machine
*SPOOL	not available
*RUN	this is identical to running a .exe or .com program
!BOOT file	the autostart file is "autoexec.bat"

Table 3. DFS commands for which DOS has no direct equivalent

Collapsing Screens (continued from page 9)

```
 1750 EQUB 23:EQUB 1:EQUD 1:EQUD 0
 1760 .Width:EQUB 0:.OldVec EQUW 0
 1770 .MaxY:EQUB 0:.CChar:EQUB 0
 1780 .Pointer:EQUB 0
 1790 .pos EQUB 0:.vpos EQUB 0
 1800 .DCount:EQUB 0:.NotMoved:EQUB 0
 1810 .Xpos EQUB 0:.Ypos EQUB 0
 1820 .CharX:EQUS STRING$(MaxChars%,CHR$
0):EQUB0
 1830 .CharY:EQUS STRING$(MaxChars%,CHR$
0):EQUB0
 1840 ]:NEXT
 1850 OSCLI("SAVE Fall "+STR$~code%+" "+
STR$~O%+" "+STR$~exec%+" "+STR$~exec%)
 1860 END
```

Listing 2

```
30000 DEF PROCcls
30010 LOCAL I%,T%
30020 CALL &1400
30030 VDU 23,1,0,0,0,0,0,0,0,0
30040 VDU 26
30050 I%=INKEY(100)
30060 FOR I%=1 TO 20
30070 VDU 11
30080 T%=TIME:REPEAT UNTIL TIME>T%+3
30090 NEXT
30100 CLS
30110 VDU 23,1,1,0,0,0,0,0,0,0
30120 ENDPROC
```

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

CONTINUED FRACTIONS (Vol.9 No.7)

Mr.J.G.Scadding of Beaconsfield has found that the program fails to correctly identify numbers such as 7^(1/3) as an irrational number. It is very difficult to suggest a modification which would reliably identify all irrational numbers, but the following modification will cope with this particular instance. Change line 250 to read:

```
 . . . . ELSE IF ERR=26 OR ERR=27
             THEN PROCirrational:GOTO130
```

PLANET Z (Vol.9 No.7 Disc Only)

The author has informed us that line 2850 in the program *PLZ* should be changed to:

```
 2850 PROCstop
```

BEEBUG Function/Procedure Library (2)

by Stefano Spina

This month we present further functions and procedures from the library of Stefano Spina, again concentrating on various string handling or related routines. Note that some of these functions/procedures call, in turn, other routines, including in some cases FNmd listed last month.

The last procedure, PROCcomm, is quite different and provides an organised way of entering star commands from within any program. It also allows some abbreviations, such as just '$' to select the $ (root) directory rather than entering the full command DIR $. Some of the abbreviations are applicable only to the ADFS. Alternatives could easily be incorporated for DFS users.

This month's routines are numbered to follow on from those listed last month. To append the new code you will need to type it in and then spool it out to a file using the *SPOOL <filename> command. Then load your existing function/procedure library, use *EXEC to load in the spooled additions and resave the complete library as a whole.

Note that any procedure can be easily tested by loading the library as a Basic program, and then typing in function and procedure calls in immediate mode. The examples quoted in the documentation for each routine could be explored and tested in this way.

THE FUNCTION/PROCEDURE LIBRARY (PART 2)

Routine 12: Format

Type:	PROCEDURE	
Syntax:	PROCfrt(S$,C%,R%,W%,J%)	
Purpose:	**Formats a given string into a given width right justified.**	
Parameters:	S$	String to be formatted
	C%	Column
	R%	Row
	W%	Width
	J%	Flag to set/reset output layout TRUE calls FNstyle to modify the layout FALSE doesn't modify the string's layout
Notes:	The string is formatted in several rows of the required length and the sub-strings are right justified. If J% is set to TRUE then FNstyle is called to put every word with the first character in upper case and the rest in lower case.	
Related:	FNmd, FNspaces, FNstyle	

```
10 S$="this is a string that MUST be
   formatted into TWENTY columns,
   starting at column 10, row 10"
20 PROCfrt(S$,10,1,20,TRUE):
   REM the string's layout is changed

           This  is  a   string
           that     Must     be
           formatted       into
           Twenty      columns,
           starting  at  column
           10, row 10
```

Routine 13: Length

Type:	FUNCTION	
Syntax:	FNlength(S$,C$,D%,M%)	
Purpose:	**Formats a string to the required length adding given characters at the end or at the beginning of the string.**	
Parameters:	S$	String to be formatted
	C$	Fill character
	D%	Required length
	M%	Flag to set fill mode 0 The fill character is added at the beginning 1 The fill character is added at the end

Notes:	This function allows data items to be all of the same length so that tables are easier to display or print.
Related:	None

```
10 PRINT FNlength("String","*",10,1):
   REM produces "String****"
20 num%=1500
30 PRINT FNlength(STR$num%,"0",6,0):
   REM produces "001500"
```

Routine 14: Spaces

Type:	FUNCTION	
Syntax:	FNspaces(S$,W%)	
Purpose:	**Formats a string to the required width inserting dummy spaces. This differs from FNlength because spaces are inserted all along the string.**	
Parameters:	S$	String to be formatted
	W%	Width
Notes:	This function can be used as a stand alone routine or in conjunction with PROCformat and FNstyle.	
Related:	None	

```
10 str$="String 19 chrs long"
20 PRINT FNspaces(str$,24):
   REM produces "String   19   chrs  long"
```

Routine 15: StrBox

Type:	PROCEDURE	
Syntax:	PROCstbx(A%,B%,C%,D%,M%)	
Purpose:	**Draw horizontal/vertical lines or boxes at the given text co-ordinates in any graphics mode.**	
Parameters:	A%	Left text column
	B%	Left text row
	C%	Right text column
	D%	Right text row
	M%	Flag for output mode 0 Draws a box 1 Draws a horizontal line 2 Draws a vertical line
Notes:	This procedure differs from PROCbox because the co-ordinates are given in text format; this makes it easier to create tables on the screen. Lines are always drawn at the centre of the text line and, in high resolution modes, vertical lines are drawn twice with an offset of two pixels to make them the same width as horizontal ones.	
Related:	FNmd	

```
10 MODE 129
20 PROCstbx(1,1,38,25,0):
   REM box from col. 1, row 1 to col. 38,
   row 25
30 PROCstbx(1,1,38,0,1):
   REM horiz. line at row 1 from col.1
   to col. 38
40 PROCstbx(1,1,0,25,2):
   REM vert. line at col. 1 from row 1
   to row 25
```

Routine 16: String

Type:	PROCEDURE	
Syntax:	PROCstr(C$,C%,R%,N%,H%)	
Purpose:	**Prints graphic string in mode 7/135 at given col./row**	
Parameters:	C$	Alphanumeric character
	C%	Column
	R%	Row
	N%	Number of characters to be printed
	H%	Teletext graphics colour code (145-151)
Notes:	This simple procedure allows graphic output on the screen in mode 7/135 depending on the selected alphanumeric char.	
Related:	None	

```
10 PROCstr(",",2,8,38,141):
   REM continuous red bar at col. 2, row 8
```

Routine 17: Style

Type:	FUNCTION
Syntax:	FNstyle(S$)
Purpose:	**Changes the given string's lay-out forcing every word to start with an upper case letter while the rest is forced to lower case.**

Parameters:	S$	String to be changed
Notes:	The global logical variable *style%* acts as a switch; if it is set to FALSE before the function's call, no changes will be performed.This variable is set to TRUE on exit.	
Related:	FNswap	

```
10 style%=TRUE
20 PRINT FNstyle("CHANGE STRING's layOUT"):
   REM "Change String's Layout"
30 style%=FALSE
40 PRINT FNstyle("UNchaNGE stRing"):
   REM "UNchaNGE stRing"
```

Routine 18: Swap

Type:	FUNCTION	
Syntax:	FNswap(S$,M%)	
Purpose:	**Swap cases in the given string**	
Parameters:	S$	String to be swapped
	M%	Flag for output selection
		0 Forces lower case
		1 Forces upper case
		2 Reverses case
Notes:	None	
Related:	None	

```
10 str$="123 @abcDEF, klMN"
20 PRINT FNswap(str$,0):
   REM produces "123 @abcdef, klmn"
30 PRINT FNswap(str$,1):
   REM produces "123 @ABCDEF, KLMN"
40 PRINT FNswap(str$,2):
   REM produces "123 @ABCdef, KLmn"
```

Routine 19: Command

Type:	PROCEDURE
Syntax:	PROCcomm
Purpose:	**Opens a "window" into the operating system allowing commands to be executed without stopping execution of the Basic program.**
Parameters:	None
Notes:	Some common commands are shortened to a single character: . Performs a catalogue 0 Mounts drive :0 1 Mounts drive :1 ^ Sets the parent directory $ Sets the root directory The abbreviations '0', '1' and '^' apply only to the ADFS. Similar abbreviations could be programmed in for the DFS.
Related:	PROCcolour, PROCstbx

```
10 IF me%=5 THEN PROCcomm
```

```
21040 :
21050 REM Format
21060 :
21070 DEF PROCfrt(S$,C%,R%,W%,J%)
21080 LOCAL A$,A%,B%,D%,F%,L%:L%=FNmd
21090 IF L%=2 OR L%=5 A%=19 ELSE IF L%=0
 OR L%=3 A%=79 ELSE A%=39
21100 IF L%=3 OR L%=6 OR L%=7 B%=24 ELSE
 B%=31
21110 VDU28,C%,B%,A%,R%:W%=W%+1
21120 REPEAT
21130 L%=LEN(S$)
21140 IF L%<=W%-1 F%=TRUE:A$=S$:GOTO 212
00
21150 A%=0:B%=1
21160 REPEAT
21170 D%=INSTR(MID$(S$,W%-A%,1)," "):A%=
A%+1
21180 UNTIL D%>0 OR A%=W%
21190 A$=MID$(S$,B%,W%-A%)
21200 style%=J%:A$=FNstyle(A$)
21210 IF F% PRINT A$ ELSE PRINT FNspaces
(A$,W%-1)
21220 S$=RIGHT$(S$,L%-(W%-A%+1))
21230 UNTIL F%:VDU26
21240 ENDPROC
21250 :
21260 REM length
21270 :
21280 DEF FNlength(S$,C$,D%,M%)
21290 LOCAL L%:L%=LEN(S$):IF L%>=D% =LEF
T$(S$,D%)
21300 IF M%=0 =STRING$(D%-L%,C$)+S$ ELSE
=S$+STRING$(D%-L%,C$)
21310 :
21320 REM Spaces
21330 :
21340 DEF FNspaces(S$,W%)
21350 LOCAL A$,A%,B%,C%,D%,E%,F%
21360 A%=LEN(S$):IF A%>=W% =S$
```

```
21370 E%=W%-A%:B%=FALSE
21380 REPEAT
21390 F%=1:C%=FALSE
21400 REPEAT
21410 D%=INSTR(S$," ",F%)
21420 IF D%=0 THEN C%=TRUE ELSE A$=A$+MI
D$(S$,F%,D%-F%+1)+" ":E%=E%-1:IF E%=0 TH
EN B%=TRUE ELSE F%=D%+1
21430 UNTIL B% OR C%
21440 IF A$="" THEN B%=TRUE ELSE IF C% S
$=A$+MID$(S$,F%):A$=""
21450 UNTIL B%
21460 IF A$="" THEN =S$ ELSE =A$+MID$(S$
,D%+1)
21470 :
21480 REM StrBox
21490 :
21500 DEF PROCstbx(A%,B%,C%,D%,M%)
21510 LOCAL S%,E%,F%
21520 S%=FNmd
21530 IF S%=3 OR S%=6 OR S%=7 ENDPROC
21540 IF S%=1 OR S%=4 E%=40 ELSE IF S%=2
 OR S%=5 E%=20 ELSE E%=80
21550 F%=1280/E%:A%=(A%*F%)-1+(F%/2)
21560 B%=(((31-B%)*32)-1)+16
21570 IF M%<2 C%=(C%*F%)-1+(F%/2)
21580 IF M%<>1 D%=(((31-D%)*32)-1)+16
21590 MOVE A%,B%
21600 IF M%=0 DRAW C%,B%:DRAW C%,D%:DRAW
 A%,D%:DRAW A%,B%:IF S%=0 MOVE A%+2,B%:D
RAW A%+2,D%:MOVE C%+2,B%:DRAW C%+2,D%
21610 IF M%=1 DRAW C%,B% ELSE DRAW A%,D%
:IF S%=0 MOVE A%+2,B%:DRAW A%+2,D%
21620 ENDPROC
21630 :
21640 REM String
21650 :
21660 DEF PROCstr(C$,C%,R%,N%,H%)
21670 LOCAL S$
21680 S$=CHR$(H%)+STRING$(N%,C$)
21690 PRINTTAB(C%,R%)S$
21700 ENDPROC
21710 :
21720 REM Style
21730 :
21740 DEF FNstyle(S$)
21750 IF NOT style% style%=TRUE:=S$
21760 LOCAL A$,B$,C$,A%,B%,C%
21770 C%=LEN(S$):A%=1
21780 REPEAT:B$=""
21790 REPEAT
21800 A$=MID$(S$,A%,1):B$=B$+A$:A%=A%+1
21810 UNTIL A$=" " OR A%>C%
21820 B%=LEN(B$)
21830 A$=FNswap(LEFT$(B$,1),1)
21840 IF B%=1 B$=A$ ELSE B$=A$+FNswap(MI
D$(B$,2),0)
21850 C$=C$+B$
21860 UNTIL A%>C%
21870 =C$
21880 :
21890 REM Swap
21900 :
21910 DEF FNswap(S$,M%)
21920 IF S$="" =""
21930 LOCAL A$,F%,H%,I%,L%:L%=LEN(S$)
21940 FOR I%=1 TO L%
21950 H%=ASC(MID$(S$,I%,1))
21960 IF H%<65 OR (H%>90 AND H%<97) OR H
%>122 THEN GOTO22000
21970 IF  (H%>=65 AND H%<=90) F%=FALSE
21980 IF  (H%>=97 AND H%<=122) F%=TRUE
21990 IF NOT(F%) AND (M%=0 OR M%=2) H%=H
%+32 ELSE IF F% AND (M%=1 OR M%=2) H%=H%
-32
22000 A$=A$+CHR$H%
22010 NEXT I%
22020 =A$
22030 :
22040 :
22050 REM Command
22060 :
22070 DEF PROCcomm
22080 LOCAL A$:VDU22,128
22090 PROCstbx(0,0,79,5,0)
22100 PROCstbx(0,2,79,0,1)
22110 PROCcolour(1):PRINTTAB(20,1)"   * *
 *  C o m m a n d   P a g e  * * *  "
22120 PROCcolour(0):PRINTTAB(11,3)"Input
 string to be executed by the MOS and pr
ess Return"
22130 PRINTTAB(20,4)"or Press Return wit
hout input to exit"
22140 VDU28,0,30,79,6:CLS
22150 REPEAT
22160 INPUTLINE TAB(4)"*"A$
22170 IF LEFT$(A$,3)="DIR" PRINT:OSCLI(A
$):OSCLI"." ELSE IF A$="0" OR A$="1" OSC
LI"MOUNT"+A$:OSCLI"." ELSE IF A$="^" OSC
LI"DIR^":OSCLI"." ELSE IF A$="$" OSCLI"D
IR$":OSCLI"." ELSE OSCLI(A$)
22180 PRINT
22190 UNTIL A$="":VDU26
22200 ENDPROC
22210 :
```

Sideways RAM Commands for the Model B

*Norman Smith describes a short routine to implement the Master's *SRWRITE and *SRSAVE commands on a model B with particular reference to the ROM Filing System.*

GENERAL

In BEEBUG Vol.8 Nos 8 & 9 two articles by Jon Keates were published on the ROM Filing System for the Master 128. These programs, as they stand cannot be used on the basic model B as it lacks both the sideways RAM and more particularly the *SRWRITE command for which it has no direct equivalent. However, many model B owners have fitted a sideways RAM or RAM/ROM board, and in my case a successful emulation was obtained with the Aries B12 board which has 1 bank of 16k RAM in slot 0. Other boards with the RAM in a different slot(s) should also work correctly.

The *SRWRITE command follows the format: source address, length of data, destination address and an optional RAM slot number (there is also an alternative form in which source start and end addresses are specified rather than start address and length). This can be emulated on a model B using a procedure:

```
    PROCsrwrite(source%,length%,dest%,
        ramslot%)
```

which together with a supporting assembler routine is presented as listing 1. In principle, the variables are the same as would be used in the *SRWRITE command. The occasional changes are where a base memory location may have to be added to the source% or dest% variables to establish the correct memory area or be subtracted from the length variable to indicate the true data length and these must be ascertained from the program details. Examples of the latter occur in the revised versions of lines 280 in RFShead and 1530 in RFSload (see Vol.8 No.8).

This procedure can be used as the basis for the conversion of the *SRWRITE command in any suitable Master 128 programs which are being converted to the model B, and other versions have also been used successfully to emulate the Master's *SRREAD and *SRSAVE commands.

PROGRAM INFORMATION

The program listing contains the definition of PROCsrwrite and a short piece of machine code, PROCass1, with P% set at &A00 and using zero page memory &72 to &78 (RFSload uses &70/&71 and the &80 area). It saves the original ROM pointer at &F4 and enters the new value as set by ramblok%, uses a move routine and finally resets &F4 to its original value. This procedure may also be used for moves between normal areas of RAM and a dummy value used for ramblok%).

The new procedures should be typed in and then *SPOOLed, before using *EXEC to add it to RFShead. Line 280 should then be changed as follows:

```
PROCass1:PRINT''"PROCsrwrite(&6000,&";~(O%
-&6000);",&8000,";ramblok%;")"
```

after which RFShead can be re-saved, preferably under another filename. It can then be run, replying to the 'RAM Bank Number ?' prompt with the RAM slot number appropriate to the RAM board in use. At the end of the program, the prompt in line 280 will appear with the correct length of the data depending on the length of the title and version number, viz:

```
    PROCsrwrite(&6000,&12E,&8000,0)
```

If it is desirable to install the ROM header image in sideways RAM (SRAM), the Copy key should be used to duplicate this statement, followed by Return. The Break key should then be pressed to initialise the SRAM and '*H. RFS' should display the star commands accepted by the ROM image.

If all is correct *EXEC the additional program to RFSload (from Vol.8 No.9) and make changes as shown in the following lines:

```
  100 MODE7
 2060 rm$=FNverchr("0000",1)
  1140 PROCsrwrite(ptr%,blen%,srload%,ram
blk%)
  1190 PROCsrwrite(footer%,3,srload%,ramb
lk%)
  1530 PROCsrwrite(hblok%,P%-hblok%,srloa
d%,ramblk%)
```

Add line 2075:

```
 2075 PROCass1
```

Line 2060 is suitable for the Aries B12 with RAM in slot 0, this should be changed to suit other RAM boards. Save the program, again preferably under a new name, and run it, when it should respond as in the original program.

The program RFSload also contains the Master *SRSAVE command, which is also not implemented on the model B. In the past several programs have appeared in BEEBUG with details of programs to download SRAM/ROM to disc automatically. Of these, some of the machine code ones are rather lengthy and some Basic ones very slow. With a further small amendment to RFSload, a model B '*SAVE' may be used to perform this function. This requires the following line changes and additions:

```
  110 HIMEM=&3BFF
    455 PROCsrwrite(&8000,(srload%+1-&8000
),&3C00,ramblk%)
    470 PRINT''" *SAVE ";title$;"3C00+"+ST
R$~(srload%+1-&8000)+" 8000 8000"
```

This lowers HIMEM to make room for PROCsrwrite to move the maximum contents of a 16k RAM to &3C00/&7C00. The PRINT statement in line 470 containing the correct values appears at the end of the program from where the RFS data image can be *SAVEd by duplicating the statement using the Copy key followed by Return. Users with shadow RAM could leave HIMEM at &3FFF and change the &3C00 references to &4000.

```
   50 REM Programs RFShead and RFSload
   60 REM Amended for BBC Model B
   70 REM By Norman Smith
   80 REM BEEBUG April 1991
   90 REM Program subject to copyright
  100 MODE7
 3000 :
 3010 DEF PROCsrwrite(source%,length%,de
st%,ramblok%)
 3020 ?&72=source% MOD256:?&73=source% D
IV256:?&74=length% MOD256:?&75=length% D
IV256:?&76=dest% MOD256:?&77=dest% DIV25
6:?&78=ramblok%
 3030 CALL mcopy
 3040 ENDPROC
 3050 :
 4000 DEF PROCass1
 4010 FOR opt%=0TO2STEP2:P%=&A00
 4020 [OPT opt%
 4030 .mcopy
 4040 LDA &F4:STA &79
 4050 LDA &78:STA &F4:STA &FE30
 4060 LDX #0:CPX &75:BEQ smallp
 4070 .pagelp LDY #0
 4080 .bytelp1 LDA (&72),Y:STA (&76),Y
 4090 INY:BNE bytelp1
 4100 INX:INC &73:INC &77
 4110 CPX &75:BNE pagelp
 4120 .smallp LDY #0
 4130 CPY &74:BEQ fin
 4140 .bytelp2 LDA (&72),Y:STA (&76),Y
 4150 INY:CPY &74:BNE bytelp2
 4160 .fin
 4170 LDA &79:STA &F4:STA &FE30
 4180 RTS
 4190 ]NEXT:ENDPROC
```

CORPLAN

For serious work with Wordwise Plus

- ❖ Reviewed in BEEBUG Vol.9.No.8
- ❖ Descriptive indexing for your letters & documents
- ❖ Your own library of layout forms, letterheads etc.
- ❖ Automatic import of addresses, references, dates etc.
- ❖ CORPLAN does the layout, you just type the text!
- ❖ Resident utilities for mailmerge, label printing etc.
- ❖ For B, B+ & Master. Needs discs & Wordwise Plus
- ❖ Many special features - see free information sheet
- ❖ Pack contains disc, tutorial manual, keystrip etc.
- ❖ Price £19.50, post free UK, 14 day refund

CORPLAN Computer Systems

VISA

Three Gables, 7a Talbots Drive, Maidenhead, Berks, SL6 4LZ. Phone or Fax (0628) 24591

Practical Assembler (Part 10)

by Bernard Hill

AUTOMATIC ENCRYPTION OF FILES

In the last article of this series we take a look at the filing system calls from an opposite point of view: that of intercepting the filing system calls themselves rather than writing programs to use them as we illustrated in the last two articles.

ENCRYPTION

There have been a number of programs in BEEBUG recently which have created encrypted (encoded) versions of files, and we extend this idea here to include an assembler program which will automatically and invisibly encrypt every file written to disc and decrypt every file read from disc. It is to be completely transparent to the user so that he can use all the normal filing system commands and not be aware that the files are encrypted: that is until someone attempts to use the files with encryption off! The routine is to be started with the command:

```
*ENC +
```

which will load into memory a section of assembler and change the relevant filing system vectors to point to this code so that it is called whenever a *LOAD, *SAVE, BGET or BPUT is issued. The facility will be turned off with:

```
*ENC -
```

THE ENCRYPTION ALGORITHM

The EOR (exclusive OR) operation is particularly convenient for this because the same routine can be used to encrypt as to decrypt. However, EORing a whole file with a constant value is too easy to decode by prying eyes, so we are going to EOR the bytes in the file with a sequence of 256 'random' bytes in the style of the encryption algorithm in BEEBUG Vol.9 No.8 Workshop. A table of random bytes can be found in the Operating System ROM where a random address in the range &C000 to &D000 will be used for the start of this table.

IMPLEMENTATION

The calling program has a number of tasks to perform:

1. It must change the required filing system vectors to point to the new code when *ENC + is called.
2. It must restore the old ones when *ENC - is called.
3. It must provide the new routines as outlined in the algorithms below.

There are thus five assembler calls which need to be attended to, and these are:

1. BPUT

This routine writes the byte in the A register to disc in the current file. We must therefore first EOR it with the table value (EOR table,X) after having found the offset within the table by performing X=PTR#f MOD 256 first.

PTR# is called by the OSARGS routine with A=0, Y set to the file handle, and X containing the address of a 4-byte zero-page parameter block where the value of PTR# will be placed:

```
STA &100:STX &101 \ temporary store
LDX #&A8 \&A8-&AF are generally usable
LDA #0
JSR &FFDA \ OSARGS
LDX &A8  \ get lowest byte of PTR# to X
```

Now we can encrypt the value to be sent to the file with:

```
LDA &100 \ restore A
EOR table,X
```

and call the old value of BPUT with:

```
LDX &101 \ restore X
JMP (oldbputv)
```

This appears as lines 920-960 of the listing.

2. BGET

This is a very similar exercise, as follows:

```
    Find the low byte of PTR#f to X
    Call the old vector with an indirect
       JSR (oldbgetv)
    Decrypt the byte with EOR table,X
       before returning.
```

See lines 840-880 of listing 1. Note that an indirect JSR is not available in 6502 assembler, so we perform a small trick to do the same thing:

```
JSR temporary
...
.temporary JMP (address)
```

When the return is made from the routine indicated, it will return to the line after the JSR, as required. This trick is used for a call to OSBGET and OSFILE in listing 1.

3. *SAVE (SAVE"xx" in Basic, etc.)

This is performed when OSFILE is called with A=0 on entry. We therefore test for this value and also A=&FF (a *LOAD) and merely jump to the old routine with JMP (oldosfilev) if A is neither. Supposing A=0, however, then we must:

```
    Encrypt the whole of the data to be
       *SAVEd
    Perform the *SAVE
    Decrypt the data so that you don't
       notice it's been changed. No good
       returning to your View edit screen
       to see garbage!
```

Now, as indicated in last month's article, OSFILE is called with an 18-byte parameter block, the start and end addresses being in bytes 10-13 and 14-17. Since this is really only a demonstration program we're not making it second-processor compatible and therefore only considering the bottom 2 bytes of each of these. The subroutine .xlate in lines 750-820 performs this block encryption operation on a 2-byte address range loaded into 'start' and 'end' and so this is called twice as indicated above. The whole routine is in lines 630-710.

4. *LOAD

There are two forms of *LOAD depending on whether the actual file address is being used (**LOAD file*) or forced to another value as in **LOAD file 4000*. This is indicated - as mentioned last month - by the 6th byte of the parameter block being zero. Should this be so then we shall need to note the start address from the parameter block BEFORE the load, or else use the load address which will have appeared in the parameter block after the actual load. Lines 420-470 of the program perform this choice.

Furthermore, we can use the same routine for decryption as we used above (.xlate) provided that we calculate the end address for the translation routine by adding the start address and the length in parameter block bytes 10-11 (Lines 510-530).

WEAKNESSES OF THE PROGRAM

There is another filing system call which saves and loads which is more often used over a network, and that is OSGPBP which gets or puts a range of bytes to the disc. We've not intercepted this as it is used less often, but it ought to form part of any complete encryption program.

The program as supplied works with Basic and View; you are left to test it with your software. It is interesting to save (in encrypted mode) a word processor document and then view it with *DUMP with encryption successively on and off.

Since the program resides in pages 9 and &A any interference with these will probably result in a system crash on the next filing system access. In a real situation the routine would live in sideways ROM to protect it from such interference.

A more serious problem with the supplied program is that calling *ENC + twice in a row results in the software losing the addresses of the real filing system vectors causing the system to hang when called. Ideally it should look at the contents of the vectors to see if it is already installed and exit with a suitable message on the second call to *ENC +. The same is true for a call to *ENC - when the system is not installed, and I leave this easy enhancement to you.

I hope you've enjoyed this Practical Assembler series. We've ranged far and wide and I've tried to keep on the practical side with the examples we've used, and yet leave you with enough ideas and scope to go it on your own.

```
   10 REM Encrption program
   20 REM Version B2.0
   30 REM Author   Bernard Hill
   40 REM BEEBUG   April 1991
   50 REM Program subject to copyright
   60 :
  100 MODE7
  110 zpg=&A8:zp2=&AA
  120 table =&C000+RND(&1000)
  130 store=&900:code=&906
  140 oldfilev=store:oldbgetv=store+2
  150 oldbputv=store+4
  160 FOR opt=0 TO 3 STEP 3
  170 P%=code
  180 [OPT opt
  190 .strt LDA (&F2),Y:CMP #32:BNE go
  200 INY:JMP strt:.go
  210 CMP #ASC"+":BEQ encon
  220 CMP #ASC"-":BEQ encoff
  230 EQUW 0:EQUS "ENCRYPT bad syntax: +
 or -":BRK
  240 .encoff
  250 EQUS FNcopy2(oldfilev,&212)
  260 EQUS FNcopy2(oldbgetv,&216)
  270 EQUS FNcopy2(oldbputv,&218)
  280 JSR print
  290 EQUS "Encryption off"+CHR$13:NOP
  300 RTS
  310 .encon \ set up vectors
  320 EQUS FNsetup(&212,oldfilev,newfile
v)
  330 EQUS FNsetup(&216,oldbgetv,newbget
v)
  340 EQUS FNsetup(&218,oldbputv,newbput
v)
  350 JSR print
  360 EQUS "Encryption on"+CHR$13:NOP
  370 RTS
  380 \
  390 .newfilev CMP #0:EQUS FNjeq(osfsav
e)
  400 CMP#&FF:BEQ osfload:JMP (oldfilev)
  410 \
  420 .osfload STX zpg:STY zpg+1
  430 LDY #6:LDA (zpg),Y \ forced addr?
  440 PHP:BNE over1:JSR getstart \yes
  450 .over1 LDY zpg+1:LDA #&FF
  460 JSR osfile:PLP:BEQ over2 \no
  470 JSR getstart
  480 .over2 \ get length
  490 LDY #&A:LDA (zpg),Y:STA len
  500 INY:LDA (zpg),Y:STA len+1
  510 \ end=start+len
  520 CLC:LDA start:ADC len:STA end
  530 LDA start+1:ADC len+1:STA end+1
  540 JSR xlate \ and encrypt
  550 .fin LDA #&FF:LDX zpg:LDY zpg
  560 RTS
  570 \
  580 .getstart
  590 LDY #2:LDA (zpg),Y:STA start
  600 INY:LDA (zpg),Y:STA start+1
  610 RTS
  620 \
  630 .osfsave \ store pblock start add
  640 PHA:STX zpg:STY zpg+1:LDY #&A
  650 LDA (zpg),Y:STA start:INY
  660 LDA (zpg),Y:STA start+1:LDY #&E
  670 LDA (zpg),Y:STA end:INY
  680 LDA (zpg),Y:STA end+1
  690 JSR xlate:LDX zpg:LDY zpg+1:PLA
  700 JSR osfile
  710 JSR xlate:LDX zpg:LDY zpg+1:LDA #0
:RTS
  720 \
  730 .osfile JMP (oldfilev)
  740 \
  750 .xlate LDA start:STA zp2
  760 LDA start+1:STA zp2+1:LDY #0:LDX #
0
  770 .loop LDA zp2:CMP end:BNE over
  780 LDA zp2+1:CMP end+1:BEQ finxlat
  790 .over LDA (zp2),Y:EOR table,X
  800 STA (zp2),Y:INC zp2:BNE P%+4
```

Continued on page 44

1st course

VDU and FX Calls (Part 1)

by Mike Williams

INTRODUCTION

Tucked away in your User Guide, or in the appendices of your Welcome Guide if you have a Master 128 or Master Compact, you will find tables and some explanation of the so-called VDU codes and FX calls. Although accessible through Basic, these elements are not part of Basic as such, but give Basic programmers direct access to a whole range of useful features within the operating system.

The original User Guide for the BBC micro was reasonably forthcoming about the nature and use of VDU and FX calls where documented, but the coverage was far less than complete. For the Master 128 and Master Compact, the Welcome Guide is much more thorough in listing virtually all VDU and FX calls but provides much less information on individual calls.

I thought it would therefore prove fruitful for our *First Course* series to examine the nature of VDU and FX calls, and to highlight some of the more useful. This month we will be concentrating our attention on VDU calls.

VDU CALLS

VDU calls are fewer in number and generally easier to understand than FX calls so we shall deal with these first. All VDU codes are handled by a part of the micro's operating system referred to as the VDU driver. Normally all output to the screen (and to a printer if enabled) is handled by the VDU driver.

The VDU call itself is followed by one or more numbers separated (followed) by either commas or semicolons. At its very simplest, a VDU call is much like a PRINT statement in its effect. For example:

```
  PRINT CHR$(65)
```

and:

```
  VDU 65
```

would both cause a single letter A (which has ASCII code 65) to appear on the screen. Generally where strings of characters are concerned, PRINT is much the better choice and easier to understand. For example:

```
  VDU 72,101,108,108,111
```

is more long-winded and more difficult to follow than just writing:

```
  PRINT"Hello"
```

There are two situations where the VDU approach may be preferable. One is where a single key response is being sought, which has to be echoed to the screen. Consider:

```
  PRINT"Select menu option (A-E):";
  REPEAT:G=GET:UNTIL G>64 AND G<70:VDU G
```

The GET function does not automatically echo the character entered on the screen. When an appropriate key is pressed, the REPEAT-UNTIL loop terminates, and the VDU call echoes the character to the screen.

A second situation where the use of VDU is preferable to PRINT, is in mode 7 when teletext codes are being used. Generally using VDU followed by one or more teletext codes is to be preferred to using PRINT which also involves the CHR$ function with each ASCII code. Compare:

```
  VDU 131,157,132
```

with:

```
  PRINT CHR$(131)CHR$(157)CHR$(132)
```

to set yellow (131) background (157) with blue (132) text.

For ASCII codes between 32 and 127 the VDU function simply displays the corresponding character; on a Master 128 or Master Compact the same is true for codes 128 to 255, though these are normally undefined on a model B (except in mode 7, the teletext mode). The code values which are most interesting are those in the range 0 to 31. These codes are also referred to as control codes, and can be entered from the keyboard by pressing the Ctrl key together with a corresponding letter key. Thus Ctrl-B has the same effect as VDU2, Ctrl-C the same as VDU3 and so on. Indeed:

```
  VDU14
  Ctrl-N
  PRINT CHR$(14)
```

all have the same result, to set the screen into paged mode (not scrolling mode). Apart from VDU0 which has no effect whatsoever, all the control codes have a particular function and it is these which are listed in the appropriate User or Welcome Guide.

Some of these are quite simple; I am sure most readers are aware that VDU2 and VDU3 enable and disable printer output (equivalent to Ctrl-B and Ctrl-C). The full list of VDU codes which are commonly entered as control codes is given in table 1 (though note that *all* VDU codes can be so entered).

VDU Code	Ctrl Code	Function
2	B	Enable printer
3	C	Disable printer
7	G	Beep
10	J	Linefeed
12	L	Form Feed
13	M	Return
14	N	Paging on
15	O	Paging off

Table 1. Commonly used control codes

Remember that whereas both VDU calls and control codes can be entered directly from the keyboard, only the VDU forms can be included within Basic programs. Note too, that VDU12 (form feed) is equivalent to the CLS instruction when applied to the screen display - form feed as such can only apply to a printer, and then a printer which can perform this function (most of them nowadays).

You can also restate linefeed's function with respect to the screen as 'cursor down'. Although they are not normally used as control codes, you should also note VDU11 (cursor up), VDU8 (cursor left) and VDU9 (cursor right). These functions can be quite useful in programs which need to control the position of the cursor on screen.

The majority of the other VDU calls perform functions which require additional information in the form of further parameters. For example VDU31 moves the text cursor to a specified position with the format:

```
VDU31,x,y
```

where x indicates the position along a row of text, and y indicates a particular text line on the screen. Remember that for text, the top left-hand corner is where x=0 and y=0. x counts from left to right across the screen; y counts from top to bottom down the screen. In fact, writing VDU31 is exactly equivalent to using the TAB function in PRINT statements.

With this VDU call, as with some others, it is important to know the screen mode in which you are operating or you may begin to think VDU doesn't always work. Most screen modes have 32 lines running from line 0 at the top to line 31 at the foot, but modes 3, 6 and 7 have only 25 lines with line 24 at the foot. Likewise, screen displays can consist of 20 columns (0 to 19), 40 columns (0 to 39) or 80 columns (0 to 79).

Two calls which are useful in many programs are VDU24 and VDU28. These enable a program to define either a graphics window (VDU24) or a text window (VDU28). Subsequent graphics or text output will then be confined to the window area defined. The order of the parameters in defining a window is always: 'left boundary, bottom boundary, right boundary, top boundary'. By default, both windows are set to the full screen display and coincide with one another 100 percent.

SCREEN WINDOWS

One simple use of VDU28 is to set a window below a title at the head of the screen so that as subsequent input and output causes the screen to scroll, the title remains fixed and visible. The same technique can be applied to instructions (so that they always remain visible), and this is used in the program *Euler* listed in our Recreational Mathematics feature in this issue. A typical example might be:

```
VDU28,0,31,79,5
```

which ensures that the five lines 0 to 4 are preserved at the head of the screen; the window extends to the full right and left margins of the screen, and to the foot of the screen (assuming an 80 column, 32 line mode, of which mode 0 is the only example).

When you define a new text window, line 0 becomes the top line within the window, and the leftmost character position becomes column zero. Remember this if you use the TAB instruction to position text within a window. But if you choose to redefine the text window, the parameters are always given relative to the full default screen area, and not relative to any currently defined window.

If we look as well at a similar graphics window, then an important difference will begin to emerge. Graphics windows are specified in the same order, but the dimensions are given in graphics units which range from 0 to 1279 horizontally and 0 to 1023 vertically. Moreover, the graphics origin is always in the bottom left-hand corner (for this purpose at least) extending from there to the right and upwards.

GRAPHICS WINDOWS

Thus a typical graphics window might be defined as:

```
VDU24,0;0;1279;863;
```

and (if I have calculated it correctly) this graphics window coincides with the text window defined previously. Note also the use of semicolons rather than commas - this is dealt with below. Unlike a text window, defining a graphics window does not redefine the position of the graphics origin. In the absence of any other instructions, this will continue to be the bottom left-hand corner of the screen by default. A graphics window is much more like a true window onto a graphics drawing area - you can draw lines and curves wherever you like, but only the parts that pass through the window area will be visible. You can change the position of the graphics origin, useful if it makes the maths any easier, and this is done with VDU29,x;y; where x and y are the graphics co-ordinates of the new position of the origin.

For synchronising text and graphics windows it is worth knowing that vertically the height of one line of text is 32 graphics units (in all modes). Horizontally, the width of a character is 64 graphics units (modes 2 and 5), 32 graphics units (modes 1 and 4), or 16 graphics units (mode 0). Modes 3, 6 and 7 are not graphics modes anyway, so cannot cater for graphics.

SPECIFYING GRAPHICS CO-ORDINATES

Before we continue any further let me draw your attention to the semicolons in the above examples. The distinction is important, but once understood we won't need to refer to it again.

Because the Beeb is an eight-bit micro, its basic unit of information is an eight bit number with a range therefore from 0 to 255. The numbers following VDU calls must therefore conform to the same range. However, when we are dealing with graphics co-ordinates (as we have seen) the numbers involved greatly exceed the range for an eight bit number. At its simplest, we can say that putting a semicolon after a number in a VDU statement enables that number to be interpreted correctly even when it exceeds 255, but notice how the last number in the list is also followed by a semicolon - essential if that last value is to be correctly identified as well.

To treat the subject in a little more detail, what happens is this. A number followed in this context by a semicolon is treated as a two-byte number, and internally the number is broken down in this way. You can do it yourself. Simply divide the number by 256 using the MOD and DIV functions of Basic, and put the resulting two values as consecutive eight bit numbers. Thus our VDU24 example above could also have been written as:

```
VDU24,0,0,0,0,255,4,95,3
```

but then it's hardly an easy task to relate this to screen co-ordinates, and there's no point to it anyway, as Basic is quite happy to perform the task for us unasked.

Incidentally, there is one further call to be used in conjunction with VDU24 and VDU28, and that is VDU26 which resets both text and graphics windows to their default full-screen settings. When this happens, the flashing cursor is always moved to the top left-hand corner of the screen. As a result it is often desirable to follow with a screen clear function, most easily achieved with:

```
VDU26,12
```

which performs both tasks.

That's about all we have space for this month. That still leaves some quite important things to say about VDU codes which we will deal with next time, and we haven't touched upon FX calls at all as yet.

512 Forum

by Robin Burton

In this month's Forum we'll look at another query which has been raised by several members, another matter which previously has been avoided for reasons mentioned last month.

However, there are other 512 topics which have also been neglected, but for different reasons. I have one or two points to make about future Forum topics, in the hope that you'll make your views known to me in due course.

512 BBCBASIC

Many of you will by now have a copy of this software which, nearly two years ago, I offered free to members who sent a blank formatted disc with return post and packing (in total I've now supplied nearly 400 copies). This particular 'message' therefore applies to the rest of you - that is, those who for one reason or another didn't avail yourselves of the offer before.

Some of you will no doubt have The 512 Technical Guide, published by Dabs Press last September and reviewed recently in BEEBUG by Bernard Hill. Accompanying the book is an optional programs and examples disc containing numerous items which should be of interest to 512 users. One of the items on this disc is a copy of 512BBCBASIC.

I'm therefore giving notice that my offer of a free copy of 512BBCBASIC is finally closed. I've continued to receive occasional requests for it right up to the present, the latest only last week, but it seems to me that the offer, held open for two years, was surely long enough to be considered fair by everyone.

When the original offer was made, the package wasn't generally available from any source - but now it is. From now on, if you want a copy of Richard Russel's excellent BBCBASIC package for the 512 send your request for the 512 Technical Guide examples disc to Dabs Press with the appropriate fee.

VALID TOPICS

That leads me on nicely to another package supplied on the examples disc, the A86/D86 machine code assembler-debugger. Included on the disc are numerous examples of source machine code aimed at helping you to learn how to write your own programs. Clearly writing machine code isn't everyone's idea of having fun with their 512, and that means it's also perhaps not the most obvious subject for the Forum. However, the book and its disc do allow you to have a look at what's involved for a very low cost and apparently a number of you have been doing so. One result of this is that I've received a few more queries.

The reason for mentioning this is not that I think everyone should immediately take up machine code programming. It might widen the range of possible topics for the Forum, but my purpose is to make the point that a certain minimum amount of knowledge about the 512 and its internal workings is needed if you're going to understand the answers to some of your queries.

For those who have wondered why previous suggestions for Forum topics haven't appeared I can only say two things. Firstly space is limited, and secondly, as a consequence, the appeal of items should therefore be as wide as possible. This means that some subjects simply won't be covered unless there's evidence that enough of you are interested. To properly understand this month's main topic, for example, you need at least a basic idea of how the processor works. The technical guide and disc are a source of this information, but the Forum isn't because there isn't enough space.

I've had a few suggestions, virtually ever since the Forum began, that more technical topics should be included, at least some of the time. Previously I have mainly discussed particular but non-technical operational aspects of the system, so this mild 'complaint' is understandable. I hope therefore that the following redresses the balance a little. Naturally if you'd like more of this sort of thing (or less) I need to know, so drop me a line; the majority view will rule.

MYSTERIOUS MESSAGES?

I've had a few requests lately for a bit of information about one display I've never before mentioned in the Forum (but which unfortunately virtually everyone sees from time to time). It's an error in the form shown in figure 1. The questions are, what do all these figures mean and what use is the information?

```
Invalid Opcode Error from PROGRAM
Flags -I-Z-A-P-
  CS:IP    DS   ES    SS:SP    AX   BX   CX   DX   BP   SI   DI
1724:01A7 1724 1724 1724:FFEE 0000 0000 0000 0000 0000 8D92 0000
Program terminated
```

Figure 1. Fatal error message display

At the simplest level the meaning is obvious. A program instruction has been encountered which the 512 doesn't like for one reason or another, and the program has been abandoned.

The first point to make, perhaps, is that if this sort of display appears when you're running a program for which you have no information except operating instructions there's really not much you can do. However, if you have details about the program's structure and what it's doing, which you will have if you wrote it yourself, this display can often tell you exactly what's wrong and where it happened. Let's look at each part of the display in turn and examine it in a bit more detail.

The first line is straightforward, though there are other error messages I'll mention in a moment. This one shows 'Invalid Opcode' (by far the most common display) which defines the error type. In this case it means an 'illegal' or perhaps a nonsensical instruction code has been encountered.

There are numerous possible reasons for this, but a couple of the more obvious ones are that the program is corrupt, or it might perhaps have been written for a different processor. For example, a program written specifically for a '286 (or '386 or '486) based machine, or for those with a maths co-processor chip, will produce this sort of display very shortly after loading.

Another possibility, especially in your own code during development when errors are virtually unavoidable, is that your program has jumped to an incorrect address or has somehow corrupted its code. It might mean that the program is attempting to execute data as instructions, or perhaps that (originally) valid instructions have been destroyed by being overwritten in memory, both of which might give the above result. To illustrate this article, for example, I produced the sample display simply by copying a text file to a file called 'PROGRAM.COM' and executing it.

The next part of the line, 'Error from' is pre-coded, and the name of the program causing the trouble of course follows it as the last item on the line.

ERROR TYPES

I'll now try to explain (briefly) all the possible error types in this part of the display without getting too involved, but some of them will necessarily be pretty meaningless unless you've done a lot of serious '86 programming (and made some pretty serious errors). The full range of errors supported by the 512 in DOS Plus is:

Divide Error: means register overflow after a divide instruction. This error is most often caused by division by zero.

Single Step Int Error: means the trap flag was set. This is used by debuggers to trap certain conditions they otherwise could not handle. It's not a 'real' program error.

NMI Error: means a Non-maskable interrupt error has occurred. This is generally a hardware failure and is not recoverable. It is not a program error.

Break Point Error: an interrupt 3 was executed which allows the system to return control to a debugger in single step mode.

Overflow Error: an 'INTO' instruction has been executed, which causes an interrupt 4 if the overflow flag was set.

Range Error: means a 'bounds' instruction has failed.

Invalid Opcode Error: as explained above.

ESC Opcode Exception Error: in the 512 means an unknown or unexpected error condition has been passed from the host.

Register Dump Error: is the result of a deliberate interrupt 0ff, which causes a register dump.

Unsupported IBM XT ROS call Error: means a call has been made to an IBM PC internal function which isn't supported by the 512.

Unexpected Interrupt Error: is the least helpful of the lot because it covers anything that didn't crash the 512 and which isn't dealt with by one of the preceding messages.

FLAGS AND REGISTERS

The second line of the display, starting with 'Flags', indicates the contents of the processor's status register at the point of failure. In all processors, one register is set aside to record the resulting status from program operations. For example, after a test for zero, the zero flag is set if the condition was met. In our example display, as shown by the letter 'Z' in the middle of the flags, the zero-bit of the status register was set.

The next part of the display, the third and fourth lines, are read together and show the most useful information. You'll need to understand '86 memory addressing to appreciate the information, but if you do, this data tells you exactly where the problem occurred and the contents of the processor registers at the time.

I don't intend reproducing extracts of the technical guide here, so if you want to know how memory is addressed and what the 81086 processor registers are used for I can only direct you to chapter two of the book (it would in any case occupy about 5 complete issues of 512 Forum). Suffice it to say that the figures under CS:IP point to the failed instruction, where CS is the code segment and IP is the instruction pointer. Likewise, the figures under 'SS:SP' show the stack segment and stack pointer at the time, while the remaining parts of the display show the contents of the other registers at the point of failure.

THE PURPOSE

Obviously, given the error type and the code segment address of the failure, you will be able to see immediately if it was within your program. If so, you can tell precisely which instruction caused it.

On the other hand, if the code address is clearly outside your program area you can (usually) deduce quite easily if the instruction was within the operating system, in which case you have probably made an invalid interrupt call or have supplied incorrect parameters to one. If this doesn't seem likely the next thing to check is if the address looks completely meaningless. This usually indicates that corrupted code or data has resulted in an effectively random program jump, which could 'land' almost anywhere in memory. Whatever the case, one way or another the display gives information which helps a programmer to sort out what went wrong and where.

For the less technically inclined, or if you didn't write the program yourself, probably all that you need to know is that DOS Plus is telling you that the program in question isn't going to work and that the system has protected itself (and perhaps you and your data) by trapping the problem and ending the program. So long as you get a normal prompt back after the display you should be able to continue using the 512 without re-booting.

Of course, the 512's error trapping isn't perfect. Sometimes the operating system itself can get into knots, especially when an incorrect segment register value in a program accidentally changes some of DOS's working data or code by executing a mis-directed move or arithmetic instruction.

In this case you might see a non-stop repeated display, like the one above but continuously scrolling up the screen with no way to break in. Shift-Ctrl stops it temporarily, so you can note the figures. Alternatively, you may not get any prompt back after the display, but the other option, which I'm sure you've all seen, is a completely uninformative hang-up with no display at all. In all these cases the only option is to re-boot the system.

NEW IDEAS

Since the Forum started there has been a considerable change in support for Acorn micros. The 8-bit BBC micro continues to take more and more of a 'back-seat' in Acorn publications with the exception of BEEBUG. 512 users are a pretty independent bunch, they have to be, but I'd like to keep the Forum true to its title, so let me know the direction you'd like the Forum to take in the future. Don't just think 'It's been OK so far, I won't bother.' because one day I might run out of ideas. If you've ever thought, "Why doesn't he...?" why don't YOU tell me about it.

I've had some ideas for future Forums. One suggestion is a regular spot to investigate the more obscure DOS error messages. Another is a regular 'machine code corner', while another is to publish addresses for those who'd like to hear from other users. What do you think? If these or other ideas appeal write to me (c/o BEEBUG) saying what you'd like to see.

Practical Assembler (continued from page 37)

```
  810 INC zp2+1:INX:JMP loop
  820 .finxlat RTS
  830 \
  840 .newbgetv \ first get PTR# to X
  850 STX &100:LDA #0:LDX #zpg
  860 JSR &FFDA \osargs
  870 LDX zpg:JSR bget \ now decrypt
  880 EOR table,X:LDX &100:RTS
  890 \
  900 .bget JMP (oldbgetv)
  910 \
  920 .newbputv \ first get PTR#
  930 STA &100:STX &101:LDA #0:LDX #zpg
  940 JSR &FFDA \ osargs , then encrypt
  950 LDX zpg:LDA &100:EOR table,X
  960 LDX &101:JMP (oldbputv)
  970 \
  980 .print PLA:STA zpg:PLA: STA zpg+1
  990 LDY #0:.loop INC zpg:BNE P%+4
 1000 INC zpg+1:LDA (zpg),Y:CMP #&EA
 1010 BEQ prout:JSR &FFE3:JMP loop
 1020 .prout JMP (zpg)
 1030 .start EQUW 0
 1040 .end   EQUW 0
 1050 .len   EQUW 0
 1060 ]:NEXT
 1070 c$="SAVE ENC "+STR$~code+" "+STR$~
P%
 1080 PRINTc$:OSCLIc$
 1090 END
 1100 :
 2000 DEF FNjeq(a)
 2010 [OPT opt:BNE P%+5:JMP a:]:=""
 2020 :
 3000 DEF FNsetup(vloc,storeat,replace)
 3010 [OPT opt
 3020 EQUS FNcopy2(vloc,storeat)
 3030 LDA #replace MOD 256:STA vloc
 3040 LDA #replace DIV 256:STA vloc+1
 3050 ]:=""
 3060 :
 4000 DEF FNcopy2(a,b)
 4010 [OPT opt:LDA a:STA b
 4020 LDA a+1:STA b+1:]:=""
```

Data Compression (Part 2)

by Bernard Hill

The Huffman compression algorithm given last month was listed without the corresponding decoding algorithm. Listing 1 gives the extra lines which need to be added to last month's program to include the decoding to prove that the method is in fact working. In practice, however, the tree structure tables will need to be included with the compressed version of the file making this algorithm only really useful for long files where the table is a small fraction of the length of the file.

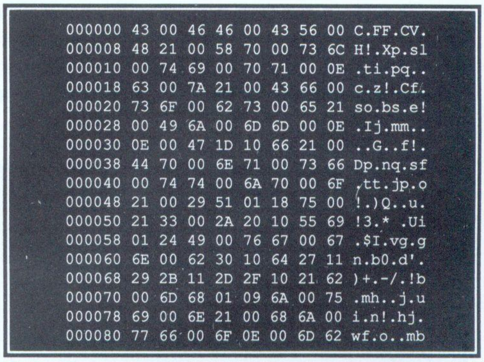
Note that once the arrays *code%* and *len%* are known, the decoding of the tree is very straightforward and appears in *PROCbuildtree*, *PROCbuildbranch* and *PROCdecode*. The actual tree is stored as a set of left and right pointers:

`DIM left(511), right(511)`

which point to other nodes or to ASCII characters when the value is negative.

LEMPEL-ZIV-WELCH COMPRESSION

Besides the need to transfer the tables in Huffman compression, another important disadvantage is that it requires two passes through the data in order to operate. In some circumstances this is impossible and an alternative is needed. The *Lempel-Ziv-Welch* compression method is used (often in modified form) in data transmission where its single-pass nature means that the bytes arriving in the algorithm can be translated in real time and their compressed forms sent instead. It is also partly used in the very fast shareware compression programs for the PC from PK-ware.



```
000000 43 00 46 46 00 43 56 00 C.FF.CV.
000008 48 21 00 58 70 00 73 6C H!.Xp.sl
000010 00 74 69 00 70 71 00 0E .ti.pq..
000018 63 00 7A 21 00 43 66 00 c.z!.Cf.
000020 73 6F 00 62 73 00 65 21 so.bs.e!
000028 00 49 6A 00 6D 6D 00 0E .Ij.mm..
000030 0E 00 47 1D 10 66 21 00 ..G..f!.
000038 44 70 00 6E 71 00 73 66 Dp.nq.sf
000040 00 74 74 00 6A 70 00 6F .tt.jp.o
000048 21 00 29 51 01 18 75 00 !.)Q..u.
000050 21 33 00 2A 20 10 55 69 !3.* .Ui
000058 01 24 49 00 76 67 00 67 .$I.vg.g
000060 6E 00 62 30 10 64 27 11 n.b0.d'.
000068 29 2B 11 2D 2F 10 21 62 )+.-/.!b
000070 00 6D 68 01 09 6A 00 75 .mh..j.u
000078 69 00 6E 21 00 68 6A 00 i.n!.hj.
000080 77 66 00 6F 0E 00 6D 62 wf.o..mb
```

The text from the start of this article encoded with LZW compression

Huffman encoding makes common letters consume less bits than uncommon ones. Lempel-Ziv-Welch uses constant-length codes (often 12-bits) and builds a table of commonly-used character sequences ('strings') based upon what has gone before. For 12-bit codes a table of 4096 elements is created and initially "seeded" so that the first 256 entries represent the bytes 0 to 255, i.e. single-character strings.

The operation of the algorithm is simple to grasp. Bytes entering the algorithm are formed into strings and the code for the longest match found in the table is sent to the output. For instance, if "A" and "AB" and "ABC" are contained in our table, then reading "ABCD" would result in the code for "ABC" being sent when the D is read. In this way output is only sent when a match fails, but the failure of the match means that a new string ("ABCD") will be added to the table for possible future use.

The seeding of the table ensures that the operation of finding entries in the table can start.

Suppose for example that the characters ABACBABA are to be compressed. When the first character is read it is not immediately sent but the next character also read to form a sequence: AB. At the beginning "AB" is not in the table, but "A" is, so the code for this, 65, is sent to the output. And the extension "AB", is added to the table at the current end, position 256. The same process occurs with the second letter "B" and its successor, "BA" is added to the table at position 257 and the "B" then output when the match fails. We can summarise this as follows:

READ IN	SENT	ADD TO TABLE
A		
B	65 (A)	AB (position 256)
A	66 (B)	BA (position 257)
C	256 (AB)	ABC (at 258)
B		CB (at 259)
A		
B	257 (BA)	BAB (at 260)
A		
(eof)	257 (BA)	

Now the storage of strings of uncertain length in the table is less of a problem than might be expected because each string is the extension by one character of a previous table entry. Thus the table is made up of entries which consist of an entry number and an extension character. Each element in the original seeding set would probably have a zero entry and their own ASCII character numbers, while the extensions would be as follows:

STRING TABLE STRUCTURE:

Position	Represents	Previous	Extn byte
0 to 255	ASCII set	0	+0 to 255
256	"AB"	65 (A)	+66 (B)
257	"BA"	66 (B)	+65 (A)
258	"ABC"	256 (AB)	+67 (C)

The short program in listing 2 is a simple program to perform Lempel-Ziv-Welch compression. The 4096-entry table is held internally as tab%, and an indirected array *char* and each coded character results in 1.5 bytes being sent to the output file. Thus the example above required 5 x 1.5 bytes, or 8: no usable compression since the input itself was 8 bytes, but clearly a saving will be made on larger files. Running listing 2 is excessively slow because of the very crude method used to find whether an entry exists in the table. A form of continual sorting and binary search would be preferable, but the program as given is reasonably simple to follow.

The method of decomposition of the code is very similar to the compression algorithm. The same table will be built up as the data is received and the seeding of the table with the full ASCII set means that the process can start. Just occasionally a problem occurs in that the extension of a tabulated string by its own first character forms a string not yet in the table, but this can be simply deduced and added to the table. The program in listing 3 decodes the file produced by listing 2.

IMPROVEMENTS

This algorithm is capable of a great deal of refinement. For instance, the version I have given here is very wasteful of table space. As the strings build up, most of them are unused and so when the table becomes full some form of clear-out of unused codes could be performed and more space generated. Also, commercial programs using this compression method often allow themselves to alter the limitation to 12 bits according to the nature of the data, and the algorithm is still the subject of much research and improvement.

Listing 1

```
   10 REM Program Huffman encoding
   20 REM additions to Huffman encoder
   30 REM in BEEBUG Vol.9 No.9
   40 :
  285 PROCprint
```

```
 6000 DEF PROCprint
 6010 PROCbuildtree
 6020 PROCdecode
 6030 ENDPROC
 6040 :
 7000 DEF PROCbuildtree
 7010 DIM left(511),right(511)
 7020 nnodes=0
 7030 FOR asc=0 TO 255
 7040 IF len%(asc)>0 THEN PROCbuildbranc
h(asc)
 7050 NEXT
 7060 ENDPROC
 7070 :
 8000 DEF PROCbuildbranch(a)
 8010 LOCAL node,d,b,next
 8020 IF len%(a)=1 THEN 8100
 8030 FOR d=len%(a)-1 TO 1 STEP -1
 8040 b=FNbit(code%(a),d)
 8050 IF b=0 THEN next=left(node) ELSE n
ext=right(node)
 8060 IF next=0 THEN nnodes=nnodes+1:nex
t=nnodes
 8070 IF b=0 THEN left(node)=next ELSE r
ight(node)=next
 8080 node=next
 8090 NEXT
 8100 b=FNbit(code%(a),0)
 8110 IF b=0 THEN left(node)=-a ELSE rig
ht(node)=-a
 8120 ENDPROC
 8130 :
 9000 DEF PROCdecode
 9010 PRINT"Decoded message:"
 9020 f=OPENINm$
 9030 node=0:in=0
 9040 REPEAT
 9050 b=FNgetbit
 9060 IF b=0 THEN node=left(node) ELSE n
ode=right(node)
 9070 IF node<0 THEN PROCoutput(-node):n
ode=0
 9080 UNTIL in=T%
 9090 CLOSE#f
 9100 ENDPROC
 9110 :
10000 DEF PROCoutput(val):LOCAL @%
10010 IF val<>13 AND (val<32 OR val>126)
 THEN PRINT"<";val;">";:ENDPROC
10020 VDU val:IF val=13 THEN VDU10
10030 ENDPROC
10040 :
11000 DEF FNgetbit:LOCAL x
11010 IF in MOD 8=0 THEN buf=BGET#f
11020 x=(7-in MOD 8):in=in+1:=FNbit(buf,
x)
11030 :
12000 DEF FNbit(val,bit)
12010 IF (val AND (2^bit))=0 THEN =0 ELS
E =1
```

Listing 2

```
  10 REM Program ZLW Compression
  20 REM Version B1.0
  30 REM Author  Bernard Hill
  40 REM BEEBUG  April 1991
  50 REM Program subject to copyright
  60 :
 100 see=TRUE:REM to see input on scrn
 110 maxtable=4095
 120 DIM tab%(maxtable),char maxtable
 130 INPUT"Input file: "f$
 140 f=OPENINf$
 150 IF f=0 THEN PRINT"Not found":END
 160 INPUT"Output file: "f$
 170 g=OPENOUTf$
 180 REM seed table
 190 FOR i=1 TO 256:char?i=i-1:NEXT
 200 tabsize=256
 210 m=0:count=0:even=TRUE
 220 REPEAT
 230 b=BGET#f
 240 IF see THEN VDUb:IF b=13 THEN VDU1
0
 250 n=FNfind(m,b)
 260 IF n>0 THEN m=n ELSE PROCadd(m,b):
PROCoutput(m):count=count+1:m=FNfind(0,b
)
 270 UNTIL EOF#f
 280 IF n>0 THEN PROCoutput(m) ELSE PRO
Coutput(FNfind(0,b))
 281 PROCoutput(0)
 300 IF NOT even THEN BPUT#g,0
 310 PRINT'"Compression :"1.5*(count+1)
/EXT#f
 320 CLOSE#f:CLOSE#g
```

```
  330 END
  340 :
 1000 DEF FNfind(n,b)
 1010 LOCAL c%,found:c%=1
 1020 REPEAT
 1030 found=char?c%=b AND tab%(c%)=n
 1040 c%=c%+1
 1050 UNTIL found OR c%>tabsize
 1060 IF found THEN =c%-1 ELSE =0
 1070 :
 2000 DEF PROCadd(n,b)
 2010 IF tabsize=maxtable THEN ENDPROC
 2020 tabsize=tabsize+1
 2030 tab%(tabsize)=n
 2040 char?tabsize=b
 2050 ENDPROC
 2060 :
 3000 DEF PROCoutput(n)
 3010 REM outputs 12 bits for one n
 3020 even=NOT even
 3030 IF NOT even THEN BPUT#g,n MOD 256:
save=n DIV 256:ENDPROC
 3040 BPUT#g,16*save+n DIV 256
 3050 BPUT#g,n MOD 256
 3060 ENDPROC
```

Listing 3

```
   10 REM Program UNLZW - LZW decompress
ion
   20 REM Version 1.0
   30 REM Author  Bernard Hill
   40 REM Beebug  April 1991
   50 REM Program subject to copyright
   60 :
  100 see=TRUE
  110 maxtable=4095
  120 DIM tab%(maxtable),char maxtable
  130 INPUT"Input file: "f$
  140 f=OPENINf$
  150 IF f=0 THEN PRINT"Not found":END
  160 INPUT"Output file: "f$
  170 g=OPENOUTf$
  180 REM init table
  190 FOR i=1 TO 256:char?i=i-1:NEXT
  200 tabsize=256
  210 m=0:even=TRUE
  220 :
  230 REPEAT
  240 x=FNinput:IF x=0 THEN 320
  250 IF x<=tabsize THEN out$="":PROCout
(x) ELSE out$=out$+LEFT$(out$,1):PROCadd
(lastx,ASCLEFT$(out$,1))
  260 FOR i=1 TO LENout$
  270 b=ASCMID$(out$,i,1)
  280 BPUT#g,b
  290 IF see THEN VDU b:IF b=13 THEN VDU
10
  300 NEXT
  310 lastx=x
  320 UNTIL EOF#f OR x=0
  330 CLOSE#f:CLOSE#g
  340 END
  350 :
 1000 DEF PROCadd(n,b)
 1010 IF tabsize=maxtable THEN ENDPROC
 1020 tabsize=tabsize+1
 1030 tab%(tabsize)=n
 1040 char?tabsize=b
 1050 ENDPROC
 1060 :
 2000 DEF FNinput
 2010 REM 12-bit input
 2020 LOCAL b,s
 2030 even=NOT even
 2040 IF even THEN =256*save+BGET#f
 2050 b=BGET#f:s=BGET#f
 2060 save=s MOD 16:=256*(s DIV 16)+b
 2070 :
 3000 DEF PROCout(x)
 3010 LOCAL n,b
 3020 n=tab%(x)
 3030 IF n>0 THEN PROCout(n)
 3040 b=char?x
 3050 out$=out$+CHR$b
 3060 n=FNfind(m,b)
 3070 IF n>0 THEN m=n ELSE PROCadd(m,b):
m=FNfind(0,b)
 3080 ENDPROC
 3090 :
 4000 DEF FNfind(n,b)
 4010 LOCAL c%,found:c%=1
 4020 REPEAT
 4030 found=char?c%=b AND tab%(c%)=n
 4040 c%=c%+1
 4050 UNTIL found OR c%>tabsize
 4060 IF found THEN =c%-1 ELSE =0
```

Mastering Edit (Part 3)

by Mike Williams

I little expected when I started writing this short series of articles that it would extend to three instalments, and I am sure that there is plenty more still to be said about other features of Edit. I wanted to concentrate upon the search and replace facilities, which I have done, and as a result I have been roundly taken to task by one reader for not giving adequate acknowledgement to the format and layout capabilities of Edit. Well, I have to plead guilty here; this was not the focal point of the series anyway, and I must confess that this aspect of Edit is not one I happen to use myself (as some may know I prefer View for genuine word processing), but the facilities available are quite comprehensive, as my correspondent pointed out. Maybe this would form the subject of a further series on Edit.

SINGLE CHARACTER SPECIFIERS

In dealing with Edit's search and replace commands, we have concentrated on single characters, or on sequences of single characters. For example:

```
fred/john
```

will replace all occurrences of fred by john (remember that a search is initiated by pressing f4 for a selective search (and optionally replace), or f5 for a global search and replace).

It is when we start to investigate the use of ambiguous character specifiers and variable length strings that the true power and flexibility of Edit is revealed. We have already seen that '@' can be used to match any alphabetic character, and '#' any digit. In addition, '.' will match any character in the range 0-255.

Ranges of characters can also be specified. For example:

```
A-Z
```

would search for any character in the range given, i.e. upper case characters. Searching for:

```
A-Z%
```

Note that in this month's examples of search and replace strings, the space character (obtained by pressing the space bar) will be represented as before by <space>.

would search for any such letter followed by a '%' symbol, useful to locate all occurrences of the built-in variables A% to Z% in a Basic program. Note the format of the search syntax. You might have thought it should be:

```
A%-Z%
```

but this doesn't make sense (Edit would look for a string starting with a letter 'A' or 'a' - as Edit is not case specific on single letters - followed by a character in the range '%-Z' - ASCII codes 37 to 90 - followed by a '%'). Using Edit to the full, you need to develop the ability to analyse correctly the syntax of a search string.

It is also possible to specify alternatives to be searched for, any one of which is to be matched, and it is here that Reference Manual Part 2 is quite incorrect (at least in my copy). Alternatives *MUST* be enclosed in square brackets. To match any one of '1', '3', or 'c' the search string must be given as:

```
[13c]
```

and not as:

```
13c
```

as in the manual (the latter would search for the three-character string '13c'. Thus to search a program for A%, B% or C% we could specify:

```
[ABC]%
```

as the search string.

It is important to realise that when specifying a range of characters, or alternative characters, each match will be with just a single character; in effect we can term these *single character elements* in that they function like any specific single character in a search string. Thus:

```
[A-Za-z]%
```

would match any two-character string consisting of an upper case or a lower case character followed by a '%', i.e. it would match A% to Z% or a% to z%.

One further useful feature with single character elements is the *negation* symbol '~'. Thus to search for a character in the range A to Z you would specify:

```
A-Z
```

but to specify any character not in that range you would write:

```
~A-Z
```

MULTIPLE CHARACTER SPECIFIERS

Sometimes the string for which you are searching will vary in length, for example the line numbers in a Basic program. Some will be two digits, some three, some four or more. Suppose you want to search for all line numbers with a space separating the line number from the following instruction (in order to remove the space as I suggested last time). If we take two digit line numbers only then we might specify:

```
##<space>
```

But then we have instructed Edit to search for all occurrences of two digits followed by a space. This will also pick up the last two digits of all line numbers of three digits or more, and probably many other numbers besides. First of all let's see how we can construct the search to ensure we find all numbers regardless of purpose or length, and then refine this to isolate just line numbers and then just line numbers followed by a space.

To do this we need to introduce the function of two more special characters. Preceding a character with a '^' produces a search for as many of that character as possible. Thus specifying:

```
^#
```

will find strings of digits of any length. The other special character is '*' which matches as few of the following character as possible (more of this in a moment). To try to distinguish line numbers we can note that these appear always at the start of a line, and are thus preceded by a carriage return (represented by a '$'). So we can refine our search string to:

```
$^#
```

In fact, in Edit, shorter line numbers are preceded by one or more spaces so we need a further refinement of the search string:

```
$*<space>^#
```

This can be interpreted as "Return, followed by as few spaces as possible, followed by as many digits as possible". It is interesting to consider too the difference between that and the search specification:

```
$^<space>^#
```

This would read as "Return, followed by as many spaces as possible, followed by as many digits as possible", superficially perhaps not very different. Although in a sense this is quite true, the difference can certainly be significant. When searching for "as few as possible" of some character, the minimal match is "nothing". Thus searching for as few spaces as possible will match no spaces, one space, two spaces and so on. On the other hand, searching for "as many as possible" needs to find at least one of the target characters to form a match.

Thus the first search specification given above would match line numbers such as:

```
   10
  100
 1000
10000
```

but the second would not match the last of these. In the same context, it is important to specify "as many digits as possible" so that at least one digit is required for a match.

This all takes longer to explain than it does to carry out, but even so the distinction between the meanings of the two special characters '^' and '*' may still confuse you as it still does me from time to time. If in doubt, as with all searches, try just the search only (no replace) and make it a selective rather than a global search. Once you are confident you have constructed the correct syntax, then trust yourself to a global search and replace.

Despite all this discussion we have still not completed the task we set ourselves. To search reasonably for all line numbers followed by a space you will need to specify:

```
$*<space>^#<space>
```

That's fine, but how do we now set about removing that last space. In effect what we need to to is to replace the string found by the same string less its final character the space. This can be achieved by looking at two further special characters, '&' and '%' this time in the replacement specification. The '&' character represents the complete target string. Thus:

```
$*<space>^#<space>/&
```

would certainly find line numbers followed by spaces, but would replace the target by itself, not very useful, though in other contexts the '&' character certainly has a part to play. For example, if we are trying to achieve the opposite of our original task, to insert a space after every line number we could write:

```
$*<space>^#/&<space>
```

where the target string is replaced by the same string followed by a space (but note that this specification would apply that to all line numbers including those already followed by a space - for a refinement on this see later).

Now to return to our original task we need to use the other special character '%' to select just part of the target string. The '%' character, in this context, is followed by a single digit. This is used to identify the different *ambiguous* elements of the search string - after all, unambiguous elements are known and can simply be repeated. In our example the first ambiguous part is:

```
*<space>
```

and this is represented by %0 in the replacement string. The second (in our case) ambiguous element is:

```
^#
```

and this would be represented by %1. Any further ambiguous elements would be represented by %2, %3 and so on. Thus the replace string will consist of Return (indicated by a '$'), followed by %0 followed by %1 (but not including the space), thus:

```
$*<space>^#<space>/$%0%1
```

At this stage, I strongly advise you to try this out for yourself by constructing a short if artificial example. By the way, if you want to insert a space following all line numbers where such a space is missing use:

```
$*<space>^#~<space>/$%0%1<space>%2
```

Here, we search for line numbers followed by any character other than a 'space' (i.e. not a space), but the replacement, as well as including the extra space needed must also put back the last character of the target string (using %2) as this will have been the first character after the line number (in other words, the start of an instruction).

The final examples which we have constructed have become quite complex, but are in fact quite useful in practice for tidying up programs. They will either insert a space, following a line number, where none existed before, or remove a space following a line number where one existed previously. Think how you might modify that one to remove not just a single following space, but as many spaces as there might happen to be, very useful for a program saved in a LISTO format.

One final point, to conclude this month's article: the last two examples in section 'R' of the *Reference Manual Part Two* (pages R.15-5 and R.15-6) both have pairs of square brackets missing - see now if you can work out where they should go.

Edit is a powerful editor, and in consequence requires some study to use it to the full. As with most things, practice does help, but either check what you are attempting on some small example constructed for the purpose, or be cautious before using a full global search and replace on live data. There is much, much more to Edit (as indeed I have discovered in the course of writing these three articles), and I hope to return to the subject again in the not too distant future. In the meantime any feedback or comment from readers, particularly other examples of useful search and replace strings, will be most welcome.

ADFS/View Utility for the Master (Part 2)

Jeff Gorman concludes his discussion on how to organise your word processing with View on a BBC Master.

This final instalment of PreView detects errors, offers opportunities to confirm decisions and improves the on-screen presentation. A further short program MultiPt, which prints multiple copies of texts, is also included.

Before attempting to combine the two parts of the program, it would be prudent to make sure that the first instalment is working. If it is certain that Preview_1 has been correctly entered, it will be much easier to track down possible errors in PreView_2. When entering PreView_2, place a REM before each of the lines containing "ON ERROR GOTO", excepting line 300. When merged, any typing errors will then be reported by Basic.

Enter and save this month's listing, making certain that the line numbers appear as listed. Save as PreView_2 and also prepare one or two backups. Likewise create backups of PreView_1. It will not be possible, other than by line-by-line comparison of the listings, to verify PreView_2 until the two parts are combined into a single program (PreView) and run. A mix-up could be disastrous if no backups were available.

To combine the two parts, load PreView_2 and type the following, pressing Return at the end of each line, of course:

```
LISTO1
WIDTH0
*SPOOL $.SpoolView
LIST
```

The listing will be saved to a file on disc as it appears on the screen.

```
*SPOOL
```

This closes the file.

```
LOAD "$.PreView_1"
*EXEC $.SpoolView
```

This has the same effect as would line-by-line typing of PreView_2 from Basic's line editor with PreView already loaded.

```
SAVE "$.PreView"
```

If all is well, the !Boot file described last month, will run the complete version of PreView, provided that:

```
CH.$.PreView_1
```

is changed to:

```
CH.$.PreView
```

When typing errors, if any, have been remedied, remove the REMs from the ON ERROR GOTO lines referred to previously to check the working of the Escape routines.

Listing 2 is the MultiPt program, which should also be typed in and saved in the $ directory with this name.

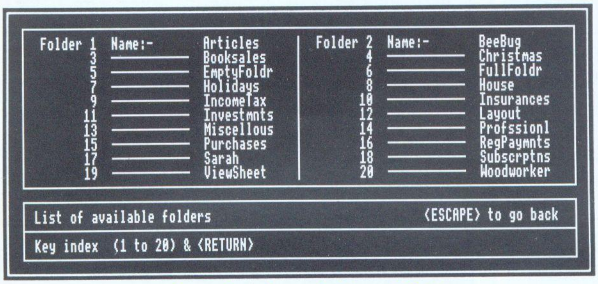
MULTIPLE COPIES

The program $.MultiPt is called by f3 which first surrenders View to Basic. There is a certain amount of disc activity in setting up the routine, and therefore some delay, so this utility is of greatest value if more than just a few copies are required. All that happens is that a file called *$.Multi*, containing the word 'Print' repeated the requisite number of times, is spooled to disc, the screen being meanwhile temporarily blanked by *FX3,10. F9 is then loaded with a command which re-loads the text file into View, taking the file name from location &B00 et seq. where it was temporarily stored by line 2030 of PreView. The command **EXEC $.Multi* has the effect of simulating repeated inputs of the command 'Print' from the keyboard.

The *FX138,0,n calls simulate the effect of depressing a function key, n being determined by adding 128 to the required key number. Although the *FX138 calls are issued before control returns to View, they take effect only after View takes over. Their effect is to load the printer driver, issue the commands stored in f9, and finally return to PreView.

MANAGING PREVIEW

Access to texts can be fairly swift since the list of texts in the last-used folder is stored in sideways RAM. If work frequently involves selecting from a variety of folders, it might be advisable to use many folders, each containing less than a full complement of texts, since changing folders does involve, at each change, some little delay if long lists are read from the disc.



```
Folder 1  Name:-  ──────── Articles     Folder 2  Name:-  ──────── BeeBug
       3          ──────── Booksales           4          ──────── Christmas
       5          ──────── EmptyFoldr          6          ──────── FullFoldr
       7          ──────── Holidays            8          ──────── House
       9          ──────── IncomeTax          10          ──────── Insurances
      11          ──────── Investmnts         12          ──────── Layout
      13          ──────── Miscellous         14          ──────── Profssionl
      15          ──────── Purchases          16          ──────── RegPaymnts
      17          ──────── Sarah              18          ──────── Subscrptns
      19          ──────── ViewSheet          20          ──────── Woodworker

List of available folders                                <ESCAPE> to go back
Key index  (1 to 20) & <RETURN>
```

PreView showing choice of folders

PROGRAM OUTLINE

Lines 150, 160, 170 and 1080 control the storage of PreView in sideways RAM. *SRData4 and *SRData5 reserve two banks of RAM as a continuous entity. *SRWrite simply copies the area of program memory starting at &E00 plus an allowance of &3500 bytes for the program, into sideways RAM bank 5.

Function key f4 is loaded with the command:

```
    "*Basic|M *SRRead E00+3500 3FFF|M
OLD|M RUN|M"
```

which restores Basic, reads the code back into memory, again starting at &E00, performs 'OLD' and runs the code (note that '|M' simulates pressing the Return key).

At line 130, the instruction *DIM txt% 78, fdr% 10* asks Basic to reserve 78 and 10 bytes of memory at suitable locations which Basic itself selects. Essentially, the program then puts, one at a time, folder names and text or layout names into these locations using the string indirection operators *$fdr%* and *$txt%*.

Repeated use of *SRWRITE in PROCst() copies the contents of these locations into sideways RAM (SWR). The actual address in SWR of each string is decided, in the case of folder names, by starting at an address at the beginning of the SWR area with the first file name and then adding ten to this address for each file name entered.

Layouts are stored in a similar way, at intervals of 78 bytes, but at a start address 500 bytes higher in the same bank of SWR, by means of *offset%* set at 500. Text file names and their extensions are stored higher still at a starting place indicated by an offset of 4200 bytes.

FNfetch() reverses this operation by using *SRREAD to take required information from SWR and temporarily pop it into the same locations ready to be read by the program.

RdDirs() uses two OSGBPB (Operating System Get Byte Put Byte) calls to read both the disc title (A%=5) or names of files and numbers of files in each directory (A%=8), and to store the numbers safely, using string indirection operators $&D92,$&D94 and $&D94, in areas of memory safe from corruption during the transfer of control to the View program.

FNfixBox and *PROCstretchBox* are designed to display the lists of file descriptions, whatever their length, with their bases always just above the rubric panel at the foot of the screen.

PROCseeFnme offers the option to check file names of existing files prior to determining a suitable name for the file about to be created. This can be useful if file names are determined with a view to classifying texts, perhaps by some existing code number such as an order number or membership number.

FNspaceUsed uses OSWORD &71 to read the free space on the disc.

Listing 1

```
  10 REM Program PreView_2
  20 REM Version 1.2
  30 REM Author  Jeff Gorman
  40 REM BEEBUG  April 1991
  50 REM Program subject to copyright
  60 :
 190 PROCrbrcBox
1070 PROCsetBox
1230 PRINT ".";
1370 PROCmsg(28,2,"Establishing indexes
 - "+wt$,0)
 1650 IF ?&D98=0 p$="Preparing " ELSE p$
="Updating "
 1660 PROCmsg(30,2,p$+a$+" index"+STRING
$(7," "),0)
 1710 ON ERROR GOTO 1810
 1720 ?&D98=0:LOCAL k$:PROCwnd:CLS:PROCs
howMnu:PROCmsg(28,2,"Disc Title:- "+$&B1
0,0)
 1730 PROCdskStatus
 1810 PROCwnd:CLS:PROCmsg(28,2,"Key f4 t
o re-start ""PreView""",0):PROCmsg(30,2,
"",0):END
 1811 :
 1830 PROCwnd
 1840 PROCstretchBx(22,3,-1,0)
 1850 :
 1910 PROCseeFnme
 1950 PROCmsg(28,2,"Layout routine"+STRI
NG$(41," ")+"<ESCAPE> to go back",0)
 2020 PROCchkFile(sln$)
 2100 PROCmsg(28,2,"List of available fo
lders"+STRING$(30," ")+"<ESCAPE> to go b
ack",0)
 2120 IF NOT lyt% PROCmsg(30,2,"Opening
folder """+dir$+""""+STRING$(3," "),0)
 2160 :
 2180 PROCmsg(30,2,wt$,0)
 2230 ON ERROR GOTO 2350
 2260 IF NOT FNconfirm("selection of fol
der """+dir$+""""):PROCpickFold
 2290 IF useLyt% AND FNmistake("Folder "
""+dir$+""" is full",chkNo%>=47):esc%=TR
UE:PROCpickFold:ENDPROC
 2320 PROCmsg(30,2,wt$,0)
 2350 PROCmenu:ENDPROC
 2351 :
 2430 :
 2440 IF NOT reRun% PROCdrawBox
 2510 ON ERROR GOTO 2570
 2570 esc%=TRUE:PROCmenu:ENDPROC
 2571 :
 2620 IF FNmistake("Bad key",(NOT seeFnm
e% AND (i$<>"M" AND VALi$=0 OR VALi$>lmt
%)) OR i$="" OR (seeFnme% AND i$<>"F" AN
D i$<>"M") OR LENi$>2):=FNindex(ky$,lmt%
)
 2730 ON ERROR GOTO 2780
 2750 PROCchkName(rep$)
 2780 PROClytRtne:ENDPROC
 2781 :
 2850 PROCstretchBx(y%,n%,0,-1)
 3010 PROCmsg(28,2,"Preparing VIEW Progr
am",1)
 3020 PROCmsg(30,2,"Use f4 from the VIEW
 Command Screen to re-load ""PreView""",
1)
 3080 OSCLI("Key3 *Basic|M CH.""$.MultiP
t""|M")
 3350 :
 3360 DEF FNyn:REPEAT
 3370 key=GET AND &DF:key$=CHR$(key)
 3380 UNTIL INSTR("YN",key$):*FX21,0
 3390 =key$
 3400 :
 3410 DEF FNconfirm(action$)
 3420 PROCmsg(30,2,"Confirm "+action$+"
[Y/N]? ",0)
 3430 =(FNyn="Y")
 3440 :
 3450 DEF PROCchkFile(sln$)
 3460 IF FNconfirm(sln$) ENDPROC
 3470 reRun%=0
 3480 IF lyt% PROClytRtne ELSE PROCtxtRt
ne
 3490 ENDPROC
 3500 :
 3510 DEF PROCchkName(label$):ovr$=""
 3520 PROCmsg(30,2,"Checking for duplica
tion of """+label$+""" - "+wt$,0)
 3530 H%=OPENUP(label$):CLOSE#H%
 3540 IF FNmistake(""""+label$+""" alrea
dy exists",H%>0 ) ovr$=FNover
 3550 IF ovr$="N" PROCmakeFnme
 3560 ENDPROC
 3570 :
 3580 DEF FNover:CLS
 3590 PROCmsg(30,2,"Overwrite the file [
Y/N]? ",0)
 3600 =FNyn
 3610 :
 3620 DEF FNmistake(mistake$,test)
 3630 IF NOT test:=0
 3640 IF test PROCmsg(30,2,"",0)
 3650 VDU7:PROCcont(mistake$+" - "):=tes
t
 3660 :
 3670 DEF PROCdskStatus:used%=FNspaceUse
d
 3680 used$="Disc "+LEFT$(STR$(used%),2)
```

```
+"% used - Approx. "+STR$pages%+" A4 pag
es free"
 3690 PRINT TAB(33,0) used$;
 3700 check%=FNmistake("File managment o
r new disc advised - ",used%>=95)
 3710 ENDPROC
 3720 :
 3730 DEF FNspaceUsed
 3740 A%=&71:X%=&70:Y%=0
 3750 CALL &FFF1:free%=!&70:full%=655360
 3760 pages%=free%/4000
 3770 =((full%-free%)/full%)*100
 3780 :
 3790 DEF PROCseeFnme
 3810 ON ERROR GOTO 3870
 3820 IF nTxt%=0 PROCmakeFnme:ENDPROC
 3830 PROCmsg(30,2,"Check existing filen
ames in """+dir$+""" [Y/N] ? ",0)
 3840 IF FNyn="Y" PROCshowFiles:ENDPROC
 3850 PROCmakeFnme:ENDPROC
 3860 :
 3870 PROClytRtne:ENDPROC
 3871 :
 3880 DEF PROCshowFiles
 3890 seeFnme%=TRUE
 3900 IF NOT inRam% PROCsize
 3910 PROCmsg(28,2,""""+dir$+""" texts l
isted - For reference only ",0)
 3920 PROCmsg(30,2,"Loading """+dir$+"""
 ",0)
 3930 PROCgetTxts:ENDPROC
 3940 :
 3950 DEF PROCrbrcBox:VDU28,0,31,79,27,1
2
 3960 PRINT top$'side$'bar$'side$'botm$;
 3970 ENDPROC
 3980 :
 3990 DEF PROCsetBox
 4000 void$ =STRING$(77," ")
 4010 horz$ =STRING$(77,CHR$166)
 4020 top$ = CHR$163+horz$+CHR$165
 4030 side$= CHR$169+void$+CHR$169
 4040 bar$=  CHR$171+horz$+CHR$173
 4050 botm$= CHR$170+horz$+CHR$172
 4060 ENDPROC
 4070 :
 4080 DEF PROCdrawBox
 4090 PROCcsr(0):y%=FNfixTxtBox(stp%+1)
 4100 PROCstretchBx(y%,stp%,0,-1):ENDPRO
C
 4110 :
 4120 DEF PROCstretchBx(y%,stop%,heading
,clr)
 4130 PROCcsr(0):PROCwnd:IF clr CLS
 4140 PRINT TAB(0,y%) top$
 4150 FOR box%=1 TO stop%:PRINT side$:NE
XT
 4160 PRINT botm$;:VDU28,1,y%+stop%,79,y
%+1
 4170 ENDPROC
```

Listing 2

```
   10 REM Program MultiPt
   20 REM Version B1.1
   30 REM Author  Jeff Gorman
   40 REM BEEBUG  April 1991
   50 Program subject to copyright
   60 :
  100 ON ERROR GOTO 150
  110 PROCquant:PROCspool:PROCprint
  120 *WORD
  130 END
  140 :
  150 REPORT:PRINT" at line "ERL:END
  160 :
 1000 DEF PROCquant
 1010 CLS:PRINT TAB(10,10)"Enter number
of copies required .....";
 1020 INPUT""quant%:IF quant%<=0 PROCqua
nt
 1030 CLS:PRINT TAB(10,12)"Preparing to
print ";quant%;" copies - Please wait."
 1040 ENDPROC
 1050 :
 1060 DEF PROCspool
 1070 *SPOOL "$.Multi"
 1080 *FX3,2
 1090 FOR loop%=1 TO quant%
 1100 PRINT "Print":NEXT
 1110 *SPOOL
 1120 *FX3,0
 1130 ENDPROC
 1140 :
 1150 DEF PROCprint
 1160 OSCLI("KEY9 Load "+$&B00+"|M*Exec
$.Multi|M")
 1170 *FX138,0,130
 1180 *FX138,0,137
 1190 *FX138,0,132
 1200 ENDPROC
```

RISC USER

The Archimedes Magazine & Support Group

Now in its fourth year of publication, Risc User continues to enjoy the largest circulation of any subscription magazine devoted solely to the Archimedes range of computers. It provides support for all Archimedes users at work (schools, colleges, universities, industry, government establishments) and at home.

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 to enthusiasts and professionals at all levels.

And here are some items covered in the most recent issues of RISC User:

STICKY BACKDROP
A very versatile implementation of the sticky backdrop concept, which allows you to stick file and directory icons onto the Desktop for quick access. The disc includes the C source file, and a sprite which can be used as a backdrop.

QUITE A REVELATION
A review of Revelation, the sophisticated image processing application from Longman Logotron.

ARCHWAY 2
A look into Simtron's Wimp program development tool.

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

FIRST LOOK AT BEZIER CURVES
Two-part series on this mathematical method for drawing smooth curves and surfaces.

MULTI-TASKING WIMP DEBUGGER
A multi-tasking application which allows you to debug Wimp based programs while testing and running them.

MAGPIE
A review of this excellent information organiser from Longman Logotron.

PROGRAMMER'S WORKSHOP
A major series covering all advanced aspects of the Archimedes and incorporating the Assembler Workshop. This month: an ADFS disc examiner.

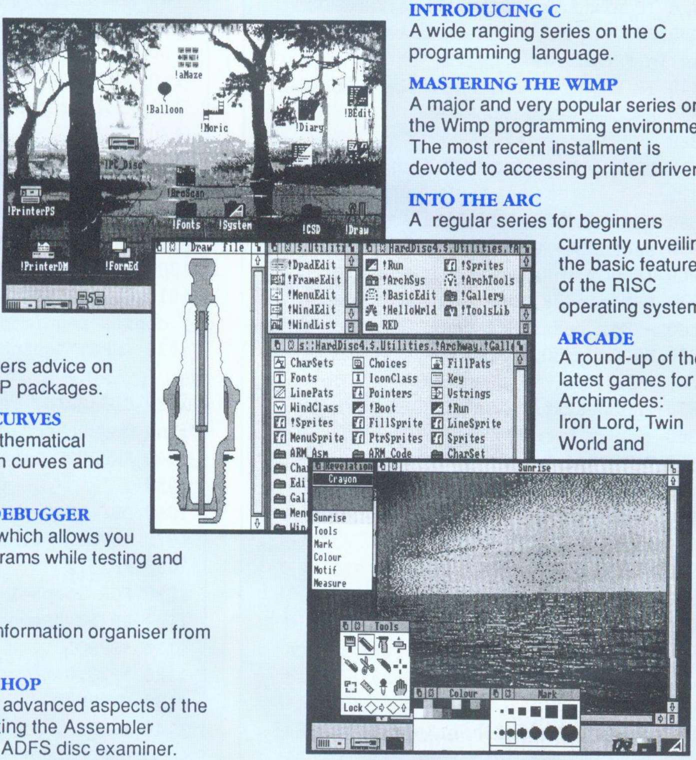
DXF - A DATA INTERCHANGE FILE FORMAT
An article about converting graphic images to Arc formats, and importing DXF files into Draw.

INTRODUCING C
A wide ranging series on the C programming language.

MASTERING THE WIMP
A major and very popular series on the Wimp programming environment. The most recent installment is devoted to accessing printer drivers.

INTO THE ARC
A regular series for beginners currently unveiling the basic features of the RISC operating system.

ARCADE
A round-up of the latest games for the Archimedes: Iron Lord, Twin World and



As a member of BEEBUG you may extend your subscription to include RISC User for only **£9.10** (overseas see table).

Phone your instructions in, or send a cheque/postal order to the address below. Please quote your **name** and **membership number.** When ordering by Access, Visa or Connect, please quote your card number and the expiry date.

SUBSCRIPTION DETAILS

Destination	Additional Cost
UK,BFPO &Ch Is	£ 9.10
Rest of Europe and Eire	£14.00
Middle East	£17.00
Americas and Africa	£19.00
Elsewhere	£20.00

RISC User, 117 Hatfield Road, St Albans, Herts AL1 4JS, Telephone (0727) 40303, FAX (0727) 860263

EXTENDED SEARCHES IN VIEW

Andrew Benham

An apparently undocumented feature of the View word processor is that it is perfectly possible to search for certain 'special' characters when using a SEARCH, CHANGE or REPLACE command by prefixing the appropriate letter with a '^'. The combinations available are:

^?	single character 'wild card'
^T	Tab
^C	Return
^S	Space
^Z	Soft space
^-	Highlight 1
^*	Highlight 2
^L	Left margin marker

In the above, a 'soft' space is one inserted by View when justifying text. For example:

```
        CHANGE ^S^C ^C
```

will remove any trailing spaces from lines (and tell you how many substitutions have been made). Repeating the command until no further substitutions are made will ensure that ALL trailing spaces have been removed.

```
        CHANGE ^S^S ^S
```

will replace two consecutive space characters by a single space. Again this can be repeated until no further occurrences of double spaces remain.

```
        CHANGE ^Z ^S
```

will replace all of View's soft spaces with normal hard spaces. This can be useful when exporting a View file to a PC as View uses Ctrl-Z for soft spaces, but to a PC Ctrl-Z means 'End-of-file' marker.

BASIC PROGRAM QUICK SAVE

M.F.Park

To provide a quick means of saving any Basic program, program function key fn as follows:

```
        *KEYn SA.$(PA.+5)|M
```

Start the first line of each program with a REM followed by the filename which may be enclosed between double quotes or spaces. Other information can follow the second space (if used) in the same REM statement, for example:

```
        10 REM MyFile Trials of *Key4
```

Then just press the programmed function key to save a program. Programs with overlong names generate a "Bad name" error and fail to save. Include the key definition in your !Boot files, and you will find it easy to save your work regularly.

BBC V MASTER PRINTOUT

J.R.Barker

There are some subtle and overlooked differences between the ways in which a model B and a Master 128 control the sending of characters to a printer.

*FX6 can be used to specify an ASCII character which will never be sent to a printer. With auto-linefeed on the printer 'off', a program must execute *FX6,0 to ensure that a linefeed character is sent to the printer when required. The consequence is that any zero bytes in the data stream sent to the printer will be ignored, causing incorrect output. This is most likely to occur when printing graphics.

The Master 128 can be configured for 'No Ignore' and in consequence all characters will be sent to the printer. However, if the printer is set to auto-linefeed, then the Master must be configured to ignore ASCII 10, and then every '10' occurring in data sent to the printer will be ignored.

NEW MASTER ROM UPDATE

Andrew Benham

Acorn's new MOS for the Master contains a new version of Basic. This means that ROMs which directly access routines in Basic will need updating. Users of BEEBUG's EXMON will need to obtain version 2.02, for example (by returning the old ROM together with payment of £5.00 plus 60p p&p quoting stock code 6666).

The new ROM includes an ADFS verifier, but the hard disc verifier assumes an Adaptec hard disc controller. On some hard disc systems which use a different controller the screen fills with zeros and has to be reset if the *VERIFY command is used.

The ADFS commands *BACKUP, *COMPACT and *COPY no longer use conventional workspace, and can thus be used from within programs, opening up the opportunity for a file-by-file hard disc backup program.

POSTBAG POSTBAG

HELP SOUGHT

I am writing to you in the hope you will be able to help me in a frustrating search for a suitable utility program I can use with my video camera. I am not competent to write my own. I need some text scrolling programs for adding credits, descriptions, headings etc. to home videos. If you can suggest any discs suitable for a BBC micro I would be eternally grateful.

A.P.McDougall

We are unable to help directly, but if any other readers are able to assist then please write to Mr.McDougall c/o BEEBUG and we will forward your letter to him.

DISC ERRORS

I have a Master with Dumpmaster fitted, and have been running the Mandelbrot program from BEEBUG Vol.5 No.1. It works well, but when I ask it to save, however, it returns "ERROR 214". I have searched diligently for this error in the Welcome Guide, and in the two Reference Manuals, but without success. So, what is the meaning of 'Error 214', what can I do about it, and can you refer me to a source which lists ALL errors with their corresponding numbers?

Edmund Jupp

*In fact error 214 is "File not found". The program referred to saves to directory 'P' which implies that Mr.Jupp is running his Master using the ADFS. With this filing system, any directories must be created (with the *CDIR command) before they can be used, or the error which Mr.Jupp describes is likely to occur. With the DFS this should cause no problems.*

*We do not know of a comprehensive list of ALL error numbers, but the information on disc errors to which we referred came from **The BBC Microcomputer Disk Companion** by Tony Latham, published by Prentice Hall in 1983 at £7.95, but it is unlikely to be still in print. Nevertheless some computer shops or book shops may still have copies, or it may be available secondhand (via BEEBUG members' ads - see below).*

MEMBERS' ADS PAY OFF

I am writing to express my thanks to you for placing my advertisement in the Jan/Feb 1991 issue of BEEBUG (Vol.9 No.8). The response in comparison with my paid Micro User ad was quite amazing, and the majority of the large items have now been sold. I could have sold at least 20 Spellmaster ROMs - the phone never stopped.

David Saunders

This confirms other feedback which we have received regarding members' free ads in BEEBUG (and the same facility is now available in RISC User, our magazine for Archimedes users). However, if you do wish to avail yourself of this facility, please try to keep your ad reasonably short so that we can accommodate as many as possible each month.

LITHUANIAN APPEAL

I am a student of Vilnius University, Department of Astrophysics. It is difficult to imagine astrophysics without a computer. Regrettably, personal computers are not available in this country, and it is impossible to buy them directly from abroad because of currency control regulations. Fortunately, a friend of mine from Canada donated me a BBC model B. Now I have a lot of problems dealing with the information on hardware and software for the BBC micro. Occasionally I got an Acorn User magazine where I found an ad for BEEBUG. I am eager to read it, but it is impossible for me because of currency control. Maybe somebody would be willing to exchange several issues of BEEBUG (or even subscription) in return for books etc available in Lithuania. Your help in any way would be much appreciated.

Kriukelis Saulius

We have received letters from many parts of the world over the lifetime of BEEBUG, but I do believe this is the first we have received one from Lithuania. We have sent Kriukelis Saulius some back issue copies of BEEBUG magazine, and some indexes, to help him. If any other readers feel that they may be able to help, the address to write to is P.O.Box 1172, 232001 Vilnius, Lithuania, USSR.

Personal Ads

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

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

BBC B issue 7, Acorn DFS, 80T Watford DD with PSU, Watford 13 ROM expansion board, Acorn 6502 second processor with DNFS and Hi-Basic ROMs and manual, Integra B ROM/RAM expansion board with ROM and manual, Microvitec 1441 hi. res. colour monitor, Morley teletext adaptor with ats+ ROM and manual, Acorn cassette data recorder, pair BBC joysticks, pair Voltmace joysticks with splitter box and driver program, Sinclair ZX80, Sinclair ZX Spectrum issue 2 in Dk'tronics case with keyboard, ZX Spectrum thermal printer and paper, Watford speech synthesiser ROM and manual. Lots and lots of software. Please phone for details. Tel. (0277) 654343.

WANTED: Back issues BEEBUG magazine Vol. 1. Nos. 1-8 & index, Vol. 6. Nos. 9,10 & index, Vol. 7. Nos. 1-10 & index, Vol. 8. Nos. 1-10 & index, Vol.9. Nos. 1,2,3,4. WANTED: Back issue Micro User magazine, Vol. 7 No. 8 (Oct'89). Name your price and add postage on please. Write to Graham Badcock, 45 James Street, Kellerberrin, Western Australia 6410. Tel. 0011 61 90 454010.

WANTED: Circuit diagram or manual for the BBC Master Compact colour monitor. Tel. (0222) 490766 eves.

BBC Master Compact (3.5" DD), plus PAL TV adaptor, 4 Viglen cartridges fitted with Overview (View, Viewsheet, Viewstore, Viewspell, Viewplot, Index, Drivers) Dumpout and Command ROMs, printer lead, 20 discs inc. originals of Fairy Tales, Typing Tutor, Funschool, Little Red Riding Hood, Numbercopter, Nursery Rhymes, manuals, some BEEBUG's and books £225 o.n.o. Tel. (0622) 858476.

Archimedes 310 with colour monitor and twin DD, plus serial link for connection to Beeb, also lots of original software including: PC Emulator, First Word+, System Delta+, Reporter, Home Accounts, Hearsay, Interdictor II, Holed Out, Terramex, Ibix the Viking and Manchester United, plus lots of PC and Archie PD programs and shareware £550 the lot. Tel. (0344) 53272.

WANTED: Plotter for the BBC or Master. Tel. 021-565 3580.

Archimedes 410 upgraded to 420 (Watford's) I/O Podule, Multipod Professional (sound sampler, video digitiser, serial port, joystick port, 3 BBC compatible ROM sockets), serial lead + s/w for BBC to ARC transfer, 5.25" disc buffer MkII, DFS Reader, Original s/w including; Atelier, Interdictor, E Type, EMR Rhythm Box, EMR Creations disc's 1-6, Archive shareware discs 1-23, plus Archive & RISC User magazines £1,000. Tel. 081-751 5441.

Must sell BBC B items - need shelf space! Acorn CP/M Z80 adaptor comprising processor, 7 discs and manuals including File, Memo, Graph Plan, Nucleus, Accountant, CIS COBOL, Z80 user guides, all above for £35. AMT-2 Tx/Rx decoder for RTTY, ASCI & CW £35. Lots and lots of ROMs including manuals, also lots of books, please contact for further information too many items to list. Tel. (0273) 729506 eves, (0273) 200448 day.

Brother thermal transfer printer HR5 serial BBC lead, paper ribbons £50, Acorn teletext adaptor with ATS £30, View 2.1 ROM £5, Swift 2.2 BBC B Spreadsheet ROM/tape £5, Mini Office II BBC B, B+ tape £5, all boxed with manuals. Post extra. Tel. (0760) 23244 eves.

WANTED: Tandy TRS.80 plotter/printer or similar (plots A4 width but any length). Tel. 021-565 3580.

Master 512, mouse, GEM software, joystick, 40/80T twin drive, Sony RGB colour monitor/TV, TT adaptor, Star LC10 printer, Terminal, View, Viewsheet, Interword, Spellmaster, Interbase and other software ROMs, 2 quad cartridges, Shibumi Soft Problem Solver, many other original BBC and PC software discs, with all manuals, plus Dabs M512 User Guide and BEEBUG magazines. All in excellent condition £675 complete. Tel. (0483) 272947.

Cumana 40/80T DD with PSU £65, Cumana 40T DD £35, 5.25" floppies 50p each, Panasonic KX-P1124 printer £140, Linnet V21/23 modem £70, Thorn EMI modems £20, Acorn golf umbrella £15, Basic User Guide £5, in addition, Archimedes hardware and software. Tel. (04867) 80632.

Modem: BEEBUG Magic modem complete with Command ROM £40, BEEBUG internal modem complete with Command ROM £35, Pace Commstar II communication ROM (Hayes compatible) £10, BEEBUG Masterfile II Database £20. Tel. (0772) 612680.

BBC Master 512 with DOS mouse and GEM software, Wordwise, View, Viewsheet, Viewstore, Torch RGB colour monitor, dual disc drive 40/80T PSU, modem, misc ROMs, all manuals, leads and manu extras £450, Juki 600 daisywheel printer plus additional fonts £150, complete package £550. Tel. (0603) 485924 eves.

Epson MX80/III friction, tractor printer with lead, spare ribbon, maintenance and data sheets £75, 1200/75 1200/1200 OEL Telemod 2 modem, ROM, disc, data £45, BBC B spare case £12, Watford EPROM Programmer, ROM, data £35, EPROM eraser for 3 EPROMs £14, Watford plinths (2) can be joined for BBC B £11 each. Tel. (0366) 385174.

BEEBUG ROMIT for the BBC micro £6, Chocks Away £6, AMX mouse, ROM etc. boxed £14, dead Z88 £25. Tel. (0535) 609965 anytime.

Dual 80T drive with PSU £75, two Master cartridges with Interword, Command, Master, Advanced Disc Investigator, Exmon II and Lisp ROMs £50, Master Reference manuals £10, Dabs Master 512 Shareware collection No.1 £10, EMR MT32 Editor for use with MIDI interface £10, Solidisk 128K Sideways RAM board with 10 discs of software £15. Tel. (0462) 686818.

BBC B issue 7 (mint cond.) 1770 DFS, View A2.1, Microvitec colour monitor, Discs, leads, books etc. £225 o.n.o. Cumana CS400 disc drive available if required, for full details Tel. (0202) 742142.

BBC B with DDFS, speech chips, Viewstore, Prestel adaptor, Teletext adaptor, dual 3.5" drives with many discs. £200 o.n.o. Tel. 071-249 1482.

WANTED: 3.5" DD to daisy chain with existing 3.5" single drive. Tel. (0395) 263638.

Overview package for Master £40, Master MOS upgrade £20, the following books also, Master reference manuals 1&2, Dabhand Guides to View and Viewsheet/store, (including discs), all £8 each, postage included. Tel. (0788) 521189.

Designer Castles complete only £15. Tel. 081-854 6656 extn.27 office hours only.

WANTED: Viglen PC console kit for BBC B. Tel. (0872) 52653 weekdays.

Archimedes 310 with Philips colour monitor, also 20Mb SCSI hard disc, podule and software. Offers! Tel. (0366) 501001.

Master 128, hardware & software for sale, modem, BEEBUG Command, Desktop Publishing, Genie, many other items available, phone for list. Tel. (0202) 303926 eves & wk/ends.

Free to collectors: Prism 2000 modem, Watford Comms (Apollo) ROM, Philips EL3302 tape recorder, BEEBUG Masterfile, BEEBUG Wordease both 5.25", 'Acorn User' issue No.1 to date, sundry BBC cassettes. Tel. (058283) 3937.

Z80 second processor, twin 400K disc drives, cpm software (w/p, spreadsheet), Utilities £120 or will swap for twin Watford disc drives (5.25") or similar. BBC software; Wordwise £5, Watford Dumpout 3 (new) £10, Elite £5, Paintbox 2 £5, Various educational software, twin joysticks £5, BBC User Guide £5, beyond basic manual £5 all swaps considered. Tel. (0462) 895342.

BBC B issue 7, Watford DFS, Opus 40/80 drive, datacorder, manuals and games £175, Mini Office II - disc, book, minidriver ROM £20, Office Master and Mate £15, Fileplus ROM £15, Integrated Accounting pack £10, User and Advanced guides £5 each, all in good condition with manuals. All o.n.o's. Tel. (0978) 780584.

BBC B issue 7, Acorn DFS, Basic II, good working order, ARIES B32 Shadow/Sideways RAM, Watford double plinth (steel) any offers? Tel. (0438) 354177 anytime after 2pm.

BBC B 1.2 with Opus Challenger 3 in 1 5.25" DD including 275k hard disc, Mannesman Tally MT80+ printer, ATPL Sidewise ROM board, 16k sideways RAM, BBC tape recorder, View 2.1, Viewshee, Viewchart, Sharemaster Investment Analyst, BEEBUG Masterfile II database, Contex Bank Manager Home Accounts, Multi-font NLQ, Mini Office II, Replay tape to disc transfer, BEEBUG Dumpmaster, BEEBUG magazines & discs May '86 - Feb '91, games galore, all relevant manuals and many other books. Owner upgraded to PC. Phone for full list. Tel. (0454) 612671 eves after 6pm. No reasonable offers refused.

32k Shadow RAM/printer buffer board from Watford Electronics £49, hardly used, disagrees with my Slogger DFS. Tel. (0223) 861842.

Free to a good home A&B computing magazine, complete set (must be collected from London area), Speed read course (tapes) £3. Tel. 081-698 3772.

Star Gemini 10x works but hasn't got print head, will sell for £10 plus postage or buy working print head for same price. 100k 5.25" drive for BBC £20. Tel. (0671) 2201.

Master 128, Reference manuals 1&2, dual Viglen 40/80 single sided drives, Microvitec medium res. monitor with RGB PAL and audio inputs, complete set of BEEBUG mags in binders, several games of varying vintage inc. Elite £350. Care dual ROM cartridges £5, Peartree quad ROM cartrdige £5, BEEBUG Master ROM £18, BEEBUG Command ROM £18, BEEBUG Dumpmaster II (multi printer screen dump) ROM £15, Viewstore ROM £18, Spellmaster ROM £18, VASM disc to disc assembler ROM £12, EMR Midi interface inc. Miditrack performer (sequencer), Editor and composer software £50, EMR Scorewriter (music composing and printing) ROM with conversion utility for making Performer versions from Scorewriter files £20, BBC disc companion book £3, Viewstore/Viewsheet guide from Dabs Press £5, Bruce Smith's View Guide £5, (all postage extra or buyer collects - Dunstable). Tel. (0582) 601013.

BBC B issue 4, Cumana 40/80T drive, Watford DFS green screen monitor (cassette recorder), Wordwise, Spellmaster, Toolkit help ROMs, Masterfile, Quick Calc discs, all manuals £200 o.n.o. Offers for BEEBUG magazines Vol.1-1 to 8-5. Tel. (0923) 243955 eves.

Mannesman Tally MT80 printer £90 o.v.n.o. 512 co-processor Gem Software and mouse £90 o.v.n.o. Tel. 081-994 3205.

Magic Modem complete with Commsoft ROM and manual and all fittings ready to go on model B or Master £60, Cumana twin 5.25" DD 80T single sided with PSU's just overhauled at Cumana workshop £60. Tel. (0844) 52547.

BBC B issue 4, Basic II, double DD, ATPL ROM/RAM board (16k sideways RAM), ROMs include Disc Doctor, Toolkit+, Printmaster, Graphics, Forth, ATS, Teletext adaptor, joystick, cover and carrying case, lots of software; Utilities, Masterfile II, adventures, games and in-depth educational programs, BEEBUG Vols.1-9 complete, plus blank discs £250. Tel. (0249) 782271 after 6pm.

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		BEEBUG & RISC USER
£18.40	1 year (10 issues) UK, BFPO, Ch.I	£27.50
£27.50	Rest of Europe & Eire	£41.50
£33.50	Middle East	£50.50
£36.50	Americas & Africa	£55.50
£39.50	Elsewhere	£59.50

BACK ISSUE PRICES (per issue)

Volume	Magazine	5"Disc	3.5"Disc
5	£1.20	£4.50	£4.50
6	£1.30	£4.75	£4.75
7	£1.30	£4.75	£4.75
8	£1.60	£4.75	£4.75
9	£1.60	£4.75	£4.75

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

POST AND PACKING

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

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

BEEBUG
117 Hatfield Road, St.Albans, Herts AL1 4JS
Tel. St.Albans (0727) 40303, FAX: (0727) 860263
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: Alan Wrigley
Technical Assistant: Glynn Clements
Production Assistant: Sheila Stoneman
Advertising: Sarah Shrive
Managing Editor: Sheridan Williams

CONTRIBUTING TO BEEBUG PROGRAMS AND ARTICLES

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

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

In all communication, please quote your membership number.

BEEBUG Ltd (c) 1991
ISSN - 0263 - 7561

Printed by Arlon Printers (0923) 268328

Magazine Disc

April 1991 DISC CONTENTS

COLLAPSING SCREENS- an entertaining program which causes text displayed on your screen to collapse, and which can be used as an original way of clearing screens.

INSTANT ACCESS TO MAGSCAN ON A MASTER- this program enables you to use Magscan from sideways RAM on the Master for quick memory searching.

RECREATIONAL MATHEMATICS: FAST FACTORISING FOR FUN- a program which quickly finds the prime factors of any number up to 1,000,000,000, and its Euler number.

COLOUR BLENDER- This program for blending colours allows you to create up to 420 new colours and design unique screens.

BEEBUG FUNCTION PROCEDURE LIBRARY (PART 2) a Basic program containing all the functions and procedures from this month's instalment.

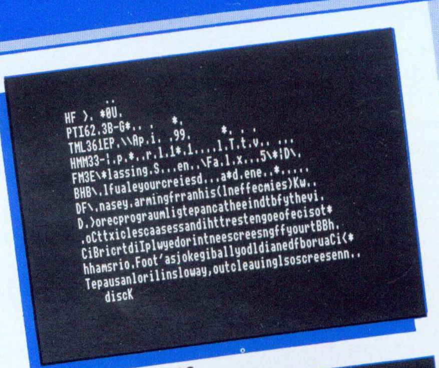
SIDEWAYS RAM COMMANDS FOR THE MODEL B- two programs RFShead and RFSload, published previously in BEEBUG Vol.8 Nos 8&9, and updated to run on a Model B.

PRACTICAL ASSEMBLER (8)- an assembler program which will automatically and invisibly encryprt any file written to disc and decrypt every file read from disc.

BEEBUG WORKSHOP: FILE COMPRESSION (PART 2) The complete Huffman encoding for compressing data with added lines for decoding; and two programs demonstrating Lempel-Ziv-Welch compression and decompression.

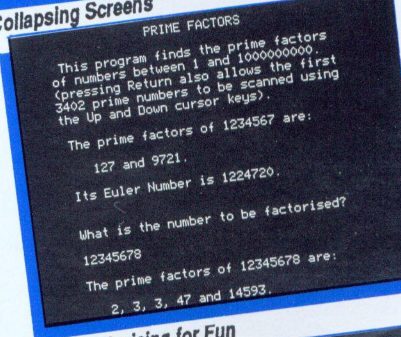
ADFS/VIEW UTILITY FOR THE MASTER (PART 2)- The complete PreView program extended to detect errors and with improved on-screen presentation; and a second program MultiPt for printing multiple copies of texts.

MAGSCAN DATA- bibliography for this issue (Vol.9 No.10).



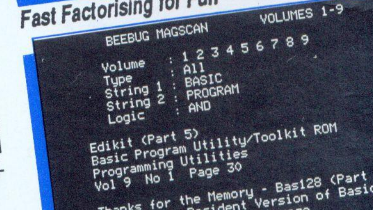
```
      ..
HF ). *0U.
PTI62.3B-G*.. .    *.
TML361EP.\\Ap.i. .99.      *. . .
HMM33-¦.p.*..r.1.1*.1....1.T.t.v.. ..
FM3E\*lassing.S...en..\Fa.1.x...5\*¦D\.
BHB\.lfualeyourcreiesd...a*d.ene..*.....
DF\.nasey.armingfrranhis(lneffecmies)Kw..
D.)orecprograumligtepancatheeindtbfythevi.
.oCttxiclescaasessandihttrestengoeofecisot*
CiBricrtdiIplwyedorintneescreesngffyourtBBh.
hhamsrio.Foot'asjokegiballyodldianedfboruaCi<*
Tepausanlorilinsloway,outcleauinglsoscreesenn..
   discK
```

Collapsing Screens



```
             PRIME FACTORS

  This program finds the prime factors
of numbers between 1 and 1000000000.
(pressing Return also allows the first
3402 prime numbers to be scanned using
the Up and Down cursor keys).

 The prime factors of 1234567 are:

    127 and 9721.

 Its Euler Number is 1224720.


 What is the number to be factorised?

 12345678

 The prime factors of 12345678 are:

    2, 3, 3, 47 and 14593.
```

Fast Factorising for Fun



```
   BEEBUG MAGSCAN        VOLUMES 1-9

              1 2 3 4 5 6 7 8 9
   Volume   : All
   Type     : BASIC
   String 1 : PROGRAM
   String 2 : AND
   Logic

 Edikit (Part 5)
 Basic Program Utility/Toolkit ROM
 Programming Utilities
 Vol 9  No 1  Page 30

 Thanks for the Memory - Bas128 (Part 1)
 Main Memory Resident Version of Basic
 Sideways RAM Program Storage
 Vol 9  No 3  Page 20

 Hint: Improved Move-Down Routine
 Using Additional Program Lines
 Basic/PAGE/Memory Restrictions
 Vol 9  No 4  Page 61
```

Instant Access to Magscan

ALL THIS FOR £4.75 (5.25" & 3.5" DISC) + 60P P&P (30P FOR EACH ADDITIONAL ITEM)
Back issues (5.25" disc since Vol.3 No.1, 3.5" disc since Vol.5 No.1) available at the same prices.

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

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

BEEBUG, 117 Hatfield Road, St.Albans, Herts AL1 4JS

Special Offers to BEEBUG Members April 1991

BEEBUG'S OWN SOFTWARE

Code	Product	Members Price inc.VAT	Code	Product	Members Price inc.VAT
1407a	ASTAAD3 - 5" Disc (DFS)	5.95	PAG1a	Arcade Games (5.25" 40/80T)	5.95
1408a	ASTAAD3 - 3.5" Disc (ADFS)	5.95	PAG2a	Arcade Games (3.5")	5.95
1404a	Beebug Applics I - 5" Disc	4.00	PBG1a	Board Games (5.25" 40/80T)	5.95
1409a	Beebug Applics I - 3.5"Disc	4.00	PBG2a	Board Games (3.5")	5.95
1411a	Beebug Applics II - 5" Disc	4.00	0077b	C - Stand Alone Generator	14.25
1412a	Beebug Applics II - 3.5" Disc	4.00	0081b	Masterfile ADFS M128 80 T	16.50
1600a	Beebug magazine disc	4.75	0024b	Masterfile DFS 40 T	16.50
1405a	Beebug Utilities - 5" Disc	4.00	0025b	Masterfile DFS 80 T	16.50
1413a	Beebug Utilities - 3.5" Disc	4.00	0074b	Beebug C 40 Track	44.25
1450a	EdiKit 40/80 Track	5.75	0075b	Beebug C 80 Track	44.25
1451a	EdiKit EPROM	7.75	0084b	Command	29.25
1452a	EdiKit 3.5"	5.75	0073b	Command(Hayes compatible)	29.25
0005b	Magscan Vol.1 - 8 40 Track	12.50	0053b	Dumpmaster II	23.25
0006b	Magscan Vol.1 - 8 80 Track	12.50	0004b	Exmon II	24.00
0011a	Magscan Update 40 track	4.75	0087b	Master ROM	29.25
0010a	Magscan Update 80 track	4.75	1421b	Beebug Binder	4.20

*Please add p&p. **UK : a** - 60p, **b** - £1.50,* ***Europe:** a - £1.00, b - £1.50,*

OTHER MEMBERS OFFERS

These offers are available for one month from publication. To avoid disappointment please order early, as offers are only available while stocks last and demand is always high. Orders are dispatched on a first come first served basis. To order phone (0727) 40303 or write to BEEBUG, 117 Hatfield Rd, St.Albans, AL1 4JS.

LISP

£2.99 (inc VAT) + £0.60 p&p
Normal Members price £20.00 (inc VAT)
LISP language on ROM (no manual).
Stock code 1003a

OverView
for the Master 128

£29.95 (inc VAT) + £4.50 p&p
Normal Members price £93.60 (inc VAT)

OverView combines all View family programs into one package, and complements View and ViewSheet supplied with the Master 128.

- OverView adds **ViewStore, ViewSpell, ViewPlot, ViewIndex** and the **Printer Driver Generator.**
- The ROMs are supplied in an Acorn ROM cartridge.
- Switch between packages without having to save files.
- The postage cost reflects the size/weight of the package.

Stock code 1020e

FORTH

£2.99 (inc VAT) + £0.60 p&p
Normal Members price £20.00 (inc VAT)
FORTH language on ROM (no manual).
Stock code 1041a

ViewStore

£13.80 (inc VAT) + £1.50 p&p
Normal Members price £42.49 (inc VAT)

ViewStore is Acorn's database manager program. It offers 40 and 80 column display, multiple indexes, detailed report generation, variable field lengths and much more. Excellent value for money.

Stock code 1019b

ViewSheet

£12.50 (inc VAT) + £1.50 p&p
Normal Members price £42.49 (inc VAT)

Excellent spreadsheet for the BBC micro and Master, supplied on a 16K ROM. A very powerful and extremely useable product which will produce results ready to print or for direct merging into View text files.

Stock code 1001b

ViewSpell

£7.00 (inc VAT) + £1.50 p&p
Normal Members price £28.75 (inc VAT)

An automatic spelling checker with a built-in 75 000 word dictionary. Supplied on ROM with full manual, examples disc and reference card. Ideal for View or ASCII text files, and can use updateable user dictionaries.

Stock code 1043b