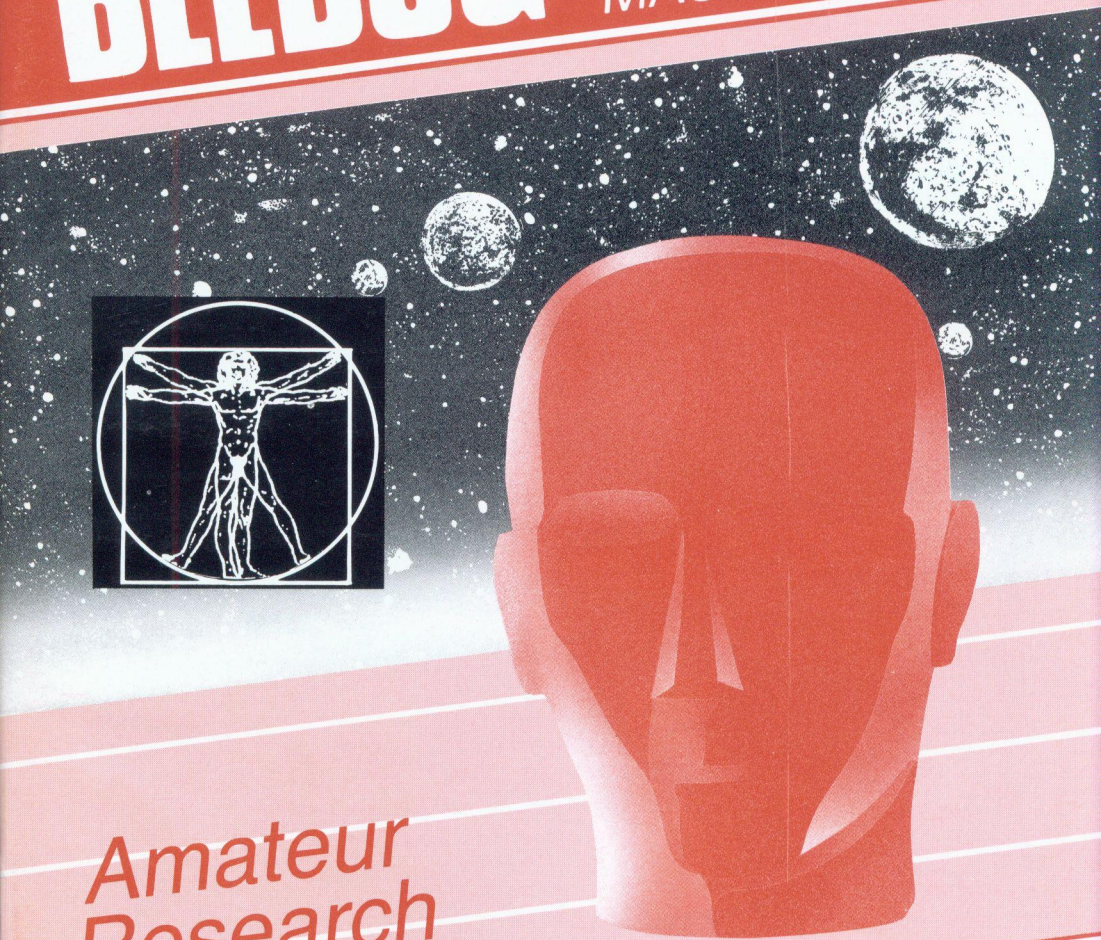
Vol.8 No.5 October 1989

BEEBUG

FOR THE BBC MICRO & MASTER SERIES



Amateur Research

- GRAPH PLOTTING
- DATAFILE SUPERDUMP
- PRINTER CONTROL
- AVON ADVENTURE

BEEBUG Vol.8 No.5 October 1989

FEATURES

REVIEWS

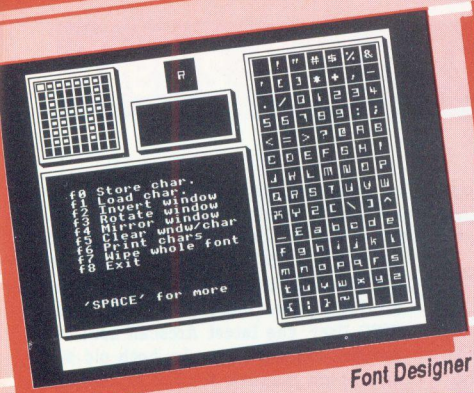
REGULAR ITEMS

HINTS & TIPS

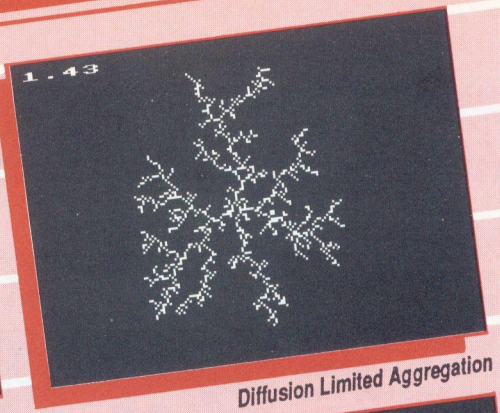
PROGRAM INFORMATION

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

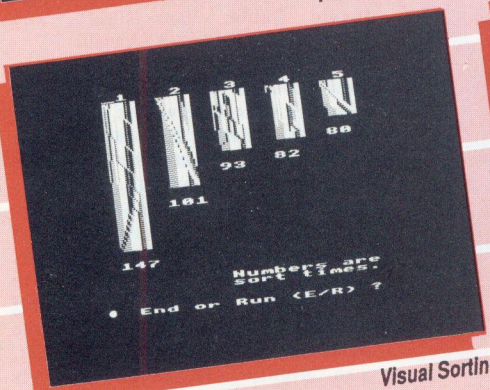
All programs in BEEBUG magazine will run on any BBC micro with Basic II or later, unless otherwise indicated. Members with Basic I are referred to the article on page 44 of BEEBUG Vol.7 No.2 (reprints



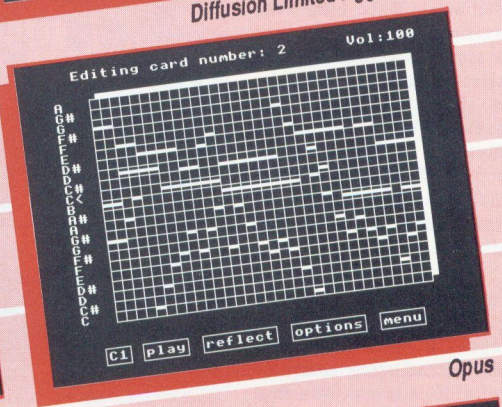
Font Designer



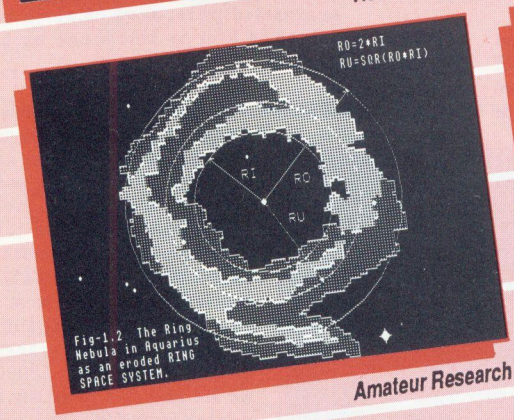
Diffusion Limited Aggregation



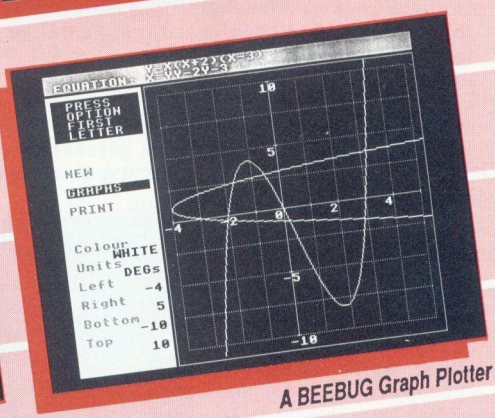
Visual Sorting



Opus



Amateur Research



A BEEBUG Graph Plotter

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

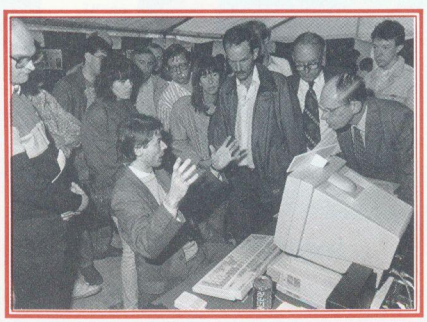
BEEBUG OPEN DAY

We were more than pleased at the high level of response to our Open Day held on Sunday 11th September. By opening time of 10am, the queue outside our new showroom entrance stretched out of sight. Much hard work went into ensuring that our new premises were as ready to receive members as possible, and BEEBUG staff and representatives from companies including Acorn, Computer Concepts, Minerva and Clares Micro Supplies did all they could to make your visit worthwhile.

Ian Mayer (prizewinner, (right) and Greg Denton (Mercantile Credit) with the Archimedes 410/1 prize.

Many visitors took the opportunity to purchase the latest in software and hardware including complete Archimedes systems. The first 25 purchasers of Archimedes systems using the 0% finance scheme received vouchers worth £50 courtesy of Mercantile Credit to spend on further items of their choice. Mercantile Credit staff were on hand to advise potential purchasers on financing schemes to support BEEBUG as a major Acorn dealer.

BEEBUG'S new handheld scanner, Scavenger, and forthcoming DTP package were both demonstrated to interested Archimedes users, while Computer Concepts, not to be outdone, were showing their forthcoming DTP package Impression, and Acorn were showing off the Archimedes range and latest software including Acorn DTP to enthralled visitors.



Computer Concepts' Charles Moir demonstrating Impression.

Many BEEBUG and RISC User readers found their way to the new magazine department offices to discuss the latest magazines, and comment on future developments. Also on show was the revised bibliography package for Archimedes users, ArcScan II, which can now be readily customised to handle a variety of suitable files. Not only that, but in addition to fully up-to-date bibliographies for both BEEBUG and RISC User, the latest Arcscan contains detailed bibliographies for both old and new Acorn manuals for the Archimedes (Basic Guide, User Guide, and Programmer's Reference Manual).

Over 1200 people, members and non-members alike entered the prize draw competition for an Archimedes 410/1 colour system donated by Mercantile Credit's Watford branch. The lucky prize winner, drawn on the Open Day, was Mr Ian Mayer of Scawthorpe near Doncaster who travelled down to St Albans on Tuesday 19th September to receive his prize in person from Greg Denton of Mercantile Credit. Ian, who is a long standing BEEBUG member, was highly delighted with his unexpected prize, news of which had fortuitously reached him by telephone on his birthday of all days.

We hope that BEEBUG and RISC User members will continue to visit us more and more in the future at our new premises which enable us to offer a better and more comprehensive service than ever before. And remember that although we shall continue to do our utmost to support Acorn's BBC Micro and Archimedes ranges (including the latest A3000), we are now able to satisfy all your Amstrad and PC compatible needs as well.

COMPETITION

We will be announcing the winner of our *Four Fours* competition in the November issue of BEEBUG, and expect to give details of the winning solution.

News News News News News News

ULTRA INTELLIGENT MACHINE

Impact Software has released first details of a brand new game for the BBC micro (with versions for the Archimedes) called *Ultra Intelligent Machine* (U.I.M.). Those who remember the classic game of Elite will find many similarities here.

The game is set in a world of the future where the greenhouse effect has had its ultimate effects, resulting in an overheated and flooded world where mankind has returned to the seas. Autonomous cities populate the oceans under threat from mutant *Replicators*.

As a trader, manufacturer, navigator, pilot and submariner your task is to overcome all dangers in your quest for the *Ultra Intelligent Machine*. There is a wealth of documentation (over 40 pages in our preview version), and the game certainly appears to have the potential to be another Elite. Prices will be £19.95 (5.25" disc) for the BBC micro (which needs sideways RAM) and Master 128, £21.95 for the Master Compact (3.5" disc), and £29.95 for the enhanced Archimedes version, release date 2nd October. Impact Software is at Neepsend House, 1 Percy Street, Sheffield S3 8AU, tel. (0742) 769950.

MATHS WORKSHOP

Three educational programs are now available from the aptly named Maths Workshop. These are *Symbolic Calculus* at £16.95, *Symbolic Algebra* at £19.95, and the *Algebra Support Disc* at £14.95. The purpose of the algebra and calculus products is designed to help a student understand how a problem may be solved. There are also opportunities for students to try their own hand at solving problems.

The support disc is intended to help with the building of libraries of suitable problems. All three products are available direct from Maths Workshop, 95 South Avenue, Worksop, Notts S80 2RE, tel. (0909) 500599.

POWER SHARING

Wordpower is a well established, full feature word processor for the BBC micro which has never achieved quite the success of View or Interword. Ian Copestake, originator of Wordpower, is now hoping to change all that by making it available as *shareware* for just £5, for which purchasers will receive a full working copy complete with documentation. *Wordpower Shareware* is available only from Ian Copestake Software.

A completely new product is a collection of *typefaces for the Star LC-10*. Typefaces released so far are called Black, Caldy, Outline, Pensby, Raby, Shadow and Universal. Each typeface has been design for proportional spacing, and may be printed in upright or italic form with the usual effects such as bold and underlining. Each typeface costs £9.90 (inc. VAT and p&p), any three for just £24.00, and the full set for £44.00.

To order, or to obtain more information, contact Ian Copestake Software, 10 Frost Drive, Wirral, Merseyside L61 4XL, tel. 051-648 6287.

ACCESSING SID

From 4th September 1989 all Prestel and Micronet users have been able to contact Acorn's *Support Information Database* (SID) through a Prestel gateway. SID provides support, information and telesoftware to both dealers and users, and has developed considerably in recent months.

For Micronet users, access to SID via the gateway will be at the rate of 1p per minute on top of any other normal time charges, and for Prestel users 3p or 4p per minute depending upon the time of access. This makes SID available to a large number of users at local call rates, rather than the long distance charges involved in dialling SID direct. For further information contact the Customer Services Department, Acorn Computers Ltd, Fulborn Road, Cherry Hinton, Cambridge CB1 4JN, tel. (0223) 245200.

PANDA DISCS FOR CHARITY

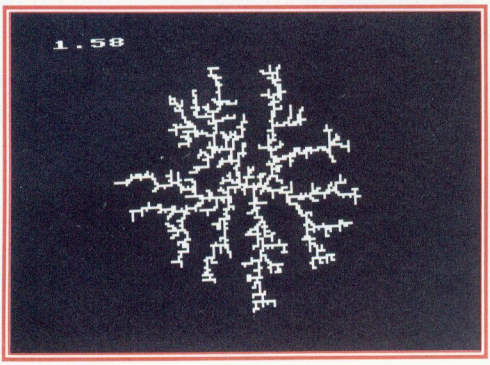
Panda Discs has released a disc of music in support of the *Children in Need* appeal. The disc contains a dozen pieces for the Hybrid Music 5000 System, including the theme music for the BBC's Software Show.

The music on the disc has all been donated by Panda Disc music programmers, whose royalties will be added to the proceeds from the disc. The disc costs just £5.00 inc. p&p from Panda Discs, Four Seasons, Tinkers Lane, Brewood, Stafford ST19 9DE.

Diffusion Limited Aggregation

by Grimble Gromble

Look at the illustration accompanying this article. What does it remind you of? Maybe nothing in particular, but your imagination, if allowed free rein, will no doubt suggest some fanciful answers. In fact, it is just one more example of a fractal. The process, which is implemented by the accompanying program, is known technically as *Diffusion Limited Aggregation*.



Typical random diffusion pattern

Starting with a single particle, successive particles move randomly until they come into contact with one already stationary, to which the new one then adheres. Another particle is then released and follows the same process. It can be likened to the movement of a particle of soot which moves aimlessly until eventually sticking to another soot particle. The growing number of soot particles form a random accumulation. The most colourful if least plausible example was given in *Computer Recreations*, Scientific American December 1988, pp 88-91 to which readers are referred for more information on this subject. This suggests a "succession of drunks wandering about in the dark until they stumble on a crowd of insensate comrades; lulled by the sounds of peaceful snoring, they instantly lie down to sleep. An aerial view of the slumbering crowd by morning might well reveal the same fractal shape."

To simulate this process, type in and save the program listed with this article. When run the program asks for three pieces of information. The *number of directions* can be either 4 or 8, and determines whether the random movement of particles is constrained to the north, south, east and west directions only, or whether the diagonal directions of NE, SE, SW and NW are also allowed.

The *number of cycles* determines, as the cluster grows, whether the colours cycle once or twice through the available shades. The third item requested is a random number seed. If you want to generate reproducible clusters then you should try specific numbers (either positive or negative) which can be used again to repeat a pattern. Just pressing Return will randomly initiate the random number sequence. However, if you want to reproduce a randomly generated sequence, the variable Z% contains the value used, and this can be found when the program terminates by typing:

```
PRINT Z%
```

The program works by releasing particles from random points on the circumference of a circle centred on the initial particle. As the cluster grows, so does the circle. Particles are constrained from moving outside the circle, otherwise the process would take forever. When a particle collides with a cluster, it sticks, and then surrounds itself with a 'sticky' layer. It is by detecting this sticky layer that the program avoids hunting around in its vicinity to find out if it has collided or not. The sticky layer is produced by simply setting the top bit of the colour number of surrounding pixels to one. Only pixels which are part of the cluster or its sticky layer have the top bit set, so a collision has occurred when POINT returns a value in which this bit is set.

The process is not a fast one, but interesting patterns can be built up in 20 to 40 minutes (much faster on an Archimedes of course). As with other fractals, it is surprising what fascinating and colourful results can be obtained from a quite simple process.

```
   40 REM Program DLA
   50 REM Version B1.0
   60 REM Author  Grimble Gromble
   70 REM BEEBUG  October 1989
   80 REM Program subject to copyright
   90 :
  100 ON ERROR GOTO 170
  110 MODE 7:PROCdir
  120 MODE 2:PROCinit
  130 PROCdla
  140 PROCanalyse
  150 END
  160 :
  170 MODE7:REPORT:PRINT" at line ";ERL
  180 END
  190 :
 1000 DEF PROCdir
 1010 PRINT "No. of directions (4 or 8):
";
 1020 REPEAT
 1030 P%=GET-ASC("0")
 1040 UNTIL P%=4 OR P%=8
 1050 PRINT;P%
 1060 PRINT "No. of colour cycles (1 or
2):";
 1070 REPEAT
 1080 N%=GET-ASC("0")
 1090 UNTIL N%=1 OR N%=2
 1100 PRINT;N%
 1110 INPUT"Random number seed:"Z%
 1120 ENDPROC
 1130 :
 1140 DEF PROCinit
 1150 E%=55:M%=3:N%=8DIVN%
 1160 IF Z% ELSE REPEAT Z%=RND:UNTIL Z%
 1170 Z%=RND(-ABS(Z%))
 1180 DIM x%(8),y%(8),d%(E%)
 1190 FOR I%=1 TO 8
 1200 READ x%(I%),y%(I%)
 1210 NEXT
 1220 FOR I%=0 TO 15
 1230 READ C%
 1240 VDU 19,I%,C%;0;
 1250 NEXT
 1260 FOR I%=0 TO E%
 1270 d%(I%)=0
 1280 NEXT
 1290 VDU 29,640;512;
 1300 VDU 23,1,0;0;0;0;
 1310 ENDPROC
 1320 :
 1330 DATA 8,0,0,8,-8,0,0,-8
 1340 DATA 8,8,-8,8,-8,-8,8,-8
 1350 DATA 0,7,7,7,7,7,7,7
 1360 DATA 0,4,1,5,2,6,3,7
 1370 :
 1380 DEF PROCdla
 1390 X%=0:Y%=0:L%=0:F%=-1
 1400 PLOT 69,X%,Y%:PLOT 65,0,4
 1410 REPEAT
 1420 IF F%<L% THEN PROCexpand:C%=FNcolo
ur
 1430 d%(L%)=d%(L%)+1
 1440 PROCsurround
 1450 L%=FNpoint
 1460 UNTIL L%>E%
 1470 ENDPROC
 1480 :
 1490 DEF PROCexpand
 1500 F%=L%
 1510 R%=8*(F%+M%)
 1520 S%=R%*R%
 1530 ENDPROC
 1540 :
 1550 DEF FNcolour=1+(F%DIVN%)MOD7
 1560 :
 1570 DEF PROCsurround
 1580 GCOL 1,8
 1590 FOR I%=1 TO P%
 1600 PLOT 69,X%+x%(I%),Y%+y%(I%)
 1610 PLOT 65,0,4
 1620 NEXT
 1630 ENDPROC
 1640 :
 1650 DEF FNpoint
 1660 A=2*PI*RND(1)
 1670 X%=(R%*COS(A))DIV8*8
 1680 Y%=(R%*SIN(A))DIV8*8
 1690 GCOL 3,C%
 1700 PLOT 69,X%,Y%
 1710 REPEAT
 1720 D%=RND(P%)
 1730 PLOT 69,X%,Y%
 1740 X%=X%+x%(D%):Y%=Y%+y%(D%)
 1750 IF X%*X%+Y%*Y%>S% THEN X%=X%-x%(D%
):Y%=Y%-y%(D%)
 1760 PLOT 69,X%,Y%
 1770 UNTIL POINT(X%,Y%)>7
 1780 PLOT 65,0,4
 1790 =(SQR(X%*X%+Y%*Y%))DIV8
 1800 :
 1810 DEF PROCanalyse
 1820 FOR I%=1 TO E%
 1830 d%(I%)=d%(I%)+d%(I%-1)
 1840 NEXT
 1850 A%=E%DIV10:B%=7*E%DIV10
 1860 @%=&20204
 1870 PRINT;(LN(d%(B%))-LN(d%(A%)))/(LN(
B%)-LN(A%))
 1880 @%=&90A
 1890 ENDPROC
```

A Printer Command Utility

Paul Pibworth describes a short program which allows a choice of printer control characteristics to be selected from a menu using a simple star command to initiate the process.

INTRODUCTION

For some time now, I have used a simple Basic program, which displays a list of useful printer facilities in the form of a menu when run. The appropriate key press then sends the necessary codes to the printer. Its aim is to save the chore of typing in VDU sequences from the keyboard while balancing the manual on one's lap!

There is, however, a need for a system that allows a printer to be reset from inside an applications package. I use Quest, a database for school use, which allows star commands. Sometimes, with Quest, it is necessary to set or cancel a margin, or even go into condensed print to make a convenient format. The method developed here allows the printer to be set up from within such a package with no trouble at all. This could prove useful with word processors and spreadsheets (for example View).

If a program allows the use of star commands, then a machine code routine which can control the printer is the answer. This is because it can be used by the *RUN command (or just *<filename>). The only problem is whether there is unused (available) space in the memory. The system described here uses pages &900 and &A00, but it could be changed to another area by modifying the value of C% in the program.

The basic idea is not difficult to achieve, but with limited space, the range of printer facilities which can be controlled is also restricted. This means that MY choice may not be YOUR choice. The way round this has been to write an assembly language program which can be customised to use your own choice of printer control codes. This program allows you to create YOUR own printer command files, each with up to ten sets of codes of YOUR choice. Using a printer command file then gives you the opportunity to select any of the printer controls you have built into that command file.

CREATING THE CODE

Type in and save the program listed at the end of this article. When you run the program you will be asked to enter details of the printer control codes you want to include. You must decide on your ten most necessary printer facilities, (e.g. NLQ, ELITE, etc.). You are only allowed 76 characters in total, so abbreviate the names you use. I also suggest that you include INIT (initialise). If you need more than ten settings, then create two (or more) separate files.

You will then be asked to supply the codes. Please note, only sequences that use the ESC code are allowed, AND only the codes that follow the 27 should be entered. Do NOT include the intervening '1's which would be needed in any VDU statement. Thus for a left-hand margin of ten, you would enter:

```
108 Return
 10 Return
```

NOT:

```
1,27,1,108,1,10
```

You are asked for a filename at the start of the proceedings. Once the printer codes have been entered, you can *SAVE this file by following the instructions.

USING THE CODE

Once a printer command file has been created, typing *<filename> will load and run the file. The screen will clear, and you will be presented with a menu of the choices that you have already included in that command file. Press the appropriate letters, (A-J), to select any facility. If you are in mode 7, the choices you make will be highlighted in green. In any other modes, the colour codes will appear as their characters. You can select more than one printer facility. If you include printer initialisation then this is sent first. If you change your mind, you can cancel. When you are ready, press Return to send the codes to the printer.

```
   10 REM Title   S.PRCOM
   20 REM Author  Paul Pibworth
   30 REM Date    Feb. 89
   40 REM A PRINTER COMMAND UTILITY
   50 REM Assembly Lang
   60 REM Master/BBC B
   70
   80 PROCcreate
   90 PROCassem
  100 END
  110
  120 DEFPROCcreate
  130 CLS
  140 PRINTCHR$131"YOU MUST FIRST ENTER
A FILE NAME FOR"
  150 PRINTCHR$131;"THE ASSEMBLED CODE I
N ORDER TO SAVE IT"
  160 PRINT'CHR$131;:INPUTLINE"ENTER HER
E "N$
  170 PRINT'"Now enter the facilities, a
s they"'"will appear in the menu."
  180 A%=INKEY(400)
  190 CLS
  200 DIM A$(10),B$(10)
  210 OD$=CHR$13+CHR$10
  220 bytes%=80-LENN$-6
  230 FOR I=1TO 10
  240 REPEAT
  250 PRINTTAB(0,0)"FACILITY NO. ";I," b
ytes remaining ";bytes%;" "
  260 PRINTTAB(0,I*2)"                "
  270 INPUTLINETAB(0,I*2)""A$
  280 B%=LENA$
  290 UNTIL LENA$<bytes%+1
  300 IFLENA$>0 A$(I)=CHR$(I+64)+" "+A$+
OD$ ELSE A$(I)=OD$
  310 bytes%=bytes%-LENA$
  320 NEXT
  330 CLS
  340 PRINT'"Now you will add the contro
l codes"'"for the facilities already ent
ered."
  350 A%=INKEY(400)
  360 FOR I=1 TO 10
  370 IFLENA$(I)>2 B$(I)=FNcode ELSE B$(
I)="*****"
  380 NEXT
  390 ENDPROC
  400 :
  410 DEFFNcode
  420 REPEAT
  430 CLS
  440 PRINT;I;"  ";MID$(A$(I),3)
  450 PRINT"ENTER";CHR$130;"ONLY";CHR$13
5;"THE CODE AFTER 27"
  460 PRINT"ie for <27> <40> <108> enter
40 & 108"
  470 PRINT"PRESS RET IF NO ENTRY"
  480 N=0
  490 D$=""
  500 REPEAT
  510 N=N+1
  520 INPUTC$
  530 IF C$<>"" D$=D$+CHR$(VALC$)
  540 UNTILC$=""  OR N>3
  550 FORJ=1TO LEND$
  560 PRINT"<";ASC MID$(D$,J,1);"> ";
  570 NEXT
  580 PRINT"IS THIS OK":A%=(GET AND 95)
  590 UNTILA%=89
  600 =LEFT$(D$+CHR$255+"****",5)
  610 :
  620 DEFPROCassem
  630 C%=&900
  640 ptr1=&70:ptr2=&72:ptr3=&74
  650 FORpass=0 TO 2 STEP2
  660 P%=C%
  670 [OPT pass
  680 PHP:PHA:TYA:PHA:TXA:PHA
  690 JSRblank
  700 LDA#12:JSR&FFEE
  710 LDA#28:JSR&FFEE
  720 LDA#4:JSR&FFEE
  730 LDA#19:JSR&FFEE
  740 LDA#35:JSR&FFEE
  750 LDA#4:JSR&FFEE
  760 LDX#menu MOD256
  770 LDY#menu DIV256
  780 JSRdisplay
  790 LDA#28:JSR&FFEE
  800 LDA#2:JSR&FFEE
  810 LDA#18:JSR&FFEE
  820 LDA#2:JSR&FFEE
  830 LDA#6:JSR&FFEE
  840 .keypress
  850 JSR&FFE0
  860 CMP#32:BEQesc
  870 CMP#13:BEQimplement
  880 CMP#127:BEQreset
  890 CMP#64:BCC keypress
  900 CMP#75:BCS keypress
  910 CMP#127:BEQreset
  920 SEC:SBC#65:STA&80
  930 LDX#hit MOD256
  940 LDY#hit DIV256
  950 JSRinsert
  960 .keypress2
  970 LDX#hit MOD256
  980 LDY#hit DIV256
  990 JSRdisplay
```

```
1000 JMPkeypress
1010 .esc
1020 LDA#26:JSR&FFEE
1030 PLA:TAX:PLA:TAY:PLA:PLP
1040 RTS
1050
1060 .reset
1070 JSRblank
1080 jmpkeypress2
1090
1100 .implement
1110 LDX#255
1120 .imp2
1130 INX
1140 LDAhit,X
1150 CMP#130
1160 BEQvdu
1170 .imp1
1180 CPX#9
1190 BNEimp2
1200 JSRblank
1210 JMPkeypress2
1220
1230 .vdu
1240 LDA#codes MOD256:STAptr3
1250 LDA#codes DIV256:STAptr3+1
1260 TXA:PHA:LDA#0
1270 .add
1280 CLC:ADC#5:DEX:CPX#0:BNEadd
1290 TAY:PLA:TAX
1300 .send
1310 LDA#2:JSR&FFEE
1320 LDA#1:JSR&FFEE
1330 LDA#27:JSR&FFEE
1340 .send2
1350 LDA(ptr3),Y:CMP#255:BEQsend3
1360 LDA#1:JSR&FFEE
1370 LDA(ptr3),Y:JSR&FFEE
1380 INY:JMPsend2
1390 .send3
1400 LDA#3:JSR&FFEE
1410 JMPimp1
1420 .codes
1430 EQUS B$(1)
1440 EQUS B$(2)
1450 EQUS B$(3)
1460 EQUS B$(4)
1470 EQUS B$(5)
1480 EQUS B$(6)
1490 EQUS B$(7)
1500 EQUS B$(8)
1510 EQUS B$(9)
1520 EQUS B$(10)
1530
1540 .insert
1550 STXptr2
1560 STYptr2+1
1570 LDA&80:TAY
1580 LDA#130:STA(ptr2),Y
1590 RTS
1600
1610 .display
1620 LDA#12:JSR&FFEE
1630 STXptr1
1640 STYptr1+1
1650 LDY#0
1660 LDA(ptr1),Y
1670 .display2
1680 JSR&FFEE
1690 INY
1700 LDA(ptr1),Y
1710 CMP#255
1720 BNEdisplay2
1730 RTS
1740
1750 .blank
1760 LDX#255
1770 .blank2
1780 INX
1790 LDA#&89:STAhit,X
1800 CPX#9:BNEblank2
1810 RTS
1820
1830 .menu
1840 EQUS" "+CHR$129+N$+OD$+OD$
1850 EQUS A$(1)
1860 EQUS A$(2)
1870 EQUS A$(3)
1880 EQUS A$(4)
1890 EQUS A$(5)
1900 EQUS A$(6)
1910 EQUS A$(7)
1920 EQUS A$(8)
1930 EQUS A$(9)
1940 EQUS A$(10)
1950 EQUS""+CHR$10+CHR$13+CHR$130+"DEL
TO CANCEL"+CHR$10+CHR$13
1960 EQUS""+CHR$129+"RETURN TO IMPLEMEN
T"+CHR$10+CHR$13
1970 EQUS""+CHR$134+"SPACE TO EXIT"
1980 EQUB255
1990 .hit
2000 EQUS"          "
2010 EQUB255
2020 ]
2030 NEXT
2040 CLS
2050 PRINT''"COPY NEXT LINE TO SAVE CODE"
2060 PRINT"*SAVE ";N$;" ";~C%;" ";~P%
2070 ENDPROC
```

Amateur Research

John Belcher presents the first of a lively new series of articles on amateur research with a personal view on the history of the solar system.

BY WAY OF INTRODUCTION

It had to come about eventually, I suppose. All that time you've spent being zapped by aliens; being devoured by monsters; and being imprisoned in dungeons, it was bound to tell on you in the end! Meanwhile, boredom setting in, you wonder what to do with an ageing BBC micro, which - when all is said and done - has more computing power in it than many a 1950's mainframe. Perhaps it's about time that you started to put it to some real use for a change.

Amateur research is one of those activities in which the average Beeb user can find space to expand, expand his or her horizons; expand his or her interests. And at the end of it all, there may be the added satisfaction of seeing your work abreast of the latest developments in the professional field - and perhaps even in advance of them.

In order to introduce you to amateur research, I have talked the Editor into letting us have a few pages of BEEBUG each month, for the purpose of research employing that personal computer you all know and love so well.

For my part I will try to ensure that all matter for discussion will be as near *original* as possible. Indeed I shall be using some of my own work in order to give you a hands-on experience of what it is all about, but much of it has not been researched in detail. It is at this point where you, the reader, come into the picture. If some aspects captivate your interest, and you feel that you can improve on the theory, or the simulation and modelling, or the application - then drop us a line and let us know about it.

At this point you will probably be thinking that this is not for you. Too mathematical for someone who failed A-level maths; and too involved for someone who can't even get to grips with assembler. Well, in the first place, if you can find your way round the Beeb keyboard, then the mathematics of this research should be no problem. And BBC Basic is fast enough and powerful enough to do all that I require of it.

GETTING STARTED

Happy now? Then let's get down to brass tacks! The preliminary stages of all research are as follows:

1. COLLECTING FACTS
2. MAKING TABLES
3. ATTEMPTING TO IDENTIFY PATTERNS

COLLECTING FACTS

Facts are those things in natural phenomena that we *perceive* and hopefully *measure*. Today we might call these facts primary data. This *primary data* is a very valuable commodity, because much time and patience can have gone into its collecting. Some poor poverty-stricken observer might have spent many solitary hours, on cold freezing nights, for months on end, at an altitude of umpteen thousand feet, looking through the end of a telescope, just so that you - you ingrate - can make use of but one item of data. So don't chuck it in the wastepaper bin without a second thought.

Data requires frequent updating, largely because, with better measurement techniques and so on, better quality data becomes available all the time. In a typical case, the gravitational constant is quoted as G = 6.670E-11 N m^2 kg^-2. Somewhere - to my shame I have lost the reference - I have discovered that a revised figure is given as 6.6732E-11, and have amended my own database accordingly. Side-by-side with this, however, is the value 1.0972E-10, which is the amended value corrected for *proximity effect*.

I suppose I'd better explain about this *proximity effect* in passing, although it might be better dealt with in more detail later on. My own research into force characteristics of *non-point-source* bodies shows that at, and in the vicinity of, the surface of a solid spherical source of force, the force itself becomes *attenuated*. This is caused by the proximity effect. You can see this in Fig.1.1, in which the resultant force, Fn, dips below the theoretical straight-line graph at the point where the distance from the centre of the spherical body is equal to its radius. At our local Amateur Research Group recently, it was mentioned -

jokingly of course - that when nuclear physicists are unable to get their equations right they invent a new particle to explain away the deficiency! That said, an important new topic in the scientific world - currently in the press and on television - is the discovery of the so-called 'fifth' and 'sixth' gravitational forces. On going down a mineshaft, these researchers discovered that gravitational force, instead of decreasing, actually *increases*. This they have called the 'fifth' force. Similarly, on climbing up a high mast, they again found that gravitational force, instead of decreasing, again *increases*. Yes, you've guessed it! This they call the 'sixth' force. All nicely predicted in Fig.1.1, based on some early work of mine.

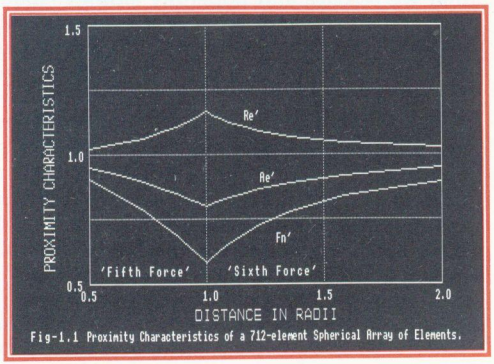


Figure 1.1

Data derived from primary data is, of course, *secondary data*. Not perhaps factual in the rigorous sense, but possibly valuable nevertheless. It follows, therefore, that if the primary data needs to be amended, then too must its secondary data be amended also. And this brings to mind another important point.

We have to be very careful how we interpret published data. Textbooks, for example, may show an accuracy of only four figures, whereas the constant in question may be known to much greater accuracy. Then again, the writer of the textbook may have copied the data from some other book, which was copied from .... who got it wrong in the first place. Moreover, you would expect, given the *radius* of the Earth, thence calculating its *volume*, multiplying by its *density*, that you would arrive at its *mass!* Oh dear me no! Important? Well, I once spent several weeks trying to debug a program before I chanced upon that particular banana-skin. The moral, of course, is TAKE NO DATA FOR GRANTED. Least of all that which you yourself may have collected!

MAKING TABLES

Having collected the facts, it then becomes necessary to arrange them in *collections*. If that seems like stating the obvious, then perhaps it should be explained that the way in which one arranges such a collection may appreciably influence the outcome. You are on your own at this point! Generally speaking, electronics data should go into the electronics collection; astronomical data should go into the astronomical collection. After that, it should be arranged in any natural or logical order; at worst numerically or alphabetically.

PATTERN RECOGNITION

This is where the intelligent bit comes in. Pattern recognition is a feature apparently that distinguishes human intelligence from general animal intelligence. Though my dog does beg to differ. Computer engineers may think that machine-intelligence, AI, should be mentioned at this point, but having read accounts - albeit journalistic versions - of the state of the art, I have my doubts. They say two sheep's heads are better than one, and I'd back them against a computer any day. They've been around that much the longer, and are no strangers to worms and viruses!

In his advice to the researcher, Bacon advises tempering pattern recognition with patience. Don't jump the gun. Consider the obvious patterns first of all. Consider the logical patterns. Chances are, that thousands of other researchers have been there before you. Not to be outdone, take a seemingly illogical approach. Stand on your head; look at the data in a mirror. Very importantly, don't be afraid to take the unorthodox approach.

In looking for patterns I have found that in natural phenomena, three features seem to appear with remarkable frequency. In that sense they can be said to be ubiquitous:

1. The constant PI = 3.14159265
2. The constant EXP(1) = 2.71828183
3. The geometric mean of a group of numbers.

BRASS TACKS AT LAST

The concept of force is the basis of all physical science. Newton established the concept of gravitational force by his study of the Earth-Moon space-system. Coulomb developed this concept in the field of electric force and magnetic force. Ergo, if we wish to understand a great deal more about the concept of force, then we must improve our understanding of the physics of space and space-systems (1).

I suppose it could be summed up by saying that astronomy is the Father and Mother of all the physical sciences. By which I mean real astronomy, and not some of the sci-fi that hits the headlines today. With no further ado, we will take a new look at the Solar System.

THE ESSENTIAL SOLAR SYSTEM

You will be pleased to learn that we are now going to examine a real live BBC Basic computer program, ARPROG1.

Some planetary data has been placed in DATA statements at line 10010 onwards. At line 110, the program reads this into defined arrays, P$(n) giving us the names of the eight planets concerned, R(n) their orbital radii in astronomical units (a.u.), and w(n) the longitudes of their perihelions (the nearest point of an orbit to the sun). The only significant pieces of information of interest are the *eight* values of R(n). We use the rest to make pretty graphics!

It so happens that those particular data have been around for a couple of hundred years or so. Presumably clever academics and students of astronomy alike during that time have examined these data, and - if they were worth their salt at all - must have attempted to make something of them. But during all that time only one person, be he called Bode or Titus, has come up with any realistic pattern. Which is to say, Bode's Law. You may wish to look it up in a suitable astronomy textbook.

We will now take those same eight pieces of information, and by a simple approach, we will determine the following:

1. The Unit Radius of the Solar System.

2. The origin and layout of the Proto-System involving the Sun and the Asteroid Belt.

3. The sequence whereby the Sun captured pairs of planets to form the Planetary System today. In other words the subsequent evolution of the Solar System.

4. With the addition of two more items of information, we then determine the orbital velocity of the Sun around its primary! I'm sorry if this comes as a shock to you, but that's the way it is!

5. With the perihelion data, I then attempt to demonstrate that Nature has much more innate beauty than all those heaps of builders' rubble that pass for art in the National Gallery.

So type in the program, save it, and run it but DON'T press any keys until I tell you to.

You will be presented with the results of some calculations headed by the title "THE SOLAR SYSTEM". The third item down is of major importance, the unit radius of the System. In order to determine this, we have to apply one of my 'ubiquitous' features to the problem, the *geometric mean*. Those of you who have 'done' this at school or elsewhere, will know that, given two variables A and B, their geometric mean is SQR(A*B). If we have more than two variables, eight in this instance, the geometric mean is the 8th root of their multiple, i.e.:

(R1*R2*R3*R4*R5*R6*R7*R8)^(1/8)

In line 120, then, we find the value of RU, the unit radius of the Solar System, expressed in a.u. I should perhaps explain that we are in unexplored country at this point, so you can put away all those astronomy textbooks.

The unit radius of any system, planetary or atomic, is of major importance for many reasons, chief among which is that it defines the *limit* of the system. Thus you will see that the planets are divided into two groups, the Inner Planetary Group - Mars, Earth, Venus and Mercury, and the Outer Planetary Group - Jupiter, Saturn, Uranus, and Neptune. I suppose you could say that the former is equivalent to the atomic nucleus, and the latter the electrons orbiting it.

The Proto-Solar System originally consisted of just the Sun and the Asteroid Belt - about which more later. The planets subsequently joined the System in *planetary pairs*, P1 - Mars-Jupiter; P2 -

Earth-Saturn; P3 - Mercury-Neptune; and P4 - Venus-Uranus. If we take the geometric mean of the orbital radii of each pair in turn we find that their resultants approximate to the unit radius. Moreover, and remember 'space is always expanding', we see that the geometric mean of the first pair is the least of the four, while that of the fourth pair is the greatest.

At this point you will be mumbling something about Pluto! Well, that is currently a question-mark. Its path has so far led it nearer to the Sun than the orbit of Neptune, and in the likelihood that it carries on towards Uranus, Saturn, Jupiter, Mars, and Earth, then I suggest that it is the Inner Planet of the FIFTH planetary-pair, (I've already named its outer companion Divinator)! But enough of all this self-pride, sufficient to say that Pluto is best ignored for the time being.

What's that? They are in the wrong order? 1-2-4-3? Well this is a *dynamic* system - just like your J-registration Mini! And its cylinders don't fire in a 1-2-3-4 order, do they?

Which brings us to the Asteroid Belt (2). By now, I hope you have forgotten all that 'balls-of-fire' nonsense describing the creation of the Solar System. Likewise, forget that other theory about a fictitious planet Vulcan which broke up to form the Asteroid Belt! In line 150, we determine the *inner* and *outer* limits of the original Belt before it became eroded. RI=SQR(1/2)*RU, and RO=SQR(2)*RU. Thus RO=2*RI. With the passage of time, the outer portion of the Belt becomes lost to the System, and the final outer limit is at RU. You can see pictures of this in your astronomy book. The Ring Nebula in Lyra is typical of the original planetary nebula in that its outer diameter is *twice* its inner diameter. The Ring Nebula in Aquarius, on the other hand, as shown in Fig.1.2, evidences signs of erosion down to its final outer limit at unit radius. You will notice in this particular instance the manner in which the unit radius of the system begins to make itself evident.

It so happens that the maximum eccentricity of orbit for a member of the Asteroid Belt is such that its perihelion is at RI, and its aphelion at RO. Under these circumstances, the value of its eccentricity is e=0.333333333. Surprise, surprise! We also see later that the length of its semi-minor axis is b=RU. You will appreciate, therefore, that the situation is a goldmine for any prospecting mathematician!

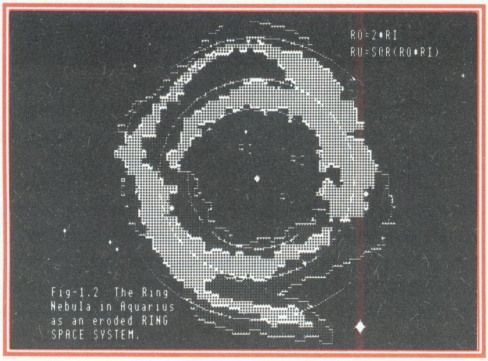


Figure 1.2

The inner limit, RI, is also important because the orbital velocity of a body at this point around its primary the Sun, is also the orbital velocity of the Sun in orbit around its primary in turn! You don't believe me? Well check with your textbook, and if it's a half decent one it will tell you about the Sun's velocity towards the *solar apex*, the value shown on the screen. You may now 'Press any key' (except Escape or Break of course).

The outer limit of the Belt is now drawn on the screen in red - or dark grey, followed by unit radius in yellow - or light grey, followed in turn by the inner limit in red again. Nice, isn't it?

Now press any key again. The elliptical orbit of a hypothetical asteroid at maximum eccentricity is now drawn. Note its perihelion at RI, its aphelion at RO, and the length of its semi-minor axis equal to RU.

We will now see something spectacular. Before I investigated *longitudes of perihelions*, nobody else would appear to have given them a second thought. Indeed one clever little academic a few years ago brought out a data book which itemised everything *except* the longitudes of perihelion.

Now, bearing in mind what I've said about the history of planetary pairs, press any key. You are now looking at a plan view of the planets' longitudes of perihelion relative to the central Sun. They all fit in a restricted arc. Naming them from the right-hand side anti-clockwise

they are - Mars (inner), Jupiter (outer); Neptune (outer), Mercury (inner); Saturn (outer), Earth (inner); - yes you've got it! - Venus (inner), Uranus (outer). With the engine cylinders arranged in the order of 1-4-2-3!

Press any key again. We now see the arithmetic mean values of each of the pairs equidistant in yellow. Notice the symmetry of this particular pattern.

Press any key - for the last time! We have now taken the arithmetic mean value of all four means and plotted its *reciprocal*. It points in the general direction of the GALACTIC CENTRE. No prize for guessing where Pluto-Divinator will fit into the scheme of things!

I hope you enjoyed that! Sometime in the future perhaps it may come to be known as the *BEEBUG Perihelion Pattern*. Who knows?

RETROSPECT AND PROSPECT

Well, so far we have dealt with the subject of data. And hopefully you have now come to realise just how significant it can be. Remember what just eight pieces of information have shown us about the Solar System for example. You'll find plenty more such data in textbooks. Of course, the discovery of the principles behind the data took time to determine.

That's all for this month. Don't forget to write in - preferably with some ideas of your own.

REFERENCES

(1) Belcher, J.C., *Unit Radius and the Resolution of Inverse Square Law Forces, International Journal of Theoretical Physics, Vol.15, No.10, (1976), pp.55-771.*

(2) Chapman, Clark R., *The Nature of Asteroids, Scientific American, Vol.232, No.1, (Jan 1975), pp.4-33.*

```
   10 REM Program ARPROG1
   20 REM Version B1.0
   30 REM Author  John C.Belcher
   40 REM BEEBUG  October 1989
   50 REM Program subject to copyright
   60 :
  100 MODE 1:PRINT'''"THE SOLAR SYSTEM"
  110 DIM P$(8),R(8),w(8),W(4):FOR I%=1
TO 8: READ P$(I%),R(I%),w(I%):NEXT
  120 RU=(R(1)*R(2)*R(3)*R(4)*R(5)*R(6)*
R(7)*R(8))^(1/8)
  130 P1=SQR(R(4)*R(5)):P2=SQR(R(3)*R(6)
):P3=SQR(R(2)*R(7)):P4=SQR(R(1)*R(8))
  140 PRINT'''"1st Pair     = "P1'"2nd Pai
r     = "P2'''"Unit Radius = "RU'''"4th Pai
r     = "P4'"3rd Pair     = "P3'
  150 RI=SQR(1/2)*RU:RO=SQR(2)*RU
  160 a=(RI+RO)/2:ae=a-RI:e=ae/a
  170 PRINT'"Inner limit of Belt  = "RI'
"Initial outer limit  = "RO'"Final outer
 limit    = "RU'''"Max eccentricity
= "e
  180 G=6.6732E-11:Msun=1.9891E30:R=RI*1
.49597892E11:Vsun=SQR(G*Msun/R)
  190 PRINT'"Sun's velocity in orbit
 - towards the'solar apex' = "Vsun/1000"
 Km/sec"'''"(NOTE: Sun's absolute velocit
y"'SPC7"= 4227 km/sec)":PROCpause
  200 CLG:GCOL0,1:PROCclipse(640,500,400
,0):GCOL0,2:PROCclipse(640,500,200*SQR(2
),0):GCOL0,1:PROCclipse(640,500,200,0)
  210 PROCpause
  220 GCOL0,3:PROCclipse(740,500,300,1/3
):PROCpause
  230 CLG:GCOL0,1:FOR I%=1 TO 8:MOVE 600
,500:DRAW600+300*COS(RAD(w(I%))),500+300
*SIN(RAD(w(I%))):NEXT:PROCpause
  240 GCOL0,2:W(1)=(w(4)+w(5)-360)/2:W(2
)=(w(3)+w(6))/2:W(3)=(w(2)+w(7))/2:W(4)=
(w(1)+w(8))/2:W(0)=(W(1)+W(2)+W(3)+W(4))
/4+180
  250 FOR I%=1 TO 4:MOVE600,500:DRAW600+
500*COS(RAD(W(I%))),500+500*SIN(RAD(W(I%
))):NEXT:PROCpause
  260 GCOL0,3:MOVE600,500:DRAW600+600*CO
S(RAD(W(0))),500+600*SIN(RAD(W(0))):PRIN
TTAB(16,30)"To Galactic Centre":END
  270 :
 1000 DEF PROCclipse(x,y,radius,ec)
 1010 MOVE x+radius,y
 1020 FOR I%=1 TO 360:X=x+radius*COS(RAD
(I%)):Y=y+radius*SQR(1-ec^2)*SIN(RAD(I%)
):PLOT5,X,Y:NEXT:ENDPROC
 1030 :
 1040 DEF PROCpause
 1050 PRINTTAB(13,30)"Press any key"
 1060 A$=GET$:PRINTTAB(13,30);SPC13:ENDP
ROC
 1070 :
10000 REM  Planet,Semi-major axis,Longit
ude of perihelion
10010 DATA Mercury, 0.387098273,73.8983
10020 DATA Venus, 0.723332758,130.16416
10030 DATA Earth, 1.0,102.2192
10040 DATA Mars, 1.5236915,334.2183
10050 DATA Jupiter, 5.2028039,12.716
10060 DATA Saturn, 9.5388437,91.083
10070 DATA Uranus, 19.181826,169.05
10080 DATA Neptune, 30.058021,43.86
```

OPUS

Ian Waugh reviews a new musical package for the BBC micro with an educational flavour.

Product	**Opus**
Supplier	**The Advisory Unit, Endymion Road, Hatfield, Herts AL10 8AU. Tel: (07072) 65443**
Price	**£15.00 inc VAT**

In spite of the fact that many pundits are trying to write off the BBC micro it is still very much alive and kicking, especially in the world of education. It has had an enormous number of music programs written for it and Opus is one of the latest.

Although neither the program nor the manual actually say, "this is an educational program", hints to that effect are scattered throughout and Opus seems to have been designed for classroom use. The new music curriculum places much emphasis on composition, a topic which both pupils and teachers often have difficulty in getting to grips with, and it is this subject which Opus tackles.

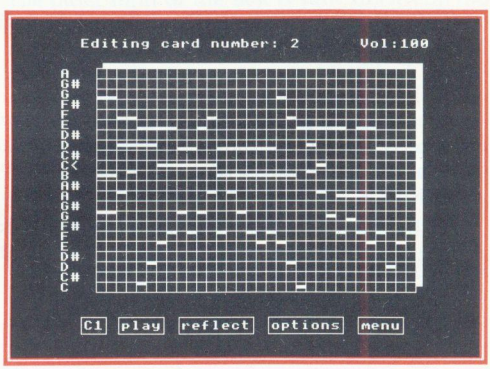
The disc is not copy-protected and the manual tells you to make a back-up copy before use - a wise precaution and a brave and welcome move on behalf of The Advisory Unit.

Opus lets you construct tunes on a grid using all three of the BBC's sound channels. You control the program by moving a pointer around the screen to point to an option and pressing Return. Control can be with the cursor keys, a joystick, a tracker ball or the Concept Keyboard. I found cursor control to be the best although the program is written in Basic and the pointer steps around the screen rather than gliding. But this is not a severe problem.

Opus has three screens - the Main menu, the Composition screen and the Editing screen. Tunes are built up in the Editing screen. This displays a grid with a list of 22 note names down the left hand side. Initially there are 34 boxes across the screen (the maximum number of notes you can enter on one grid) but this can be reduced to 24. Notes are toggled on and off by pointing to a box and pressing Return. The pointer moves on automatically ready for another note, a nice touch.

You can enter a three-part tune on a single grid (or 'card' as the program calls them) in which case the notes appear at the bottom, centre and top of the box. This works quite well although it's not always easy to tell which notes belong to which channel at a glance.

Unfortunately, you don't hear the notes as you enter them on the card. The manual says you don't need any previous musical experience or a knowledge of music notation in order to use Opus but as you don't hear notes when you select them and as reading notes on a grid is not exactly an instinctive process (although neither is traditional notation, come to that), the novice will probably only be relying a crude sense of 'higher' or 'lower' when entering notes. To this end some sort of supervision would be helpful (back to education again) unless you are happy with a trial and error approach.



Opus musical notation using 'cards'

To actually hear what you've written you select the Play option - of course. The program compiles the data (taking a second or two to do so) and then you hear the music. The delay between entering notes and hearing the card

may be slight but but, given the nature of the display and the note representation, a single 'instant' play button - say a function key - would have been very useful.

Notes placed in adjacent boxes sound as one long note so to repeat the same pitch you must leave an empty box between notes. Although not a severe limitation given the introductory nature of the program this just goes to demonstrate that even alternatives to traditional notation have their own problems.

Some particularly interesting composition options can be found in the Reflect menu. You can reverse the notes so the card effectively plays backwards; you can invert the tune so the high notes play low and vice versa; and you can shift the tune, effectively a transpose function. These can be applied to individual channels or to the whole card.

These are the sort of musical manipulations which are ideally suited to computer control, and they offer much scope for experimentation. Music by Bach in particular lends itself well to such functions. New pieces of music can be constructed which still have a degree of musical coherency with the original card. Experimenting with these operations is fascinating.

You can copy one card to another although a function to copy one channel's note to another would have been useful, too, especially when experimenting with the Reflect options.

Opus has another trick up its sleeve, too. You can alter the Note List which appears down the left of the grid. There are two fixed lists - major and chromatic - and a programmable list. This is originally set to pentatonic (the black notes on a piano) but you can create, save and load your own lists using a separate program on the disc. Files are supplied for whole tone and quarter tone scales.

You can alter the pitch and volume of the notes and select one of six types of envelope. I didn't find the envelope definitions particularly impressive, however - two are very similar and one produces a horrible heavy vibrato. There is no facility to create your own envelopes. On the one hand that's a shame, but on the other it avoids confusing the user with a mass of envelope data; probably a good thing.

You can create up to 12 music cards and link them together in the Composition screen. Again, this is grid based and card numbers are entered in a 10 by 6 grid to form the final tune. You can alter the tempo from the Main menu and this ranges from 2 to 10.

Three music files are supplied on disc - a Bach three-part Invention, In Dulcio Jubilo, and an original composition which makes use of the Reflect functions. Try some of these on the Invention file, too - great fun. A printout can be produced by pressing Ctrl/P - very useful for classroom work.

For a program which is so ideally suited to education, it's surprising that the manual doesn't include any suggestions for use in that area. In particular, the Note Lists can be loaded independent to the grid, and users could explore the melodic and harmonic differences produced using the same grids and different notes. There is also scope for experiment in song construction by linking grids together in different orders.

The manual is short - 20 pages - but to the point, and with screen dumps on every other page.

While the program is easy to use and understand, a couple of aspects of its implementation could be tidied up and improved a little, perhaps. But don't let that deter you from investigation as it could certainly give an insight into the role of melodic and harmonic lines in a composition.

While the manual claims you don't need any musical knowledge to use Opus, you do need at least a basic concept of pitch and duration. The solo user could have fun with it, but I reckon it will really come into its own in the classroom under supervision of an enthusiastic teacher where it could be form a small part of a course on composition.

A BEEBUG Graph Plotter

Robin Murphy describes the first part of a comprehensive graph plotting package which has many applications in education and elsewhere.

A graph plotting program is one that mathematicians find extremely useful. There are plenty to choose from, and writing one can be a very instructive programming exercise. Presented here is one which is particularly easy to use, presents the graphs very clearly, and allows a wide range of applications. It has been developed using ideas tried and tested with the pupils and staff of Ampleforth College.

This first part is self-contained, but will allow only simple graphs such as $y=x^2+1$ and $x=\tan y$ to be entered. In Part 2, appearing next month, the program will be extended to permit more general relations (such as $x+2y=6$ and $x^2-3xy-2y^2=6$), families of graphs, and graphs defined parametrically ($x=t-2:y=4+t$) or using polar co-ordinates ($r=1+2\cos\Theta$).

THE LISTINGS

The plotter consists of two programs:

GRPLOT performs a variety of initialisation tasks, including the preparation of the screen, before chaining:

GRPLOT1 which accepts all the input and does all the plotting.

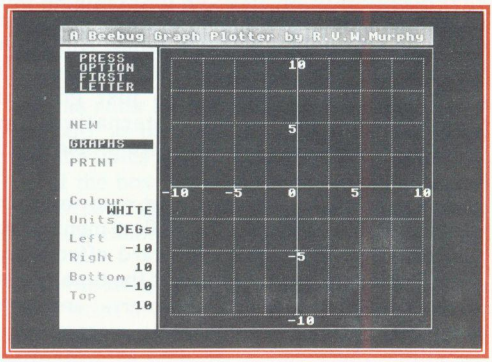
Each should be typed in and saved before attempting to run GRPLOT. The line numbering used in GRPLOT1 should be followed closely and it should not be "renumbered" as the error-trapping involves statements like:

```
C = ....
```

and:

```
GO TO C
```

Statements consisting of only a colon (:) have been used to show where the usual sequence of line numbers is altered, or where extra code will be added in Part 2. Some code may appear redundant at this stage as it will only be employed by the complete program.



Opening screen display

DEBUGGING

When testing GRPLOT1 (if you have typed it in from the magazine) it is sensible to remove the line:

```
120 ON ERROR
```

until any simple errors have been discovered. The program should be re-saved after each correction. Locking GRPLOT by:

```
*A.GRPLOT L
```

to prevent accidental over-writing is also a wise precaution. Pressing Break will re-start the whole program.

Line 130 will be required when the mathematics causes errors (such as DIVISION BY ZERO with $y=1/x$) but the line:

```
140 IF INKEY-1.....
```

is provided so that Shift-Escape will stop the program.

To make correction less tedious, the f0 key has been programmed to allow editing in mode 7, and the f1 key to restart GRPLOT1. This avoids the necessity to save every time. The colours will be incorrect to warn you to save the corrections at some stage.

RUNNING THE PROGRAM

The illustration shows the opening screen display. The menu, to the left, is used by

pressing the first letter of the option required. The cursor keys may also be used to initiate changes in the screen scales (cursor-left to change *Left* etc.).

For ease and speed of operation, the highlighted option may be selected by typing in the appropriate input or pressing Return. Thus the graph of y=x may be plotted by the instructions:

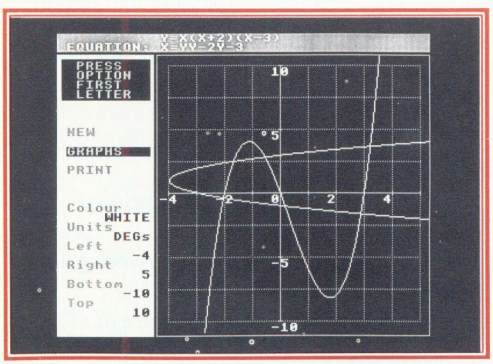
```
CHAIN"GRPLOT"     Return
Y=X               Return
```

provided you do not type too quickly! The menu system tries to anticipate your needs in order to speed the input process. For example, resetting the scales (Left, Right, Top and Bottom) to show 0<x<5 and -2<y<6 needs only:

```
 L     (or cursor left)
 0     Return
 5     Return
-2     Return
 6     Return
       Return
```

The initial letters are still available if the program guesses wrongly!

The scale values may be entered as decimals or expressions like 1/2, E^2 (where E=e=2.718...), TAN70, or 2PI (there is no need for 2*PI).



Examples of cubic and quadratic curves

The graph colours available are white, blue, and green (initially white), and the angle units are chosen from degrees and radians (default degrees).

The default screen scales (-10<x<10 and -10<y<10) and units (degrees) are set in GRPLOT by the line:

```
670 DATA -10,10,-10,10,0
```

and may easily be changed. Anything other than 0 at the end chooses radians for the units.

The *Print* option is included to allow you to patch into your own screen dump routine. Change the line:

```
280 *scrdump
```

to the command or call needed for your own system. Econet users could also put their *PS command here.

New will clear the current graph display, while the *Graphs* option changes the program to graph plotting mode ready to enter an equation.

Note that both when selecting from the menu options and when plotting graphs, Shift-Escape will exit from the program.

ENTERING EQUATIONS

This first part of the program will only plot Cartesian graphs which can be expressed either as:

y = expression involving x

or: x = expression involving y

anything else will be *TOO HARD!*, and the computer will warn you and wait until the next key is pressed.

To reduce the need for Shift, square brackets may be used instead of round ones and a ";" instead of a "+". Other *forgiving* features of the program include converting lower case letters to capitals and accepting the "underline" character for a "minus".

The equations may be entered in algebraic form. The program will insert "missing" multiplication signs. Repeating a letter is therefore equivalent to raising it to a power. This is best illustrated with a few examples which are suitable for a screen showing -3<x<3 and -5<y<5:

```
Y=X(X-1)(X+2)
Y=XXX    or  Y=X^3      i.e. y=x³
Y=XE^X   or  Y=XEXPX    i.e. y=x eˣ
Y=4SIN(50X)             4SIN50X is (4SIN50)X
X=1/(YY)          but: 1/YY is 1/Y*Y i.e. 1
Y=SQR(4-XX)
```

When entering a subsequent equation the cursor keys are enabled to allow copying of the previous one. Alternatively the right-hand side of the previous one may be modified using entries like:

-2, +3, *4, /5, ^2

This is particularly useful when building up a series expansion like:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

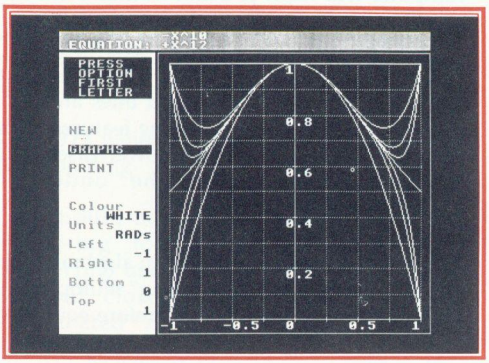
by:

```
Y=X                Return
-XXX/6             Return
+X^5/120           Return
-X^7/5040          Return
```

Use radians and 0<x<6 and -2<y<2 here.

THE PLOTTING

If the computer "beeps" repeatedly instead of plotting, you have an error! It could be purely mathematical, indicating that there is no curve for the current x or y value, and it will cure itself. It could be you have a typing or syntax error in your equation. If so Escape to the menu and enter the correct form. Otherwise you may still have another typing error to correct.



Illustrating $1/(1+x^2) \approx 1-x^2+x^4-x^6+\ldots$

No curve will be drawn if successive points are a long way apart as the program will assume a *discontinuity* (try Y=5+10(X<0) as an application of Boolean algebra). This can cause problems with curves having very steep gradients; changing scales is the usual cure.

After a plot, further equations can be entered or you can Escape or Return to the menu. The latter means that the sequence:

Return C Return

will ensure that the next graph will be drawn in a different colour.

PROGRAM DETAILS (PART 1)

PROCEDURES USED:

axis	print a value on an axis.
convert	change algebraic form to BASIC, including angle unit conversion and overcoming the "-x^2" bug in BBC Basic.
grid	define graphics origin and prepare graphics background.
input	accept input up to a specified maximum length.
lim1	accept new values for screen limits, and ...
lim2	refresh values shown on the screen.
message	display "help" message in options panel.
off/on	cursor control.
plot	draw the graph.
show	refresh menu to show current values and highlighted option.
top	leave OPTIONS message and create EQUATIONS window.
type	decide the type of (Cartesian) equation.

FUNCTIONS USED:

@	control format of any printed numbers.
int	decide the interval between grid lines.
swap	replace every occurrence of x$, in a$, by y$.
x/y	decide the x or y value for given value of y or x.

NOTE:In order to save memory space, and for other reasons, these programs do not conform to our usual style for program listings. Please take care when typing the programs in.

```
  10 REM Program GRPLOT
  20 REM Version B1.0
  30 REM Author  R.V.W.Murphy
  40 REM BEEBUG  October 1989
  50 REM Copyright strictly reserved
  60 :
  70 MODE1:HIMEM=&2D00:@%=0
  80 *K.0*FX11 11|M*FX12 4|M*FX4|MMODE7
|M|N
  90 *K.1*FX4 1|MMODE1|MHIMEM=&2D00:COL
.131|M|LGCOL0,128:VDU24,350;4;1274;952;:
CLG:RUN|MN
 100 *K.10CH."GRPLOT"|M
 110 REM Title
 120 VDU23;10,32;0;0;0
 130 VDU19,1,4;0;19,2,2;0;
 140 COLOUR3:COLOUR129
 150 PRINTSPC(80)
 160 MOVE32,1000:VDU5
 170 PRINT" Beebug Graph Plotter by R.V
.W.Murphy"
 180 VDU4
 190 REM Menu Titles
 200 N%=&2D00:O%=N%+12:O$="":FOR I=0 TO
 9:READ T$
 210 O$=O$+LEFT$(T$+O$,1)
 220 $(O%+I*10)=LEFT$(T$+"       ",8):NE
XT:$N%=O$
 230 M%=O%+I*10:G%=M%+10:A%=G%+10:B%=A%
+10:C%=B%+10:D%=C%+10:Z%=D%+10
 240 $M%="   WHITE"
 250 $G%="       "
 260 $Z%="DEGsRADs    BLUE   GREEN   WH
ITE"
 270 FOR I=A% TO D% STEP 10
 280 READ V$:$I=RIGHT$("        "+V$,8)
:NEXT
 290 a=VAL$A%:A=VAL$B%:b=VAL$C%:B=VAL$D
%
 300 READ R%:REM 1=RADIANS, 0=DEGREES
 310 R%=-4*(R%<>0):G%=G%+4:$G%=MID$($Z%
,1+R%,4)
 320 REM Prepare Screen
 330 L=382:R=1246:F%=48:T=912
 340 H%=T-F%:W%=R-L:L%=L:P%=0:Q%=2:E%=3
 350 REM Menu Panel
 360 GCOL0,131:VDU24,0;0;328;948;:CLG
 370 REM Info Panel
 380 GCOL0,128:VDU24,16;796;316;932;:CL
G
 390 REM Message
 400 COLOUR2:COLOUR128
 410 PRINTTAB(2,4)"PLEASE"TAB(3,5)"WAIT
"
 420 REM Border
 430 VDU26:GCOL0,2:MOVEL-40,F%-48
 440 PLOT1,0,H%+84:PLOT1,W%+72,0
 450 PLOT1,0,-H%-84:PLOT1,-W%-72,0
 460 REM Graphics Screen
 470 VDU24,L-32;F%-36;R+28;T+32;
 480 REM DEFAULT VALUES
 490 I%=Z%+33:K%=I%+80:J%=K%+80
 500 $I%="":$K%="Y":$J%="0"
 510 L%=1:REM CLG flag
 520 REM Character Definitions
 530 REM theta
 540 VDU 23,253,60,102,102,126,102,102,
60,0
 550 REM t
 560 VDU 23,254,48,48,124,48,48,48,28,0
 570 REM Define "beep"
 580 *fx212,168
 590 *fx213,201
 600 *fx214,1
 610 REM Load Main Program
 620 IF PAGE>&1200:PAGE=&1200
 630 *OPT 1 0
 640 CHAIN"GRPLOT1"
 650 :
 660 DATA NEW,GRAPHS,PRINT,,Colour,Unit
s,Left,Right,Bottom,Top
 670 DATA -10,10,-10,10,0

 100 REM "GRPLOT1" October 1989
 110 PROCoff:CLEAR:HIMEM=&2D00:PROClim2
:a$="":y$=a$:c=0:z=PI-2:DIMZ(6):C=0:PROC
grid:E=EXP1:e=E:h=E%:x=0:y=0
 120 ONERROR
 130 IFINKEY-113:ELSEIFERR<>17VDU7:IFC:
GOTOC
 140 IFINKEY-1:MODE7:OSCLI("FX4"):END
 150 IFC:RUN
 160 PROClim2:REPEATPROCoff:VDU4:C=0
 170 IFL:P%=L:VDU7:F=0:PROCmessage("LIM
ITS'HAVE'AN'ERROR"):ELSEPROCmessage("PRE
SS'OPTION'FIRST'LETTER"):IFF<0PROCgrid
 180 REPEATPROCshow:*FX4 1
 190 I=GET AND &DF:P=INSTR($N%,CHR$I)
 200 IFP:ELSEIFI<127P=P%:IFI>32OSCLI("F
X138,0,"+STR$I):ELSEIFABS(I-137.5)<2:P=I
-129:ELSE*FX15
 210 IFP=5:h=(h MOD3)+1:E%=h:P=0:$M%=LE
FT$($(Z%+h*8),8)
 220 IFP=6R%=4-R%:P=0:$G%=LEFT$($(Z%+R%
),4)
 230 UNTILP:P%=P:PROCshow:IFP=1F=L=0:PR
OCgrid:P=0:P%=Q%:ELSEIFP%>6PROClim1
 240 UNTILP>0ANDL=0
```

```
  250 IFP=3GOSUB280:ELSEQ%=P%:PROCgrid:P
ROCtop:GOSUB1000
  260 GOTO130
  270 REM YOUR OWN SCREEN DUMP ROUTINE
  280 *scrdump
  290 RUN
  300 DEFPROCshow:VDU28,0,31,9,0
  310 FORI=0TO9:Q=(I=P%-1):VDU31,1,10+I+
I:COLOUR1-Q:COLOUR131+3*Q:PRINT$(0%+I*10
)" ";
  320 IFABS(I-6.5)<3COLOUR-2*Q:PRINTCHR$
9" "$(M%+(I-4)*10)CHR$11;
  330 NEXT:ENDPROC
  340 DEFPROClim1:COLOUR3:COLOUR128
  350 PRINTTAB(1,9+2*P%)SPC(9)CHR$11CHR$
9CHR$9;
  360 PROCinput(7):PROCconvert:X=EVALa$
  370 F=1:L%=1:$(A%+(P%-7)*10)=RIGHT$("
      "+a$,8):P%=P%+1
  380 DEFPROClim2:F=L%:P%=Q%
  390 IFP%=11P%=1
  400 a=EVAL$A%:A=EVAL$B%:b=EVAL$C%:B=EV
AL$D%:IFa<A ANDb<B:L=0:P=0:ELSEL=P%
  410 ENDPROC
  420 DEFPROCtop:PROCmessage("PRESS'ESCA
PE'FOR'OPTIONS"):COLOUR129:COLOUR3:VDU28
,0,1,39,0,12:ENDPROC
  430 DEFFNswap(a$,X$,Y$):Y=INSTR(a$,X$)
:IFY:=LEFT$(a$,Y-1)+Y$+FNswap(MID$(a$,Y+
LENX$),X$,Y$):ELSE:=a$
  440 DEFPROCconvert
  450 a$=FNswap(a$,"EXP","e^")
  460 a$=FNswap(a$,"E^","e^")
  470 y$=FNswap(a$,"(-","(0-")
  480 a$="":p$=CHR$0
  490 FORI=1TOLENy$:c$=MID$(y$,I,1)
  500 IF(INSTR(".0123456789)XYK",p$)>0AN
DINSTR("(eACDILMNPRSTXYK",c$))OR(p$=")"A
NDINSTR(".0123456789eACDILMNPRSTXYK",c$)
):a$=a$+"*"
  510 IFR%:ELSEIFc$="A"ANDINSTR("SCT",MI
D$(y$,I+1,1))a$=a$+"DEG"
  520 IFR%:ELSEIFINSTR("IN.OS.AN.",p$+c$
+".")c$=c$+"RAD"
  530 a$=a$+c$:p$=c$:NEXT:ENDPROC
  540 DEFPROCinput(l):a$="":PROCon:REPEA
T
  550 REPEATM$=LEFT$(a$,1):M$=CHR$(254+(
M$="R")):y$=GET$:IFy$>CHR$140ORy$="t"y$=
"@":ELSEIFy$>="a"ANDy$<="z"y$=CHR$(ASCy$
-32)
  560 UNTIL(y$>" "ANDASCy$<96)ORy$=CHR$1
27ORy$=CHR$13
  570 y$=MID$(M$+CHR$13+"+()-"+y$,INSTR(
"@,;[]_"+y$,y$),1)
  580 IFy$<>CHR$127:ELSEIFa$>""a$=LEFT$(
a$,LENa$-1):VDU127:UNTIL0:ELSEUNTIL0
  590 IFy$>" "ANDLENa$<l:a$=a$+y$:PRINTy
$;
  600 UNTILy$=CHR$13:PROCoff:ENDPROC
  610 DEFPROCoff:W=32
  620 DEFPROCon:W=96
  630 VDU23;10,W;0;0;0:ENDPROC
  640 DEFPROCmessage(M$):VDU4,28,1,6,9,3
  650 COLOUR2:COLOUR128:CLS:REPEAT
  660 G=INSTR(M$,"!"):PRINT" "LEFT$(M$,G
-1);:IFG>0ANDG<9PRINT
  670 M$=MID$(M$,G+1):UNTILM$=""ORG<1:EN
DPROC
  680 DEFFNdegree(X,Y)
  690 FORI=0TO6:Z(I)=FNeval(z+I*X,z+I*Y)
:NEXT
  700 p=4:REPEATp=p-1:G=0
  710 FORI=0TOp:Z(I)=Z(I+1)-Z(I):IFABS(Z
(I))>G:G=ABS(Z(I))
  720 NEXT:UNTILG<1E-50Rp=0:=3-p
  730 DEFPROCgrid
  740 X=A-a:Y=B-b:f=W%/X:g=H%/Y:j=a*f:J=
A*f:k=b*g:H=B*g:VDU29,382-j;48-k;
  750 IFF=0ENDPROC
  760 F=1:GCOL0,2:CLG:L%=0
  770 MOVEj,0:DRAWJ,0:MOVE0,k:DRAW0,H:x=
0:x=FNint(X,f):D=0:IFk>0ORH<0D=k
  780 VDU5:d=0:IFj>0ORJ<0d=j
  790 Q=D:I=D:y=0:FORG=-INT(-a/x)TOA/x
  800 Z=G*x:P=Z*f:GCOL0,1:MOVEP,k:PLOT17
,0,H%:PROCaxis:IF(G MODN)OR(D ANDd)=0:GC
OL0,2:MOVEP,Q-8:PLOT1,0,16
  810 D=0:NEXT
  820 y=FNint(Y,g):D=I:P=d:I=d:F=0:FOR G
=-INT(-b/y)TOB/y
  830 Z=G*y:Q=Z*g:GCOL0,1:MOVEj,Q:PLOT17
,W%,0:PROCaxis:IFG MODN=0:GCOL0,2:MOVEP-
8,Q:PLOT1,16,0
  840 NEXT:VDU4:ENDPROC
  850 DEFFN@(X)
  860 @%=&1000600:X$=STR$ABSX:p=INSTR(X$
,"E")
  870 IFp:q=VALMID$(X$,p+1):IFq<0ANDq>-4
@%=&1020000+256*(p-q-2)
  880 =STR$X
  890 DEFPROCaxis:IF(G MODN)OR(D ANDd):D
=0:ENDPROC
  900 IFI=0GCOL0,3:ELSEGCOL1,1:IFG=0ENDP
ROC
  910 X$=FN@(Z):Z=P-64-J+32*LENX$:MOVEP-
32+Z*(Z>0),Q-8:IFF*Z<=0ORN=4:PRINTX$
```

Continued on page 58

1st course

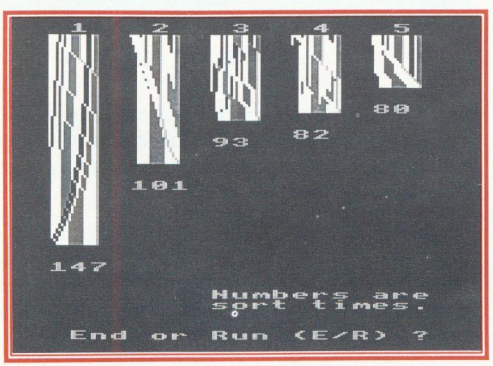
Visual Sorting

This month's First Course is contributed by Andrew Watkins and is a discussion of sorting methods illustrated in a highly visual way.

Sorting data, whether it be alphabetical or numerical, is a process commonly assigned to computers because of their speed and accuracy. There are many sorting routines, or algorithms, available, and each has its own good and bad points.

The purpose of the program listed here is to demonstrate graphically how these different algorithms work when applied to arrays of numbers. This is shown in graphical form, with the array to be sorted displayed horizontally and time being represented vertically down the screen.

The array is represented by a line of coloured squares, where each colour represents a number. The array, initially random, is sorted as a set of numbers, and the numbers are converted to colours for the display. A horizontal line is drawn each time a swap occurs during sorting, and slowly a pattern, typical of the particular algorithm used, emerges.



Comparison of 5 sorts

The program should be typed in and saved as usual. On running the program, you can choose from two modes of operation: comparing five sort algorithms, or viewing a single algorithm.

When comparing the algorithms, the array to be sorted contains twenty random elements, and the same set of data is used for each of the five algorithms, drawn side by side. The number of elements used here has to be a compromise between fitting in all five algorithms - leaving enough room to print as many iterations as possible - and being able to see the patterns generated.

When viewing a single algorithm, you can choose any of the five algorithms, and any array size, up to eighty elements. Some combinations can give very lengthy patterns, especially Bubblesort. Should this occur, and the pattern reaches the bottom of the screen, the bottom line is repeatedly re-drawn with the new array order.

The program also incorporates a timing mechanism, although demonstrating speed is not the purpose of the program. This times only the sort, 'stopping' the clock while drawing the array on the screen. Sort times are given in hundredths of a second. Sorting and drawing can be a slow process, however; the drawing process can take around 35 times longer than the sorting on large arrays!

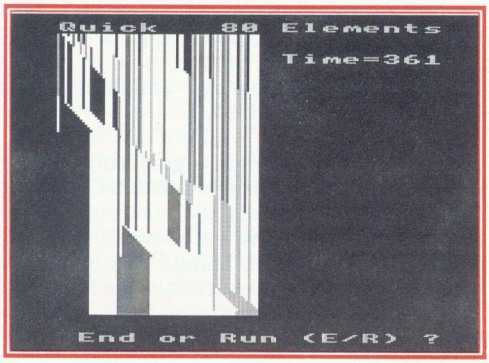
THE ALGORITHMS

All the algorithms rely on swapping pairs of elements in the array, if a pair is in the 'wrong' order. Not every comparison will therefore result in a swap. Each of the sorts described below is coded as a procedure with an appropriate name. This should ease the 'lifting' of any routine into a program of your own, but remember that the array used will need to be dimensioned before any procedure is called.

Bubblesort potentially swaps an element with its right-hand neighbour, and repeats this process, moving along the array again and again, until no more swaps occur. This is the slowest algorithm (though easy to understand and thus often used), except for arrays which are already nearly sorted.

This pattern is invariably the longest. If you can keep it from leaving the bottom of the screen,

lines of white or cyan move diagonally to the right. Once the whole pattern is generated, curved lines of colour appear to sweep from top right to bottom left. Anything over twenty elements is likely to reach the bottom of the screen.



Quicksort

Selectsort compares the first item in the array with every other and swaps them if necessary. It then repeats the process, comparing the second element with all those after it, and so on until the end of the array is reached.

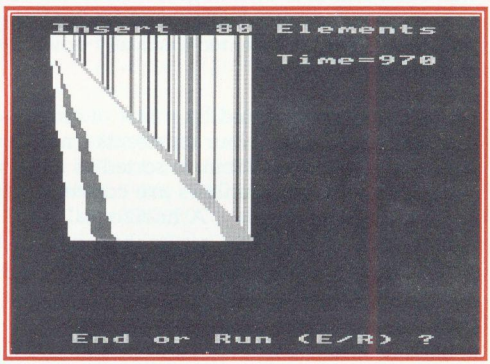
A line appears diagonally from the top left, and the sorted colours emerge directly under it. Above it, these colours are reflected as if a mirror were placed on the diagonal, but with streaks of other colours through them.

Shellsort (named after its originator David Shell) works like Bubblesort, but starts by comparing well separated elements, not neighbours. When no more swaps occur, the distance between elements compared is reduced. This is repeated until the distance between elements is one, i.e. the last step is a true Bubblesort.

It is difficult to analyse the pattern produced in this case to see what is happening, although bands of similar colours may be seen. There does not seem to be a pattern, merely 'fingers' of colour interlocking, pulling colours to one side or the other. This algorithm works from both ends of the array, sorting the middle last; unlike the other algorithms.

Quicksort chooses a number in the range of the elements in the array and places every value numerically below that number to one side of the number pointed to, and everything equal or above that value to the other side of the pointer. This is done recursively, until no more swaps can be performed. Quicksort is, for large arrays, by far the quickest algorithm, hence its name.

On small arrays, the pattern is difficult to see. On large arrays, it becomes obvious, though. An increasing band of higher-numbered colours moves to the right, and columns of colour, capped with a 'sloping roof' appear, in combinations, and in single colours. Try eighty elements with this one.



Insertsort

Insertsort moves a pointer along the array, picking off one element at a time and inserting it in its correct place, in much the same way as one would sort playing cards.

This algorithm looks the neatest, and usually involves the least number of swaps, on relatively small arrays. A diagonal line from top left to lower right marks the boundary between sorted and unsorted elements.

FURTHER READING

Many textbooks give examples of sort algorithms, but often these are not general. A common problem is to forget to cater for two equal values in the unsorted list (the *Advanced Disk User Guide* has this problem).

A book to be recommended is *Algorithms* by Robert Sedgewick, published by Addison-

Wesley (1983). This book contains many useful techniques, written in Pascal, though it is not difficult to translate them into Basic.

The idea for using colours to show sorting algorithms at work, came from the teaching program at Brown University, USA. The cover of Sedgewick's book shows Insertsort at work, using many more colours than are available on a BBC Micro.

```
   10 REM Program VSort
   20 REM Version B1.1
   30 REM Author  Andrew Watkins
   40 REM BEEBUG  October 1989
   50 REM Program subject to copyright
   60 :
  100 ON ERROR GOTO 300
  110 VDU23,255,192,192,192,0,0,0,0,0
  120 DIM random(80),obj(80)
  130 REPEAT
  140 max=15
  150 REPEAT
  160 MODE2:VDU5,23;8202;0;0;0;
  170 PROCwrite("S O R T",6,2,2)
  180 PROCwrite("=======",6,3,1)
  190 PROCwrite("1 Compare sort        a
lgorithms",1,5,7)
  200 PROCwrite("2 View single sort    a
lgorithm",1,8,7)
  210 PROCwrite("Choose 1 or 2",3,12,7)
  220 ask=GET-48
  230 UNTIL ask>0 AND ask<3
  240 CLS
  250 IF ask=1 PROCcompare
  260 IF ask=2 PROCsingle
  270 UNTIL key=69 OR key=101
  280 VDU4:CLS:END
  290 :
  300 MODE7:REPORT:PRINT" at line ";ERL
  310 END
  320 :
 1000 DEF PROCcompare
 1010 max=20:top=992:left=0:Y%=top:X%=le
ft:unfin=0:xmagn=8
 1020 PROCwrite(" 1   2   3   4   5",0,0
,7)
 1030 FOR I%=1 TO max
 1040 random(I%)=RND(7)
 1050 obj(I%)=random(I%)
 1060 NEXT:stime=TIME:wait=0
 1070 PROCprint:PROCbubblesort
 1080 PROCagain:PROCselectsort
 1090 PROCagain:PROCshellsort
 1100 PROCagain:PROCquicksort(1,max)
 1110 PROCagain:PROCinsertsort
 1120 PROCwrite("Numbers are",8,26,7):PR
OCwrite("sort times.",8,27,7)
 1130 PROCrerun
 1140 ENDPROC
 1150 :
 1160 DEF PROCsingle
 1170 PROCwrite("1 Bubblesort",2,3,7)
 1180 PROCwrite("2 Selectsort",2,5,7)
 1190 PROCwrite("3 Shellsort",2,7,7)
 1200 PROCwrite("4 Quicksort",2,9,7)
 1210 PROCwrite("5 Insertsort",2,11,7)
 1220 PROCwrite("Choose 1 to 5",4,14,7)
 1230 REPEAT:ask=GET-48:UNTIL ask>0 AND
ask<6:CLS
 1240 REPEAT
 1250 PROCwrite("Number of elements  (1
- 80)",0,3,7)
 1260 VDU4:INPUTTAB(0,6),max:VDU5:CLS
 1270 UNTIL max>0 AND max<=80
 1280 top=992:left=0:Y%=top:X%=left:unfi
n=0:xmagn=8
 1290 FOR I%=1 TO max
 1300 obj(I%)=RND(7)
 1310 NEXT
 1320 PROCwrite(STR$(max)+" Elements",8,
0,7)
 1330 stime=TIME:wait=0
 1340 IF ask=1 PROCwrite("Bubble",0,0,7)
:PROCbubblesort
 1350 IF ask=2 PROCwrite("Select",0,0,7)
:PROCselectsort
 1360 IF ask=3 PROCwrite("Shell",0,0,7):
PROCshellsort
 1370 IF ask=4 PROCwrite("Quick",0,0,7):
PROCquicksort(1,max)
 1380 IF ask=5 PROCwrite("Insert",0,0,7)
:PROCinsertsort
 1390 IF unfin=-1 PROCnearly
 1400 PROCrerun
 1410 ENDPROC
 1420 :
 1430 DEF PROCprint
 1440 entime=TIME
 1450 LOCAL I%
 1460 Y%=Y%-8
 1470 IF Y%<0 THEN unfin=-1:Y%=4
 1480 X%=left
 1490 FOR I%=1 TO max
 1500 MOVE X%,Y%
 1510 GCOL0,obj(I%)
 1520 PRINT;CHR$255
 1530 X%=X%+xmagn
 1540 NEXT:GCOL0,0:MOVE X%,Y%:PRINT;CHR$
255:GCOL0,7
 1550 wait=wait+(TIME-entime)
 1560 ENDPROC
```

```
 1570 :
 1580 DEF PROCnearly
 1590 entime=TIME:time=entime-stime-wait
 1600 PROCwrite("Time="+STR$(time),11,3,
7)
 1610 PROCwrite("Could not",10,6,7)
 1620 PROCwrite("quite fit",10,7,7)
 1630 PROCwrite("on screen",10,8,7)
 1640 ENDPROC
 1650 :
 1660 DEF PROCagain
 1670 entime=TIME:time=entime-stime-wait
 1680 MOVEleft,Y%-64:GCOL0,7:PRINT;time;
 1690 left=left+(1280/5)
 1700 Y%=top
 1710 FOR I%=1 TO max
 1720 obj(I%)=random(I%)
 1730 NEXT I%
 1740 stime=TIME:wait=0
 1750 PROCprint
 1760 ENDPROC
 1770 :
 1780 DEF PROCrerun
 1790 entime=TIME:time=entime-stime-wait
 1800 IF unfin=-1 GOTO 1820
 1810 IF max>20 PROCwrite("Time="+STR$(t
ime),11,3,7) ELSE MOVEleft,Y%-64:PRINT;t
ime;
 1820 PROCwrite("End or Run (E/R) ?",1,3
0,7)
 1830 REPEAT key=GET:UNTIL INSTR("EeRr",
CHR$(key))<>0
 1840 ENDPROC
 1850 :
 1860 DEF PROCswap(A%,B%)
 1870 fin=0
 1880 count=count+1
 1890 sto=obj(A%)
 1900 obj(A%)=obj(B%)
 1910 obj(B%)=sto
 1920 PROCprint
 1930 ENDPROC
 1940 :
 1950 DEF PROCselectsort
 1960 FOR J%=0 TO max-1
 1970 FOR K%=J%+1 TO max
 1980 IF obj(J%)>obj(K%) THEN PROCswap(J
%,K%)
 1990 NEXT K%
 2000 NEXT J%
 2010 ENDPROC
 2020 :
 2030 DEF PROCbubblesort
 2040 REPEAT
 2050 fin=1
 2060 FOR I%=1 TO max-1
 2070 IF obj(I%)>obj(I%+1) PROCswap(I%,I
%+1)
 2080 NEXT
 2090 UNTIL fin=1
 2100 ENDPROC
 2110 :
 2120 DEF PROCshellsort
 2130 int=max
 2140 REPEAT
 2150 int=int DIV 3
 2160 IF int=0 int=1
 2170 REPEAT
 2180 fin=1
 2190 FOR I%=1 TO max-int
 2200 IF obj(I%)>obj(I%+int) PROCswap(I%
,I%+int)
 2210 NEXT I%
 2220 UNTIL fin=1
 2230 UNTIL int=1
 2240 ENDPROC
 2250 :
 2260 DEF PROCquicksort(L%,R%)
 2270 IF L%>=R% THEN ENDPROC
 2280 T%=obj(L%)
 2290 M%=L%
 2300 FOR I%=L%+1 TO R%
 2310 IF obj(I%)<T% THEN M%=M%+1:PROCswa
p(M%,I%)
 2320 NEXT I%
 2330 IF obj(L%)<>obj(M%) PROCswap(L%,M%
)
 2340 PROCquicksort(L%,M%-1)
 2350 PROCquicksort(M%+1,R%)
 2360 ENDPROC
 2370 :
 2380 DEF PROCinsertsort
 2390 FOR I%=2 TO max
 2400 J%=I%:T%=obj(J%)
 2410 IF J%<=1 OR obj(J%-1)<=T% THEN GOT
O 2450
 2420 obj(J%)=obj(J%-1)
 2430 J%=J%-1
 2440 GOTO 2410
 2450 obj(J%)=T%
 2460 PROCprint
 2470 NEXT I%
 2480 ENDPROC
 2490 :
 2500 DEF PROCwrite(text$,xtab,ytab,col)
 2510 VDU4
 2520 COLOUR col
 2530 PRINTTAB(xtab,ytab);text$
 2540 VDU5
 2550 ENDPROC
```

Adventure Games

by Mitch

Game	**Avon**
Supplier	**Topologika P.O.Box 39, Stilton, Peterborough PE7 3RL. Tel. (0733) 244682**
Price	**£17.50 inc. VAT. (disc)**

Ill met by moonlight, proud adventurer. Now's your chance to brush up your Shakespeare - and have a lot of fun doing it. Topologika has taken the Bard's world and created a stage upon which you may now strut and fret your hour in its latest adventure game called *Avon*.

Whilst wandering the sunlit streets of modern-day Stratford, your mind lost in the past, a sudden turn in the road leaves you lost and confused in the half-world of Shakespeare's characters. Up ahead lies the barren moor where three hags await your coming. What magical offering will you accept from their bony claw to help you on your way? Eye of Newt perhaps, or Tongue of Dog. Better take something, you're going to need all the help you can get.

No need to worry that your grasp of the rhyming couplet is going to prevent you solving this all-text adventure. Whilst the game is liberally sprinkled with characters and quotes from the master, you'll need no prior knowledge to help you. Of course you are free to swell your doublet with pride each time you recognize the source!

Falstaff sits gulping ale in the nearby tavern, and providing you can make it through the doorway without being drowned in a butt of malmsey wine, you might just care to take up his challenge of a drinking match. Beware the jealous rage of the dark Moor, (of the African variety), who waits upon the dark moor, (of the Scottish variety), to strangle any passing stranger. You should also avoid falling under the sensuous spell of the Egyptian Queen who lies amid a cloud of perfumed air aboard her silk-sailed, love barge in the nearby harbour. For the cartographers amongst you there is the maze of Birnham Wood to negotiate. Let's hope that you can make it through its leafy glades before it becomes bored with your efforts and start to move things along on its own accord.

You might like to help an old king choose which of his daughters should be his heir, or sit inside a laundry basket and wait for something to turn up. There goes that strange fellow, still looking for his horse. Pity you don't know where to find one, he would appear to be willing to pay a king's ransom. Either way, the games the thing, and fun is king.

Topologika has a well-deserved reputation for quality adventures and I know that its game designers strive to give their puzzles that little something extra. If you have already wrestled with the other games from the same stable such as Philosopher's Quest, Countdown to Doom and Return to Doom, you'll already know what devious minds they have. They were also responsible for the large Acheton adventure; so beloved of the hardened adventuring classes - and the curse of the more feeble-minded such as myself!

Should the puzzles prove to be too much for you, the game designer has thoughtfully included a comprehensive Help facility which can cure all the ills that flesh is heir to. The puzzles seem to be pitched at a level suitable for the majority of players and the game's text is humorous despite its worthy title. So boot-up your trusty disc drive and it's - 'Once more into the breech dear friends'.

A Datafile Superdump

Gareth Leyshon describes a utility which should prove invaluable to anyone attempting to debug file handling programs by analysing the contents of data files.

The article *File Handling For All (Part 12)* (see BEEBUG Vol.8 No.2) explained how to use the *DUMP command to investigate database files as a debugging aid, giving the identifying codes for the different types of data (integer numbers, real numbers, and strings) that are used with INPUT# and PRINT# instructions. However, it is tedious to decode manually the information thus provided.

From Basic, you can use OPENIN to open a file and then use INPUT# to load the values and test them, but if you are working with a complex file with different types of data, while changing pointer values to investigate different parts of the file, you may well end up getting "type mismatch" errors if, for example, you try to read a string when the pointer is on an integer, and vice versa.

Wouldn't it be much easier if the computer could automatically identify each value in the file, tell you what type it was,and show you its value? That is what the listing accompanying this article does.

Type in the program as shown, and save it. When you wish to use it, first select the filing system you need (TAPE, DISC or ADFS) and type RUN.

The program gives you the opportunity to catalogue the filing system if you wish to - but do not use this facility with cassette. If you do want a catalogue, answer "Y" to the prompt.

When prompted, enter the name of the file to be investigated. You will be asked if you require a printout. If so, put your printer on-line and press "Y".

The program will then dump the contents of the file, displaying this in three columns. The first is the position in the file (the PTR# value) in hex. The second is the type of variable stored. The third is its value.

There are four possible responses for each piece of data analysed. Real and integer values are shown in cyan and yellow respectively. String variables are displayed in green. The string is shown in full if possible. The column is 20 characters wide, but the text will flow onto the next line when the first is full. If the string contains codes greater than 126 or less than 32, these are displayed as a red hex value in brackets within the printable characters.

If the program finds a value which does not seem to belong to any of these three types, it will classify it, byte by byte, as "Unknown!" and will display the byte's hex value in red in brackets in the value column. Where appropriate, this will be followed by its ASCII character in magenta. Remember that a bug in a file handling program may well corrupt one of the data fields, which is then no longer recognisable as one of the three standard data types, hence the fourth or *error* category described above. Be warned though. Unrecognised or corrupted bytes in a datafile can be misleading to the program, so that on some occasions valid data is not recognised as such. All will be revealed in the display.

```
SUPERDUMPING orders
Point  Type        Content
000000 Integer..: 1
000005 Integer..: 20
00000A Integer..: 10
00000F Integer..: 36
000014 Integer..: 4
000019 Integer..: 1
00001E Integer..: 5
000023 Integer..: 11
000028 Integer..: 3
00002D String...: ITEMS (&00) (&00)
000036 Integer..: 2
00003B String...: NAME
```

Sample dump of a typical data file

A value such as (&0FFFFFFF) in strings and unknowns is typically a hex representation of a negative number, so if this comes up, don't think it's an error.

When the end of the file is reached, the program will give a message "Dump complete", and display the total length of the file.

If you have also asked for a printout, you may find that any graphics characters appear as spaces - these are generated by teletext control codes. These should not spoil the layout of the printout, but if you wish to remove them (and get a monochrome screen display) then replace every CHR$(x), where x is between 129 and 137, with a CHR$(32).

You can try the program with many type of database files, but other types of file are likely to give misleading and confusing results. The program should be compatible with all versions of Basic and Acorn filing systems, and prove an ideal companion for those developing and debugging file handling programs.

```
   10 REM Program Superdump
   20 REM Version B2.2
   30 REM Author  Gareth Leyshon
   40 REM BEEBUG  October 1989
   50 REM Program subject to copyright
   60 :
   70 ON ERROR VDU3,15,26,12:CLOSE#0:REP
ORT:PRINT " at line ";ERL:END
   80 MODE7
   90 VDU 14,131
  100 PRINTSPC(15)"SUPERDUMP"
  110 PROCpreliminary
  120 Channel%=OPENIN(Filename$)
  122 IF Channel%=0 PRINT'"No such file:
 ";Filename$:END
  130 REPEAT
  140 VarType$=STR$~(BGET#Channel%):IF L
EN(VarType$)=1THEN VarType$="0"+VarType$
  150 PTR#Channel%=PTR#Channel%-1:Positi
on%=PTR#Channel%
  160 Position$=STR$~Position%:Position$
=STRING$((6-LEN(Position$)),"0")+Positio
n$
  170 ON (INSTR("40FF00",VarType$)+1)/2
GOSUB 250,410,310 ELSE PROCother
  180 UNTIL EOF#Channel%
  190 PRINT"Superdump of ";Filename$;" c
omplete."'" File length:&"~EXT#Channel%"
 bytes."''
  200 CLOSE#Channel%:VDU 15,3,26
  210 PRINT TAB(2,23)"Press any key to e
nd the program.":*FX 21,0
  220 IF GET
  230 CLS:END
  240 :
  250 REM integer subroutine
  260 PRINT Position$;CHR$131;"Integer..
: ";
  270 INPUT#Channel%,Integer%
  280 PRINT;Integer%
  290 RETURN
  300 :
  310 REM string subroutine
  320 PRINT Position$;CHR$130;"String...
: ";
  330 INPUT#Channel%,String$
  340 FOR Loop%=1 TO LEN(String$)
  350 Character%=ASC(MID$(String$,Loop%,
1))
  360 IF Character%<127 AND Character%>3
1 THEN PROCprintchar(Character%) ELSE PR
OCprintcode(Character%)
  370 NEXT
  380 VDU 10,13
  390 RETURN
  400 :
  410 REM real number subroutine
  420 PRINT Position$;CHR$134;"Real.....
: ";
  430 INPUT#Channel%,Real
  440 PRINT;Real
  450 RETURN
  460 :
 1000 DEF PROCother
 1010 PRINT Position$;CHR$133;"Unknown!.
: ";
 1020 Code%=BGET#Channel%
 1030 VDU 129,40,38,(-48*(Code%<16)):PRI
NT;STR$~Code%;:VDU 41
 1040 IF Code%>31 AND Code%<127 THEN VDU
 133,Code%
 1050 VDU 10,13
 1060 ENDPROC
 1070 :
 1080 DEF PROCprintchar(Char%)
 1090 IF POS>=38 THEN PRINT'SPC(17);:VDU
 130,Char% ELSE VDU Char%
 1100 ENDPROC
 1110 :
 1120 DEF PROCprintcode(Char%)
 1130 IF POS>32 THEN PRINT 'SPC(17);
 1140 VDU 129,40,38,(-48*(Char%<16)):PRI
NT;STR$~Char%;:VDU 41,130
 1150 ENDPROC
 1160 :
 1170 DEF PROCpreliminary
 1180 PRINT'"Page mode on: Press Shift t
o scroll."
 1190 PRINT'"Do you require a catalogue
(Y/N)?"
 1200 IF FNyn THEN *CAT
 1210 INPUT LINE '"File to dump? "Filena
me$
 1220 PRINT'"Do you require a hard copy
(Y/N)?"
 1230 IF FNyn THEN Hard%=2 ELSE Hard%=3
 1240 VDU12,30,32,32,32,32,32,132,136,Ha
rd%
 1250 PRINT"SUPERDUMPING ";Filename$'"Po
int  Type";SPC(7);"Content"
 1260 VDU 28,0,24,39,2:ENDPROC
 1270 :
 1280 DEF FNyn:*FX21,0
 1290 REPEAT
 1300 Key%=INSTR("YynN",GET$)
 1310 UNTIL Key%>0
 1320 =(Key%<3)
```

BEEBUG Education

by Mark Sealey

INTRODUCTION

A number of small but significant developments in IT use in education have occurred over the summer, and since the last BEEBUG Education was published. This month's feature briefly outlines what has been happening, and also includes a book review.

RECENT DEVELOPMENTS

It should be noted, in passing, that the supply of software for the BBC and Master machines is not so plentiful as it once was. This is an indication that, although still nominally supporting the older, 8-bit, hardware, the educational thrust of most publishers (including Acorn, which has recently ceased production of the Compact, for example) is on the Archimedes range - especially the new A3000.

On a rather negative note, the BBC announced in July that the BBC Ceefax Telesoftware for education was to cease with just one month's notice (indeed all Ceefax Telesoftware operations have now ceased). This is a pity, to say the least, since many schools and colleges have invested in Teletext adaptors specifically to avail themselves of the many interesting programs which until now have been down-loadable via this service.

This appears to be part of a cost-cutting move connected with substantial pruning of the entire BBC Soft project. Given the income which the BBC derives both directly from Acorn machines and indirectly through their association with education, it is to be regretted that we are increasingly badly served by it.

Just as teachers were preparing to go on holiday at the end of last term, the Design and Technology Working party published their section of the National Curriculum. It was greeted with a number of horrified and strident reactions in the informed press.

Papers like the Times Educational Supplement suggested gloomily that the report's minimal requirements of pupils would seriously undermine progress in IT of which British schools could be proud. Especially where some of the most innovative work is being done: at the lower end of the age spectrum. Acorn's Managing Director, Harvey Coleman, gave apparently equally pessimistic evidence to a Parliamentary committee at about the same time.

In both cases, one thing - money - or the lack of it is a major cause for concern amongst educationalists. It would certainly be most interesting to have YOUR views on this and to know just how deeply LEA underfunding is affecting what you are doing in schools.

Another DES publication worthy of attention is *Information Technology 5-16*, published by Her Majesty's Inspectorate in the *Curriculum Matters* series. Although not relating exclusively to any one sector or phase of education, nor indeed to Acorn machines in particular, it should be read closely by anyone working in or interested in this field.

It is a refreshing document, addressing some of those questions of standards and curriculum which certain other recent official publications do not! To read this short book is to be reminded of much familiar good practice and several laudable goals. There is nothing startlingly new here, but the early sections (pages 2 to 8) would form an admirable shell around which you could construct and debate an IT policy to see your school/college well past the National Curriculum fad.

Anyone unsure of what they can contribute to their establishment (or as a parent to their own child's education) ought to be encouraged by some of the informed debate in this publication. There is much talk of planning, and an emphasis on the need for IT to be integrated with what pupils are already working on in such a way that they see the connections between them for themselves.

It is not hard to think of many excellent software titles for BBC machines that go a long way towards this. The crunch would seem to come when teachers are insufficiently prepared or experienced in the deployment of IT across the curriculum for this to happen.

One project set up specifically to help in this direction was a casualty of the demise of the MEP. The Special Education Microelectronics Resource Centres - the four SEMERCs - did not have their life prolonged in the same way as that of one of the other success stories of the 1980's IT Education scene, the MESU/NCET partnership.

Now, consortia of LEAs have increased their support for erstwhile SEMERCs and three exist in one form or another: the old Manchester site has moved to Fitton Hill CDC in Oldham, at Rosary Road, OL8 2QE (tel. 061-627 4469). No fewer than a dozen LEAs from the South West and South Wales have rescued the Bristol one, which can be contacted at the old address at Bristol Polytechnic: Redland Hill, BS6 6UZ or tel. (0272) 733141. The ever enterprising Resource, which is based at Doncaster - and which should be making some quite exciting announcements in the near future (watch this space) - now houses the old Newcastle SEMERC, called a SENSU, Special Educational Needs Support Unit. Paul Meakin may be contacted at Resource or on (0302) 340331.

BOOK REVIEW

Computer Assisted Learning in the Humanities and Social Sciences edited by Kent and Lewis, Blackwell Scientific Publications £27.50.

The use of computers in education has long since advanced beyond the stage when general publications will satisfy the experienced or specialist user - or indeed the 'expert' who wants to experiment. For instance an international journal is shortly to be published (*Computer Assisted Language Learning* - Intellect Books, Suite 2, 108/110 London Road, Oxford OX3 9AW) covering aspects of language learning and computers.

In a not dissimilar vein, a very wide-ranging collection of contributions is contained in the book under review this month. Some two dozen closely argued and well presented chapters survey the field of Humanities and Social Sciences with many references to software that will run on the BBC/Acorn machines.

The studies are written mostly by acknowledged experts with experience from a range of backgrounds and educational systems. To read how countries like the Netherlands and New Zealand have tackled IT in several curricular areas, for example, is quite revealing. The material contained in this 200-odd page publication is not for the casual reader, though. Whilst being a book that can be read selectively, and not necessarily in strict sequence, it is definitely not of the 'hints and tips' type.

In fact, the volume gathers together the papers given at a recent conference in London, and presents them under six thematic headings. What these six are (and how they are linked) in itself stands as a significant guide to current concerns and issues in IT and education. *Handling Data* would provide a useful commentary on some of the issues discussed in the last BEEBUG Education; while *Curriculum Developments* forms a stimulating complement to debates both on resourcing and curriculum change.

Practising teachers will perhaps find the third, fourth and sixth sections most interesting. *Teaching Strategies, Children Learning* and *Strategies for CAL* deal with both fundamental questions of whether to use a computer at all, as well as individual evaluations, from a strictly pedagogical point of view, of the type of programs regularly reviewed in BEEBUG Education.

The fifth section looks at *Developing Software* and refers, for example, to MicroProlog, another piece of software available for and used on Acorn computers - though not as widely as it deserves to be, if some of the conclusions reached in this book are to be credited.

There is a rather basic index but each chapter is also summarised, and suggestions laid out for some directions in which IT in education might go. This expensive book, then, is not for everyday unwinding in the staff room. It will prove both a useful source and provoker of discussions on certain types of course and more substantial INSET sessions. The material is relevant to all three educational phases but with a bias towards the Secondary.

If you find much of the anecdotal material elsewhere relating to software and hardware uninspiring, then it is likely that selective use of specific Chapters of *Computer Assisted Learning in the Humanities and Social Sciences* will give you food for thought. B

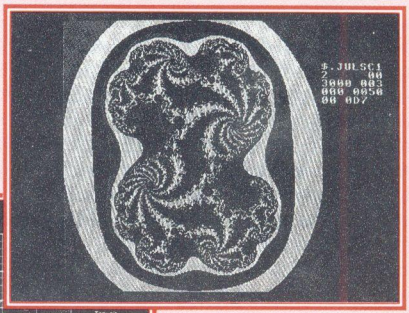
The Best of BEEBUG
Applications II

• Share Investor

A useful program for the increasing number of share investors. This program not only allows you to monitor the fluctuations in share prices, but also assists the decision making process when buying and selling stocks and shares.

• Running Four Temperatures

Monitor the temperature of your home, greenhouse, fish tank or even fridge using this four channel temperature probe project. Associated programs allow for calibrating and plotting temperatures.

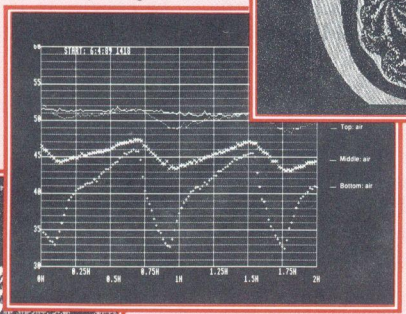


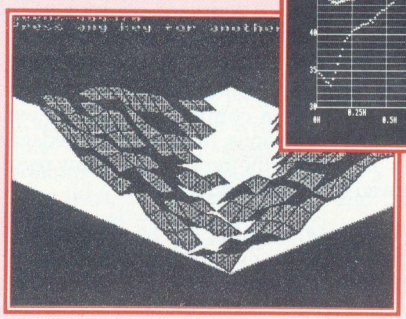
• Label Processor

Do you need to create your own labels? This program will help you design and save labels quickly and easily, and print them at any size on an Epson compatible printer.

• 3D Landscapes

Create an infinite number of computer-generated three dimensional landscapes with this program.





• Monthly Desk Diary

A month-to-view calendar which you can use with your favourite word processor to enter your appointments and print out for handy reference.

• Real Time Clock

You can have a very useful real time digital clock displayed permanently on your screen. The clock will work in any mode and on any machine, and will even provide an alarm when necessary.

• Crossword Editor

Design and edit your own crosswords or use this program to solve crosswords and anagrams. The program can optionally access SpellMaster.

• Julia Sets

Fascinating visual programs for Basic II and Basic IV displaying Julia sets, the extensions of the Mandelbrot set.

• Foreign Language Tester

Define foreign language character sets for screen and printer in French, German and most European languages. Test your knowledge in foreign languages with the accompanying programs and sample French-English files.

Please rush me my Best of BEEBUG Applications II disc at the members price of £5.75
(non-members price £15)

Code 1411A *(80 track DFS)* ❑ Code 1412A *(3.5" ADFS)* ❑

Name _____

Address _____

_____

Memership No _____

Price £_____

Postage £ 0.60

Total £_____

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

Send to: BEEBUG Ltd, 117 Hatfield Road, St Albans, Herts AL1 4JS. Telephone (0727) 40303.

RISC USER

The Archimedes Support Group

Our Risc User magazine will shortly start its third year of publication and is enjoying the largest circulation of any magazine devoted to the Archimedes. The list of members seeking support from the Risc User group is growing steadily and as well as private individuals includes schools, colleges, universities and industry and government establishments.

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

Here are just some of the topics covered in more recent issues of RISC User:

A COLOUR IMAGE PROCESSOR
A powerful program by Roger Wilson for manipulating screen images.

UNDER THE LID
A tour of the Archimedes Hardware.

THE ARM 3 RISC PROCESSOR
An in-depth look into this exciting development by its designer.

THE R140 UNIX WORKSTATION
A look at Acorn's entry into the business market.

PARTIAL RENUMBER UTILITY
A machine code utility for renumbering parts of a Basic program.

MASTERING THE WIMP
A major new series for beginners to the WIMP programming environment.

DESKTOP BASIC HANDLER
A Desktop utility to convert basic programs into their text equivalents and vice versa.

A MULTI-TASKING NOTE-PAD
A comprehensive RISC OS multi-tasking application which offers a multi-page, multi-file notepad with editing and printing facilities.

INTO THE ARC
Series for newcomers to the Archimedes.

RISC USER TOOLBOX
A comprehensive toolbox module for the Archimedes.

MOUSE MENU
A general purpose WIMP based menu, which is fully RISC OS compatible.

RISC USER DESKTOP DIARY
A multi-tasking Desktop diary facility.

ARM CODE SINGLE STEPPER
An ARM code debugging aid.

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

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

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

SUBSCRIPTION DETAILS

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

For existing members new rates apply after 31 Dec 1989.

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

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

Signed: ..... *Expiry Date:* _____/ _____/

Send to: RISC User, 117 Hatfield Road, St Albans, Herts AL1 4JS, or telephone (0727) 40303

512 Forum

by Robin Burton

At last it appears that we have come to the end of the Problem Solver saga, and I'm pleased to say that the conclusion appears to be a happy one.

PROBLEM SOLVER LATEST

Joaquim Carvallio, one of the authors of Problem Solver (I hope I got the surname right), visited the UK in late July with the intention of identifying and curing the difficulties experienced by some users of his program.

He called on me on the 2nd of August and I'm pleased to say that he delivered a new copy of the program which appears to suffer from none of the difficulties I'd previously found. We spent an interesting afternoon and evening discussing various aspects of the 512, including the reasons for applications incompatibility, and all in all a good time was had by all.

Joaquim explained to me how the program works and was kind (and confident) enough to leave the source code with me too. Knowing what PS (Problem Solver) sets out to achieve it's clear how, in the previous version this led to two particular areas of difficulty. I promised not to give away too many secrets, so the explanations which follow are fairly general, but even so I hope that the reasons for the problems experienced by some with earlier versions of the program will become clearer.

PROBLEM RESOLVED

After gaining access to other 512 systems (he himself has a Master based system) he has been able both to identify these problems and to cure them. He was quite confident that the new version of the program would be much more reliable in all versions of BBC micro/512 system, and my own tests bear this out.

The first difficulty was that the earlier version of Problem Solver had assumed that various operating system variables were always located at certain addresses. While this is true some of the time it isn't absolutely reliable. This was the reason that PS simply would not work at all, or might partly work, but only sometimes.

The program has been changed so that these locations are now interrogated by legal calls, rather than being 'hard coded'. The result is that the new version installs itself correctly because the relevant memory addresses are now accurately known. So far as I have been able to test it (using DOS Plus 2.1, where most of the difficulties occurred) the program now always installs correctly and I've had no hang-ups with the new version except a few when trying out programs that I knew didn't normally run in the 512.

HOW PS WORKS

When Problem Solver is running correctly, what it sets out to do is to examine the code in applications so that it can locate and replace certain instructions which are known to give trouble in the 512. This examination and replacement can occur either as a program is loaded, if PS is enabled at the time, or subsequently by pressing <Shift-Ctrl-Copy followed by P>.

Some 80186 instructions always cause trouble in the 512, and as has been mentioned before, direct hardware access is the main area. Problem Solver attempts to find these instructions within the code and replace them, depending on the original instruction, with an invalid code which it will later recognise.

At the same time, it intercepts the 512's 'invalid opcode' error routine, so that when the application comes across one of these (replaced) invalid instructions, Problem Solver takes over, executing the appropriate legal sequence of instructions, identified by the (invalid) code. Control is then returned to the original program after the operation has been carried out.

Naturally this technique is quite fiddly to implement, and one reason for Problem Solver's previous 'delicacy' with some programs was that embedded data can sometimes look like a 512 incompatible instruction. The result was that in some cases bytes would be changed which should not have been.

The 'instruction search and replace' routine has been amended in such a way that Joaquim believes it is now much less likely to make this type of mistake. This means not only that its success rate with incompatible packages should be better, but interference with already compatible programs should be reduced.

As an indication of this, one package I use has always worked perfectly without Problem Solver, but it now also works normally with it, even in my own system. When running the previous version of Problem Solver with this package (in a Master, which was the only way I could get it to work at all) it always crashed the system, so this claim seems to be justified.

PROBLEM SOLVED?

As it now seems that Problem Solver has matured into a reliable product I'll comment on the type of help it can be reasonably expected to offer. This might give some indication of the type of program which is likely to benefit from its use and to some extent these can then be predicted.

We'll start by eliminating the main groups of program that Problem Solver can't help.

It might seem unnecessary to say so, but obviously programs which require a 640K (or even a 512K MS-DOS) machine, or which need a different graphics adaptor to the CGA are first on the list. Next, any program which is not compatible with MS-DOS or PC-DOS 2.1 will remain incompatible. This is because these programs will by definition use operating system calls and facilities (like the extended memory system) which are simply impossible or missing in DOS Plus 2.1.

Additionally, utilities which attempt to do 'clever' things with a PC disc controller still won't work, because the BBC micro's WD1770 disc controller used by the 512 is not the correct type. Even if you gave it the 'correct' instructions in a legal manner it wouldn't know what to do with them.

Enough of the negatives, what about programs that can be helped? Perhaps the easiest to categorise are those which do run unaided, but produce a garbage display. This can happen for a number of reasons, but the most common is that the screen mode change within the program doesn't work in a 512.

The DOS Plus utility provided on issue disc 1, 'PCSCREEN', is a special for the 512, needed because all 512 displays are actually done (as far as the BBC micro is concerned) in a mode 3 screen. The different character sizes and colours in the 40 column graphics display are actually achieved by re-programming the BBC's 6845 CRTC registers directly, and by MOS calls to re-program the video ULA. Obviously PC programs know nothing about any of this.

Because Problem Solver can issue the 512's 'mode change' command correctly, in some cases it will correct the screen display automatically. In other cases you might need to assist programs which otherwise display rubbish by manually changing the screen mode through Problem Solver.

It should be said that this type of problem doesn't apply to 80 column text output, and is therefore essentially restricted to games and graphics programs. However, if this is your problem PS may well hold the solution. The list of software it 'fixes' is quite lengthy, even though games do form a sizeable proportion.

I'll be honest and admit that I don't have any PC games of my own, but Joaquim did show me a dozen or more that definitely don't run without Problem Solver, but certainly do when it's installed. I can therefore personally confirm this point too.

Another area which is improved is keyboard handling. Programs that read keyboard entry through a character input function don't give

trouble, but for various reasons some try to read the keyboard directly and don't work in the 512. This doesn't usually cause such problems as a crash, but instead a program will probably do nothing at all, failing to respond to particular keys and in some cases ignoring all of them.

By intercepting the keyboard input interrupt vector (INT 9), Problem Solver traps some of these calls and implements them legally. The result is that the calling routine receives the correct keyboard response and therefore functions accordingly.

I've tried this too, with a couple of shareware TSRs that rely on 'hot keys' and which totally ignore key depressions without Problem Solver. With it they function correctly. Again this depends to a degree on the particular program, because I also have other programs which still don't work even when Problem Solver is active.

To some extent common sense can tell you which programs might be helped and which won't, so long as you have a bit of information on their operation. For example, if a program requires a key which simply doesn't exist in the 512 system, for example function keys 11 to 20, it's unreasonable to expect Problem Solver to overcome this. However, if the keys required are available on your keyboard but the program doesn't work on its own, there's a good chance that Problem Solver will do the trick.

Users of model B and B+ micros should note that Master numeric keypad emulation is provided via one of the three function key definitions. On the first setting they operate as usual, the second selection provides insert, delete, +, -, *, /, left-shift, right-shift and scroll lock, while the third gives insert, end, cursor down, page down, cursor left, cursor right, home, cursor up and page up. I've had mixed success with these and again it obviously depends on the program in question.

THE FINAL DECISION

Ultimately I can't tell you whether to buy Problem Solver or not. As I said to Joaquim, I still feel that it's a little over-priced, given that there can be no guarantee that it will cure a specific problem. It's only a personal opinion and you might disagree, but I think £15 or £20 would have been a better price, since it's still a gamble whether any particular package's problems will be solved or not.

Of course there are two ways to look at it. If it makes an otherwise useless £150 package into your favourite piece of software you will no doubt think it's excellent value. If you've managed without Problem Solver so far, it's equally possible that you, like me, have simply avoided problem packages and found alternatives that suit you and do the job. If your current range of software does everything you need you might have little to gain from Problem Solver.

However, if there's a specific package (or several) that you are keen to use and you think it falls into one of the categories outlined above, then Problem Solver might be just what you need.

What I can say is that the latest version of Problem Solver is clearly more robust and reliable than previous ones, and I would now expect it to be well behaved in all systems. It does meet its claims for a respectably large number of packages that will not run correctly in the 512 without it. As such it can now be recommended. There isn't space this month, but in the next Forum I'll include a list of packages that Problem Solver is known to 'fix'.

A few final points will, I'm sure, interest existing users. If you own an early copy of Problem Solver and have had trouble, Shibumi Soft will exchange it for the current version if you return the original disc. They will also replace any disc which has become corrupted. Finally, the latest version is unprotected, allowing hard disc users to transfer Problem Solver to their Winchester.

MAGAZINE DISC

512 users should note that this month's magazine disc contains the second of two MS/DOS utilities by Bernard Hill, together with a documentation file covering both (repeated from last time). This month's program is a SORT utility. B

Essential Software for the 512

Bernard Hill reviews the first in a new line of software specifically for 512 enthusiasts.

Product	**512 Ramdisc Utility**
Supplier	**Essential Software** **P.O.Box 5, Groby,** **Leicestershire LE6 0ZB.**
Price	**£14.95 inc. VAT and p&p**

A new software house has arrived on the scene. Essential Software consists of our regular contributor Robin Burton in collaboration with Mike Ginns in a new venture to supply software to users of Acorn micros. Their aim is to produce useful and novel utility software at sensible prices, and their first clutch of products is aimed at the undersupported area of the 512 co-processor. At the moment there is only one utility disc available, but more are planned.

The first release consists of an 800K disc containing three programs to help you organise your disc drives and create a RAM disc in memory, and works on DOS+ 1.2 or 2.1.

The utility MEMDISK supplied with the 512 is virtually unusable due to the ludicrous 90K overhead which it consumes, in addition to the RAM disc itself (see 512 Forum, Beebug Vol.7 No.4). This disc from Essential contains its definitive replacement.

The program RAMDISC (notice the English spelling!) performs an identical function but uses only a 1K system overhead - plus the size of your RAM disc of course. Furthermore, you're not stuck with drive M: as with MEMDISK; you can use any drive letter except C:, that's reserved for the Winchester, but any letter from D: to K: Sadly you can't expand the size of the RAM disc without deleting it, but you can make it any size you like (in 2K units), and here comes the best news: it's deletable without rebooting your 512.

This is the function of the second program on the disc: AMNESIA (good name that). Provided the RAM disc is empty, this simple command removes it from the 512's configuration and gives you back the RAM. Simple.

The third program on the disc is harder for me to find a use for, but in the old days of PC software, some packages insisted on using fixed drives such as A: or B: (or even C:). To the rescue comes DISCID, which allows you to interchange the names of any two drives. For instance, the command DISCID A: B: would swap drives A: and B:, or you could even swap drives A: and M: to run such old-fashioned software from the RAM disc. DISCID can also be used to remove drives from the system (giving "Invalid Drive Specification") if you're developing software potentially damaging to your Winchester, perhaps. Or it can just report on which drives actually exist.

The documentation supplied with the three programs is minimal, a READ.ME file on the disc which keeps the costs down, but when there is so little need or documentation that's no problem: I already have a shelf full of no-longer-read glossy manuals, and every set of bad parameters I've given the programs results in a sensible error message.

So there you are. £14.95 will bring you these three programs, but to my mind it's worth it for a RAM disc you can really use (and it works with the PC PLUS) and already I am finding mine indispensable. If you don't like the risk of running on RAM disc all the time, put your most commonly run utilities on drive M:; they'll load lightningly fast if you preface the command with M:.

Spin a Disc (6)

David Spencer concludes this mini-series with a look at the recovery of lost files.

There can be little that is as infuriating as spending hours working on a file only to accidentally lose it. However, this situation all too frequently arises, either as a result of a file being deleted by mistake, or the disc containing the file becoming corrupted. In the first case, the file's entry will be removed from the catalogue but the actual contents are still intact, and in the second case the file will still be in the catalogue, but the contents will be corrupted in some way.

PREVENTION IS BETTER THAN CURE

As with most things in life, prevention is better than cure when it comes to file security. By making regular backups you can greatly reduce the damage done when a file is lost. In the best case situation, the file in question will not have been changed since the last backup, and recovering it is merely a minor inconvenience. More likely is that the file will have been modified since the last backup, and hence some work will be lost. However, by varying the period between backups depending on how often and by how much a file is changed, this loss can be minimised.

For example, if you had a disc of programs that was only modified once in a blue moon, then it is sensible to back it up only after any changes have been made. On the other hand, with a stock control system for a large company, it might prove wise to perform several backups each day. Another consideration is how valuable the data in question is. For example, consider the possible effects if a bank's transaction records for just one hour were lost!

So, having considered when to backup a disc, the other choice to make is how. The simplest way is to keep a second disc which is simply a copy of the first. One of the discs is used as the day-to-day working disc, and this is backed up onto the other disc at the chosen interval. However, this system is not without problems. Suppose, horror of horrors, a power failure occurs at the precise time that the backup is being done. It is possible that the entire contents of both discs would be lost, and we would have been better off not having a backup in the first place. An alternative method is to keep several backup discs, and use them in rotation. This way, even if a disaster does occur during the backup, there will still be an older version to fall back on.

The techniques just described are equally applicable to a floppy or hard disc. However, it can be impractical to perform regular backups of an entire hard disc. Instead, hard discs are normally handled using a so-called incremental backup. With this method, the entire disc is backed up once, and the date and time at which the backup took place noted. Each subsequent backup then copies only the files that have been created or modified since the last backup was made.

THE REAL WORLD

That's enough of the ideal world - in real life files do get lost, and they need to be recovered, but how?

The first thing is to ensure that you have the correct tools. You will need the following:

1. A disc sector editor
2. A utility to read disc sectors directly into memory.

3. Pen and paper

4. Patience!

A sector editor is a utility which allows individual sectors of the disc to be examined and modified regardless of their contents, or position on the disc. With such a tool you can potentially alter any piece of data on the disc, and therefore a sector editor must be used with care because you could do more damage than good. Most 'programmer's' utility ROMs contain sector editors, a particularly good one being the Advanced Disc Toolkit (ADT) from PRES, which has a sector editor that will work with both DFS and ADFS format discs. BEEBUG Vol.8 No.3 contained a sector editor for ADFS discs.

The second requirement is a utility that can read a block of disc sectors directly from disc into memory. Unlike *LOAD, these work by specifying a start track and sector, and a length, rather than a filename. Hence any sectors can be read, regardless of whether the filing system thinks they form part of a file or not. Again, many utility ROMs contain suitable commands. Alternatively, the General Purpose Disc Access ROM from the Workshop in BEEBUG Vol.8 Nos.1 & 2 can be used to implement both sector editors and readers for DFS and ADFS.

THE DREADED MOMENT

The first thing to do when you discover that a file has been deleted or corrupted is to remove the disc from the drive, and above all, not to panic. It is absolutely vital that the data on the disc is not changed further, hence the reason for removing it. Before going any further, stop and think. Make sure that you don't waste time recovering a file only to discover that you have another copy somewhere else, and also consider the possibility that the file might still be in the computer's memory. This may sound unlikely, but say for example that you deleted instead of renaming the file you were working on. If you had been editing the file at the time, then it is likely to still be in memory. Even if the file in memory appears to have been wiped, it is still worth saving all of the RAM using:

```
*SAVE Core E00 8000
```

For example, a file in the View word processor will be lost when Ctrl-Break is pressed, but the text is still in memory and can be recovered in this way.

The action to take next depends on whether the file has been accidentally deleted, or the disc has been corrupted. We will consider the former first, as it is more straightforward. Before recovering the file it is necessary to locate its whereabouts on the disc. To do this, you can use the *MAP command to display details of the free space on the disc. This produces a list of all the areas of free space in the form of a start sector and a length. The older 8271 DFS does not support the *MAP command. In this case, *INFO *.* can be used to find out the start sector and length of each file on the disc. Using this information, and remembering that a file always occupies a whole number of sectors, you can construct a free space map by working out which sectors are not used.

If a file has been deleted, then the space used for that file will now appear to be free, and will therefore appear in the free space list. If you have a rough idea of how long the file is, then you can look for a free space entry which is of the same length. This is particularly true when the disc was compacted prior to the file being deleted, because in this case all the free space will have been collected into one chunk at the end of the disc. But, NEVER compact a disc after a file has been lost, as the data contained in it will almost certainly be lost. For example, the *MAP command might produce the following output:

```
Start   Length
045     400
114     7C00
```

Assuming that this is a 40 track single sided disc, then it will contain 400 (&190) sectors. Hence, the second entry in the list above represents the area of free space at the end of the disc. It is therefore fair to assume that the deleted file started at sector &45 and was between &301 and &400 bytes long (because of the length of the chunk of free space). In any case, the next move is to check all the possible areas of free space until the correct one is found. This is done using the sector editor to examine the first sector of each chunk, starting with what you see to be the most likely candidate. It should normally be fairly easy to identify a lost file, particularly if it was plain text or a well commented Basic program. As soon as you find the file, make a note of its start sector.

One thing to be very wary of is thinking that you have found the lost file, when in fact you have come across an older version which was 'over written' a long time prior. You might therefore want to step through the file with the sector editor, and try to identify features and changes that you know exist only in the latest version. Another possibility is that you won't find the file at all. This does not mean that the data has been erased, rather that it does not start at the beginning of a free space chunk. This can happen when the lost file was preceded immediately on the disc by an area of free space. The filing system will concatenate the areas, and hence the file will begin in the middle. In this case, all that is necessary is a more exhaustive search through the free space areas using the sector editor. Remember though, the lost file will always be contained entirely within an area of apparent free space, and will always start at the beginning of a sector.

Having found the file on the disc you can load it directly into memory using the sector reading utility. If you know the length of the file, then you can round that up to a whole number of sectors and read them in. If you don't know the length then it is safest to read the entire chunk of free space. For example, in the above example if we knew the file length was &125 then the command to read the sectors using ADT would be:

```
*SECTORS E00 45 2 R
```

The '2' specifies the number of sectors, and the 'R' is for Read. In this example, the data was read into memory at address &E00 which is suitable for a Master. For a model B you should substitute &1900, or a higher address if 'memory hungry' ROMs are fitted.

After the sectors have been loaded, they can be saved as a named file to a new disc using *SAVE. This will result in a new file which contains the contents of the lost file, and most likely some 'junk' at the end corresponding to the extra data read in. Getting rid of this extra, unwanted, data depends on the contents of a file. With a Basic program, simply loading and resaving it is enough to remove the garbage from the end. A text file can be loaded into a word processor where the junk will appear at the end of the document and can be deleted with a block delete command or similar. The pruned file can then be re-saved. For machine code programs, and other types of data files the situation can be more involved, and you will need to study the file's format to work out what should be in it, and what shouldn't.

The procedure for recovering a lost file may seem rather long winded, but is straightforward once you have mastered it. You might like to try it out by deliberately deleting an unwanted file and recovering it!

RECOVERING CORRUPTED FILES

When dealing with a corrupted file the situation is rather different. In this case we can find the exact whereabouts of the file on the disc using the *INFO command to obtain the start sector and length, but we can't load the file because it is corrupted. It is quite likely, though, that only a single sector will be damaged and hence the recovery action involves salvaging what is left of the file.

The first operation is to find which sectors of the file are corrupted. This can be done by using the *VERIFY command to find out which sectors on the disc are damaged, and then subtracting the start sector number of the file to find where the sectors are within the file. In general, the file will consist of a number of sectors before the damaged bit, followed by the bad sectors themselves, and then some more good sectors. A sector reading utility can be used to read in the blocks of good sectors and save them to a separate disc. All that can be done is to then try and patch the file together by combining the recovered sections and making up as best as possible for the lost sectors. Unfortunately, such a recovery operation is very much hit and miss, and you should consider whether it would not be easier to recreate the lost data from scratch.

The worst case of disc corruption is when the sectors holding the catalogue have been lost. Here, all the files are intact but the filing system has no way of identifying them. The only course of action is to search the disc sector by sector with a sector editor and locate each file in turn. The sectors constituting the file are then read into memory and saved to a new disc - an operation requiring a lot of patience. In any case, a corrupted disc should never be re-used - all the surviving files should be copied off, and the disc thrown away.

Introducing Postscript (2)

David Spencer concludes his introductory look at this page description language.

Last month we looked at the general features of *PostScript*, and explained how printed objects are laid out on the page. In this article we turn our attention towards *PostScript* as a programming language, and give some examples of its use.

LANGUAGE STRUCTURE

Like Basic, *PostScript* is an interpreted language, meaning that programs are executed directly, rather than first being converted to machine code. However, this is where the similarity ends, because, unlike Basic, *PostScript* falls into the class of languages known as 'Threaded Interpretive Languages' (TILs) - Forth being the most widely known.

The main feature of a TIL is that rather than having a complete program, standalone procedures are built-up using the primitives provided by the language. More complex procedures can then be built-up from these and so on. The eventual result is a main procedure which is effectively an entire program, but which is built-up from procedures at a lower level. You can visualise such an arrangement as a tree diagram. The root corresponds to the main program, and the children of any node are the procedures making up that node, with the leaves being the primitives provided by the *PostScript* language.

For example, consider a simple program to calculate (x+2)/(x-2) for any value of x. *PostScript* provides all the normal arithmetic functions, and while we could perform the operation with a single expression, we could also write two procedures called *add2* and *sub2* which would use these primitives to perform the operations of adding and subtracting two. We could then implement another procedure, say calc, to perform our calculation by using *add2* and *sub2*. Figure 1 shows the tree structure for this.

Obviously this is a very contrived example, but the important thing to note is that once *add2* and *sub2* have been defined they become a part of the language (albeit temporarily), and can be used in just the same way as any primitive of the language. On the other hand, with a language such as Basic, individual procedures are bound tightly to a particular program, and cannot be used directly by any other program.

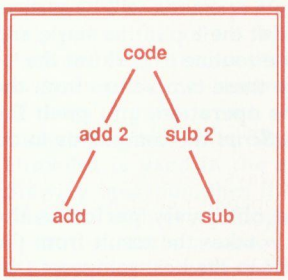


Figure 1. The tree-like structure of Postscript procedures

The way that *PostScript* implements this system is by means of a data structure called a dictionary. As with a printed dictionary, the *PostScript* form consists of a number of entries, each consisting of a key and a corresponding definition. In this case, the key is the name of a primitive or defined procedure, and the definition is either the code for implementing the primitive, or a list of the existing procedures making up the procedure in question. To speed up the searching of the dictionary, each entry has a further field - a link to the next entry. Hence, all of the entries are threaded together as shown in figure 2. This is the origin of the 'T' in TIL.

STACKING IT UP

Another characteristic of most TILs is that they perform all arithmetic, and many other operations, using a stack. For those unfamiliar with the term, a stack is a data structure onto which items can be *pushed*, and off which they can be *pulled*. However, things must be pulled off the stack in the exact reverse order to which they were put on. This is analogous with a pile of cards on a table. Cards can be added to the top (pushed), or removed (pulled). But, as you can only get to the top of the pile, the only card that can be taken off is the one on the top.

Arithmetic is performed on a stack based system using a method known as Reverse Polish Notation (RPN). The Polish term arises from the nationality of the inventories, though same would say that it is because RPN is about as easy to understand as Polish! With RPN, the operands of an expression are pushed onto the stack, and the operators (+, -, etc.) take the appropriate number of operands from the stack, perform the operation, and push the result back. For example, consider the sum 12 + 8. With RPN the two operands will be pushed in turn, so that 8 is at the top of the stack, and 12 just below it. The routine to perform the '+' operation will take these two values from the stack, perform the operation, and push the result back. In *PostScript* the commands to do this would be:

```
12 8 add =
```

The *add* keyword obviously performs the addition, and the = takes the result from the stack and outputs it to the host computer (the remote computer sending data to the printer).

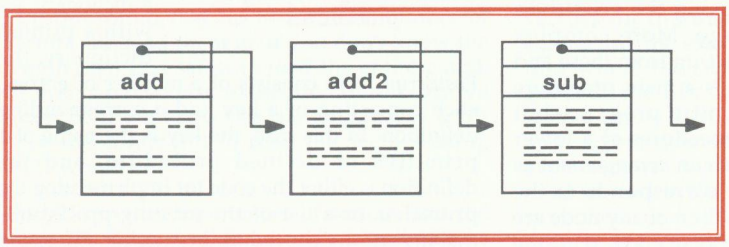


Figure 2. The threaded structure of a Postscript dictionary

A more complex example of RPN would be the expression (4+5)*2. The *PostScript* for this is:

```
4 5 add 2 mul =
```

You will notice that brackets are not necessary with RPN. Because operators always take their operands form the stack, the precedence of them can be allowed for ny changing their position. For example, the PostScript for the expression 4+5*2 is:

```
4 5 2 mul add =
```

Here the multiplication is performed before the addition.

Converting between normal expressions and RPN is a skill which is quickly acquired. This month's magazine disc contains a program (written in Basic) which will do the conversion automatically for expressions consisting of numbers, +, -, *, /, and brackets. The program produces its output as *PostScript* statements, using *add, sub, mul, div* and *neg* for the operators, the latter being unary negation.

PostScript goes a lot further than using a stack simply for arithmetic. In *PostScript,* the operand stack, as the main stack is called, can hold a wide range of data types, such as, a number, a string, an array, an executable procedure, or an entire dictionary. We will come back to some of these data types a little later. For example, the following sequence of PostScript commands will set up the printer so that all subsequent text (until the next change) will appear in 12 pt Helvetica typeface:

```
/Helvetica findfont
12 scalefont
setfont
```

The first line stacks the literal 'Helvetica' (a literal is a constant string, such as a procedure or font name), and then executes the operator *findfont*. This takes the literal off the stack, locates the appropriate font definition in the printer's ROM, and pushes the definition back onto the stack. A font definition is stored in the form of a dictionary, with a number of key-value (definition) pairs indicating details about the font, including the characters available and their outline shape.

The second line of the code above pushes the value 12 onto the stack, and executes *scalefont*. By definition, all fonts or stored as 1pt representations. The function of *scalefont* is to take a scale factor from the stack followed by a font dictionary, scale the appropriate bits of the font, and push the new amended dictionary back onto the stack. The final command is *setfont*. This takes a font dictionary from the top of the stack and *sets* it as the current font. You can easily verify that exactly the same number of operands are

pushed onto the stack as are pulled off. Therefore, the above code could be executed in the middle of a complex RPN expression, and both would work properly.

A POSTSCRIPT PROCEDURE

Having explained the fundamentals of the language, we will now present a definition for a fairly simple procedure, as this is the best way to demonstrate a number of language features. Our procedure will move to a specified position and draw a square box on the paper. The edges of the box will be 50 units in length. Initially these units will be points, so each edge will be 50/72", however as explained last month the scaling can be changed so that one unit is whatever length you want it to be. We will call the procedure *DrawBox*, and its definition is:

```
/DrawBox
{ gsave newpath
moveto
0 50 rlineto
50 0 rlineto
0 -50 rlineto
closepath
1 setlinewidth
stroke
grestore
} def
```

During the following explanation of how the procedure works, you might like to keep a track of the stack contents on paper.

The first line of the definition pushes onto the stack the name of the procedure - *DrawBox*. The '/' specifies a literal, as with the font name. The next section of the definition consists of all the text between the '{' and '}'. These brackets specify a procedure definition, and *PostScript* will build up a new procedure consisting of the values and calls to the operators. Because this is a procedure definition none of the commands are executed, they are simply added to an array which will form the procedure definition internally. When the final '}' is encountered the array representing the definition is pushed onto the stack.

One point to note here is that in *PostScript* an array is simply a collection of objects treated together, with no restriction as to what the objects are. Therefore, each element of a array can be of a totally different type. Back to the definition, and the final part is the *def* which takes the procedure array and then the name from the stack, makes them into a new dictionary entry, and puts them in the current dictionary, making *DrawBox* part of the language.

Having defined the procedure, we can now use it with a call such as:

```
100 200 DrawBox
```

This places the values 100 and 200 onto the stack and then executes the commands forming the body of the procedure definition. The first of these is *gsave* which saves the so-called current graphics state. This is done so that if *DrawBox* is used in the middle of a larger drawing operation, then it won't inadvertently interfere with everything else. The *newpath* operator starts a new drawing path, as explained last month, and *moveto* moves to the position specified by the top two values on the stack. These will be the values put there before *DrawBox* was called, and hence in this case specify the co-ordinates (100,200).

The next three commands draw the bottom, right and top edges of the box in that order. *rlineto* again takes two parameters, this time the x and y distances to move by. The *closepath* operator finishes off the box by drawing the fourth edge. This method is used in preference to drawing a line back to the start for the reasons explained last time. The *setlinewidth* operator specifies that the box is to be drawn with lines 1pt thick, and the *stroke* performs the actual drawing. Finally, the *grestore* returns the graphics state to how it was before the call was made, by performing the reverse of the earlier *gsave*.

If we were to define the procedure as above, and execute it on a laser printer, then nothing would happen. This is because we have to issue a *showpage* operator before anything is actually printed. As its name suggests, this prints out (shows) the current page, and also wipes it ready for the next page. A related operator, *copypage*, does the same, but doesn't erase the page, allowing multiple copies of a page to be printed.

PRINTING TEXT

Placing text on a page is very straightforward. The first operation is to select the correct font at the correct size using the code shown earlier. You then move to where you want the text to be positioned, using *moveto*, and print the text using a command of the form:

```
(Hello World) show
```

Parentheses are *PostScript's* way of specifying a text string, so in this example 'Hello World' is pushed onto the stack as a string. The operator *show* takes a string off the stack and renders it character-by-character using the current font on the page. The spacing between characters is determined by tables built into the font definition, and is therefore effectively fixed. There are several more adventurous forms of the show command which allow the width of each character to be adjusted, or just the width of one special character changed. This latter form is normally used when fully justifying text to increase the width of spaces. A 'super-dooper' form of show is *kshow*, which calls a specified procedure between each character. This allows the user to implement kerning, which is the moving together of certain characters to make them visually more pleasing. For example, if A and V are placed next to each other then the fact that the right edge of the A slopes in the same direction at the left edge of the V gives an impression of extra space between them. To cure this, it is customary to overlap the characters slightly, as shown in figure 3. The procedure called by *kshow* can keep tables of 'overlap' for each possible pair of characters, these being called *kerning* tables, *and perform the necessary move.*

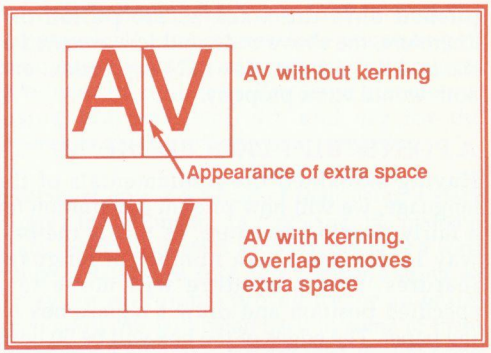


Figure 3. The process of kerning

As a demonstration of the theory covered so far, figure 4 shows the result of both our *DrawBox* procedure and a text printing call. The rotated part was done by using the command:

```
45 rotate
```

Which rotates the user space co-ordinate system, as explained in last month's article. Hence, the output of all graphics commands will appear rotated.

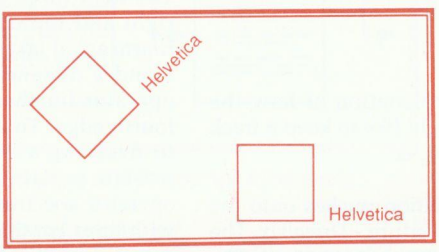


Figure 4

FINDING OUT MORE

Obviously in the very limited space of these two articles, we have only been able to scratch the surface of what is a very powerful programming language. The next issue of BEEBUG will contain a program for creating a dump of a Beeb screen on a *PostScript* printer. This program will use many of the operators covered here, together with a number for rendering arbitrary bit-images which we haven't looked at. These will be explained briefly next time. If you are interested in finding out more about *PostScript*, then I highly recommend the books referred to last month. They are the Cookbook, Language Reference manual, and Program Design manual. The first of these books explains how to perform various tasks in *PostScript*, while the second is a complete guide to all the features of the language. The third book is really aimed at experienced programmers. All the books are published by Addison-Wesley, and should be available from computer science bookshops. B

Font Designer (2)

Ian Stewart adds the final touches to the Font Designer, and provides a comprehensive description of this professional looking software.

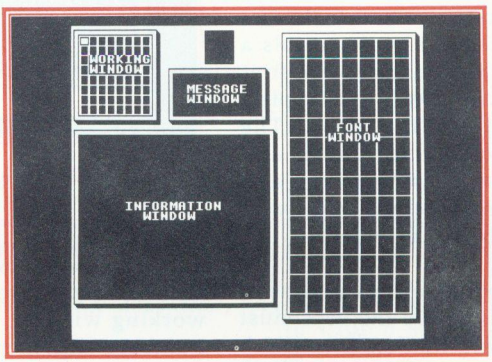
It will be easier perhaps if you start by typing in this month's code and adding this to the original program, as you will then be able to try out all functions as they are described. It is essential that the line numbering is unaltered in both the original and new listings. Type in the new code from this issue as a separate program, and then save in ASCII format with:

```
*SPOOL CDF2
LIST
*SPOOL
```

Then load the original program, and type:

```
*EXEC CDF2
```

finally saving the complete program as *Design*. However, in case of error, make sure you retain a copy of the original program under some other name (say CDF1). To run the program you will need to run *SetUp* which was listed last month, which in turn chains *Design*.



Layout of Font Designer screen

BOXED WINDOWS

The main display produced by the program shows a number of boxed windows as in figure 1. These are described below.

Working Window

This is positioned at the top left corner of the screen. It is in the form of a grid, eight squares by eight, in which the user can define or edit a character. There is a smaller window above the message window (q.v.) which shows the contents of the working window eight times smaller, the same size as a character in the font window.

Message Window

This is in the middle next to the working window on the screen. This window tells the user the instruction or function just called. It is also used for inputting a file name when saving or loading fonts.

Font Window

This window, at the right of the screen, shows the whole of the current font that can be used by this program. There are also two other spaces at the bottom right-hand side of the window which are not part of the font but which may be used as work space by the user.

Information Window

Normally this window shows the list of functions available to the user. There are two lists, one where the functions are accessed from function keys direct, and the other where the functions are accessed by Shift and the function keys. The former is the default but this can be changed with the *i%* variable at line 180.

Since this window is the widest, it is also used by some functions that need to use a larger area on screen, such OS commands, or listing a tape or disc catalogue.

STARTING THE PROGRAM

When the program is first run (using *SetUp)* the screen goes blank and a bleep is sounded. This indicates that the screen display is being created ready for use. It then makes another bleep and the four window display appears. The program then waits for a function to be called, or the cursor in the working window to be used.

To fill or unfill a square on the grid of the working window, a cursor must be moved to the appropriate square, and the Ctrl key pressed. This toggles the square from filled to unfilled or vice-versa. The cursor keys on the right of the keyboard move the cursor round the grid. If the Ctrl key is held down while the cursor is moved then the squares are filled/unfilled wherever the cursor is, although they are not toggled.

USING THE FUNCTIONS IN THE PROGRAM

Sixteen different functions are available.

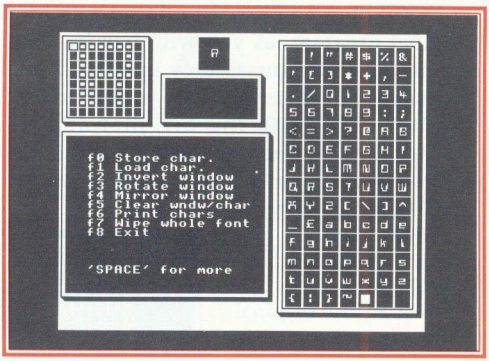
Storing a character means saving the character definition from the working window to the font window. Once selected the user controls a cursor to pick out the square in the font window to which the character is to be copied. Press Return to effect the copy. The corresponding ASCII code (and letter) is also displayed on the screen. This function is also used to save characters to the two workspaces in the bottom right of the font window.

Loading a character is effectively the reverse of the above process. As before the user must select a square in the font window from which the character will be copied into the working window, where it can be edited. The *Invert* function can invert (or toggle) all the squares in the working window, or just the current row or column, depending upon the next key pressed.

Rotate is a function which 'rotates' the whole contents of the working window by ninety degrees in either a clockwise or anti-clockwise direction. The *Mirror* function can reflect the whole grid in the working window either from top to bottom or left to right.

Clear asks for a square in the font window to be cleared. *Wipe* is a global version of clear which (with due caution) will clear the entire font in the font window, but leave the two workspaces intact.

The *Print* function provides three different forms of printout. The first dumps the whole font to the printer as seen in the font window. The second uses the information window and asks for a character string of not more than two hundred and fifty characters, and then sends this to the printer using the current font. This can be useful when testing newly defined characters against other characters.



Example of a newly designed font

The third and last output asks for a square in the font window. As with *clear*, the space bar can be used to select the contents of the working window. Once selected, the eight numbers representing the design of the character are shown in the instruction window in binary, hexadecimal and decimal.

Select Disc makes the current filing system disc (DFS or ADFS depending upon the setting of the variable *Disc$* at line 180). *Select Tape* makes tape the current filing system. *Catalogue* asks for a drive and makes a list of file names on the current (or default) directory on the specified drive. This can also be used with tape.

To *Load a font* the user must enter a specific file name. If tape or ADFS is active then the file name must not exceed 10 characters, 7 characters for DFS. For disc only, a catalogue of the current drive is shown first. The same applies to *Saving a font*. Fonts are loaded into or saved from the font window. Saving a font therefore creates a permanent copy.

Delete file can only be used when a disc filing system is active. The *Star command* option is

Load Routine for Model B

```
?&367 =  112                          - The font flag. This tells the computer if a 'font zone' is
                                        RAM or ROM. The number 112 means use the main
                                        ASCII characters 32 to 127.
?&368 =  (font address) DIV 256       - characters 32 to 63
?&369 =  ((font address) DIV 256)+1   - characters 64 to 95
?&36A =  ((font address) DIV 256)+2   - characters 96 to 127
```

&368 to &36E are the font location bytes which tell the computer where each character can be found. Type '?&367=15' to reset to default font.

Load Routine for the Master

```
M%=font address       - Set pointer to the start of the font address
FOR F%=32 TO 127      - Loop for character number
VDU 23,F%             - Start defining character
FOR N%=0 TO 7         - Loop for pointer
VDU N%?M%             - Get number of pointer + loop
NEXT N%
M%=M%+8               - Get ready for next character
NEXT F%
```

Type '*FX25' to reset to default font. See the Advanced User Guide for more details.

Table 1. Font load routines for model B and Master

obvious as to its purpose and uses the information window.

Lastly *Restore font* can recover a badly corrupted font. Each time a font is loaded, or the program is run a copy is made. This copy replaces the one in the font window. The Restore function will restore the contents of the font window to be the last loaded font, or if one has not been loaded, to the default font in ROM when you first switch the computer on. This provides a useful starting point when designing your own fonts.

USING A FONT IN YOUR PROGRAMS

To use any font in a program, the programmer must allocate 367 (&300) bytes of memory for each font to be used. Load a font using the command:

```
*LOAD <font_name> <font_address>
```

where <font_address> is the memory area reserved for the font. A small Basic routine must be run before any text can be displayed on the screen. The details of the routine are different for the model B and the Master, as shown in table 1.

A typical routine on a model B might be:

```
10 *LOAD U.myfont 900
20 ?&367=112
30 ?&368=&9:?&369=&A:?&36A=&B
```

on a Master:

```
10 *LOAD U.myfont 2000
20 M%=&2000
30 FOR F%=32 TO 127
40 VDU 23,F%
50 FOR N%=0 TO 7:VDU N%?M%:NEXT N%
60 M%=M%+8
70 NEXT F%
```

NOTE: A number of fonts supplied by the author are included on the magazine disc for this issue. There is also a small correction to last month's listing - the variable Disc$ in line 180 was set to "ADFS" and not "DISC" as stated in the text.

```
 1020 *|" *** INVERT *** "
 1030 :
 1040 COLOUR128:COLOUR1
 1050 PRINT TAB(12,5)"'V' Vert."TAB(12,6
)"'H' Hori."TAB(12,7)"'W' Whole"TAB(12,8
)"Esc Quits";
 1060 SOUND3,-14,200,4:Z%=1:E%=0
 1070 xx%=x%:yy%=y%:x2%=0:y2%=0
```

```
 1080 REPEAT
 1090 IF INKEY-113 THEN E%=1
 1100 IF INKEY-34 THEN E%=2:x%=1:y%=1:Z%
=8:x2%=1
 1110 IF INKEY-100 THEN E%=3:y%=1:y2%=1
 1120 IF INKEY-85 THEN E%=4:x%=1:x2%=1
 1130 UNTIL E%>0
 1140 IF E%=1 THEN 1210
 1150 FOR N%=1 TO Z%:FOR F%=1 TO 8
 1160 PROCbit(gr%(x%,y%))
 1170 PROCcur(FALSE,FALSE)
 1180 x%=x%+x2%:y%=y%+y2%
 1190 NEXT:y%=y%+1:x%=1
 1200 NEXT
 1210 PROCreset
 1220 RETURN
 1230 :
 1240 *|" *** ROTATE *** "
 1250 :
 1260 COLOUR128:COLOUR1
 1270 PRINT TAB(12,5)"'C'   CWZ"TAB(12,6
)"'A'  ACWZ"TAB(12,7)"Esc Quits"
 1280 SOUND3,-14,200,4
 1290 E%=0:xx%=x%:yy%=y%
 1300 REPEAT
 1310 IF INKEY-113 THEN E%=3
 1320 IF INKEY-66 THEN E%=2
 1330 IF INKEY-83 THEN E%=1
 1340 UNTIL E%>0
 1350 IF E%=3 THEN 1440
 1360 PROCdnld
 1370 FOR y%=1 TO 8:FOR x%=1 TO 8
 1380 ON E% GOSUB 1480,1460
 1390 G%=-SGN?(N%+(ws1%+8*F%))
 1400 gr%(x%,y%)=G%
 1410 PROCplot(NOTG%+1)
 1420 PROCcur(FALSE,FALSE)
 1430 NEXT x%,y%
 1440 PROCreset
 1450 RETURN
 1460 N%=8-y%:F%=x%-1
 1470 RETURN
 1480 N%=y%-1:F%=8-x%
 1490 RETURN
 1500 :
 1510 *|" *** MIRROR *** "
 1520 :
 1530 COLOUR128:COLOUR1
 1540 PRINT TAB(12,5)"'V' Vert."TAB(12,6
)"'H' Hori."TAB(12,7)"Esc Quits"
 1550 SOUND3,-14,200,4
 1560 E%=0:xx%=x%:yy%=y%
 1570 REPEAT
 1580 IF INKEY-113 THEN E%=3
 1590 IF INKEY-85 THEN E%=2
 1600 IF INKEY-100 THEN E%=1
 1610 UNTIL E%>0
 1620 IF E%=3 THEN 1700
 1630 PROCdnld
 1640 FOR y%=1 TO 8:FOR x%=1 TO 8
 1650 ON E% GOSUB 1720,1740
 1660 gr%(x%,y%)=G%
 1670 PROCplot(NOTG%+1)
 1680 PROCcur(FALSE,FALSE)
 1690 NEXT x%,y%
 1700 PROCreset
 1710 RETURN
 1720 G%=-SGN?(x%-1+(ws1%+8*(8-y%)))
 1730 RETURN
 1740 G%=-SGN?(8-x%+(ws1%+8*(y%-1)))
 1750 RETURN
 1760 :
 1770 *|" *** CLEAR *** "
 1780 :
 1790 C%=FNget(TRUE)
 1800 IF C%=TRUE THEN 1900
 1810 IF FNsure THEN 1900
 1820 IF C%=1 THEN 1860
 1830 M%=FNaddr2(C%):!M%=0:M%!4=0
 1840 VDU31,x%,y%,32
 1850 GOTO 1900
 1860 FOR y%=1 TO 8:FOR x%=1 TO 8
 1870 gr%(x%,y%)=0
 1880 PROCplot(0):PROCcur(FALSE,FALSE)
 1890 NEXT x%,y%
 1900 PROCreset
 1910 RETURN
 1920 :
 1930 *|" *** PRINT *** "
 1940 :
 1950 PRINT TAB(12,5)"'W' Whole"TAB(12,6
)"'C' Chars"TAB(12,7)"'V' Value"TAB(12,8
)"Esc Quits"
 1960 SOUND3,-14,200,4
 1970 xx%=x%:yy%=y%:E%=0
 1980 REPEAT
 1990 IF INKEY-113 THEN E%=4
 2000 IF INKEY-83 THEN E%=3
 2010 IF INKEY-100 THEN E%=2
 2020 IF INKEY-34 THEN E%=1
 2030 UNTIL E%>0
 2040 IF E%=4 THEN 2060
 2050 ON E% GOSUB 2080,2160,2440
 2060 PROCreset
 2070 RETURN
 2080 PROCcl_win:PRINTTAB(13,6)"DUMPING"
 2090 VDU1,13
 2100 FOR L%=32 TO129
 2110 PROCpt_ch
 2120 IF (L%-31)MOD7=0 THEN VDU2,1,10,3
 2130 NEXT
 2140 VDU2,1,13,1,10,3
 2150 RETURN
 2160 PROCcl_win
 2170 PRINT TAB(12,5)"'P' Print"TAB(12,6
)"'S'  Scr"TAB(12,7)"Esc Quits"
 2180 SOUND3,-14,200,4
 2190 Z%=FALSE:E%=0
```

```
 2200 REPEAT
 2210 IF INKEY-113 THEN E%=3
 2220 IF INKEY-56 THEN E%=2
 2230 IF INKEY-82 THEN E%=1
 2240 UNTIL E%>0
 2250 IF E%=3 THEN 2150
 2260 PROCcl_win:C%=FNget(TRUE)
 2270 PRINTw0$;
 2280 IF C%=1 THEN 2510
 2290 M%=FNaddr2(C%)
 2300 IF E%=2 THEN VDU2,1,10
 2310 PRINT" Binary   Hex Dec"
 2320 FOR F%=M% TO M%+7
 2330 V%=?F%:B%=128
 2340 FOR N%=1 TO8
 2350 IF V% DIVB% V%=V%-B%:VDU49 ELSE VD
U48
 2360 B%=B%/2
 2370 NEXT
 2380 V%=?F%:PRINT"  &";
 2390 IF V%<16 THEN VDU48
 2400 PRINT ~V%;" "FNdec(3,V%)
 2410 NEXT
 2420 IF E%=2 THEN VDU1,60,3 ELSE VDU13
 2430 PROCspc:GOTO 2150
 2440 PRINTw1$;"Type Sentence"'"(1 - 250
)"
 2450 L$=FNinput(TRUE):VDU2,13,13,3
 2460 FOR F%=1 TO LEN(L$)
 2470 L%=ASC(MID$(L$,F%,1)):PROCpt_ch
 2480 NEXT
 2490 VDU2,1,13,1,10,3
 2500 RETURN
 2510 PROCval
 2520 FOR F%=0 TO7:F%?ws2%=val%(F%):NEXT
 2530 M%=ws2%:GOTO 2300
 2540 :
 2550 *|" *** ERASE *** "
 2560 :
 2570 PRINT TAB(12,5)"'I' Invrt"TAB(12,6
)"'E' Erase"TAB(12,7)"Esc Quits"
 2580 SOUND3,-14,200,4
 2590 Z%=FALSE:E%=0
 2600 REPEAT
 2610 IF INKEY-113 THEN E%=3
 2620 IF INKEY-38 THEN E%=2
 2630 IF INKEY-35 THEN E%=1
 2640 UNTIL E%>0
 2650 IF E%=3 THEN 2730
 2660 IF FNsure THEN 2730
 2670 T%=0
 2680 FOR F%=font% TO font%+&2FC STEP 4
 2690 IF E%=2 THEN T%=!F% EOR-1
 2700 !F%=T%
 2710 NEXT
 2720 PROCfont
 2730 PROCr_ins
 2740 RETURN
 2750 :
```

```
 2760 *|" *** SELECT DISC *** "
 2770 :
 2780 op%=TRUE:S$=disc$:PROCos(S$)
 2790 IF disc$="DISC" PROCos("DIR "+dir$
) ELSE PROCos("DIR :"+STR$(dr%)+"."+dir$
)
 2800 PRINT TAB(14,6)S$TAB(12,8)"SELECTE
D"
 2810 SOUND3,-14,150,2
 2820 *FX21
 2830 I%=INKEY(200)
 2840 PROCr_ins
 2850 SOUND3,-14,150,2
 2860 RETURN
 2870 :
 2880 *|" *** SELECT TAPE *** "
 2890 :
 2900 op%=FALSE:S$="TAPE":PROCos(S$)
 2910 GOTO 2800
 2920 :
 2930 *|" *** CAT *** "
 2940 :
 2950 IF NOT(op%) THEN 2980
 2960 PROCdisc:PROCspc:PROCr_ins
 2970 RETURN
 2980 PRINTw1$;"'Esc' Quits"
 2990 PROCescT:*CAT
 3000 :
 3010 *|" *** LOAD *** "
 3020 :
 3030 IF op% THEN PROCdisc
 3040 F$=FNinput(FALSE)
 3050 IF FNsure THEN 3100
 3060 IF NOT(op%) THEN PROCc_esc
 3070 PROCos("LOAD "+F$+" "+STR$~font%)
 3080 PROCmove(font%,copy%):PROCfont
 3090 IF op% THEN PROCescF
 3100 PROCr_ins
 3110 RETURN
 3120 :
 3130 *|" *** SAVE *** "
 3140 :
 3150 IF op% THEN PROCdisc
 3160 F$=FNinput(FALSE)
 3170 IF FNsure THEN 3210
 3180 IF NOT(op%) THEN PROCc_esc
 3190 PROCos("SAVE "+F$+" "+STR$~font%+"
 +300 0 0")
 3200 IF op% THEN PROCescF
 3210 PROCr_ins
 3220 RETURN
 3230 :
 3240 *|" *** STORE FONT *** "
 3250 :
 3260 IF FNsure THEN 3280
 3270 PROCmove(font%,copy%):PROCfont
 3280 PROCr_ins
 3290 RETURN
 3300 :
```

```
 3310 *|" *** STAR COMMAND *** "
 3320 :
 3330 PRINT TAB(13,6)"Type in"TAB(12,7)"
*Command"w1$;CHR$42;
 3340 PROCos(FNinput(TRUE)):PROCspc
 3350 PROCr_ins
 3360 RETURN
 3370 :
 3380 *|" *** RESTORE *** "
 3390 :
 3400 PRINT TAB(12,5)"'L' Last"TAB(16,6)
"font"TAB(12,7)"'D' Deflt"TAB(16,8)"font
"
 3410 E%=0
 3420 REPEAT
 3430 IF INKEY-113 THEN E%=3
 3440 IF INKEY-87 THEN E%=2
 3450 IF INKEY-51 THEN E%=1
 3460 UNTIL E%>0
 3470 IF E%=3 THEN 3490
 3480 ON E% GOSUB 3510,3600
 3490 PROCr_ins
 3500 RETURN
 3510 IF FNsure THEN 3590
 3520 A%=10:X%=pa% MOD &100
 3530 Y%=pa% DIV &100:M%=font%
 3540 FOR F%=32 TO 127
 3550 ?pa%=F%:CALL!&20C
 3560 !M%=pa%!1:M%!4=pa%!5:M%=M%+8
 3570 NEXT
 3580 PROCfont
 3590 RETURN
 3600 IF FNsure THEN 3580
 3610 PROCmove(copy%,font%)
 3620 GOTO 3580
 3630 :
 3640 *|" *** FUNCTIONS *** "
 3650 :
 3660 DEF FNinput(p%)
 3670 F$="":PROCescT
 3680 *FX202,32
 3690 IF p% THEN p0%=1:p1%=250:GOTO 3730
 3700 VDU28,12,8,20,5,12
 3710 IF op% THEN p0%=1:p1%=7 ELSE p0%=1
:p1%=10
 3720 PRINT"File Name ("p0%" - "p1%")"
 3730 *FX21
 3740 X%=POS:Y%=VPOS:*FX154,136
 3750 REPEAT
 3760 REPEAT G%=GET:UNTIL (G%>31 ANDG%<1
28) OR(INKEY(-74) AND LEN(F$)>=p0%)
 3770 IF G%=13 THEN 3810
 3780 IF G%=127 THEN F$=LEFT$(F$,LENF$-1
):GOTO 3800
 3790 F$=F$+CHR$G%
 3800 PRINT TAB(X%,Y%)F$;CHR$32;CHR$8;
 3810 UNTIL LEN(F$)=p1% OR G%=13
 3820 *FX154,8
 3830 VDU10,13:PROCescF
 3840 =F$
 3850 :
 3950 DEF FNsure
 3960 R%=1
 3970 PRINT CHR$26TAB(12,5)" ARE YOU "TA
B(12,6)"  SURE?  "TAB(12,7)"  (Y/N)  "TA
B(12,8)SPC9;
 3980 REPEAT
 3990 G%=GET AND &DF
 4000 IF G%=89 R%=0 ELSE IF G%=78 OR INK
EY-113 R%=-1
 4010 UNTIL R%<>1
 4020 =R%
 4030 :
 4460 DEF PROCc_esc
 4470 PROCescT:PRINTw1$;
 4480 ENDPROC
 4490 :
 4500 DEF PROCescT
 4510 esc%=TRUE:*FX220,27
 4520 ENDPROC
 4530 :
 4580 DEF PROCdisc
 4590 PRINT TAB(13,6)"DRIVE ?"TAB(12,8)"
(0,1,2,3)"
 4600 REPEAT G%=GET-48:UNTIL G%>-1 AND G
%<4
 4610 PRINTw2$;:dr%=G%:IF disc$="DISC" P
ROCos("DR. "+STR$(dr%)) ELSE PROCos("DIR
:"+STR$(dr%)+"."+dir$)
 4620 pa%!1=ws1%:pa%!5=31:pa%!9=0
 4630 A%=8:X%=pa% MOD &100:Y%=pa% DIV &1
00:CALL &FFD1
 4640 fi%=31-pa%?5:PRINT fi%" files"'
 4650 IF fi%=0 THEN PRINT"No fonts on di
sc!"':GOTO 4770
 4660 N1%=?ws1%
 4670 FOR F%=0 TO fi%-1
 4680 FOR N%=1 TO N1%
 4690 V%=?(ws1%+F%*(N1%+1)+N%)
 4700 IF V%<32 THEN V%=63
 4710 IF V%>127 THEN V%=V%-128
 4720 VDU V%
 4730 NEXT
 4740 IF F%/2=INT(F%/2) PRINT SPC(-7*(N1
%=7)-(N1%=10));
 4750 NEXT
 4760 VDU10,10,13
 4770 ENDPROC
 4780 :
 4790 DEF PROCpt_ch(P%)
 4800 A%=P%:CALLprn%
 4810 ENDPROC
 4820 :
 4830 DEF PROCdnld
 4840 FOR F%=0 TO 7:FOR N%=0 TO 7
 4850 N%?(ws1%+8*F%)=gr%(N%+1,F%+1)
 4860 NEXT N%,F%
 4870 ENDPROC
 4880 :
```

Twiddle

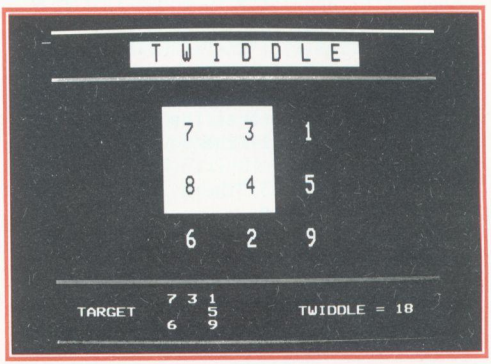
Martin Sann describes a computer implementation of a simple yet frustrating and addictive game.

This game is a close relative of Sam Lloyd's sliding block puzzle and Rubik's cube. You have to move numbers around on a 3 by 3 grid to match a target. Four numbers are highlighted at any one time. You can rotate these (in a clockwise direction) with the Return key, or swap the top pair with the bottom pair using the S key. Move to a new set of four numbers by using the curso eys.

When you firs art you'll find it best to choose a simple tar and the easy option. After you've found an efficient way of matching the target try a random target with the easy option.

Your status depends only on the number of twiddles you make and whether you've chosen the easy or hard option. So if you want to reach the elevated status of TWIDDLER then be prepared for a lot of frustrating twiddling using the hard option.

If you find any game too frustrating, pressing Escape will allow you to abort the current game and start a new one, while Shift-Escape will exit from the program altogether.



MAJOR PROCEDURES

info	explain game and control keys
choose	let player choose option
numbers	print 3x3 grid of random numbers
target	print the target numbers
twiddle	main procedure
square	move to new block of four numbers
keys	gets player's key presses
spin	rotate the four numbers in block
swap	swap top and bottom pair in block
new	new string for numbers procedure
rnd	randomise the numbers 1 to 9
status	allocate the status of the player

MAJOR VARIABLES

comp%()	holds the nine random numbers
P$	current string for player
T$	target string for player
X%,Y%	determines position on screen
go	number of twiddles made
st$	status of player

```
 10 REM Program TWIDDLE
 20 REM Version B1.0
 30 REM Author  Martin Sann
 40 REM BEEBUG  October 1989
 50 REM Program subject to copyright
 60 :
100 MODE7:VDU23,1,0;0;0;0;
110 ON ERROR GOTO340
120 DIM d%(10),comp%(10)
130 *FX4,1
140 *FX11,0
150 j$=CHR$150+STRING$(39,CHR$112)
160 k$=CHR$150+STRING$(39,CHR$96)
170 bar$=CHR$157+CHR$132
180 nobar$=CHR$135+CHR$156
190 PROCinfo:exit%=FALSE
200 REPEAT
210 PROCchoose:PROCscreen
220 P%=11:Q%=7:go=0
230 PROCsquare(P%,Q%)
240 REPEAT
250 PROCtwiddle(X%,Y%)
260 UNTIL P$=T$
270 PROCstatus
280 UNTIL exit%
290 CLS:PRINT"Goodbye for now."
300 *FX4
310 *FX11,32
320 END
330 :
340 IF ERR=17 AND NOT INKEY-1 GOTO 200
350 MODE 7:REPORT:PRINT" at line ";ERL
360 *FX4
370 *FX11,32
380 END
```

```
 390 :
 1000 DEFPROCinfo
 1010 PROCtitle
 1020 PRINTTAB(3,5)"Match the numbers in
 the large grid"
 1030 PRINTTAB(3,6)"with those in the sm
all target grid."
 1040 PRINTTAB(2,8)"The"+CHR$131+"Cursor
"+CHR$135+"keys move the white square."
 1050 PRINTTAB(2,10)"The"+CHR$131+"Retur
n"+CHR$135+"key rotates the numbers in"
 1060 PRINTTAB(2,11)"the square in a clo
ckwise direction:"
 1070 PRINTTAB(2,13)"There is also an ea
sy option but with"
 1080 PRINTTAB(2,14)"it you cannot achie
ve TWIDDLER status."
 1090 PRINTTAB(2,16)"If you choose the e
asy option the"+CHR$131+"S"+CHR$135
 1100 PRINTTAB(2,17)"key swaps the top a
nd bottom numbers."
 1110 PRINTTAB(0,19)k$
 1120 PROCbig(CHR$131+CHR$157+CHR$132+"R
eturn to continue  "+CHR$156,8,20)
 1130 PRINTTAB(0,22)j$;
 1140 REPEAT:G=GET:UNTIL G=13
 1150 ENDPROC
 1160 :
 1170 DEF PROCtitle
 1180 CLS:PRINTTAB(0,0)k$
 1190 PROCbig(CHR$131+CHR$157+CHR$132+"T
  W  I  D  D  L  E  "+CHR$156,7,1)
 1200 PRINTTAB(0,3)j$
 1210 ENDPROC
 1220 :
 1230 DEF PROCchoose
 1240 PROCtitle:PRINTTAB(0,22)j$
 1250 PROCbig("key 1 for a simple target
",7,6)
 1260 PROCbig("key 2 for a random target
",7,9)
 1270 REPEAT:G=GET-48:UNTIL G=1 OR G=2:I
F G=1 random=0 ELSE random=1
 1280 PROCbig("press E for the easy opti
on",6,14)
 1290 PROCbig("press H for a really hard
 time",5,18)
 1300 REPEAT:G$=CHR$(GET AND &DF):UNTIL
G$="E" OR G$="H"
 1310 IF G$="E" easy=1 ELSE easy=0
 1320 ENDPROC
 1330 :
 1340 DEF PROCbig(A$,X%,Y%)
 1350 FOR row=0 TO 1
 1360 PRINTTAB(X%,Y%+row)CHR$141+A$
 1370 NEXT row
 1380 ENDPROC
 1390 :
 1400 DEF PROCscreen
 1410 PROCtitle
 1420 FOR row=1 TO 3
 1430 PRINTTAB(0,19+row)CHR$129+CHR$157
 1440 NEXT row
 1450 PRINTTAB(2,21)CHR$135+"TARGET"
 1460 PRINTTAB(0,19)k$
 1470 PRINTTAB(24,21)CHR$135+"TWIDDLE =
0"
 1480 PRINTTAB(0,23)j$;
 1490 PROCnumbers:PROCtarget
 1500 ENDPROC
 1510 :
 1520 DEF PROCnumbers
 1530 PROCrnd(9):T$=""
 1540 FOR N=1 TO 9
 1550 IF random=0 T$="123456789" ELSE T$
=T$+STR$(comp%(N))
 1560 NEXT N
 1570 PROCrnd(9):R$=""
 1580 FOR N=1 TO 9
 1590 R$=R$+STR$(comp%(N))
 1600 NEXT N
 1610 P$=R$:PROCprintnum(2)
 1620 ENDPROC
 1630 :
 1640 DEF PROCprintnum(S%)
 1650 FOR ypos=0TO2:FOR xpos=0TO2
 1660 P=xpos+ypos*3+1
 1670 IF MID$(P$,P,1)=MID$(T$,P,1) ok$=C
HR$135 ELSE ok$=CHR$130
 1680 IF S%=1 PRINTTAB(11+xpos*2,20+ypos
)ok$+MID$(T$,P,1)
 1690 IF S%=2 PROCbig(MID$(R$,P,1),13+xp
os*6,7+ypos*4)
 1700 NEXT xpos,ypos
 1710 ENDPROC
 1720 :
 1730 DEF PROCtarget
 1740 PROCprintnum(1)
 1750 ENDPROC
 1760 :
 1770 DEF PROCtwiddle(X%,Y%)
 1780 IF P$=T$ ENDPROC ELSE PROCkeys
 1790 IF K=13 PROCspin:PROCtarget:GOTO017
80
 1800 IF K=83 OR K=115 PROCswap:PROCtarg
et:GOTO1780
 1810 PROCclear(X1%):PROCsquare(X%,Y%)
 1820 ENDPROC
 1830 :
 1840 DEF PROCkeys
 1850 REPEAT:K=GET
 1860 UNTIL (K>135 AND K<140) OR K=13 OR
 ((K=83 OR K=115) AND easy=1)
 1870 IF K=136 P%=11
```

```
1880 IF K=137 P%=17
1890 IF K=138 Q%=11
1900 IF K=139 Q%=7
1910 X%=P%:Y%=Q%
1920 ENDPROC
1930 :
1940 DEF PROCsquare(X%,Y%)
1950 X1%=P%
1960 PRINTTAB(X%+11,Y%-1)nobar$
1970 PRINTTAB(X%+1,Y%-1)bar$
1980 FOR y=0 TO 2
1990 PROCbig(nobar$,X%+10,Y%+y*2)
2000 PROCbig(bar$,X%,Y%+y*2)
2010 NEXT y
2020 PRINTTAB(X%+11,Y%+6)nobar$
2030 PRINTTAB(X%+1,Y%+6)bar$
2040 ENDPROC
2050 :
2060 DEF PROCclear(X%)
2070 FOR y=5 TO 17
2080 PROCbig("  ",X%,y)
2090 PROCbig(" ",X%+11,y)
2100 NEXT y
2110 ENDPROC
2120 :
2130 DEF PROCspin
2140 IF X%=11 AND Y%=7 A=4:B=1:C=2:D=5
2150 IF X%=17 AND Y%=7 A=5:B=2:C=3:D=6
2160 IF X%=11 AND Y%=11 A=7:B=4:C=5:D=8
2170 IF X%=17 AND Y%=11 A=8:B=5:C=6:D=9
2180 PROCturn(A,B,C,D)
2190 ENDPROC
2200 :
2210 DEF PROCswap
2220 IF X%=11 AND Y%=7 A=4:B=5:C=2:D=1
2230 IF X%=17 AND Y%=7 A=5:B=6:C=3:D=2
2240 IF X%=11 AND Y%=11 A=7:B=8:C=5:D=4
2250 IF X%=17 AND Y%=11 A=8:B=9:C=6:D=5
2260 PROCturn(A,B,C,D)
2270 ENDPROC
2280 :
2290 DEF PROCturn(A,B,C,D)
2300 a$=MID$(P$,A,1):b$=MID$(P$,B,1)
2310 c$=MID$(P$,C,1):d$=MID$(P$,D,1)
2320 PROCprint(X%,Y%)
2330 PROCgo:PROCnew
2340 ENDPROC
2350 :
2360 DEF PROCprint(X%,Y%)
2370 PROCbig(bar$+a$,X%,Y%)
2380 PROCbig(b$,X%+8,Y%)
2390 PROCbig(c$,X%+8,Y%+4)
2400 PROCbig(bar$+d$,X%,Y%+4)
2410 ENDPROC
2420 :
2430 DEF PROCgo
2440 go=go+1:go$=STR$(go)
```

```
2450 PRINTTAB(24,21)CHR$135+"TWIDDLE =
"+go$
2460 ENDPROC
2470 :
2480 DEF PROCnew
2490 IF X%=11 AND Y%=7 P$=a$+b$+MID$(P$
,3,1)+d$+c$+MID$(P$,6,4)
2500 IF X%=17 AND Y%=7 P$=MID$(P$,1,1)+
a$+b$+MID$(P$,4,1)+d$+c$+MID$(P$,7,3)
2510 IF X%=11 AND Y%=11 P$=MID$(P$,1,3)
+a$+b$+MID$(P$,6,1)+d$+c$+MID$(P$,9,1)
2520 IF X%=17 AND Y%=11 P$=MID$(P$,1,4)
+a$+b$+MID$(P$,7,1)+d$+c$
2530 ENDPROC
2540 :
2550 DEF PROCrnd(total)
2560 FOR N = 1 TO total+1
2570 d%(N)=N
2580 NEXT
2590 REPEAT
2600 rnum=RND(total)
2610 comp%(total)=d%(rnum)
2620 FOR N=rnum TO total-1
2630 d%(N)=d%(N+1)
2640 NEXT
2650 total=total-1
2660 UNTIL total=1
2670 comp%(1)=d%(1)
2680 ENDPROC
2690 :
2700 DEF PROCstatus
2710 PROCtarget
2720 PROCbig(CHR$136+"G O T  I T  A T
L A S T !",6,1)
2730 PROCclear(X1%)
2740 TIME=0:REPEAT UNTIL TIME>275
2750 FOR row=1 TO 3
2760 PRINTTAB(0,row)CHR$131+CHR$157+STR
ING$(38," ")
2770 NEXT row
2780 PRINTTAB(0,4)j$
2790 IF easy=1 AND go<101 st$="NOVICE"
2800 IF easy=1 AND go>100 AND go<201 st
$="AMATEUR"
2810 IF easy=1 AND go>200 st$="ABYSMAL"
2820 IF easy=0 AND go<201 st$="TWIDDLER
"
2830 IF easy=0 AND go>200 AND go<501 st
$="BRAVE"
2840 IF easy=0 AND go>500 st$="NOVICE"
2850 PROCbig(CHR$132+"STATUS = "+st$,11
,1)
2860 PRINTTAB(11,3)CHR$132+"ANOTHER GO?
 (Y/N)"
2870 REPEAT:G$=GET$:UNTIL G$="Y" OR G$=
"N"
2880 IF G$="N" exit%=TRUE
```

A General Purpose Line-Input Function (2)

Gareth Williams explains how to use the comprehensive line input routine listed last month.

Last month's issue of BEEBUG contained the listing of the line input function together with a demonstration program. However, to use the input function in your own programs you need to understand the format used to specify the details of data entry, and this is what is explained here.

Remember that you will need to run the program from the last issue to generate the machine code routines. These are save as the file *Edline*. Any program which intends to use these routines must then include a line such as:

```
*LOAD Edline
```

The program assumes a mode 3 screen, but if this is not the case, change the value assigned to X% in line 1060 accordingly.

The code is also assembled to be loaded at a position &500 bytes below HIMEM. Thus any calling program should include a line such as line 100 in last month's demo program to reserve the space for the code. In fact, we shall be making reference to this demo program quite frequently when discussing the details of data entry formatting, so it may be as well to have the listing handy. All line number references in what follows refer to this program.

You may also find it useful to include in your own program the function FNEditLine from the demo program as this makes the specification of data entry formats described below much easier to handle. This consisted of lines 1060 to 1260 of the demo program.

USING EDITLINE

The principle of operation is quite simple - for each piece of data to be input, the user's program sets up details of the input prompt, the maximum length for the data, any default input, and a buffer to receive the input, and then calls the machine code routine Edline. This handles as much of the input as is possible, and returns the final string entered to the user. It also copes with special cases, such as when an unrecognised key is pressed, or the insert/overwrite mode is changed.

Editline uses a 19-byte parameter block to control its operation. This parameter block is addressed by pointing the 6502 X (lo) and Y (hi) registers to the parameter block before calling the routine - in exactly the same way as the OSWORD calls operate. From Basic, this is done by setting up the parameter block using the indirection operators ?, ! and $, pointing X% and Y% to the block, and calling the routine with CALL (see FNEditLine). Since information is returned by Editline the parameter block should be in RAM. The layout of this parameter block is detailed in table 1.

Offset XY+	Entry conditions	Exit conditions
0	See below	Unchanged
1	Flag-byte 1	Unchanged
2	Flag-byte 2	Unchanged
3	Flag-byte 3	See below
4	X co-ord for prompt	Unchanged
5	Y co-ord for prompt	Unchanged
6	X co-ord for input	Unchanged
7	Y co-ord for input	Unchanged
8	Addr of prompt message	Unchanged
10	Addr of input buffer	Unchanged
12	Addr of help message	Unchanged
14	Maximum length of input	Input length
15	Special-key value	Unchanged
16	Help-key value	Unchanged
17	Ignored	ASCII value of unrecognised key
18	Initial position of cursor within line	Current position of cursor within line

Table 1. Format of parameter block

The first byte of the parameter block, offset 0, is reserved for use by the user's own program. This could be used by the application for storing additional flags for the program itself. However, for simple use this byte can be ignored.

The four bytes from XY+4 to XY+7 specify the screen co-ordinates of the prompt, and the

input display (see lines 1100 to 1130). If the X co-ordinate for either is &FF then the cursor will not be moved from its current position. This means that the position for the input can be made to follow on immediately from the prompt message.

The prompt message, input buffer and help message are all terminated by Carriage Return, making it easy to set them up with the $ indirection operator (see lines 1140 to 1160). The first two bytes of the help message should be the X and Y co-ordinates of where the help message is to appear.

Flag-byte 1	Bit	Function when set
	7	Prevent Escape aborting input
	6	Allow Special key
	5	Allow Help key
	4	Return unrecognised keys
	3	Suppress spaces
	2	Restrict input to Special key
	1	Restrict input to Y or N
	0	Single-key entry
Flag-byte 2	**Bit**	**Function when set**
	7	Return after displaying initial string
	6	Display input length
	5	Hide input
	4	Special numeric input
	3	Restrict to numeric input
	2	Flush input stream on entry
	1	Clear input buffer on entry
	0	Capify alphabetic input
Flag-byte 3	**Bit**	**Function when set**
	7	Escape key ended input
	6	Up-Cursor ended input
	5	Down-Cursor/Return ended input
	4	Special key pressed
	3	Unrecognised key returned
	2	Help key pressed
	1	Editing-mode change
	0	Editing-mode status

Table 2. Format of the flag bytes

Flag bytes 1 and 2 are used to pass information to Editline. Flag byte 3 contains the editing mode bit, and is also used to return various flags. The format of these bytes is shown in table 2. The remaining bytes specify the maximum line length and initial cursor position, and the ASCII codes of the 'help' and 'special' keys. The help key is used to pop up a help message (Ctrl-H, ASCII code 8 - see line 1190) in the demonstration, and the special key is a key which causes an immediate return from Editline when it is pressed. Copy is used in this role in the demonstration to exit from the program (see line 1180).

```
Surname      SMITH.....................     Current editing mode is Overwrite
Forename(s)  John Julian..............................
House name   Dunromin.................  Town        Littlehampton.............
House number 28..                      County      EAST SUSSEX...............
Street       Bognor Street.........    Postcode    CH1 2BX..
District     ......................                Type any digit 1
How many CSEs do you have?    3.     Do you like football? (Y/N)          Y
How many O-levels do you have? 5.    Are you a yuppie?.(Y/N)              N
How many A-levels do you have? 3.    Are you an estate agent? (Y/N)       N
Would you mind answering extremely personal questions? (Y/N) Y
Please enter the top secret password:****.............   Press COPY to finish: .
```

Completed data entry screen from last month's demo program

Flag bits are set to 1 (to select a function) or to 0 otherwise. The eight bits together are then specified as a single number (usually in hex). Thus a flag byte of 101101011 becomes &BB, and this is how flag bytes are specified in the demo program (see the DATA statements from line 1560 onwards). Each hex digit corresponds to four bits of a byte. This is the easiest way for quick conversion between binary and hex.

The functions of the bits in flag byte 1 should be fairly obvious. Setting bit 2 will allow only the special key to be entered. Setting bit 0 will cause a return as soon as one character is entered, and is automatically set for Y/N and special key entry. In this case Editline returns with bit 5 of flag byte 3 set (Return ended input).

An unrecognised key is one which does not perform a function known to Editline (e.g. Cursor-Left, Shift-Cursor-Left, Delete), or which has an ASCII value less than 32 or greater than 127. If bit 4 of flag-byte 1 is set, then the value of the unrecognised key is returned in XY+17 of the parameter block, and bit 3 of flag-byte 3 is set to indicate that an unrecognised key has been returned.

Turning to flag byte 2, bit 0 causes all alphabetic input to be converted to capitals - this is automatic if Special key or Y/N input is selected. Setting bit 1 causes any input already in the buffer to be deleted, and bit 2 will force the keyboard buffer to be flushed as well. If bit 1 is clear then any data already in the input buffer will be treated as the initial input.

Setting the *special numeric* input bit will implicitly set the *restrict to numeric input* bit. Numeric input only accepts the digits 0 to 9, plus spaces (if they are not suppressed). Special numeric input also accepts '-' (unary minus), '+' (unary plus) and '.' (decimal point). There can only be one decimal point, and any sign must be the first character in the line.

Bit 5 set will cause the input to be hidden, ideal for passwords, and bit 6 makes Editline indicate the maximum length by display a series of dots in the relevant screen position.

If bit 7 is set, then the prompt message and the initial input string are displayed, after which Editline returns immediately to the calling program. This facility is to enable all the prompts on the current screen to be displayed at the beginning of data entry, as in the demonstration program.

Flag byte 3 is set up by Editline to indicate why the routine has returned control to the calling program. One of bits 1 to 7 will be set accordingly. With bits 1 or 2, you should perform any actions you wish, and call Editline again with the parameter block unchanged. For bits 5 to 7, the input has been terminated, and the block should be reset before Editline is called again. The action taken if bits 3 or 4 are set depends on the program. The demonstration program exits if the special key is pressed, and ignores unrecognised keys totally. Bit 0 of flag byte 3 is the insert/overwrite flag (0 for insert), and can be set up when Editline is called to force a particular mode. When bit 1 is set on return to indicate that the editing mode has been changed, then bit 0 is toggled automatically by Editline.

The demo program stores the information for each entry in a DATA statement, the contents of which are read into arrays when the program starts (lines 1290 to 1350 of PROCdemo). The very first data item, the value 19 at line 1550, indicates the number of data entry fields being defined.

The order of items in each DATA statement is as follows:

Prompt text
X co-ordinate of prompt
Y co-ordinate of prompt
Default input (as text)
X co-ordinate for input field
Y co-ordinate for input field
Maximum length of input field
Help text
Flag 1 settings (given in hex)
Flag 2 settings (given in hex)

As the data is read for each input definition, *FNEditLine* is then called (with its final parameter *Disp%* set to TRUE) to display the prompts on the screen. This function sets up the parameter block and calls the machine code. The 'return immediately' bit is set so that the prompts are displayed without waiting for input (see lines 1330 to 1350).

FNEditLine is then called again for each field to accept the actual input, which is assigned to the relevant element of the array *Input$()*, although this is not used further in the demo. The reason for a return from the machine code Edline routine is provided in the variable *Return%*. Note how this is tested and action taken accordingly 1390 to 1440.

Finally, you will need to make sure at the outset that you assign enough memory for the parameter block, prompt, input buffer and help message. In the demo program this is dealt with in line 1280.

Controlling the input function may seem very complex, but studying the demonstration program in conjunction with the above notes should clarify the situation, and help you to use it in your own programs. You may find it helpful to use the demo as the basis of your own programs until you become more familiar with the operation of the general purpose line input function.

Our hints this month concentrate on various aspects of !BOOT files.

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

SELF BOOTING BASIC

Sebastian Lazareno

Any Basic program with its first line number greater than 0 can be made to boot itself. With the program in memory, type in the following:

```
0::::CLOSE#0:CHAIN"!BOOT"
!(PAGE+4)=&7F7FF4
```

Save the program to drive 0 as !BOOT and set the !BOOT option to EXEC with *OPT4,3. The program will now run itself when Shift-Break is pressed.

&7F is the ASCII for Delete, and &F4 is the Basic token for REM, so the start of the line will look funny when listed. As long as it starts with CLOSE#0, the line may contain any legal commands, so for example PAGE could be set to &1100 before !BOOT is chained, or a function key set up and called to load the program and run it from &E00. The line cannot simply be edited - it must be reconstructed as described above.

CONDITIONAL !BOOT FILES (MASTER ONLY)

A.J.Morison

Philip Hetherington's hint on !Boot files (BEEBUG Vol.7 No.10) is interesting but it is possible to improve on his ideas even further. With his listing everything is loaded and run on the first Shift-Break, and you also have to execute a manual Ctrl-Break. My version loads any ROM images first, and then does a programmed Ctrl-Break, with the second Shift-Break then chaining or running any other file without needing a condition attached to it.

This method has been used with numerous applications, and the one shown here is a !Boot file for View:

```
*BASIC
J%=?&2A5
IF J%=0 THEN *SRLOAD HYPERDR 8000 W Q
IF J%=0 THEN *FX151,78,127
IF J%=0 THEN CALL !-4
*EXEC !View
```

!View is a further Exec file which sets up everything needed for View. Incidentally, there is no need to even do a manual second Shift-Break. The second line of the !Boot file could be *CONFIGURE BOOT which reverses the configuration and performs a Shift-Break on Break while your machine is on, or until Ctrl-Break is pressed. In that case *CONFIGURE NOBOOT must be written into the beginning of the file to be loaded or run on the second Shift-Break. Only then will Ctrl-Break or switching off reset to a normal configuration.

IMPROVED COLOUR MIXING

Jonathan Hogg

There is a simpler way of mixing colours with the GXR ROM, or on a Master, than the hint in BEEBUG Vol.8 No.2. These support special *simple pattern* colour fills which work in any mode. The following routine will assign pattern *pat_no* (range 1-4) with a stippled pattern made up of colours *col1* and *col2* (range dependent on the number of colours in the selected mode):

```
 1000 DEF PROCmix(pat_no,col1,col2)
 1010 VDU11+pat_no,col1,col2,col2,col1,
col1,col2,col2,col1
 1020 ENDPROC
```

The command GCOL 16*(n+1),0 (or 128) can then be used to select the pattern as the current foreground (or background) colour, where n should be replaced with the pattern number.

BOOTING THE OTHER SIDE

J.M.Mellard

With the following !BOOT file it is possible to auto-boot the other side of a disc if Shift is held down until the !BOOT file is executed:

```
IF NOT INKEY-1 THEN <command><filename>
*DRIVE 2
<command><filename>
```

where *command* and *filename* are the command and optional filenames of your choice. If both command and filename are the same for both sides then a more succinct version would be:

```
IF INKEY-1 THEN *DRIVE 2
<command><filename>
```

```
  920 ENDPROC
  930 DEFFNint(Z,G)
  940 @%=&1010100:i$=STR$(Z/4.9)
  950 p=ASCi$-48:a$="&"+MID$("12255AAAA"
,p,1)
  960 Z=EVALa$*10^VALMID$(i$,4)/2:N=2
  970 IFZ*G<64:Z=Z+Z
  980 @%=&1000200:IFx=0AND(2+LENSTR$(Z+Z
)>Z*G/16):N=4
  990 =Z
 1000 PROCtop:GCOL0,h:PRINTSPC(11)$I%'"
EQUATION: ";:C=1100:F=0:*FX4
 1010 PROCinput(68):IFa$=""RUN
 1020 $I%=a$:PRINT:D=C:F%=0
 1030 :
 1040 K=TRUE:L=0:d=1
 1050 :
 1060 PROCconvert:PROCtype(0):R=F:S=W:T=
Z:IFF<0GOTOD
 1070 D=1070:FORK=K+d TOL STEPd
 1080 F=R:W=S:Z=T:IFd*K>d*L D=1100:ELSEP
ROCplot
 1090 NEXT
 1100 IFF>0ORERR=13THEN1000
 1110 COLOUR2:IFF<0PRINT" "M$;:ELSE@%=0:
VDU11:REPORT:PRINT" @ "ERL;
 1120 COLOUR3:COLOUR129:PRINT" TAP A KEY
 ";:*FX15
 1130 PROCon:G=GET:GOTO1000
 1140 DEFPROCtype(I)
 1150 Y=INSTR(a$,"=")
```

```
 1160 IFY:$K%=LEFT$(a$,Y-1):$J%=MID$(a$,
Y+1):ELSEIFINSTR("+-*/^",CHR$ASCa$):$J%=
"("+$J%+")"+a$:ELSE$J%=a$:?K%=89+41*(INS
TR(a$,"Y")>0)
 1170 IF?K%=45$K%="0"+$K%
 1180 IF?J%=45$J%="0"+$J%
 1190 W=9:Z=9:IFI:F=1:ENDPROC
 1200 S%=0:F=0
 1210 IF$K%="Y"AND(INSTR($J%,"Y")<1)F=1.
5:Z=1:S%=1:ENDPROC
 1220 IF$K%="X"AND(INSTR($J%,"X")<1)F=1:
W=1:S%=1:ENDPROC
 1230 :
 1290 M$="TOO HARD!":F=-1:ENDPROC
 1300 DEFFNy(X)
 1310 IFS%:=EVAL$J%
 1320 :
 1340 DEFFNx(Y)
 1350 IFS%:=EVAL$J%
 1360 :
 1400 DEFPROCplot
 1410 IFF=INT(F):s=f:t=g:u=k:U=H:ELSEs=g
:t=f:u=j:U=J
 1420 W=u-4
 1430 :
 1600 C=1600:w=1E9:IFW>U C=D:ENDPROC
 1610 FORW=W+4TOU STEP4
 1620 IFF=1Y=W:X=f*FNx(Y/g)+2:Z=X:ELSEX=
W:Y=g*FNy(X/f)+2:Z=Y
 1630 IFABS(Z-w)<&FF DRAWX,Y:ELSEMOVEX,Y
 1640 w=Z:NEXT:ENDPROC
```

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

ROLLER RALLY (BEEBUG VOL.7 NO.9)

The high score feature does not work as described. The following amendment will fix this, and set "John's" high score to a more reasonable 200.

```
 1070 PRINTTAB(5,15)"The shortest time
  is "STR$(Hisc%DIV100+1)" seconds"'TAB
(15)"by "Hisc$
 1520 Q%=INKEY(500):CLS
 1730 Hisc%=19900:Hisc$="JOHN"
```

Thanks to Ashley Ward for this information.

ACES HIGH CARD GAME (BEEBUG VOL.8 NO.4)

Although unlikely to occur, if any column of cards reaches the foot of the display then this is corrupted. This situation can be forced by selecting (D)eal each time. This problem can be corrected with the following amendments:

```
 1415 IF row%>18 row%=18
 1995 IF Y%>20 Y1%=20
 2010 PRINTTAB(X%,Y1%-2+Step%) SPC8
 2040 xcoord%=X%:IF Y%<22 row%=Y%-4
 2045 PROCdisplay
```

In addition, Ian McEwen has sent in the following addition to the program which puts a thin dotted line along the top edge of overlapped cards for greater clarity:

```
 1272 gx%=col*256-128
 1281 gy%=1024-(row%*32)
 1282 GCOL 0,0:IF row%>2 MOVE gx%,gy%:
PLOT21,gx%+224,gy%
```

POSTBAG

HELP NEEDED

I am 11 years old and I live in Coventry. Is there anyone who could help me with my BBC micro, or is there a club which caters for beginners?

Emma Pithers

If anyone can help Emma please write to her c/o BEEBUG, and we will pass your letters on.

ENHANCED BASIC LINE EDITOR

Paul Pibworth's Basic Line Editor (BEEBUG Vol.8 No.2) can be enhanced by using the Copy key to delete the character under the cursor (as in the Master's Edit program). Just add or change the following lines:

```
1890 BEQ goright:CMP#135:BEQ copy:CMP
#0:BMI cursor2
2012 .copy CPY len:BNE copy1
2014 JMP cursor2
2016 .copy1 JSR upcount:JSR cursorpla
ce:JMP del1
```

David Stevens

DUAL COLUMN LISTINGS IMPROVED

With reference to my Dual Column Listing utility (BEEBUG Vol.7 No.9), the following code added to the program immediately after the label *start* in line 380 allows the utility to be *RUN:

```
\Re-enter Basic
LDX #0:LDY #255:LDA #252
JSR &FFF4:STX &F4:STX &FE30
```

The code 'tells' the computer that it is back in Basic. My thanks to Dr.L.Calcraft for his help with this.

Paul Pibworth

NO MORE CROSS WORDS

Prompted by Mr.Ford's letter (BEEBUG Vol.8 No.1) I have trapped the error which causes the Crossword Editor (first published in BEEBUG Vol.7 No.4) to crash. The program can be kept running as follows. First change line 140 to read:

```
140 ON ERROR GOTO 4000
```

and then add the following lines:

```
1465 M%=1
2135 M%=2
4000 IF A=149 AND M%=1 PROCsolve
4010 IF A=149 AND M%=2 PROCend:END
4020 PRINT"Not found"
4030 VDU7:PRINT"Press any key":*FX15
4040 B=GET
4050 VDU15:COLOUR 129:CLS
4060 PROCspell
4070 ON M% GOTO 1465,2135
```

Jean Barnard

EVEN MORE NIMBLE

The use of William Tunstall-Pedoe's Nimble Typer (BEEBUG Vol.8 No.3) can be improved even further if the '@' character is permitted in the definition of abbreviations. The amendments are minor:

Listing 1, line 350: change 65 to 64
Listing 2, line 410: change 65 to 64

A word or phrase can then be abbreviated to initial letter(s) plus '@' (e.g. "BEEBUG Limited" to "B@"), and possible confusion between abbreviations and text is eliminated.

Tony Moore

Our thanks for the bountiful supply of helpful information above from several BEEBUG readers.

MULTI-COLOUR PRINTING

May I thank Dorian Goring for his introduction to the techniques of multi-colour printing in BEEBUG Vol.8 No.4. As he says these techniques can be most rewarding. I would, however, like to express my concern that his article takes a somewhat simplistic approach.

I wish to assure those interested in colour printing that *Peacock Printer* from Datassistance (advertised in the same issue) provides an excellent implementation of these techniques in that it not only performs the splitting required for colour separation, but also performs the actual printing (without the need to store intermediate screen files), automatically re-aligning the paper ready for the next colour, a procedure which can be very fiddly to do manually. *Peacock Printer* can also be invoked directly from graphics applications without the need to first store the screen, and it works in all graphics modes.

I would not claim that Dorian's results are in any way inferior to those achieved with *Peacock Printer*, simply that the latter gives a simple-to-use integrated approach to achieving them.

Dave Moll Datassistance

If Dorian Goring's article has sparked off a new area of interest for BBC micro users then this is all to the good. We are also pleased to be able to draw readers attention to an interesting new product.

Personal Ads

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

We also accept members' Business Ads at the rate of 30p per word (inclusive of VAT) and these will be featured separately. Please send us all ads (personal and business) to MEMBERS' ADS, BEEBUG, 117 Hatfield Road, St. Albans, Herts AL1 4JS. The normal copy date for receipt of all ads will be the 15th of each month.

Delta/Card Index ROM £30, Delta Reference Guide £8, Delta Gamma ROM £20, Delta Mailshot £8, Delta Reporter £8, Delta Inter Link £8, (all with manuals) Edword Superpack £20, 'Double View' BBC/B 2x ROM £18, Mini Office II 80T + Dabs mini driver + Dabs Guide £15, Multi Font NLQ £8. All prices include P&P. Tel. 01-399 2865.

Games software. £100 worth for just £20. Titles include; Elite, Exile, Superior Coll. Vol. 1 all in good condition. Tel. 01-698 0215.

Master 128K Turbo, Microvitec med. res. monitor, Cumana DS/DD 40/80T Dual Drive, Viglen 20 M6 Winchester. Inter ROM set (Base, Sheet, Word, Spellmaster), AMX mouse, Superart & Pagemaker, ADT Master ROM, Viewspell ROM, Genie Junior, Games (Elite, Repton etc), manuals, books, discs. Tel. (0732) 862404 for more information.

Kaga Taxan KP810 printer (8K RAM fitted), 2 discs of down loadable fonts, Wordwise plus, WYSIWYG+, Toolkit Plus, Spellcheck III, Iconmaster, PMS NTQ with two extension discs, Replay for 8271, £130. Morley EPROM programmer, eraser, HCR EPROM programmer (requires repair) 23 various EPROMs 8 and 16k, £12. Pace Nightingale modem with autodial/answer board, OBBS software, Commstar 2, £50. Tel. (0226) 285172.

Z88 computer, 128k RAM, 32k epROM, Spellmaster, Parallel printer cable, Z88-BBC link, mains unit, carrying case, Topper keyboard cover, complete set Z88 EPROM magazines £300 may split. BEEBUG Toolkit ROM £7, Wordwise £7, TFS ROM £3, ATS ROM £5, all manuals included. Tel. (0377) 42037 eves/weekends or (0262) 677555 day.

Master 128m Philips 12" hi/res mono monitor, Cumana DD 5.25" and 3.5", dust covers £325. Master Compact, integral drive, Philips 14" med. res. colour monitor , mouse, dust covers £295. Panasonic KX-P1080 printer, dust cover £40, BBC Buggy (Master 128 version), stablised PSU, pen kit, large park £80, Lego interface, PSU, Lego Logo Pkg, Lego Kit 1090 £120, Interword, Dumpout 3, Logotron Logo ROMs £15, all complete with manuals, discs, leads where appropriate, some educ. software and books. Tel. 01 349 4302.

Business Ad

WHERE IS FATHER CHRISTMAS

Christmas Activity Pack for children age 7 to 9.
Includes adventure game for BBC computers (40 track disc), play script & activity sheets.
Only £8.
Send cheque to:
G. Dean,
3 Sunnyhurst Cottages,
Darwen, Lancashire. BB3 1JX

BBC M128, Sanyo monitor, Epson 40T DS DD, joysticks, Care quad cartridge, Data Recorder. ROMs; View, Viewsheet, Viewstore, Wordwise plus, Word Aid, G.Dump disc box with lots of utilities and games inc. TMS Music System. Lots of tapes of games, blank discs. Manuals, books, complete set of BEEBUG mags. All in excellent condition. £550 o.n.o. Tel. (0455) 284201 eves and weekends.

BBC Master series micro double disc drive/2 ROMs Microvitec colour CUB653 screen, Panasonic KX-P1080 printer, Magic Modem, WS4000 modem, assorted software and manuals £700. Tel. 01-994 7556.

WANTED: for compact, DFS ROM or EPROM. Tel. (0343) 545716.

BBC B iss 7, £250. Tel. (0562) 820910 day or (0562) 885983 eves.

Watford Shadow RAM 32k complete with manual £40. Tel. (0670) 860170.

Master 128 DSDD 40/80 DD, video digitiser, BEEBUG C, AMX Super Art and Stop Press, other ROMs, reference manuals, books and software. Worth £1100. £500 o.n.o. Tel. 091-387 1039.

Pace Nightingale modem with Commstar II ROM £55, ATPL ROM board incl 16k RAM £15, 8271 Replay ROM £10. Tel. (0782) 550207 after 6pm.

Wanted for BBC B. Printer, joystick, word processor chip plus spell check, mouse and games for absolute beginner. Tel. (0203) 591084 evenings for more information.

Master refs 1&2, £9 each, advanced sideways RAM user guide, £7, Master operating system; Dabs Press £7, incl. software, or £20 the lot. Tel. (0254) 61492.

WANTED: 'Advanced Disc User Guide' by Pharo. Tel. 051 531 7140.

Master 512, DOS 1.2, GEM, mouse, Shibumi problem solver, all manuals plus Dabs Guide and Shareware Vol II, Acorn teletext adaptor, ATS 3.0, Viewstore, Viewindex, Barbarian II, Superman and many more £500. Tel. (0344) 55772.

HCR 21V EPROM blower £10, 10x 27128 EPROMs £3 each, 6x 6264 SWR RAM chips £3.50 each, Disc games; Spycat £5, Icarus £5. Tel. (0395) 512660.

Turbo 65C102 co-processor board for Mster 128, (or BBC B/B+ with a co-pro adaptor) includes Advanced Basic by Tubelink £80 or will exchange for a 512 board. InterWord V1 02 £27. Tel. 051 647 5367.

BBC Master, 2 cartridges, dust cover £300. BBC B+ ADFS, ATPL board, new 64k upgrade (not installed) £250, BBC B 32k upgrade, ADFS, ATPL board £225, All with Interword, 6502 second processor £50, Cumana CD800S double drive with PSU £130, AMS double 3" drive (uses Amstrad discs) £100, Taxan and Ferguson 12" green monitors £50 each, AMX mouse & SuperArt £25, Interface equipment, ROMs and books. Tel. 01 836 7039 for list.

BBC issue 3 DNFS, Acorn 40T S/S S/D disc drive, Acorn AP100A dot matrix printer, 60 discs (most contain BBC software) £250 or can split. Various books, write for details to: 2 Juniper Drive, Allesley Green, Coventry CV5 7QH.

BBC B (series 4) with Solidisc 256k 4Meg board and WW+, Interword, Commstar. Twin d/s 40/80 Opus disc drives with PSU. Kaga Vision-III hi. res. RGB monitor, Nightingale modem. Split possible. Offers to (0252) 617309.

BBC issue 7 complete with DDFS and joystick £200, 2 d/s d/d 40T drives £60, med/res "Cub 1451" monitor £150, ATPL ROM board with 16k RAM and utilities ROM £30, Solidisk 32k RAM board £20, 27128 16k (21v) EPROMs £3.50 each, EpROM Programmer £10, BBC Advanced User Guide, Basic ROM User Guide £5, BBC printer cable £5, 55 Micro User magazines (offers), 45 Acorn User magazines (offers), The Home Computer Advanced Course (two bound volumes) £10, Assembly Language Programming for BBC (Birnbaum) £3.50, Beginners Assembly Language course for BBC (book & tape) £5, many discs some with software on, offers? Tel. (0332) 556381.

Three BBC games are yours to keep if you will transfer them to disc for me. Spacestation Alpha by Icon, Spitfire by Alligata, 3D Grand Prix by Software Invasion. Please Tel. 01- 878 1962.

BBC B OS 1.20 DFS, with ATPL, Sidewise extension board, £120, Akhter (1987) single sided 40/80 DD £75, Teco TM1265 monochrome monitor £30, View £25, Viewsheet £25, Wordwise + £25, Wordease £10, Masterfile II £10, all for £275. Tel. (0865) 250680.

BBC Master 512, dual d/s Cumana CD800S discs, Microvitec med/res colour, 3 Master ROM adaptors, Peartree 64k battery backed RAM cartridge, 2 joysticks, 2 mice, Acorn tape drive, ADT, Publisher, Powerfont NTQ, MasterROM, Userdump, Viewspell ROMs, 27 program discs including Viewstore, StopPress, ArtRoom, Speech, Heritage, ImaginA and -L, Bank Manager etc. 4 BBC Master 512 discs, all manuals and numerous books. £650. Tel. (0235) 21436.

BBC B issue 7, Acorn DFS, Viglen Console, built in twin drives, Solidisk 128k SRAM, ROM Cartridge System, AMX Mouse, VIEW, System Delta, AMX software, etc £350. Acornsoft Sotware Developers Toolbox, unused £80. Tel. (05394) 33441.

Twin 80T disc drives with PSU £50, tape based Gemini Beebcalc spreadsheet £3. Tel. (0684) 563408.

Sleuth £10, Norwich CP ROM III £10, Exmon II £15, all for BBC boxed and perfect. Tel. (0684) 572295.

WANTED: Tandy TS-80 CGP-115 Colour Graphic Printer must be in good condition. Tel. (0533) 792702.

Acorn Archimedes 310M mono system, fitted with RISC OS in perfect order, (upgraded to 440 Arc) includes some software. £895 o.n.o. Tel. (0234) 740866 eves.

Master 512 v2.1 Microvitec Cub, Cumana 80T d/s Twin drives, WW+, Spellcheck III, BROM+ Masterfile, Modem & Software, AMX art & mouse all manuals. Books on DOS Basic M/C Word processing etc. 4 bound BEEBUG vols. Plenty of Software. All boxed & in VGC £700 o.n.o. Tel. (0375) 677640.

View Professional £33, Exmon II £7, BBC soft Monitor £7, all complete and in original packing. Tel. (0304) 368644 before 8pm.

Master 512, 2 x 40/80T DS DD, Shinwa CP-80 printer, Demon modem, RGB TV, reference manuals, books, firmware, software (including Shareware and Public Domain), mice etc. Prefer to sell complete. Offers?? Tel. 045527 4018.

WANTED: 512 Board with or without co-pro adaptor. Tel. (0295) 711383.

Universal Teletext adaptor and Micronet 800 modem £75. Tel. (0903) 755412.

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

BACK ISSUE PRICES (per issue)

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

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

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

POST AND PACKING

Please add the cost of p&p as shown opposite.

BEEBUG
117 Hatfield Road, St.Albans, Herts AL1 4JS
Tel. St.Albans (0727) 40303, FAX: (0727) 60263
Manned Mon-Fri 9am-5pm
(24hr Answerphone for Connect/Access/Visa orders and subscriptions)

BEEBUG MAGAZINE is produced by BEEBUG Ltd.

Editor: Mike Williams
Assistant Editor: Kristina Lucas
Technical 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.

Printed by Newnorth-Burt Ltd (0234) 41111 ISSN - 0263 - 7561

Magazine Disc/Cassette

OCTOBER 1989 DISC/CASSETTE CONTENTS

DIFFUSION LIMITED AGGREGATION - another example of the intriguing visual patterns which result from apparently random processes.

A PRINTER COMMAND UTILITY - prepare your own customised printer control routine.

AMATEUR RESEARCH (Part 1) - program demonstrating the ideas discussed in the first article of this new series.

A BEEBUG GRAPH PLOTTER (Part 1) - a complete working program to plot graphs of the form y=f(x) or x=f(y).

FIRST COURSE (Visual Sorting) - a fascinating visual investigation of five different sorting techniques.

A DATAFILE SUPERDUMP - examine datafiles in detail with this handy debugging aid.

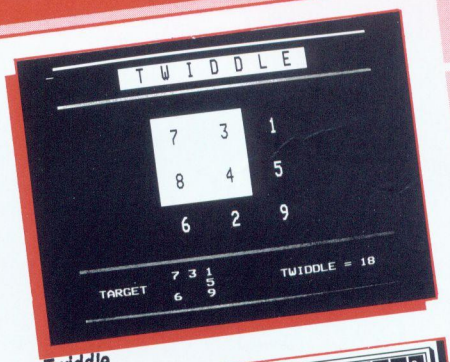
INTRODUCING POSTSCRIPT (Part 2) - a program showing the conversion of expressions to reverse Polish notation.

FONT DESIGNER (part 2) - the completed program together with a number of sample fonts.

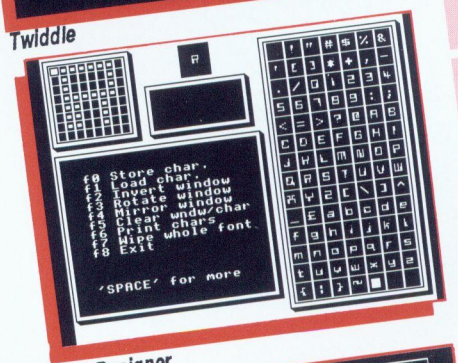
TWIDDLE - a frustratingly simple form of sliding block puzzle.

A GENERAL PURPOSE LINE-INPUT FUNCTION - the complete input routine and demo program repeated from last month.

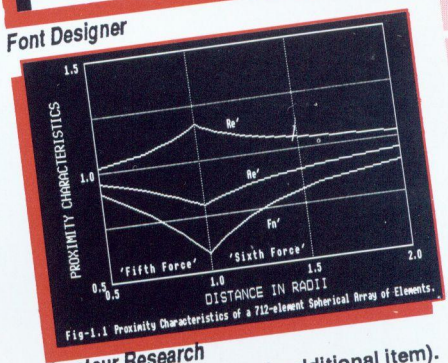
512 FORUM - a SORT utility for DOS Plus.



Twiddle



Font Designer



Amateur Research

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

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

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

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

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

FREE ON SITE MAINTENANCE

with all A3000's and Archimedes FROM BEEBUG

the Archimedes Specialists

COMPLETE CONFIDENCE

Quite simply this means that if your Archimedes goes wrong within the first year, an engineer will call at your premises within 24 hours to repair it. If it cannot be repaired, a replacement will normally be left.

NATIONWIDE COVERAGE

Beebug are using CRAY ELECTRONICS PLC to provide the maintenance cover which is available throughout the whole of mainland UK. Simply make one phone call and an engineer will call. There is no limit to the number of call-outs that you can make over the year.

ALL ARCHIMEDES & A3000

This service is included on all new A3000, A410, A420 and A440/1 systems sold by Beebug from September 10th 1989. It also covers the Acorn monitor if sold at the same time.

PRICES	BASE	COLOUR
A3000	649.00	869.00
A410/1	1199.00	1419.00
A420/1	1699.00	1919.00

Special Offers

In addition to On-Site Maintenance, if you subscribe to BEEBUG or RISC User, (or take out a new subscription) and purchase an Acorn A3000 we will also supply: Fish, Pacmania, Printer Lead, 10 x 3.5" Discs and Lockable Disc Box. With an Archimedes purchase you can choose FREE goods to the values shown below:

	Colour System	Entry System
A410/1	£123	£110
A420/1	£180	£167
A440/1	£272	£259

Speedy Service

We have a new showroom in St. Albans where you can see and try any of the systems. They are always on display and in stock. Our Mail Order Service is also very efficient, with over 80% of orders going out the same day. 48 hour fully insured delivery charges on complete system are just £7.00 and 24 hour delivery £11.50.

0% Finance

As a licensed credit broker, Beebug can offer you 0% finance over 9 months or 13.75% flat rate (typical APR 28%) over 12,24 or 36 months. Please phone to check the members special offers with 0% finance.

Technical Support

Unlike many dealers we don't lose interest as soon as you have made your purchase. You can always telephone our Technical Support Department or Showroom for some friendly and impartial advice.

Archimedes Magazine – RISC User

RISC User is the leading magazine dedicated solely to the Archimedes and A3000. For £14.50 a year (UK), you will receive this 56 page magazine packed full of useful information, news, reviews, programs, hints and tips etc; mailed directly to your home 10 times a year.

Ordering

All items are currently in stock. Prices are exclusive of VAT, please add 15%. We accept payment by cash, cheque, Visa, Access, Connect etc as well as official orders. You may order by phone, fax or letter. Alternatively please phone for more information.

GUARANTEE

As a BEEBUG customer you can rely on our guarantee of unbeatable service and support.

BEEBUG

117 Hatfield Road, St. Albans, Herts AL1 4JS.
Tel: 0727 40303 Fax: 0727 60263