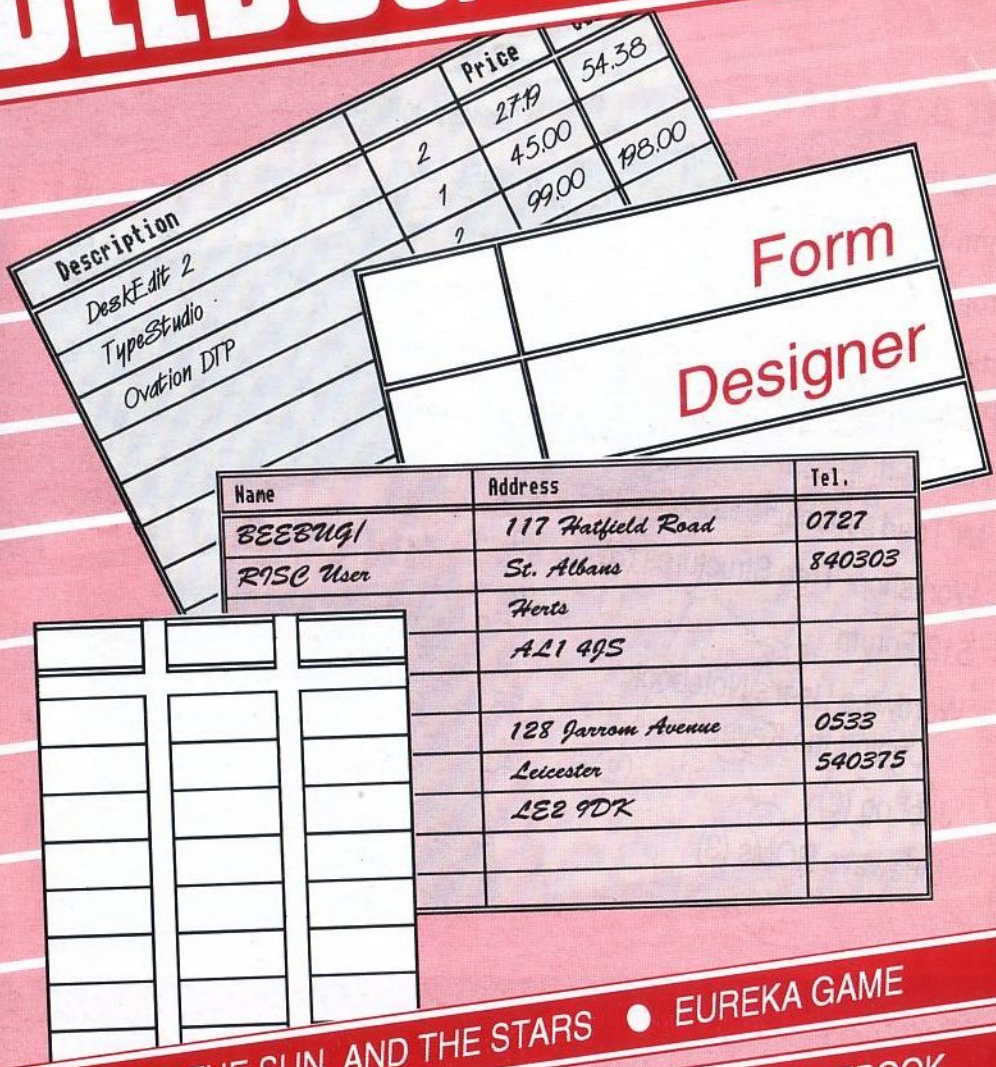
Vol.11 No.8 January/February 1993

BEEBUG

FOR THE BBC MICRO & MASTER SERIES



- FIXING THE SUN AND THE STARS
- EUREKA GAME
- TROUBLESHOOTING
- WORDWISE USER'S NOTEBOOK

BEEBUG Vol.11 No.8 January/February 1993

FEATURES

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

```
SUN & STARS
                    Currently:  1:4:1992
Input > Date                    00.04
        Time, GMT               84°30'W
        Latitude                000°00'
        Longitude
        SHA & Dec, star

Find    Altitude & Azimuth - Sun
        Maximum Altitude
        Sunrise & Sunset
        Altitude & Azimuth - Star

        Quit

Cursor keys to select        RETURN to execute
```

Fixing the Sun and the Stars

Name	Address	Tel.

Form Designer

```
EUREKA

Try to match the target using
A B C D E F and / only once
combined with ( ) + - and *
Complying with BBC Basic syntax

Round 1

Target number= 706

A=1
B=2
C=7
D=9
E=5
F=75

To correct input errors
Press Delete key, which
erases the whole line.

(F*B)*E-(C+D)-A
```

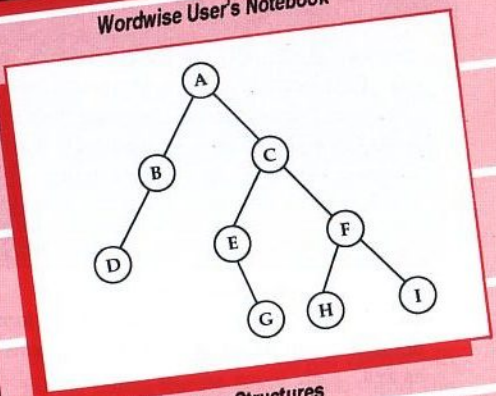
Eureka

Key string	Key Function
!!<Space>	f0
!!!	f1
!!"	f2
!!#	f3
!!$	f4
!!%	f5
!!&	f6
!!'	f7
!!(	f8
!!)	f9
!!,	Cursor L

Wordwise User's Notebook

```
Base date:- today, Wednesday 02.12.1992
Principal Index                  Deadline ____ Lead time
Finish Lemnings................* Sat 01.01.00 + 369 w 3 d
Edit BEEBUG Jan/Feb............* Sat 05.12.92 +     3 d
See Lydia......................* Wed 02.12.92 +   Today
Xmas Shopping..................* Thu 24.12.92 +   3 w 1 d

Lead times are
from 00:00 hrs

Or type/tick
Extend Event/Date
Delete a row
Modify a row
Sort this list
Print this list
Save this list
Change base date

ESCAPE
to quit

Use the cursor
keys to select
the option
and key RETURN
```

JobLog



Tree Structures

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

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

Program needs at least one bank of sideways RAM.

Program is for Master 128 and Compact only.

Editor's Jottings

Welcome to the first issue of 1993. Amazingly we will soon be nearing the end of yet another volume of BEEBUG, and the commencement of volume 12.

As readers will no doubt be aware, there are very few new products, if any, being produced these days for the BBC micro. This month we have therefore amalgamated the editorial column with the news page, and we will continue this practice whenever appropriate. One consequence of this, of course, is that we have an extra page to allocate to other editorial features.

This issue of BEEBUG, like the one before it, has a number of strong items which we hope will appeal to many readers. Following the item on Jupiter's satellites, which appeared in the November issue (see also comment in this month's Postbag), we have another astronomy related program in this issue. There is also the concluding part of the JobLog personal organiser program, and the first part of yet another useful and practical application, a form designer. There are all our usual regular items as well.

As always, we look forward to your views and comments on what we publish in the pages of BEEBUG, and contributions for publication in these pages are always welcome from readers. All material which we publish is paid for. To help you, we can also supply a copy of our "Notes of Guidance for Contributors" on receipt of an A5 (or larger) stamped addressed envelope.

BEEBUG PROGRAMS IN THE PUBLIC DOMAIN

If you read this month's article on public domain software by Alan Blundell, you will see that he has been pursuing the idea of encouraging software houses which have more or less abandoned their software titles for the BBC micro, to consider putting these programs into the public domain. This seems quite a good idea, provided that adequate disc-based information can also be provided to make an application understandable.

To this end, we have decided to make the contents of BEEBUG magazine discs from volumes 1 to 5 available in the public domain via Alan Blundell. All such programs remain copyright BEEBUG, and it will be up to Alan to decide which programs he provides, and in what format. All later BEEBUG programs will continue to be available only from RISC Developments Ltd., the publishers of BEEBUG magazine.

A condition of this move is that we will not be able to provide any support for these earlier programs. Public domain software is supplied for the cost of distribution alone, and at that price should be accepted for what it is. Given the age of this software, it is unlikely that the original authors will be able to help either. However, many interesting programs were published in BEEBUG throughout those issues, and by allowing them to be distributed in this way does at least make them available to all BBC micro users.

Any queries about the availability of BEEBUG programs in the public domain MUST be addressed to Alan Blundell, BBC PD, 18 Carlton Close, Blackrod, Bolton BL6 5DL.
M.W.

Fixing the Sun and the Stars

Bob Dixon-Stubbs shows how to find out exactly where they are.

The height and direction of the Sun or the stars, the times of sunrise and sunset and the maximum height of the Sun can all be found accurately with this program, from anywhere on the Earth's surface and for any date after 1900. Use it to see when it will get dark during your holiday in Timbuktu, to identify the stars, or to get an accurate reading to calibrate your car compass. Play with it and you'll find that "the Sun sinks slowly in the west" is one of the wilder generalisations - according to where you are, it can set anywhere between north and south as long as there is a bit of left-hand bias!

Type the program in and save it. When run, a menu will be displayed allowing date, time, latitude and longitude to be set, and offering a choice of calculations. Start by making a check, since mistakes in typing in the constants could result in errors which are not obvious. The cursor will be opposite 'Date', so press Return and enter day 7, month 5 and year 2000. Time, latitude and longitude will all be zero. Use the cursor keys to place the cursor against 'Altitude & Azimuth - Sun', press Return and when the result appears return to the menu and quit. Type:

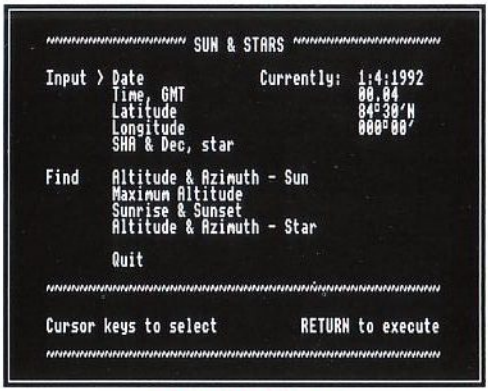
```
PRINT az
```

which should give 2.850331. If not - hunt for the error!

USING THE PROGRAM

The input variables can be entered in any order, and can be changed independently. Latitude and longitude are always required and are entered in degrees and minutes as two separate values. A date is also needed - the year can be entered either as YY or 19YY, but after 1999 it must be entered in full. Dates prior to 1901 are invalid, since the program thinks that 1900, being divisible by four, was a leap year (which it wasn't).

The times of sunrise and sunset, the maximum height that the Sun will reach (and the time it gets there - seldom noon) can all be found from the above inputs.



The Menu screen

Enter time in addition to the above - in GMT, using the 24 hour clock - and the altitude (height) and azimuth (bearing from true north) can be found. Altitude is shown as 0° when the Sun is on the horizon to 90° when overhead (or directly under one's feet if it's on the other side of the world - below horizon cases are annotated as such). Azimuth is 0° if due north, increasing clockwise through 360°.

The position of each star is defined by its Siderial Hour Angle (SHA) which runs from 0° to 360° and Declination (analogous to latitude). Approximate positions can be found from star charts (in the Times Atlas for example), more accurate positions for many stars are given in Nautical Almanacs (held in the larger reference libraries), encyclopedia and astronomical books. The term Right Ascension is sometimes used - SHA = 360° - 15*Right Ascension in hours.

Unlike the Sun, planets and moon, star positions have a long shelf life - many change by less than one minute of arc in a century. For those seriously afflicted by star-gazing, they could therefore safely be added as data or filed on disc, and called by star name or number.

To get the altitude and azimuth of a star, enter the date, time and co-ordinates as above, select 'SHA & Dec, Star', and enter the SHA and Declination of the star in degrees and minutes. Then select 'Altitude & Azimuth - Star'.

Zone times are the standard 15° bands of longitude. They are often correct, but the program does not allow for non-standard zone times nor for summer times.

If the program is used in conjunction with a compass, allow for the difference between true and magnetic north. If using it with a map such as Ordnance Survey sheets, remember that grid north and true north are not the same - use the longitude gradations to get a true north value.

WHAT IT'S DOING

The position of the Sun is defined by its Greenwich Hour Angle or GHA (analogous to longitude but going from 0 to 360°) and its Declination. To find these the value of the Equation of Time is needed. This is mainly a function of the elliptical path of the Earth round the Sun, and it is calculated in seconds by *PROCet*. *PROCaltaz* then solves the spherical triangles to find altitude and azimuth.

The maximum altitude is determined by using the Equation of Time to find the local time at which the Sun is exactly due north or south of the specified position, and then calculating the associated altitude.

Sunrise and sunset occur when the upper limb of the Sun appears to be tangential to the true horizon, so 16′ of arc must be allowed for the radius of the Sun. In addition, there is some 34′ due to atmospheric refraction when the Sun is at very low altitudes. *PROCss*, the sunrise and sunset routine, is therefore looking for an altitude of 50′ below the horizon.

The routine starts with a seed time (in local time) which depends on whether or not the Sun and the specified position are in the same hemisphere. Declination is calculated, and the loop checks to see if the Sun rises or sets. If it does, a time is found at which it is 50′ below the horizon, this time is used to recalculate declination, and the process is repeated until two successive iterations are within 30 seconds of time.

The formulae can fix the position of the Sun to within one minute of arc, but computational errors may exceed this at very high latitudes or when the altitude of the Sun or star is nearly 90°. Precise observation of a rising or setting Sun in high latitudes, or of the azimuth of a celestial body when it is virtually

overhead are both extremely difficult, so such errors are of no consequence.

The Siderial Hour Angle of a star is measured from a meridian which passes through a theoretical point known as the First Point of Aries. *PROCaltazstar* calculates the Greenwich Hour Angle of this point and adds it to the SHA to get the GHA of the star. Altitude and azimuth can then be calculated as for the Sun.

APPARENT ODDITIES

At high latitudes the program may give a sunrise time but report "No sunset" - try 1:4:92 at 84°30'N. This is the start of permanent day - the previous day will have a rise and a set, the next day neither. At the end of summer the opposite will occur.

If a maximum altitude has been found and the 'Altitude' option is then called without entering time, the program will use the exact time it found for the maximum altitude, and azimuth will be as expected - 0° or 180°. If the time of maximum altitude is entered using the 'Time' option and the 'Altitude' option used, azimuth may be slightly different, since the time used then will be to the nearest minute.

```
   10 REM Program CELEST
   20 REM Version B 1.00
   30 REM Author  Bob Dixon-Stubbs
   40 REM BEEBUG  Jan/Feb 1993
   50 REM Program subject to copyright
   60 :
  100 ONERROR CLS:REPORT:PRINT" at line
";ERL:GOTO 350
  110 MODE0:@%=&70A:DIM days%(12):*FX4,1
  120 FOR I%=2TO12:READ days%(I%):NEXT
  130 VDU23,240,120,72,72,120,0,0,0,0
  140 VDU28,16,24,79,3:lt=0:lg=0
  150 yy%=0:mm%=0:dd%=0:hr=0:sha=0:dec=0
:ds=0:ns=0:ew=0:t$=STRING$(48,"~")
  160 REPEAT
  170 CLS:op=2:PRINTt$:VDU31,17,0:PRINT"
 SUN & STARS "'''Input   Date";
  180 PRINTSPC14"Currently:  ";dd%":"mm%
":"yy%'SPC8"Time, GMT"SPC21;
  190 PRINTLEFT$(FNtime(hr,""),6)'SPC8"L
atitude"SPC22FNdd(lt,2,240)CHR$ns
  200 PRINTSPC8"Longitude"SPC21FNdd(lg,3
,240)CHR$ew'SPC8"SHA & Dec, star"'
  210 PRINT"Find    Altitude & Azimuth -
 Sun"'SPC8"Maximum Altitude"
  220 PRINTSPC8"Sunrise & Sunset"'SPC8"A
ltitude & Azimuth - Star"''SPC8"Quit"
  230 PRINT't$'''"Cursor keys to select"S
PC10"RETURN to execute"''t$
  240 REPEAT
  250 PRINTTAB(6,op)">";:ptr=GET:VDU127
  260 op=op-(ptr=138)+(ptr=139)
  270 IF op>13 op=13 ELSEIF op<2 op=2
  280 UNTIL ptr=13:CLS
  290 IFop=2 PROCdate ELSEIFop=3 PROCtim
  300 IFop=4 PROClat ELSEIFop=5 PROClong
  310 IFop=6 PROCstar ELSEIFop=8 PROCalt
  320 IFop=9 PROCmax ELSEIFop=10 PROCss
  330 IF op=11 PROCaltazstar
  340 UNTIL op=13
  350 VDU26,14:*FX4,0
  360 END
  999 :
 1000 DEF PROCdate
 1010 INPUT"Day   "dd%
 1020 INPUT'"Month "mm%
 1030 INPUT'"Year  "yy%:IF yy%<100 yy%=1
900+yy%
 1040 ENDPROC
 1050 :
 1060 DEF PROCtim
 1070 INPUT"Hours (GMT) "hr
 1080 INPUT'"Minutes     "mn:hr=hr+mn/60
 1090 ENDPROC
 1100 :
 1110 DEF PROClat
 1120 INPUT"Degrees latitude "lt
 1130 INPUT'"Minutes latitude "mn
 1140 lt=lt+mn/60:IF lt=0 ns=0:ENDPROC
 1150 REPEAT:INPUT'"North or South  (N o
r S) ";temp$:ns=ASCtemp$ AND &5F
 1160 UNTILns=78 ORns=83:IF ns=83 lt=-lt
 1170 ENDPROC
 1180 :
 1190 DEF PROClong
 1200 INPUT"Degrees longitude "lg
 1210 INPUT'"Minutes longitude "mn
 1220 lg=lg+mn/60:IF lg=0 ew=0:ENDPROC
 1230 REPEAT:INPUT'"East or West (E or W
) ";temp$:ew=ASCtemp$ AND &5F
 1240 UNTILew=69 ORew=87:IF ew=87 lg=-lg
 1250 ENDPROC
 1260 :
 1270 DEF PROCet
 1280 date=(yy%-1900)*365+INT((yy%-1901)
/4)+days%(mm%)+dd%+hr/24-.5
 1290 IF yy%MOD4=0 AND mm%>2 date=date+1
 1300 jc=date/36525
```

```
 1310 ml=279.697+36000.769*jc-360*(yy%-1
900):ml=ml+360*(ml>360):IF op=11 ENDPROC
 1320 et=-(93+14.23*jc-.0144*jc^2)*SIN(R
AD(ml))
 1330 et=et-(432.5-3.71*jc-.2063*jc^2)*C
OS(RAD(ml))
 1340 et=et+(596.9-.81*jc-.0096*jc^2)*SI
N(RAD(2*ml))
 1350 et=et-(1.4+.28*jc)*COS(RAD(2*ml))
 1360 et=et+(3.8+.6*jc)*SIN(RAD(3*ml))
 1370 et=et+(19.5-.21*jc-.0103*jc^2)*COS
(RAD(3*ml))
 1380 et=et-(12.8-.03*jc)*SIN(RAD(4*ml))
 1390 dec=DEG(ATN((.43382-.00027*jc)*SIN
(RAD(ml-et/240))))
 1400 gha=hr*15+et/240+180:IF gha>360 gh
a=gha-360
 1410 ENDPROC
 1420 :
 1430 DEF PROCaltaz(dln)
 1440 lha=gha+lg:lha=lha+360*((lha>360)-
(lha<0))
 1450 ht=DEGASN(SINRADlt*SINRADdln+COSRA
Dlt*COSRADdln*COSRADlha)
 1460 temp=(SINRADdln-SINRADlt*SINRADht)
/(COSRADlt*COSRADht)
 1470 IF ABStemp>1 temp=SGNtemp
 1480 az=DEGACStemp:IF lha<180 az=360-az
 1490 ENDPROC
 1500 :
 1510 DEF PROCalt
 1520 PROCet:PROCaltaz(dec):PROCposndate
 1530 PRINT"Time"SPC18FNtime(hr," GMT")'
 1540 PRINT"Altitude"SPC14FNdd(ht,0,240)
 1550 PRINT"Azimuth"SPC15FNdd(az,3,240)
 1560 PRINT'''"Any key for Menu";:M=GET
 1570 ENDPROC
 1580 :
 1590 DEF PROCmax
 1600 hr=12-lg/15:PROCet:hr=hr-et/3600
 1610 PROCet:PROCaltaz(dec):PROCposndate
 1620 PRINT'"Maximum altitude"SPC6FNdd(h
t,0,240)
 1630 PRINT"Time"SPC18FNtime(hr+(lg+7.5*
SGNlg)DIV15," Zone")
 1640 PRINTSPC22FNtime(hr," GMT")
 1650 PRINT'''"Any key for Menu";:M=GET
 1660 ENDPROC
 1670 :
 1680 DEF PROCss
 1690 FOR I%=1 TO 2
 1700 none=FALSE
 1710 hr=12-lg/15:PROCet:hr=hr-et/3600:I
F SGNdec=SGNlt hr=hr+12*(2*(I%=1)+1)
 1720 REPEAT
 1730 xhr=hr:PROCet
 1740 IF ABSdec>ABS((90-ABSlt)*SGNdec-5/
6*SGNlt) none=TRUE:GOTO 1770
 1750 cossd=-(1.45438973E-2+SINRADlt*SIN
RADdec)/COSRADlt/COSRADdec
 1760 hr=12+DEGACScossd*(2*(I%=1)+1)/15-
et/3600-lg/15
 1770 UNTIL (ABS(hr-xhr)<1/120) OR none
 1780 IF I%=1 PROCposndate
 1790 IF none=TRUE no$="No "ELSE no$=""
 1800 IF I%=1 PRINT'no$"Sunrise";ELSE PR
INT'no$"Sunset ";
 1810 IF none=TRUE PRINT:GOTO 1840
 1820 PRINTSPC15FNtime(hr+(lg+7.5*SGNlg)
DIV15," Zone")
 1830 PRINTSPC22FNtime(hr," GMT")
 1840 NEXT
 1850 PRINT'''"Any key for Menu";:M=GET
 1860 ENDPROC
 1870 :
 1880 DEF PROCstar
 1890 INPUT"SHA star"'''"Degrees  "sha
 1900 INPUT"Minutes  "mn:sha=sha+mn/60
 1910 INPUT'"Dec star"'''"Degrees  "ds
 1920 INPUT"Minutes  "mn:ds=ds+mn/60
 1930 REPEAT:INPUT'"North or South  (N o
r S) ";temp$:ud=ASCtemp$ AND &5F
 1940 UNTILud=78 ORud=83:IF ud=83 ds=-ds
 1950 ENDPROC
 1960 :
 1970 DEF PROCaltazstar
 1980 PROCet:gha=ml-180+15*hr+sha:gha=gh
a+360*((gha>360)-(gha<0))
 1990 PROCaltaz(ds):PROCposndate
 2000 PRINT"Time"SPC18FNtime(hr," GMT")'
 2010 PRINT"Altitude"SPC14FNdd(ht,0,240)
 2020 PRINT"Azimuth"SPC15FNdd(az,3,240)
 2030 PRINT'''"Any key for Menu";:M=GET
 2040 ENDPROC
 2050 :
 2060 DEF FNdd(dat,nd,chr)
 2070 dh=INTABSdat:mn=INT((ABSdat-dh)*60
+.5):IF mn=60 dh=dh+1:mn=0
 2080 bh$="":IF nd=0 nd=2+(dh<10):IF ht<
0 bh$=" below horizon"
 2090 fmt$=RIGHT$("000"+STR$(dh),nd)+CHR
$chr+RIGHT$("00"+STR$(mn),2)
 2100 IF chr=46 =fmt$ ELSE =fmt$+"'"+bh$
 2110 :
 2120 DEF FNtime(tm,sfx$)
 2130 pn=0:IF tm>24 OR tm<0 pn=SGNtm
 2140 IF pn=-1 sfx$=sfx$+" (previous day
)"ELSEIF pn=1 sfx$=sfx$+" (next day)"
 2150 =FNdd(tm-24*pn,2,46)+sfx$
 2160 :
 2170 DEF PROCposndate
 2180 PRINT"Co-ordinates"SPC10FNdd(lt,2,
240)CHR$ns" "FNdd(lg,3,240)CHR$ew
 2190 PRINT"Date"SPC18;dd%":"mm%":"yy%
 2200 ENDPROC
 2210 :
 2220 DATA31,59,90,120,151,181,212,243,2
73,304,334
```

Form Designer (Part 1)

Lol Taylor presents a comprehensive form designer for the Master.

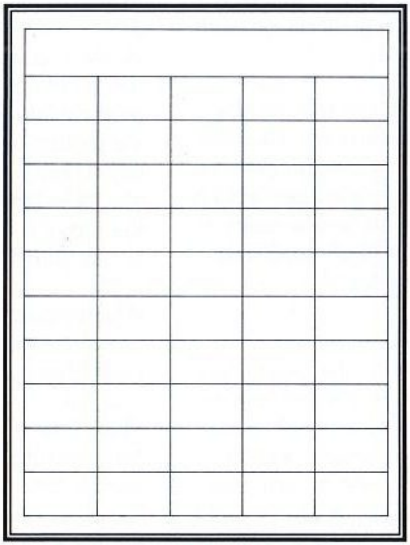
A lot of us, at one time or another, use forms to record information. The problem often is that we don't have access to exactly the layout we would like and end up with something scribbled on a tatty piece of paper. The series of programs presented here will help with this by allowing you to develop your own layouts as hard copy. The forms are actually designed by creating a Basic program so that, although the system is not that friendly it is very flexible.

The software is in three parts. The first, *SetUpCh*, defines a set of screen characters to reflect the printer characters, and these are used to show your design on screen as you develop it. The second, *FdShell*, is a shell program into which you add the form design instructions themselves. Two examples of these instructions are provided here.

Before you start, a word about printers is needed. The Epson FX-850, on which the programs are based, has characters for printing out lines and corners which are used for designing borders and boxes, these are numbered &B0 (176) to &DF (223) and are selected with DIP switch SW1-3, other printers will have similar, if not the same, facilities - you will need to check this with your manual.

GETTING STARTED

Let's look at the software in action first, then we'll go on to explain how it works. First type in and save SetUpCh and FdShell as listed below. Reload FdShell and add the lines in *+Boxes* - save this as *FdBoxes*. This program creates the simple form shown in the illustration. Run SetUpCh to redefine the screen characters; these will then remain defined until you press Break or switch off. Set up your printer and run FdBoxes, following the prompts. There are one or two reminders about printers, and then you will see the form on screen. The proportions of this will be a little distorted but the layout will be correct. After the screen picture you will be able to send the form to your printer. If you get complete rubbish first check the printer is set up for the graphic character set. If it still won't work don't loose heart, as long as your printer can produce lines and corners the program can be customised.



The form produced by FdBoxes

DESIGNING FORMS

To help you design your own forms let's have a look at how the FdBoxes form was created. Keep in mind that this is designed using double width printing so it is 48 characters wide and 80 lines high.

First we need to be able to deal with the graphics printer characters in a meaningful way. *PROCsetupChars* gives string names to the form characters available. If you study them in the printer manual (CHR$176-223) you will see that in most of them, lines radiate from the centre point of the character in two or more directions. To help my memory they have been given variable names based on the points of the compass where capital letters represent double lines and lower case letters single lines. So *WNSE$* is a cross in double lines, whereas wnse$ is a cross in single lines. *NSwe$* produces two vertical lines crossed by a single line. This is the procedure you may need to alter to suit your own printer codes.

The above variables are the smallest building blocks of the form. As the form is almost bound to repeat certain patterns there are three building stages which allow you to create and re-use units to cut down development time - each unit is stored as a string variable.

SHORT STRINGS

These are dealt with in the procedure *shortstrings* (lines 6000 to 6999). There are four 'short' strings to be created, 'short' doesn't refer to actual length, rather to the complexity. We need a line eight characters long (line 6010), a row of eight spaces (line 6020), a line 44 characters long (line 6030) and a row of 44 spaces (line 6040). You can call these strings what you like but meaningful names always help so L8$ is a line eight characters long, s8$ eight spaces.

COMPLETE STRINGS

PROCcompletestrings (lines 7000 to 7999) is where you assemble strings that are the full width of the form; in FdBoxes there are six of these. The first is a line across the page with a corner at each end (line 7010), the second a row of spaces across the page with a vertical line at each end (line 7020) and so on. When creating these more complex strings it is very helpful to sketch your form out on graph paper.

THE FORM PROPER

PROCprintout assembles the whole form on the screen or printer. As you can see it's a series of loops, some nested, to print the complete strings in the correct order to produce the final form. The procedure ends with a form feed to the printer just to tidy thing up.

While you are developing your form on the screen it will be useful to run the program at various stages to see how you are getting on. To help with this function key f0 has been set up to enter RUN. As you develop the form you will find that the top of it will be scrolled off the screen so use Shift-Ctrl to stop this.

When you are ready to print out your form you will need to set up the printer procedures; these are *PROCprinterOn* and *PROCprinterOff* at lines 4000 and 4200. This is simply a set of VDU codes that configure the printer the way you want it before printing - the examples in the program refer to the FX-850 - and re-set it at the end.

COMPASS CODES

If you would like a print out of the codes available to you in *PROCsetupChars* with their compass names adjacent, then load FdBoxes and add or change the lines in *+Codes* as listed below.

There are two new procedures which are necessary for selecting the right compass

codes and their graphic display in the appropriate boxes. They are listed in lines 5200 to 5710.

Theoretically this program is quite a jump forward at this stage, but don't be put off by it. After studying the examples next month in the second part of this article you will find matters becoming much clearer.

```
   10 REM Program FdShell
   20 REM Version B 1.0
   30 REM Author  Lol Taylor
   40 REM BEEBUG  Jan/Feb 1993
   50 REM Program subject to copyright
   60 :
  100 title$="SHELL":ON ERROR GOTO10000
  110 *K.0 RUN|M
  120 MODE7:PROCtitle
  130 REM delete the next two lines if n
ot required.
  140 PRINT'':q%=FNyn("Is SW1-3 up")
  150 IFNOTq% PRINT''"Please attend to i
t and restart":END
  160 PROCsetupChars
  170 PROCshortstrings
  180 PROCcompletestrings
  190 PROCcopies
  200 MODE0
  210 PROCprinterOn
  220 FORcopies=1TOn%:PROCprintout:NEXT
  230 VDU1,7,1,7,1,7:REM Reminder from p
rinter
  240 PROCprinterOff
  250 IF n%=0 PRINT''"Press any key to co
ntinue":IFGET
  260 MODE7
  270 REM delete the next line if not re
quired
  280 PRINT''"Remember to put SW1-3 down
 when finished"
  290 FORI%=1TOn%:SOUND1,-15,150,3:SOUND
1,-15,134,3:NEXT
  300 OSCLI"FX5,1"
  310 END
  320 :
 1000 DEF PROCc(Q$,Y%):REM centre text
 1010 LOCAL T%
 1020 T%=(40-LEN(Q$)) DIV 2
 1030 PRINTTAB(0,Y%);STRING$(T%," ");Q$
 1040 ENDPROC
 1050 :
 1200 DEFFNyn(A$):REM Yes/no?
 1210 LOCAL reply$
 1220 PRINT' A$;" (Y/N) ? ";
 1230 REPEAT
 1240   reply$ = CHR$(GET AND &DF)
 1250 UNTIL reply$="Y" OR reply$="N"
 1260 PRINTreply$;
 1270 PRINT
 1280 =(reply$="Y")
 1290 :
 1400 DEFFNrj(f$,L%):REM right justify
 1410 f$=STRING$(L%-LEN(f$)," ")+f$
 1420 =f$
 1430 :
 1600 DEF PROCcopies
 1610 OSCLI"FX5,1"
 1620 CLS:PROCc("How many copies ?",15)
 1625 PROCc("RETURN only if print-out no
t required",17)
 1627 INPUTTAB(19,19)n%:PRINTTAB(19,19)"
  "
 1630 PROCc(STR$(n%),19)
 1640 IFn%=0 OSCLI"FX5,0"
 1650 PROCc("Press any key to proceed or
<ESCAPE>",21):IFGET:CLS
 1660 ENDPROC
 1670 :
 2000 DEF PROCsetupChars
 2010 WN$=CHR$188:WNS$=CHR$185:WNE$=CHR$
202:WNSE$=CHR$206:WS$=CHR$187
 2020 WSE$=CHR$203:WE$=CHR$205:WEn$=CHR$
207:WEs$=CHR$209:WEns$=CHR$216
 2030 Wn$=CHR$190:Wns$=CHR$181:Ws$=CHR$1
84:NS$=CHR$186:NSE$=CHR$204
 2040 NSw$=CHR$182:NSwe$=CHR$215:NSe$=CH
R$199:NE$=CHR$200:Nw$=CHR$189
 2050 Nwe$=CHR$208:Ne$=CHR$211:SE$=CHR$2
01:Sw$=CHR$183:Swe$=CHR$210
 2060 Se$=CHR$214:En$=CHR$212:Ens$=CHR$1
```

```
98:Es$=CHR$213:wn$=CHR$217
 2070 wns$=CHR$180:wne$=CHR$193:wnse$=CH
R$197:ws$=CHR$191:wse$=CHR$194
 2080 we$=CHR$196:ns$=CHR$179:nse$=CHR$1
95:ne$=CHR$192:se$=CHR$218
 2090 B$=CHR$219:BW$=CHR$221:BN$=CHR$223
:BS$=CHR$220:BE$=CHR$222
 2100 b1$=CHR$176:b2$=CHR$177:b3$=CHR$17
8:`$=CHR$35
 2110 ENDPROC
 2120 :
 2400 DEF PROCtitle
 2410 FORI%=0TO6:PRINTCHR$151:NEXT
 2420 PRINTTAB(8,0);CHR$224;:FORI%=1TO21
:VDU240:NEXT:PRINTCHR$176
 2430 PRINTTAB(8,1);CHR$106;TAB(30,1);CH
R$53
 2440 PRINTTAB(7,2);CHR$141;CHR$106;" FO
RM DESIGN UTILITY ";CHR$53;TAB(7,3);CHR$
141;CHR$106;" FORM DESIGN UTILITY ";CHR$
53
 2450 PRINTTAB(8,4);CHR$106;TAB(30,4);CH
R$53
 2460 PRINTTAB(8,5);CHR$162;:FORI%=1TO21
:VDU163:NEXT:PRINTCHR$33
 2470 PROCc(title$,12)
 2480 wait%=INKEY(200)
 2490 PROCc("""SetUpCh"" must have been
RUN",14):PROCc("if screen representation
 is required",15)
 2500 wait%=INKEY(200)
 2510 ENDPROC
 2600 :
 4000 DEF PROCprinterOn
 4010 REM Enter printer instructions her
e
 4020 ENDPROC
 4030 :
 4200 DEF PROCprinterOff
 4210 REM Enter printer defaults here
 4220 ENDPROC
 4230 :
 5000 DEF PROCprintout
 5010 REM Enter complete strings in prin
t-out sequence here
 5020 ENDPROC
 5030 :
 6000 DEF PROCshortstrings
 6010 REM Assemble part-strings here
 6020 ENDPROC
 6030 :
 7000 DEF PROCcompletestrings
 7010 REM Assemble complete strings here
 7020 ENDPROC
 7030 :
10000 REM Error routine
10010 PROCprinterOff
10020 OSCLI"FX5,1"
10030 MODE7
10040 REPORT:PRINT" at line "ERL
10050 ONERROROFF
10060 END
```

```
   10 REM Program Setupch
   20 REM Version B 1.0
   30 REM Author  Lol Taylor
   40 REM BEEBUG  Jan/Feb 1993
   50 REM Program subject to copyright
   60 :
  100 MODE7
  110 PROCredefine
  120 PRINTTAB(10,6);CHR$130;CHR$141;"FO
RM DESIGNER":PRINTTAB(10,7);CHR$130;CHR$
141;"FORM DESIGNER"
  130 PRINTTAB(9,9);CHR$130;"SCREEN CHAR
ACTERS"
  140 FORI%=1TO10:PRINTCHR$134:NEXT
  150 VDU28,1,19,39,10
  160 PRINT'"The characters 35,176-223 h
ave been"'"redefined. They will stay thu
s until:"
  170 PRINT'''  The computer is switched
 off, OR"'"     CTRL+BREAK is pressed, O
R"'"      they are defined again."
  180 VDU26
  190 PRINTTAB(4,20);CHR$130;CHR$136;"No
w LOAD the next program."
  200 I%=INKEY(100)
  210 REM Enter CHAIN"<your program>" he
re if you like (but make sure you save t
```

```
his program before RUNning it)
  220 END
  230 :
 1000 DEF PROCredefine
 1010 VDU23,35,28,54,48,124,48,48,126,0
 1020 VDU23,176,0,170,0,170,0,170,0,170
 1030 VDU23,177,168,252,252,252,252,252,
84,0
 1040 VDU23,178,252,252,252,252,252,252,
0,0
 1050 VDU23,179,16,16,16,16,16,16,16,16
 1060 VDU23,180,16,16,16,240,16,16,16,16
 1070 VDU23,181,16,16,240,16,240,16,16,1
6
 1080 VDU23,182,40,40,40,232,40,40,40,40
 1090 VDU23,183,0,0,0,248,40,40,40,40
 1100 VDU23,184,0,0,240,16,240,16,16,16
 1110 VDU23,185,40,40,232,8,232,40,40,40
 1120 VDU23,186,40,40,40,40,40,40,40,40
 1130 VDU23,187,0,0,248,8,232,40,40,40
 1140 VDU23,188,40,40,232,8,248,0,0,0
 1150 VDU23,189,40,40,40,248,0,0,0,0
 1160 VDU23,190,16,16,240,16,240,0,0,0
 1170 VDU23,191,0,0,0,240,16,16,16,16
 1180 VDU23,192,16,16,16,31,0,0,0,0
 1190 VDU23,193,16,16,16,255,0,0,0,0
 1200 VDU23,194,0,0,0,255,16,16,16,16
 1210 VDU23,195,16,16,16,31,16,16,16,16
 1220 VDU23,196,0,0,0,255,0,0,0,0
 1230 VDU23,197,16,16,16,255,16,16,16,16
 1240 VDU23,198,16,16,31,16,31,16,16,16
 1250 VDU23,199,40,40,40,47,40,40,40,40
 1260 VDU23,200,40,40,47,32,63,0,0,0
 1270 VDU23,201,0,0,63,32,47,40,40,40
 1280 VDU23,202,40,40,239,0,255,0,0,0
 1290 VDU23,203,0,0,255,0,239,40,40,40
 1300 VDU23,204,40,40,47,32,47,40,40,40
 1310 VDU23,205,0,0,255,0,255,0,0,0
 1320 VDU23,206,40,40,239,0,239,40,40,40
 1330 VDU23,207,16,16,255,0,255,0,0,0
 1340 VDU23,208,40,40,40,255,0,0,0,0
 1350 VDU23,209,0,0,255,0,255,16,16,16
 1360 VDU23,210,0,0,0,255,40,40,40,40
 1370 VDU23,211,40,40,40,63,0,0,0,0
 1380 VDU23,212,16,16,31,16,31,0,0,0
 1390 VDU23,213,0,0,31,16,31,16,16,16
 1400 VDU23,214,0,0,0,63,40,40,40,40
 1410 VDU23,215,40,40,40,255,40,40,40,40
 1420 VDU23,216,16,16,255,16,255,16,16,1
6
 1430 VDU23,217,16,16,16,240,0,0,0,0
 1440 VDU23,218,0,0,0,31,16,16,16,16
 1450 VDU23,219,255,255,255,255,255,255,
255,255
 1460 VDU23,220,0,0,0,0,255,255,255,255
 1470 VDU23,221,240,240,240,240,240,240,
240,240
 1480 VDU23,222,15,15,15,15,15,15,15,15
 1490 VDU23,223,255,255,255,255,0,0,0,0
 1500 ENDPROC
```

```
   10 REM Program +Boxes
  100 title$="BOXES":ON ERROR GOTO10000
 4000 DEF PROCprinterOn
 4010 REM Printer codes: on, elite, doub
le width, 8 lines to the inch, English f
ont
 4020 VDU2,1,27,1,77,1,27,1,87,1,1,1,27,
1,48,1,27,1,82,1,3
 4030 ENDPROC
 4040 :
 4200 DEF PROCprinterOff
 4210 REM Default printer codes restored
 in reverse order and off
 4220 VDU1,27,1,82,1,0,1,27,1,50,1,27,1,
87,1,0,1,27,1,80,3
 4230 ENDPROC
 4240 :
 5000 DEF PROCprintout
 5010 PRINT'''
 5020 PRINTa1$
 5030 FORI%=1TO7:PRINTa2$:NEXT
 5040 PRINTa3$
 5050 FORJ%=1TO9
 5060 FORI%=1TO5:PRINTa4$:NEXT
 5070 PRINTa5$
 5080 NEXT
 5090 FORI%=1TO5:PRINTa4$:NEXT
 5100 PRINTa6$
 5110 VDU1,12
```

Continued on page 56

Troubleshooting Guide (Part 2)

Gareth Leyshon points out more pitfalls for users.

In this second article I'll be looking at the keyboard, sideways ROMs and RAMs, and present a few hints that don't collect under any specific heading.

KEYBOARD PROBLEMS

After a few years of punishment, some keys may begin to work erratically. In the worst case, you will need to get your keyboard serviced. But first, try blowing around the key - get close to the keyboard and give a good hard puff. Failing this, prise the top off the key (lever a nailfile or thin-bladed screwdriver under the key's front edge and twist) and blow hard on the space beneath. Replace the keytop with a gentle press. If this doesn't help either, repeatedly pressing the key usually gets it working long enough to finish your document, but beware, over-violent treatment may cause physical damage.

If you find that a key absolutely refuses to work then all is not lost. You can redefine the Tab key to give the character you need. This has the advantage (unlike using one of the red function keys) that if the faulty key is alphabetic or one which gives a different symbol with Shift, then Tab will respond correctly to Shift, Ctrl, Caps Lock and Shift Lock. You do not lose the Tab function, either, for holding Ctrl and pressing 'I' will still generate a proper tab.

To make Tab produce ASCII code c, for example, (look up the ASCII code of the character you need in the Welcome Guide or type PRINT ASC"c"), type the command:

```
*FX 219,99
```

*FX 219 re-defines the Tab key and 99 is the ASCII code for c. To make the redefined Tab key work properly with Shift, then it must be defined to give the unshifted version of the key it replaces, i.e. a lower case letter or a number. Remember that the Tab key reverts to normal whenever Break is pressed, and will need to be re-defined if this happens. It may be the case that you can't type the FX command because it includes the character that doesn't work. If that is a number, say 5, then type PRINT 4+1 and use the cursor keys and Copy to copy the 5; if it is one of the letters of *FX, VDU 42,70,88 will generate a line that you can again duplicate using the cursor keys and Copy. If more than one key has failed, you will have to implement the others using the function keys, with separate keys defined for the upper and lower case versions. This is done by entering:

```
*KEY1 c
```

which would program key f1 to produce 'c'.

Master users can tell the keyboard whether or not they want Caps Lock switched on: *CONFIGURE CAPS brings the computer on in Caps Lock mode, while *CONFIGURE NOCAPS puts Caps Lock off. A third setting, *CONFIGURE SHCAPS, gives you the lock but pressing Shift together with a letter key will give you the lower case letter, a mode which Master and most Beeb users can access at any time by pressing Caps Lock while holding down Shift.

*CONFIGURE DELAY m and *CONFIGURE REPEAT n control

keyboard rates. When a key is pressed it is registered once. The computer waits for m hundredths of a second (called the delay) and then re-registers the key every n hundredths of a second (the repeat rate). These can also be changed directly, until Break is pressed, on both Beeb and Master by *FX 11,m for the delay and *FX 12,n for repeat.

SIDEWAYS THINKING

In theory, the Beeb or Master can cope with sixteen so-called Sideways ROM chips which carry programming languages or utilities. What you can do in practice depends on the machine you have. A standard Beeb has only four sockets (for chips numbered 12-15) of which one will hold the Basic language and another the disc filing system. You can buy extra circuit boards in which to plug up to twelve other ROMs.

The Master contains one spare chip socket (number eight) inside the computer. Sockets 9 to 15 are taken up by Basic, the disc filing facilities and the other systems built-in; type *ROMS to see what they are. Slots 0 to 3 are provided via the two cartridge sockets. Normally slots 4 to 7 are taken up with four banks of extra memory called Sideways RAM. Properly-prepared software can be transferred from a disc into this memory where it acts as if it was on a built-in ROM until you turn the power off. In a few rare cases (perhaps if a machine is a reconditioned second-hand one) two banks of Sideways RAM (either 4 & 5 or 6 & 7) may have been deselected allowing extra chips to be installed internally hence rendering two slots unavailable. If you're very unlucky, both pairs may be deselected - you must get your dealer to make a small hardware correction inside the machine.

Some users may be using a BBC Model B+128 machine. This can be identified by it giving an OS number 2.0 in reply to typing *HELP. This also possesses four banks of sideways RAM - in this case in slots 0, 1, 12 and 13. In the following description of SRLOAD, the numbers 0,1,12,13 (B+) or 4,5,6,7 (Master) may be used explicitly, or the codes W,X,Y,Z work on either machine.

Data provided to be placed in sideways RAM - usually referred to as ROM images - is transferred into sideways RAM using the *SRLOAD command. This is only standard on the B+ and Master; Beeb users who have some sort of sideways RAM made by a third party manufacturer will have their own instructions. To take a typical example, the schools' wordprocessor EDWORD, filename *EDWRD2E*, would be transferred to RAM using:

```
*SRLOAD EDWRD2E 8000 W Q
```

where 8000 is the memory location (nearly always with the value of 8000), Q denotes 'quickly' and the W is the bank of memory to use. You must then perform a Ctrl-Break before the RAM is ready for use, after which *EDWORD will activate this particular piece of software. If this fails, a possibility is that you don't have a usable bank 4 as described above. In this case repeat the process and change the W for a Z. If this works you know you only have two usable banks of RAM instead of four. If you omit the Q the process takes rather longer - but including the Q causes the current main memory contents to be wiped (that memory is used to speed up the process) so if you want to keep something that's in memory at the time, leave out the Q.

Sometimes a particular ROM clashes with a piece of software, or two ROMs

use identically named commands for different jobs. In this case Master users can type *ROMS for a list showing which slot which ROM is in, and type *UNPLUG &n, where n is the ROM number, to render any of the ROMs unusable, effective from the next Ctrl-Break (this setting is remembered while the computer is off). *ROMS shows which ROMs are unplugged and *INSERT &n will reinstate any of them.

MISCELLANEOUS TIPS

If you find at any time that the cursor keys and Copy are not working as they usually do to copy characters from the screen, typing *FX 4 should restore their usual function. If the cursor vanishes, pressing one of the cursor keys may be sufficient to revive it; if not then try typing MODE 7 - changing mode cures most cursor and screen layout problems.

To get the computer working properly you must type in instructions exactly as shown on any instruction sheet or manual. Using lower case letters where the computer wants capitals, or inserting spaces (or leaving them out) can make what seems quite intelligible to you just gibberish to the computer. One easy mistake to make, especially when using the cursor keys and Copy, is to copy trailing spaces where they aren't needed: beware of failing to find a match when using the Master's LIST IF command simply because 'term ' doesn't match 'term'.

If a command doesn't work, check the way it's typed. Be careful: don't confuse letters i and l with digit 1, or letter O with digit 0. Zero is often written with a distinguishing slash (Ø) which is how it appears on the keyboard, and sometimes on screen.

In a few cases, a Basic program which works well on a Beeb refuses to do so on a Master. One cause may be that the screen display needs a "non-shadow" mode as discussed last time. If this fails, try using the program called *CONVERT* as described in the Master's Welcome Manual. Note, however, that it is certainly NOT necessary to run *CONVERT* with every program you use, as I found one user doing.

Other freak circumstances make the computer seem to hang up; if all else fails, press Break a few times, switch the computer off for more than a minute, or try disconnecting and reconnecting all the leads while switched off. There seems no logical reason behind some of these, but they have been known to work. You can also try removing unnecessary peripherals from the system, just in case they are causing some unforeseen clash. If this does the job, why knock it? You can check, of course, that the problem is not something simple like the power switch at the back of the disc drive being in the off position (not all drives have one, and it's easy to forget). If you are using two drives with individual power supplies, you may need to switch on the power to the drive you are not using, otherwise the one that you are using may not work properly.

In my next article I shall look at printers, and in subsequent issues at disc drives. For any users who still use cassette, the following tip may help: if you have trouble loading data when using jack plugs, you may find that easing the jack from the EAR socket so it is about a millimetre short of being fully inserted may sometimes improve matters. That's all for this time - happy troubleshooting!

Eureka

Alan Gray helps you brush up your gameshow skills.

Eureka is a mathematical game designed to test your mental arithmetic, based loosely on the numbers game from Channel 4's Countdown. The rules are fairly simple, you are given five small random numbers (range 1-10), one large random number (25, 50, 75 or 100), and a random target number (range 101-999).

```
EUREKA

Try to match the target using
A B C D E F and / only once
combined with ( ) + - and *
Complying with BBC Basic syntax

Round 1

Target number= 706          A=1
                            B=2
                            C=7
                            D=9
To correct input errors     E=5
Press Delete key, which     F=75
erases the whole line.
```

Starting the game

The six random numbers are held in the variables *A*, *B*, *C*, *D*, *E* and *F*. The object of the game is to try and construct a formula using some or all of these variables, along with the operators +, -, *, / and up to 3 pairs of brackets, so that the formula yields a result as close as possible to the target number. The / operator must only be used to give an integer result; failure to comply will give a zero score.

The variables *A* to *F* are used instead of *A%* to *F%* to simplify inputs from a single key stroke, using GET$. A typical formula is in the form (A-B)*F-C/D; here MOD D must be zero.

```
EUREKA

Try to match the target using
A B C D E F and / only once
combined with ( ) + - and *
Complying with BBC Basic syntax

Round 1

Target number= 706          A=1
                            B=2
                            C=7
                            D=9
To correct input errors     E=5
Press Delete key, which     F=75
erases the whole line.

(F*B)*E-(C*D)-A
```

Doing the best I can . . .

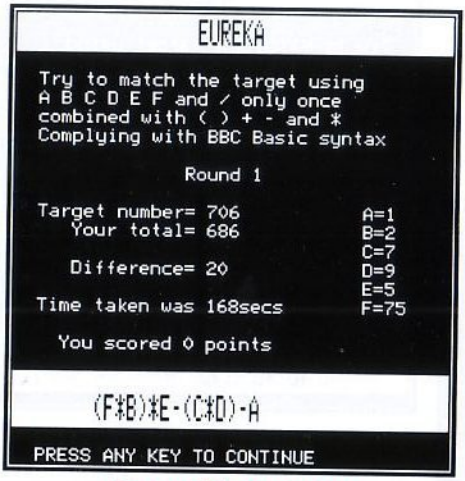
ENTERING THE FORMULA

Each key press is checked for syntax etc., before inclusion in the formula, and you cannot enter the same variable or the divide symbol more than once. If Return is pressed with missing closed brackets or an incomplete statement, then you are warned and may carry on with the next entry. If you enter some sign or variable by mistake, you may delete the whole formula by pressing the Delete key, but the clock isn't reset to 0.

SCORING

The difference (positive or negative) between the target number and the result of the formula is used to calculate the score. You score nothing if the difference is more than 10, and with the difference between 4 and 10 you score 10 points. Naturally the scores are higher when the

difference is only between 1 and 3 (20 points) and if there is no difference at all, you score 50 points. To give the game a little edge, the score is doubled if you manage to enter the formula within 1 minute, but if you take longer than 2 minutes the score is halved.



But not doing very well

The game consists of 8 rounds of solving the formulae. The final score is the sum of all 8 rounds plus a bonus for consistent scoring. The bonus is doubled every time a round scores more than zero, with a maximum of 250 points when you score in all 8 rounds. If you are fast accurate and lucky, you can score more than 1000 points.

Good luck!

```
 10 REM Program Eureka
 20 REM Version B 1.0
 30 REM Author  Alan Gray
 40 REM BEEBUG  Jan/Feb 1993
 50 REM Program subject to copyright
 60 :
100 DIM tag(6),m$(9),rn(6),sc%(8)
110 b$=CHR$132+CHR$157+CHR$131
120 r$=CHR$129+CHR$157+CHR$135
130 cy$=CHR$134+CHR$157+CHR$132
140 d$=CHR$141
150 topsc%=100
160 FOR j=1 TO 9:READ m$(j):NEXT
170 REPEAT
180 FOR j%=1 TO 8:sc%(j%)=0:NEXT
190 n%=0
200 REPEAT
210 n%=n%+1
220 FOR j%=1 TO 6:tag(j%)=0:NEXT
230 PROCscreen(8)
240 PRINTTAB(3,4)"Try to match the tar
get using"
250 PRINTTAB(3,5)"A B C D E F and / on
ly once"
260 PRINTTAB(3,6)"combined with ( ) +
- and *"
270 PRINTTAB(3,7)"Complying with BBC B
asic syntax"
280 A=RND(4)
290 REPEAT:B=RND(4):UNTIL B<>A
300 C=RND(6)+4:D=RND(6)+4
310 REPEAT:E=RND(6)+4:UNTIL E<>D
320 F=RND(4)*25
330 IF F=25 T%=RND(499)+100 ELSE T%=RN
D(899)+100
340 rn(1)=A:rn(2)=B:rn(3)=C
350 rn(4)=D:rn(5)=E:rn(6)=F
360 PRINTTAB(16,9);"Round ";n%;
370 PRINTTAB(3,11);"Target number= ";T
%
380 PRINTTAB(3,15);"To correct input e
rrors"
390 PRINTTAB(3,16);"Press Delete key,
which"
400 PRINTTAB(3,17);"erases the whole l
ine."
410 FOR j=1 TO 6
420 PRINTTAB(32,10+j);CHR$(64+j);"=";r
n(j)
430 NEXT
440 f$="":pos=0:pdiv=0:br1=0:br2=0
450 TIME=0:nc=0:pc=0
460 PRINTTAB(0,21);
470 REPEAT
```

```
 480 ch=GET:ok=0:test=-1
 490 IF ch=127 THEN ch=7: PROCdel
 500 IF ch>64 AND ch<71 THEN PROCvar
 510 IF ch>41 AND ch<46 THEN PROCoper
 520 IF ch=40 OR ch=41 THEN PROCbrac
 530 IF ch=47 THEN PROCdiv
 540 IF ok THEN PROCformula
 550 IF ch=13 AND pos=0 THEN ch=7:PROCe
rr(9)
 560 IF ch=13 AND pc=3 THEN ch=7:PROCer
r(7)
 570 IF ch=13 AND br1<>br2 THEN ch=7:PR
OCerr(2)
 580 UNTIL ch=13
 590 time%=TIME/100
 600 IF pdiv>0 THEN PROCchdiv
 610 FOR L%=15 TO 17:PRINTTAB(3,L%);SPC
(25):NEXT
 620 PROCscore
 630 sc%(n%)=score
 640 PRINTTAB(3,24)"PRESS ANY KEY TO CO
NTINUE";
 650 w=GET
 660 UNTIL n%=8
 670 CLS:PROCtable
 680 UNTIL FALSE
 690 END
 700 :
1000 DEF PROCscreen(s%)
1010 CLS
1020 FOR j=0 TO 2:PRINTcy$:NEXT
1030 FOR j=3 TO s%:PRINTb$:NEXT
1040 FOR j=s%+1 TO 19:PRINTr$:NEXT
1050 FOR j=20 TO 22:PRINTcy$:NEXT
1060 PRINTTAB(16,1)d$;"EUREKA"
1070 PRINTTAB(16,2)d$;"EUREKA"
1080 ENDPROC
1090 :
1100 DEF PROCdel
1110 FOR j=1 TO pos
1120 PRINTTAB(j+7,21);" "
1130 PRINTTAB(j+7,22);" "
1140 NEXT
1150 f$="":pos=0:pdiv=0:br1=0:br2=0:nc=
0:pc=0
1160 FOR j=1 TO 6:tag(j)=0:NEXT
1170 FOR j=1 TO 6
1180 PRINTTAB(28,10+j);r$:NEXT
1190 ENDPROC
1200 :
1210 DEF PROCformula
1220 f$=f$+CHR$(ch):pos=pos+1
1230 PRINTTAB(6,21)d$;TAB(pos+7,21);CHR
$(ch)
1240 PRINTTAB(6,22)d$;TAB(pos+7,22);CHR
$(ch)
1250 ENDPROC
1260 :
1270 DEF PROCvar
1280 IF pc=2 OR pc=4 THEN PROCerr(8): E
NDPROC
1290 IF tag(ch-64)=0 THEN ok=-1:pc=4:ta
g(ch-64)=1:nc=nc+1 ELSE PROCerr(5)
1300 IF ok PRINTTAB(28,ch-54);b$
1310 ENDPROC
1320 :
1330 DEF PROCoper
1340 IF nc=6 THEN PROCerr(1):ENDPROC
1350 IF ch=44 ENDPROC
1360 IF pc=2 OR pc=4 THEN ok=-1:pc=3
1370 ENDPROC
1380 :
1390 DEF PROCdiv
1400 IF pdiv>0 THEN PROCerr(3):ENDPROC
1410 IF pc=0 OR pc=1 OR pc=3 THEN PROCe
rr(8):ENDPROC
1420 IF pdiv=0 THEN pdiv=pos+1:ok=-1:pc
=3
1430 ENDPROC
1440 :
1450 DEF PROCchdiv
1460 test=-1
1470 den$=MID$(f$,pdiv+1,1)
1480 num$=MID$(f$,pdiv-1,1)
1490 IF den$="(" THEN n=pdiv+1:PROCfcb
1500 IF num$=")" THEN n=pdiv-1:PROCfob
1510 den%=EVAL(den$)
1520 IF den%=0 THEN test=0:PROCerr(4):E
ND
1530 IF NOT test PROCerr(1):ENDPROC
1540 div$=num$+"/"+den$
1550 num=EVAL(div$)
1560 IF ABS(num-INT(num))<.0001 THEN te
```

Continued on page 28

Public Domain Software

Alan Blundell's campaign to revitalise old commercial Beeb software as PD or shareware gets an airing this month...

At the end of my last column, I said that I would give the up to date story about my dealings with commercial software houses who once marketed good quality software for the BBC micro, but who no longer sell it, or at least no longer market it so that anyone can find it for sale.

The idea occurred to me some time ago that very little software is actively marketed nowadays for our 'mature' models. Nevertheless, much of it was good software and, of course, it still is. As there is still wide interest in the BBC micro, often from new users who have bought a secondhand machine, it seemed a shame that they might no longer have the opportunity to see the vast range of software which exists. So, I started to write to software houses which I knew of.

My proposal was that their software products which are no longer marketed could be distributed for them by PD libraries (I actually only mentioned my own, but that wasn't the point). This would give users the chance to see their software, give the software house wider recognition and perhaps also the goodwill of a group of 'computerate' people most likely to upgrade to an Arc one of these days, which would be of benefit to those companies who now concentrate on the later micros. I also mentioned the shareware concept as a possibility so that they could even generate some income from products which they no longer expected to make any money for them.

Early successes included Phoenix Software's excellent 'Fast Access' disc magazine. This is a subscription only magazine, although 'magazine' is probably the wrong word. It actually consists of regular distributions of discs containing a wide range of good quality software contributed by subscribers, all with documentation on disc in text files. Phoenix agreed to let me distribute the first two volumes (of 6 issues each), not as PD or as shareware, but nevertheless very cheaply. As I saw it, the benefit was that more people would have access to the software; from their point of view, there was the possibility of gaining new subscribers.

Alpine Software were another early success, when they agreed to let me distribute their 'Plague Planet' adventure game as shareware. John Lyons sent me a wide range of educational software which he had successfully marketed for specific college courses (mentioned previously in this column). And Howard Spurr gave the OK for his well regarded 'Disk Duplicator III' system to be distributed as PD.

I was encouraged by these to try a bit harder to persuade the software companies and by now I have written to more software houses than I previously knew existed. Of course, I haven't tried to persuade the most well-known companies which all readers will immediately recognise, because I am sure that their products are still selling well. I wrote to Garland Computing, though, who produce a range of quality educational software. They sent me a pleasant reply with good wishes for my efforts but pointed out that their products are still going strong. In fact, I was pleased to hear that, although their sales are obviously nothing like they were before the Arc came to prominence, they are experiencing a revival of interest

from customers. It shows that there is life in the old CPU yet!

The reply from Garland was in fact notable by its very existence; most of the letters (and sample PD catalogues, etc.) have gone without a reply. I suspect that many of the smaller software houses have gone altogether; companies whose names seasoned BBC users would recognise with a fond "Ah, yes, I remember Enigma Disc Imager...." (insert the software title of your choice) have simply disappeared and I have had a fair share of letters returned marked "not at this address".

However, there is a slow but steady flow of good news. Vine Micros has agreed to let their 'AddComm' and 'Matrix Rom' ROM images be distributed, and has offered special prices if anyone wants the full product, complete with original manuals. Both of these products are compatible with the Model B and Electron, but sadly not Master series micros; I suspect that this is because of the changes made in the Basic language ROM in the later machines. Both received excellent reviews in their day (they were first released in 1984/5).

AddComm extends the BBC Basic language by adding 40 extra commands. These are not star-commands as in most utility ROMs, but can be used as normal Basic commands in direct mode, within programs and even within multi-statement lines. The new commands fall into four main groups: graphics; Logo/turtle graphics; toolkit; and general purpose. The Matrix ROM is the answer to a prayer for anyone of a mathematical inclination who wants to solve linear simultaneous equations without resorting to programming languages other than BBC Basic.

Chris Reynolds, of CODIL Language Systems, is looking at ways of packaging his MicroCODIL expert systems language in such a way that it can be usable without the lengthy and comprehensive manuals for re-release, and has already given permission for his 'Psychebrot' disc of fractal graphics to be distributed. ('Psychebrot' was actually the first ever disc of 'careware' for the BBC - if you like it you are encouraged to make a donation to MIND, the mental health charity.) Thanks should go to Rik Gray, who actually suggested to Chris that he contact me - I hadn't yet got round to pestering him myself.

There are one or two other companies who are actively looking at their products and sorting out what is possible, although it isn't appropriate for me to pressurise them by mentioning them here. I have more yet to contact before I get into second stage persuasion on the companies who I know still exist but who haven't yet replied.

Also, several people have written to me with suggestions for likely candidates. I am following these up but would be interested to hear of any product which you may know of, whose author or copyright holder is still traceable, and which you consider worth rescuing from oblivion when it is no longer marketed. It would be a shame for all that good software to disappear without trace... If you have a suggestion to make, I can be contacted direct at 18 Carlton Close, Blackrod, Bolton, BL6 5DL (it would save BEEBUG a small fortune on the cost of forwarding the mail, if the response to this is anything like the response to my 'SPRITER' plea of a few issues ago!).

Next time, I plan to look at some more of the good quality software which has been contributed to the public domain direct by the writers. Looking back over previous columns, I find that there are many programs deserving a mention which I haven't yet got round to talking about.

1st course

Input (1)

Alan Wrigley describes the means by which Basic allows you to input data while your programs are running.

Most programs are interactive to some degree. That is, they expect to receive data from the user (normally typed in at the keyboard), and to display the results of any processing on the screen, via a printer or in a file. In addition, there is often a requirement for the user to indicate choice or confirmation by making a single keypress - perhaps to choose an item from a menu or to confirm the deletion of some data, and so on.

This article and the one which follows next month will look at the input side of this process. Basic provides a comprehensive set of statements for data input from the keyboard; these are GET, GET$, INKEY, INKEY$ and INPUT (the latter can take the optional extension LINE). The first four of these are essentially for the input of single characters, or the detection of single keypresses, and will be considered next month.

INPUT

Whenever a program requires an item of data to be typed in at the keyboard, whether it is a numerical value or a string, the keyword INPUT should be used. This suspends execution of the program until a line of text has been typed at the keyboard and Return pressed, whereupon the value of the data entered is placed into the specified variable. The syntax allows for quite a number of variations in the use of the statement, so we will look at a few examples. The simplest form is:

```
INPUT variable
```

where *variable* is the name of any type of variable (integer, real or string). A question mark will be displayed and the user's input awaited. If a string variable is used, then the resulting string will consist of the characters typed in. If it is a numeric variable, then Basic will try to make sense of the input as a number, or return a value of zero if it cannot.

If we take an actual example:

```
INPUT value%
```

in this case an integer is expected; entering "12" or "256" for example will set *value%* to that value. Entering "45.13" will set *value%* to 45, while entering "Hello world" will return a value of 0, and "24 Acacia Road" will return 24.

Using this simple form of the statement will always display a question mark for a prompt. However, in most cases you will want to prompt the user with some words of your own, such as "What is your name". To do this, you precede the variable name with a string as follows:

```
INPUT prompt variable
```

If you place a comma between the prompt and the variable name, the question mark will appear after the prompt, otherwise it will not. So you might use:

```
INPUT "What is your name",name$
```

if you are asking a question, or:

```
INPUT "Height of room: "height%
```

if you are not.

The prompt can contain other statements which relate to screen display, such as TAB or SPC. In fact, quite complex prompts can be displayed, using most of the facilities available with the PRINT statement. For example, you could specify a prompt string such as:

```
INPUT TAB(10,3)"Enter your name"'SPC(10)
"using upper case letters"'SPC(10)"and
not more than 10 characters" name$
```

which uses three strings separated by newlines (the ' characters) as well as TAB and SPC. However, you *cannot* include variables in your prompt string; if you think about it for a moment the reason is obvious - INPUT will assume that the first variable name it finds is to be used for the returned value. If you want to prompt with a variable, you must use a PRINT statement for the prompt, and follow it with INPUT to read the data.

When a string variable is used, you can input a whole line of data (up to about 238 characters). However, leading spaces will be stripped, and the string will be terminated at the first comma if there is one. In order to enter strings containing leading spaces or commas, you must use a variant of INPUT, which is INPUT LINE. This works in exactly the same way, except that *all* characters will be returned. If your programs request string input that may contain these elements, you should always use INPUT LINE rather than INPUT.

So far we have only described inputting one value at a time, but it is quite possible to input several values with one statement, for example:

```
INPUT "Enter height, width and depth",h%
,w%,d%
```

Each item is prompted in turn with a question mark on a new line. You can include more than one string prompt, too:

```
INPUT "Height",h%,"Width",w%
```

EVALUATION OF EXPRESSIONS

Often it is useful to be able to enter an expression instead of a simple value. This might save the user time when entering data. For example, if the volume of a room is required, and you know the room is 14 x 13 x 8 ft, it is much easier to type "14*13*8" than to get out the calculator. This can be handled quite easily by using a string variable with the INPUT statement, and then evaluating the string using EVAL, as in the following example:

```
INPUT "Volume of room: "input$
volume%=EVAL(input$)
```

Thus you will get the correct result whether the user types in a simple numerical value or an expression. EVAL can handle strings containing any functions which are normally used in Basic (SQR, SIN, OR, AND and so on).

NEATER DISPLAY OF INPUT

One of the problems with INPUT is that after the statement is issued and before Return is pressed, the program has no control over the process. It is quite easy for the user to enter a ridiculous value, or even to type in a string of excessive length which overwrites other important information on the screen, and the program can do nothing whatsoever about it. All that can be done is to indicate after the event that the input was not acceptable.

A further problem is that, even if the input is sensible, it may be a value which the program cannot handle. It must then repeat the request for input on a subsequent line, continuing until it gets a response that fits the requirements. The screen then just scrolls with one prompt after another, perhaps losing other information from the top of the screen. This does not look very professional, and can be frustrating for the user.

What is needed is an input routine which is completely under the control of the program, rejecting unacceptable characters and ensuring that the input remains within a well-defined area of the screen by repeating prompts in the same place as

before and refusing to allow overlength strings to be entered.

Listing 1 is an example of such a routine, which places a prompt at the specified screen co-ordinates, and then processes keypresses to compile an input string. The maximum length of the string can be specified, together with a list of characters which are to be allowed. If a null string is given then all are allowed. The routine uses the GET statement to get each character typed in turn. We will be looking more closely at this next month.

If maximum flexibility is required, the approach adopted here for allowable characters is rather cumbersome - the parameter string might end up being very long. However, for most purposes it is merely necessary to ensure that only numeric characters are entered when a numeric result is required, and so the most likely string passed as a parameter would be ".0123456789".

Listing 1

```
1000 DEF FNinput(prompt$,x%,y%,max%,al
low$)
1010 allow%=(allow$>""):in$=""
1020 PRINT TAB(x%,y%)prompt$;SPC(max%+
1)STRING$(max%,CHR$8);
1030 REPEAT:A%=GET
1040 IF A%<>13 AND allow%=0 OR INSTR(a
llow$,CHR$A%)>0 PROCchar
1050 UNTIL A%=13
1060 =in$
1070 :
1080 DEF PROCchar
1090 IF A%=127 AND LEN(in$)>0 in$=LEFT
$(in$,LEN(in$)-1):VDUA%
1100 IF A%<127 AND LEN(in$)<max% in$=i
n$+CHR$A%:VDUA%
1110 ENDPROC
```

You would call the routine in the following way:

```
a$=FNinput("Name:",0,10,20,"")
```

and if a numeric value is expected then an EVAL statement would follow.

The routine works as follows: firstly in line 1010 a flag is set *(allow%)* depending on whether allowable characters are restricted, and the output string *(in$)* is initialised. The prompt is then printed at the specified place in line 1020. After the prompt, spaces equal to the maximum input length are printed, to ensure that the space is blanked out if the input has to be repeated, followed by a string of backspaces (ASCII 8) of similar length to put the cursor back in the right place for input.

Lines 1030-1050 form a loop which processes each character typed in turn. If the character is not Return, and if either the restricted characters flag is unset or the character is within the string specified, the character is passed to PROCchar for processing (line 1040). PROCchar itself examines the character. If Delete was pressed (ASCII 127) and the length of *in$* is currently greater than zero, the string is shortened by one character (line 1090), while if the character is less than 127 and the length is less than the maximum, the character is added to the string (line 1100). In both cases the character is echoed to the screen with a VDU statement.

The routine as it stands is very basic but it will handle most requirements. It can easily be expanded, however; for example, you could accept function keys by incorporating the appropriate lines into PROCchar. For those who are interested, a rather more comprehensive data input routine was published in an earlier *1st Course* (Vol.9 No.3).

Mr Toad's Machine Code Corner

This month Basic will eat itself!

Now that the single market is finally upon us, I trust that you have all gone over to the new European Standard Byte of 7.3 bits. If not, you should solder a potentiometer between earth and track 8 on your Beeb's data bus, and adjust it until you get exactly 1.66 volts when the line goes high. Voila - sept point trois morceaux. But wait! I forgot. The Beeb was designed before 1896, and is therefore exempt. Phew! Nevertheless, whilst we are on a cosmopolitan note Mr T will let you in on the latest Kraze to sweep the international 6502 scene: Kamikaze Kode.

How many of you have written, or use, a Basic program which calls machine-code routines? Mr T wrote a specialised word processor for handling lists of names - nothing world-shattering, but handy. The associated machine-code routines speed up the processing a lot, but the procedure which assembles it takes up a lot of space which I could well use for data. OK, you say, save the object code to a separate file and load it in first. Yes, but Mr T hates that; not only is it fiddly and slow, but it gets in the way of the tinkering-around with the listings, which is what I'm really enjoying while Mrs T thinks I'm printing out membership lists for PMS Help. Many of us get a lot of enjoyment from messing about with our listings - oops, sorry, program development.

Mr T learned a lot and got a lot of fun from engineering one solution to the problem - Kamikaze Kode. The assembler part of the program does its job and then makes itself disappear, leaving just the Basic behind - plus, of course, the assembled object code. To get it back to work on it again, you just reload the one program, and everything is in one listing. The kamikaze routine itself can be assembled to a place which can later be overwritten, so it uses no extra memory whatsoever.

You could do it by noting the address of the desired cutoff point, and doing a few pokes from Basic, but you'd need to reset the address after even the smallest change. K.K. works by placing a marker in the listing at the point where you want the program to be cut off. To the existing assembly text you add a short routine which is called as soon as the assembly is finished. This routine works its way back from the end of the program towards the beginning until it finds the marker, then it cuts the program off at that point. The assembly text disappears and the space will be overwritten by the Basic variables and data. There are therefore a couple of constraints. One, the assembler routines have to be at the end of the Basic, as everything down-memory from the marker will disappear. It should be in a procedure which is called at the very beginning, before any variables are declared. That's not a problem; in fact, most people lay their programs out like that anyway.

Secondly, any labels declared by the assembly text are erased, since the variables live just above the end of the Basic. You can't write 'CALL code' in the Basic, because the variable .code no longer exists. Fortunately, the resident integer variables *A%* to *Z%* are not cleared by anything short of power-off: they live all on their own in page 4. You can, therefore, write *.A%* as a label in the assembly text and 'CALL A%' in the Basic, even if RUN, CLEAR or Kamikaze Kode has come between the two and wiped out all the other variables. Note

that if the object code is loaded in from a separate file, methods of calling it from the Basic are even more restricted.

Before we get down to details, a couple of points about the organisation of Basic itself. A Basic line begins with the line number in two bytes, MSB-LSB. Next comes one byte containing the length of the line, including the line number, itself and the final &0D. Then come the contents of the line, with keywords tokenised, and finally the carriage return &0D. For example:

```
10 REM123456
```

looks like this if you use a memory editor or peek the first ten locations after PAGE: 00 0A 0B F4 31 32 33 34 35 36 0D. 00 0A is the line number, 10. Next, 0B, that's a line length of 11 bytes. F4, the token for REM, then 31 32 33 34 35 36 for the numerals; then 0D, the carriage return. Try it, bearing in mind that if PAGE is at &0E00, the first line of Basic begins at &0E02, since (PAGE) always holds &0D and (PAGE+1) holds zero. Upset the line length with ?(PAGE+3)=9, and you won't be able to list the line; you get 'Bad program'. Repair the damage with ?(PAGE+3)=11 and you can list it again. After the &0D at the end of the last line, Basic puts &FF as an end-of-program marker.

Secondly, Basic stores the value of TOP in locations 0 and 1 - that's the address of the location *after* the &FF. The address of the end of the variables (called VAR-TOP, but Basic doesn't understand that mnemonic) is stored in locations 2 and 3. Basic also keeps a lot of other information in page zero, but 0 to 3 are all that matter here.

Our overall layout will be as follows: at the very beginning of the Basic program, call the procedure with the assembler in it. Let's name it *PROCcode*.

```
40 PROCcode
```

All line numbers in this article are, of course, only examples. The last part of the whole listing will be the procedure code, with our marker on the next line. You can't rely on a single character as a marker, as it would be far too likely to be duplicated by accident. Bear in mind that a line number or line length byte could accidentally be the same. Basic gets round that by calculating the function of these bytes from their context, but that would mean far too much code for this purpose. Instead, we'll use a combination which is very, very unlikely to be duplicated accidentally: I chose (%@. If, by Murphy's law, that sequence does get duplicated in your *PROCcode*, change it. OK...

```
3000 DEF PROCcode
3010 REM**(%@
```

The asterisks are just spacers, as we'll see in a minute; you can use any characters. What matters is the (%@. Then comes the existing assembly text.

The listing of the Kamikaze Kode could be anywhere in *PROCcode*, but I'm assuming that you've tacked it onto the end. If memory for the existing object code is tight, assemble the Kamikaze Kode somewhere else - I've used &7800. It can be overwritten when the Basic is working. Don't forget to come out of the assembler with a square bracket before P%=&7800, then back into the assembler with a left-hand bracket and restate OPT. You don't need a separate FOR - NEXT loop; keep everything inside the existing one. The last thing in *PROCcode* will be CALL &7800:ENDPROC. The Kamikaze Kode will truncate the procedure, but the return addresses which Basic has stored won't be affected, so that the ENDPROC after the call will successfully get you back to the beginning of the program, even though that ENDPROC is no longer inside the program. Right...

We'll need a subroutine which moves back by one address and updates the pointers. Basic's own pointers at 0 and 1 will have to be changed anyway, so we'll use those ourselves:

```
.backOne DEC 0:LDA 0
CMP #&FF:BNE ret
DEC 1
.ret LDA (0):RTS
```

This decrements the pointers and returns with A holding the contents of the byte pointed to. In fact, we'll place *.backOne* at the end, and begin our routine with a loop to whizz back up the listing looking for the *last* character of the marker:

```
.kami
JSR backOne
CMP #ASC"@":BNE kami
```

When that loop exits, 0/1 are pointing to a '@'. Now check for the other two characters, with a branch back to the loop if they're not there:

```
JSR backOne
CMP #ASC"%":BNE kami
JSR backOne
CMP #ASC"(":BNE kami
```

If we get here, we've found our marker. Now to store the address presently held in 0/1, as it's the one we need at the end - the one *after* Basic's &FF marker. Eventually, 2 and 3 will have to hold this address too, so let's store it there now:

```
LDA 0:STA 2
LDA 1:STA 3
```

Now we'll replace the second asterisk with Basic's &FF 'end-of-listing' marker:

```
JSR backOne
LDA #&FF:STA (0)
```

Now the next address up, presently also holding an asterisk, must hold the end-of-line &0D:

```
JSR backOne
LDA #&0D:STA (0)
```

At this point we have to go two bytes further up to adjust the line length byte. Why not stop en route and change the REM to an ENDPROC (token &E1)? That way, if you do ever have to restart the program after shortening, the call to *PROCcode* will hit an ENDPROC and return. If it hit a REM with no code following, it would simply stop.

```
JSR backOne
LDA #&E1:STA (0)
```

Finally, adjust the line length byte to 5 - there's only an ENDPROC, but remember those extra four bytes. Then replace the correct address of the new TOP in 0/1 and exit:

```
JSR backOne
LDA #5:STA (0)
LDA 2:STA 0
LDA 3:STA 1:RTS
```

Exit the assembler, tidy up and... dunnit! Try it - *after* careful saving - and list *PROCcode*. You should see only:

```
3000 DEF PROCcode
3010 ENDPROC
```

We have reclaimed for variables and data very nearly all the memory formerly occupied by the assembly text. The object code should be snug in its new home. Two points remain: one: whatever you do, don't go and save the sawn-off text under the same filename as the original; secondly, a limitation, Mr T hates programs that say things like 'The filename must be in capitals' - the computer should fix such things for the user. Here, however, I break my rule. The marker, REM**(%@, *must* be the first thing on its line, otherwise the routine can't find the line length byte. The program still runs, but you get 'Bad program' if you try to list or restart it. Mr T wrote a fix for this, but it's too long to justify inclusion here. This month's competition - write your own fix for the above problem. There are still lots of I'M A SWOT badges to be awarded.

Continued on page 58

Eureka (continued from page 19)

```
st= -1 ELSE test=0:PROCerr(6):ENDPROC
 1570 ENDPROC
 1580 :
 1590 DEF PROCfcb
 1600 REPEAT
 1610 n=n+1
 1620 IF MID$(f$,n,1)="(" THEN PROCfcb
 1630 UNTIL MID$(f$,n,1)=")" OR n=LEN(f$
)
 1640 IF MID$(f$,n,1)=")" THEN den$=MID$
(f$,pdiv+1,n)
 1650 ENDPROC
 1660 :
 1670 DEF PROCfob
 1680 REPEAT
 1690 n=n-1
 1700 IF MID$(f$,n,1)=")" THEN PROCfob
 1710 UNTIL MID$(f$,n,1)="(" OR n=1
 1720 IF MID$(f$,n,1)="(" THEN num$=MID$
(f$,n,pdiv-1)
 1730 ENDPROC
 1740 :
 1750 DEF PROCbrac
 1760 IF br2=br1 AND ch=41 PROCerr(8):EN
DPROC
 1770 IF ch=40 AND(pc=2 OR pc=4) PROCerr
(8):ENDPROC
 1780 IF ch=41 AND(pc=1 OR pc=3) PROCerr
(8):ENDPROC
 1790 IF br1<3 AND ch=40 br1=br1+1:ok=-1
:pc=1
 1800 IF br2<br1 AND ch=41 br2=br2+1:ok=
-1:pc=2
 1810 ENDPROC
 1820 :
 1830 DEF PROCerr(l%)
 1840 PRINTTAB(2,24);m$(l%);
 1850 SOUND 0,-10,1,5
 1860 w=INKEY(200)
 1870 PRINTTAB(2,24);SPC(37);
 1880 ENDPROC
 1890 :
 1900 DEF PROCscore
 1910 ans%=EVAL(f$):dif%=ABS(T%-ans%)
 1920 PRINTTAB(6,12);"Your total= ";ans%
 1930 PRINTTAB(6,14);"Difference= ";dif%
 1940 IF dif%=0 score=50
 1950 IF dif%>0 AND dif%<4 score=20
 1960 IF dif%>3 AND dif%<11 score=10
 1970 IF dif%>10 score=0
 1980 IF time%<60 score=score*2
 1990 IF time%>120 score=score/2
 2000 PRINTTAB(3,16)"Time taken was ";ti
me% "secs";
 2010 PRINTTAB(5,18);"You scored ";score
;" points"
 2020 ENDPROC
 2030 :
 2040 DEF PROCtable
 2050 PROCscreen(15)
 2060 tot%=0:top%=0:bon=1
 2070 FOR j%=1 TO 8
 2080 tot%=tot%+sc%(j%)
 2090 PRINTTAB(4,3)"Current top score =
";topsc%
 2100 PRINTTAB(4,j%+4);"round ";j%,sc%(j
%)
 2110 IF sc%(j%)>top% top%=sc%(j%)
 2120 IF sc%(j%)>0 THEN bon=bon+bon
 2130 NEXT
 2140 PRINTTAB(4,13)"Total ",tot%
 2150 tot%=tot%+bon
 2160 IF tot%>topsc% THEN topsc%=tot%
 2170 PRINTTAB(4,15)"Bonus ",bon
 2180 PRINTTAB(4,17)"Score ",tot%
 2190 PRINTTAB(4,21)" Another game? Y/N"
 2200 REPEAT:an$=GET$:UNTIL an$="Y" OR a
n$="N"
 2210 IF an$="N" END
 2220 ENDPROC
 2230 :
 2240 DATA "All Variables have been used
"
 2250 DATA "Missing bracket"
 2260 DATA "Only one divide allowed"
 2270 DATA "You can't divide by zero"
 2280 DATA "That variable already used"
 2290 DATA "Division not integer"
 2300 DATA "You can't end with an operat
or"
 2310 DATA "You can't enter that charact
er"
 2320 DATA "You pressed Return by mistak
e"
```

BEEBUG WORKSHOP

Tree Structures (Part 3)

by Mike Williams

As I said last time, I want to look at one further way of treating tree traversal. This method depends upon the language you use to write your program, and Beeb users are fortunate because BBC Basic, but not necessarily other versions, allows the use of a technique called recursion.

RECURSIVE TREE TRAVERSAL

As I stated in the first Workshop on tree structures, the essential definition of Inorder traversal is:

Traverse the left subtree
Visit the root
Traverse the right subtree

Suppose we want to write a procedure which will perform Inorder traversal starting at any node of the tree. Let us suppose we define a procedure called PROCinorder(root) for this purpose.

For this technique we do not have to worry about threaded trees (which we used last time), nor do we have to implement a stack for pointers as we did with our first approach to tree traversal. Our tree structure will be simple, with left and right-hand pointers to any subtrees, and pointer values of -1 when there is no subtree. Given this information we can define our procedure as follows:

```
DEF PROCinorder(root)
IF LLink(root)>-1 THEN
PROCinorder(LLink(root))
PRINT Data$(root)+" ";
IF RLink(root)>-1 THEN
PROCinorder(RLink(root))
ENDPROC
```

Believe it or not that is all the coding which is necessary. The oddity, if that's the right word, in this procedure definition is that that the procedure is defined (in part) in terms of itself. Now at first sight, this may seem impossible, but that is not the case. What the procedure says is "for as long as there is a left-hand link to a subtree follow this link and repeat the process with the sub-subtree, and so on". Now every binary tree is finite, so eventually we reach a left-hand link which contains a null pointer (a value of -1 in our case). When that happens, we move to the second line of the procedure definition and display the value of the data at that node. Then the procedure looks at the right-hand link.

Let us look at this another way. When the procedure is called the first time, the first IF statement in the procedure is executed. If the left-hand link exists, the procedure is called again, substituting this left-hand link for the original root. In effect we have a second incarnation of the procedure, and every time we follow another link down a level of the tree, in effect another incarnation of the procedure will come into play. Once the

end of these links is reached, Basic starts to climb back up the chain. How does it do this? - by creating a stack on which it automatically stores all the pointers needed to climb back up the tree.

In fact, Basic is doing automatically what we programmed ourselves before, either by implementing a stack or by using threads, but the coding is very much simpler. A program, Tree03a, implementing this method of Inorder tree traversal is listed here. For comparison, the time to repeat this traversal 100 times is just 7.52 seconds, compared with 22.00 seconds for our explicit stack method, and 13.47 seconds for our threaded tree approach. The unseen payoff is in the extra storage taken up by Basic to manage its stack.

PREORDER AND POSTORDER

In fact we can take the use of recursion a stage further, for it is just as simple to implement Preorder and Postorder traversal of a binary tree as it is to implement Inorder - it is merely the order of the three statements as at lines 1080 to 1100 which makes the difference.

To illustrate this I have extended listing Tree03a to include all three forms of tree traversal, and this appears as listing Tree03b. This program uses exactly the same data as all our previous tree programs, but it now includes a simple menu system allowing a choice of any of the three traversal techniques.

Note that because of the essential recursive nature of the traversal procedures, each has been given a separate 'pre-procedure'. This just ensures that the title line and terminating line appear once only - it is the data in the tree which is traversed recursively.

Recursion can be a powerful technique - in this case it considerably simplifies the coding - but unless adequately understood and used wisely it can lead to excessive use of memory and be a nightmare to debug, as I know to my cost!

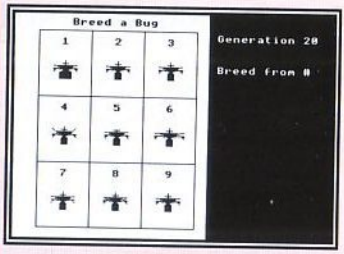
Next time, in a final look at tree structures, we'll see how data can be inserted and deleted in a tree structure so that the ordering of the data is preserved.

```
   10 REM Program Tree03a
   20 REM Version B 1.0
   30 REM Author  Mike Williams
   40 REM BEEBUG  Jan/Feb 1993
   50 REM program subject to copyright
   60 :
  100 ON ERROR REPORT:PRINT" at ";ERL:EN
D
  110 DIM Data$(100),LLink(100),RLink(10
0)
  120 130 MODE 3:PRINT'"Inorder Tree Tra
versal"
  130 PROCcreate_tree
  140 PRINT
  150 PROCinorder(0)
  160 END
  170 :
 1000 DEF PROCcreate_tree
 1010 LOCAL I%,N%:READ N%
 1020 FOR I%=0 TO N%-1
 1030 READ Data$(I%),LLink(I%),RLink(I%)
 1040 NEXT I%
 1050 ENDPROC
 1060 :
 1070 DEF PROCinorder(root)
 1080 IF LLink(root)>-1 THEN PROCinorder
(LLink(root))
 1090 PRINT Data$(root)+" ";
 1100 IF RLink(root)>-1 THEN PROCinorder
(RLink(root))
 1110 ENDPROC
 1120 :
 2000 DATA 9
 2010 DATA A, 1, 2,B, 3,-1,C, 4, 5
 2020 DATA D,-1,-1,E,-1, 6,F, 7, 8
 2030 DATA G,-1,-1,H,-1,-1,I,-1,-1
```

```
   10 REM Program Tree03b
   20 REM Version B 1.0
   30 REM Author  Mike Williams
   40 REM BEEBUG  Jan/Feb 1993
   50 REM program subject to copyright
   60 :
  100 ON ERROR REPORT:PRINT" at ";ERL:EN
D
  110 DIM Data$(100),LLink(100),RLink(10
0)
  120 end%=FALSE
  130 PROCcreate_tree
  140 :
  150 REPEAT
  160 MODE3:choice=FNmenu
  170 IF choice=1 THEN PROCpreorder1(0)
  180 IF choice=2 THEN PROCinorder1(0)
  190 IF choice=3 THEN PROCpostorder1(0)
  200 IF choice=4 THEN end%=TRUE
  210 UNTIL end%
  220 END
  230 :
 1000 DEF PROCcreate_tree
 1010 LOCAL I%,N%:READ N%
 1020 FOR I%=0 TO N%-1
 1030 READ Data$(I%),LLink(I%),RLink(I%)
 1040 NEXT I%
 1050 ENDPROC
 1060 :
 1070 DEF PROCpreorder1(root)
 1080 PRINT''"Preorder Tree Traversal"'
 1090 PROCpreorder(root)
 1100 PRINT''"Press any key to continue.
":G=GET
 1110 ENDPROC
 1120 :
 1130 DEF PROCpreorder(root)
 1140 PRINT Data$(root)+" ";
 1150 IF LLink(root)>-1 THEN PROCpreorde
r(LLink(root))
 1160 IF RLink(root)>-1 THEN PROCpreorde
r(RLink(root))
 1170 ENDPROC
 1180 :
 1190 DEF PROCinorder1(root)
 1200 PRINT''"Inorder Tree Traversal"'
 1210 PROCinorder(root)
 1220 PRINT''"Press any key to continue.
":G=GET
 1230 ENDPROC
 1240 :
 1250 DEF PROCinorder(root)
 1260 IF LLink(root)>-1 THEN PROCinorder
(LLink(root))
 1270 PRINT Data$(root)+" ";
 1280 IF RLink(root)>-1 THEN PROCinorder
(RLink(root))
 1290 ENDPROC
 1300 :
 1310 DEF PROCpostorder1(root)
 1320 PRINT''"Postorder Tree Traversal"'
 1330 PROCpostorder(root)
 1340 PRINT''"Press any key to continue.
":G=GET
 1350 ENDPROC
 1360 :
 1370 DEF PROCpostorder(root)
 1380 IF LLink(root)>-1 THEN PROCpostord
er(LLink(root))
 1390 IF RLink(root)>-1 THEN PROCpostord
er(RLink(root))
 1400 PRINT Data$(root)+" ";
 1410 ENDPROC
 1420 :
 1430 DEF FNmenu
 1440 LOCAL c%
 1450 PRINT"BINARY TREE DEMONSTRATION"'
 1460 PRINTTAB(5)"1. Preorder Traversal"
 1470 PRINTTAB(5)"2. Inorder Traversal"
 1480 PRINTTAB(5)"3. Postorder Traversal
"
 1490 PRINTTAB(5)"4. Exit"
 1500 PRINT'"Enter 1 - 4:";
 1510 REPEAT:c%=GET-48:UNTIL c%>0 AND c%
<5
 1520 =c%
 1530 :
 2000 DATA 9
 2010 DATA A, 1, 2,B, 3,-1,C, 4, 5
 2020 DATA D,-1,-1,E,-1, 6,F, 7, 8
 2030 DATA G,-1,-1,H,-1,-1,I,-1,-1
```

Best of BEEBUG

Applications I Disc



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

File Handling for All

on the BBC Micro and Acorn Archimedes

by David Spencer and Mike Williams

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

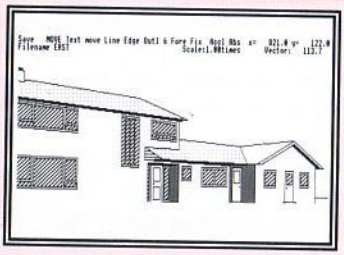
File Handling for All, written by highly experienced authors and programmers David Spencer and Mike Williams, offers 144 pages of text supported by many useful program listings. It is aimed at Basic programmers, beginners and advanced users, and anybody interested in File Handling and Databases on the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

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

The topics covered by the book include:

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

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



ASTAAD

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

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

	Stock Code	Price
ASTAAD (80 track DFS)	1407a	£5.95
Applications II (80 track DFS)	1411a	£4.00
Applications I Disc (40/80T DFS)	1404a	£4.00
General Utilities Disc (40/80T DFS)	1405a	£4.00
Arcade Games (40/80 track DFS)	PAG1a	£5.95
Board Games (40/80 track DFS)	PBG1a	£5.95

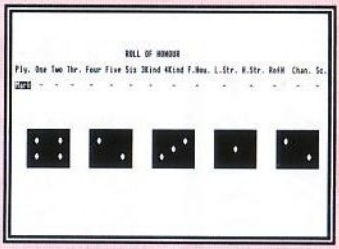
	Stock Code	Price
ASTAAD (3.5" ADFS)	1408a	£5.95
Applications II (3.5" ADFS)	1412a	£4.00
Applications I Disc (3.5" ADFS)	1409a	£4.00
General Utilities Disc (3.5" ADFS)	1413a	£4.00
Arcade Games (3.5" ADFS)	PAG2a	£5.95
Board Games (3.5" ADFS)	PBG2a	£5.95

All prices include VAT where appropriate. For p&p see Membership page.

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

Best of BEEBUG

Board Games



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

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

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

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

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

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

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

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

Applications II Disc

```
                                    14:39:34
AppsII    (009)        Owner
FS                     Option 00 (Off)
Dir. AppsII            Lib. Library1

3DLAND    UR/          F          UR/
FOREIGN   UR/          SHARES     UR/
TEMP      UR/
>HEX
AppsII    (009)        Owner
FS                     Option 00 (Off)
Dir. AppsII            Lib. Library1

3DLAND    FFFF3000 FFFF3000   005000
UR/       08:06:92 00DE4B
F         00003747 00003747   0005C8
UR/       17:06:92 006144
FOREIGN   FFFF6000 FFFF6000   002000
UR/       17:06:92 004660
SHARES    FFFF7C00 FFFF7C00   000400
UR/       17:06:92 003E24
TEMP      FFFF7C00 FFFF7C00   000400
UR/       17:06:92 005D92
>*SAVE CLOCK 7C00 8000 FFFF7C00 FFFF7C00
```

CROSSWORD EDITOR - for designing, editing and solving crosswords

MONTHLY DESK DIARY - a month-to-view calendar which can also be printed

3D LANDSCAPES - generates three dimensional landscapes

REAL TIME CLOCK - a real time digital alarm clock displayed on the screen

RUNNING FOUR TEMPERATURES - calibrates and plots up to four temperatures

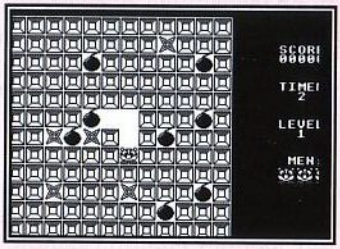
JULIA SETS - fascinating extensions of the Mandelbrot set

FOREIGN LANGUAGE TESTER - foreign character definer and language tester

SHARE INVESTOR - assists decision making when buying and selling shares

LABEL PROCESSOR - for designing and printing labels on Epson compatible printers

Arcade Games



GEORGE AND THE DRAGON - Rescue 'Hideous Hilda' from the flames of the dragon, but beware the flying arrows and the moving holes on the floor.

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

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

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

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

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

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

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

	Stock Code	Price		Stock Code	Price
File Handling for All Book	BK02b	£ 9.95			
File Handling for All Disc (40/80T DFS)	BK05a	£ 4.75	**File Handling for All** Disc (3.5" ADFS)	BK07a	£ 4.75
Joint Offer book and disc (40/80T DFS)	BK04b	**£ 11.95**	**Joint Offer book and disc** (3.5" ADFS)	BK06b	**£ 11.95**
Magscan (40 DFS)	0005a	£ 9.95	**Magscan Upgrade** (40 DFS)	0011a	£ 4.75
Magscan (80T DFS)	0006a	£ 9.95	**Magscan Upgrade** (80T DFS)	0010a	£ 4.75
Magscan (3.5" ADFS)	0007a	£ 9.95	**Magscan Upgrade** (3.5" ADFS)	1458a	£ 4.75

All prices include VAT where appropriate. For p&p see Membership page.

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

512 Forum

by Robin Burton

This month the Forum's again a bit of a mixture. I hope there's something for everyone, but I apologise in advance for the first item.

There's more software compatibility, a tip that might save a few of you a small but annoying bit of typing, and a source of very useful information on some of the 512's quirks, both for those who write programs and for the less technically inclined.

OLD CHESTNUTS

First, yet another repetition of facts which most Forum readers know. I hope no-one minds a bit of space for this; with luck it might save me time, paper and frustration (sorry to moan!).

The most up-to-date version of DOS which runs on the 512 is the Acorn issued DOS Plus 2.1. Don't write to me about DR DOS, MS-DOS, Windows or OS/2 (please!): the answer's 'NO'! Here are some more points on which I will probably not mention in the future.

You can't add anything to the 512 to give an EGA or VGA display, and even if you could your monitor couldn't handle it (I'll expand on PC graphics next month). You can't even have full colour CGA: your monitor couldn't handle that either without modification. You can't have more than 1024K of RAM, as that's the limit the 512's 80186 processor can address. You can't have more than one hard drive (at a time), but it doesn't necessarily have to be drive C: once the system has been started. Finally, you can use a tracker-ball instead of a mouse, but you cannot have a games port or a joystick.

These points and more have been covered in 512 Forum previously, so please don't write to me about them: it'd be better to order Beebug back issues instead.

NO RTC?

This tip won't save hours, but it might be useful to some model B and B+ users. Since these machines have no battery backed CMOS RAM there's no time or date. Users must therefore manually supply these every time they boot the 512, or do without knowing when files were created or updated. For backups particularly this information can be critical, so its absence isn't a good idea.

PMS's Genie Watch fills the gap permanently, although I don't know if it's still available. If you're interested, details are on page 366 of the 512 Technical Guide (which is still available from Dabs Press, as is the 512 User Guide). By the way, in case you hadn't noticed, Dabs are currently supplying their two Shareware collections for £25.00 the pair, each containing five discs. Previously these cost £30.00 for each set.

Recently I've been using an XT with no battery backed clock. Since I have to re-boot a lot, having to re-enter the date and time again and again is most tiresome. So I thought of a solution which means I never enter the date more than once. All you need is a small change to AUTOEXEC.BAT and two very short, simple files, the first of which creates the second for you.

Here's how to set up the first file, I called mine NEWDATE.BAT, so whenever I do need to change the date I just type NEWDATE. The file should be in the root directory of your boot disc, be it hard or floppy, since the output file is

used by AUTOEXEC.BAT. Enter it via an editor, or more easily, as it's only two lines, use:

```
COPY CON NEWDATE.BAT
```

This produces an empty screen line with a cursor but no prompt. That's fine, just carry on. Enter these two lines, pressing Return after each.

```
COPY CON TODAY
DATE <TODAY
```

Next, press Ctrl-Z, then Return again to close the file. You now have a new batch file which, when run, will open a file called TODAY, presenting you with a blank line as described above. You then enter the date in exactly the same format as you do at the standard date prompt, for example '7-6-92' (how many of you have been typing in leading zeros and the century too?). Follow that by Return, Ctrl-Z, Return, then the file is written and the system date is (re)set immediately as well.

Next amend AUTOEXEC.BAT. Since this is an update, edit your existing file, as the alternative, 'COPY CON', requires re-entry of the entire file. Remember you may need to use 'FSET' (see last month's Forum) before you can update the file if it's read-only or if the system bit is set. Often forgotten is 'SDIR', which shows file attributes. Use it instead of DIR to see if FSET is needed.

The line to change is the one that contains 'DATE' (or if you've previously removed it, add it now.) Amend it to match the second line of NEWDATE.BAT, that is add a space followed by '<TODAY' to the command on the same line. Resave the file, then reset its attributes with FSET to suit your preferences.

Note that, having amended AUTOEXEC.BAT, you must run NEWDATE before you switch the machine off again. If you don't, when you next reboot you'll get a 'file not found' message because 'TODAY' won't exist (because you didn't create it - this is a 'one-off' situation of course).

With the changes in place, when you reboot, the date stored in 'TODAY' is piped into the date command automatically, so you no longer have to enter it. Of course each day the date needs changing, but all you do is run 'NEWDATE' (once) which corrects the system's internal date and the one stored in 'TODAY'. The date is then set for the rest of the day and you don't need to re-enter it no matter how often you reboot.

This isn't much use for the time, but it's certainly better than having no file date or time stamps which, judging from some discs I've seen, isn't an uncommon choice. Remember too, that the same technique can be used to 'automate' any program that expects keyboard entry, including command line parameters, when the data is always the same.

PD INFORMATION

David Harper has provided a good deal of valuable information for 512 Forum as regular readers know, most notably all the research for the recent series on GEM. What you may not know, however, is that he has now produced a 512 information disc which he's released into the public domain. The disc contains a number of information files, in text, each on a specific topic, which you can print or which you can keep in your system as an ad-hoc reference.

Necessarily some, though by no means all the information is of a very technical nature, so if you write your own 512 machine code routines you may well find the solutions to numerous puzzles on this disc. However, even if you don't write 512 programs don't be put off. Even for the simply curious there are answers to some of the questions I'm asked from time to time and I'm sure most users will find a great deal of interest.

In particular there's a very interesting and entertaining demonstration of how to really liven up menu screens and the like, including adding colour, changing mode, hiding text and so on, using nothing more than an editor or word processor capable of handling non-ASCII characters.

I've said before I don't get time to 'poke around' inside the system much as I'd like to, but I do know from experience how much work it takes. David has clearly spent a considerable number of hours putting together the information on this disc. The result is an excellent and unique source of reference. If you were paying commercial rates for it I'd suggest you couldn't afford it; instead, as it's PD, I'd say that no 512 enthusiast can afford not to have it.

See Alan Blundell's PD column in Vol.10 No.9, or his advertisement in the personal ads page of BEEBUG, to obtain a copy.

SENSIBLE QUESTIONS

I recently had a letter from David MacGraw that queried a number of points which users new to DOS might not know.

He asked if there was a DOS equivalent to the BBC program ADU, once produced by Pineapple, which is a menu-driven front-end for ADFS. Yes, David, the answer is there are quite a few, and they are naturally much more sophisticated than BBC programs too. There is of course the file manager which was provided on issue disc one of DOS Plus 1.2 (only). If you have it this will give some idea, but it's neither the best nor the most reliable example of the type.

PC-Tools is close to the ultimate, but Xtree and Norton Commander (1.01 works) are two popular, easily obtained alternatives. There are also numerous similar programs in shareware too, so choosing which to try is really the main problem. It's personal taste, I don't much like XTree and would recommend PC-Tools so long as you have sufficient disc and memory space (see Vol.11 No.4).

Another query was whether there's a program to blank the 512's screen on demand, to hide sensitive data from prying eyes. There is, but modesty (almost) prevents me from telling you. The only program I know of is 'LOCKWORD' on Essential Software's Miscellaneous disc 2. (PC screen blankers WON'T work, so don't waste time trying.)

David also asked if there's any way the 512 can handle sound in PC programs, but this time the answer's 'NO'. Ultimately almost anything's possible of course, but this would take a great deal of complicated code and it's just not practical.

MORE APPLICATIONS

Now for more software compatibility. I've used quotes in the same way as I did in issue 4. Unfortunately the version number is missing for many of these, as is the publisher, but I present the information in case it's useful. After all, in some cases there may be only one version. Items in brackets are comments I've added where I can.

Bear in mind too (we haven't started yet) that many graphics applications, such as DTP, will expect a minimum of an EGA if not a VGA display, plus appropriate drivers. Forget these for the 512. I'll expand on PC graphics cards and drivers next month, but for now let's just look at the software.

Member I.Cook sent a brief note to say that Mini Office Personell version 3.1 runs on the 512. (Should that be 'personal'? I don't know, but you will if you come across it.)

Another member, who wishes to remain anonymous supplied the following.

Lotus 1-2-3 release 2.3 works with or without a mouse. There are some problems with graphs created in the 512, although graphs created on a PC and transferred to the 512 are OK. Also, of course, it expects a VGA display, so the pure graphics functions don't work.

Mind-Reader (a shareware word processor also supplied as part of the Dabs 512 Shareware collection) works too, but as users have found, it gives problems (a crash) if you try to change the drive, path or name under which a file is saved. Avoid renaming files in the program (copy or rename them in DOS beforehand) and it's OK.

Word-Fugue (shareware) works "in a similar manner" presumably referring to paths and filenames, but requires a second copy of COMMAND.COM to be loaded (or better, run FIXEXE).

PC-Outline "works brilliantly, if only it had a word-count, spelling checker and mouse control I'd use it in preference to anything else", says my correspondent. (Shareware. How about letting me have version numbers? 1.08 is in the Dabs collection. See also issue 4.)

First Choice is an integrated package with Comms (which won't work), a spreadsheet, a flatfile database plus a word processor with spelling checker. The member warns that text "files 'MOVE'd to the BBC contain lots of spurious characters". (This is true of numerous DOS word processors, but it's also true for InterWord files imported into the 512, and they're not perfect even if they're spooled first.)

Both PC-Type and PC-Word work, but as this member doesn't use them much he refrains from further comment. (PC-Type II requires real CGA for graphing, I don't know about earlier versions, and a hard disc is needed for version IV.)

He also confirms that Mini Office (above) plus MoneyBox, both by Database Software (remember Micro User?) also work.

Dbase III Plus "works a treat".

"Wampum, PC-File and PCBase all work OK". (These are shareware databases. They're not for the 'dabbler', they're serious programs. Wampum needs EGA/VGA for graphics.)

Press, "a sort of mini-DTP package", works, but although printing's fine "some of the pictures can't be seen on screen".

Print Partner, a similar program "is OK too". My correspondent says, "Both of the above are poster and banner makers really, but they do work." (Print partner is shareware and ideally requires 'real' CGA. Expect screen trouble with many DTP programs.)

Flodraw is a flowcharting system. It "works quite well with few problems". (Shareware, also included in the Dabs collection.)

"Fontasy works, but a little slowly".

XTree "works a treat" (as mentioned above).

NEXT ISSUE

Well, that's the 45th Forum completed. I've a bit more (yet more is always welcome, but please - complete info) on applications compatibility and incompatibility, but I mustn't spoil you, must I?

I also have the concluding episode in the directory extension saga, thanks to Philip Draper, but all these, along with PC graphics standards and methods, must wait until next time.

Wordwise User's Notebook:
Cutting the Keys

by Chris Robbins

A while back I attempted an explanation of some of the minor mysteries associated with the black art of programming the Beeb's function keys for use with Wordwise (WW) and Wordwise Plus (WW+) - see BEEBUG Vol.11 No.4.

Although not as all-powerful as the spells to be found in the average D&D Wizard's spell book, using the function keys, as I illustrated in that earlier article, can prove remarkably effective against finger gremlins, both chronic and spontaneous brain fatigue, spots before the eyes, hexadecimalization, and many other computing complaints. They can even extend your Wordwise Wizardry; that is, if you know what to program them with.

For instance, the example I gave of an improved version of the WW operation 'Save marked text' packed the equivalent of 18 separate editing operations into one function key, but in order to do so it required the setting up of an arcane string of some 50 magical symbols.

The problem is that the sequences of characters required to implement operations are both highly forgettable and easily mistyped; table 1 gives the key strings and their associated key functions.

Using this table 'manually', as it were, it's possible to devise all manner of complicated and esoteric sequences of operations that can be invoked at the mere touch of a key.

But, wouldn't it be nice to have an expert, user friendly, WW+ 'locksmith' to 'cut' these magical symbols automatically into the keys you require, instead? Well, that's just what the segment program listed here for WW+ aims to be.

Key string	Key Function
!!<Space>	f0
!!!	f1
!!"	f2
!!#	f3
!!$	f4
!!%	f5
!!&	f6
!!'	f7
!!(	f8
!!)	f9
!!,	Cursor L
!!-	Cursor R
!!.	Cursor D
!!/	Cursor U
!I	Tab
!M	Return
![	Escape
!!!\	Ctrl-Cursor L
!!!}	Ctrl-Cursor R
!!!^	Ctrl-Cursor D
!!!-	Ctrl-Cursor U
!!!L	Shift-Cursor L
!!!M	Shift-Cursor R
!!!N	Shift-Cursor D
!!!O	Shift-Cursor U
!A	Ctrl-A
!S	Ctrl-S
!D	Ctrl-D

Table 1. Note - <Space> denotes the space character.

PROGRAM FILES

The program comes in two parts, the executable code (saved as *DEFKEYS* in directory P) to be loaded into segment 0, and its associated list of key strings (saved as *CLIST* also in directory P) to be loaded into segment 1.

This is best done automatically by setting up an EXEC file as follows:

```
*|DEFKEYS
*|EXEC file to set up and run the WW+
*|segment program to define soft keys
*|
*WORD.
:SELECT SEGMENT 0
*|Load segment program
:LOAD TEXT "P.DEFKEYS"
:SELECT SEGMENT 1
*|Load key strings
:LOAD TEXT "P.CLIST"
*|Invoke program
:DOLINE"|GSEG0|W'
<Return>
```

and then EXECuting it to load and invoke the program.

The last line in the file, <Return>, signifies a blank line that's used to avoid an additional key depression, whilst the lines beginning *| are simply explanatory comments ignored by the system.

USING THE PROGRAM

EXECuting the EXEC file at command level e.g. from Basic or WW+ Main Menu, loads the relevant segments and invokes the program, displaying a large heading and a prompt for the number of a key (in the range 0-15) to be set up.

Checks prevent any number outside the valid range being accepted.

Having entered a valid key number, the key function menu is displayed, from which any (and all) of the key functions listed above can be selected and automatically included in a definition. Selection is simply a matter of moving a highlight bar using the Cursor Up and Cursor Down keys to the required function and then pressing Return.

The number of the key being set up, together with directions on how to use the menu are displayed at the top of the screen.

STRINGS

Strings (e.g. a file name) can be incorporated into key definitions by selecting the *String* option. Enter the required characters at the flashing prompt which appears at the foot of the screen, and press Return to terminate the string.

The program makes no assumptions about strings; that is to say you can enter whatever you like, including string quotes (") as necessary (since it doesn't assume that they're wanted). Note - if you want leading spaces in a key definition, then the key definition will have to be delimited by string quotes.

TERMINATING A KEY DEFINITION

A key definition is held in a string variable whilst it's being built up. Selecting *End definition* from the menu terminates the definition process, transfers the completed key description to the main text area, and returns to the key number prompt screen, at which point a new key can be selected.

However, bear in mind that the program makes no attempt to check that a key has already been set up in the main text area, nor does it clear the main text area prior to execution; this latter allows multiple key definitions to be assembled, e.g. from earlier sets of definitions.

SAVING KEY DEFINITIONS

Pressing Escape from either the key number prompt screen or the key function menu transfers control to the WW+ main menu. At this point the definitions in the main text area can be examined, edited in

the normal way of text, or saved under some appropriate file name (using option 1) from the main menu) for subsequent EXECution.

FINAL THOUGHTS

Although devised with WW+ in mind, the program can also be used to generate labour/time/error saving mechanisms for use in other applications; Computer Concepts' latter day word processor, InterWord, for instance, where leaping between menus, jumping between packages, selecting functions, entering printer Escape sequences etc, can prove a mite irksome.

For example, I tend to make a lot of use of special printer effects, e.g. italics, proportional spacing, double width/height, colour etc., all of which consumes a deal of finger and brain energy. So I've used the program to set up keys to do all the brain fatiguing work for me, and automatically embed the requisite printer control commands in Interword text.

True I still have to press the odd key or two to produce the words and generate the special effects I want - I haven't found a way around that yet. But when I do I'll let you know. Meanwhile I'll keep on cutting magical keys!

```
REM DEFKEYS
REM WW+ Function Key definition program
REM
REM The program simplifies the process
REM of setting up complex strings of
REM WW+ (and other) key commands.

GOTO main-program

REM PROCS ******************************

REM Clears previous highlight
.clear
  VDU 31,1,T%
  VDU 156,135
ENDPROC

REM Moves highlight bar down list
.down
  PROCclear
  C%=C%+1
  T%=C%
  IF C%<=14 THEN GOTO down-end
  IF C%<=29 THEN GOTO down-right
  C%=0
  T%=0
  PROCleft-window
  GOTO down-end
.down-right
  IF S$<>"R" THEN PROCright-window
  T%=T%-15
.down-end
ENDPROC

REM Moves highlight bar up list
.up
  PROCclear
  IF C%<>0 THEN GOTO up-decrement
  C%=29
  T%=14
  PROCright-window
  GOTO up-end
.up-decrement
  C%=C%-1
  T%=C%
  IF C%<>14 THEN GOTO up-right
  PROCleft-window
  GOTO up-end
.up-right
  IF S$="R" THEN T%=T%-15
.up-end
ENDPROC

REM Highlights selected key command
.show
  P%=C%
  IF P%>14 THEN P%=P%-15
REM Move cursor to selected command
  VDU 31,0,P%
REM White background, blue text
  VDU 131,157,132
```

```
  VDU 31,18,P%
  VDU 156,135
ENDPROC

REM Sets up RH window
.right-window
  S$="R"
  VDU28,20,23,39,7
  VDU31,0,0
ENDPROC

REM Sets up LH window
.left-window
  S$="L"
  VDU28,0,23,19,7
REM Position cursor at start of list
  VDU31,0,0
ENDPROC

REM Message in T$
.large-msg
  REPEAT
    VDU131,141
    PRINT T$
  TIMES 2
ENDPROC

.space-line
  PRINT
  VDU134
ENDPROC

REM Transfers control to WW+ Main Menu
.exit
  SELECT TEXT
REM Restore cursor control
  *FX4,0
REM Flush keyboard buffer
  *FX21,0
REM Restore ESC key function
  *FX229,0
  *WORD.
  END
ENDPROC

REM Gets a word from Key Command list
.get-word
  W$=""
  REPEAT
   C$=GCT$
   IF C$<>"*" THEN W$=W$+C$
  UNTIL C$="*"
ENDPROC

REM End of PROCs **********************

.main-program

SELECT TEXT
REM Set default window
VDU26
CLS
REM Set ESC key to return ASCII value
*FX229,1
T$="       Function Key Definer"
PROClarge-msg
PRINT
PROCspace-line

.inkey
VDU31,0,3
PRINT "Enter Key number (0-15) ";

REM Get-key-number
K$=""
K%=0
REPEAT
  REPEAT
    R%=GET
  UNTIL (R%>=48 AND R%<=57) OR R%=13 OR
R%=27
  IF R%=13 OR R%=27 THEN GOTO inkey-exit
  D$=CHR$(R%)
  PRINT D$;
  K$=K$+D$
  K%=VAL(K$)
.inkey-exit
UNTIL R%=13 OR R%=27 OR K%=0 OR K%>9
IF R%=27 THEN PROCexit
IF K%>15 OR LEN(K$)=0 THEN GOTO inkey


REM Display Key Command Menu
```

```
REM Turn cursor off
VDU23;11,0;0;0;0
REM Disable cursor editing
*FX4,1
CLS
T$="                    "+"KEY"+K$
PROClarge-msg
PROCspace-line
PRINT "Use Cursor keys+RETURN for
Selection"
PROCspace-line
PRINT "ESC For WW+ Main Menu"
PRINT

REM Display 1st list of key commands

PROCleft-window

SELECT SEGMENT 1
CURSOR TOP
P%=0
REPEAT
  PROCget-word
  PROCget-word
  PRINT "    "+W$
  P%=P%+1
  CURSOR AT 0
  CURSOR DOWN
UNTIL P%=15

REM Display 2nd list of key commands

PROCright-window
REPEAT
  PROCget-word
  PROCget-word
  PRINT "    "+W$
  P%=P%+1
  CURSOR AT 0
  CURSOR DOWN
UNTIL P%=28

PRINT "    String"

PRINT "    End Definition"
```

```
REM Set up key number
K$="KEY"+K$

.select-command

PROCleft-window
T%=0
C%=0
S$="L"
REM Highlight 1st entry in list
PROCshow
.get-code
REPEAT
REM Flush keyboard buffer
  *FX21,0
  REPEAT
    R%=GET
   UNTIL R%=138 OR R%=139 OR R%=13 OR
R%=27
  REM If ESC then exit
  IF R%=27 THEN PROCexit

  IF R%=138 THEN PROCdown
  IF R%=139 THEN PROCup
  IF R%<>13 THEN PROCshow
UNTIL R%=13

IF C%=29 THEN GOTO end-definition
IF C%=28 THEN GOTO enter-string
CURSOR TOP
IF C%<>0 THEN CURSOR DOWN C%
PROCget-word

REM Add code sequence to key definition
K$=K$+W$
GOTO get-code

.end-definition
REM End current key definition
REM and transfer to main text area
SELECT TEXT
TYPE CHR$(13)
TYPE K$
GOTO main-program

REM String entry into key definition
```

Continued on page 50

JobLog (Part 2)

Jeff Gorman concludes his time manager.

Last month's listing produced code for a dated 'To do' list. This month we add a heading showing the base date and the event (or data) heading and the deadline for an event (if deadlined). It increases the range of date functions and enables entry of times of day and/or memos. 'Extensions' are activated, and deletion and modification of entries are enabled together with sorting, printouts in three formats and a quick return to the Principal Index from analysis levels deeper than level one. Escape now takes the user back to the previous stage, including the final QUIT.

Type in the program given here. Since combining the two listings involves overwriting certain lines, take care to enter the line numbers exactly. Save as *$.JobLogTwo*, and prepare one or two backups.

Until the two parts are combined, it will not be possible to verify JobLogTwo, other than by line-by-line comparison of the listings. It is unwise therefore, to proceed with the next operation until JobLogOne is working properly. Once you are certain that this is the case, it will be easier to identify typing errors in JobLogTwo.

Likewise make sure you have at least one backup of JobLogOne since a mix-up will be disastrous if no backups are available.

Spool out a copy of JobLog2 to disc using *SPOOL filename. Then load JobLog1 and *EXEC the spooled copy of JobLog2 to form the complete program. Save it as *$.JobLog*.

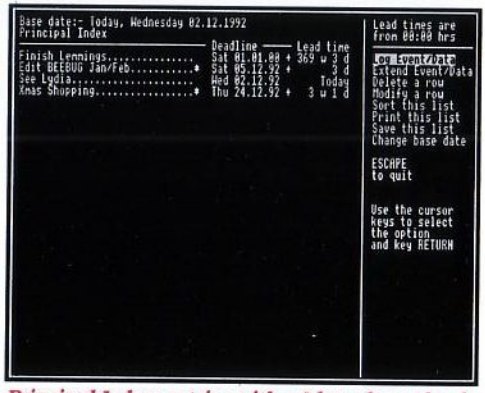
If all is well, the !Boot file described last month will run the complete version of JobLog provided that !Boot is changed to:

```
CHAIN "$.JobLog"
```

Create five sub-directories by entering, from the root ($) directory:

```
*Cdir Level1
```

and repeat, changing the level number until *Cdir Level5 is reached.



*Principal Index: entries with a * have lower levels*

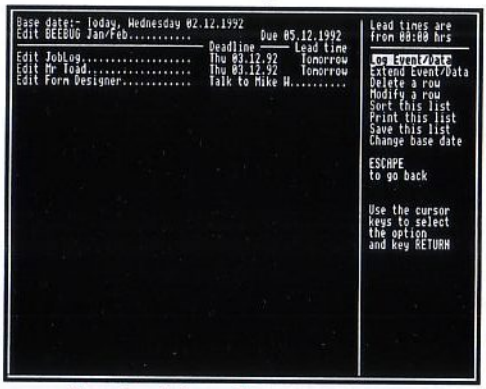
USING JOBLOG

Extending a listed item puts an asterisk at the end of its title, opens a new file in a sub-directory (e.g. Level 1) and offers a new screen for composing a new list.

Any line can be modified from the 'Modify' option and to save typing effort, there is an option to change either an item's title or the date/memo side of the screen. If an asterisked item is modified by changing its title, its file will be automatically renamed.

Similarly, deleting such an item also deletes its file from the disc. To delete a

series of related extensions, 'Extend' down to the deepest level, delete the item and use Escape to work backwards up the list, otherwise you will be left with unwanted files cluttering the disc.



Next level down for BEEBUG entry

All changes are noted and when necessary, files are saved before certain commands are implemented.

'Sort' will rearrange a list by order of date, with an alphabetically sorted series of Memos below. Times of day (24 hour system), e.g. appointment times, are sorted into the natural order and placed at the foot of the list.

Ensure that the printer is on-line before booting JobLog. Each list can be printed separately, or a number of lists can be printed one after another. JobLog keeps a tally of the number of printed lines and provides automatic form feeds on the basis that no list should overflow onto another sheet. Do not operate the printer's form feed control button since this will dislocate the line tally. The prompt 'Using cut sheets?' enables single sheets to be catered for. Since the formatting codes will be lost, do not switch off the printer when changing sheets. Instead, insert a new sheet and adjust until the print head is at its normal place at the head of the sheet. Line 1240 contains the variable *pLen%* which is currently set at 66 lines for 11" paper. For A4 paper, change this to 70, or 72 for 12" paper or 40 for 'Organiser' size. Variable *tearOff%* represents the top margin for those printers which leave a space between the tear-off edge and the printer head. The bottom margin is represented by *botM%*.

Lines 80 and 90 ensure that JobLog will also run on the Archimedes and related computers though it will only run on RISC OS 2. As we said last month, JobLog is an open ended organiser and you really need to explore its functions to find out just how much help it can be to you.

```
   10 REM Program JobLogTwo
   20 REM Version B 1.0
   30 REM Author  Jeff Gorman
   40 REM BEEBUG  Jan/Feb 1993
   50 REM Program subject to copyright
  550 PRINT "Enter "typ$" date"'''(dd.mm
.yyyy)":IF ok PRINT'''"ESCAPE"'"to step
back"
  670 end=0:IF escape THEN *fx 229,1
  680 IF NOT escape THEN *fx 200,1
  730 IF G%=27 esc=TRUE:end=TRUE:*fx229
  740 IF G%=13 AND memo AND a$="":end=TR
UE:*fx229
 1440 hndl%=OPENUP("$."+fnme$):PROCheadi
ng
 1650 IF mod same=FNsameText:PRINT
 1670 IF extend depth$=MID$(task$(rw%),3
6) ELSE depth$="0"
 1680 IF same PROCrhColumn(rw%):ENDPROC
 1710 PROCwnd("mid"):IF star AND mod PRI
NT TAB(30,rw%)"*";
 1740 IF esc AND row%=0 VDU7:PROCgetTask
(rw%)
 1750 IF esc AND mod AND NOT same replac
e$=MID$(task$(rw%),4,30) ELSE IF esc rep
```

```
lace$=STRING$(30," ")
 1760 IF esc PROCreplace(replace$,rw%):E
NDPROC
 1820 IF NOT prnt VDU31,0,rw%
 1840 st=(MID$(rw$,34,1)="*"):spc%=2
 1850 IF st PRINT "*";:spc%=spc%+prnt
 1860 memo=(MID$(rw$,35,1)="@")
 1870 m$=MID$(rw$,37,24):t$=LEFT$(m$,1):
p$=MID$(m$,3,1):punct=((ASCp$>31)AND(ASC
p$<48))OR(ASCp$=58)
 1880 time$=LEFT$(m$,2)+MID$(m$,4,2)
 1890 IF memo PRINT SPC(3+prnt+st);
 1900 IF memo AND NOTprnt PRINT m$; ELSE
 IF memo PRINT m$
 1910 IF memo AND ASC(t$)>47 AND ASC(t$)
<58 AND punct sort%(rw%)=1E9+EVAL(time$)
:ENDPROC
 1920 IF memo sort%(rw%)=1E7+((100*(ASC(
MID$(rw$,4,1))AND &DF))+(ASC(MID$(rw$,5,
1))AND &DF)):ENDPROC
 2310 predMenu=0
 2320 IF esc AND mod replace$=MID$(task$
(rw%),4,30) ELSE IF esc replace$=STRING$
(59," ")
 2330 IF esc PROCwnd("mid"):VDU31,0,rw%-
1:PROCreplace(replace$,rw%):PROCmenu:END
PROC
 2350 IF choice%=1 PROCintvl(rw%)
 2360 IF choice%=2 PROCmemo(rw%)
 2370 IF choice%=3 skip=TRUE:change=TRUE
:task$(rw%)=FNbuildRow(MID$(task$(rw%),3
7,24)):sort%(rw%)=1E7+((100*(ASC(MID$(ta
sk$(rw%),4,1))AND &DF))+(ASC(MID$(task$(
rw%),5,1))AND &DF))
 2440 IF escape THEN *fx 229,1
 2450 IF NOT escape THEN *fx 200,1
 2690 IF predMenu num%=(2+((NOTsame AND
mod)*-1)):PROCgetList(10,num%)
 2700 IF intvlMenu num%=(2+(index OR noD
lne)):PROCgetList(20,num%)
 2710 IF printMenu num%=2:PROCgetList(30
,num%)
 2820 UNTIL z%=n%:IF escape PRINT'"ESCAP
E"'"to "m$;
 2970 :
 2980 DEF FNyn:yn$=FNip(1,"YNyn",-1,0,0)
:=(CHR$(ASC(yn$) AND &DF))
 2990 :
 3000 DEF PROCretrace:full=0
 3010 IF change PROCwriteData(fnme$(leve
l%))
 3020 level%=level%+(level%>0)
 3030 IF opt%=8 level%=0
 3040 index=(level%=0):extend=NOTindex
 3050 PROCreadData:ENDPROC
 3060 :
 3080 PROCwnd("blank"):PROCwnd("mid"):to
pMenu=TRUE:*fx200,1
 3130 IF opt%=1 PROCextend
 3140 IF opt%=2 PROCdelete
 3150 IF opt%=3 PROCmodify
 3160 IF opt%=4 PROCsort(row%)
 3170 IF opt%=5 PROCprintout
 3190 IF opt%=7 PROCgetStrtDt:PROCmenu
 3200 IF opt%=8 PROCretrace
 3320 DEF PROCgetStrtDt:PROCwnd("blank")
 3330 strtDate$=FNgetDate("base")
 3340 IF esc PROCmenu:ENDPROC
 3350 day% =FNsplitDate(strtDate$,1,2)
 3360 mth% =FNsplitDate(strtDate$,4,2)
 3370 year%=FNsplitDate(strtDate$,7,4):P
ROCgetDatum(STR$day%,STR$mth%,STR$year%)
:IF change PROCwriteData(fnme$)
 3380 PROCreadData:ENDPROC
 3390 :
 3430 PRINT'"Use the cursor"'"keys to se
lect"'"the "rubric$'"and key RETURN"
 3460 DEF PROCquit:IF change PROCwriteDa
ta(fnme$(level%))
 3470 PROCwnd("blank"):PRINT"Program end
ed":PRINT '"Key f4 to re-run":*CLOSE
 3480 PROCcursor(1):END
 3490 :
 3500 DEF PROCreplace(rplc$,r%):VDU31,0,
r%:PRINT rplc$;:IF NOT(mod OR delete) ro
w%=row%-1
 3510 PROCmenu:ENDPROC
 3520 :
 3530 DEF FNsameText:PROCwnd("blank")
 3540 PRINT"Change text (Y/N) ?"
 3550 same=(FNyn="N"):IF same event$=MID
$(task$(rw%),4,30)
 3560 =same
 3570 :
```

```
 3590 IF esc predMenu=TRUE:PROCrhColumn(
rw%):ENDPROC
 3750 :
 3760 DEF PROCheading
 3770 PROCcursor(0):ex%=0
 3780 IF NOT prnt VDU28,0,1,58,0,12
 3790 IF datumDt$=todayDt$ base$="Today,
 " ELSE base$="New base, "
 3800 IF prnt VDU2:*fx3,10
 3810 PRINT "Base date:- "base$+name$ SP
C(1)datumDt$
 3820 IF prnt THEN *fx3,10
 3830 noDlne=(MID$(title$(level%),35,1)=
"@"):PRINT MID$(title$(level%),4,30) SPC
(10+prnt);
 3840 IF NOTnoDlne PRINT "  Due " MID$(t
itle$(level%),37,10);
 3850 IF prnt THEN *fx3,0
 3860 IF prnt VDU2,1,27,1,70,3:*fx3,10
 3870 IF prnt numLines%=numLines%+3:PRIN
T'STRING$(56,"-"):*fx3,0
 3880 ENDPROC
 3890 :
 3900 DEF PROCintvl(rw%):memoMrk$=" ":s$
="+":PROCwnd("blank"):
 3902 PROCselectBase:IF esc AND mod ENDP
ROC ELSE IF esc PROCrhColumn(rw%):ENDPRO
C
 3910 IF mod PROCprintDt(select%) ELSE P
ROCprintDt(rw%)
 3920 IF mod AND extend extndDln$=exd$+e
xm$+exyr$
 3930 ENDPROC
 3940 :
 3950 DEF PROCmemo(rw%)
 3960 memoMrk$="@":PROCwnd("blank")
 3970 PRINT"Enter the time/memo";
 3980 PROCwnd("mid"):VDU31,33,rw%
 3990 memo$=FNip(24,let$+no$,0,-1,-1)
 4000 IF esc PROCrhColumn(rw%):ENDPROC
 4010 t$=LEFT$(memo$,1):time$=LEFT$(memo
$,2)+MID$(memo$,4,2):p$=MID$(memo$,3,1):
punct=((ASCp$>31)AND(ASCp$<48))OR(ASCp$=
58)
 4020 IF ASC(t$)>47 AND ASC(t$)<58 AND p
unct sort%(rw%)=1E9+EVAL(time$) ELSE sor
t%(rw%)=1E7+((100*(ASC(MID$(event$,3,1))
AND &DF))+(ASC(MID$(event$,4,1))AND &DF)
)
 4030 dtData$=FNpad(24,memo$,".",0)
 4040 task$(rw%)=FNbuildRow(dtData$)
 4050 new=0:ENDPROC
 4060 :
 4070 DEF PROCselectBase
 4080 intvlMenu=TRUE:PROCwnd("blank")
 4090 ch%=FNmarker(2,-1):intvlMenu=0
 4100 IF esc ENDPROC
 4110 IF ch%=1 PROCnewDate
 4120 IF ch%=2 tempDtID%=FNdayID(VAL(LEF
T$(extndDln$,2)),VAL(MID$(extndDln$,4,2)
),VAL(RIGHT$(extndDln$,4)))
 4130 PROCgetIntvl(yr%):PROCwnd("mid")
 4140 ENDPROC
 4150 :
 4160 DEF PROCnewDate
 4170 PRINT"Calculate from: "
 4180 newDate$=FNgetDate("new")
 4190 IF esc AND mod PROCrhColumn(select
%):ENDPROC ELSE IF esc PROCrhColumn(rw%)
:ENDPROC
 4200 day%=FNsplitDate(newDate$,1,2)
 4210 mon%=FNsplitDate(newDate$,4,2)
 4220 yr% =FNsplitDate(newDate$,7,4)
 4230 tempDtID%=FNdayID(day%,mon%,yr%)
 4240 ENDPROC
 4250 :
 4260 DEF PROCgetIntvl(yr%):PROCwnd("bla
nk")
 4270 PRINT "Enter interval"
 4280 IF ch%=0 tempDtID%=strtDtID%
 4290 IF ch%=0 PRINT' "from base date"
 4300 IF ch%=1 PRINT' "from "newDate$
 4310 IF ch%=2 PRINT' "from "extndDln$
 4320 PRINT'"<+> = ahead"'"<-> = earlier
"':s$=FNip(1,"+-",TRUE,-1,0)
 4330 IF esc PROCselectBase:ENDPROC
 4340 PRINT'"Weeks"'"Days"':PRINT TAB(7,
8);:w$=FNip(3,no$,0,-1,0)
 4350 IF esc PROCrhColumn(rw%):ENDPROC
 4360 PRINT TAB(7,9);
 4370 d$=FNip(1,no$,0,-1,0):IF VALd$ >6
VDU7:PROCgetIntvl(yr%):ENDPROC
 4380 IF esc PROCgetIntvl(yr%):ENDPROC
 4390 intvl%=(VALw$*7)+VALd$
```

```
4400 IF s$="-" intvl%=intvl%*-1
4410 execID%=tempDtID%+intvl%
4420 execName$=FNdayName(execID% MOD 7)
4430 execDayNo%=FNexecDayID(execID%):IF
execDayNo%<=0 VDU7:PROCgetIntvl(yr%):EN
DPROC
4440 execMonth$=FNexecMonth(execDayNo%)
4450 execDate$=FNexecDay(execDayNo%,exe
cMonth$,execYr%)
4460 daysToExec%=execID%-strtDtID%
4470 sort%(rw%)=execID%:ENDPROC
4480 :
4490 DEF FNexecDayID(execID%)
4500 yr%=1990:REPEAT:yr%=yr%+1:a%=FNday
ID(01,01,yr%):UNTIL a%>execID%:execYr%=y
r%-1:=execID%-FNdayID(31,12,execYr%-1)
4510 :
4520 DEF PROCprintDt(rw%)
4530 exd$=FNpad(2,execDate$,"0",-1)+"."
4540 exm$=FNpad(2,execMonth$,"0",-1)+"."
":exyr$=STR$(execYr%)
4550 task$(rw%)=FNbuildRow(exd$+exm$+ex
yr$):PROCwnd("mid"):VDU31,33,rw%
4560 PROCprintRHS(daysToExec%,execID%,t
ask$(rw%),rw%):ENDPROC
4570 :
4580 DEF PROCextend:IF task$(0)="" PROC
nothing("extend"):ENDPROC
4590 full=0:extend=TRUE
4600 PROCrubric("extended file",0):PROC
wnd("mid")
4610 IF level%<5 select%=FNmarker(row%,
-1):star=(MID$(task$(select%),34,1)="*")
:REM extend=0
4620 IF esc PROCmenu:ENDPROC
4630 parent$=task$(select%)
4640 level%=VAL(MID$(parent$,36,1))
4650 IF level%=5 PROClimit:ENDPROC
4660 title$(level%+1)=parent$
4670 memoMrk$=MID$(parent$,35,1)
4680 extndDln$=MID$(parent$,37,10)
4690 row$=LEFT$(parent$,33)+"*"+memoMrk
$+STR$level%+MID$(parent$,37,24)
4700 task$(select%)=row$
4710 IF NOTstar AND level%=0 PROCwriteD
ata("Index")
4720 extndMrk$="$":level%=level%+1
4730 fnme$(level%)="Level"+STR$(level%)
+"."+MID$(row$,4,3)+LEFT$(row$,3)
4740 IF level%>0 AND NOT star PROCwrite
Data(fnme$(level%-1))
4750 PROCwnd("mid"):CLS
4760 PROCreadData:ENDPROC
4770 :
4780 DEF PROClimit
4790 VDU7:PROCwnd("blank")
4800 PRINT"Extend limit"'"is five stage
s"'"Press any key"'"to continue"
4810 G=GET:PROCmenu:ENDPROC
4820 :
4830 DEF PROCdelete:IF task$(0)="" PROC
nothing("delete"):ENDPROC
4840 full=0:delete=TRUE
4850 numRows%=row%:REPEAT:PROCrubric("i
tem",full):PROCwnd("mid")
4860 fldNo%=FNmarker(numRows%,0)
4870 repeated=(LEFT$(task$(fldNo%),3)="
00"):change=TRUE:PROCwnd("blank")
4880 IF NOT repeated PRINT "Confirm del
etion?"'"Enter (Y/N) ";:IF FNyn="N" CLS:
PROCreprint:GOTO 4950
4890 type$=MID$(task$(fldNo%),34,1)
4900 fName$=MID$(task$(fldNo%),4,3)
4910 delFnme$=fName$+LEFT$(task$(fldNo%
),3):IF type$="*" PROCdeleteFile
4920 PROCreprint:PROCrvrs (bl%,wh%)
4930 PRINT TAB(33,fldNo%)STRING$(6," ")
"To be deleted" STRING$(5," ");:PROCrvrs
(wh%,bl%)
4940 IF repeated VDU7:ELSE task$(fldNo%
)="000"+MID$(task$(fldNo%),4):row%=row%-
1
4950 PROCwnd("blank")
4960 IF row%<0 del$="N" ELSE PRINT"More
deletions?"'"Enter <Y/N> ";:del$=FNyn:P
ROCwnd("blank")
4970 UNTIL del$="N":PROCrePackRows
4980 PROCdisplay:ENDPROC
4990 :
5000 DEF PROCreprint
5010 PROCwnd("mid"):PROCrvrs(wh%,bl%)
5020 PRINT TAB(0,fldNo%)MID$(task$(fldN
o%),4,30);:ENDPROC
5030 :
```

New Generation!

RISC *user*

RISC User, the highly popular magazine for Archimedes users, is bigger and better. The new RISC User is now B5 size which offers a sophisticated design, bigger colour illustrations and bigger pages with more information. Altogether better value and no increase in price.

RISC User is still a convenient size to assemble into an easy-to-use reference library, containing all the information you need as an Archimedes user. Every issue of RISC User offers a wealth of articles and programs with professionally written reviews, lively news, help and advice for beginners and experienced users, and items of home entertainment. Altogether RISC User has established a reputation for accurate, objective and informed articles of real practical use to all users of Acorn's range of RISC computers.

CAN COMPUTER GAMES EDUCATE?
First of a new series of educational features looking at education in the school and in the home.

DISC FILE RESCUER
A complete application for rescuing files from corrupt or damaged discs.

COLOUR GRAPHICS CARDS
Two new cards for enhanced colour graphics reviewed and compared.

THE TIMES AND SUNDAY TIMES ON CD-ROM
More new releases for CD-ROM

IMAGERY
Objective review of a new colour image processing package.

FLOPPY FAX
Software to turn an Arc and modem into a high quality fax machine.

STENCIL
New techniques with Draw files.

PC PAGES
A regular series devoted to the PC emulator and to the running of PC software on the Archimedes.

WRITE-BACK
A readers' section of RISC User for comment, help, information - a magazine version of a bulletin board.

WP/DTP
Articles on using Ovation and Impression DTP packages

INTO THE ARC
A regular series for beginners.

TECHNICAL QUERIES
A regular column attempting to answer your technical queries.

SUBSCRIPTION DETAILS

As a member of BEEBUG you may extend your subscription to include RISC User for only £10.50 (overseas see table).

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

```
 5040 DEF PROCrePackRows:PROCwnd("mid")
 5050 s%=-1:code%=0:REPEAT:s%=s%+1
 5060 IF VAL(LEFT$(task$(s%),3)) > 0 tas
k$(code%)=task$(s%):code%=code%+1
 5070 UNTIL s%=29:FOR A%=code% TO 29:tas
k$(A%)="":NEXT:ENDPROC
 5080 :
 5090 DEF PROCdeleteFile:PROCwnd("blank"
):PRINT"Updating"'"Please wait"
 5100 OSCLI("Delete $.Level"+STR$(level%
+1)+"."+delFnme$):PROCwnd("mid")
 5110 PROCrvrs(wh%,bl%):PRINT TAB(0,fldN
o%)MID$(task$(fldNo%+1),4,30)
 5120 ENDPROC
 5130 :
 5140 DEF PROCdisplay:LOCAL max%
 5150 PROCwnd("mid"):VDU12,31,0,0
 5160 max%=VAL(LEFT$(task$(1),3))
 5170 FOR read%=0 TO 28:rcd$=LEFT$(task$
(read%),3)
 5180 IF VALrcd$>max% max%=VALrcd$
 5190 IF VALrcd$>0 PROCformRow(task$(rea
d%),read%)
 5200 NEXT:rcdNo%=max%:PROCwriteData(fn$
):PROCmenu:ENDPROC
 5210 :
 5220 DEF PROCmodify:IF task$(0)="" PROC
nothing("modify"):ENDPROC
 5230 mod=TRUE:PROCrubric("event",full):
select%=FNmarker(row%,-1)
 5240 IF esc PROCmenu:ENDPROC
 5250 change=TRUE
 5260 rcd$=FNpad(3,LEFT$(task$(select%),
3),"0",-1):depth%=VAL(MID$(task$(select%
),36,1)):star=(MID$(task$(select%),34,1)
="**")
 5270 oldFnme$="$.Level"+STR$(depth%+1)+
"."+MID$(task$(select%),4,3)+rcd$
 5280 PROCgetTask(select%):IF esc PROCrh
Column(select%)
 5290 newFnme$="$.Level"+STR$(depth%+1)+
"."+LEFT$(event$,3)+rcd$
 5300 IF star title$(level%+1)=LEFT$(eve
nt$,32)+extndMrk$+memoMrk$+extndDln$:PRO
CwriteData(fnme$(level%))
 5310 IF star AND NOT same PROCrename
 5320 IF row%<1 GOTO 5370
 5330 PROCwnd("blank")
 5340 PRINT"More modifications?"
 5350 PRINT'"Enter <Y/N>";
 5360 IF FNyn="Y" PROCwnd("blank"):PROCm
odify:ENDPROC
 5370 mod=0:skip=0:PROCmenu:ENDPROC
 5380 :
 5390 DEF PROCrename
 5400 PROCwnd("botSide"):PRINT"Renaming"
 5410 PRINT"this file"'"Please wait"
 5420 OSCLI("RENAME "+oldFnme$+" "+newFn
me$)
 5430 hndl%=OPENUP(newFnme$):PRINT#hndl%
,rcd$+LEFT$(event$,30)+memoMrk$+depth$+e
xtndDln$:CLOSE#hndl%:ENDPROC
 5440 :
 5450 DEF PROCsort(r%)
 5460 IF task$(0)="" OR r%=0 PROCnothing
("sort"):ENDPROC
 5470 change=TRUE:PROCcursor(0)
 5480 PROCwnd("mid")
 5490 FOR outer%=0 TO r%-1:pointer%=oute
r%:FOR inner%=outer%+1 TO r%
 5500 IF sort%(inner%)<sort%(pointer%) p
ointer%=inner%
 5510 NEXT inner%:IF pointer% <>outer% P
ROCswap(pointer%,outer%)
 5520 NEXT outer%:PROCdisplay:ENDPROC
 5530 :
 5540 DEF PROCswap(a,b)
 5550 LOCAL spare$:spare$=task$(b)
 5560 task$(b)=task$(a):task$(a)=spare$
 5570 spare=sort%(b):sort%(b)=sort%(a)
 5580 sort%(a)=spare:ENDPROC
 5590 :
 5600 DEF PROCprintout:IF task$(0)="" PR
OCnothing("print"):ENDPROC
 5610 printMenu=TRUE
 5620 IF change PROCwriteData(fnme$(leve
l%))
 5630 prnt=TRUE:PROCwnd("blank")
 5640 format%=FNmarker(3,-1):printMenu=0
 5650 IF esc prnt=0:PROCmenu:ENDPROC
 5660 IF format%=0:VDU2,1,27,1,15,1,27,1
,108,1,5
 5670 IF format%=1:VDU2,1,18,1,27,1,77,1
,27,1,108,1,7
```

```
 5680 IF format%=2:VDU2,1,18,1,27,1,80,1
,27,1,108,1,7
 5690 PROCpaperControl:PROCheading
 5700 tally%=-1:REPEAT:tally%=tally%+1
 5710 serial%=VAL(LEFT$(task$(tally%),3)
):IF serial%>0 PROCtype
 5720 UNTIL serial%=0:VDU3:newSheet=0
 5730 VDU2,1,10,1,10,3:free%=free%-2
 5740 free%=free%-((row%+1)+3):CLOSE#0:p
rnt=0
 5750 PROCformFeed:ENDPROC
 5760 :
 5770 DEF PROCnothing(detail$):*fx200,1
 5780 PROCwnd("blank")
 5790 PRINT "Nothing to "detail$
 5800 PRINT'"Press any key"'"to continu
e":IF GET:*fx200,0
 5810 PROCmenu:ENDPROC
 5820 :
 5830 DEF PROCtype:*fx3,10
 5840 PROCformRow(task$(tally%),tally%):
*fx3,0
 5850 ENDPROC
 5860 :
 5870 DEF PROCpaperControl
 5880 newSheet=(free%<row%)
 5890 IF NOT newSheet ENDPROC
 5900 VDU3,7:PROCwnd("blank"):CLS
 5910 PRINT "Not enough space"'"on this
sheet"'"for this list"
 5920 PROCformFeed:ENDPROC
 5930 :
 5940 DEF PROCformFeed
 5950 IF NOT prnt CLS:PRINT "Continue pr
inting?"'"Answer (Y/N)":ans$=FNyn:IF ans
$="N" PROCfeed:PROCmenu:ENDPROC
 5960 IF newSheet prnt=0:PROCfeed:PRINT'
"Using cut sheets?":IF FNyn="Y" PRINT'"C
hange paper then"'"press any key":IF GET
 5990 PROCmenu:ENDPROC
 6000 :
 6010 DEF PROCfeed:VDU2,1,12,3:free%=pLe
n%-(tearOff%+botM%):ENDPROC
```

B

Wordwise User's Notebook (continued from page 42)

```
.enter-string
REM Set up string input window
VDU28,0,24,39,23
VDU136
PRINT "String";
VDU137
PRINT ":";
K$=K$+GLK$
CLS
PROCright-window
GOTO get-code
```

CLIST

```
|! *f0*
|!!*f1*
|!"*f2*
|!#*f3*
|!$*f4*
|!%*f5*
|!&*f6*
|!'*f7*
|!(*f8*
|!)*f9*
|!,*Cursor L*
|!-*Cursor R*
|!.*Cursor D*
|!/*Cursor U*
|I*TAB*
|M*RETURN*
|[*ESCAPE*
|!!\*CTRL-Cursor L*
|!!}*CTRL-Cursor R*
|!!^*CTRL-Cursor D*
|!!-*CTRL-Cursor U*
|!!L*SHIFT-Cursor L*
|!!M*SHIFT-Cursor R*
|!!N*SHIFT-Cursor D*
|!!O*SHIFT-Cursor U*
|A*CTRL-A*
|S*CTRL-S*
|D*CTRL-D*
```

B

Sideways ROMs (Part 3)

Mr Toad concludes his epic series on the mysteries of Sideways ROMs.

This month we bring you the second and third off-the-peg sideways ROM headers. If you typed the last one in, you can use over half of it to give you a head start with these.

First we'll examine the 'medium-length' header, Listing 1. Starting from last month's short version, there are a few changes up to (the old) line 400; note the following significant changes:

1. The old line 240 is now 230. The new line 240 sets the start-point for O% in a new way - in fact, we hold it in Z%, as we need to preserve the start address for the *SRWRITE. DIM is all right once you know the approximate length of your code, but this header might be used for quite a long ROM, so we could do with a way of starting the assembled code immediately after the variables, without having to worry about the size of the DIM. You could, of course, just DIM some ridiculously large amount of space; after all, you are not going to need either the Basic or the assembled code after the *SRWRITE; the chances are that next you will load a new Basic program, a word processor file or whatever, so main memory will be used from scratch again. Still, there is an easy and tidy way which is much more businesslike.

VAR-TOP, the address of the first available byte after the Basic variables, is not a Basic pseudo-variable like TOP, but it is held LSB-MSB in locations 2 and 3, so line 240 simply peeks them and sets Z% to that point. At the beginning, when there are no variables, VAR-TOP will be the same as TOP, so on the first pass the variables area will overwrite the start of the assembled code, but at the start of the second pass all the variables have been declared and line 240, being inside the loop, starts the assembly at exactly the right place.

2. The new title in line 320 is followed by some new text. I decided this time to double up the title and the start of the *HELP text, so as to be able to discuss the technique. Now you will recall that we have no choice about what happens in response to *ROMS; the MOS prints out everything between &8009 and the next zero, taking it as a string, so this is where you must put the title. Next, it prints a space and the number in &8008, which is the 'binary version number'. It is different in the case of *HELP: you decide what, if anything, happens in response to call 9. The *ROMS routine does make provision for another string, the ASCII version string, to be included after the title, which is ignored by *ROMS but can be used by a *HELP routine. It must end with another zero - or more accurately, the next thing is the copyright string which must begin with a zero.

Now Acorn may call this string what they like, but you can include what you need, so long as the copyright string remains within &100 bytes of the start, because the part of the initialisation routine which uses the copyright offset pointer at &8007 can't cope with crossing a page boundary. The style of the *HELP text here does not follow the standard layout, but I have no scruples about breaking the rules as long as it can't mess up other software. If we included an EQUB &0C at the start of our text so as to clear the screen and leave our own message in splendid isolation, or if we claimed 'all ROMs' calls, now that really would be a bit much, but surely matters of style are up to us?

We can't use a zero as end marker in our *HELP data, since printing would stop at the first zero after the title. Some programmers get round this by calling a print routine twice, but we can just as

easily use a different end marker. &0D won't do, as you can see, but a 6 (enable VDU) or a &1B (does nothing) are fine, so the new *HELP routine uses CMP #&1B to find the end of the text. This replaces a PHP and a PHP, so it's the same length. Now, we must print a blank line between our message and the previous one. Problem - we can't start the title string with EQUB &0D; the *ROMS routine wouldn't like it, so we squander three bytes on a JSR osnewl at the start.

Going back to to the Acorn convention: on *HELP, a ROM should print a title, the ASCII version string, and, if it's going to provide 'extended help', then on the line below, indented by two spaces, the keyword to produce that extended help. Example: do *HELP and note something like:

```
Advanced DFS 1.50
ADFS
```

Typing *HELP ADFS produces a list of all ADFS commands plus details of parameters. This is a bit much for a ROM with only three or four commands, so I have here in effect provided a variant on that system which comes up on the normal *HELP call. Incidentally, if you do provide extended help, you should check whether (&F2),Y points to a full stop; if so, you should respond, because that's a request for 'extended help, all ROMs'. Now you shouldn't really respond with even the normal help message if the call was in fact a request only for extended help from some other ROM, so you should check that (&F2),Y points to a carriage return before you print. This I have added here (line 450). Why the heck don't ROMs claim these special calls? Then the rest of us wouldn't have to code tests for them.

The interpreter at *isItOurs* is totally different. Quite often, a ROM will provide several closely associated functions - a good example was Mr Toad's Macro ROM in BEEBUG Vol.11 No.4, where everything had to do with macros. I was able to choose reasonable mnemonic commands, all of two letters of which the last was M: *DM disable macros, *EM to enable macros, to *LM list macros, and so on. If you can manage to find such a set of commands without too much strain on the English language, you can use a novel and very compact form of interpreter. If it's no good for your purpose at some point in the future, listing 2 has a conventional multi-command version.

When the interpreter is called, (&F2),Y points to the first non-asterisk, non-space character of the star command which has just been issued. If it's one of ours, then the last letter will always be the same; for demo purposes here we have a P for print. After that there will be a carriage return. Thus we can more easily work through the sequence backwards. First INY:INY, then we test for &0D, and if the test fails, then the length was wrong and we exit immediately. Now DEY and test for the P. If that's there, then we DEY again and test for all the possible first letters. Notice that in the demo I test for the last possibility with a BNE exit, so if it's DP the code simply continues into the *DP code. I do a similar trick later in line 720; it's nothing special, but good design in this respect can save a lot of labels and branches when you're hard pressed for memory.

Obviously, the actual star commands and their functions are dummies for test and demo purposes, but you'll be able to substitute your own set of strings. You may not be able to get all your real routines within branching range of the interpreter. Branches to 'staging-post' jumps are the obvious answer, but don't forget that you can add a routine or two *before* the interpreter, thus maybe getting within range. In fact, the branching range is a fair old stretch - about the length of this whole listing, either way.

Now for a look at Listing 2, the big version. It will be best if we again start from last month's small version. Lines 20 to 400 are identical except that we take 230 and 240 from Listing 1 - set up the loop in 230 and set Z% by peeking VARTOP in 240. The only reason I didn't do it last month is that I wanted to provide alternatives wherever possible. You should also change line 10 and the title-string (320), in case there's ever a mix-up of files.

Listing 2, then, begins with the above-mentioned replacement lines for the *small* header, and then goes straight into a full *HELP routine from line 410, providing the extended help service discussed above. Line 440 discards spaces, 450 detects ordinary *HELP calls, 460 catches the 'help-dot' call and the next three pick up "*HELP BR" - you would no doubt change it to your own string - lines 470, 490 and 1000 - 1050. You'll note that there's a bit of fiddling been done in 1010/20 to keep to the conventional layouts for the two levels of help, since that's what I set out to provide in this version; your chosen strings should be able to imitate it. If you want a longer string you can add an INY, then duplicate the pattern of lines 480 - 490.

As before, the command interpreter is real but the commands and functions are dummies. The code of this interpreter is subtle and hard to grasp if you've not seen it before. This time it isn't mine - I wish it were, but it's been around so long that I've no idea who to credit.

The first thing to note is that each star command string listed in *.pickYourOwn* is followed by the address of the corresponding routine, stored the unusual (for computing) way round, MSB-LSB. That's why we didn't use the EQUW statement. Now the ASCII codes for A-Z are positive in two's complement - bit 7 is not set, and loading one into A will RESET the negative flag. Since any sideways ROM routine comes after &8000, the high byte of its address is negative - bit 7 is set and loading it will SET the negative flag. There's one other point to recall: way, way back, (or so it seems - actually it was about seven bytes of code ago) we pushed all the registers in case of a no-claim exit. Y was pushed last and at the moment it's on top of the stack. Throughout the routine, X will be the pointer within our table; of course, since Y points to the characters of the star command we are testing.

After skipping any spaces and masking to upper case (710) a comparison is made between the byte at (&F2),Y and the first string in our table (720) - if it succeeds, we go round the small loop *.chkLp* to the next character. If it fails, the next byte in our table is loaded into A and the negative flag is tested (730). If the test succeeds, we branch to *.haveWeGotOne*. At this point, either we've passed the end of a correctly matched string, in which case *.pickYourOwn,X* holds the high byte of that routine's address, or we have come to the end marker of our table (&FF - also negative), so we test for &FF (780). If that test fails, we've got a match: we branch to *.gotOne* where we load the address into two spare bytes of main memory - the more usual way round - and jump to it. If the test for &FF in line 780 succeeds, no branch is made and we hit an exit routine.

Now let's go back to the first loop, *.chkLp* and the test for negative at 730. If it fails, then we dropped out of *.chkLp* because of a simple mismatch. Line 750 steps through our table, incrementing X, until the negative flag is set. Now we're at a high byte. In 760 we increment X once more, to pass over the low byte. It is now pointing to the start of the next command in our table. We pull Y off the stack to point once more to the start of the command under test, push it back to save it again and branch back to *.chkLp*.

As I said, I just wish I could claim to have written that little lot! Incidentally, if your whole command table won't fit into &100 bytes, you'll have to re-code this whole routine so as to go round the main bits twice or more. I've never done it, but it shouldn't take much more than a week - *if* you don't run out of tranquillisers.

If you can write assembly language at any level, then after these three articles you should now be equipped to produce a sideways ROM to order. Maybe you were before! Maybe you understood it before, but now I've confused you. Anyway, the headers should come in handy. And if I were you, looking at this lot, if I wanted the code I'd buy the magazine discs.

```
  10 REM LISTING 1:Medium ROM Header
  20 REM Version B 1.0
  30 REM Author  David Holton
  40 REM BEEBUG  Jan/Feb 1993
  50 REM Program subject to copyright
  60 :
 100 PROCassem
 110 FOR N%=7 TO 4 STEP-1
 120 IF N%?&2A1 NEXT:PRINT''"No free SR
AM slot."':END
 130 OSCLI "SRWRITE "+STR$~Z%+" "+STR$~
(O%+1)+" 8000 "+STR$ N%
 140 PRINT''"READY IN SLOT ";N%
 150 N%?&2A1=&82:N%=4:NEXT:END
 160 :::::::::::::::::::::
 170 DEF PROCassem
 180 osasci=&FFE3
 190 osnewl=&FFE7
 200 oswrch=&FFEE
 210 osword=&FFF1
 220 osbyte=&FFF4
 230 FOR N%=4 TO 6 STEP 2
 240 Z%=?2+&100*?3
 250 P%=&8000:O%=Z%
 260 [ OPT N%
 270 BRK:BRK:BRK
 280 JMP checkCalls
 290 EQUB &82
 300 EQUB copyright MOD &100
 310 EQUB &93
 320 EQUS "Medium-sized ROM":BRK:EQUB &
0D
 330 EQUS "  *AP (print 'A')":EQUB &0D
 340 EQUS "  *BP (print 'B')":EQUB &0D
 350 EQUS "  *CP (print 'C')":EQUB &0D
 360 EQUS "  *DP (print 'D')":EQUW &1B0
D
 370 .copyright
 380 BRK:EQUS"(C) me 1992"
 390 :
 400 .checkCalls
 410 PHA:PHX:PHY
 420 CMP #4:BEQ isItOurs
 430 CMP #9:BNE noClaim
 440 :
 450 LDA (&F2),Y:CMP #&0D:BNE noClaim
 460 JSR osnewl:LDX #&FF
 470 .helpLoop
 480 INX:LDA &8009,X:JSR osasci
 490 CMP #&1B:BNE helpLoop
 500 :
 510 .noClaim
 520 PLY:PLX:PLA:RTS
 530 :
 540 .isItOurs
 550 INY:INY
 560 LDA (&F2),Y
 570 CMP #&0D:BNE noClaim
 580 DEY:LDA (&F2),Y:AND #&DF
 590 CMP #ASC"P":BNE noClaim
 600 DEY:LDA (&F2),Y:AND #&DF
 610 CMP #ASC"A":BEQ a
 620 CMP #ASC"B":BEQ b
 630 CMP #ASC"C":BEQ c
 640 CMP #ASC"D":BNE noClaim
 650 :
 660 LDA #ASC"D":JMP go
 670 .a
 680 LDA #ASC"A":JMP go
 690 .b
 700 LDA #ASC"B":JMP go
 710 .c
 720 LDA #ASC"C"
 730 :
 740 .go
 750 JSR oswrch:JSR osnewl
 760 LDA #7:JSR oswrch
 770 :
 780 .claimExit
 790 PLY:PLX:PLA:LDA #0:RTS
 800 ]:NEXT:ENDPROC
```

```
 10 REM LISTING 2:Full-size ROM Header
 20 REM Version B 1.0
 30 REM Author  David Holton
 40 REM BEEBUG  Jan/Feb 1993
 50 REM Program subject to copyright
 60 :
230 FOR N%=4 TO 6 STEP 2
240 Z%=?2+&100*?3
320 EQUS "Full-size sideways ROM"
400 :
410 JSR osnewl
420 .tests
430 LDA (&F2),Y:INY
440 CMP #&20:BEQ tests
450 CMP #&0D:BEQ simpleHelp
460 CMP #ASC".":BEQ bigHelp
470 AND #&DF:CMP #ASC"B":BNE noClaim
480 LDA (&F2),Y:AND #&DF
490 CMP #ASC"R":BNE noClaim
500 :
510 .bigHelp
520 LDA #0:STA &8F:JMP help
530 :
540 .simpleHelp
550 LDA #&1B:STA &8F
560 :
570 .help
580 LDX #&FF
590 .loop
600 INX:LDA helpText,X:JSR osasci
610 CMP &8F:BNE loop
620 JSR osnewl
630 :
640 .noClaim
650 PLY:PLX:PLA:RTS
660 ::::::::::::::::
670 .isItUs
680 LDX #&FF
690 .chkLp
700 LDA (&F2),Y:INY
710 CMP #&20:BEQ chkLp:AND #&DF
720 INX:CMP pickYourOwn,X:BEQ chkLp
730 LDA pickYourOwn,X:BMI haveWeGotOne
740 .tryNext
750 INX:LDA pickYourOwn,X:BPL tryNext
760 INX:PLY:PHY:BRA chkLp
770 .haveWeGotOne
780 CMP #&FF:BEQ noClaim
790 .gotOne
800 STA &DEFF
810 INX:LDA pickYourOwn,X:STA &DEFE
820 JMP (&DEFE)
830 ::::::::::::::::::::::::::::
840 .function1
850 LDA #7:JSR oswrch:LDA #ASC"A":JSR
oswrch:JMP claimOut
860 .function2
870 LDA #7:JSR oswrch:LDA #ASC"B":JSR
oswrch:JMP claimOut
880 .function3
890 LDA #7:JSR oswrch:LDA #ASC"C":JSR
oswrch
900 .claimOut
910 PLY:PLX:PLA:LDA #0:RTS
920 ::::::::::::::::::::::::::::
930 .pickYourOwn
940 EQUS"BRA"+CHR$ &0D:EQUBfunction1 D
IV &100:EQUBfunction1 MOD &100
950 EQUS"BRB"+CHR$ &0D:EQUBfunction2 D
IV &100:EQUBfunction2 MOD &100
960 EQUS"BRC"+CHR$ &0D:EQUBfunction3 D
IV &100:EQUBfunction3 MOD &100
970 EQUB &FF \ End-Marker!
980 :
990 .helpText
1000 EQUS CHR$ &0D+"Full-size sideways
ROM"+CHR$ &0D
1010 EQUS"  BR"+CHR$ &1B
1020 EQUS"A"+CHR$ &0D
1030 EQUS"  BRB"+CHR$ &0D
1040 EQUS"  BRC"+CHR$ &0D
1050 EQUS" etc etc...
1060 BRK \ End-Marker!
1070 ]
1080 NEXT:ENDPROC
```

```
5120 ENDPROC
5130 :
6000 DEF PROCshortstrings
6010 L8$=STRING$(8,we$)
6020 s8$=STRING$(8," ")
6030 L44$=STRING$(44,we$)
6040 s44$=STRING$(44," ")
6050 ENDPROC
6060 :
7000 DEF PROCcompletestrings
7010 a1$=" "+se$+L44$+ws$
7020 a2$=" "+ns$++s44$+ns$
7030 a3$=" "+nse$+L8$+wse$+L8$+wse$+L8$
+wse$+L8$+wse$+L8$+wns$
7040 a4$=" "+ns$+s8$+ns$+s8$+ns$+s8$+ns
$+s8$+ns$+s8$+ns$
7050 a5$=" "+nse$+L8$+wnse$+L8$+wnse$+L
8$+wnse$+L8$+wnse$+L8$+wns$
7060 a6$=" "+ne$+L8$+wne$+L8$+wne$+L8$+
wne$+L8$+wne$+L8$+wn$
7070 ENDPROC
```

```
10 REM Program +Codes
100 title$="Form Designer Compass Code
s ":ON ERROR GOTO 10000
5030 PRINTa2$'a2$'a2$'a7$'a2$'a2$'a2$
5060 PROCcodes(J%):PRINTa4$'a4$:PROCcha
rs(J%):PRINTa4$
5090 PROCcodes(10):PRINTa4$'a4$:PROCcha
rs(10):PRINTa4$
5200 DEF PROCcodes(j%)
5210 ON j% GOSUB 5310,5320,5330,5340,53
50,5360,5370,5380,5390,5400
5220 ENDPROC
5230 :
5310 PRINTa8$:RETURN
5320 PRINTa9$:RETURN
5330 PRINTa10$:RETURN
5340 PRINTa11$:RETURN
5350 PRINTa12$:RETURN
5360 PRINTa13$:RETURN
5370 PRINTa14$:RETURN
5380 PRINTa15$:RETURN
5390 PRINTa16$:RETURN
5400 PRINTa17$:RETURN
```

```
5410 :
5500 DEF PROCchars(j%)
5510 ON j% GOSUB 5610,5620,5630,5640,56
50,5660,5670,5680,5690,5700
5520 ENDPROC
5530 :
5610 PRINTa8a$:RETURN
5620 PRINTa9a$:RETURN
5630 PRINTa10a$:RETURN
5640 PRINTa11a$:RETURN
5650 PRINTa12a$:RETURN
5660 PRINTa13a$:RETURN
5670 PRINTa14a$:RETURN
5680 PRINTa15a$:RETURN
5690 PRINTa16a$:RETURN
5700 PRINTa17a$:RETURN
5710 :
6050 s6$=STRING$(6," ")
6060 s5$=STRING$(5," ")
6070 s4$=STRING$(4," ")
6080 s3$=STRING$(3," ")
6090 :
7070 a7$=" "+ns$+s8$+"FORM DESIGNER COM
PASS CODES."+s8$+ns$
7080 a8$=" "+ns$+"WN$"+s5$+ns$+"WNS$"+s
4$+ns$+"WNE$"+s4$+ns$+"WNSE$"+s3$+ns$+"W
S$"+s5$+ns$
7090 a8a$=" "+ns$+s6$+WN$+" "+ns$+s6$+W
NS$+" "+ns$+s6$+WNE$+" "+ns$+s6$+WNSE$+"
 "+ns$+s6$+WS$+" "+ns$
7100 a9$=" "+ns$+"WSE$"+s4$+ns$+"WE$"+s
5$+ns$+"WEn$"+s4$+ns$+"WEs$"+s4$+ns$+"WE
ns$"+s3$+ns$
7110 a9a$=" "+ns$+s6$+WSE$+" "+ns$+s6$+
WE$+" "+ns$+s6$+WEn$+" "+ns$+s6$+WEs$+"
"+ns$+s6$+WEns$+" "+ns$
7120 a10$=" "+ns$+"Wn$"+s5$+ns$+"Wns$"+
s4$+ns$+"Ws$"+s5$+ns$+"NS$"+s5$+ns$+"NSE
$"+s4$+ns$
7130 a10a$=" "+ns$+s6$+Wn$+" "+ns$+s6$+
Wns$+" "+ns$+s6$+Ws$+" "+ns$+s6$+NS$+" "
+ns$+s6$+NSE$+" "+ns$
7140 a11$=" "+ns$+"NSw$"+s4$+ns$+"NSwe$
"+s3$+ns$+"NSe$"+s4$+ns$+"NE$"+s5$+ns$+"
Nw$"+s5$+ns$
7150 a11a$=" "+ns$+s6$+NSw$+" "+ns$+s6$
```

```
+NSwe$+" "+ns$+s6$+NSe$+" "+ns$+s6$+NE$+
" "+ns$+s6$+Nw$+" "+ns$
 7160 a12$=" "+ns$+"Nwe$"+s4$+ns$+"Ne$"+
s5$+ns$+"SE$"+s5$+ns$+"Sw$"+s5$+ns$+"Swe
$"+s4$+ns$
 7170 a12a$=" "+ns$+s6$+Nwe$+" "+ns$+s6$
+Ne$+" "+ns$+s6$+SE$+" "+ns$+s6$+Sw$+" "
+ns$+s6$+Swe$+" "+ns$
 7180 a13$=" "+ns$+"Se$"+s5$+ns$+"En$"+s
5$+ns$+"Ens$"+s4$+ns$+"Es$"+s5$+ns$+"wn$
"+s5$+ns$
 7190 a13a$=" "+ns$+s6$+Se$+" "+ns$+s6$+
En$+" "+ns$+s6$+Ens$+" "+ns$+s6$+Es$+" "
+ns$+s6$+wn$+" "+ns$
 7200 a14$=" "+ns$+"wns$"+s4$+ns$+"wne$"
+s4$+ns$+"wnse$"+s3$+ns$+"ws$"+s5$+ns$+"
wse$"+s4$+ns$
 7210 a14a$=" "+ns$+s6$+wns$+" "+ns$+s6$
+wne$+" "+ns$+s6$+wnse$+" "+ns$+s6$+ws$+
" "+ns$+s6$+wse$+" "+ns$
```

```
 7220 a15$=" "+ns$+"we$"+s5$+ns$+"ns$"+s
5$+ns$+"nse$"+s4$+ns$+"ne$"+s5$+ns$+"se$
"+s5$+ns$
 7230 a15a$=" "+ns$+s6$+we$+" "+ns$+s6$+
ns$+" "+ns$+s6$+nse$+" "+ns$+s6$+ne$+" "
+ns$+s6$+se$+" "+ns$
 7240 a16$=" "+ns$+"B$"+s6$+ns$+"BW$"+s5
$+ns$+"BN$"+s5$+ns$+"BS$"+s5$+ns$+"BE$"+
s5$+ns$
 7250 a16a$=" "+ns$+s6$+B$+" "+ns$+s6$+B
W$+" "+ns$+s6$+BN$+" "+ns$+s6$+BS$+" "+n
s$+s6$+BE$+" "+ns$
 7260 a17$=" "+ns$+"bi$"+s5$+ns$+"b2$"+s
5$+ns$+"b3$"+s5$+ns$+"#$"+s6$+ns$+s8$+ns
$
 7270 a17a$=" "+ns$+s6$+b1$+" "+ns$+s6$+
b2$+" "+ns$+s6$+b3$+" "+ns$+s6$+`$+" "+n
s$+s8$+ns$
 7280 ENDPROC
 7290 :
```

B

```
  10 REM Program Kami
  20 REM Version B 1.0
  30 REM Author  Mr Toad
  40 REM BEEBUG  Jan/Feb 1993
  50 REM Program subject to copyright
  60 :
3000 DEF PROCcode
3010 REM**(%@ NB the REM must be at the
start of a line!
3020 REM Kamikaze Kode
3030 FOR n%=0 TO 2 STEP 2
3040 ::::::
3050 REM The main assembly-text goes in
here.
3060 ::::::
3070 P%=&7800
3080 [ OPT n%
3090 :
3100 .kami
3110 JSR backOne
3120 CMP #ASC "@":BNE kami
3130 :
3140 JSR backOne
3150 CMP #ASC "%":BNE kami
3160 JSR backOne
3170 CMP #ASC "(":BNE kami
3180 LDA 0:STA 2
3190 LDA 1:STA 3
3200 JSR backOne
3210 LDA #&FF:STA (0)
3220 JSR backOne
3230 LDA #&0D:STA (0)
3240 JSR backOne
3250 LDA #&E1:STA (0)
3260 JSR backOne
3270 LDA #5:STA (0)
3280 LDA 2:STA 0
3290 LDA 3:STA 1:RTS
3300 :
3310 .backOne
3320 DEC 0:LDA 0
3330 CMP #&FF:BNE P%+4 \ save the rain-
forests - use fewer labels.
3340 DEC 1:LDA (0):RTS
3350 :
3360 ] NEXT
```

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

Please do keep sending in your hints for all BBC and Master computers. Don't forget, if your hint gets published, there's a financial reward.

BEEBUG TAPES ON MASTERS

Gareth Leyshon

Users of Masters may find that early BEEBUG magazine cassette programs, when chained or recalled by OLD after a hard reset, may hang up the first time they are run. This is because these programs include a title page procedure called PROCHP. Within this procedure, a VDU 22,7 call is issued to select mode 7. If the Master is configured to come up in any shadow mode, then HIMEM will initially be set to &8000. Basic will put the stack, which stores the return addresses for procedures and functions, directly below HIMEM.

When the VDU 22,7 command is executed, the Master uses the memory from &7C00 to &7FFF to store the screen display, but does not reset HIMEM. This corrupts the stack with the result that the procedure runs its course, but the computer hangs up when the end of the procedure is reached. This can be avoided by configuring the Master to start up in mode 7 rather than 135, hence reserving the non-shadow memory as required. If your computer has hung in this way, then Break, OLD, RUN will usually get it working properly again, since a soft break preserves the mode set by VDU 22 but allocates HIMEM properly.

MERGE AWAY

Mr.Toad

What a pity the Beeb lacks the MERGE command found on the Spectrum, whereby two Basic listings can be merged into one with all the line numbers in the right places. I've just found out that it can be done very easily on the Master (in fact, it's so easy I'm sure some others must have spotted it already!). Load one listing, then type EDIT. Save the now de-tokenised version under a new filename, via key F3. Now come out of Edit and load the second listing. Type

```
*EXEC <filename>
```

where *filename* is the filename of the first listing saved from Edit, and sit back and watch the fun. Do note that if the two listings have any line numbers in common, the line in the first listing will replace the one in the second.

This method is equivalent to using the *SPOOL command with *EXEC, but is a lot easier and less messy.

EXTRACTING A SINGLE BIT FROM USR

John Carter

Further to Mr.Temple's useful hint in BEEBUG October 1992 (Vol.11 No.5), it is worthwhile remembering that a single bit can be extracted directly from the value returned by the USR function. A typical application is in the use of OSBYTE 117 (&75) to determine whether or not page mode scrolling is in use (this is the scrolling mode which is turned on by Ctrl-N and turned off by Ctrl-O). This information is returned in bit 2 of the X register, and can be used to set a Boolean variable as in the following example:

```
A%=117
S%=-(USR(&FFF4) AND &400)DIV &400
```

The &400 is calculated by setting bit 10, which is the bit we are interested in. The AND part picks out the scroll bit, and the DIV part reduces this to a 1 or a 0. The minus sign then converts this to TRUE or FALSE. The machine can then be reset to its original state by VDU 15+S%.

Personal Ads

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

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

BBC B issue 4 with DFS £75, Wordwise £7. Tel. (0226) 762450.

WANTED: Watford 30Mb Winchester for BBC, BEEBUG Vols. 1,2,3,4,5 no missing issues or pages, complete set good price paid also wanted HCR external ROM/RAM board. Also Deluxe Paint II/enhanced GEM 3 software fonts and drivers Gem First Word Plus, Gem Draw Plus, Gem Artline and other good PC software for M512 also Shibumi Problem Solver and MS flight simulator 2.12 or v3/4 with Shibumi Problem Solver. Tel. (0621) 815162 after 6pm.

BBC single disc drive £60, Solidcad Superdump package also £60, 15 books 10/12 cassette games, Dumpout 3 ROM, Termulator ROM, Wordwise ROM, GXR B+ ROM, Office Mate, Office Master, Personal money manager, master keyboard case plus lead, Grafik disc and Digimouse wapping editor, all four discs and beeb handscanner ideal for posters, all must go cash required, everything at low prices telephone for details. Tel. (0621) 815162 after 6pm.

ARM 3 A3000, 2Mb, serial interface £630, unused (new) Acorn AKF18 multisync monitor £280, Watford Ultimum Expansion unit £90, 40Mb IDE HD £80, Acorn PC Emulator (v1.7) £50. Tel. (0895) 635695 w/ends only.

Acorn A440/1 18 months old, 4Mb RAm, 53Mb hard disc, 3.5" disc drive, colour monitor, excellent for DTP, games and educational uses. Provided with Acorn DTP, First Word Plus and lots of PD £1000. Free delivery within 100 miles of Wolverhampton. Tel. (0902) 850605.

FunSchool 2, 3 discs 40/80T, under 6's, 6-8's, over 8's £5 each. Tel. 073 129 372.

WANTED URGENTLY: Aries B20 Shadow RAM board for BBC B (not Aries B32). Tel. 051-608 5238.

M128 fitted with 512 co-processor complete with Gem mouse and software, BEEBUG Exmon II and Master ROMs, Replay (tape to disc) ROM, twin 40/80T disc drives with own PSU, manuals including View, Viewsheet and Master Reference I & II. Software on tape and disc and BEEBUG magazine from issue 1 £400 the lot. Tel. (0733) 53924 eves or weekends.

Recently moved house and have lost my copy of Wordwise Plus manual. Can anyone help me? Tel. (0254) 701573.

2X 5.25" double sided 40/80T disc drives, £30 each, Viewspell £5. Tel. 031-441 1464 eves or weekends.

MAYDAY! MAYDAY! MAYDAY! Has anyone got a formatter for an NEC hard disc on the BBC B? Tel. (0689) 871223.

M128 as new, complete with handbook, dust cover and original packaging £115. Tel. (0245) 225671 after 7pm.

Brother HR-15 daisywheel printer for sale £50, buyer collects or carriage extra. Tel. (0923) 775098 eves.

Viewstore Database: ROM, handbook and disc £10, Master Reference manuals I&II £13, Assembly Language pro-gramming £5, Structured Basic £4, Advanced User Guide £6, The ABC's of Windows 3.0 £6, The ABC's of Word for Windows £6 all inclusive of postage. Tel. 051-677 1518.

M512 in excellent working order, included are two 5.25 floppies, AMX mouse, joysticks and the following EPROMs in addition to the resident View and Viewsheet: Epson printer driver, Wordwise Plus, Pascal, Screendump, Toolkit and Graphics, full original manuals and discs for the Master 512 plus various extra manuals, years of BBC dedicated magazines, software for both BBC & PC mode including games. Only £350 o.n.o. Tel. (0535) 662157.

BBC Master 40/80T external drive, Philips 80T monitor, Brother HR25 daisywheel printer, Interword, graphics included, all v.g.c little used £300. Tel. (0727) 52366 eves or (0865) 512361 extn 216 day.

Master Compact 128, complete with printer lead, joystick, PAL TV adaptor, Dumpmaster ROM, Repton 3, Repton Infinity, Acornsoft Hits Vols.1 and 2, 13 issues of BEEBUG (consecutive) together with discs, all for £160. Tel. (0446) 744327.

Master 128 (latest MOS), Turbo Co-Pro board, Morley AA ROMboard with Micro-Prolog, Forth, BEEBUG C, Spellmaster, ADI, Ample ROM installed, Morley 20Mb hard disc, 40/80T DD + PSU, Music 5000, AMX mouse + SuperArt, Morley Smart cartridge, Reference manuals I&II, Advances Reference manual, WH Smiths Datacorder, huge collection of books and software on tape/disc (call for list) £300 o.n.o. Buyer collects please. Tel. (0923) 230097.

MORE SATELLITES WANTED

BEEBUG, November issue - Galilean Satellites: what a good program! Best for years. A real application. If you want to see the satellites move, try the following modifications.

Change these lines as shown:

```
 120 MODE7:PROCinput:CDY=DY:MODE1:REPEAT:
PROCjulian:PROCcalc
 130 PROCdisplay
1010 DY=CDY+(XD+XM/60)/24
1890 COLOUR3:COLOUR128:PRINTTAB(5,17)Z$(M
N-1);" ";INT(DY);", ";YR;"      ";INT(DH);"
h ";INT(DM);"m   ":PRINTTAB(5,19)"Julian D
ay No.: ";INT(J*100)/100
```

Add lines:

```
 132 XD=XD+3
 133 IF XD>24 XD=1:CDY=CDY+1
 134 IF CDY>31 CDY=1:MN=MN+1
 135 IF MN>12 MN=1:YR=YR+1
```

Delete lines 1920, 1930 and 1940.

Stephen Rose

Thanks for the bouquets, and for the update. Astronomy fans should find this month's feature on Celestial Bodies of interest too.

SPELLMASTER MALFUNCTIONS WITH VIEW

Using Spellmaster on a Master fitted with the new system ROM (as a spelling check after typing), one can either make a note of spelling errors and then reload the text to correct them, or one can *SAVE LOST 0EFD+HEX where HEX is a suitable value for the length of the text, press Break to reactivate View and READ LOST, which then requires editing at beginning and end. I have not found it possible to avoid editing the beginning whatever address I adopt for the start of the SAVE. Both methods are rather clumsy.

However, as far as I can see from examining memory (with Advanced Disc Toolkit) the text is all there and has not been corrupted either at the beginning or the end after the escape from Spellmaster, even though View reports "No text". Surely it should be possible to tweak the byte that causes the "No text" message and restore View to operation?

D.Ambrose

We have put this problem to Mr Toad, and will publish his response when he has finished his investigations. In the meantime, if any other readers can offer a solution we will pass it on, and publish the details for the benefit of others.

USING BASIC'S GCOL 4

I recently purchased from you many back numbers of your very useful magazine. For years I did not know of it existed and am now catching up on so much which is not easily available from the BBC User Guide.

In Vol.6 Nos.6 & 7 your First Course contributor covered the whole question of colours which explained some mysteries for me. Unfortunately, the articles stopped short of explaining GCOL 4 which I have struggled unsuccessfully to resolve. Is there any coverage of this item in any of your back issues?

John Brown

We have published nothing specifically on GCOL 4 (as this format is not often used), but I will try and explain further. Suppose you select MODE 2, a 16 colour mode, and then use GCOL 0,129 followed by CLG. This will give a red background (by default colour 1 is red, and adding 128 specifies a background). Now select GCOL 4,4 and enter:

```
MOVE 0,0
DRAW 500,500
```

You should see a diagonal line in flashing cyan/red. In binary, the background colour (colour 1) is 0001. Inverting these bits gives 1110 (colour 14) which is flashing cyan/red, again by default. The resulting colour depends only on the colour already on the screen - the colour specified in GCOL 4 has no effect. B

BEEBUG MEMBERSHIP

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

BEEBUG SUBSCRIPTION RATES

	BEEBUG & RISC USER
1 year UK, BFPO, Ch.I	£28.90
Rest of Europe & Eire	£42.90
Middle East	£53.10
Americas & Africa	£58.40
Elsewhere	£62.50

£18.40
£27.50
£33.50
£36.50
£39.50

BACK ISSUE PRICES

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. There is no VAT on magazines.

Volume	Magazine	5"Disc	3.5"Disc
6	£1.00	£3.00	£3.00
7	£1.10	£3.50	£3.50
8	£1.30	£4.00	£4.00
9	£1.60	£4.00	£4.00
10	£1.60	£4.75	£4.75
11	£1.90	£4.75	£4.75

POST AND PACKING

Please add the cost of p&p when ordering. When ordering several items use the highest price code, plus half the price of each subsequent code.

Stock Code	UK, BFPO Ch.I	Europe, Eire	Americas,Africa, Mid East	Elsewhere
a	£1.00	£1.60	£2.40	£2.60
b	£2.00	£3.00	£5.00	£5.50

BEEBUG
117 Hatfield Road, St.Albans, Herts AL1 4JS
Tel. St.Albans (0727) 840303, FAX: (0727) 860263
Manned Mon-Fri 9am-5pm (for orders only 9am-6pm and 9am-5pm Saturdays)
(24hr Answerphone for Connect/Access/Visa orders and subscriptions)

BEEBUG MAGAZINE is produced by RISC Developments Ltd.

Editor: Mike Williams
Assistant Editor: Kristina Lucas
Technical Editor: Mark Moxon
Editorial Assistance: Marshal Anderson
Production Assistant: Sheila Stoneman
Advertising: Sarah Shrive
Subscriptions: Sue Baxter
Managing Editor: Sheridan Williams

CONTRIBUTING TO BEEBUG PROGRAMS AND ARTICLES

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

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

In all communication, please quote your membership number.

Printed by Arlon Printers (0923) 268328 ISSN - 0263 - 7561

Magazine Disc

January/February 1993

CELESTIAL BODIES - The height and direction of the Sun or the stars, the times of sunrise and sunset and the maximum height of the Sun can all be found accurately with this program.

JOBLOG (PART 2) - This is the complete Job Log Personal Organiser program. The new features include a heading showing the base date and the event (or data) heading and the deadline for an event (if deadlined).

USING SIDEWAYS RAM (PART 3) - The programs ROMHdr2 and ROMHdr3 assemble sideways ROM images into the first free bank of sideways RAM. The ROMs are examples of medium and large ROM headers respectively.

WORDWISE USER'S NOTEBOOK - The two Wordwise Plus segment programs CList and DefKeys implement a system which allows sequences of keypresses to be 'recorded' and then 'replayed' via much shorter key sequences.

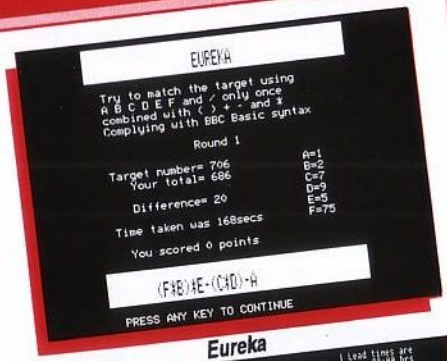
FORM DESIGNER (PART 1) - On this disc are four programs which make up the first part of the Form Designer. There are two complete examples of form printing programs, along with a shell program which can be tailored to produce any form.

EUREKA GAME - Eureka is a mathematical game designed to test your mental arithmetic, based loosely on the numbers game from Channel 4's Countdown. Given a set of numbers, you must combine these numbers using the four rules of arithmetic so that the result is as near to the given number as possible.

BEEBUG WORKSHOP - The two programs on this disc demonstrate preorder, inorder and postorder tree traversal using recursion.

MR TOAD'S MACHINE CODE CORNER - The program given this month shows how to use the method of kamikaze coding to alter a program while it is running.

MAGSCAN DATA - Bibliography for this issue (Vol.11 No.8).



Eureka

JobLog

Fixing the Sun and the Stars

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

Back issues (5.25" and 3.5" discs from Vol.6 No.1) available at the same prices.

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

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

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

Upgrading to an Archimedes

We know that many BEEBUG readers have already upgraded to an Archimedes, and no doubt many more will choose to follow a similar route. For their benefit we offer our advice to help them make a sensible decision on whether to upgrade and if so, what path to take.

Any prices quoted relate to our associated company Beebug Ltd., but note that all prices, particularly those on trade-ins and secondhand items, are likely to change without notice. You should always telephone or write for the latest information.

Archimedes A5000

What System to Choose

All new Archimedes systems are now supplied with the RISC OS 3.10 operating system. Any secondhand system should be upgraded to this. Based on the experience of existing users, we would strongly recommend a minimum of 2Mb of RAM. Most users find a hard disc adds significantly to the convenience of using an Archimedes, but you can always add a low-cost hard drive later, and more memory, but check on the likely price of future expansions - it is not necessarily the same for all machines. If you might be interested in more specialised add-ons (scanners, digitisers, etc.) then check the expansion capability of your preferred system.

Compatibility and Transferability

You will need to decide to what extent you wish to continue using existing discs and disc drives on an Archimedes. An Archimedes and a BBC micro can be directly connected for transfer of files. You can also connect a 5.25" drive to an Archimedes via an additional interface to continue to access 5.25" discs (ADFS format). Our DFS Reader will also allow files to be transferred to the Arc from DFS format discs. However, none of this is possible with the latest A3010/A3020/A4000 systems.

Much BBC micro software will run directly on an Archimedes, or via the 6502 emulator. However, consider this carefully; in our experience, despite prior misgivings, most Archimedes users find that they rapidly adjust to the Desktop environment of the Archimedes, and quickly abandon the software and data of their old system after an initial period.

Software for the Archimedes

The Archimedes is supplied complete with a range of basic applications software. Before embarking on any further purchases it may be better to familiarise yourself with the new machine. Most users look for a word processor (or DTP package), maybe a spreadsheet, or a database, plus other more specialist software. We cannot give detailed guidance here, but back issues of RISC User contain a wealth of useful information - we can advise on suitable issues.

General Advice

It is advisable to discuss your requirements with the BEEBUG technical team before making a final decision on what you want. Try to anticipate future expansion needs at the outset. Note: the price on some systems includes a monitor; in other cases a choice of monitor is available at an additional cost. The details given in the table are minimum specifications of the different Archimedes models.

The A3010

It may also be possible to trade in an existing monitor and/or disc drive, but check if your existing monitor is suitable for use with an Archimedes first. You may find it better to advertise your BBC system in BEEBUG and sell privately - this applies particularly to any software and hardware add-ons which cannot be accepted for a trade-in. In future, all personal ads for Archimedes systems in RISC User will also be included in BEEBUG. You may also defer a trade-in until a later date provided you make this clear at the time of purchase.

Archimedes Systems - Typical or Current Prices

		Secondhand	New
+	A310 1Mb RAM	£350	
+	A410/1 1Mb RAM	£565	
+	A420/1 2Mb RAM, 20Mb hard drive	£650	
+	A440/1 4Mb RAM, 40Mb hard drive	£725	
+*	A3000 1Mb RAM	£350	
*	A3010 1Mb RAM, Family Solution		£ 499.00
*	A3020 2Mb RAM, 60Mb hard drive		£1056.33
*	A4000 2Mb RAM, 80Mb hard drive		£1115.08
*	A5000 2Mb RAM, 80Mb hard drive		£1643.83
+*	Acorn standard colour monitor	£145	£ 258.50

All systems above include a single floppy disc drive.
New (*) and secondhand (+) - all prices inc. VAT.
The A5000 price includes a multiscan colour monitor,
A3020/A4000 price includes standard colour monitor.

BBC Micros - Typical Trade-in Prices

Model B (Issue 7)	£ 35
Model B (issue 7) + DFS	£ 75
Master 128	£125
Master Compact	£ 50

For further information on all Archimedes systems contact:
BEEBUG Ltd. 117 Hatfield Road, St Albans, Herts AL1 4JS. Tel. 0727 40303 Fax 0727 860263