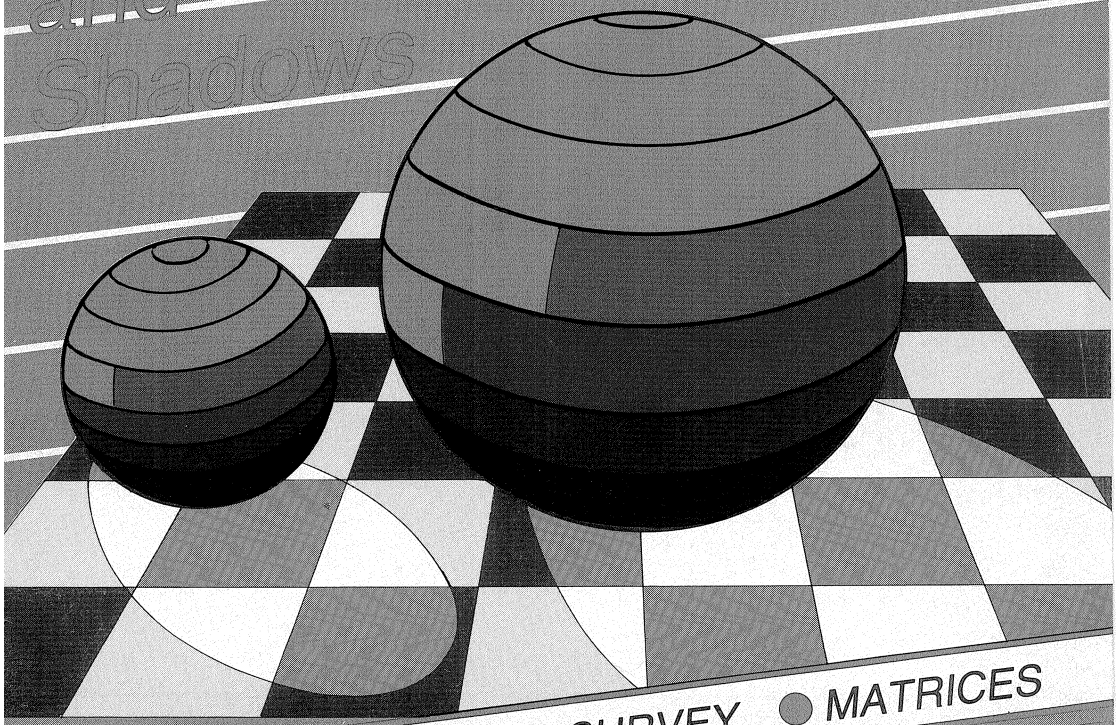
Vol.7 No.3 July 1988

BEEBUG

FOR THE BBC MICRO & MASTER SERIES

3D Shades and Shadows



- 512 FORUM
- PRINTER SURVEY
- MATRICES
- ASTROLOGICAL BIRTH CHARTS

BEEBUG Vol.7 No.3 July 1988

FEATURES

REVIEWS

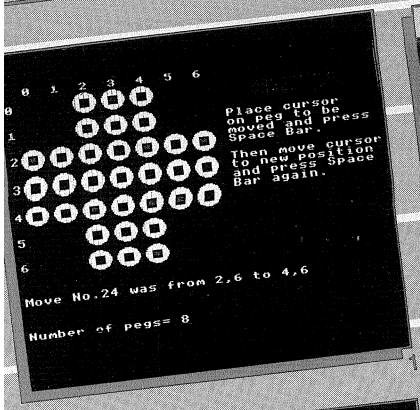
REGULAR ITEMS

HINTS & TIPS

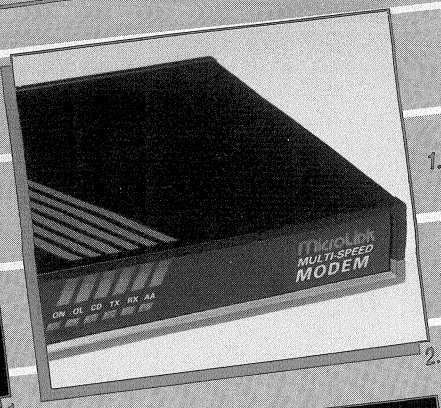
PROGRAM INFORMATION

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

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

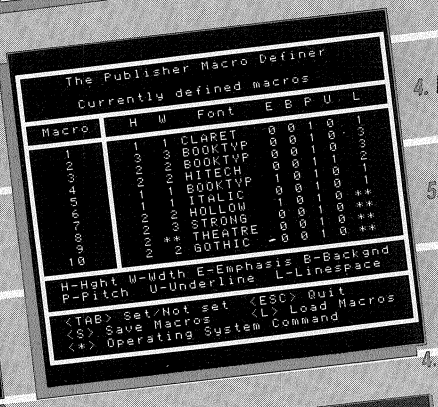


1. Solitaire



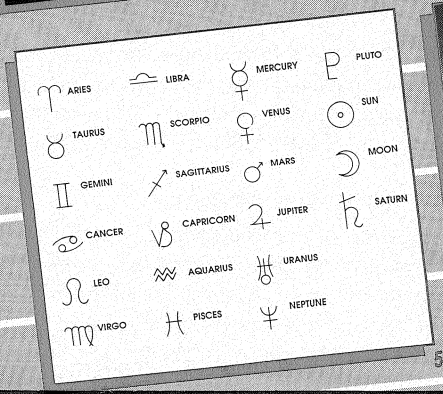
2. Microlink Communications

3. Shades & Shadows



4. PMS Publisher

5. Astrological Birth Chart



6. Latest Printers Survey

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

Computer System

- Master (Basic IV)
- Compact (Basic VI)
- Model B (Basic II)
- Model B (Basic I)
- Electron

Filing System

- ADFS
- DFS
- Cassette

Tube Compatibility

- Tube

Editor's Jottings

BEEBUG MODEM FOR THE MASTER 128

We are pleased that at long last we have finally received BABT approval for the BEEBUG internal modem for the Master 128, and supplies of complete units should be ready for shipment by the end of this month. The modem, which is fully Hayes compatible, is supplied complete with a revised version of our Command software to give full control over all your communications requirements. All orders are being fulfilled as quickly as possible, and we hope other members with a Master 128 will consider this new product for their own machine.

MAGAZINE BACK ISSUES

Enclosed with this issue of BEEBUG you will find a leaflet with special offers on magazine back issues. We have always considered that the many articles and programs which we have published over the years since the BBC micro was launched provide an invaluable library for all BBC micro owners, including those with the Master 128 and Compact.

Our intention has always been to keep back issues in stock (though we have now exhausted stocks of many issues from volume 1), and it is quite remarkable how long lasting some of those earlier programs and articles are. Indeed, we still receive enquiries about programs from as early as volume 2, often with suggestions or requests for further enhancements. We urge all readers to consider taking advantage of our special summer offers, and complete their sets of BEEBUG while copies still remain.

MEMBERS' INFORMATION

From time to time we publish information on user groups and bulletin boards which are known to us. The full lists are normally published twice a year, with amendments and updates at other times. In preparation for our next full printings we would appreciate any information that will help us ensure that the information is as up to date as possible. If you know of any changes or additions to our previously published lists (Bulletin Boards Vol.6 Nos.4, 6 & 7, User Groups Vol.6 Nos.9 & 10 and Vol.7 No.1) then please let us know.

NEWS NEWS

MASTER 512 PRICE CUT

Acorn have recently announced a large reduction in the price of the Master 512 upgrade. The list price of the 512 co-processor, which allows a BBC machine to run IBM PC software, is now just £115 inclusive. When you consider that for this price you get not only a very fast 80186 board with 512K of RAM, but also a mouse and the complete set of GEM software, it becomes a very attractive proposition. This now means that a complete Master 512 system compares favourably with a decent IBM PC compatible, both in terms of price and performance. For users who want to connect a 512 co-processor to a model B, the Universal Second Processor box has also been reduced in price to just £56.35 inclusive. BEEBUG members do of course get the usual discount on these prices, and additionally, BEEBUG can for a limited period supply to members a Master 128 with the 512 co-processor fitted, for just £475, or even less if you take advantage of BEEBUG's trade-in offers.

Acorn have also reduced the price of two of their most popular software products. View Professional now retails at £79.35, while Acornsoft C for the model B and Master is just £67.85. Both prices include VAT.

MASTERISE YOUR MODEL B

Determined that the model B has a lot of life in it yet, a company by the name of Computech has released an upgrade board called *Integra B*. The new board features four banks of sideways RAM, 20K of shadow RAM, 12K of private RAM (as on the B+) and eight spare sockets which can be used for either ROM or RAM. There is also a real-time clock and CMOS RAM on the board.

Integra B simply plugs into the 6502 socket inside the computer, with the processor being moved onto the board. The only other connections are a handful of flying leads that clip to various locations. It is claimed

that, unlike previous attempts at upgrading a model B to a Master, the software provided with Integra B really is compatible with that on a Master. The system costs £130 inclusive, with additional 32K RAM chips costing £11.50 each. Computech are at The Garth, Hamsfell Road, Grange over Sands, Cumbria LA11 6BG. For more information phone (0448) 44604.

REDWOOD TAKEOVER

In a surprise move, Redwood Publishing has been taken over by BBC Enterprises. Redwood will be best known to BEEBUG readers as the company that produces Acorn User Magazine, and also organises the Acorn User show. BBC Enterprises is the commercial wing of the BBC, and is itself no newcomer to magazine publishing, producing Radio Times and The Listener, among others. It is thought that one reason for the takeover is to allow the BBC to produce more specialist magazines. They have already started producing a monthly wildlife magazine, and a fashion magazine that is tied to *The Clothes Show*. It seems in the future that other such ventures will be launched, and that this is made much easier with the expertise of a well established publishing company.

It is not thought that the takeover will have any effect on Acorn User, which will continue to be published by Redwood. There is however, no word on the future of the Acorn User show, which as BEEBUG members will know has suffered a number of setbacks in the past months.

KEYPAD PROBLEMS

Back on the subject of the Master 512, Programmable Systems Design (PSD) have just launched a package to make using the 512 on a model B easier. Much IBM PC software makes use of the numeric keypad of the PC to provide such functions as cursor movement. On a Master this is no problem, because there is an almost identical keypad. However, on a model B there is no keypad, which means that much PC software cannot be used. The new program from PSD allows keys on the main keyboard to emulate the most important ones on the keypad, namely INS, DEL, PG-UP, PG-DOWN, HOME, and END. To make way for these additions, certain other changes have had to be made. For example, the £ symbol is now generated by Ctrl-$.

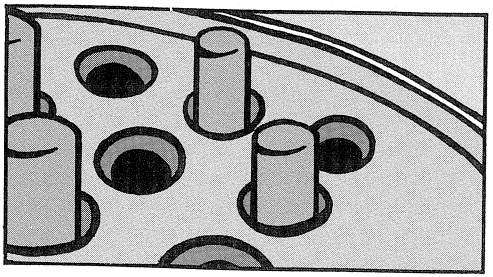
The new key emulator is only available by returning your 512 DOS+ boot disc, for the upgrade to be installed, together with a cheque for £15, to Programmable Systems Design, 20 Beechwood Road, Easton-In-Gordano, Avon BS20 0NA. Phone (0275) 813570 for more details.

FACING THE TYPE

Ian Copestake Software has produced a number of new typefaces for use with twenty four pin dot matrix printers, such as the Epson LQ series. The fonts include Personal, Irish Gaelic, Shadow, and German Fraktur. All the fonts are proportionally spaced, and are very well designed. One interesting feature is the inclusion of ligatures (fi and fl for instance), which can be used to make text much more readable. All the typefaces are supplied on disc for £21.85 in a format for downloading into the printer, although it is hoped to produce ROM module versions that can just be plugged into the printer.

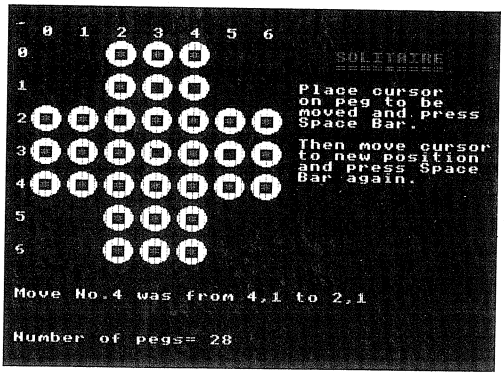
Also new from Ian Copestake Software is a series of special fonts for use with their Wordpower word processor. These new fonts extend the range of *Power Fonts* already produced by the company, and cover the specialist areas of chemistry and physics, containing most of the special symbols needed. The new fonts cost £21.85 for 24 pin printers, or £36.80 for 9 pin printers. Ian Copestake Software are at 10 Frost drive, Wirral, Merseyside L61 4XL, phone 051-648 6287. B

SOLITAIRE



Wile away these lazy summer evenings with a game of Solitaire on the old Beeb using this superb implementation presented by W.E. Gander.

I would imagine that everybody will have come across the game of Solitaire at one time or another. It is such a simple game, yet it continues to intrigue generation after generation. For those that need an explanation, the game is played on a grid of 33 positions arranged in the shape of a symmetrical cross. At the beginning of the game 32 pegs are placed on the board occupying every position except for the very central one. The game can then commence.



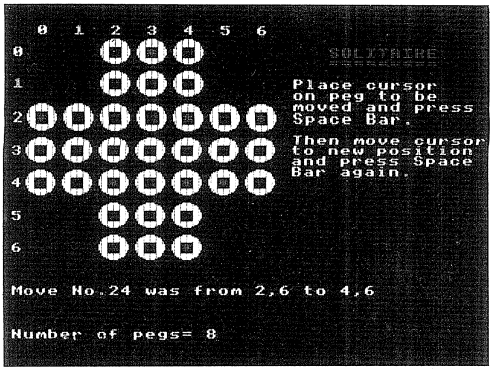
The aim of the game is to remove all of the pegs bar one. A peg is removed when another peg is lifted over it into an empty position. Pegs may only move horizontally and vertically. Although this may sound simple a couple of games may well convince you otherwise. It is not unusual to be left with six or seven pegs on the board and nowhere to move.

The program listed here provides a most attractive implementation of this ancient game. Enter the program as printed, and save it to disc or tape. When you run the program you will be presented with the grid of pegs and a cursor. Select the peg to move by using the cursor keys and then press the Space Bar. Next move to the position where you wish to move the peg and press the Space Bar again. The move will be checked for validity and, providing that all is well, the peg that has been jumped over will be removed from the board. The game will end when you have removed all the pegs bar one. If you get to a position where you have many pegs but can make no further moves press Escape and start again.

I am sure that this implementation of Solitaire will keep you occupied whether you have played the game before or not. If you are not completely convinced that the puzzle is solvable, we have placed an additional program on this month's magazine disc that will show you how to solve the puzzle.

```
 10 REM Program  Solitaire
 20 REM Version  B0.56
 30 REM Author   W.E. Gander
 40 REM BEEBUG   July 1988
 50 REM Program  subject to copyright
 60 :
100 ON ERROR MODE 7:PROCerror:END
110 PROCinit
120 MODE1
130 VDU23,1,0;0;0;0;
140 PROCintro
150 PROCboard
160 DIM dir$(4)
170 READ dx%,dy%
180 DIM a%(dx%,dy%,2)
190 FOR y%=0 TO dy%:FOR x%=0 TO dx%
200 READ a%(x%,y%,0)
210 NEXT:NEXT
220 READ wx%,wy%
230 PROCpins
240 PROCboardplot(x%,y%,dep%)
250 nx%=352:ny%=672:nogood=FALSE
260 REPEAT
270 PROCmove
280 SOUND1,-14,150,1:PROCdelay(1)
290 X%=nx%:Y%=ny%
```

```
  300 x%=((X%/32)-2)/3:y%=(((1024-Y%)/32
)-2)/3
  310 PROCcheck
  320 IF nogood THEN nx%=352:ny%=672:nog
ood=FALSE:UNTIL FALSE
  330 PROCmove
  340 SOUND1,-14,150,1:PROCdelay(1)
  350 PROCsolit(x%,y%,dir%,dep%,np%)
  360 IF nogood THEN nx%=352:ny%=672:nog
ood=FALSE:UNTIL FALSE
  370 np%=np%-1:PRINTTAB(16,28)SPC(2)
  380 PRINTTAB(16,28);np%
  390 UNTIL np%=1 AND a%(wx%,wy%,dep%)=1
  400 CLS:COLOUR1
  410 PRINTTAB(4,5);"WELL DONE - YOU'VE
CRACKED IT !"
  420 PRINTTAB(7,8);"NOW PLEASE PRESS AN
Y KEY"
  430 A$=GET$
  440 RUN
  450 END
  460 :
 1000 DEF PROCinit
 1010 dx%=0:dy%=0
 1020 sol%=0:mx%=0:dep%=0
 1030 VDU23,228,0,0,0,1,7,15,15,31
 1040 VDU23,230,0,0,126,255,255,255,255,
255
 1050 VDU23,232,0,0,0,128,224,240,240,24
8
 1060 VDU23,234,31,63,63,63,63,63,31,31
 1070 VDU23,236,255,255,255,255,255,255,
255,255
 1080 VDU23,238,248,252,252,252,252,252,
248,248
 1090 VDU23,240,15,15,7,1,0,0,0,0
 1100 VDU23,242,255,255,255,255,126,0,0,
0
 1110 VDU23,244,240,240,224,128,0,0,0,0
 1120 Piece$=CHR$228+CHR$230+CHR$232+CHR
$10+CHR$8+CHR$8+CHR$8+CHR$234+CHR$236+CH
R$238+CHR$10+CHR$8+CHR$8+CHR$8+CHR$240+C
HR$242+CHR$244
 1130 ENDPROC
 1140 :
 1150 DEF PROCboard
 1160 FOR i = 0 TO 6
 1170 PRINT TAB(i*3+2,0);i
 1180 NEXT
 1190 FOR i = 0 TO 6
 1200 PRINT TAB(0,i*3+2);i
```




```
 1210 NEXT
 1220 VDU19,3,6;0;:COLOUR3
 1230 FOR E%=7 TO 13 STEP 3
 1240 FOR F%=1 TO 4 STEP 3
 1250 PRINTTAB(E%,F%);Piece$
 1260 NEXT:NEXT
 1270 FOR E%=1 TO 19 STEP 3
 1280 FOR F%=7 TO 13 STEP 3
 1290 PRINTTAB(E%,F%);Piece$
 1300 NEXT:NEXT
 1310 FOR E%=7 TO 13 STEP 3
 1320 FOR F%=16 TO 19 STEP 3
 1330 PRINTTAB(E%,F%);Piece$
 1340 NEXT:NEXT
 1350 ENDPROC
 1360 :
 1370 DEF PROCboardplot(x%,y%,dep%)
 1380 COLOUR1
 1390 FOR x%=0TO0dx%:FOR y%=0TO0dy%
 1400 nx%=(x%*3)+2:ny%=(y%*3)+2
 1410 IF a%(x%,y%,dep%)=1 THEN PRINTTAB(
nx%,ny%);"*" ELSE PRINTTAB(nx%,ny%);" "
 1420 NEXT:NEXT
 1430 COLOUR3
 1440 ENDPROC
 1450 :
 1460 DEF PROCinkey(U,D,L,R)
 1470 TIME=0:REPEATUNTIL TIME>10
 1480 IF INKEY-U ny%=ny%+96:dir%=1
 1490 IF INKEY-D ny%=ny%-96:dir%=3
 1500 IF INKEY-L nx%=nx%-96:dir%=4
 1510 IF INKEY-R nx%=nx%+96:dir%=2
 1520 ENDPROC
 1530 :
 1540 DEF PROCsight:GCOL3,3
 1550 FOR I%=1TO2
```

```
 1560 MOVEnx%,ny%:DRAWnx%+32,ny%:DRAWnx%
+32,ny%-32:DRAWnx%,ny%-32:DRAWnx%,ny%
 1570 NEXT:ENDPROC
 1580 :
 1590 DEF PROCmove
 1600 *FX4,1
 1610 REPEAT:PROCsight:K$=INKEY$(0)
 1620 PROCinkey(58,42,26,122)
 1630 UNTIL K$=CHR$32
 1640 *FX4,0
 1650 ENDPROC
 1660 :
 1670 DEF PROCdelay(X)
 1680 NOW=TIME:REPEAT UNTIL TIME=(NOW+X*
100):ENDPROC
 1690 :
 1700 DEF PROCpins
 1710 np%=0
 1720 FOR y%=0TOdy%
 1730 FOR x%=0TOdx%
 1740 IF a%(x%,y%,0)=1 THEN np%=np%+1
 1750 NEXT:NEXT
 1760 PRINTTAB(0,28);"Number of pegs= ";
np%
 1770 ENDPROC
 1780 :
 1790 DEF PROCcheck
 1800 IF a%(x%,y%,dep%)<>1 THEN PROCmess
age("No pin at this position !"):nogood=
TRUE:ENDPROC ELSE ENDPROC
 1810 :
 1820 DEF PROCsolit(x%,y%,dir%,dep%,np%)
 1830 LOCAL xx%,yy%
 1840 IF dir%=1 THEN destx%=x%:desty%=y%
-2:skipx%=x%:skipy%=y%-1
 1850 IF dir%=2 THEN destx%=x%+2:desty%=
y%:skipx%=x%+1:skipy%=y%
 1860 IF dir%=3 THEN destx%=x%:desty%=y%
+2:skipx%=x%:skipy%=y%+1
 1870 IF dir%=4 THEN destx%=x%-2:desty%=
y%:skipx%=x%-1:skipy%=y%
 1880 IF destx%<0 OR desty%<0 OR destx%>
dx% OR desty%>dy% THEN PROCmessage("Move
 off board !"):nogood=TRUE:ENDPROC
 1890 IF a%(skipx%,skipy%,dep%)<>1 THEN
PROCmessage("There is no peg to jump ove
r"):nogood=TRUE:ENDPROC
 1900 IF a%(destx%,desty%,dep%)=1 OR a%(
destx%,desty%,dep%)=9 THEN PROCmessage("
Destination is occupied or illegal !"):n
ogood=TRUE:ENDPROC
 1910 m%=m%+1:PRINTTAB(0,24)STRING$(39,"
 "):PRINTTAB(0,24);"Move No."m%;" was fr
om ";x%;",";y%;" to ";destx%;",";desty%
 1920 a%(x%,y%,dep%)=0:a%(skipx%,skipy%,
dep%)=0:a%(destx%,desty%,dep%)=1
 1930 FOR yy%=0 TO dy%
 1940 FOR xx%=0 TO dx%
 1950 a%(xx%,yy%,dep%+1)=a%(xx%,yy%,dep%
)
 1960 NEXT:NEXT
 1970 PROCboardplot(x%,y%,dep%)
 1980 nx%=352:ny%=672
 1990 ENDPROC
 2000 :
 2010 DEF PROCerror
 2020 IF ERR=17 RUN
 2030 REPORT
 2040 PRINT " at line ";ERL
 2050 ENDPROC
 2060 ENDPROC
 2070 :
 2080 DEF PROCmessage(M$)
 2090 PRINTTAB(0,26);M$:PROCdelay(3)
 2100 PRINTTAB(0,26)SPC(39)
 2110 ENDPROC
 2120 :
 2130 DEF BROCintro
 2140 VDU28,23,23,39,0
 2150 COLOUR1:PRINTTAB(3,2);"SOLITAIRE"
 2160 PRINTTAB(3,3);"=========="
 2170 COLOUR3:PRINTTAB(0,5);"Place curso
r"
 2180 PRINTTAB(0,6);"on peg to be"
 2190 PRINTTAB(0,7);"moved and press"
 2200 PRINTTAB(0,8);"Space Bar."
 2210 PRINTTAB(0,10);"Then move cursor"
 2220 PRINTTAB(0,11);"to new position"
 2230 PRINTTAB(0,12);"and press Space"
 2240 PRINTTAB(0,13);"Bar again."
 2250 VDU26
 2260 ENDPROC
 2270 DATA 6,6
 2280 DATA 9,9,1,1,1,9,9
 2290 DATA 9,9,1,1,1,9,9
 2300 DATA 1,1,1,1,1,1,1
 2310 DATA 1,1,1,0,1,1,1
 2320 DATA 1,1,1,1,1,1,1
 2330 DATA 9,9,1,1,1,9,9
 2340 DATA 9,9,1,1,1,9,9
 2350 DATA 3,3
```

B

PMS PUBLISHER

If you want multiple printer fonts, a WYSIWYG preview facility, a Page Description Language, and more, then The Publisher from PMS could be for you. David Somers explains all.

Product	**The Publisher**
Supplier	**Permanent Memory Systems** **38 Mount Cameron Drive North,** **St. Leonards, East Kilbride,** **Scotland G74 2ES.** **Tel. (03552) 32796.**
Price	**£45.85 inc VAT and p&p.**

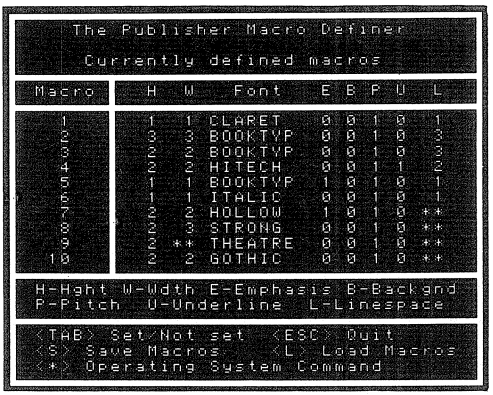
About two years ago PMS released a piece of software called Multi-Font NTQ, which allowed you to produce a number of different fonts on your printer by using its graphics facility. The Publisher is a hybrid of the NTQ system, incorporating many additions and enhancements.

It seems that most BBC micro software is now being supplied as a PAL-PROM combination, and The Publisher is no exception, consisting of a 64K PAL-PROM. It comes packed in an attractive box also containing ROM fitting instructions, an A5 size comb-bound instruction manual (circa 60 pages), utility disc, function key strip, and a Font Library booklet.

The Publisher will work on a model B, B+, Master, or Master Compact. It is compatible with second processors and Shadow RAM. Your printer must be Epson compatible, and capable of double or quad density graphics.

USAGE

Word processing is one of the most frequent tasks a computer is asked to perform. However, more and more facilities are required of word processors these days (such as WYSIWYG, the availability of different fonts, etc.), so that they are beginning to look more like mini Desk Top Publishing (DTP) packages. The trouble is that most users are fairly conservative and they are reluctant to learn all the complexities of a DTP package after they have spent many hours learning to use their word processor.



This is where The Publisher from PMS comes in, for it offers a Page Description Language (PDL). Text is prepared on your own word processor, but it has special codes inserted into it. When the document is 'printed' The Publisher takes over and recognises these special codes to produce a variety of effects. When printing text The Publisher utilises your printer's graphics capabilities. For each line the print head makes two passes to increase the resolution.

FEATURES GALORE

The Publisher takes care of all the formatting of your text completely independently of your word processor. Four pitches are available: 10, 12, or 15 cpi, and proportional spacing. The width and height of the characters can be up to sixteen times normal size and the characters can be emphasised. Justification can be either left, right, centred, or block. Two background shading patterns are available. The distance between lines (leading) can be altered and text can be put into super or subscript styles if required.

The more specific DTP facilities include the ability to produce galley lines, boxes around text, underlining in various styles, and the provision of both horizontal and vertical tabs (which makes headers and footers easier).

FONTS GALORE

As the saying goes: "Variety is the spice of life". To prove this point, The Publisher comes complete with no less than sixteen different built-in fonts. Further fonts can be added by using Font Extension ROMs, each one of which is capable of holding four complete fonts.

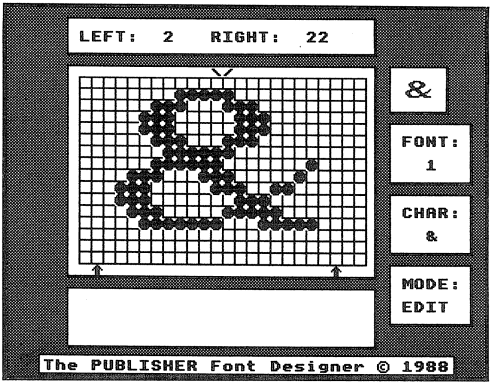
The utility disc contains a Font Editor. This allows you to design your own fonts from scratch (which is VERY time consuming), or to modify one of the existing ones. After a font has been designed, another program will convert it into a Font Extension ROM suitable for loading into sideways RAM or programming into an EPROM.

If, however, designing letter styles is not up your street, PMS have over fifty fonts (and these are increasing every week) that you can choose from and purchase. If you do not have sideways RAM or an EPROM programmer they can also supply Font Extension ROMs programmed to your requirements. Contact PMS for further details and pricing.

```
This paragraph
contains left
justified text
(ragged right
margin)

  This paragraph
  contains right
  justified text
    (ragged left
          margin)

 This paragraph
     contains
   centred text

This  paragraph
contains  block
justified  text
```

ESCAPE CODES

Just like NTQ, The Publisher works by Escape sequences. These allow you to alter various parameters, such as the current font, pitch, etc. Unlike the codes which control your printer, these are mnemonics of the functions they perform, and are quite easy to learn. e.g. ESC "H" for height.



Users of Wordwise (including Wordwise Plus and Plus II) and Interword can place these Escape codes directly in their text by using the embedded code facility. For View users it is necessary to load up a special printer driver, then 'fence' the escape codes between highlights two and one.

MACROS

When printing documents we all have our own 'style' of doing things, e.g. all the headings in a document are printed in a certain font and pitch with the text body in another. Instead of having to place sequences of (possibly) long Escape codes throughout a document, they can be replaced by a single "macro" Escape code.

The Publisher supports up to ten user-definable macros which, when called, set up the font, pitch and any required characteristics. The macro editor is entered with *MACRO, with the screen changing to a colourful MODE 7 display showing the values for all the macros. It is then a simple case of moving the cursor over the parameters and altering them as necessary.

Once a set of macros has been defined, it can then be saved to a disc or tape and loaded up when necessary. It is quite easy to build a

!BOOT file so the macros are automatically loaded.

PREVIEW TIME

One of the biggest drawbacks with other Font software is the inability to preview the output on screen. With The Publisher, a WYSIWYG display is possible with the simple command *PREVIEW. Thereafter, any output to the printer is shown on screen for checking before printing (but only when in mode 0 or 3). This can save a lot of printer paper from the rubbish bin!

THE QUICK BROWN FOX JUMPED OVER THE LAZY DOGS
THE QUICK BROWN FOX JUMPED OVER THE LAZY DOGS
THE QUICK BROWN FOX JUMPED OVER THE LAZY DOGS
THE QUICK BROWN FOX JUMPED OVER THE LAZY DOGS
THE QUICK BROWN FOX JUMPED OVER THE LAZY DOGS
THE QUICK BROWN FOX JUMPED OVER THE LAZY DOGS
THE QUICK BROWN FOX JUMPED OVER THE LAZY DOGS
THE QUICK BROWN FOX JUMPED OVER THE LAZY DOGS
THE QUICK BROWN FOX JUMPED OVER THE LAZY DOGS
THE QUICK BROWN FOX JUMPED OVER THE LAZY DOGS
THE QUICK BROWN FOX JUMPED OVER THE LAZY DOGS
THE QUICK BROWN FOX JUMPED OVER THE LAZY DOGS
THE QUICK BROWN FOX JUMPED OVER THE LAZY DOGS
THE QUICK BROWN FOX JUMPED OVER THE LAZY DOGS
'ΘΕ ΡΦΙΓΛ ΒΣΟΨΞ ΖΟΩ ΚΦΝΠΕΔ ΟΧΕΣ ΥΘΕ ΜΑ ΔΟΗΤ

When previewing text, page breaks are also indicated. This is very useful as your word processor's paging facilities cannot be relied upon when you start using some of The Publisher's more advanced features, such as multiple height text.

A problem when previewing the text on screen is that its aspect ratio is not the same as it will be on paper. However, a special 'interlaced sync & video' mode can be set so that the aspect ratio is roughly that of the printed output. OK, the screen is reduced to half its normal size and it does flicker a bit, but its works.

CONCLUSIONS

The Publisher is certainly an interesting piece of software and it is worth every penny. The price you have to pay in terms of memory is that of one page (256 bytes); Master and Master Compact users aren't affected as Private RAM is used instead. Fortunately, when you are not using The Publisher a simple star command can be issued to free this memory.

The instruction manual is very good, with separate sections detailing how to use it with View, Wordwise and Interword. The Hints & Tips section is particularly helpful in that respect.

With careful selection of its facilities you could use The Publisher to produce impressive looking documents in conjunction with your word processor. If you have a printer then you can't afford to be without The Publisher.

Users of PMS NTQ can obtain a £10 trade-in when purchasing The Publisher. Alternatively, they can either receive a Font Extension ROM containing four fonts, or ten fonts on disc. B

3-D SHADES & SHADOWS

David Williams demonstrates how a simple BBC micro can perform the shading and shadowing usually associated with much more powerful computers.

Have you ever seen those computer generated images produced by expensive computers which perform shading and shadowing? This program does just that on a BBC. If you want to try out this type of graphics, type in the program and save it away. When you run it you will see a chequer board and two circles drawn on the screen. The program will then scan the screen in the region of the board (the circles are drawn to save the program wasting its time by processing a board that will be hidden from view), to find the regions in shadow.

The shadows are drawn in at this stage. This takes a long time, about 25 minutes on a model B. After scanning the screen two spheres are drawn, in place of the original 'disc' images. It may seem to be drawing these rather slowly, but this is because it is calculating the angle of each part of the sphere with the sunlight vector, and drawing that part in a lighter colour if it is facing the sun.

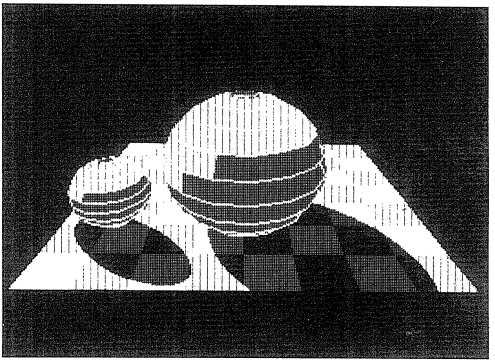
The whole program takes 35 minutes to run, but the result is well worth waiting for. If anything, the final image looks better on a monochrome display, or as a black and white printer dump. You may like to save the screen so that it can be quickly reproduced again. This can be done by inserting the line:

```
175 *SAVE image 3000 8000
```

This will save the screen under the filename "image" to disc or tape. The screen can be re-displayed by the following:

```
MODE1
VDU19,2,4,0,0,0
*LOAD image
```

You can change the direction of the sunlight vector in the program by changing the arguments in the call to PROCscan (SLx,SLy,SLz). The arguments refer to the direction ratio of the sunlight vector x:y:z. The co-ordinate system used has the x-y plane parallel to screen, and the z axis pointing away from you.



HOW IT WORKS

When the program draws the checker board, it projects its image onto the z=0 plane, using PROCperspective(F) where F is the distance of the imaginary viewpoint to the screen. After drawing the board a call to PROCDefinePlane is made to define the plane, this is done by finding the plane's DRN (A:B:C) [DRN = Direction Ratio of the Normal] to define the plane in the form:

```
Ax+By+Cz=D
```

As each pixel on the board is scanned an imaginary line is constructed from the viewpoint to the (x,y) co-ordinate of the screen (which is conveniently positioned on the z=0 plane). The intersection of this point and the plane is then calculated. Another parametric equation is formed using this point on the plane and the sunlight vector. This is substituted into the equation of a sphere. The resulting quadratic can then be solved to give the points of intersection.

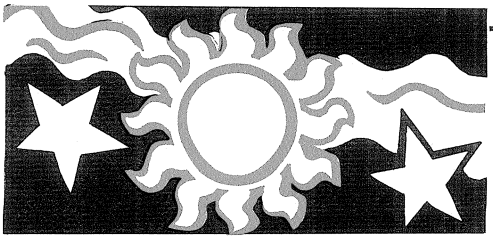
In fact, we don't need to know the intersection points as such, only if they exist or not. Given

A(x^2)+Bx+C=0, the roots are real if B*B-4*A*C>0, in which case the line intersects the sphere, the pixel on the board is in a shadow region, and so the colour of the pixel on the screen must be changed. This lengthy procedure is repeated for each point on the board. Finally the spheres are drawn by stepping around the circumference, each triangle has its DRN calculated, and the dot product is taken between the normal and the sunlight vector (the dot product routine returns the cosine of the angle).

The theory behind the display may appear quite daunting, but with this program you can ignore that if you wish and concentrate on the highly effective display produced.

```
   10 REM Program  Shadows
   20 REM Version  B0.25
   30 REM Author   David Williams
   40 REM BEEBUG   July 1988
   50 REM Program  subject to copyright
   60 :
  100 MODE1:ON ERROR GOTO 210
  110 VDU23,1,0;0;0;0;
  120 VDU29,640;512;
  130 VDU19,2,4,0,0,0
  140 DIM d(11)
  150 PROCinitrotate(0,0,-20)
  160 PROCboard
  170 PROCscan(-50,100,50)
  180 *BPRINT
  190 END
  200 :
  210 IF ERR=17 END
  220 REPORT:PRINT"at line ";ERL
  230 END
  240 :
 1000 DEF FNBallLineIntersect(a,b,c,ra,p
,q,r,s,t,u)
 1010 LOCAL d,e,f,A,B,C,f%
 1020 d=q-a:e=s-b:f=u-c
 1030 A=p*p+r*r+t*t: IF A=0 THEN =TRUE
 1040 B=2*(p*d+r*e+t*f)
 1050 C=(d*d+e*e+f*f)-(ra*ra)
 1060 =B*B-4*A*C>0
 1070 :
 1080 DEF PROCinitrotate(A,B,C)
 1090 LOCALa,b,c
 1100 a=RAD(A):d(4)=COS(a):d(5)=SIN(a):d
(6)=-d(5):d(7)=d(4)
 1110 b=RAD(B):d(8)=COS(b):d(9)=SIN(b):d
(10)=-d(9):d(11)=d(8)
 1120 c=RAD(C):d(0)=COS(c):d(1)=SIN(c):d
(2)=-d(1):d(3)=d(0)
 1130 ENDPROC
 1140 :
 1150 DEF PROCrotate
 1160 Rx=0:Ry=0:Rz=400
 1170 x=x-Rx:y=y-Ry:z=z-Rz
 1180 LOCALxs,ys
 1190 xs=x
 1200 x=(xs*d(4))+(y*d(6)):y=(xs*d(5))+(
y*d(7))
 1210 xs=x
 1220 x=(xs*d(8))+(z*d(10)):z=(xs*d(9))+
(z*d(11))
 1230 ys=y
 1240 y=(ys*d(0))+(z*d(2)):z=(ys*d(1))+(
z*d(3))
 1250 x=x+Rx:y=y+Ry:z=z+Rz
 1260 PROCperspective(800)
 1270 ENDPROC
 1280 :
 1290 DEF PROCperspective(F)
 1300 x=(F*x)/(F+z)
 1310 y=(F*y)/(F+z)
 1320 ENDPROC
 1330 :
 1340 DEF PROCmove(x,y,z)
 1350 PROCrotate:MOVEx,y
 1360 ENDPROC
 1370 :
 1380 DEF PROCdraw(x,y,z)
 1390 PROCrotate:DRAWx,y
 1400 ENDPROC
 1410 :
 1420 DEF PROCplot85(x,y,z)
 1430 PROCrotate:PLOT85,x,y
 1440 ENDPROC
 1450 :
 1460 DEF PROCboard
 1470 LOCAL X%,Y%,C%
 1480 FOR X%=-400 TO 350 STEP 100
 1490 FOR Y%=-250 TO 400 STEP 100
 1500 IF C%=3 C%=1 ELSE C%=3
 1510 GCOL0,C%
 1520 PROCmove(X%,-100,Y%):PROCmove(X%+1
00,-100,Y%):PROCplot85(X%,-100,Y%+100):P
ROCplot85(X%+100,-100,Y%+100)
 1530 NEXT:NEXT
 1540 PROCDefinePlane(-400,-100,-250,350
 -100,-250,350,-100,400)
 1550 ENDPROC
```

```
 1560 :
 1570 DEF PROCDefinePlane(x1,y1,z1,x2,y2
,z2,x3,y3,z3)
 1580 LOCAL a,b,c,d,e,f,g,h,i
 1590 x=x1:y=y1:z=z1:PROCrotate
 1600 x1=x:y1=y:z1=z
 1610 x=x2:y=y2:z=z2:PROCrotate
 1620 x2=x:y2=y:z2=z
 1630 x=x3:y=y3:z=z3:PROCrotate
 1640 x3=x:y3=y:z3=z
 1650 a=x2-x1:b=y2-y1:c=z2-z1
 1660 d=x3-x1:e=y3-y1:f=z3-z1
 1670 g=b*f-e*c:h=c*d-a*f:i=a*e-b*d
 1680 A=g:B=h:C=i:D=A*x2+B*y2+C*z2
 1690 ENDPROC
 1700 :
 1710 DEF PROCPlaneLineIntersect(A,B,C,D
,p,r,t,q,s,u)
 1720 LOCAL k
 1730 k=(D-(A*q+B*s+C*u))/(A*p+B*r+C*t)
 1740 Vx=p*k+q:Vy=r*k+s:Vz=t*k+u
 1750 ENDPROC
 1760 :
 1770 DEF PROCscan(SLx,SLy,SLz)
 1780 LOCALx%,y%,col%,test%,Vx,Vy,Vz
 1790 PROCcircle(0,0,100,200)
 1800 PROCcircle(-350,-100,100,100)
 1810 FOR x%=-400TO555STEP4
 1820 FOR y%=-50TO-430STEP-4
 1830 col%=POINT(x%,y%)
 1840 IF col%=0 OR col%=2 GOTO 1910
 1850 PROCPlaneLineIntersect(A,B,C,D,0,0
,-800,x%,y%,0)
 1860 test%=FNBallLineIntersect(0,0,100,
200,SLx,Vx,SLy,Vy,SLz,Vz)
 1870 IF test%=FALSE test%=FNBallLineInt
ersect(-350,-100,100,100,SLx,Vx,SLy,Vy,S
Lz,Vz)
 1880 IF test%=TRUE AND col%=1 col%=0
 1890 IF test%=TRUE AND col%=3 col%=2
 1900 GCOL0,col%:PLOT69,x%,y%
 1910 NEXT:NEXT
 1920 PROCsphere(0,0,100,200,SLx,SLy,SLz
)
 1930 PROCsphere(-350,-100,100,100,SLx,S
Ly,SLz)
 1940 ENDPROC
 1950 :
 1960 DEF PROCcircle(x,y,z,ra)
 1970 LOCAL a
 1980 PROCmove(x,y,z):GCOL0,2
 1990 FOR a=0 TO 2*PI+0.1 STEP 0.1
 2000 PROCmove(x,y,z):PROCplot85(x+ra*CO
S(a),y+ra*SIN(a),z)
 2010 NEXTa
 2020 ENDPROC
 2030 :
 2040 DEF PROCsphere(x,y,z,ra,Lx,Ly,Lz)
 2050 LOCAL a,b,da,db,I,s,c,sd,cd
 2060 da=0.4:db=0.3
 2070 I=0:PROCmove(x+ra*COS(I),y+ra*SIN(
I),z)
 2080 FOR b=-0.5*PI TO 0.5*PI STEP db
 2090 s=SIN(b):c=COS(b):sd=SIN(b+db):cd=
COS(b+db)
 2100 FOR I=0 TO 2*PI STEP da
 2110 sini=SIN(I):cosi=COS(I):sinida=SIN
(I+da):cosida=COS(I+da)
 2120 PROCtriangle(x+(ra*c)*cosi,y+s*ra,
z+(ra*c)*sini,x+(ra*cd)*cosi,y+sd*ra,z+(
ra*cd)*sini,x+(ra*c)*cosida,y+s*ra,z+(ra
*c)*sinida)
 2130 PROCtriangle(x+(ra*cd)*cosi,y+sd*r
a,z+(ra*cd)*sini,x+(ra*cd)*cosida,y+sd*r
a,z+(ra*cd)*sinida,x+(ra*c)*cosida,y+s*r
a,z+(ra*c)*sinida)
 2140 PROCmove(x+(ra*c)*cosi,y+s*ra,z+(r
a*c)*sini)
 2150 GCOL0,3:PROCdraw(x+(ra*c)*cosida,y
+s*ra,z+(ra*c)*sinida)
 2160 NEXT
 2170 NEXT
 2180 ENDPROC
 2190 :
 2200 DEF PROCtriangle(x1,y1,z1,x2,y2,z2
,x3,y3,z3)
 2210 PROCillumination(x1,y1,z1,x2,y2,z2
,x3,y3,z3,Lx,Ly,Lz)
 2220 PROCmove(x1,y1,z1)
 2230 PROCmove(x2,y2,z2)
 2240 PROCplot85(x3,y3,z3)
 2250 ENDPROC
 2260 :
 2270 DEF PROCillumination(x1,y1,z1,x2,y
2,z2,x3,y3,z3,Lx,Ly,Lz)
 2280 LOCALa,b,c,d,e,f,g,h,i,ans
 2290 a=x2-x1:b=y2-y1:c=z2-z1
 2300 d=x3-x1:e=y3-y1:f=z3-z1
 2310 g=b*f-e*c:h=c*d-a*f:i=a*e-b*d
 2320 temp1=(g*g+h*h+i*i):temp2=(Lx*Lx+L
y*Ly+Lz*Lz)
 2330 IF temp1*temp2=0 THEN ans=0 ELSE a
ns=(g*Lx+h*Ly+i*Lz) /SQR(temp1*temp2)
 2340 IF ans>0 GCOL0,1 ELSE GCOL0,2
 2350 ENDPROC
```



ASTROLOGICAL BIRTH CHARTS

Donald Tattersfield offers a program to draw your very own personal birth chart.

Whether you believe in the predictions of astrologers or not, there is something fascinating about the way they draw birth charts to show the position of the planets, the earth and the moon at the time of birth. Figure 1 shows such a birth chart for someone born at 11 a.m. GMT on 25 Dec 1976 in a place that is 40° 43' North and 74°West. Each sign of the zodiac occupies 30° of arc, starting at the Ram, or the First Point of Aries, referred to by astrologers as the *datum*. The inner circle is divided into twelve equal *houses*, one for each sign of the zodiac. These are located anticlockwise from a line called the *ascendant*, the position of which depends not only on the time of your birth but also on the geographical

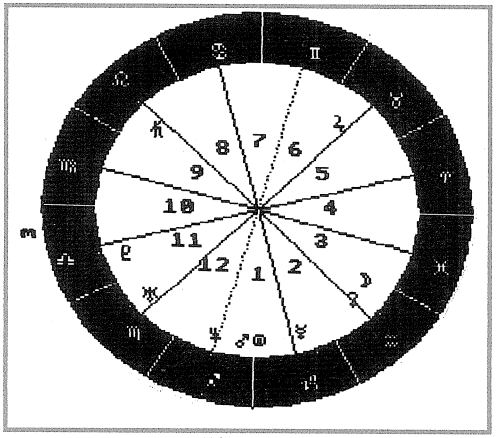


Figure 1

latitude and longitude of the place of birth. The position of the *mid-heaven* 'm' or *medium coeli* as the astrologers call it, is also important in the interpretation of the birth chart.

The program given here will draw a birth chart for a particular time and place of birth. Any budding amateur astrologers can then use this to predict the future! It is not possible in this short article to explain the ins and outs of astrology, but there are several good books on the subject. For beginners, I recommend *Astrology*, by Sheila Geddes, which is part of the Macdonald Guidelines series. A more advanced work is *Practical Astronomy with your Calculator* by Peter Duffett-Smith, published by Cambridge University Press.

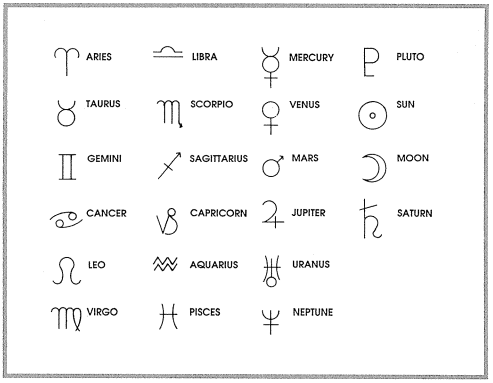


Figure 2

The program should first be typed in and saved. When run, you are prompted to enter the latitude and longitude of the place of birth. These values should be entered as degrees and minutes, separated by either Return or a comma. For longitudes west of Greenwich, and latitudes south of the equator, both the degrees and minutes should be entered as negative values. For example, 40° 43'S is -40 degrees and -43 minutes. The best way to find out the latitude and longitude of a particular place in the UK is to use an ordnance survey map. After this, you should enter the date and time of birth. The date should be specified as three numbers separated by a Return or a comma. For example, 19th February 1967 would be entered as 19,2,1967. The time must be adjusted to Greenwich Mean Time (GMT). This means that any time difference, and any local seasonal variations, should be taken into account.

When all the information has been entered, the program prints out the sidereal time of birth, the longitudes of the planets, the sun, the moon, the ascendant and the mid-heaven. All these

figures will be of significance in interpreting the birth chart. The birth chart itself is then drawn as in Figure 1. Note that the actual positions of the planets are the small dots just outside the inner circle. The ascendant and mid-heaven are shown by dots just inside the outer circle. The meaning of the symbols is shown in Figure 2. The only planet not shown is the Earth. This is because the Earth is taken to be at the centre, with all longitudes calculated as seen from the Earth. It is further assumed that all the planets orbit in the plane of the ecliptic, which while not strictly true, is a good enough approximation. The orbits of the planets are stored as data at the end of the program.

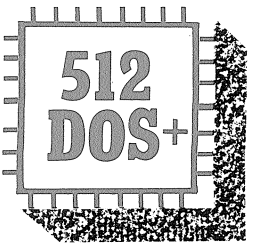
If you want to produce a printout of the birth chart, the commands to call a suitable screen dump routine can be inserted between lines 1790 and 1800. Happy fortune telling!

```
   10 REM Program BirthCharts
   20 REM Version B 1.0
   30 REM Author  Donald Tattersfield
   40 REM BEEBUG  July 1988
   50 REM Program subject to copyright
   60 :
  100 ON ERROR MODE7:REPORT:PRINT" at li
ne ";ERL:END
  110 MODE 7:VDU23,1,0;0;0;0;
  120 FOR I=5 TO 20
  130 PRINT TAB(1,I) CHR$(132);CHR$(157)
;CHR$(131);TAB(16,10);"BIRTHCHART"
  140 NEXT I
  150 PRINT TAB(6,15);CHR$(129);"Donald
Tattersfield 1988"
  160 A=INKEY 100:CLS:PRINT
  170 DIM P(9),CY(11),W(11),AM(9),O(9)
  180 DIM S(9),ep(11),l(9),nu(9),r(9)
  190 DIM dr(9),sr(9),sd(9),cd(9)
  200 DIM pr(9),ps(9),lr(9),ld(9)
  210 DIM rd(9),N(11),lm(9),ml(9)
  220 DIM le(11),LR(11),PTS(100,2)
  230 FOR Q=1 TO 9
  240 READ P(Q),CY(Q),W(Q),AM(Q),O(Q),S(
Q),ep(Q)
  250 NEXT Q
  260 READ CY(10),W(10),ep(10)
  270 READ Lo,Po,No,Io:RESTORE:PRINT
  280 CC=180/PI:EP=23.4522:RC=0.013
  290 PRINT "* FOR PLACE OF BIRTH *"'
  300 INPUT "LATITUDE NORTH (degrees,min
utes)"',dg,mi
  310 IF dg*mi<0 THEN VDU 7:CLS:GOTO 300
  320 IF ABS(dg)>90 OR ABS(mi)>=60 THEN
VDU 7:CLS:GOTO 300
  330 PRINT:ph=FNgetdeg
  340 INPUT "LONGITUDE EAST (degrees, mi
nutes)"',dg,mi
  350 IF dg*mi<0 THEN VDU 7:CLS:GOTO 340
  360 IF ABS(dg)>360 OR ABS(mi)>=60 THEN
 VDU 7:CLS:GOTO 340
  370 LB=FNgetdeg
  380 PRINT '"* FOR TIME OF BIRTH *"'
  390 INPUT "DAY,MONTH,YEAR"',J,M,Y
  400 IF M>12 OR J>31 THEN VDU 7:CLS:GOT
O 390
  410 INPUT '"GMT (hour,minute)"',ho,mi
  420 IF ho>23 OR mi>=60 THEN VDU 7:CLS:
GOTO 410
  430 IF ho<0 OR mi<0 THEN VDU 7:CLS:GOT
O 410
  440 PRINT:JJ=J+(ho+mi/60)/24:PROCsider
  450 J=JJ:PROCcalc:d=(I-2442412)+(D-0.5
)
  460 PRINT "LONGITUDES  Mercury....Plut
o (degrees)"'
  470 PRINT CHR$(134);"Please wait"'
  480 FOR Q=1 TO 9
  490 N(Q)=(360/365.25)*(d/(P(Q)))
  500 N(Q)=FNround(N(Q))
  510 l(Q)=N(Q)+(360*CY(Q)/PI)*SIN((N(Q)
+ep(Q)-W(Q))/CC)+ep(Q)
  520 N(Q)=FNround(N(Q)):nu(Q)=l(Q)-W(Q)
  530 r(Q)=AM(Q)*(1-CY(Q)*CY(Q))/(1+CY(Q
)*COS(nu(Q)/CC))
  540 dr(Q)=(l(Q)-O(Q))/CC:sr(Q)=S(Q)/CC
  550 sd(Q)=SIN(dr(Q)):cd(Q)=COS(dr(Q))
  560 pr(Q)=ASN(SIN(dr(Q))*SIN(sr(Q)))
  570 ps(Q)=pr(Q)*CC
  580 lr(Q)=ATN(TAN(dr(Q))*COS(sr(Q)))+O
(Q)/CC
  590 IF sd(Q)<>0 AND cd(Q)<0 THEN lr(Q)
=lr(Q)+PI
  600 IF sd(Q)<0 AND cd(Q)>0 THEN lr(Q)=
lr(Q)+2*PI
  610 ld(Q)=lr(Q)*CC:ld(Q)=FNround(ld(Q)
)
  620 rd(Q)=r(Q)*COS(pr(Q))
  630 NEXT Q
  640 FOR Q=1 TO 9
  650 lm(Q)=lr(3)-lr(Q):ml(Q)=lr(Q)-lr(3
)
  660 IF Q<3 THEN le(Q)=180+ld(3)+ATN((r
d(Q)*SIN(lm(Q)))/(rd(3)-rd(Q)*COS(lm(Q))
))*CC
  670 IF Q>3 THEN le(Q)=ATN((rd(3)*SIN(m
l(Q)))/(rd(Q)-rd(3)*COS(ml(Q))))*CC+ld(Q
)
  680 le(Q)=FNround(le(Q))
  690 PRINT "le(";Q;")";" ";le(Q)
  700 NEXT Q
  710 PRINT 'CHR$(134);"Press any key to
```

```
 continue":KEYHIT=GET
  720 N(10)=(360/365.25)*d
  730 N(10)=FNround(N(10))
  740 Md=N(10)+ep(10)-W(10)
  750 Md=FNround(Md):Mr=Md/CC
  760 Es=360*CY(10)*SIN(Mr)/PI
  770 le(10)=N(10)+Es+ep(10)
  780 le(10)=FNround(le(10))
  790 PRINT '"LONGITUDE Sun (degrees)";"
 ";le(10)'
  800 l=360*d/27.3217+Lo:l=FNround(l)
  810 Mm=l-(360*d)/(365.25*8.85)-Po
  820 Mm=FNround(Mm)
  830 N=No-(360*d)/(365.25*18.61)
  840 N=FNround(N)
  850 Ev=1.274*SIN((2*(l-le(10))-Mm)/CC)
  860 Ae=0.186*SIN(Mr):A3=0.37*SIN(Mr)
  870 mm=Mm+Ev-Ae-A3
  880 Ec=6.289*SIN(mm/CC):ld=l+Ev-Ae+Ec
  890 V=0.658*SIN(2*(ld-le(10))/CC)
  900 lt=V+ld:Nd=N-0.16*SIN(Mr)
  910 Nm=(lt-Nd)/CC:sd=SIN(Nm)
  920 cd=COS(Nm)
  930 Lm=ATN(TAN(Nm)*COS((Io)/CC))+Nd/CC
  940 IF sd<>0 AND cd<0 THEN Lm=Lm+PI
  950 IF sd<0 AND cd>0 THEN Lm=Lm+2*PI
  960 LD=Lm*CC:LD=FNround(LD)
  970 PRINT '"LONGITUDE Moon (degrees)";
" ";LD''
  980 e=EP-RC*(Y-1900)/100:e=e/CC
  990 aA=ATN(-COS(TR)/(SIN(TR)+TAN(ph)*T
AN(e)))
 1000 HA=TR-aA
 1010 IF HA<PI THEN aA=aA+PI
 1020 td=TAN(e)*SIN(aA)
 1030 lA=ATN(TAN(aA)*COS(e)+td*SIN(e)/CO
S(aA))
 1040 IF HA<PI THEN lA=lA+PI
 1050 lA=lA*CC:PRINT "LONGITUDE Ascendan
t (degrees)";" ";lA''
 1060 aM=TR:td=TAN(e)*SIN(aM)
 1070 lM=ATN(TAN(aM)*COS(e)+td*SIN(e)/CO
S(aM))
 1080 lM=lM*CC:IF lM>lA OR lM<lA-180 THE
N lM=lM+180
 1090 lM=lM-INT(lM/360)*360
 1100 PRINT "LONGITUDE Mid-heaven (degre
es)";" ";lM'
 1110 PRINT CHR$(134);"Press any key to
continue"
 1120 KEYHIT=GET
 1130 MODE 4:VDU23,1,0;0;0;0;
 1140 X=600:Y=500:R=300
 1150 R=R+100:RO=R-100:RZ=RO+45:RP=RO-50
 1160 VDU 19,1,3,0,0,0:GCOL 0,1
 1170 PROCCIRCLE(X,Y,R):GCOL 0,0
 1180 PROCCIRCLE(X,Y,RO)
 1190 VDU 23,224,0,20,42,42,8,8,8,0
 1200 VDU 23,225,66,66,60,24,36,36,36,24
 1210 VDU 23,226,0,124,40,40,40,40,124,0
 1220 VDU 23,227,120,148,146,96,12,146,8
2,60
 1230 VDU 23,228,24,36,66,66,36,40,169,7
0
 1240 VDU 23,229,0,170,85,85,82,85,85,5
 1250 VDU 23,230,24,36,36,231,0,231,0,0
 1260 VDU 23,231,84,170,42,42,42,42,2,3
 1270 VDU 23,232,0,14,6,74,48,48,72,128
 1280 VDU 23,233,6,10,10,100,170,33,49,3
4
 1290 VDU 23,234,0,42,84,0,0,42,84,0
 1300 VDU 23,235,130,68,40,254,40,40,68,
130
 1310 AS=PI/6:GCOL 0,1
 1320 FOR Q=0 TO 11:MOVE X,Y
 1330 DRAW X+RO*COS(Q*AS+lA/CC),Y+RO*SIN
(Q*AS+lA/CC)
 1340 NEXT Q:VDU 5:GCOL 0,1
 1350 FOR I=0 TO 11
 1360 MOVE 300+135*COS((30*I+lA+15)/CC),
520+135*SIN((30*I+lA+15)/CC)
 1370 PRINT I+1:NEXT I:GCOL 0,0
 1380 FOR Q=0 TO 11
 1390 MOVE X+RO*COS(Q*AS),Y+RO*SIN(Q*AS)
 1400 DRAW X+R*COS(Q*AS),Y+R*SIN(Q*AS)
 1410 NEXT Q:VDU 5
 1420 FOR Q=0 TO 11
 1430 IF Q*AS>PI/2 AND Q*AS<PI THEN Z=40
 ELSE Z=0
 1440 MOVE X+(RZ+Z)*COS(Q*AS+PI/12),Y+(R
Z+Z)*SIN(Q*AS+PI/12)
 1450 PRINT CHR$(224+Q):NEXT Q
 1460 VDU 23,236,68,68,56,68,56,16,56,16
 1470 VDU 23,237,56,68,68,40,16,16,56,16
 1480 VDU 23,239,0,14,6,10,48,72,72,48
 1490 VDU 23,240,48,8,8,8,16,62,4,4
 1500 VDU 23,241,40,48,32,120,168,40,40,
36
 1510 VDU 23,242,146,84,124,84,84,146,40
,16
 1520 VDU 23,243,84,84,84,56,16,124,16,0
 1530 VDU 23,244,56,36,36,36,56,32,32,60
 1540 VDU 23,245,0,120,132,180,180,132,1
20,0
 1550 VDU 23,246,224,80,40,40,40,80,224,
0
 1560 VDU 23,247,128,0,0,0,0,0,0,0
 1570 le(11)=LD:GCOL 0,1
 1580 FOR Q=1 TO 11:LR(Q)=le(Q)/CC
 1590 MOVE X-20+RP*COS(LR(Q)),Y+RP*SIN(L
R(Q))
 1600 IF Q<>3 THEN PRINT CHR$(235+Q)
 1610 NEXT Q
 1620 FOR Q=1 TO 11
```

```
 1630 IF POINT(X+(RO+10)*COS(LR(Q)),Y+(R
O+10)*SIN(LR(Q)))=1 THEN GCOL 0,0 ELSE G
COL 0,1
 1640 MOVE X+(RO+10)*COS(LR(Q)),Y+(RO+10
)*SIN(LR(Q))
 1650 IF Q<>3 THEN PRINT CHR$(247)
 1660 NEXT Q:MOVE X,Y
 1670 PLOT 21,X+RO*COS(lA/CC),Y+RO*SIN(l
A/CC)
 1680 MOVE X,Y
 1690 PLOT 21,X-RO*COS(lA/CC),Y-RO*SIN(l
A/CC)
 1700 VDU 30:VDU 5
 1710 MOVE X+(R-10)*COS(lA/CC),Y+(R-10)*
SIN(lA/CC)
 1720 IF POINT(X+(R-10)*COS(lA/CC),Y+(R-
10)*SIN(lA/CC))=1 THEN GCOL 0,0 ELSE GCO
L 0,1
 1730 PRINT CHR$(247):MOVE X+(R-20)*COS(
lM/CC),Y+(R-20)*SIN(lM/CC)
 1740 IF POINT(X+(R-20)*COS(lM/CC),Y+(R-
20)*SIN(lM/CC))=1 THEN GCOL 0,0 ELSE GCO
L 0,1
 1750 PRINT CHR$(247):GCOL 0,1
 1760 MOVE X+(R+40)*COS(lA/CC),Y+(R+40)*
SIN(lA/CC)
 1770 PRINT CHR$(97)
 1780 MOVE X+(R+45)*COS(lM/CC),Y+(R+45)*
SIN(lM/CC)
 1790 PRINT CHR$(109)
 1800 VDU 30:INPUT "Another birthchart",
Q$
 1810 MODE 7:IF Q$="Y" THEN 230
 1820 END
 2000 DEF FNround(A)
 2010 =A-INT(A/360)*360
 2020 DEF FNgetdeg
 2030 =(dg+mi/60)/CC
 2040 DEF PROCsider
 2050 LO=LB*CC/15:PROCcalc
 2060 N=JD-2415384.5:n=N/3600
 2070 T=6.59730556+236.555362*n
 2080 Mn=(ho*60+mi)
 2090 t=0.00069634577*Mn*24
 2100 L=T+t:L=L-INT(L/24)*24:LL=L+LO
 2110 LL=LL-INT(LL/24)*24
 2120 PRINT "LOCAL SIDEREAL TIME (hours)
";TAB(25);LL'
 2130 TR=LL*15/CC
 2140 ENDPROC
 2150 DEF PROCcalc
 2160 IF M>2 THEN 2200
 2170 I=365*Y+INT((Y-1)/4)-INT((Y-1)/100
)+INT((Y-1)/400)+1721059
 2180 D=31*(M-1)-INT(M-1)+J+0.5
 2190 GOTO 2220
 2200 I=365*Y+INT(Y/4)-INT(Y/100)+INT(Y/
400)+1721059
 2210 D=31*(M-1)-INT((M-1)*0.4+2.7)+J+0.
5
 2220 I=I+INT(D):D=D-INT(D)
 2230 PRINT:JD=I+D
 2240 ENDPROC
 2250 DEF PROCCIRCLE(X,Y,R)
 2260 DA=2*PI/72
 2270 CQ=COS(DA):SQ=SIN(DA)
 2280 PTS(1,1)=X+R:PTS(1,2)=Y
 2290 PLOT 69,PTS(1,1),PTS(1,2)
 2300 FOR J=2 TO 72
 2310 XD=PTS(J-1,1)-X
 2320 YD=PTS(J-1,2)-Y
 2330 PTS(J,1)=X+XD*CQ-YD*SQ
 2340 PTS(J,2)=Y+XD*SQ+YD*CQ
 2350 MOVE X,Y
 2360 PLOT 85,PTS(J,1),PTS(J,2)
 2370 NEXT J
 2380 PTS(72,1)=PTS(1,1)
 2390 PTS(72,2)=PTS(1,2)
 2400 MOVE X,Y
 2410 PLOT 85,PTS(72,1),PTS(72,2)
 2420 ENDPROC
 2430 DATA 0.2409,0.2056,77.06645
 2440 DATA 0.3871,48.03493,7.00427
 2450 DATA 320.6631
 2460 DATA 0.6152,0.006785,131.2193
 2470 DATA 0.7233,76.4548,3.3944
 2480 DATA 310.9745
 2490 DATA 1.00004,0.016720,102.5104
 2500 DATA 1.00000,0,0
 2510 DATA 99.5343
 2520 DATA 1.8809,0.09338,335.5988
 2530 DATA 1.5237,49.3647,1.8498
 2540 DATA 249.6292
 2550 DATA 11.8622,0.04860,13.9199
 2560 DATA 5.202804,100.19608,1.30450
 2570 DATA 355.2141
 2580 DATA 29.4577,0.05563,92.5583
 2590 DATA 9.5384,113.4384,2.4893
 2600 DATA 104.1728
 2610 DATA 84.01247,0.04725,170.2547
 2620 DATA 19.1819,73.8728,0.7732
 2630 DATA 205.7829
 2640 DATA 164.7956,0.008586,44.4059
 2650 DATA 30.05796,131.5051,1.7724
 2660 DATA 249.9146
 2670 DATA 246.378,0.2461,224.2580
 2680 DATA 39.2998,109.9965,17.1445
 2690 DATA 202.3345
 2700 DATA 0.01672,282.5104,279.04147
 2710 DATA 124.8756,145.9601,248.6441,5.
1453
```

B



Robin Burton presents the first of an occasional series for users of the 512 co-processor.

The 512 co-processor, which for a reasonable cost, allows a Model B or a Master 128 to run IBM PC software, has proved a popular upgrade among users. With the current special offers on the 512 board (see the News section in this issue), the number of users can only grow. However, the support for 512 users, both in the form of books, and coverage in BEEBUG and other magazines, has been limited. Judging by the letters received recently, many people would like to see this situation changed, and this is your chance.

The size, frequency and content of this column depends on you. We wont to know what you want to read about. With such a vast range of software available we cannot cover everything, so your input is vital. There are no restrictions: anything to do with the 512 will be considered.

For example, have you upgraded to DOS+ 2.1? Did it solve any problems? Did it cause any? Which packages do you use? Do you have any hints or tips you would like to pass on to other users? Have you had any particular problems? If you have and you have solved them, how? Are there any user groups out there? We know the Big Ben Club in Holland has a very active 512 section of about 80 members, and there is a group run in the Cambridge area by Andy Smith from Acorn, but what about the rest of us?

I'll start the ball rolling with an example of using the batch facility. It's designed to make life easier, but for many users it is just a source of confusion. You might be aware that batch files are similar to the Beeb's !BOOT or EXEC files, but they can do very much more, because they operate at a completely different level.

Batch files are text files which contain one or more commands of the type you could enter through the keyboard, for example to catalogue a disc, or copy a file. In practice though, you can regard batch files as a means of defining complete self contained functions. They work by stringing simpler commands together, to be executed in sequence within the framework of a very simple 'language'.

DOS+ reserves certain file extensions to indicate file types, so a '.cmd' extension means a command file, '.exe' means an executable file, while '.bat' indicates a batch file. If any command line is not known to DOS+, it looks on disc for a file of that name, and if there is one it takes the action indicated by the file extension. This is why a command error produces 'Bad command or filename', DOS+ can't tell the difference. A '.bat' extension means process the commands in this file automatically line by line. The 512 User Guide does mention batch files, but only in passing, and it does not make any attempt to show their power.

We will use word processing as an example, but the techniques can be applied to virtually any application. Our starting point is a system which has just been started up, so all you have on the screen is the DOS+ message and the A> prompt. The batch file must therefore carry out all the tasks that would otherwise require laborious and error-prone keyboard entry.

To create a batch file, you must be able to enter a text file that contains just printable ASCII characters, with no control codes. Quite a few word processors are able to store text in this way, and those which don't, generally have an option to save plain text. Obviously, the manual for your particular word processor will give more details. Another possibility is to use a text editor such as ED, which is supplied with the system and documented in the 512 User Guide. As an alternative, you could enter the batch file using your favourite Beeb editor or word processor, and then use the 'getfile' utility to transfer it to DOS+ format.

For our example job we will turn the interlace off, select the screen mode and colour and load our word processor software together with an entered filename of our choice. On completion of the edit we also want to make a backup of what we have been working on. Here are the major differences from BBC EXEC files. Firstly a batch file can run the entire session from beginning to end, and on leaving the application it can still be in control. This means that the application actually can be run from within the batch file, rather than simply being started by it. Secondly we can supply variable information, (e.g. filenames), with the initial command. These are passed to the batch file for use during execution.

We will call our batch file 'WP.BAT'. Before writing the batch file, we need to know exactly how to perform each of our operations. For our example, we will assume that our word processor is started using 'wordproc <filename>', where <filename> is a file to be loaded. Further, we will assume that the word processor software disc is in drive B, and the disc in drive A contains the text files, the batch file itself, and a command called 'fset' which is copied from the DOS+ system disc. All our text files will have a '.doc' extension, and the backup files will use '.bak'.

With the document disc in drive A and the applications disc in B, to edit the file 'TEXT.DOC' all we would need to do is enter 'wp text', and press Return. Everything else would then be automatic (except the actual word processing, of course). The wp part of the command triggers the execution of 'WP.BAT', while the filename 'text' is passed as the first parameter. DOS+ just takes the rest of the command line after the filename, and splits it into parameters, with each parameter being separated by one or more spaces. For example, if you typed 'comm hello goodbye', comm is the filename, hello is the first parameter, and goodbye is the second parameter. Within the actual batch file, the first parameter is referred to as '%1', the second as '%2' and so on. The system just substitutes the parameter from the command line for the '%n' in the file.

Our batch file would typically start something like this:

```
rem check file and screen setup
echo off
cls
if not exist A:%1.doc goto error1
star TV0,1
colour 1 2
pcscreen 3
```

The first line, starting with 'rem', is a comment line, and these can be used wherever necessary to make the file easier to read. Blank lines are also ignored, and can therefore be used to separate different parts of the file. Going through line by line: 'echo off' stops DOS+ displaying the rest of the batch file as it is executed, (rather like using VDU 21 in a Beeb EXEC file), while 'cls' simply clears the screen.

The command in the fourth line of the file shows some of the power of batch files. This is a conditional statement, which looks for a file (whose name is substituted for %1), and if it does not exist, jumps to a point elsewhere in the file. The syntax is very similar to that of IF-THEN in most languages. 'exist' is a command to check for a file existing, and in this example, the file is specified as being on drive A:, with the filename being the first parameter, and the extension being '.doc'. The 'not' reverses the result of the call to exist, and the 'goto' goes to a label called 'error1' which we will specify later. Assuming the file is found, the next three lines are executed. These execute *TV 0,1 to turn interlace off, set the text colour to green, and change mode so that the *TV can take effect.

Our batch file continues:

```
rem load application with named file
echo Loading wordproc with A:%1.doc...
b:
a:fset a:%1.doc [rw]
wordproc a:%1.doc
rem the application executes here
a:fset a:%1.doc [ro]
echo Remove disc from drive B:
echo Insert wpbackup in drive B:
pause
if not exist b:*.bak goto error2
```

This section starts by printing a message using 'echo' to confirm all is OK, and setting the default drive to B:. The next line calls the command 'fset' with the parameters being the name of our text file, and [rw]. The effect of this is to ensure both read and write access to the file before we start the word processor.

After this, the actual word processor is started up with the filename being specified in the command. At this point, the batch file will be suspended, and the word processor can be used normally. It is only when you leave the word processor to return to the system that execution of the batch file is continued.

When the batch file is continued, it first of all makes our text file read only again, and then prints two messages asking for the word processor disc to be replaced by the backup disc. The 'pause' command makes DOS+ print a prompt and wait for a key to be pressed. As a check that the correct disc has been inserted,we check that at least one file with the extension '.bak' exists already, jumping to an error routine if it doesn't.

The next three lines are:

```
echo Securing A:%1.doc to B:%1.bak
if not exist b:%1.bak goto copy
b:fset b:%1.bak [rw]
:copy
```

These print a message to show what is being done, and then check if the backup file already exists. If it does, 'fset' is used to make sure the file can be over-written. The ':copy' in the last line is a label. Whenever 'goto copy' is executed, the commands in the file are skipped over until the ':copy' is found, and then execution is continued from that point.

The last part of the body of the batch file is:

```
copy a:%1.doc b:%1.bak
a:fset b:%1.bak [ro]
echo Backup of A:%1.doc to B:%1.bak
     complete
echo Run completed OK
goto exit
```

This just copies the text file to the backup disc, and makes the backup read only. Two messages are printed to tell the user that everything is done, and the last line goes to a routine called exit, which we will define in a minute.

The rest of the file handles the errors:

```
rem errors and exit
:error1 - named input file missing
echo %1.doc is not on drive A:
echo Load abandoned!
goto exit

:error2 no .bak files in drive A:
echo Disc in drive B is not a backup disc
echo ************************************
echo *                                  *
echo * Backup of A:%1.doc HAS FAILED!!! *
echo *                                  *
echo ************************************

:exit
a:
```

These lines just print warning messages and goto the exit routine. The exit routine simply re-selects drive A and exits at the end of the file.

Any errors that occur during execution of a batch file will cause it to terminate. You can also leave a batch file when it is paused by pressing Ctrl-C. If necessary, batch files can be nested, with one calling another by specifying its name, and execution returning to the first one when the second has finished.

If you are new to batch files, it is worth trying the example given here. Just enter the file as one long block, and make any modifications necessary for your particular word processor.

That's all for this month. If you have any suggestions for future 512 articles, please write in and tell us about them. B

REDUCED PRICES

Acorn has recently announced substantial price reductions on the 512 Co-processor for the model B and Master 128. See this month's news pages for further details.

Matrices In Basic

Jan Stuurman extends Basic to provide a range of matrix operations on the BBC micro.

As anybody who has studied maths to O-Level standard will know, a matrix is a collection of values organised in rows and columns. For example:

```
1   2
3   4
5   6
```

is a matrix containing six values, or 'elements'. This matrix is said to be a 3 by 2 matrix, because it has three rows and two columns.

An obvious way of storing a matrix in Basic is in the form of an array. For example, the matrix given above could be stored in an array using the instructions:

```
DIM A(1,2)
A(0,0)=1:A(1,0)=2
A(0,1)=3:A(1,1)=4
A(0,2)=5:A(1,2)=6
```

Note that when the array is dimensioned, the number of columns is given first, and then the number of rows. Also, the values in the DIM are one less than the number of rows or columns, because in Basic array subscripts start at zero. Many people forget the zero subscript, and start from one. While this will not normally cause any problems, it may do with the routines in this article. You should therefore start all subscripts at zero.

However, when a matrix is stored in memory as an array, any operations performed on that matrix have to be done element by element. For example, to set all the elements of a matrix to zero you must reference each element one at a time, and zero it. This becomes even more involved when doing things such as adding two matrices, or multiplying two matrices together.

This is all solved by the extensions to Basic provided by the programs given in this, and the next issue of BEEBUG. These add a new statement, MAT, to Basic, which can be used in different ways to perform a number of very useful matrix operations.

The program given in listing 1 should be entered, and saved to disc before running it. As it is listed, the program is suitable for Basic II users only. People with Basic I should substitute the lines given in listing 2, for the corresponding lines of listing 1. The changes to make the program run on a Master 128 or Compact are rather more extensive, and these will be covered next month. Do not renumber the program at all, because next month's additions will have to be merged into it. When the program is run, it saves to disc a machine code program called 'MAT'. This is assembled to load at &7500, which means that the computer must be in mode 7, or a shadow mode, and the value of HIMEM should be set to &7500. If you want to assemble the program elsewhere, you can change the value of HIMEM in line 100. To install the assembled program, set HIMEM to the appropriate value, and type *MAT. This will load the extra machine code, and link it into Basic.

Matrices to use with the new commands should be dimensioned as shown above, remembering that it is:

```
DIM array(columns-1,rows-1)
```

The program restricts the array name to being a single character, and also places an upper limit on the size of the matrix, of 50 by 50. In practice, the amount of memory available might impose a lower limit on size than this.

The new statements offered are given below. The letters A, B, and C many be substituted by the appropriate array name.

`MAT A=ZER`
sets all elements of the matrix to zero.

`MAT A=CON`
sets all elements of the matrix to one.

`MAT A=IDN`
sets the matrix to the identity matrix. This is a matrix with the leading diagonal set to ones, and all other entries zero. The matrix must be

square (have the same number of columns as rows).

`MAT A=B+C`

adds the matrices B and C together, element by element, and stores the result in matrix A. A, B and C must all have the same number of rows and columns.

`MAT A=B-C`

is the same as above, except that the matrices are subtracted.

`MAT A=B*(exp) or MAT A=(exp)*B`

multiplies each element of the array by the expression in brackets.

`MAT A=B*C`

multiplies the matrices B and C together, storing the result in A. The element in the top left of the matrix A is the sum of the values obtained by multiplying the elements of the first row of the matrix B, with the corresponding elements of the first column of matrix C. This is repeated for all the rows and columns to form the rest of the elements of the result. For matrix multiplication to work, the number of columns in B must be the same as the number of rows in C. The result will have the same number of rows as B, and the same number of columns as C. Therefore matrix A must have these dimensions.

`MAT A=COP(B)`

copies the elements of matrix B into matrix A. Both matrices must be the same size.

`MAT A=TRN(B)`

copies matrix B into matrix A, but swapping rows and columns as it goes. The result in A is called the transpose of B. Clearly, A must have the same number of rows as B has columns, and the same number of columns as B has rows.

Next month we add matrix inversion and simultaneous equation solving to our matrix commands.

Listing 1

```
  10 REM Program MATRIX
  20 REM Version B 1.00
  30 REM Author  Jan Stuurman
  40 REM BEEBUG  July 1988
  50 REM Program subject to copyright
  60 :
```

```
 100 MODE7:HIMEM=&7500
 110 ON ERROR GOTO 150
 120 PROCassem
 130 OSCLI ("SAVE MAT "+STR$~vector+" "
+STR$~P%)
 140 END
 150 ON ERROR OFF
 160 MODE7:IF ERR=17 END
 170 REPORT:PRINT" at line ";ERL
 180 END
 190 :
1000 DEF PROCassem
1010 PROCvariables
1020 FOR PASS=0 TO 2 STEP 2
1030 P%=HIMEM:[OPT PASS
1040 .vector LDA &202:LDX &203
1050 CMP #start MOD256:BNE notdone
1060 CPX #start DIV256:BEQ done
1070 .notdone STA brkv:STX brkv+1
1080 LDA #start MOD256:STA &202
1090 LDA #start DIV256:STA &203
1100 .done RTS
1110 .start PHA:TYA:PHA
1120 LDY #0:LDA (&FD),Y:CMP #4
1130 BEQ mistake
1140 .notours PLA:TAY:PLA
1150 JMP(brkv)
1160 .mistake LDY &A:DEY:TYA
1170 CLC:ADC &B:STA &7E:LDA &C
1180 ADC #0:STA &7F:JSR nxtwrd
1190 BCS notours
1200 DEY:TYA:CLC:ADC &A:STA &A
1210 PLA:PLA:PLA:PLA:PLA:PLA:PLA
1220 JMP (&7E)
1230 .nxtwrd:LDX #0:.nxt1:LDY #0:.nxt2
1240 LDA (&7E),Y:STA &72:LDA table,X
1250 CMP #&FF:BEQ not:CMP #0
1260 BEQ command:CMP &72:BEQ retest
1270 .nxt3 INX:LDA table,X:BNE nxt3
1280 INX:INX:INX:JMP nxt1:.retest
1290 INX:INY:JMP nxt2:.not SEC:RTS
1300 .command INX:LDA table,X:STA &7E
1310 INX:LDA table,X:STA &7F:CLC
1320 RTS
1330 .table EQUS "MAT":EQUB 0:EQUW mat
1340 EQUB 255
1350 .brkv EQUW 0
1360 .mat JSR getcha:JSR letter:STA arn
ame
1370 JSR getcha:CMP #&3D:BEQ equals
1380 JMP syntax:.equals LDX #0
1390 JSR findar:JSR getopr:JSR chkend
1400 JSR optype:JSR operat:JMP cont
1410 .letter CMP #&41:BCC arnerr
1420 CMP #&5B:BCC arnok
1430 CMP #&60:BCC arnerr
1440 CMP #&7B:BCC arnok
1450 .arnerr JMP syntax
1460 .arnok RTS
```

```
1470 .findar LDA #&28:STA arname+1
1480 LDA #(arname-1)MOD256:STA &37
1490 LDA #(arname-1)DIV256:STA &38
1500 LDA #2:STA &39:JSR schvar
1510 BNE afound:JMP varnfd
1520 .afound LDY #0:LDA (&2A),Y
1530 CMP #6:BCC dimok:JMP baddim
1540 .dimok PHA
1550 CLC:ADC &2A:STA base,X
1560 LDA &2B:ADC #0:STA base+1,X
1570 LDA #1:STA dim,X
1580 PLA:CMP #3:BEQ dm2
1590 INY:LDA (&2A),Y:STA dim,X:INY
1600 .dm2 INY:LDA (&2A),Y:STA dim+1,X
1610 RTS
1620 .getopr JSR getcha
1630 CMP #&28:BNE nobrc1:JSR getval
1640 .nobrc1 STA oper
1650 JSR getcha:STA oper+1:JSR getcha
1660 CMP #&28:BNE nobrc2:JSR getval
1670 .nobrc2 STA oper+2
1680 JSR getcha:CMP #&28:BNE nobrc3
1690 JSR getcha:STA oper+3:JSR getcha
1700 CMP #&29:BEQ brcok:JMP misbrc
1710 .nobrc3 DEC &A
1720 .brcok RTS
1730 .optype LDX #0:LDA oper+1
1740 .oloop1 CMP optab,X:BEQ opfnd
1750 INX:INX:INX:CPX #12:BCC oloop1
1760 .oloop2 LDY #0
1770 .oloop3 LDA oper,Y
1780 CMP optab,X:BNE nxtop
1790 INX:INY:CPY #3:BCC oloop3
1800 .opfnd INX:LDA optab,X:STA &37
1810 INX:LDA optab,X:STA &38:RTS
1820 .nxtop INX:LDA optab,X:BNE nxtop
1830 INX:INX:INX:LDA optab,X
1840 BPL oloop2:JMP syntax
1850 .optab EQUS "+":EQUW add
1860 EQUS "-":EQUW sub
1870 EQUS "*":EQUW mul
1880 EQUS ",":EQUW 0
1890 EQUS "ZER":EQUB 0:EQUW zer
1900 EQUS "CON":EQUB 0:EQUW con
1910 EQUS "IDN":EQUB 0:EQUW idn
1920 EQUS "COP":EQUB 0:EQUW cop
1930 EQUS "TRN":EQUB 0:EQUW trn
1940 EQUS "INV":EQUB 0:EQUW 0
1950 EQUB &FF
1960 .operat JMP (&37)
1970 .zer JSR ldfan0:BEQ fillar
1980 .con JSR ldfan1
1990 .fillar JSR initma
2000 .fill JSR store:BCC fill:RTS
2010 .idn JSR chksqr:JSR zer
2020 LDA base:STA &4B
2030 LDA base+1:STA &4C:JSR ldfan1
2040 CLC:LDA cdim:ASL A:ASL A
2050 ADC cdim:ADC #5:STA doff
```

```
2060 .iloop JSR stfam
2070 CLC:LDA &4B:ADC doff:STA &4B
2080 BCC inoc:INC &4C:.inoc
2090 DEC cdim:BNE iloop:RTS
2100 .chksqr LDA dim
2110 CMP dim+1:BNE notsqr:RTS
2120 .notsqr JMP baddim
2130 .initma
2140 LDA #0:STA off:STA off+1
2150 LDA dim+1:STA cdim
2160 .ouloop LDY #0:LDA cdim:STA dim+1
2170 .inloop CLC:TYA:RTS
2180 .store
2190 CLC:LDA base:ADC off:STA &4B
2200 LDA base+1:ADC off+1:STA &4C
2210 JSR stfam
2220 CLC:LDA off:ADC #5:STA off
2230 BCC onoc:INC off+1:.onoc
2240 DEC dim+1:BNE inloop
2250 DEC dim:BNE ouloop:SEC:RTS
2260 .getval
2270 LDA &B:STA &19:LDA &C:STA &1A
2280 LDA &A:STA &1B:JSR getnsb:PHP
2290 CPX #&29:BEQ chktyp:JMP misbrc
2300 .chktyp PLP:BEQ string:BPL integr
2310 .real JSR nmlfa:JSR stfat1
2320 LDY &1B:INY:STY &A:LDA #0:RTS
2330 .string JMP mismat
2340 .integr JSR citof:JMP real
2350 .add LDA #0:STA asflag
2360 .addsub JSR chkopr:JSR initma
2370 .asloop
2380 CLC:LDA base4:ADC off:STA &4B
2390 LDA base4+1:ADC off+1:STA &4C
2400 JSR ldfam
2410 CLC:LDA base2:ADC off:STA &4B
2420 LDA base2+1:ADC off+1:STA &4C
2430 BIT asflag:BNE subtrt
2440 .additn JSR addmfa:JMP astore
2450 .subtrt JSR subfam
2460 .astore JSR store:BCC asloop:RTS
2470 .sub LDA #&FF:STA asflag
2480 JMP addsub
2490 .chkopr LDA oper:LDX #2
2500 JSR chkar:BNE misdim:BCC misdim
2510 .chkop2 LDA oper+2:LDX #4
2520 JSR chkar:BNE misdim:BCC misdim
2530 RTS
2540 .chkar JSR letter:STA arname
2550 JSR findar:LDA #0:PHA
2560 LDA dim,X:CMP dim:BNE md1
2570 .cd2 LDA dim+1,X:CMP dim+1
2580 BEQ same:CLC
2590 .same PLA:RTS
2600 .md1 PLA:LDA #1:PHA:BNE cd2
2610 .misdim JMP baddim
2620 .mul LDA oper:BEQ nummul
2630 LDX oper+2:BNE matmul:STA oper+2
2640 .nummul JSR chkop2:JSR initma
```

```
2650 .muloop JSR ldfat1
2660 CLC:LDA base4:ADC off:STA &4B
2670 LDA base4+1:ADC off+1:STA &4C
2680 JSR mufamo:JSR store:BCC muloop
2690 RTS
2700 .matmul LDA oper:LDX #2
2710 JSR chkar:BCC misdim
2720 LDA oper+2:LDX #4
2730 JSR chkar:BNE misdim
2740 LDA dim2:CMP dim4+1:BNE misdim
2750 STA mdim
2760 CLC:LDA dim+1:ASL A:ASL A
2770 ADC dim+1:STA poff
2780 CLC:LDA mdim:ASL A:ASL A
2790 ADC mdim:STA qoff
2800 LDA base2:STA cbase
2810 LDA base2+1:STA cbase+1
2820 JSR initma
2830 .mloop1 LDA mdim:STA ldim
2840 LDA base2:STA pbase
2850 LDA base2+1:STA pbase+1
2860 LDA base4:STA qbase
2870 LDA base4+1:STA qbase+1
2880 JSR ldfan0
2890 .mloop2 JSR stfat1
2900 LDA pbase:STA &4B
2910 LDA pbase+1:STA &4C:JSR ldfam
2920 LDA qbase:STA &4B
2930 LDA qbase+1:STA &4C:JSR mufamo
2940 JSR pntmt1:JSR addmfa
2950 CLC:LDA pbase:ADC poff:STA pbase
2960 BCC noc1:INC pbase+1:.noc1
2970 CLC:LDA qbase:ADC #5:STA qbase
2980 BCC noc2:INC qbase+1:.noc2
2990 DEC ldim:BNE mloop2
3000 JSR store:BCS mready:BEQ frmout
3010 .fromin CLC
3020 LDA base2:ADC #5:STA base2
3030 BCC noc3:INC base2+1:.noc3
3040 JMP mloop1
3050 .frmout CLC
3060 LDA base4:ADC qoff:STA base4
3070 BCC noc4:INC base4+1:.noc4
3080 LDA cbase:STA base2
3090 LDA cbase+1:STA base2+1
3100 JMP mloop1
3110 .mready RTS
3120 .trn
3130 LDA oper+3:LDX #2:JSR letter
3140 STA arname:JSR find`Mar
3150 LDA dim:CMP dim2+1:BNE wrgdim
3160 LDA dim+1:CMP dim2:BNE wrgdim
3170 CLC:LDA dim:ASL A:ASL A
3180 ADC dim:STA poff:JSR initma
3190 .tloop1 LDA base2:STA pbase
3200 LDA base2+1:STA pbase+1
3210 .tloop2 LDA pbase:STA &4B
3220 LDA pbase+1:STA &4C:JSR ldfam
3230 JSR store:BCS tready:BEQ fout
```

```
3240 .fin CLC
3250 LDA pbase:ADC poff:STA pbase
3260 BCC tnoc1:INC pbase+1:.tnoc1
3270 JMP tloop2
3280 .fout CLC
3290 LDA base2:ADC #5:STA base2
3300 BCC tnoc2:INC base2+1:.tnoc2
3310 JMP tloop1
3320 .tready RTS
3330 .wrgdim JMP baddim
3340 .cop LDA oper+3:LDX #2:JSR chkar
3350 BNE wrgdim:BCC wrgdim
3360 .copy JSR initma
3370 LDA base,X:STA cbase
3380 LDA base+1,X:STA cbase+1
3390 .cloop
3400 CLC:LDA cbase:ADC off:STA &4B
3410 LDA cbase+1:ADC off+1:STA &4C
3420 JSR ldfam:JSR store:BCC cloop
3430 RTS
9000 ] NEXT
9010 ENDPROC
9020 :
10000 DEF PROCvariables
10010 arname=&70:oper=&72:asflag=&75
10020 base=&76:base2=&78:base4=&7A
10030 cbase=&7C:pbase=&7E:qbase=&80
10040 dim=&82:dim2=&84:dim4=&86
10050 cdim=&88:ldim=&89:mdim=&8A
10060 poff=&8B:qoff=&8C:off=&8E
10070 doff=&8D
10080 cont  =&8B9B:chkend=&9857
10090 syntax=&982A:getcha=&8A97
10100 schvar=&9469:varnfd=&AE43
10110 baddim=&9127:ldfan0=&A686
10120 ldfan1=&A699:stfam =&A38D
10130 getnsb=&9B29:mismat=&8C0E
10140 citof =&A2BE:ldfam =&A3B5
10150 addmfa=&A500:subfam=&A4FD
10160 nmlfa =&A303:stfat1=&A385
10170 ldfat1=&A3B2:mufamo=&A656
10180 pntmt1=&A7F5:misbrc=&AE61
10190 ENDPROC
```

* * * * * * *

Listing 2

```
10080 cont  =&8B0C:chkend=&9810
10090 syntax=&9839:getcha=&8A1E
10100 schvar=&9429:varnfd=&AE72
10110 baddim=&90D5:ldfan0=&A691
10120 ldfan1=&A6A4:stfam =&A37E
10130 getnsb=&9B03:mismat=&8B7E
10140 citof =&A2AF:ldfam =&A3A6
10150 addmfa=&A50E:subfam=&A50B
10160 nmlfa =&A2F4:stfat1=&A376
10170 ldfat1=&A3A3:mufamo=&A661
10180 pntmt1=&A7FB:misbrc=&AE90
```

MICROLINK COMMUNICATIONS PACK

We asked Peter Rochford, author of our regular Comms Spot, to review the latest software and hardware package offered by Microlink Communications.

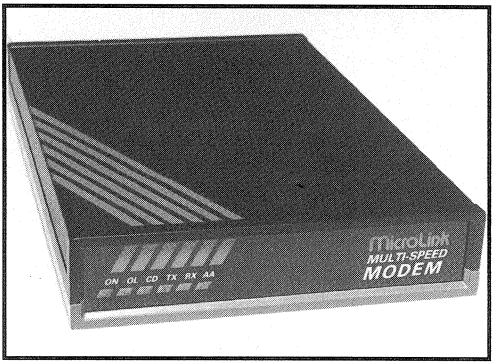
Title	Microlink Communications Package
Supplier	Microlink Communications Ltd, Europa House, Adlington Park, Adlington SK10 4NP. Tel. (0625) 879940
Price	£169.00 (inc.VAT)

Database Publications is a well known name in the field of computer journalism, in particular to Beeb users because of the Micro User magazine. Some while ago, Database entered the world of comms by launching their own closed user group on the BT Telecom Gold network, to compete in some ways with Micronet on Prestel. Now, in a similar fashion to Micronet in its early days, Database is offering a combined hardware/software package along with a subscription to its Microlink database on Telecom Gold.

The modem supplied with the package is a multi-speed Hayes compatible unit, and operates at V21 (300/300), V23 (1200/75) and the higher V22 (1200/1200) speed. It comes housed in a neat, attractive red and black case with the power supply as a separate unit ready to plug straight into a mains socket.

The rear panel of the modem has a socket for a telephone, the PSU input and a 25-way D connector for connection to your Beeb's RS423 port. A reset switch is also provided for resetting the modem's own internal software. The front panel of the modem features six LEDs indicating power-on and the status of the modem. There are no switches or any other external controls.

As well as standard Hayes compatibility, the modem has extra AT commands and S registers along with auto-dial, auto-answer and number store facilities, making it flexible, powerful and easy to use.



The modem is 'intelligent' and has its own microprocessor and operating software, much like other Hayes compatible units. It can sense both the speed of the terminal it is connected to and the speed of the host, adjusting itself accordingly. In addition, it can detect whether a dial tone is present on the line and if the number it is calling is engaged. Both of these conditions will cause the modem to report back to the terminal with an on-screen message.

To understand more about 'intelligent' modems and their many advantages you may wish to refer to the article in BEEBUG Vol.6 No.2 and the review of six intelligent modems in BEEBUG Vol.6 No.6.

THE SOFTWARE

The bundled software supplied with the package is a cut down version of Database's

own Mini Office suite on 5.25" disc. It comprises just the comms section and the word processor.

The wordprocessor is supplied to allow off-line editing of text files ready for uploading when linked to Telecom Gold. It is a remarkably competent piece of software, and bears many similarities to Wordwise. It is easy to use, and has a wealth of features and facilities that make it a real bonus.

The comms section of the software allows easy dial-up and log-on to Microlink at several different baud rates, all by means of a single key press.

Amongst the many features available while on-line are uploading of text from disc or memory buffer, output to disc or memory, or sending of all screen output to a printer. There is also a split screen facility allowing received text to be displayed in one window and outgoing text in the other. A nice addition is that the software may be used with an AMX mouse.

MICROLINK

Microlink is a closed user group on BT's Telecom Gold scrolling text Email service. The package provides free registration to Microlink along with telex validation registration and one month's connection time without time charges.

Joining Microlink is a cheap way of gaining access to, and using Telecom Gold. Having said that, you must remember that after the first month all of the normal rates apply. This involves a standing charge of 3 pounds per month and a connection charge of 3.5p per minute cheap rate and 11p per minute standard rate.

In addition to that, you will be charged 20p per month for every 2048 characters of information stored in your mail or file store on Telecom Gold. Add to all of this your telephone charges whilst online and it starts to look expensive. It can be, but using the high speed modem helps as you can download and upload text very fast and then get offline quickly. Long browsing sessions and learning to use the system are best confined to the first month's free connect time.

So what are you getting for your money? Well, access to one of the most sophisticated Email systems in the world and access to worldwide Email too. Telecom Gold has links with other international online databases, and also provides access to the Packet Switch Stream whereby you can link into overseas systems at a fraction of the cost of dialling direct.

As well as all this, Microlink itself provides many services for computer users including computer news, bulletin boards and downloadable telesoftware.

CONCLUSION

I can honestly say that I really enjoyed reviewing this package. The whole thing is well thought out and presented, with excellent documentation throughout.

Both the software and hardware are a delight to use. I will be sad to part with the modem at the end of the review. The software does lack the facility for accessing viewdata systems, but I did use the modem with both Commsoft and BEEBUG's own Command ROM to access Prestel successfully.

Whether the free registration to Microlink is worthwhile you will have to weigh-up carefully. To allow a free peek at what it offers there is a demonstration account which you can dial up and sample for yourself.

For me, I think the modem alone is worth the £169 asking price. It is an excellent piece of hardware. The inclusion of the software and the Microlink registration are a bonus to what is already a bargain package. B

PASSING ARRAYS TO FUNCTIONS AND PROCEDURES

Jan Stuurman offers a clever little technique to let you pass arrays to functions and procedures.

In all the versions of Basic up to and including Basic IV, it is not possible to pass arrays as parameters to functions and procedures. This can be a serious drawback. For example, you cannot write a procedure that will calculate the inverse of any matrix, because the procedure has to be written to be specific to just one named array.

This problem is overcome in Basic V on the Archimedes, which allows arrays to be specified as parameters, and also extends the DIM statement to provide a function that returns the number of dimensions of an array, and the size of each dimension. While it is not practical to implement a system quite like this for earlier versions of BBC Basic, it is possible to achieve a similar result.

The short machine code routine given here allows the name of an array to be passed to a procedure or function as a string parameter. The definition of the procedure is then written with reference to the same array by using the underscore character (_), followed by '%' or '$' if you are specifying an integer or string array. The underscore is accepted by Basic as a valid name, and is unlikely to be used for this purpose elsewhere in the program.

Listing 1 is a demonstration to illustrate this technique. When the program has been entered, the address at the end of line 1420 should be changed according to the version of Basic you have in your machine. The possible values are:

```
Basic I          &9429
Basic II         &9469
Basic IV & VI    &8087
```

If you are using Basic IV or VI, you must also add the line:

```
1385 INC &39
```

The demo program creates a five element real array, a five element integer array, and a five by two string array. It then assigns values to each element of the three arrays, and uses a single routine, where the name of the appropriate array is passed as a string parameter, to print out each array in turn. If you type it in and run it, you should be able to see the results.

The key parts of the example program are the procedure PROCassemble, and the CALL in line 1180. To use the array passing in your own program, you must first add the definition of PROCassemble to the program, and call this procedure at the start of the code (see line 1020). This sets up a machine code routine that is used later. Line 1180 in the example program is the important line as far as passing the array parameter is concerned, and a line similar to this should be included at the start of any procedure definition which uses this technique. This line stores the name of the array in memory (starting at location &2EE, which is used by the operating system as temporary workspace), and then calls the machine code routine assembled by PROCassemble. The remainder of the procedure definition should then refer to the array using the underscore character, as in PROCdisplay from line 1170 to 1300. Note how the procedure copes with the possibility of the three different types of array (integer, floating point and string) in lines 1230 to 1260.

The effect of executing line 1180 is to create a variable name that consists of an underscore, possibly followed by a '$' or a '%', and to make this new name point to the array specified when the procedure was called. Therefore, instead of passing the contents of the array to the procedure, you pass just its name. This is known technically as a 'Call by Reference'. The net effect of this is that if the procedure changes any elements of the array referred to by the underscore, these changes will be reflected in the original array. While this is not always what you might want, it is the same method that is used by Basic V on the Archimedes, and is much more efficient than creating a local copy of the entire array.

Another point to note is that this technique is not restricted for use within a function or

procedure, although this is possibly its best use. Indeed, each time you call the machine code, it will merely make the underscore variable name point to the specified array. This can happen anywhere in the program, although it is most useful in the context of passing arrays as parameters.

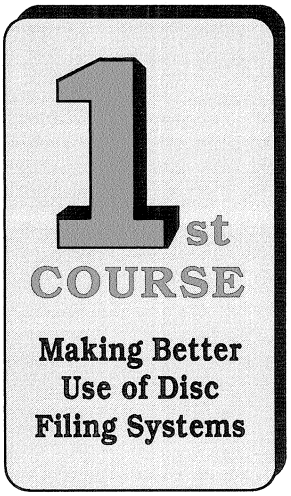
There are three restrictions to the use of this technique. Firstly, the array name passed to the routine must be no longer than 16 characters, including any $ or %, and secondly, there must be no other variables whose names start with the underscore character. In actual fact, any such variables in existence when the machine code is called will be lost, and any assigned afterwards would cause all the other variables to be corrupted. Finally, this technique cannot be used in nested routines. For example, if procedure 'A' has an array parameter, and this then calls procedure 'B' which also has an array parameter, the original array pointer will be lost.

HOW IT WORKS

The key to this array passing technique is the way in which Basic stores variables. When the machine code is executed, it calls a routine in the Basic ROM which is used to locate where a variable is stored. This routine is called with memory locations &37 and &38 pointing to the location one before the place where the variable name is stored, and with &39 containing the length of the name. Because the name is poked into locations &2EE onwards, &37 and &38 are set to &2ED. The value to store in location &39 is calculated by counting the characters of the name up to and including the '(' which is used to distinguish arrays from simple variables. The Basic ROM routine returns, in locations &2A and &2B, a pointer to the start of the actual variable in memory. For an array, this will in fact point to a block of data giving the size of the array, which is in turn followed by the elements of the array.

Once the address of the array is known, the machine code sets up a dummy variable called _, and points this to the array. Basic stores variables as a series of linked lists (see the Workshop in this issue), with a separate list for each possible first letter. This program works by making the list for names starting with an '_' point to the appropriate array.

```
  10 REM Program Array Passing
  20 REM Version B 1.0
  30 REM Author  Jan Sturrman
  40 REM BEEBUG  July 1988
  50 REM Program subject to copyright
  60 :
 100 MODE 7
 110 PROCinit
 120 PROCdemo
 130 END
 140 :
1000 DEF PROCinit
1010 DIM A(4),B%(4),C$(4
1)
1020 PROCassemble
1030 ENDPROC
1040 :
1050 DEF PROCdemo
1060 FOR I%=0 TO 4
1070 A(I%)=I%*PI
1080 B%(I%)=INT(I%*PI)
1090 C$(I%,0)=STR$(I%)+" * PI = "
1100 C$(I%,1)=STR$(I%*PI)
1110 NEXT
1120 PROCdisplay("A",4,0)
1130 PROCdisplay("B%",4,0)
1140 PROCdisplay("C$",4,1)
1150 ENDPROC
1160 :
1170 DEF PROCdisplay(array$,row,col)
1180 $&2EE=array$+"(":CALL code
1190 CLS:PRINT "Array ";array$;"()"
1200 FOR I%=0 TO row
1210 FOR J%=0 TO col
1220 PRINT TAB(5+10*J%,5+I%);
1230 type$=RIGHT$(array$,1)
1240 IF type$="%" PRINT_%(I%)
1250 IF type$="$" PRINT_$(I%,J%)
1260 IF type$<>"%" AND type$<>"$" PRINT
_(I%)
1270 NEXT:NEXT
1280 PRINT TAB(6,20)"Press space to con
tinue."
1290 REPEAT UNTIL GET=32
1300 ENDPROC
1310 :
1320 DEF PROCassemble
1330 DIM code 70
1340 FORpass=0 TO 2 STEP 2
1350 P%=code:[OPT pass
1360 LDA #&ED:STA &37:LDA #2:STA &38
1370 LDY #0:.loop INY:LDA (&37),Y
1380 CMP #&28:BNE loop:STY &39
1390 LDX #4:DEY:LDA (&37),Y
1400 CMP #&25:BEQ inc
1410 CMP #&24:BNE call:.inc INX
1420 .call STX sub+1:JSR &9469
1430 SEC:LDA &2A:.sub SBC #0:STA &4BE
1440 LDA &2B:SBC #0:STA &4BF:RTS
1450 ]NEXT:ENDPROC
```



Mike Williams gives some more advice on the use of your disc filing system.

This month I'd like to delve a little deeper into the intricacies of disc filing systems. This article will refer particularly to the DFS, but much of what I have to say applies equally to the ADFS (see additional comments later).

The DFS maintains a catalogue (maximum 31 entries on an Acorn DFS) which contains the names of the files stored on your disc, and a pointer to the position on the disc where each file starts. Since every DFS file occupies a contiguous area of the disc (that is, all the sectors are adjacent), once the start address is known the rest of the file is easily located.

In fact the DFS stores more than just a name and a pointer for each file. If you insert a DFS format disc into a drive and type *INFO *.* <Return> you will get a listing of all the files on your disc like the example shown here.

```
D.FLASH    000000 000000 000740 08F
V.Beebad1  006576 026565 0002B0 08C
V.ADRIAN1  016576 000000 000A82 081
$.BGRAPH   FF1900 FF8023 0030CE 050
$.SARAH1   000000 000000 0002F0 04D
B.POINT2   FF0E00 FF802B 000581 047
B.POINT1   FF0E00 FF802B 000577 041
$.CROSSRE  FF0E00 FF802B 002113 01F
V.SPELL    016576 000000 001BB7 003
B.German   000000 FFFFFF 000061 002
```

DFS Disc Catalogue Using *INFO *.*

Unlike the results of using *CAT, files are no longer listed in alphabetical order, but in the order in which they are located on the disc. To make matters a little confusing you will find that the first file is listed at the bottom, and the last file at the top of the list. You can see this from the numbers at the right-hand side of the list. These are in fact the addresses (sector numbers) for the start of each file, and are in hexadecimal not decimal. You should see that the first (bottom in the list) file on your disc starts at sector 2, because the first two sectors (numbered 0 and 1) are used for the catalogue itself.

The number listed immediately before the sector number in each case indicates the length of the file in bytes. Again this is given in hexadecimal. The simplest way to convert between one and the other is to get Basic to do the job for you. If as a result of using the *INFO command you see that the length of a file is given as, say, 1000 (hex), then you can convert this to decimal by typing:

```
PRINT &1000
```

and the decimal equivalent (4096) will be displayed. Likewise, if you want to convert a decimal number to hex, say 3156 for example, just type:

```
PRINT ~3156
```

to see the result (C54).

Thus when you load a file (or program), the DFS searches through the catalogue for the file name and extracts the corresponding address (sector number) and length. The DFS then goes directly to that sector on the disc and reads the file into memory, block by block, until the specified number of bytes has been found.

Despite the information about the location and length of each file on a DFS disc, there is no easy way of determining how much free space is left over, nor the extent to which this free space is fragmented (leaving gaps between files). But the simplest way to get the necessary information is to use the two utilities by Bernard Hill which were published in Vol.6 No.8, and these have been included again on this month's magazine disc and tape. The utilities implement two additional star commands, the first being *FREE which gives the number of free bytes on a disc, and secondly *MAP which shows how this free space is distributed on the disc.

LOAD AND EXECUTION ADDRESSES

The other two numbers included for each file in the *INFO listing are the *load* address and the *execution* address, again these are both in hex. In fact, you don't really need to bother about either of these most of the time. The load address indicates where in memory a file should be loaded from disc, and the execution address indicates the point at which a machine code program should start running.

For example, if you look at the load address given for any Basic program you will normally find that this has been set to the value of PAGE (usually the default for your machine) which was operative at the time that the program was saved. The execution address will usually be set to 8023 or similar depending on your version of Basic; this does not indicate where the Basic program should start execution, but marks the entry point to the Basic assembler itself. On the Master series you may find these addresses preceded by FF (hex). This is of significance only on 2nd processor systems.

Of course, the idea of an 'execution' address for a data file is pretty meaningless, but some software writers make use of this to store other information (the date when a file was last updated for example). Generally, this information can simply be ignored.

UNDERSTANDING *LOAD AND *SAVE

There are two star commands which can be useful for many purposes, viz *LOAD and *SAVE. These may be used to load a file from disc into memory, or to save a file from memory to disc. In effect, *SAVE simply saves to disc a specified area of the computer's memory. Unlike the LOAD and SAVE commands used with Basic programs, the star command equivalents may be used to load and save any files (including Basic programs if you so wish).

Before looking at some of their applications, we need to understand the basic syntax of these two commands. The simplest is *LOAD, so we'll consider that first. The shortest form of this is:

```
*LOAD <filename>
```

which will load a file from disc into memory at the address specified in the stored load address for that file. The alternative form of the command is:

```
*LOAD <filename> <address>
```

which involves specifying (in hex) the memory address at which the file is to be loaded. This overrides the load address saved in the disc's catalogue.

The syntax of the *SAVE command is more complex. The simplest form of this is:

```
*SAVE <filename> <address>+<length>
```

where *filename* is the name of the disc file to be used, *address* is the start address in memory, and *length* is the number of bytes to be saved. All values must be given in hex, but without the preceding ampersand ('&') usually associated with hex numbers. An alternative form of the command is:

```
*SAVE <filename> <add1> <add2>
```

where *add1* is the start address, and *add2* is the final address *plus 1* of the area of memory to be saved.

When a file is saved, the execution address in the catalogue is set by default to be the same as the load address, but you can specify your own choice by appending a further hex number to either form of the *SAVE command given above. It is even possible to append a further hex number known as the reload address which sets the load address in the file's catalogue entry to be different from the address in memory from which the file was saved. Our examples will use only the two simpler versions.

USING *LOAD AND *SAVE

We'll now deal with some useful applications of *LOAD and *SAVE. Although Basic programs are saved and loaded using the two Basic commands SAVE and LOAD, there are occasions when *SAVE and *LOAD may be useful. Sometimes a disc which is difficult to read may respond to *LOAD where it will not work correctly when LOAD is used. To load a program in this way just type:

```
*LOAD <progname>
```

If successful, you should then be able to list and run the program as normal, though it is advisable to type OLD <Return> so that TOP and LOMEM are correctly set (using LIST will do this automatically).

Occasions also arise when it may be useful to use *LOAD and *SAVE as a means of copying a file (you can then save the copy under a new name, unlike *COPY). To do this, use *INFO

first and note the length of the file in question. Then use *LOAD to load the file into memory specifying the current value of PAGE as the load address, and save using *SAVE in the form:

```
*SAVE <name> <address>+<length>
```

where *name* is the new name of the file, *address* the current value of PAGE, and *length* is the length of the file obtained previously. For example, if *INFO shows:

```
Olddata 0025C3 0025C3 001C32 BC3
```

we could then make a new copy with:

```
*LOAD Olddata 1900
*SAVE Newdata 1900+1C32
```

Another application of *SAVE can be a real life saver with earlier versions of View, where pressing Break leads to a complete loss of your current text file and a return to Basic. In fact, very little is lost as your text is very likely still sitting in the computer's memory. The precise action to be followed depends on the screen mode you were using. The simplest thing to do, and whatever you do must be done as soon as the loss occurs, is to save the entire contents of memory from the current value of PAGE up to the end of user RAM:

```
*SAVE MyText E00 8000
```

This assumes PAGE is at &E00 - change this as appropriate (&1900 on a model B for example). Then re-activate View and use the *read* command to read into View the file just saved. When complete, press Escape and you should see your 'lost' text. Because we saved the whole of memory you will probably find a lot of garbage at the end of the file, but this can be quickly marked and deleted to restore the original text, which should be immediately saved for safety.

As an alternative to saving the entire contents of memory, if no shadow memory is in use, just save the contents of memory from PAGE up to the start of the screen display. These start at different places as follows:

MODE	ADDRESS
0,1,2	&3000
3	&4000
4,5	&5800
6	&6000
7	&7C00

Thus if you were working in mode 3, you would use:

```
*SAVE MyText E00 4000
```

The read the text back into View and tidy up as before.

Another application for these star commands is in saving and re-displaying screens. This will only work in non-shadow modes, and you need to refer to the same screen addresses given above. In every case the end of screen memory is at &7FFF. Thus to save a mode 2 screen write:

```
*SAVE Screen 3000 8000
```

and the screen display will be saved under the name *Screen*. This can be used in a program, but even better is to use an OSCLI command which then allows several screens to be saved with different names. For example, using:

```
OSCLI("SAVE Screen"+STR$(s%)+" 3000 80
00")
```

will save a mode 0, 1, or 2 screen with a filename determined by the current value of s% (e.g. Screen1, Screen2 etc.).

Any screen which has been saved in this way can be easily re-displayed using:

```
*LOAD <screen-name>
```

You will need to select the correct mode first, and if the logical colours of the original screens had been altered using VDU19, a similar VDU19 call will be needed when the screens are re-displayed or they will appear in the default colour settings. The following code will re-display a sequence of screens saved as above (Screen1, Screen2, etc.):

```
100 MODE 2
110 s%=1
120 REPEAT
130 OSCLI("LOAD Screen"+STR$(s%)+" 3000")
140 G=GET
150 s%=s%+1
160 UNTIL s%>n%
```

Note that n% is the number of screens to be displayed. Pressing any key will move on to the next screen.

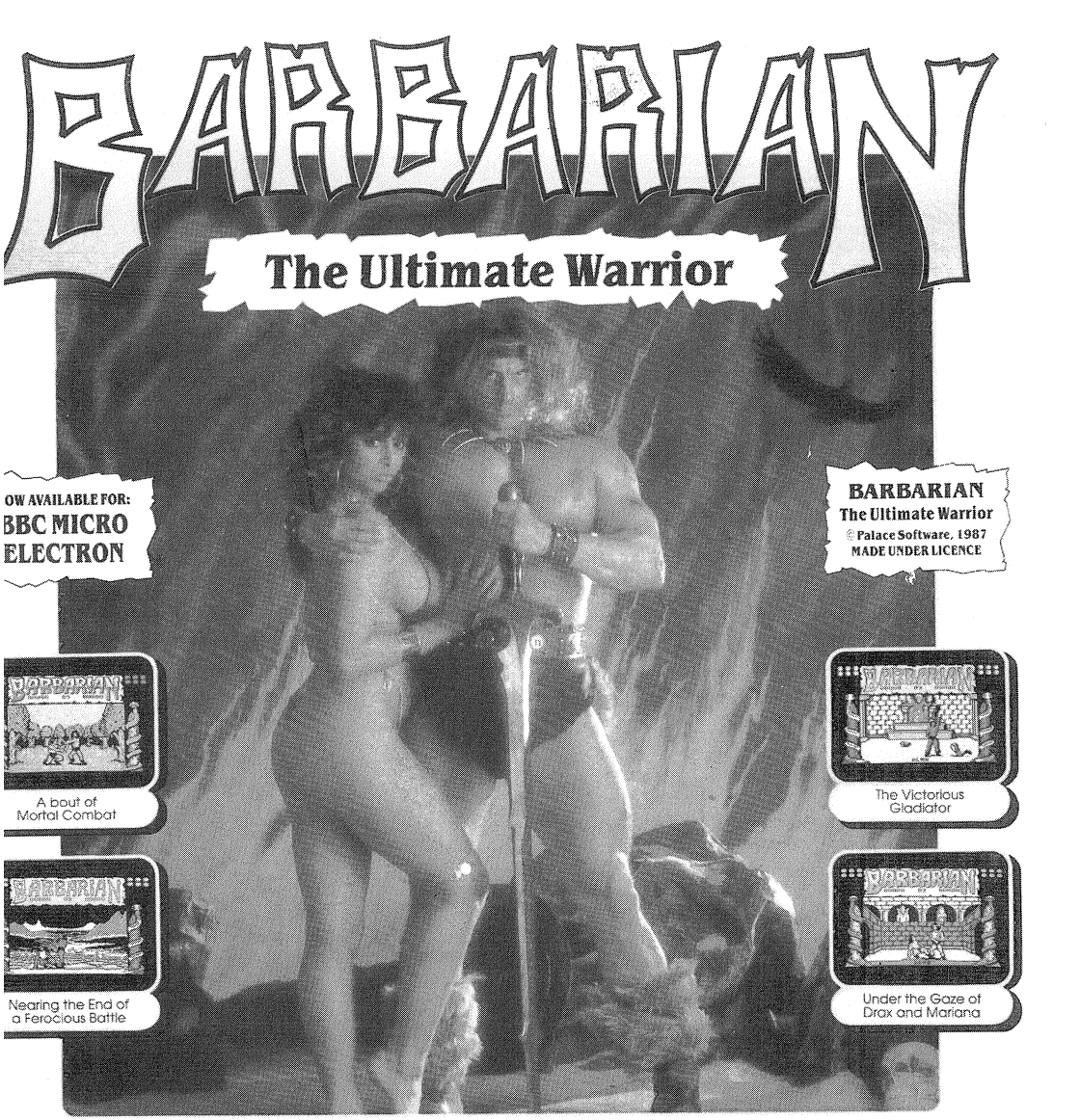
ADFS USAGE

Most of what I have described will work in almost the same way on ADFS systems. The way in which files are stored on disc is different, but a file still has a load address, an execution address, a length and a sector number, all given in hex in that order in response to the *EX command. The two commands *FREE and *MAP are built into the ADFS (and 1770 DFS), but have similar meanings. The commands *LOAD and *SAVE may be used exactly as described.

I hope these two First Course articles have helped to clarify some of the more confusing points of disc filing systems. B

BARBARIAN

The Ultimate Warrior



OW AVAILABLE FOR:
BBC MICRO
ELECTRON

BARBARIAN
The Ultimate Warrior
© Palace Software, 1987
MADE UNDER LICENCE

A bout of Mortal Combat

Nearing the End of a Ferocious Battle

The Victorious Gladiator

Under the Gaze of Drax and Mariana

THE ULTIMATE WARRIOR • THE ULTIMATE GAME

At last, BARBARIAN, the most realistic and exciting of sword-fighting games reaches the BBC Micro and Electron.

One or two players—fight against the computer or a friend.

The evil sorcerer Drax has abducted the beautiful Princess Mariana to satiate his nefarious desires. A powerful warrior is sought to vanquish against Drax's demonic guardians and free the princess? You are that warrior: a mighty barbarian wielding your broadsword with deadly skill.

BBC Micro Cassette ..... £9.95 Acorn Electron Cassette £9.95
BBC Micro 5¼" Disc ..... £11.95 BBC Master Compact 3½" Disc £14.95

(Compatible with the BBC B, B+ and Master Series computers)

Please make cheques payable to "Superior Software Ltd".

PRIZE COMPETITION

£100 is the first prize in our competition, with 20 congratulatory certificates for runners-up.

To enter the competition, you must complete the game and write to us describing the final messages that you receive.

Closing Date: 30th September, 1988.

SUPERIOR SOFTWARE Limited

ACORNSOFT

PALACE SOFTWARE

VISA

24 HOUR TELEPHONE ANSWERING SERVICE FOR ORDERS

Dept. 2, Regent House, Skinner Lane, Leeds LS7 1AX. Telephone: 0532 459453

Personal Ads

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

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

Watford Electronics DUMPOUT 3 ROMs, with manual, £17. View 3.0 manual, £6. Tel. (0924) 826483 eves.

Wordwise Plus. Both manuals and key-strip included, all as new, £25. Tel. High Wycombe (0494) 716037.

InterWord £20, Toolkit plus £15, ROMIT £13, all complete with manuals. APTL board inc RAM chips £20. Modem 2000 and Micronet ROM £25. Tel. (0923) 775098.

Disc based games: Barbarian, Spy vs Spy, Roundheads, White Knight Mk12, all 4 for only £10. J Crabtree, 10 Pathfields, Dartmouth, Devon TQ6 9HL. Tel. Prestel 21997318.

Z88 with 2 extra RAM packs (32k and 128k) and mains adaptor, £300 all in price. Tel. (0734) 784897.

Colour Monitor: Microvitec 1431 in excellent condition £120. Prefer buyer collects. Carriage extra. Tel. (0689) 57245.

Cumana lock disc drive £35 plus ROMs: Toolkit £5, Replay (8271) £10, Comunicator £5, Wordwise £10. Tel. 01-699 5087.

BBC Model B 32k DNFS 1.2, O.S. 1.2, excellent condition, £225. Tel. (0273) 516941.

Archimedes 305 (OS 1.2), Zenith 12" high-resolution green screen. User guide, Programmer's Reference Manual (Vols 1 & 2) Welcome guide & disc, Arcwriter, parallel printer cable, and fifteen 3.5" discs, £730 ono. Matched pair Celestion Ditton 44 hi-fi loudspeakers, £80. Tel. Oxford (0865) 246892 after 8pm.

Z88-BBC Link £10. 32k RAM and EPROM £15 each, Z88 computing by Ian Sinclair £6. Tel. Ratna 455 1069 eves.

Electron plus one and ACP plus 4 DFS both as new, with the manual and in original packaging £60. Tel. St. Albans 72865 eves.

BBC B series 7, Solidisk DDFS, 80T DSF, Microvitec Cub monitor, InterWord, Wordwise +, Paul Beverly's continuous processing ROM, Disc Doctor, Watford 32k RAM card, data recorder together with all manuals etc., offers in the region of £475. Will split. Tel. (0502) 57227.

B plus 64 or 128 owners only, ATPL Side plus ROM board £10. Watford DDFS plus £10, both with manuals. Tel. Farnham Common (028 14) 5332.

Master 128, Cumana 40/80 DS/DD PSU disc drive, Brother M1009 NLQ printer, InterWord, Exmon II, AMX Superart Dumpout 3, extra 32k SWR write protect cartridge, User manuals 1 & 2, other books data recorder, joystick, covers, 30 discs, BEEBUG Vol.2-6, Acorn User magazines, lots of software on disc and tape, mint condition, £650. Tel. (0375) 380369.

BBC B 1.2 O.S. Issue7 + games + 40 BEEBUG mags, £199 only, negotiable. Tel. W.O.T. 242960 (Middx) after 7pm.

Solidisk 2M 128K memory expansion £30. Watford Mk 2 ROM/RAM expansion fitted 16k RAM £15. EDUCAD £20. AMX Super Art + mouse £20. Sensible offers considered. Tel. 01-500 5701.

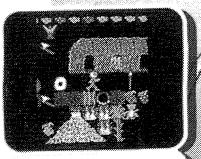
Master Compact computer with second disc drive, printer lead and amber monitor, £375. Computer Concepts Mega-3 ROM £15. BEEBUG Master ROM (InterWord, InterSheet & InterChart) £50. Spellmaster ROM £25. Watford Dumpout 3 ROM £15. BEEBUG Master ROM £15. Masterfile II ADFS £15. Beta-Base with utilities £25. Signwriter plus four extra fonts £20. All with full documentation. 30 3.5" discs in storage boxes £20. Sell separately or whole lot for £450. Tel. Newmarket (0638) 62044.

Microvitec 452 Colour Monitor £120. Viglen console system for BBC B + separate keyboard case £28, ROMit ROM complete £15, Zenith green screen monitor £40, Aries B20/B12 complete £60. Tel. (0403) 814976.

BBC B issue 7 Watford DFS, dual 40/80 switchable double sided drive, 30+ discs, games, ROMs, manuals and books £375 ono. Tel.(0703)553237.

Printer: Shinwa CP-80A in box with lead, manual and ribbons, £99. Also two 100k disc drives with dual lead, £60. Tel. (0257) 278286 Lancs.

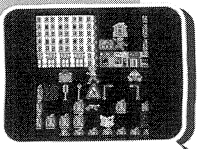
REPTON THRU TIME



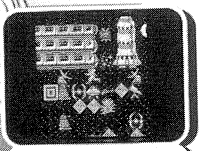
PREHISTORIC REPTON

EGYPTIAN REPTON

VICTORIAN REPTON



PRESENT DAY REPTON

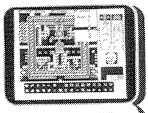


FUTURE REPTON

REPTON — THE TIME TRAVELLER

Where did Repton come from? Does he have an Egyptian mummy? Is he the real Jack the Repper? Where is he now? Where is he going to?! We have been inundated with these and many other questions about our lovable hero. Now all is revealed in 40 new screens that vividly reveal Repton's evolution from prehistory to the future.

The PREHISTORIC screens with their caves, mountains and volcanoes, have Repton battling against pterodactyls and dinosaurs to collect edible berries. In EGYPTIAN times, Repton chases around pyramids and sphinxes collecting scrolls and meeting a mummy or two! Amidst the smog and grime of VICTORIAN times, Repton collects gold coins — but can he avoid the police and the gallows? PRESENT DAY Repton has even greater dangers to face as he rushes around the city's jungle of parking meters and skyscrapers, looking for cans of cola. If the gangster's machine gun doesn't get him, the traffic wardens will. Amazingly Repton may make it to the FUTURE. As he zooms through space collecting crystals, the Martians give chase. Will he succeed or disappear forever into the infinite depths of a black hole? Only you can help Repton Thru Time!



THE SCREEN EDITOR

Each copy of Repton Thru Time includes:
● the Repton 3 main program ● the Repton 3 editor and ● the 40 new game screens.

BBC Micro Cassette.....**£6.95**
BBC Micro 5¼" Disc.....**£7.95**
Acorn Electron Cassette.....**£6.95**
BBC Master Compact 3½" Disc..**£9.95**

(Compatible with the BBC B, B+ and Master Series computers).

The screen pictures show the BBC Micro version of the game.

THE CHARACTER EDITOR

PRIZE COMPETITION

If you complete all 40 screens of Repton Thru Time without using passwords, you can enter our competition. A draw will be made from all the correct entries received to select 25 prize winners. Each will receive an EGYPTIAN REPTON Cuddly Toy and a signed certificate.

Closing date: 30th September, 1988.

REPTON ORIGINAL
Cuddly Toy

SPECIAL OFFER — NEW!

A unique range of Repton Cuddly Toys is now available. Each Cuddly Toy is about 24 inches high and has a realistic face and features corresponding to the particular Repton character. They are only available from Superior Software.

REPTON ORIGINAL **£9.95**
ARCTIC REPTON **£9.95**
TEENAGE REPTON **£9.95**
Postage and packing free.

SUPERIOR SOFTWARE Limited

ACORNSOFT

Please make all cheques payable to "Superior Software Ltd".

ACCESS VISA

24 HOUR TELEPHONE ANSWERING SERVICE FOR ORDERS

OUR GUARANTEE
- All mail orders are despatched within 24 hours by first-class post.
- Postage and packing is free.
- Faulty cassettes and discs will be replaced immediately. (This does not affect your statutory rights).

Dept. T5, Regent House, Skinner Lane, Leeds LS7 1AX. Telephone: 0532 459453.

BBC USER GROUP UPDATE

SWEDEN
Acorn User Society of Sweden
Tord Israelson, Tuppa Klars Vag 12, S-702 29 Orebro, Sweden.

BIRMINGHAM
Birmingham Amateur Computer Club
Kingsbury Road Community Centre, Kingsbury Road, Erdington, Birmingham. Less than 5 minutes from M6 Junction 6. Chairman: Roy Eddington 021-378 3404. Secretary: Ray Moorby 021-742 4649.

DEVON
North Devon Computer User Group
The North Devon Computer User Group holds its meetings on the first Wednesday of every month. New members and visitors are welcome to come along. For more information please contact David Roberts on Ilfracombe (0271) 64882.

New Address:

BUCKINGHAMSHIRE
South Bucks Acorn Computer Club
Contact Alan Lilley, 8 Cross Meadow, Chesham, Bucks HP5 2RU. Tel. (0494) 785549
Meeting 8pm-10pm on 1st Tuesday of the month at St.Leonards Church Hall, **Chesham Bois.**

LIVERPOOL
Mersey BBC User Group (MBUG)
Now meets twice monthly in college terms. First Wednesday of each month: Sandown College (Old Swan Site), formerly Old Swan Tech College, Room C33 from 7.30 pm. Third Wednesday of each month: St Julie's College, Woolton, 6th Form Block, from 7.30 pm. Refreshments available (at St Julie's). For more information call Nik Kelly on 051 525 2934.

HAMPSHIRE
Portsmuth & Fareham
Michael Archer, 113 Beerhirst Crescent, Paulsgrove, Portsmuth, Hampshire PO6 4EJ. Tel. (0705) 325647. Group meets at 7pm on Tuesdays at the Portchester Community Centre.

LONDON
The WC1 User Group (University College London) is not operational any more.

BIRMINGHAM
MARS BBC User Group
Meets on the second Tuesday of each month 7.30pm at the Midland Amateur Radio Society headquarters, Unit 16, 60 Regent Place, off Caroline Street, Birmingham 1. Tel. Michael Nyman 021-382-3606

EVENTS

PCW 88 Show
11th Personal Computer World Show
Earls Court, London
14th-18th September 1988

The Electron & BBC Micro User Show
New Horticultural Hall
Westminster, London
11-13th November 1988

YET MORE PRINTERS

Our printer expert Geoff Bains is back with details of the latest printers to hit the market place.

Dot-matrix printers with 24 pins are definitely here to stay. There are now a good many 24-pin machines on the market with prices within reach of BBC micro owners (and many even more expensive machines).

24-pin machines offer all the advantages of 'normal' 9-pin dot-matrix printers (versatility, speed, and so on) with a fundamental improvement - the NLQ print quality is that much better and at a speed considerably faster than the 9-pin cousins can manage, because on these machines NLQ printing is achieved in a single pass.

However, there still are some 9-pin machines worthy of note and two of them (the LC-10 and the BX-480) get some attention here. With more pins (or their equivalent) than most of the others put together, the Deskjet from Hewlett-Packard gets the BEEBUG look over too.

One major criterion of dot-matrix printers is speed, and the printers here have been tested to find a speed figure which is directly comparable between models and which represents a speed you can expect in actual use - unlike the usually misleading figures claimed by manufacturers.

CITIZEN HQP-40 £574

The HQP-40 is a mid-priced Epson-compatible 24-pin machine which concentrates on speed. It is a large machine with all the mechanics at the back and big areas of mostly empty case at the front. The interface sockets are on the same side as the paper wind knob, so positioning this printer on the side of the Beeb is inconvenient.

However, it is reasonably made if a little on the 'plasticy' side, and the front panel switches are real push button switches - a pleasant change to the usual membrane type.

Normal NLQ text
Emphasised text
super- & sub- script
Italic text
Proportional text
Underlined text
Reverse text
Double height text

The tractor feed unit can be swung into one of two positions to push or pull the paper through. With a suitable printer stand, paper can even be fed into this machine from the base which allows the paper stack to be kept out of the way below the machine.

For cut sheet paper, the HQP-40 has an unusual automatic paper loading mechanism which requires no triggering. The paper is simply placed in the fold-up guide and, without further prompting, the printer feeds it through to the right position for printing. A good idea, but the mechanism is prone to feeding the sheets crookedly.

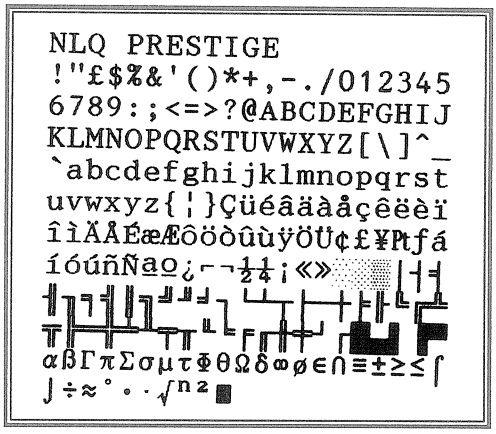
```
NLQ MODE
!"#$%&'()*+,-./012345
6789:;<=>?@ABCDEFGHIJ
KLMNOPQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrst
uvwxyz{|}
```

As well as a reasonably fast, decent quality NLQ print at 46cps and a good speed draft print at 114cps, the HQP-40 can also produce a 'correspondence' mode print at a speed of 75cps, but this is barely distinguishable from the draft quality, so most people are unlikely to use it much.

HQP-40 offers the usual effects of underlining, double width, proportionally spaced, bold, italic, super and subscript characters, as well as novel ones such as double height text and reversed (white on black) text. The HQP-40 can also take credit card sized 'smart cards' containing other NLQ fonts. These cost about £35 each.

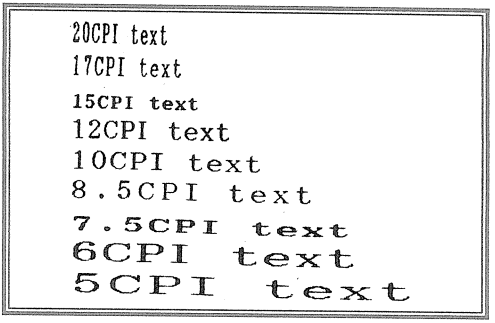
EPSON LQ-500 £443

The LQ-500 is one of the cheapest 24-pin printers around, beaten only by the £349 Amstrad LQ 3500. However, it is better thought out and made. A push-feed tractor unit is provided, but this must be removed if the printer is to be used for cut sheet paper. The automatic paper loading requires triggering from a front panel button, but it is very smooth and efficient. It copes with all reasonable thicknesses of paper without ever catching or feeding crookedly.



The LQ-500 is (of course) Epson compatible but it also provides all the IBM graphics characters and accented letters. The LQ-500 can produce a good quality NLQ print at 40cps (about a minute per page of text - much faster than the Amstrad LQ3500) and draft at a respectable 97cps - under 30 seconds for a typical page.

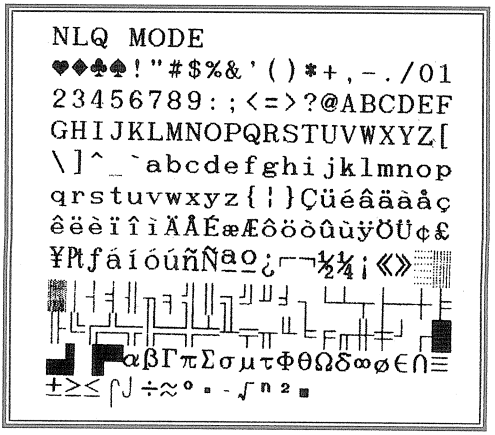
However, this printer is extremely noisy. Without the plastic cover in place (and it often gets in the way) the noise makes it totally impossible to hold a telephone conversation nearby.



The LQ-500 has three NLQ fonts to choose from - the normal typewriter look-alike style (which Epson calls 'Roman'), Sans Serif (a plain modern-looking style), and Prestige (similar to Roman). Further print styles can be added on plug-in cartridges for £44 a time.

SEIKOSHA SL-80AI £401

Even though the SL-80AI costs more than the Epson LQ-500 it has fewer features and is of inferior construction. This printer has a distinctly cheap feel about it, and it doesn't look as if it will have a long life. A clip-on tractor feed unit is provided for continuous stationery but this uses the wasteful pull-feed method. Annoyingly, the tractor unit has to be removed for friction feeding cut sheet paper. However, the SL-80AI does have efficient automatic paper

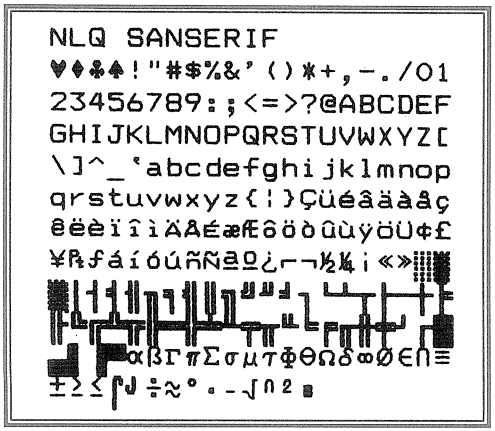


loading which rarely requires any intervention by the user.

The SL-80AI produces reasonable quality NLQ print, but it is slow and ponderous. The printer seems to stop for thought between each line. Even feeding the paper at the end of a page is visibly slower than on other machines. The draft speed is only 53cps and the NLQ speed 26cps - over a minute and a half for a typical page of text. For a considerably lower price a 9-pin printer can produce NLQ print of similar quality at the same or a faster speed. Strangely, when draft mode is selected with the front panel switch, control codes from the computer cannot override it to put the printer into NLQ mode - a silly arrangement which can prove frustrating in use.

The SL-80AI is both Epson and IBM compatible and can manage most of the basic printing effects. There is no 15 and 7.5 characters per inch (cpi) text size, which most printers provide, and the SL-80AI cannot produce condensed text (17 and 20cpi) in NLQ mode. Some other printers suffer this problem too but they print condensed draft text when condensed print is selected in NLQ mode. The SL-80AI just ignores the command altogether.

STAR LC-10 £263



The LC-10 proves all is not lost for 9-pin machines. For a remarkably low price this printer offers almost all the features of machines several times its price. It is fast, achieving 79cps in draft mode and 18cps NLQ mode - about 2.5 minutes for a page of text.

The NLQ quality is excellent - far better than other machines of this price. The letters are well styled and crisp and dark. Unusually for this price of printer, the LC-10 offers three different styles of NLQ character. As well as the usual 'Courier' typewriter look-alike font, there is a choice of the plain 'San Serif' style or the modern 'Orator', which can print small letters either in lower case, or as smaller sized capitals. The LC-10 can perform all the usual dot-matrix printing effects and it can produce enlarged letters nearly half an inch high.

It is equipped with both friction and tractor feeds. The tractor feed is hidden away under a back panel when not in use. Unusually for a budget printer, the LC-10 has automatic paper loading too. The printer is solidly made and well designed. Its only silly feature is the placing of the interface socket on the same side as the paper wind knob.

BX-480 £424
Sold as: Walters WM480
Precision Software 4010
Micro Peripherals MP-480

For pure speed of printing there is nothing to beat the BX-480. This is a peculiar printer in a couple of ways. It is sold by three different companies in the UK, under three different

names - as the Micro Peripherals MP-480, the Precision Software 4010 and the Walters WM480. BX-480 is its Japanese manufacturer's name.

It is so fast because it uses four separate print heads spaced across the printer width, each of which prints only a quarter of each line of text. In draft mode this printer can print at a staggering 237cps - about six pages a minute. Only a laser printer (costing upward of 1200) can match that. In NLQ mode the speed is reduced but still a healthy 53cps - 1.5 pages a minute which is at least 1.5 times the speed of any other printer under £1000.

However, in NLQ mode the print quality is appalling. It is little more than a bold version of draft print with no attempt to style the characters. In addition, the 'descenders' of letters such as g and j are not printed below the level of letters without tails, and letters such as b's and d's similarly have poor 'ascenders'.

However, for fast rough copies, especially when used alongside a daisywheel printer for the final neat printout, the BX-480 is unbeatable because of its speed. Both Centronics and RS232 versions are available at the same price, allowing the BX-480 to be connected to the BEEB together with an existing daisywheel machine.

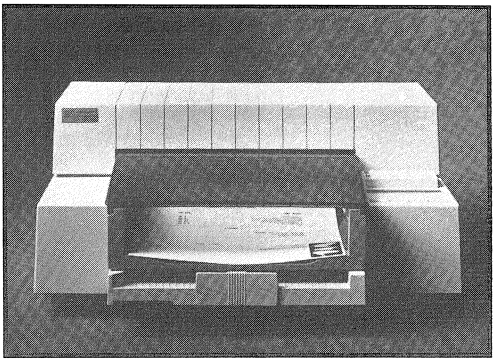
The BX-480 is designed very much for fanfold paper. No automatic paper loading is provided and the top cover has to be removed to load each sheet. However, to use this machine for cut sheet paper is to largely waste its talents.

If you want a fast printer, the BX-480 is the fastest and good value with it.

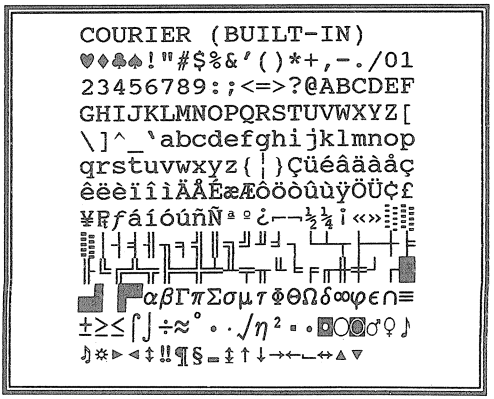
HEWLETT PACKARD DESKJET £978

The last printer looked at here is the most expensive ever considered in BEEBUG. It may seem over the top to spend more on a printer than the rest of your micro system put together, but in the case of the Deskjet this is an economy. You would be saving around £800.

This is because the Deskjet is all but a laser printer. It has the laser quality print, laser quality graphics and laser flexibility. All it is missing is the speed.and of course the laser! This is an ink jet printer. It is a dot-matrix machine but it prints about 300 dots per inch. The result is truly difficult to tell from typed or typeset copy.

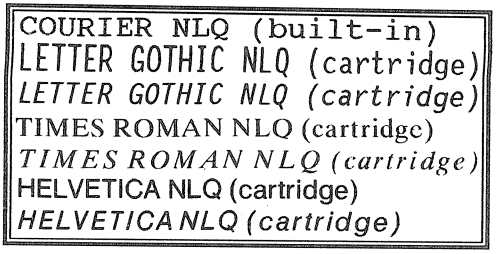


The Deskjet even looks totally different from other printers. It has more in common with a photocopier than a trusty Epson FX-80. It has no tractor feed and it cannot use fanfold paper. Instead it has a bulk sheet feeder built-in as standard. This takes a stack of cut paper, automatically loads it as required and deposits the printed sheets in a neat pile in the out tray.



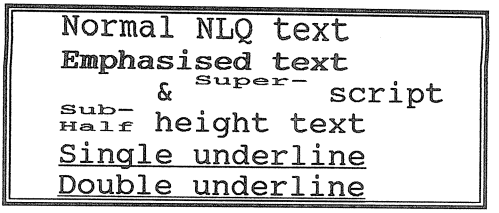
There is no need for fanfold paper with this kind of treatment. The feeder is happy with all

sizes from the width of A4 to the length of foolscap. Other sizes and envelopes can be fed in by hand.



The Deskjet has one font built-in and others are available on cartridges (for £66 each). The range of print styles and sizes is somewhat limited with any one cartridge, but a RAM cartridge for downloadable fonts is also available for changing fonts 'on the fly', although software for this is confined to IBM PC programs.

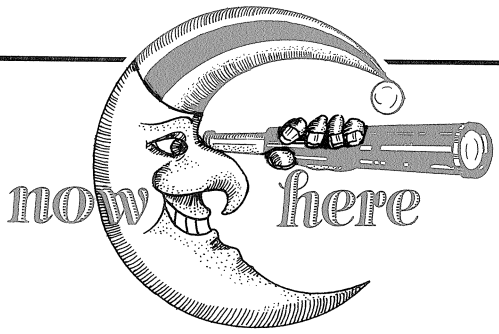
The Deskjet can print at a reasonable speed - 93cps in draft mode and 67cps in NLQ. This is not in the same league as a laser printer but few dot-matrix machines under £1000 can manage this NLQ speed and certainly not to this quality.



The big problem with the Deskjet is in the software codes used to control it all. These are to Hewlett-Packard's own standard - a particularly nasty bunch of long and unmemorable numbers. The codes used are actually a subset of those in the Hewlett-Packard Laserjet, but as no Beeb software is designed for that either, this doesn't help much. An Epson emulation cartridge is available (also for £66) and that makes life much easier for the BBC micro owner, but adds to the cost. B

Make & model	Price (inc.VAT)	Draft speed claimed	Draft speed actual	NLQ speed claimed	NLQ speed actual	NLQ quality	Supplier
Citizen HQP-40	£574	200	114	66	46	8	Citizen (0895) 72621
Epson LQ-500	£443	150	97	50	40	9	Epson 01-902 8892
Seikosha SL-80AI	£401	120	53	47	34	7	Seikosha (0753) 685873
Star LC-10	£263	120	79	30	18	8	Star Micronics 01-840 1800
Walters WM480	£424	480	237	80	53	1	Walters (0494) 32751
Precision Software 4010	"	"	"	"	"	"	Precision Software 01-330 7166
Micro Peripherals MP-480	"	"	"	"	"	"	Micro Peripherals (0265) 473232
Hewlett-Packard Deskjet	£978	240	93	120	67	10	Hewlett-Packard (0734) 696622

* *Speed is mentioned in characters per second.*
* *NLQ quality is on a scale of 1 to 10, with 10 the best.*



Part 5

ANY OTHER BUSINESS

Dec McSweeney concludes his introduction to C with some apposite words about the use of the BEEBUG C compiler, and a quick overview of C's many other facilities.

This is the final part of our introduction to the C programming language. Up to now, the series has been implementation-independent; that is, all the information applies to any version of C - on any machine. This month we will be concentrating on the BEEBUG implementation, and focusing on the compilation process and library facility. Owners of other versions of C should still find plenty of interest so stay tuned!

A quick recap first. C is a compiled language, which is to say that object code is generated from the source program by a process known as compilation. When the program is run, it is the object code rather than the source which is used. BBC Basic programmers will be familiar with the concept of generating machine code by assembly - a similar process. Compilation of C on the BBC micro and other computers is essentially a two-part operation - the source is first compiled, then external function calls are included by the linker.

Almost all the functions used in C are not part of the language itself. For example, **printf()**, the equivalent of the Basic PRINT instruction, for example, is a separate routine which is supplied as part of the implementation. The object code for **printf()** exists in a library of routines called **rtlib** (Run-Time LIBrary). When a source program is compiled, an intermediate ("semi-compiled") file is created which contains references to functions outside the program itself - a bit like calling non-existent procedures in Basic. The program cannot be run in this form, even if there are no unresolved references.

These references are resolved by LINKing the semi-compiled file - a process where the external functions are fetched from the library and added to the program, creating an executable file. The BEEBUG C user guide, supplied with the software, contains descriptions of all available functions in **rtlib**. You may find that you require other routines regularly - for example to calculate VAT. Rather than include the same code in every program you write, it is possible to add such a routine to the run-time library, or create your own. The BEEBUG C package includes a utility to maintain libraries, and its use is described here.

First let's write a short routine which calculates the VAT on a given quantity, returning the total. Type in the following and save it in a file called **c.addvat**:

```
static double addvat(double amount)
{
return(amount * 1.15);
}
```

The first line may not look familiar. It defines the function to be of type **double**, which means it returns a value of type **double**. It is declared as **static** to avoid confusing the linker. In the brackets we have declared the type of parameter the function expects as well as its local name. This is a useful extension to "standard" C, part of the new ANSI (American National Standards Institute) standard which is destined to become the "definitive" version of the language.

You can compile this routine, even though it has no **main()** function. Since it will be a library routine, we will use the **NODEBUG** option when compiling:

```
COMPILE/NODEBUG ADDVAT
```

This will create a 'semicompiled' file called **o.addvat**. To install this in a library, we must run LIBRARY - an object program supplied with BEEBUG C which is very easy to use. Your conversation with the computer proceeds as shown.

Computer	Programmer
C prompt ($)	RUN LIBRARY
(Displays version no etc.)	
LIBRARY>	CREATE MYLIB
Identification text:	Anything you like
Copyright text:	Whatever
LIBRARY>	INSERT addvat
Name of object file:	O.ADDVAT
LIBRARY>	QUIT
Commit changes?	Y
Change identification text?	N

Now you've created a library, here is our main program, to be saved in **c.libtest**:

```
/* use your own libraries */

#include <h.stdio>
extern double addvat(double amount);
main()
{
double money=0;
while(money < 100){
 printf("\nType in the amount : ");
 scanf("%f",&money);
  printf("Adding VAT we get %4.2f",
addvat(money));
 }
}
```

Here, after the **#include** line, we have declared that a function called **addvat** exists, external to this source file. This is necessary to avoid upsetting the compiler. If you look through the header file **h.stdio** you will find similar declarations for all standard routines. If you have many new routines, perhaps you should create your own header file to be #included.

Compile this and try to link it:

```
LINK libtest
```

This should fail with the message "Undefined symbol in O.LIBTEST - addvat". This is because the linker uses **rtlib** as the default library and **addvat** is in MYLIB. The correct command is:

```
LINK/LIBRARY=rtlib,MYLIB libtest
```

This causes the linker to look in both libraries for the external functions. You should get an error-free linkage, and an executable file which you can run in the normal way to give hours of fun(!).

The main advantage of using libraries is that the code for commonly-used routines can reside in just one place. Should a library routine require amendment, those programs which use it can simply be re-linked, instead of having to edit, recompile and link each one afresh. This has advantages in commercial environments where any revised routine can be easily installed.

ANY OTHER BUSINESS

In this short series we have only covered some of the features of C; it is a complex language to master fully, though by now you should be able to write and understand simple programs. Several areas merit a brief mention before we close this concluding article. For more complete explanations I would refer you to *The C Programming Language* by Kernighan and Ritchie (known as K & R), or one of the many C tutors which are available (for example, the excellent *C: A Dabhand Guide*, reviewed in BEEBUG Vol.7 No.2).

SWITCHES

A **switch** is is a multiple-choice conditional statement (called a CASE statement in some other languages) which could be used in many situations instead of a sequence of if statements, as in **main()** in the line editor (see last month). The switch takes the form:

```
switch(exp) {
      case const1 :
```

```
            statement1;
            break;
        case const2 :
        case const3 :
            statement2;
            break;
    /*    **** etc. ****    */
        default:
            statementx;
}
```

The expression exp is evaluated and then tested against each constant (const1, const2 ...) in turn. If the expression matches the constant, the corresponding statement (statement1, statement2 ...) is executed.

The optional break command forces an exit to the statement following the closing brace. Several cases may precede a statement. Statements may be compound (several statements within braces). The expression and constants must be of type int or char.

This is a useful command but not easy to get to grips with if your first language is Basic.

POINTERS TO FUNCTIONS

As well as allowing you to access and manipulate pointers to data, C permits you to do the same with functions. To declare a pointer to a function which returns an integer use:

```
int (*ifunc)();
```

To call the function pointed to by this pointer write:

```
(*ifunc)(arg1, arg2, ....);
```

The commonest way of assigning values to *function pointers is in a function call:*

```
do_something(7, some_function);
...
do_something(how_often, ifunc)
int how_often;   /* number of calls */
int (*ifunc)();  /* argument is a pointer */
{
int x;
for(x=0; x<how_often; x++)
  (*ifunc)(arg1 etc.); /* calls some_function
*/
}
```

The address of the function some_function is passed to do_something() because of the context (no brackets following the function name). Most reference works (even K & R) are curiously short on details about this facility; it strikes me as a solution looking for a problem!

CASTING

This feature allows the type of a variable to be explicitly changed. As an example, consider a call to addvat() (which expects an argument of type double) where the argument starts life in a variable of type int:

```
int n=3;
double x;
...
addvat(n);    /* This might generate nonsense
or worse */
x = n;        /* implicit conversion to double
*/
addvat(x);    /* This would work */
addvat((double) n); /* So would this */
```

The last line here shows casting at work. A copy of the value held in n is passed to addvat, but as if it had come from a variable of type double. Nifty, what?

UNIONS

This is a feature whereby the same area of memory can hold data of several different types and sizes (one type at a time). Unions are defined like structures:

```
union this_union{
     int memnum;
     double salary;
     char age_group;
     } u_details;
```

This defines an area of memory big enough for the largest variable type declared within the braces. To access the variables, you could use:

```
u_details.memnum = new_number++;
```

or, using a pointer:

```
this_union *u_pointer;
        /* pointer of type this_union */
u_pointer = &u_details;
total_salary += u_pointer->salary;
```

It is up to the programmer to keep track of the data type currently in the union. This can be a cause of much grief, so take care!

INPUT AND OUTPUT

We have only scratched the surface of C's input and output facilities. Many standard functions exist to provide formatted input and output, as well as in-memory formatting. Full definitions of **printf()**, **scanf()** and their siblings **fprintf()**, **fscanf()** (file input-output) and **sprintf()**, **sscanf()** (in-memory formatting) will be found in your user guide. Nor should we forget the file-handling functions **fread()**, **fseek()**, **fwrite()**...

BBC FACILITIES

The BEEBUG implementation of C allows access to all the operating system facilities (OSWORD, OSBYTE etc.), as well as to the Basic extensions in the form of **draw()**, **point()**, **gcol()**, **vdu()** and so on. You can even change modes within a called function.

BIT MANIPULATION

Several bitwise operators exist and may be used on all but types float and double:

&	bitwise AND
|	bitwise OR
^	bitwise EOR
<<	shift left (zero fill)
>>	shift right (zero fill (unsigned), sign fill (signed))
~	one's complement

THE PRE-PROCESSOR

We have seen the use of **#include** as an instruction to the compiler. In fact there are a number of compiler commands which can make the C programmer's life easier. In the line editor, we used **#define** to set the values of TEXTMAX and LINEMAX. To change these values, we need only alter the **#define**, instead of changing many lines throughout the program - and perhaps missing one! **#define** can also be used to set up macros - expanding these at compile time into executable statements. The draw command, for example, exists as a macro in **h.stdlib**:

```
#define draw(x,y) plot(5,x,y)
```

Other pre-processor commands include **#undef** (forget a **#define**), **#ifdef** and **#ifndef** (check if a #definition has been made), and **#pragma** (controls compiler listing options among other things).

Well that about wraps up this introduction to C, said by many to be the language of the future. Those of you who work with computers will have seen the increase in interest within the industry as the operating system UNIX, which is written in C and for which C is the preferred language, becomes available on more and more machines. C has advantages over almost all other commercially-used languages and is still being improved - watch out for C++!

If you have found the series enjoyable and would like to see more articles and programs in C please write to the editor.

B C-ing you! B

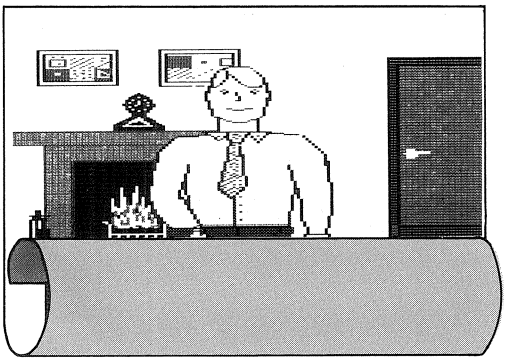
This article concludes our introductory series on the C language. If you have any requests for further C programming topics, or C programs, then let us know. If you have a C program which you think would be suitable for publication we would be pleased to consider this. If you missed any of the earlier parts in this series (five in all), then all the relevant back issues are available (see inside back cover for prices).

MASTER SERIES

Roll-Down Screen Displays

This novel utility by Brian Knott will allow your Master to roll pictures down the screen as seen in so many Archimedes demonstrations.

You will no doubt have seen on television impressive graphics effects where the picture curls up and shoots off the screen, or is sucked away down a 'plughole' in the centre of the screen - these are produced by dedicated graphics systems like Quantel's Paintbox. However, the Master is capable of some of the simpler graphics tricks, although at a somewhat slower speed. This article explains how two of these effects can be achieved. The first is as if the display has been rolled into a tube and this is slowly unrolled down the screen. The second is a Venetian blind effect where the display appears in gradually widening strips across the screen.



Applications for these effects are numerous. They could be used to display title screens, graphic pictures, rolling demonstrations and so on. The effects are achieved by first loading the picture into memory above the Basic program and workspace with the Master in a shadow screen mode so that this will not affect the display. Next, the machine code utility will transfer this image, piece by piece, into screen memory creating the desired effect. The screen image must be in mode 0, 1 or 2 and must have been saved in the normal manner (i.e. using *SAVE <filename> 3000 8000).

ENTERING THE PROGRAM

The program should be typed in and saved. When run, a machine code file will be assembled. A checksum is included to make sure that the assembly language has been entered correctly. If this is not so, the file will not be saved.

USING THE PROGRAM

Once the machine code utility has been successfully assembled it is ready to be used by your own program to create the desired effects. This small program demonstrates how the utility may be used:

```
10 *SHADOW
20 MODE 0
30 HIMEM=&2E00
40 VDU23,1,0|
50 *RUN ROLLDN
60 *LOAD IMAGE1
70 CALL &DD6B
80 VDU 23,1,1|
90 END
```

Line 10 ensures that the machine will remain in a shadow screen mode when you select the appropriate mode for the screen image in Line 20. Line 30 will ensure that the Basic stack is put below the area in which we will load the screen image initially. Lines 40 and 80 simply turn the cursor on and off. These are not essential but improve the display. Line 50 loads and initialises the machine code utility. Line 60 loads the appropriate screen image into memory. In this case it is assumed that there is screen on disc called IMAGE1. Line 70 calls the machine code routine to create the desired effect. Calling &DD6B will perform the roll-down effect whilst &DDB5 will perform the blinds effect. Following the same procedure any Basic program can use the utility just as easily.

It is worth noting that some screens may need the logical colours to be changed with a VDU 19 command. If this is done before the machine code is called, it may result in a colour change in the existing picture on the screen. Executing the VDU 19 after the machine code has been called will result in the effect being in the wrong colours. You will have to decide what best suits your application.

MEMORY USAGE

The machine code is assembled into the 'transient program area' at &DD00 recommended by Acorn for short utility routines. A number of zero page locations between &70 and &83 are used as counters and markers, while locations &2E00 to &2FFF are used to hold a table of values which the machine code accesses. Locations &3000 to &7FFF are used to hold the screen image which will be displayed when the effects are called.

Call Addresses	
Roll Down	&DD6B
Blinds	&DDB5

PROGRAM NOTES

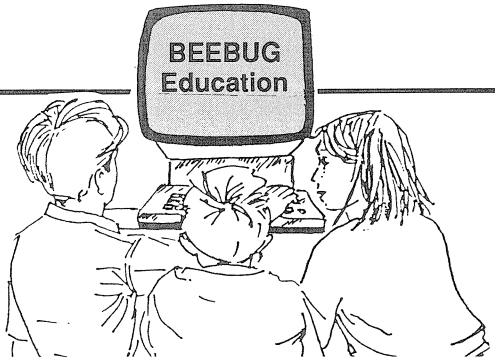
There are actually three different machine code routines. The first routine is called only once, when you *RUN ROLLDN, and creates a table of start addresses for each row on the screen. The other two routines performthe roll-down effect and the blinds effect.

You should now be able to implement this handy routine in your own programs to make your screen displays even more interesting.

```
   10 REM Program  ROLLDOWN
   20 REM Version  B0.25
   30 REM Author   Brian Knott
   40 REM BEEBUG   July 1988
   50 REM Program  subject to copyright
   60 :
  100 ON ERROR MODE 7:PROCerror:END
  110 MODE7
  120 OSWRSC=&FFB3
  130 PROCass
  140 Y%=0:FOR X%=&DD00 TO &DE17:Y%=Y%+?
X%:NEXT
  150 IF Y% <> &95C0 PRINT"Error in mach
ine code.":END
  160 *SAVE ROLLDN DD00 DE18
  170 END
  180 :
 1000 DEF PROCerror
 1010 REPORT
 1020 PRINT " at line ";ERL
 1030 ENDPROC
 1040 :
 1050 DEF PROCass
 1060 FOR pass%=0 TO 3 STEP 3
 1070 P%=&DD00
 1080 [:OPT pass%
 1090 .store
```

```
 1100 LDA #&F8:STA &80
 1110 LDA #&2E:STA &81
 1120 LDX #32
 1130 .st1 LDY #7
 1140 .st2 TYA:CLC
 1150 ADC #&80:STA (&80),Y
 1160 DEY:BPL st2
 1170 LDA &80
 1180 SEC:SBC #8:STA &80
 1190 LDY #7
 1200 .st3 TYA:STA (&80),Y
 1210 DEY:BPL st3
 1220 LDA &80:SEC
 1230 SBC #&8:STA &80
 1240 DEX:BNE st1
 1250 LDA #&F8:STA &80
 1260 LDA #&7D:STA &82
 1270 LDA #&2F:STA &81
 1280 LDX #32
 1290 .st4 LDY #7:LDA &82
 1300 .st5 STA (&80),Y
 1310 DEY:BPL st5:DEX
 1320 DEC &82:DEC &82
 1330 LDA &80:SEC
 1340 SBC #8:STA &80
 1350 LDY #7:LDA &82
 1360 .st6 STA (&80),Y
 1370 DEY:BPL st6
 1380 DEC &82:DEC &82
 1390 DEC &82:LDA &80
 1400 SEC:SBC #8:STA &80
 1410 DEX:BNE st4
 1420 RTS
 1430 rolldown
 1440 LDA #8:STA &70
 1450 STA &82:TAX
 1460 TAY:INY
 1470 .rd1
 1480 LDA #&FF:SEC
 1490 SBC &70:SBC &70
 1500 INA:STA &83
```

Continued on page 62



This month Mark Sealey reviews some educational books and software.

INSIDE INFORMATION. Open Learning Pack.
Supplied by: BBC Software, 80 Wood Lane, London W12 0TT. Tel. 01-576 0548. Price £41.35 inc VAT

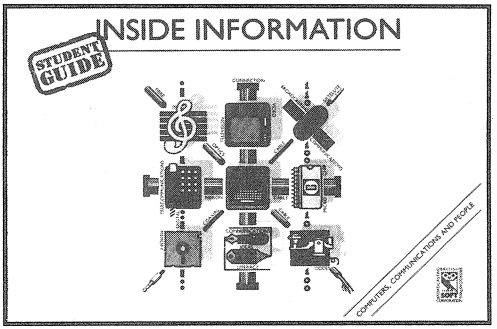
It is common on Computer Literacy courses at Further Education (FE) level for students to follow the subject matter presented to them with interest, but to be left looking for more than either the software, literature or even handouts (and certainly the full manuals) can offer.

What is needed is a compilation of sources that takes the student or any comparative newcomer to information technology (IT) further. It must satisfy their general interest, and answer particular questions about the sorts of software which they have been using. Purchasers of *Inside Information* from BBC Soft can also follow both the Radio 4 series of the same name and the City and Guilds assessment 444. Restructured and re-presented versions of the former come on cassette with this package.

The review that follows and forms the bulk of this month's BEEBUG Education is for readers new (and not so new) to IT, as well as teachers and lecturers who want to provide students with extra material. Here, Inside Information is specifically tied to software which is written to expose the workings of computers and their programs. This, together with the fact that you have all the resources in one product, makes it very attractive.

So why *Inside Information*? Herein lies the strength of the whole pack, which comprises three discs (in BBC DFS 40 track and MS-DOS formats), two audio cassettes with edited versions of the radio series, manuals and a very well written and appealingly illustrated book by Jacquetta Megarry of 200 pages.

There are five programs in the software suite. LID-OFF simulates the four stages of a von Neumann computer: Input, Processing, Memory and Output. LIDTEXT - perhaps the weakest - is a simple text editor. If you wanted to use this for writing your first novel, you would be disappointed. It is clear from the style of the whole package, though, that its authors know just how patchy and incomplete the understanding of students can be even after learning the mechanics of View, Wordwise or whatever. LIDTEXT, in common with the simulated spreadsheet, LIDCALC, and the two data handling programs, LIDDATA and LIDBASE, really does take off the lid and shows how a computer (rather than a typewriter or a shoebox full of cards) handles information. This is a task that few other pieces of software for the BBC Computer have ever done (remember PEEKO Computer?) and these do it very well.



LID-OFF, for instance, shows conceptually what happens inside the computer while running a simple program in Basic to calculate VAT and add it to a price. Programming is becoming less common both in GCSE and in the increasing number of non-specialist FE courses. Yet students often ask about it, so it is good to see it demonstrated so well here. Essential notions like exit conditions and error trapping are thankfully prominent.

Yet that is not all: the screen is divided into several areas, and you have infinite control over the speed at which various messages appear informing you of exactly how the Operating System (modelled on that of the BBC computer) controls input and output, and how algorithms are translated. This is a compelling piece of software that does its job well.

LIDTEXT takes as its model the way that a trained typist or word processor operator would set about correcting text which has been entered at speed and contains errors. It all takes place slowly so that you can see what is happening. This blend of open-ended and pre-determined use of the software, with plenty of opportunity for self-assessment (and with learning objectives spelt out at all stages) is in the best educational traditions.

The database is the most complex program and contains such delights as a slow-motion sorting routine and the ability to achieve a detailed record format. The latter also crops up frequently on computer literacy courses.

By now you will have got the idea. There are many video films aiming to explain how *computers in control* work. The BBC itself did a series on the subject some years back. LIDTURN has adequate graphics to hammer home the essential message: computers can only execute commands one at a time. It is a pity that pseudo-code alone was used and a pseudo-code fairly close to English.

The book, subtitled *Computers, Communications and People*, fills in many of these gaps: the chapter on telephones was particularly enjoyable to read. If you are looking for a well-written comprehensive package for pupils from 15 upwards, which starts with the very basics (recognising a computer), is modelled on the BBC system and covers its ground remarkably well, Inside Information should be seriously considered.

3D LOGO EXTENSION

Supplied by: Logotron Ltd, Dales Brewery, Gwydir Street, Cambridge CB1 2LJ. Tel. (0223) 323656. Price £18.40 inc VAT

In BEEBUG Education Vol.6 No.1 we looked at a range of utilities for the LOGO programming language. LOGOTRON has recently released a 3D extension on disc to run with its original ROM. It is an excellent product and much overall thought has gone into it.

At heart, it provides some twenty-two primitives which have been added to those available on the by now standard BBC implementation. Chief among them are ROLL, PITCH and YAW, which - with SETZ and ZCOR - allow the turtle to operate in a third dimension: that is 'backwards' INTO the monitor in addition to horizontal and vertical directions on the screen.

After playing with the new facility (a cut-out card is provided to help in visualising the process) and trying some of the usual spectacular graphics, you can get down to exploring such structures as polyhedra, perspective and spiral staircases. There are literally hundreds of suggestions for using this extension in other project work, and these are well presented to say the least. A very significant piece of educational software, which, if taken with the books also under review, will set you up nicely for the summer holidays.

MAKING LOGO WORK

by Janet Ainley and Ronnie Goldstein

Supplied by: Basil Blackwell, 108 Cowley Road, Oxford OX4 1JF. Tel. (0865) 791100. Price £6.95

LET'S TALK BBC TURTLE

by Liddy Neville and Carolyn Dowling

Supplied by Cambridge University Press. The Edinburgh Building, Shaftesbury Road, Cambridge CB2 2RU. Tel. (0223) 312393. Price £3.50 Pupil's book and £6.95 Teacher's book

You have to be very careful with books about Logo for it is not only the purists who insist that written guidance can stifle children's activities with the language. In the case of *Let's Talk BBC Turtle*, the strengths of the books are threefold: they are well laid out - especially for children, they are written around the Logotron ROM and, although overpriced, contain things that children of 8 to 14 will actually want to do. If you believe in the assignment approach, and are maybe not too sure about Logo yourself, they are as good as any of their type.

Of greater significance, and with a surer theoretical grasp, is the more comprehensive book by Ainley and Goldstein. It makes a readable and well-argued case for doing worthwhile things with Logotron Logo, and is packed with resources and ideas. The Chapter on Microworlds, a term often used but not so often actually understood, is a good one. But if you are tired by now of the anecdotal, narrative approach - describing what other classes have done with Logo - there will be less in this to appeal to you. B

This month Mike Williams and David Spencer discuss a more general and professional approach to the requirements of file handling.

In the first two articles in this series we discussed how to set up a simple database, and wrote a set of short programs to provide us with the basic requirements of adding, deleting, amending and displaying the records in our file. The system we set up may well suffice for many simple applications, particularly where the number of records, and size of record is reasonably small.

However, the approach we used, while itself simple, suffers from a number of drawbacks. First of all the routines used were specific to the particular application we used as our example. Of course we could write similar programs to handle any other application, but if we want to set up several databases, this does seem to be a clumsy approach. What is required is a more general approach which can be applied to any data file we may wish to use.

In this and succeeding articles in this series we shall be looking at ways of overcoming these limitations, and discussing the many implications that arise as a result. The scale of programming required will also be much greater than before. Because we are not intending to create a comprehensive database program ourselves, but to cover the many ideas and techniques which you may wish to know about, we shall henceforth concentrate on writing procedures which you can use (and modify) in your own programs.

FIXED LENGTH RECORDS

In our previous database, each piece of information used just the amount of space in the file which it needed. The consequence of this is that every record is (nominally) of a different length to every other, and consequently there is no way of determining where in a file a particular record starts.

The solution to this, and one used by many commercially available databases, is to insist that every field and record in each file is of a fixed length. Their sizes may differ from one database to another, but within any one file their lengths will be the same throughout. In these circumstances, if we know which record in a file we require we can calculate exactly where in the file it is (record number multiplied by record length), and use PTR# (described last month) to move the file pointer directly to that position.

Now this does not mean that the data itself is fixed in length, just the size of the space allocated for the storage of that data. Thus each field in a record will be set to a certain

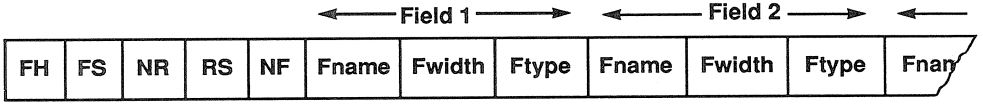


Fig 1. Format of File Description Record

Secondly, the only way in which we could access the records in our file was by starting with the first record and then reading each successive record in turn. We had no way of putting the records into any kind of order (unless they were entered in order at the outset), nor of locating any individual record directly.

maximum size. In the majority of cases of course, the data will use less than the space allocated.

We are already encountering an important characteristic of file handling: in order to facilitate the processing of data the amount of storage space allocated on disc (and in

memory) will be more than that strictly required by the data used. Some forms of data storage can easily double (or more) the space required. All our discussion from now on will assume that we are using fixed length fields, and hence fixed length records.

FILE HEADERS

If we are going to write a program (or set of programs) to deal with all our data files, then with each file we shall have to find some way of storing all the information (like the names of fields, number of fields, number of records etc.) which describes the format of each file. We will refer to this as a *File Description Record (FDR)*. It is like adding an extra record at the start of our data file, but one that is different from those that actually contain data.

Now this may seem a backwards step. We have only just said that the key to file handling is the use of one standard size and format of record throughout a data file, and here we are thinking of introducing another record in each file which is quite different. What we can do though, is to make the File Description Record the *same* format in all files. In that way we can write one piece of code to read the File Description Record of any file, and that in turn will provide the detailed description of the structure of the data file itself.

In fact, you can store as much descriptive information about the contents of a file as you like in the File Description, but this is all an overhead on disc space, so it is best to keep it as short as is reasonably possible. The File Description will also contain things like the names and lengths of the fields which make up a record. Now some data files may contain just a few fields, while others may contain very many (even hundreds). The simplest approach would be to allocate a size for the File Description large enough to store the maximum amount of information which is ever likely to occur. But of course, the majority of files will need much less space so this arrangement is also wasteful.

The solution is to make the File Description modular. We will allocate a basic space of 256 bytes, but add further blocks of 256 bytes for those files which need it. And how will the program know just how much space has been allocated? By specifying as the first byte in the File Description the number of blocks comprising the File Description record for that file. It's easy really.

DEFINING A FILE DESCRIPTION RECORD (FDR)

Let's now consider the information required in a File Description. The minimum we would suggest is as in Table 1.

Identifier	Meaning
FH	Number of 256 byte blocks in FDR
FS	Physical size of file in records
NR	Number of records currently in the file
RS	Size of record in bytes
NF	Number of fields

Table 1

The information in the table applies to the file as a whole, and could be extended further (date created, date last accessed, password etc.). This would then be followed by additional information for every field in the file. This is more open to variation in response to individual requirements, but we suggest you consider the following:

Function	Data Type	Size
Field Name	String	10+2
Field Width	Integer	4+1
Field Type	Integer	4+1
Field Format	String	
Field Prompt	String	
Field Default	String	
Field Checks	String	

The information on fields above the line we consider to be essential for all database systems, though it is possible to omit the file type. The field descriptors below the line are examples of optional extras depending on what is important to you. For example, the field format could specify different layouts for numbers and dates. A field prompt would appear on the screen when prompting for data entry (rather than just the field name). Field checks could be used to validate data on entry against pre-determined limits, ranges etc. The basic arrangement for the File Description Record is shown in the accompanying diagram.

We'll now examine the details of the FDR further. The first five entries relate to the file as

a whole, and all will be stored as integers giving ample scope when it comes to numbers of fields, records etc. The first item, the number of File Description blocks, has already been referred to.

We are assuming that when a data file is first created, the user specifies the maximum number of records the file is to hold, and that the file will be created at that size. In the context it seems better to specify the size in terms of records rather than bytes, and an integer will suffice for this. Likewise, we will also need to store the number of records currently in use, again as an integer. We are assuming that new records will be added following the last existing record in the file. Setting NR equal to zero will indicate that there are no records in the file. An alternative is to have this value point to the next empty position in the file (i.e. one more than the current number of records). Both approaches have their merits when it comes to coding.

The size of a record will be measured in bytes, and will be the sum of the lengths of the individual fields within a record. This could always be calculated when required but it is more convenient to store it once and for all. Given all this information we can derive a useful formula for locating any record. The nth record in a file will start in position p, where:

```
p=256*FH+(n-1)*RS
```

The file pointer (the value of PTR#) starts at zero. The first data record starts immediately after the last FDR block. So if the File Description consists of just one 256-byte block, the first record in the file will start in position 256. Check this out for yourself to make sure you understand this formula. Finally, this section contains the number of fields per record in the file.

INFORMATION ON FIELDS

The information you will need to store about the fields is more arbitrary. Each field will need to be identified by a name, but the maximum length of this name is up to you. We would recommend a maximum of 10 characters. The field width represents the maximum space available for a piece of data. It makes great sense to store all data in string format for uniformity of processing at this level, but note that this is less economical of storage space. A number like -12.345678 consists of 10 characters, but will only use 6 bytes if stored in a file as a real number. Nevertheless, string format is much to be preferred.

The third item of data about each field is the field type. This is not essential (our highly popular BEEBUG Filer database recognised only string data). However, a file type is useful, and will be helpful to us in future developments in this series. By marking each field with a simple code to indicate its type, the file handling program can treat different types of data in different ways. The most likely field types are shown in Table 2.

string	date
integer	logical
real	pointer

Table 2

Another alternative is to have a data type called *numeric*, and avoid the distinction between real and integer. As an example of the usefulness of this approach, consider the *date* type. Dates are most economically stored as a reversed 6 byte string (yymmdd). On displays and on printout it looks more friendly to present a date in the form:

```
<day>th <month> <year>
```

e.g. 27th June 1988. If a field is marked as of type date, it is then straightforward to write code which will recognise a date and automatically ensure that it is displayed in the 'long' form. The same can be applied on data entry, with a routine to automatically compress a date for storage in a file. Of course we pay for this convenience in the time taken to make the conversion.

A logical data type is one where only two possibilities exist (like TRUE or FALSE). An example is the common requirement to indicate the sex of a person (either male or female). The last data type we have suggested is *pointer*. This allows a record to have a pointer to either another record in the same file, thus opening the possibility of chaining records, or to a record in a completely separate file. The latter arrangement can avoid the unnecessary repetition of extensive data by storing the data once with a corresponding pointer in each record which would otherwise have contained a copy of these details.

An example might be a database of student records where the record for each student might contain a pointer to the course for that student in a separate *courses* file, or a pointer to the student's tutor in a further *tutors* file. In an example like this it does not take much imagination to see how a highly complex database might well emerge with many files involved and with many pointers between files.

Don't get too carried away. Large and complex databases like this require rather more power and storage capacity than the humble Beeb can offer, but the same principles can still be applied on a smaller scale.

PUTTING THEORY INTO PRACTICE

We will conclude this month's article by giving a procedure to create a data file according to data input by the user. We will use the code letters as integer variable names, and we will also assume that suitably sized arrays have been set up for the field information:

Fname$()	Field Names
Fwidth%()	Field Widths
Ftype%()	Field Types

The relevant procedure could then be defined as follows.

```
 1000 DEF PROCcreate_file(name$)
 1010 LOCAL F%,I%
 1020 FH%=(25+22*NF%)DIV256+1
 1030 RS%=0
 1040 FOR I%=1 TO NF%
 1050 RS%=RS%+Fwidth%(I%)+2
 1060 NEXT
 1070 OSCLI("SAVE "+name$+" 0+"+STR$~(25
6*FH%+FS%*RS%))
 1080 NR%=0:F%=OPENUP(name$):PTR#F%=0
 1090 PRINT#F%,FH%,FS%,NR%,RS%,NF%
 1100 FOR I%=1 TO NF%
 1110 PRINT#F%,Fname$(I%),Fwidth%(I%),Ft
ype%(I%)
 1120 NEXT
 1130 CLOSE#F%
 1140 ENDPROC
```

The five initial items in the FDR will be stored as integers, a total of 25 bytes. Storing field names, field widths and field types as described will require 12, 5 and 5 bytes for a total of 22 per field. Thus the size of the FDR will be (25+22*NF), and the number of 256 FDR modules required can then be calculated as in line 1020. Remember that in a file a string occupies two more bytes than the number of characters in the string, while an integer requires 5 bytes (this was discussed in detail last month).

Next, the size of each record consists of the total of the field widths, plus a two-byte overhead (again) for every field. This total is calculated in lines 1030 to 1060 to determine the value of RS. We can now create the correct size of file using OSCLI to pass a *SAVE command to the filing system. This simply calculates the size of the file in bytes and copies that number of bytes from memory to disc as the initial file. This is the quickest way of creating a file of a given size, though it will be full of garbage.

The procedure then opens the file just created, and outputs to the file the five initial parameters followed by the field information before closing the file and terminating the procedure. You might wish to set a flag to indicate successful completion of this process immediately before the exit from the procedure (or indeed convert the procedure to a function which returns such a flag). This can be useful for detecting situations such as insufficient disc space for the file specified.

Notice too that all the variables which we have chosen to define as integer are so marked in the procedure with the '%' sign. This ensures that their values are stored in the file in the correct integer format. If you assign an integer value to a real variable and then store that in a file it will be stored as a real number using a total of 6 rather than 5 bytes. This would ruin all our careful calculations.

In the above we have assumed that the contents of the arrays Fname$(), Fwidth%() and Ftype%(), together with the values of FS and NF have been correctly entered by the user before this procedure is called.

This is where we pause for breath, and give you an opportunity to digest all this detail before continuing further in the next issue. You might also like to consider the reasons for the present built-in limit of 255 characters on the size of any data field, and how this might be overcome. B

The magazine disc/tape contains a short demonstration of the use of PROCcreatefile.

USING ASSEMBLER PART 1

This month we begin the sequel to our Exploring Assembler series, in which we will cover practical applications of assembler programming on the Beeb. We begin with a look at machine code graphics.

Providing you do not need to write directly to the screen, implementing graphics routines in 6502 assembler for the BBC micro is a relatively straightforward affair. It involves repeated use of the OSWRCH operating system call to send sequences of VDU codes to the screen. To take a very simple example, the sequence VDU22,1 will select screen mode 1. This is performed in assembler as follows:

```
LDA #22
JSR oswrch
LDA #1
JSR oswrch
```

You may remember from earlier in the series that the OSWRCH call sends the contents of the accumulator to the screen. In this example it is assumed that the variable oswrch is set to &FFEE, the OSWRCH entry point. The code sends first the value 22, and then a 1 to the screen, and thus engages mode 1. When changing mode in this way you should remember that Basic pseudo-variable HIMEM is not reset in accordance with the new mode, so that there can be conflict between Basic workspace and screen RAM. Of course, if you are using machine code this is usually of little consequence.

Table 1 gives the VDU control codes associated with graphics functions. As you can see, many Basic commands used in graphics have a direct VDU equivalent. Thus for example to select cyan as the current text colour when in mode 2 (colour 6), we can issue VDU17,6. You can test this out from Basic, and can even string a sequence of codes together. For example:

```
VDU22,2,17,6,65
```

will set up mode 2, select cyan as the text colour, and print the letter "A" (ASCII 65). In assembler this becomes:

```
LDA #22 :JSR oswrch
LDA #2  :JSR oswrch
LDA #17 :JSR oswrch
LDA #6  :JSR oswrch
LDA #65 :JSR oswrch
```

VDU Code	EFFECT
12	CLS
16	CLG
17	COLOUR
18	GCOL
19	Logical to Physical Colour
20	Restore Default Colours
22	Mode Change
23	User Characters/Cursor on-off
24	Define Graphics Window
25	PLOT
26	Restore Default Windows
28	Define Text Window
29	Define Graphics Origin
30	Home Cursor
31	TAB(X,Y)

Table 1. VDU control codes associated with graphics functions.

GCOL

Using the equivalent of Basic's GCOL to set up graphics colours involves a similar process, except that an extra parameter is required. The syntax of GCOL is:

```
GCOL mode,colour
```

where *mode* is a number between 0 and 4 giving the plotting mode, and *colour* is the logical colour to be plotted. VDU18 is the code for GCOL, and to set up colour 6 in Exclusive OR plotting mode, we could send the sequence:

```
VDU18,3,6
```

In assembler this becomes:

```
LDA #18 :JSR oswrch
LDA #3  :JSR oswrch
```

Where graphics calls are repeatedly made in assembler, it usually pays to write them in the form of subroutines. These could take parameters passed in the accumulator and X and Y registers, where required. The following routine executes a generalised GCOL:

```
.gcol
LDA #18 :JSR oswrch
TXA     :JSR oswrch
TYA     :JSR oswrch
RTS
```

It is called with the X register holding the desired plot mode, and the Y register the colour number. On exit the contents of the accumulator are undefined, while the X and Y registers are preserved.

Another way to implement this, and one which illustrates the use of the PHA instruction introduced last month, is to call the routine with the required colour number in the accumulator, and with the X register holding the required plot mode. The new routine is as follows:

```
.gcol
PHA
LDA #18 :JSR oswrch
TXA     :JSR oswrch
PLA     :JSR oswrch
RTS
```

This version of the routine preserves all registers. It works by pushing the initial contents of the accumulator onto the stack at the start. It then sends first the value 18, and then the plot mode to OSWRCH before pulling the colour value off the stack, and calling OSWRCH for a third time.

LONGER VDU SEQUENCES

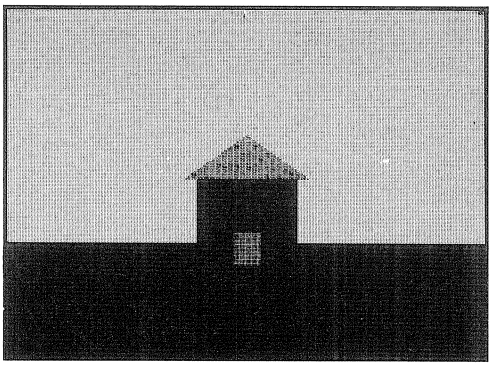
A similar technique can be applied to a variety of graphics calls, but in cases where a large number of parameters are required, it is cumbersome to load them each individually. A more convenient approach is to hold the parameters in a data block. By way of illustration, the subroutine below implements a cursor off command:

```
.coff
LDX #0
.loop
LDA base,X
JSR oswrch
INX
CPX #10
BNE loop
RTS
:
.base
EQUB 23 :EQUB 1
EQUD 0  :EQUD 0
```

In this routine the X register is used as a loop counter. Ten complete cycles of the loop are made, and with each the accumulator is loaded with a new value from the data block located at base, and then OSWRCH is called. The routine sends the equivalent of:

```
VDU23,1,0,0,0,0,0,0,0,0
```

Note the use of EQUB to place a single byte into the assembled code, and of EQUD to place four bytes.



Exactly the same technique could be used for VDU23 sequences designed for redefining characters. In this case the data block would take a very similar form, and might look like the following:

```
.base
EQUB 23:EQUB 255  Redefine chr 255
EQUD &818181FF
EQUD &FF818181
```

Now each pair of digits in the EQUD sequence represents a single byte of the eight bytes required to define the character. The important thing to remember here is that the lowest byte of each four-byte sequence is sent first. The data in this particular block will redefine character 255 as an empty box, and is the equivalent of:

```
VDU23,255,255,129,129,129,129,129,129,255
```

USING PLOT

We have intentionally left until last the most important of the graphics commands: the equivalent of Basic's PLOT. The syntax of the Basic command is:

```
PLOT n,x,y
```

where n is the plot number - an integer in the range 0 to 255 which defines the type of plotting action - and x and y are the graphics co-ordinates. The X co-ordinate ranges from 0 to 1279, and the Y co-ordinate from 0 to 1023. The VDU equivalent is:

```
VDU25,n,x;y;
```

The two semicolons in the sequence indicate that both x and y are 16-bit values rather than 8-bit bytes. When translating this into machine code, we must generally treat the two 16-bit co-ordinates as pairs of bytes, in which case the VDU string reduces to the following:

```
VDU25,n,x MOD 256,x DIV 256,y MOD 256,y DIV 256
```

In other words the x and y co-ordinates are sent low byte first.

Suppose we wish to plot a point on the screen at the centre. We could use any of:

```
PLOT 69,640,512
```

or:

```
VDU25,69,640;512;
```

or:

```
VDU25,69,128,2,0,2
```

In the latter case, the X co-ordinate is made up from 128+2*256, while the Y co-ordinate is exactly 2*256. We could translate this into assembler as follows:

```
LDA #25  :JSR oswrch
LDA #69  :JSR oswrch
LDA #128 :JSR oswrch
LDA #2   :JSR oswrch
LDA #0   :JSR oswrch
LDA #2   :JSR oswrch
```

Using sequences of code such as this to perform a plotting function is all very well if such plots are few in number. But in the majority of cases programs will need to make repeated calls of this kind to build up any given screen display. Broadly speaking there are two quite different ways in which we can efficiently incorporate multiple plotting calls. One approach is to build up a large data block of VDU codes, which will be sent to OSWRCH in sequence. An alternative approach makes use of a short plotting subroutine which is called once for every VDU25 call made, with parameters being passed in the 6502's registers or in zero page RAM. The former approach, to be examined below, is really only suited to situations where the values sent to OSWRCH are known at the time of assembly. The second, and more flexible approach, allows parameters to be processed during run time, and we will explore this in the next issue.

SENDING LONG VDU SEQUENCES

The first of the two methods is really very simple. The only skill involved is in putting the correct values into the data block, and arguably the best way to check that you have got the data correct is to test out the Basic equivalent before coding up the assembler version. For the purposes of illustration, listing 1 gives a short Basic program to draw a house with a red roof and magenta door. It is set in a field of green grass against a cloudless blue sky.

Listing 1

```
 10 REM .>Assem12-1
 20 REM:Basic house and grounds
 30 :
 40 MODE2
 50 GCOL0,132:REM Blue sky
 60 CLG
 70 :
 80 GCOL0,2:REM Green grass
 90 MOVE 0,0
100 MOVE 1279,0
110 PLOT 85,0,320
120 PLOT 85,1279,320
130 :
140 GCOL0,7:REM White house
150 MOVE 512,512
160 MOVE 512,256
170 PLOT 85,768,512
180 PLOT 85,768,256
190 :
200 GCOL0,1:REM Red roof
210 MOVE 480,512
220 MOVE 640,640
230 PLOT 85,800,512
240 :
250 GCOL0,5:REM Magenta door
260 MOVE 608,256
270 MOVE 608,352
280 PLOT 85,672,256
290 PLOT 85,672,352
```

The assembler equivalent is given in listing 2. The vast bulk of this consists of EQUB, EQUW and EQUD directives to build up the block of VDU data (maximum size 256 bytes). The working part of the program is itself very short. This just loads the X register with zero, and uses indexed addressing to load the accumulator with the first byte of the data block. This is sent to OSWRCH, the X register is incremented, and the process repeated. One trick well worth noting is that we have used the assembler itself to calculate how many data items there are. On line 160 the X loop is checked against the value endtable-table. The value endtable will have been assigned a value because we have inserted a label of just this name in line 470.

Listing 2

```
 10 REM .>Assem 12-2
 20 REM:M/C house and grounds
 30 :
 40 MODE1
 50 oswrch=&FFEE
 60 FOR pass=0 TO 1
 70 P%=&900
 80 [
 90 OPT pass*3
100 .start
110 LDX #0
120 .loop
130 LDA table,X
140 JSR oswrch
150 INX
160 CPX #endtable-table
170 BNE loop
180 RTS
190 :
200 .table
210 EQUB22:EQUB2        \Mode 2
220 EQUB18:EQUB0:EQUB132\Blue sky
230 EQUB16\CLG
240 :
250 EQUB18:EQUB0:EQUB2  \Green grass
260 EQUB25:EQUB4:EQUW0:EQUW0
270 EQUB25:EQUB4:EQUW1279:EQUW0
280 EQUB25:EQUB85:EQUW0:EQUW320
290 EQUB25:EQUB85:EQUW1279:EQUW320
300 :
310 EQUB18:EQUB0:EQUB7  \White House
320 EQUB25:EQUB4:EQUW512:EQUW512
330 EQUB25:EQUB4:EQUW512:EQUW256
340 EQUB25:EQUB85:EQUW768:EQUW512
350 EQUB25:EQUB85:EQUW768:EQUW256
360 :
370 EQUB18:EQUB0:EQUB1  \Red Roof
380 EQUB25:EQUB4:EQUW480:EQUW512
390 EQUB25:EQUB4:EQUW640:EQUW640
400 EQUB25:EQUB85:EQUW800:EQUW512
410 :
420 EQUB18:EQUB0:EQUB5  \Magenta Door
430 EQUB25:EQUB4:EQUW608:EQUW256
440 EQUB25:EQUB4:EQUW608:EQUW352
450 EQUB25:EQUB85:EQUW672:EQUW256
460 EQUB25:EQUB85:EQUW672:EQUW352
470 .endtable
480 ]
490 NEXT
500 CALL start
```

The data itself needs little comment, but to illustrate the way in which it has been encoded, take a look at line 280. This performs the equivalent of:

```
PLOT 85,0,320
```

or of:

```
VDU25,85,0;320;
```

The first two bytes have been encoded as:

```
EQUB25:EQUB85
```

But the next two values have been encoded using EQUW. This is because the VDU sequence expects two bytes to be used for each of the position co-ordinates of the PLOT. The EQUW directive handles this automatically, storing the low byte of the two-byte co-ordinates first, which is exactly the order demanded by the VDU sequence. To verify the way in which it works, you may like to supply the house with a chimney pot, windows and a garden path.

The EQU family of assembler directives are not available on Basic I. See BEEBUG Vol.7 No.2 for suitable conversion routines. To check whether you have Basic I or not, switch on your machine, and type REPORT. If the copyright message is 1981, you have Basic I.

Next month we will continue with the graphics theme, developing subroutines for drawing a variety of objects. B

BEEBUG WORKSHOP Linked Lists Part 1

David Spencer deals with a very important data storage technique which is often misunderstood.

When programmers design a new masterpiece, they tend to concentrate on how the various tasks within the program should be performed, and think very little about how best to store the data used by the program. This is a great pity, because as we shall show, a little thought in this area can greatly improve performance.

There are two different stages to the design of data storage systems. Firstly, you must decide how to store a single data record. For example, in a database program you have to design a structure that will allow all the various fields to be stored. This topic is covered in this month's File Handling article. Secondly, you must design a system to link together individual data records. The way this is done depends a lot on how the data is to be used. It is this second subject that we will be concentrating on in this, and subsequent, workshops.

ARRAYS

The only basic data structure available in BBC Basic is the array. This is merely a collection of records all with the same data type. What's more, in Basic, this means that the elements can only be real, integer, or string. An element within an array is referenced by its position in the array. For one dimensional arrays, this is simply a number specifying where the element appears in the list. In two dimensions, you need two numbers, one to specify the row in which the element appears, and the other to specify the column. For three dimensions there are three numbers, and so on.

Arrays have several advantages over other methods of data storage. For one, they are fairly easy for beginners to understand. Arrays can also be accessed very quickly. Given the start address of the array in memory, the element we want to find, and the length of each element, we can calculate its position and access it directly.

There are however, quite a few disadvantages to using arrays. We have already said that in Basic the elements of an array can only be one of three simple types. While other languages, such as Pascal and C, allow the elements of an array to be of any data type, even another array, there are no languages that allow a mixture of different data types for different elements within a single array. Further, almost all languages insist that an array must be defined before it is used, and that the definition must say how many elements the array will contain. Often, you will be uncertain of how many elements will be needed, for example in a database, and so you must allocate the largest size allowed. This will mean that storage technique is often very wasteful of space.

What is needed is some way of storing a set of records, all of which are possibly in different formats, and linking them together somehow. If we can do this, each record can still be treated as an individual entity, but it is also possible to perform operations such as finding the next record after the current one, or finding the nth record. The answer to all our problems, (for the time being anyway), is the Linked List.

As the name suggests, a linked list is a list of records that are linked together. The key to this 'linking' is the use of pointers, which will be

familiar to anybody used to C or Pascal, but which is less common in Basic. To understand pointers, you must first of all realise that when a record is stored in memory, the fundamental way of referring to that record is by its address in memory. When you refer to a variable by a variable name, all that really happens is that the language uses its name to find the address in memory of the associated data. For any particular record stored in memory, it is possible to take the address of that record, and store it in another variable in some way. The content of this second variable then becomes a 'pointer' to the original record.

Assembly language programmers will already be familiar with the concept of pointers, under the guise of indirect addressing. For example, the instruction:

```
STA &70
```

will store the contents of the accumulator in location &70 of memory. Now consider the instruction:

```
STA (&70)
```

The addition of the brackets makes a major difference to the action taken by the instruction. What happens now, is that the processor takes the contents of locations &70 and &71, and makes them into a sixteen bit address. It then stores the contents of the accumulator in the location pointed to by this new address. What has happened, is that instead of using location &70 as the actual record, the processor has used the contents of this, and the next location, as a pointer to the data record.

If you stop and think for a moment, you should be able to come up with some reasons why pointers are so important. For example, consider the situation where you want to sort a large number of lengthy records into order. Most sort routines involve a great many swaps between records, and if you had to swap the entire record each time, the sort could be very time consuming indeed. However, what about keeping the records in one place all the time, and having a set of pointers to the records? This way, rather than actually swapping records, all we need to do is to swap the pointers to those records. This will in general be much faster, because there is less data to swap. As always, there is a slight drawback. Now that we are referring to the records by means of pointers, accessing each record is slightly more time consuming. Instead of going straight to a particular record, you now have to go to the pointer for that record, and use that pointer to locate the record. However, the extra time this takes will most likely be insignificant when compared to the time saved through not shunting large records around.

Now that we know roughly what a pointer is, we can show how these can be used to implement linked lists. This is really quite simple. Given a set of records, the first thing to do is to add an extra field to each record, this new field being a pointer. Normally, this pointer field will be placed at the start of each record, so that when the location of a record is known, the pointer field can be read without knowing such things as the length of the particular record. Then all you have to do to connect the records together into a linked list is to make the pointer field of each record point to the next record. This is shown in figure 1, with the arrows representing pointers.

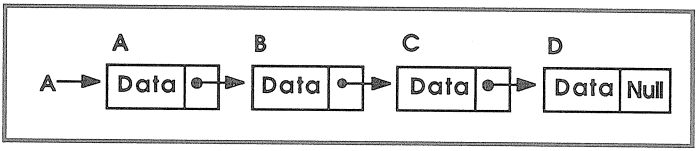


Figure 1. The structure of a linked list.

The first and last records have to be considered as a special case. There must be some way of referring to the first record in the list, because no other record points to it. This can be easily be achieved done by having a separate pointer that points to the first record, although there is a better method which we will explain in the next workshop. The other special case is the pointer field of the last record. Because there are no more records to point to, this last pointer should have a value which can be identified as marking the end of the list. This is traditionally called the 'null' pointer, though its actual value depends on such things as the language in use. We will always give the null pointer a value of

0, because our pointers directly represent memory addresses, and it is very unlikely that a record will be located at location 0 in memory.

READING A LINKED LIST

To read all of the records in a linked list, or to get at just one record in the middle, is really quite simple. You start by taking the initial value of the pointer to read the first record. You can then read the contents of that record, and take the pointer field from that record and use it to find the next record. This is then repeated until you get to the null pointer, showing that the whole list has been read. If you want to read the nth record, you just skip through the list, taking notice only of the pointer fields, until you get to the record you want, and then just read that as before.

ADDING TO THE END OF A LIST

Similarly, it is fairly straightforward to add an extra record to the end of a linked list. Firstly, the new record must be stored in memory somewhere, complete with a null pointer field. Then, search through the current list until the null pointer is found, and change it so that this points to the new record.

LINKED LISTS IN BASIC

Having just covered the theory, you may be wondering how you can actually implement and use a linked list from within a Basic program.

Clearly, there is no way of directly implementing a linked list in Basic. One possibility is to use an array to store the actual data, and another array to store pointers to that data. This can however get very messy, and by far the best method is to use indirection operators to access a block of memory directly. The linked list can then be built up within this block.

The manipulation of a linked list is best illustrated by an example. The program given here creates a linked list of words which must be typed in, and when Escape is pressed prints the list out. Line 100 allocates a block of memory to use for the list. There is no checking on the size of the list, so if it runs over the 1000 bytes allocated, the program's variables will be corrupted. Line 120 sets up an empty record at the start of a list. This is so that should Escape be pressed before any words are entered, the program will still function correctly. The variable 'free' is set up to contain the address of the next free byte in our block of memory, which initially is at the start of the block. Words are entered into the list by lines 130 to 210, which take a word, search through for the end of the list, and then add the word as a new 'record'. Each record consists of a 4 byte pointer, followed by the actual word, terminated with a carriage return. This allows the pointer to be accessed using !, and the word to be accessed with the $ indirection operator. Lines 230 to 280 print out the contents of the list, by starting at the first record and going through until the null pointer is reached.

Next month, we will look further at linked lists, and in particular, we will show how to add and delete records in the middle of a list.

```
   10 REM Program Linked List Demo
   20 REM Version B 1.0
   30 REM Author  David Spencer
   40 REM BEEBUG  July 1988
   50 REM Program subject to copyright
   60 :
  100 DIM list 1000
  110 ON ERROR IF ERR=17 THEN 230 ELSE R
EPORT:END
  120 free=list:!list=0:$(list+4)=""
  130 REPEAT INPUT word$
  140 ptr=list
  150 REPEAT
  160 IF !ptr<>0 THEN ptr=!ptr
  170 UNTIL!ptr=0
  180 !ptr=free
  190 !free=0:$(free+4)=word$
  200 free=free+LEN(word$)+5
  210 UNTIL FALSE
  220 :
  230 PRINT''
  240 ptr=list
  250 REPEAT
  260 PRINT$(ptr+4)
  270 ptr=!ptr
  280 UNTIL ptr=0                    B
```

COMMUNICATING IN PACKETS

For the uninitiated Peter Ball conveys some of his new found enthusiasm for packet radio, and the freedom of the airways.

Things have been quite exciting here lately in what has come to be known as my *technology centre*. Let me explain. At one end of my study is my computer, a BBC Master 128 with the usual array of peripherals. For my Christmas present last year my wife bought me a Watford Apollo modem and associated software and I have been probing the depths of Prestel, Telecom Gold and a host of independent databases. This has been with increasing anxiety because of rapidly escalating phone bills.

At the other end of the study is my ham radio gear. I had been persuaded to take my radio amateur's licence (though you can *receive* any broadcast without a licence) last year but, in all honesty, I can't work up a lot of enthusiasm for general chat about my rig, the aerial and how much power I am putting out, and noting in the log the same details for whomsoever I manage to contact on the air.

It seemed to me that the two ends of the study needed connecting together, and this should be possible with the recent developments in what is called packet radio. It's a bit like communicating with a modem but using radio rather than a phone line. So I bought a bit of gadgetry known variously as a Terminal Node Controller (TNC) or Packet Modem.

Now, as we technical people know only too well, you don't connect everything together, throw a big switch and expect everything to work. So bit by bit the various parts were made to 'talk' to each other, and at last it seemed that the throwing of the big switch might be possible. It was.

The various parameters were input, mainly relating to baud speed of the different elements. The VHF transceiver was set to 144.65 MHz, the amateur radio frequency used primarily for packet radio. At the computer keyboard the system was put into *monitor* mode in the hope that I could decipher something from those bursts of data that could be heard in the loudspeaker. Immediately a flood of information came pouring in that was much too quick to read on the screen. No problem; the whole stream could be diverted to the disc to be studied at leisure.

What was immediately apparent was that well within radio range were two packet *mailboxes*, one at Stevenage and another at the Radio Society of Great Britain (RSGB) HQ at Potters Bar (I now know that there is a network of several dozen of these mailboxes around the country). After more reading of the manual for the TNC, and a very good American book on the subject, it seemed I could try to connect to one of these mailboxes. I typed in CONNECT G4SPV (the callsign of the Stevenage station) and almost immediately came back a reply saying that I was indeed connected.

It further told me that I was not recognised by the mailbox and would I enter my first name. I did as I was bid, and immediately I was welcomed as Peter (as I have been ever since) and would I please enter my commands, for which a string of single letter options were given. This was a bit like the genie of the lamp, "What is your wish, oh master?" but I wasn't sure what wishes I had. For most of my wishes the system came back and told me that I had given it an INVALID COMMAND. Eventually it gave up in disgust with my efforts, and timed out with the statement that I was DISCONNECTED.

Back to read the written text, and the next attempt was more successful. I found that if I simply entered a question mark or H for HELP it would guide me through the various procedures. So we were in business and since then the whole thing has been addictive. It is really astounding how much traffic there is to read on subjects ranging over Sir Ranulph Fiennes attempts to get to the North Pole, the current state of the Russian MIR satellite, and how to cure RFI (radio frequency interference) on your home computer.

The wonder is that all of this is totally error free; the system works on standards well known to data transmission engineers as X25. When monitoring traffic, the repeats resulting from the error correcting processes are all received, but it is a simple procedure with the word processor to remove these and get absolutely clean copy. When receiving traffic directly addressed to you the copy is totally clean every time.

I find that it is possible not only to communicate with the local mailboxes but also, via them, to communicate with mailboxes over the whole country and even, via a gateway at the University of Surrey and the UOSAT2 satellite, to mailboxes anywhere in the world.

There are quite a number of versions of TNC (packet modem) available, although most now operate to the AX25 standard protocol (the amateur radio version of X25), and will therefore talk to each other. I am using one made by Kantronics and this and other varieties are available from Ham Radio suppliers. There are also various terminal programs available specifically designed to drive the TNC, but I have found no need for these and find that the Watford Apollo ROM does all I need with the added advantage that it is a simple matter to switch to Prestel, etc. using the same ROM.

Further information on Packet Radio, including specialist publications, can be obtained from the Radio Society of Great Britain (RSGB) at Lambda House, Cranborne Road, Potters Bar, Herts EN6 3JW.

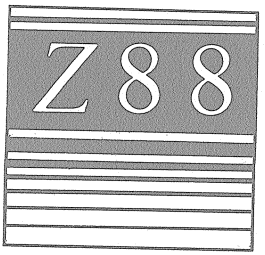
Now I'm very much a beginner in all this, but then nobody has a lifetime of experience in packet radio, have they? I am all excited. Is anybody else? Are there any experts who would like to give me more guidance? Are there any beginners who would like to know how I have got as far as I have? I would be delighted to communicate with them either by traditional communication channels or via packet - G1YFL @ G4SPV-2 or @ GB3HQ-2. B

Roll-Down Screen Displays (Continued from page 47)

```
1510 .rd2 JSR doroll
1520 INC &82:LDA &82
1530 TAX:TAY:INY
1540 DEC &83
1550 BNE rd2
1560 LDA &70:STA &83
1570 ASL &83:TXA:SEC
1580 SBC &70:TAX:TAY
1590 .rd3 JSR dobyterow
1600 INY:INX
1610 DEC &83:BNE rd3
1620 RTS
1630 .doroll
1640 LDA &70:STA &75
1650 .rd4 JSR dobyterow
1660 INY:DEX
1670 DEC &75
1680 BNE rd4
1690 TXA:TAY
1700 JSR dobyterow
1710 RTS
1720 .blinds
1730 STZ &77
1740 LDA #16:STA &78
1750 .b1 LDA #16:STA &83
1760 .b2 LDA &77:TAX:TAY
1770 JSR dobyterow
1780 LDA &77:CLC
1790 ADC #16:STA &77
```

```
1800 DEC &83:BNE b2
1810 INC &77
1820 DEC &78:BNE b1
1830 RTS
1840 .dobyterow
1850 LDA &2E00,X:STA &D6
1860 LDA &2F00,X:STA &D7
1870 LDA &2E00,Y:STA &80
1880 LDA &2F00,Y:STA &81
1890 LDA #80:STA &76
1900 PHY:LDY #0
1910 .show
1920 LDA (&80)
1930 JSR OSWRSC
1940 .incmemptr
1950 LDA &80:CLC
1960 ADC #8:STA &80
1970 LDA &81:ADC #0
1980 STA &81
1990 .incscrptr
2000 LDA &D6:CLC
2010 ADC #8:STA &D6
2020 LDA &D7:ADC #0
2030 STA &D7
2040 DEC &76:BNE show
2050 PLY
2060 RTS
2070 ]:NEXT
2080 ENDPROC
```

B



In the Z88 page this month, David Spencer continues his look at the VDU drivers.

CURSOR POSITIONING

It is often essential to be able to position the cursor at either a particular point on the screen, or at a particular row or column. This is easily accomplished on the Z88 using one of three commands. The first:

```
VDU 1,51,64,32+x,32+y
```

will move the cursor directly to the character at (x,y). The point (0,0) is the top left corner of the application window (the area used for printing by Basic), and x can be any value up to 93, while y must be in the range 0 to 7. The next command:

```
VDU 1,50,88,32+x
```

moves the cursor to the given x coordinate (column), whilst remaining on the same row. Similarly:

```
VDU 1,50,89,32+y
```

moves the cursor to the given y coordinate (row), keeping in the same column.

HIGHLIGHTING TEXT

BEEBUG Vol.7 No.1 showed the use of various codes to select highlights such as underlining. Once an effect has been selected it is applied to all the text that is printed until the effect is turned off again. There is however another way of using four of these effects (Reverse, Grey, Flash and Underline) which allows you to change the style of text already on the display. This technique is a four stage process:

1) Move the cursor to the text to be highlighted
2) Select the effect, or effects, to be applied.
3) Execute a command to highlight a certain number of characters.
4) De-select the effects.

The cursor moving is done with the VDU sequence given above, and the highlights are selected and de-selected using:

```
VDU 1,ASC"<letter>"
```

where <letter> is the first letter of the effect in upper case. For example, flashing is selected and de-selected using:

```
VDU 1,ASC"F", or VDU 1,70
```

The command to highlight the characters takes one of two forms. The simplest is:

```
VDU 1,50,65,32+n
```

which will apply the current attributes to the 'n' characters starting at the cursor. The second form of the command is:

```
VDU 1,50,69,32+n
```

which instead of directly applying the effects, Exclusive ORs the current effects with those that the characters already have. This means for example, if grey is currently selected, any of the 'n' characters not grey will be made so, and any already grey will return to normal. This is very useful for things such as reversing out menu options.

As an example, try out the following program:

```
10 CLS:VDU1,ASC"C"
20 ON ERROR VDU1,50,ASC"+",ASC"C",1,51,
   ASC"-",ASC"U",ASC"R":END
30 VDU 1,51,64,40,35
40 PRINT "BEEBUG   BEEBUG   BEEBUG"
50 REPEAT VDU1,51,64,40,35
60 VDU1,ASC"U",1,50,65,38,1,ASC"U"
70 VDU1,50,88,58
80 VDU1,ASC"R",1,50,69,38,1,ASC"R"
90 A=GET:UNTIL FALSE
```

You should be able to go through this line by line, and predict what it will do in the light of the commands described above. The only VDU codes not covered in this article are the use of VDU1,ASC"C" to turn the cursor off, and the string of codes in line 20 which ensure that the effects are switched off, and the cursor turned on, if Escape is pressed.

There are a couple of points to note when using this technique. Firstly, the operation of highlighting the 'next n' characters moves the cursor right by that number of characters. Secondly, this method only works with display attributes, not character attributes. For example, you can 'grey out' a string of characters because that is a display effect, but you cannot embolden the same string, because bold is achieved by using a different character set when the characters are printed. Similarly, you cannot change a group of characters to use the tiny font, without reprinting them. B

POSTBAG

POSTBAG

SYSTEM GAMMA - A PERSONAL VIEW

I read with interest Geoff Bains' review of System Gamma (BEEBUG Vol.6 No.9), and I would like to reply to the first part of what he says.

It is not a "con and extremely silly" that the Programmers Reference Guide is sold as an extra. Both System Delta and System Gamma are sold as general programs, and most of these will be used by non-programmers. It is not fair that these 90% plus users should be penalised for the inclusion of this manual. You must also bear in mind with regard to its price that the Reference Guide is complete with technical support, and is not simply the extra cost of printing.

Secondly, the inclusion of the Programmers Reference Guide would automatically lead to the package being labelled as being for programmers, and most non-programmers would not give the program a second glance. I personally do not like the way in which reviewers continually try to label System Delta and Gamma as packages *for programmers*. I am sure that you agree System Gamma is so simple to use that even a school child could pick it up and produce graphs in a very short time.

Nova Fisher
Minerva Software

Through lack of space we have had to omit sections of the letter received from Minerva, but believe the summary above is a fair representation of the views expressed. Geoff Bains responds as follows.

"If the real expense of the Manual/Support package is the support then why not charge extra for that alone? I still stand by my remark that including the manual in the System Delta package would only increase the total price of the package by a very small amount (due to supply in large numbers, etc.).

"That way, support would not be charged for those not needing it (or in Minerva's terms - not 'penalising' those not wanting it).

"A programmers' reference manual and the support are not inseparable. Quite the opposite. Those that will really benefit from the manual (i.e. 'programmers') are less likely to want extensive support than those merely 'using' the applications packages. The programmers tend to be able (certainly with a manual!) to work it out for themselves.

"Neither have I said that System Gamma (or Delta) is a programming language only, nor that the applications programs are not simple to use. However, System Gamma is based around a programming language. Because of this it is longer and more complicated and therefore more expensive. Since buyers have to purchase the expensive programming language and since Minerva has gone to the (exemplary) effort of writing it, shouldn't they get the full benefit of it?"

Geoff Bains

MULTI-COLUMN ON THE MX80

There must be many Epson MX80 and similar printer owners who would like to use your most effective multi-column print routine (Vol.7 No.1). The following modifications to the program will enable them to do so.

```
 150 VDU2,1,27,1,64,1,27,1,Q%,3
2070  CLS:PRINTCHR$130"Enter
print mode (N/E/C)"
2080 PRINTCHR$130" (Normal/Emph
asized/Condensed)";TAB(25,0)CHR
$131;
2090 P$=GET$
2092 IFP$="N"THEN mlin%=80:Q%=
64:GOTO2100
2094 IFP$="E"THEN mlin%=80:Q%=
69:GOTO2100
2096 IFP$="C"THEN mlin%=132:Q%=
15:GOTO2100
2098 GOTO 2080
2100 PRINT P$'
2120 PRINT'TAB(5)CHR$134"Charac
ters/line:" CHR$133;mlin%
```

Lines 2092 to 2098 are the new ones. Note that the original line 2110 should be deleted.

Derek Lucas

We have checked this out and it works well. Our thanks to Derek Lucas for his efforts. B

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

In future we will be awarding five pounds to every hint or tip published, and fifteen pounds to what we consider to be the best hint or tip of the month. Hints may be submitted on any aspect of the BBC micro or Master series. So, why not put pen to paper, and make the benefit of your experience available to other BBC micro users.

REMEMBER YOUR 'THEN'S
by Stuart Lafosse

Nowadays it is common programming practice to leave the THEN out of a:

```
IF <condition> THEN
<operation>
```

statement. In most cases this is perfectly acceptable. However, the THEN must not be forgotten if you are using any of the Beeb's pseudo variables such as HIMEM, PAGE or LOMEM. For instance the statement:

```
IF PAGE<>&E00 THEN
PAGE=&E00
```

may not be shortened to:

```
IF PAGE<>&E00 PAGE=
&E00
```

This is simply because pseudo variables are given different tokens depending upon whether they are on the left or right of the THEN command. If there is no THEN command the second pseudo variable will be given the wrong token. The same is true with star commands.

ROM CLASHING (Revisited)
by S.Harp

As more and more ROMs become available and the software becomes more sophisticated, the likelihood of ROMS clashing with one another becomes even more likely. Master owners have the *UNPLUG command to turn offending ROMs off, but unfortunately the model B has no such command.

However the machine keeps a sixteen byte table, from &2A1 to &2B0, so that it knows exactly what ROMS are present at any one time. This table may be directly poked to 'unplug' a ROM from the machine, and stop it from interfering with other software.

Location &2A1 represents ROM socket 0, &2A2 ROM socket 1, and so on up to &2B0 which represents ROM socket 15. Find out which socket the offending ROM is plugged into and poke the relevant location with zero. For instance, to turn off the ROM in socket five enter the command ?&2A6=0. This command will not survive pressing Break or Ctrl-Break, so this is the simplest way to recover a ROM when required.

If you would like a slightly more automated method of controlling the ROMs in your machine an excellent ROM Controller program was published in Vol.5 No.3.

EDWORD EPROM
by Malcolm Chisholm

One major drawback with the Edword word processor, popular in education, is that it uses many disc-resident files to carry out its commands. There are 27 possible commands, leaving space for only four document files on a DFS disc. This problem may be overcome by turning these files into a form where they can be placed in a Sideways ROM. If you do not have access to an EPROM programmer the ROM image may be loaded into sideways RAM. Watford Electronics' ADDER package will allow the files to be arranged into the right format.

Still on Edword, there have been two versions of the software. Unfortunately files created on one version will not work on the other. Files originating from the first version require an execute address of &FFFE while files originating from the second version require an execute address of &FFFB. To transfer files between the two versions simply use a disc editor to change the execute addresses.

B

BEEBUG MEMBERSHIP

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

BEEBUG SUBSCRIPTION RATES

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

BACK ISSUE PRICES

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

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

FURTHER DISCOUNTS

We will allow you a further discount:

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

POST AND PACKING

Please add the cost of p&p:

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

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

BEEBUG MAGAZINE is produced by BEEBUG Ltd.

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

CONTRIBUTING TO BEEBUG PROGRAMS AND ARTICLES

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

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

In all communication, please quote your membership number.

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

Magazine Disc/Cassette

JULY 1988 DISC/CASSETTE CONTENTS

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

3D SHADES AND SHADOWS - create solid looking objects with shading and shadowing.

ASTROLOGICAL BIRTH CHARTS - have fun with this program to display astrological birth charts complete with all the signs of the zodiac.

MATRICES IN BASIC - a first program to implement a series of matrix-handling commands in Basic.

FILE HANDLING FOR ALL (Part 3) - a program to demonstrate the process of file creation.

FIRST COURSE
•FREE & •MAP - two short utilities to implement new star commands for DFS systems.

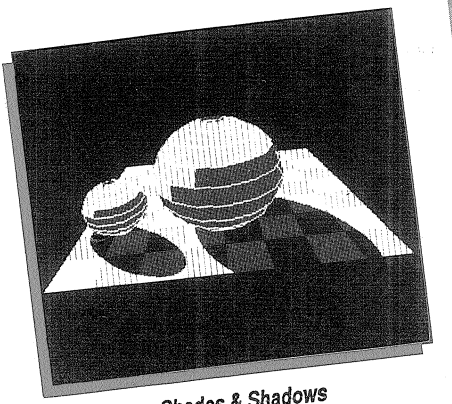
THE MASTER SERIES
ROLL DOWN SCREEN DISPLAYS - implement the code for two striking ways of putting graphic displays on the screen.

PASSING ARRAYS TO FUNCTIONS AND PROCEDURES - a demonstration of a clever technique for passing arrays as parameters to procedures and functions.

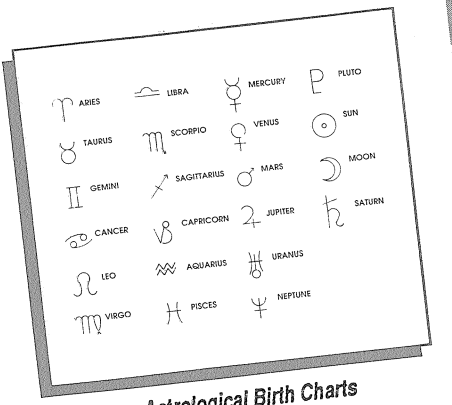
USING ASSEMBLER (Part 1) - two programs showing how Basic's graphic commands may be implemented in assembler.

BEEBUG WORKSHOP - a demonstration of list processing, displaying the elements in a linked list.

MAGSCAN - bibliography for this issue (Vol.7 No.3).



Shades & Shadows



Astrological Birth Charts

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

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

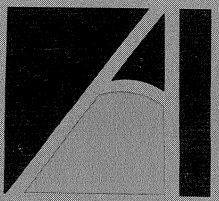
Prices are inclusive of VAT and postage as applicable. Sterling only please.

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

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

5% Discount On ALL Archimedes From BEEBUG

In Addition To...



The Archimedes Specialists

1 0% FINANCE

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

	Deposit	9 Payments		Deposit	9 Payments
A305			A310		
A305 Base	£79.66	£76.00	A310 Base	£93.24	£91.00
A305 Mono	£91.21	£82.00	A310 Mono	£104.79	£97.00
A305 Colour	£104.01	£100.00	A310 Colour	£117.59	£115.00
A310M			A440		
A310M Base	£104.79	£97.00	A440 Base	£278.93	£276.00
A310M Mono	£107.34	£104.00	A440 Mono	£290.48	£282.00
A310M Colour	£129.14	£121.00	A440 Colour	£303.28	£300.00

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

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

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

Please phone for allowances on other Compact and Master systems.

3 EXPORT

Although unable to offer finance to overseas customers, we can offer an efficient export service with delivery to your door. Please write for a quotation.

4 FREE PC EMULATOR AND 1st WORD PLUS

Purchase your Archimedes by Cheque, Access, Visa, Official Order or 11.5% finance and we will supply you, absolutely free, 10 3.5" discs, a lockable disc storage box, printer lead and the latest version of The PC Emulator from Acorn. Additionally if you are purchasing a 440 system you will receive 1st Word Plus.

Prices Including VAT

A305 Base	£763.66	Mono £829.21	Colour £1004.01
A310 Base	£912.24	Mono £977.79	Colour £1152.59
A440 Base	£2762.93	Mono £2828.48	Colour £3003.28

5 11.5% FINANCE OVER 12 TO 36 MONTHS

As a Licensed Credit Broker we are able to offer finance on the purchase of any equipment, including the Archimedes. You still benefit from the free PC Emulator, discs, disc box and printer lead. (Typical APR 23% on the purchase of a 310 Colour system over 36 months.
Deposit £152.59 36 payments of £37.36).

6 DISCOUNTS FOR EDUCATION

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

These discounts are available to you as a member of RISC User, THE leading magazine dedicated solely to Archimedes. So if you are not yet a member, join now for only £14.50 for a full year.

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

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

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

I enclose a cheque value £.....

Please debit my Access/Visa/Connect Card No

Expiry...../..... with £.....

Name .....

Address .....

.....

Signature .....

Beebug, Dolphin Place, Holywell Hill, St. Albans, Herts AL1 1EX. Tel: 0727 40303

BEEBUG – The Archimedes Specialists