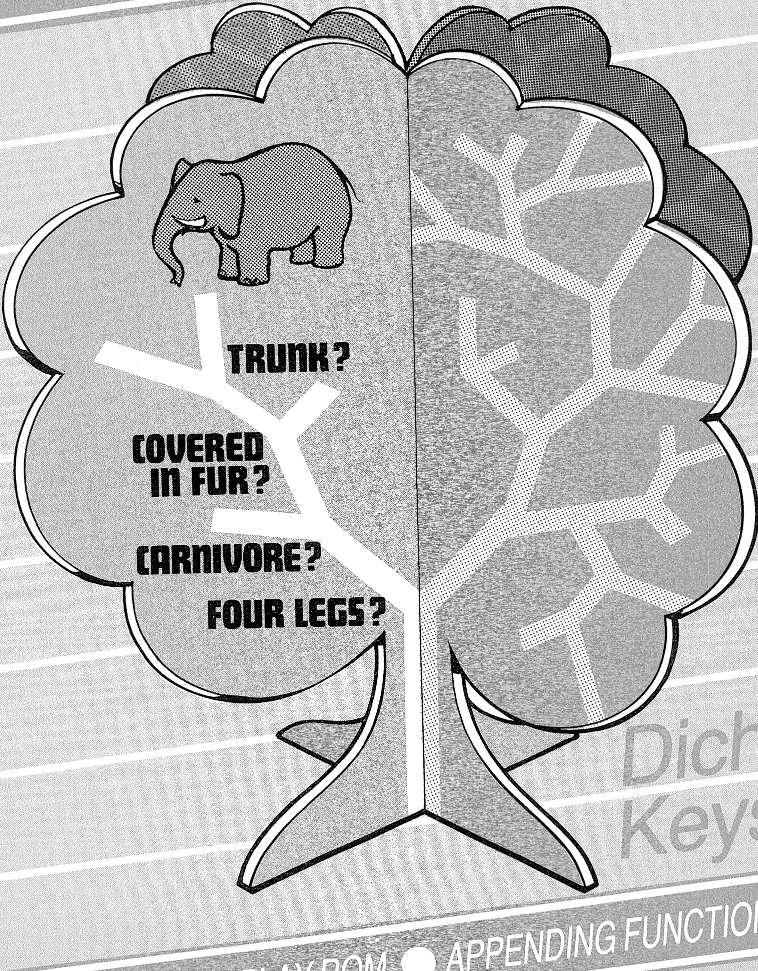
Vol.8 No.8 January/February 1990

BEEBUG

FOR THE BBC MICRO & MASTER SERIES



Dichotomous Keys

- MASTER DISPLAY ROM
- APPENDING FUNCTIONS & PROCEDURES
- THE ROM FILING SYSTEM
- EXPLORING COMMUNICATIONS

BEEBUG Vol.8 No.8 January/February 1990

FEATURES

REVIEWS

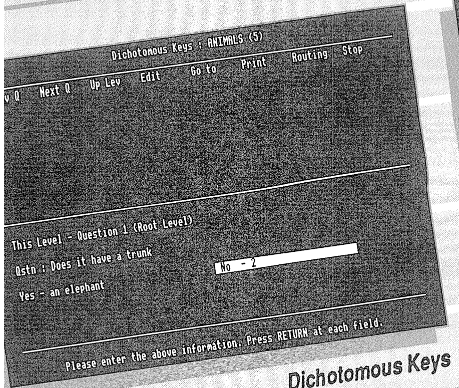
REGULAR ITEMS

HINTS & TIPS

PROGRAM INFORMATION

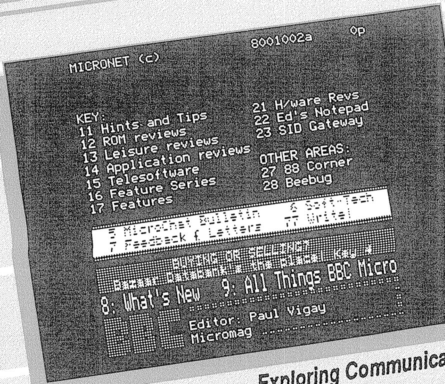
All listings published in BEEBUG magazine are produced directly from working programs. They are formatted using LISTO 1 and WIDTH 40. The space following the line number is to aid readability only, and may be omitted when the program is typed in. However, the rest of each line should be entered exactly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

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



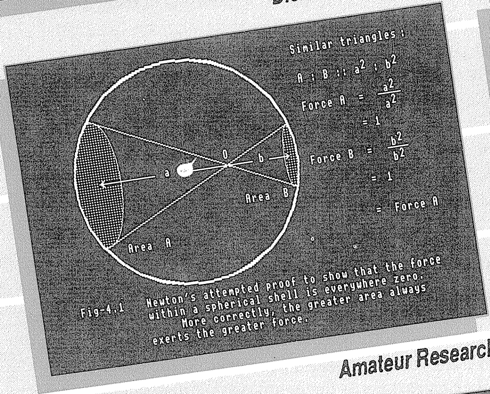
```
                    Dichotomous Keys : ANIMALS (5)
v Q   Next Q   Up Lev   Edit    Go to    Print    Routing   Stop

This Level - Question 1 (Root Level)
Qstn : Does it have a trunk
                                      No  - 2
Yes - an elephant

    Please enter the above information. Press RETURN at each field.
```

Dichotomous Keys



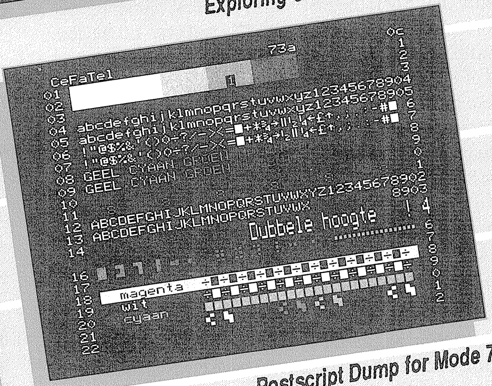
```
MICRONET (c)                    8001002a     0p

KEY:                          21 H/ware Revs
11 Hints and Tips             22 Ed's Notepad
12 ROM reviews                23 SID Gateway
13 Leisure reviews
14 Application reviews        OTHER AREAS:
15 Telesoftware               27 88 Corner
16 Feature Series             28 Beebug
17 Features
 5 MicroChat Bulletin    6 Soft-Tech
 7 Feedback & Letters   77 Write!
       BUYING OR SELLING? Key 4
       Databank & the place!
8: What's New   9: All Things BBC Micro
BBC  Editor: Paul Vigay
     Micromag
```

Exploring Communications



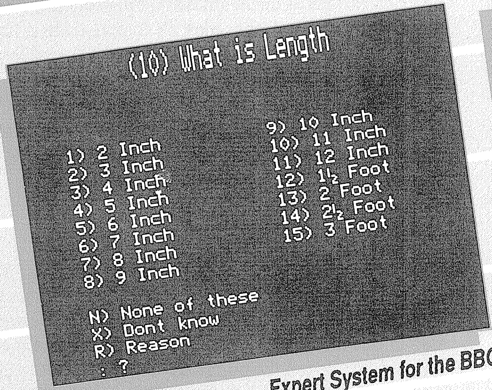
```
                                   Similar triangles :
                                   A : B :: a² : b²
                                   Force A  =  a²/a²
                                            = 1
                                   Force B  =  b²/b²
                                            = 1
                                            = Force A

Fig-4.1   Newton's attempted proof to show that the force
          within a spherical shell is everywhere zero.
              More correctly, the greater area always
          exerts the greater force.
```

Amateur Research



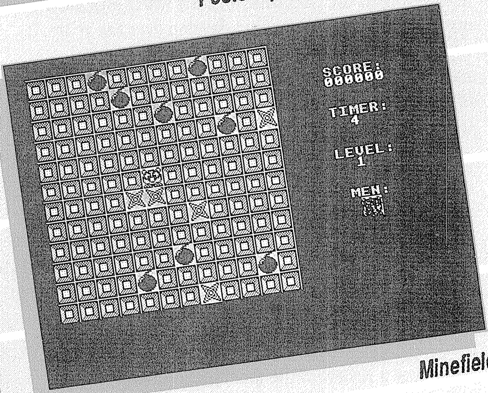
```
                         73a                 0c
CeFaTel                                       1
                                              2
                                              3
01                  1                         4
02            rstuvwxyz1234567890             5
03 abcdefghijklmnopqrstuvwxyz1234567890 -#█   6
04 abcdefghijklmnopqrstuvwxyz←£↑... -#█       7
05 !"@$%&'()0=?/-><=█+*¾→½‖¼←£↑.              8
06 !"@$%&'()0=?/-><=                          9
07 GEEL CYAAN GROEN                           0
08 GEEL CYAAN GROEN                           1
09                    QRSTUVWXYZ1234567890    2
10                                   8903     3
11 ABCDEFGHIJKLMNOPQRSTUVWX              !    4
12 ABCDEFGHIJKLMNOPQRSTUVWX  Dubbele hoogte   5
13                                            6
14                                            7
                                              8
16                                            9
17                                            0
18  magenta                                   1
19  wit                                       2
20  cyaan
21
22
```

Postscript Dump for Mode 7



```
(10) What is Length

1) 2 Inch          9) 10 Inch
2) 3 Inch         10) 11 Inch
3) 4 Inch         11) 12 Inch
4) 5 Inch         12) 1½ Foot
5) 6 Inch         13) 2 Foot
6) 7 Inch         14) 2½ Foot
7) 8 Inch         15) 3 Foot
8) 9 Inch

N) None of these
X) Dont know
R) Reason
: ?
```

Expert System for the BBC



Minefield

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

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

Program will not function on a cassette-based system.

Program needs at least one bank of sideways RAM.

Program is for Master 128 and Compact only.

Editor's Jottings

It is quite thought provoking that in a couple of issues time we shall be concluding volume eight of BEEBUG magazine and starting on our ninth year of publication. I am sure that when the BBC micro was first launched, no one dreamed that it would eventually sell over 1.5 million. And with sales of the Master 128 still continuing, total sales of the eight bit BBC micro must by now have reached the two million mark. One wonders where all these machines are today. No doubt many now languish unused, but many more are still in active use and likely to continue so for a good while to come. Indeed, the influence of the BBC micro on personal computing in this country must be enormous, and many of those now in business or industry may well have had their first encounter with a computer when they first met the trusty Beeb.

As with Acorn, so can the founders of BEEBUG, Sheridan Williams and Lee Calcraft, hardly have envisaged the way in which both the magazine and the whole BEEBUG organisation would grow and develop over the years. It would be interesting to know too just how many subscribers from issue one are still reading the magazine today (or its sister publication for the Archimedes, RISC User).

It is interesting to look back at the very first issue of the magazine published in April 1982, the first magazine of any kind devoted to the then new BBC micro. That issue amounted to 28 pages including covers, and was called the BEEBUG Newsletter. The word 'magazine' did not appear until March 1983.

There was a review of the machine (originally in two versions, model A and model B), comments on the late delivery of machines, a 3D surface program comprising all of 23 lines, an article on the then undocumented (remember the early, grey provisional User Guide?) SOUND and ENVELOPE commands, programs for drawing an ellipse (35 lines), Moon Lander game (nearly 100 lines) and 3D Noughts and Crosses (over 250 lines). There was also information on FX calls (did you know that 'FX' was Acorn jargon for 'effects'?), how to upgrade a model A to a model B, and pages of hints and tips, which have always been one of the most popular features of the magazine. The magazine was reproduced directly from daisywheel (text) and dot-matrix (programs) printout. There were no illustrations and no use of colour.

Comparing that issue with the BEEBUG of today reveals a wealth of changes. The contents have grown to a regular 64 pages, and there are good quality illustrations throughout the magazine which is printed in two colours (black and one other) from laser printed originals. There are more and longer articles and programs, and the contents reflect the much more sophisticated and computer literate user base which is the BBC micro market of today.

As we enter the 1990s, we are left to ponder how the BBC micro will fare in the future. BEEBUG will, of course, continue to support the BBC micro and Master series (and there is still a thriving secondhand market with a dozen or so trade-ins and re-sales per week). At the same time, other magazines in the Acorn market are concentrating increasingly on the Archimedes. Of course, there is not the number of new products for the BBC micro these days, more's the pity, and the number of reviews in the magazine has decreased accordingly. On the other hand, if our reader survey is anything to go by, there is increasing emphasis on the use of the micro for word processing, spreadsheet and database usage, rather than for programming.

Both the BBC micro and BEEBUG have a valuable role to play in the future. We hope that you will stay with us for as long as you have a BBC micro, and as Robin Burton says in this month's 512 Forum, if you do eventually sell your system, do tell the purchaser about BEEBUG and the support that we provide. Far from fading away, we expect BEEBUG to become even more valuable as support from other magazines diminishes.

Note: Only a few copies of issues 6, 7 and 9 remain from volume one. All copies of the first issue have now been sold.

News News News News News News

MINI OFFICE BREAKS RECORDS

Database Software has revealed that its top selling Mini Office package has now passed the half million mark in total sales. The lucky purchaser of the 500,000th copy won a state-of-the-art video recorder. Mini Office became the first business software to enter the best seller charts, normally the exclusive domain of games. Since its debut on the BBC micro, Mini Office has also been released for the Amstrad PCW and for the IBM PC and compatibles.

The current version for the BBC micro is Mini Office II, costing from £16.95 (5.25" disc) through to £21.95 (3.5" disc for the Master Compact), both prices inc. VAT. Database Software is at Europa House, Adlington Park, Adlington, Macclesfield SK10 4NP, tel. (0625) 878888.

KEEPING WARM

D.J.Holden has made available a suite of programs to assist in calculating the heat requirements of a building. If you are planning to install or upgrade your central heating systems then this software may help. Using the package enables the number and sizes of radiators to be assessed. The software is available in two forms. Firstly there is a demonstration disc, which is fully functional except that the programs do not have the facility to select radiators, and work only with a limited number of rooms. This version is claimed to be quite adequate for DIY use, and full instructions are contained within the programs themselves. The full version for central heating engineers costs just £15 inclusive, and is claimed to be better than equivalent PC programs costing over £200.

The software is available for the model B and Master on 5.25" discs, and for the Archimedes on 3.5" discs from D.J.Holden, 39 Knighton Park Road, Sydenham, London SE26 5RN, tel. 01-778 2659.

DIAL UP COMMUNITEL

The University of Nottingham has announced two software products aimed principally at schools and other educational users. *DialUp* is supplied on two discs and is intended to make the use of the Communitel Viewdata Terminal package much simpler and easier to use for school librarians and teachers. When accessing a viewdata system or a bulletin board with this software there is also a facility to monitor who has used the system, when and for how long.

The second product is *Themewise*, which provides a simple yet effective way of processing text files to locate, retrieve and collate either paragraphs containing particular words or identifiable sections of text. Once located the sections of text are combined into a new file which can be loaded, edited and printed as normal. As such the software forms a useful adjunct to any word processor or text editor.

DialUp costs £8 while *Themewise* costs just £7 (specify 40 or 80 track disc), both fully inclusive prices. Site licences are also available. Contact Dr.D.Lawrence, Department of Sociology, University of Nottingham, University Park, Nottingham NG7 2RD or phone (0602) 484848 ext.3080.

LAST DAYS OF DOOM

Fans of Peter Killworth's *Doom* adventure games will be interested to learn that Topologika will shortly release the third and final part of this trilogy. Entitled *Last Days of Doom*, this adventure sells at £19.95 in all versions (BBC micro and Archimedes). Orders placed before 31st January 1990 will qualify for a special introductory price of just £14.95 (both prices include VAT and p&p). To place your order, or for more information, contact Topologika at P.O.Box 39, Stilton, Peterborough PE7 3RL, tel. (0733) 244682.

SHOWING INTEREST

Acorn-specific shows are not as popular with organisers as they once were, but Acorn products have increasingly featured at more general shows. Taking place soon is the Spring Computer Show '90 at Olympia, London 4th to 6th May (tel. 061-998 0643) specifically for the home and educational user. Before that the European Computer Trade Show will be at the Business Design Centre, London 1st to 3rd April (tel. (0625) 879965) following the inaugural show of this name in 1989.

Later in the year, there is The Education Management Exhibition at Wembley, London 2nd to 4th October (tel. (0984) 23053), the Design and Technology Education Show at the NEC, Birmingham 25th to 27th October (tel. (0425) 272711), and a repeat of the successful Computer Shopper Show 7th to 9th December (tel. (0625) 879965), this time at the New Hall, Wembley.

Details of shows at which BEEBUG will be exhibiting will be given in future issues.

TINT PANELS

Dorian Goring shows how to add more style to your printouts using tint panels.

INTRODUCTION TO THE PRINCIPLES

Page layout and design is the complex interactive process of combining pictures, body-text, headlines, working white, and graphics into a pleasing and harmonious composition.

Desktop Publishing (DTP) helps orchestrate this composition of tones and shapes into a balanced, lively arrangement which attracts readers and keeps them reading longer (see *Instant Publishing* by the author in BEEBUG Vol.8 No.6).

Pick up any magazine and it is the use of colour, amongst other things, which attracts your attention. By *colour,* I mean to include tones or shades of grey, as well as hues.

One way to make a DTP page special is to use tint panels - flat areas made up of tiny dots of tone or hue. Used creatively, they also provide an effective visual means of knitting text and graphics together. Across the page, they signal a priority of importance.

Looking at a page through half-shut eyes helps us *see* its design. We're looking at an equilibrium, a balance, between simple shapes - squares, circles, triangles - and their relative visual tones or *weights.*

These juxtapositions set up contrasts, create movement and rhythm. Squares and boxes signal balance and order, of symmetry and enclosure. The space defined is without tension. It is static, inert, and neutral. Circles generate a sense of movement and direction both imploding and exploding - we feel compelled towards the centre. Triangles force our attention to the apex.

But, how can you create tones and hues (colours) on a standard mono dot-matrix printer? Answer - use those old ribbons you still kept (but don't know why)! Using old ribbons avoids tint panel clashes with the typeface. To avoid the expense of a colour printer, simply use coloured ribbons (available for most dot-matrix printers) - see *Multi-Coloured Printing* in BEEBUG Vol.8 No.4.

Colour tints mean creating masks for each primary - yellow, magenta, cyan, and black - by making identical copies of your finished page, and using each of these for a particular hue.

Some DTP environments automatically produce colour separations. I do this editing 'freehand' because you then have greater control.

One sheet of paper makes several passes through the printer, each time with a different page, or mask as it's known in the trade, and a well-worn ribbon. Registration marks are essential for accurate printing, and using lighter colours first then darker ones, minimises "muddying" ribbons.

This approach links Art and Design because tint panels are one way of exposing design. On their own, they are like abstract paintings. Here, pixels become dots of ribbon ink almost *Impressionistically,* matching Seuret's pointillist method of painting in the late nineteenth century.

Finally, a word of caution. Tints must be used thoughtfully because they can create additional problems.

The main drawback is that strong or dark tints weaken the impact of text and break down the resolution of a typeface, or mask it completely, especially when printing colour on colour. Fine dots, light grey, or pale hues are best but it depends very much on the font (typeface design and size) you're using.

Basically, the typeface must be strong enough to take the tint and the best way of finding out is to experiment! Be careful, though, because boxed tints with heavy rules have the effect of emphasising, or even over-emphasising, a panel - make sure the message is this important!

Text with tints needs borders set slightly wider, but make sure narrow columns don't create very ragged text. If they do, try irregular shaped tints which follow ragged ends.

TINT PANELS IN PRACTICE

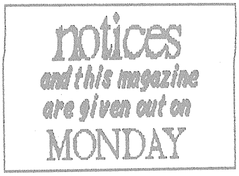
We will now look at some particular examples using tint panels. The work in each case was accomplished using *Stop Press* and *Pixel Perfect* DTP software.

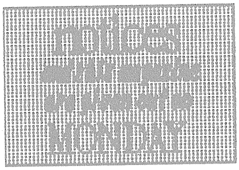
EXAMPLE 1 - creating a tint box for words.

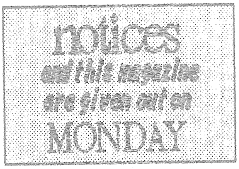
Strong tints (b) can weaken or dissolve text (a) completely, but using well-worn ribbons solves the problem (c, d).

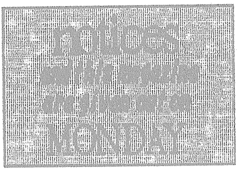
METHOD

- Begin by writing the words in a box in the appropriate place on your page. Save the completed page or screen.
- Next, make a copy of the original page onto another page - this page will hold the various tint panels.
- Load the tint page and locate the box. Delete the words and fill with black or suitable pattern - note: some patterns work better than others. Experiment!
- Now, load the printer with your most worn ribbon (black or coloured), make registration marks on the paper and print.
- Finally, replace your 'good' ribbon and overprint with the page of copy.


a

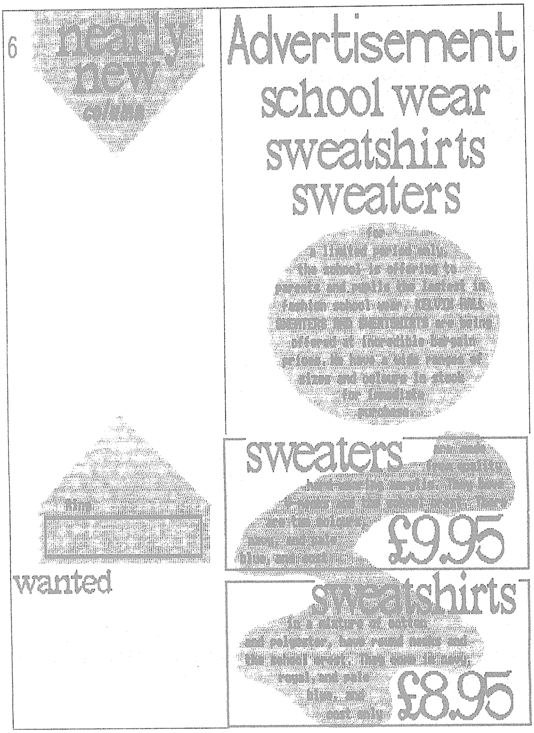

b


c


d

EXAMPLE 2 - page from a school magazine.

Harmonious page composition means balancing tones and shapes. These juxtapositions set up contrasts, create movement and rhythm. Once your page design is complete, duplicate it and use this to identify and construct a tint template for overprinting with your finished page.



My finished page was in two main parts - *nearly new & wanted*, and *advertisement*. Taking each part in turn...

nearly new - triangular tint, top and base of box (e, left).
The header words were centre justified to form a triangle which points its apex towards the text.

METHOD

- First, mark out the top triangle. Estimate an average line end, and use this to mark the triangle's base.
- Next, construct the apex taking account of words crossing horizontally. Paint or fill with black, or pattern, and airbrush out any blemishes.
- Finally, copy the panel to the base of the box and flip top to bottom.

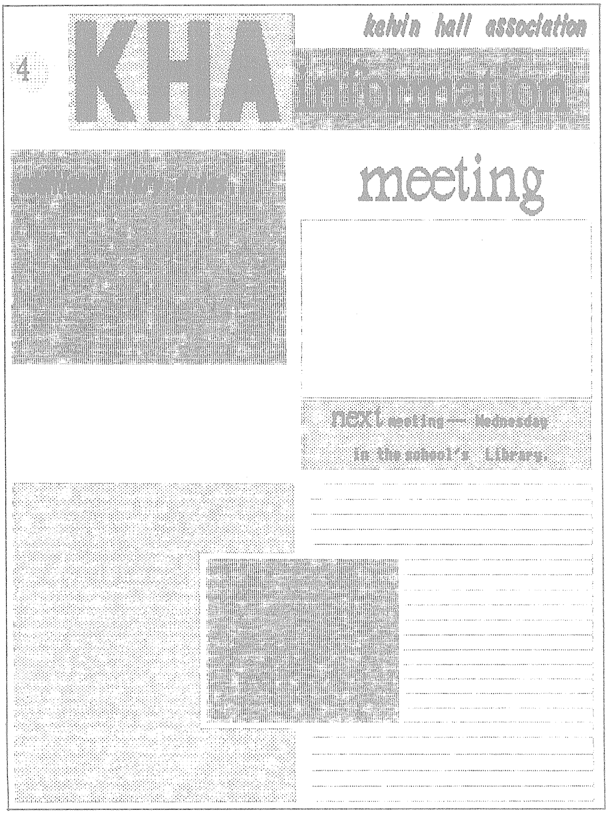
You're not forced to use black only. By reversing out, re-painting with pattern, and reversing out again, it is possible to create different patterned tints without black ruled edges.

advertisement - ghosted lettering, circle and ragged tints (e, right, together with f, an additional intermediate graphic showing the process of constructing the tints).

METHOD

- Firstly, mark out the circle. Paint black and airbrush blemishes. Reverse out, fill with new pattern (taking account that the pattern reverses too), and reverse out again.
- Next, airbrush black round text edges (bottom half) allowing a margin of overlap, paint, and follow the reversed out procedure.
- Finally, move the word 'advertisement' lower and right slightly to create a shadow effect, airbrush white to erase large text and all rules, and save.

When you're ready to print, remember, load the printer with your most worn ribbon first, make registration marks, then dump. Replace your 'good' ribbon and overprint with the final page.



EXAMPLE 3 - style sheet showing tint panels in place (g).

Boxes and tint panels are good for grouping and separating text, holding-up or strengthening the page, or indicating different kinds of information. This style sheet was designed for a local school magazine.

Dichotomous Keys - Part 1

Rupert Thompson describes a useful technique for building a simple expert system which will identify any item by means of a branching structure.

Imagine that you are an expert in some particular field, botany for example, and that you want to enable other people to identify plant species with the same knowledge that you yourself have. One way in which you could do this would be to write down a series of questions, which expect the answer 'Yes' or 'No'. The user answers these questions, and depending upon his answer is directed to another question until the species (or whatever) has been identified. Such an arrangement is called a *dichotomous key*, and is a very powerful tool for identification of almost anything, since its use is so very simple. So useful, in fact, are dichotomous keys that they form part of the GCSE Biology syllabus, but they can be applied to virtually any field of knowledge. Consider the following example:

1. Does the shape have four or more sides?
   YES - Question 2 NO - Question 5

2. Does it have over four sides?
   YES - a HEXAGON NO - Question 3

3. Does it have only right-angles?
   YES - Question 4 NO - a TRAPEZIUM

4. Are all its sides the same length?
   YES - a SQUARE NO - a RECTANGLE

5. Are two or more sides the same length?
   YES - Question 6 NO - a SCALENE TRIANGLE

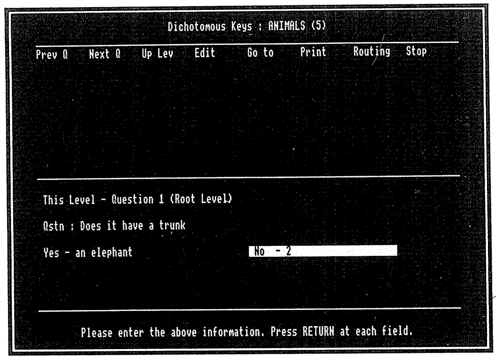
6. Are all 3 sides the same length?
   YES - an EQUILATERAL TRIANGLE NO - an ISOSCELES TRIANGLE

This simple key will identify seven geometrical shapes: Hexagon, Trapezium, Square, Rectangle, Scalene, Isosceles and Equilateral. Try it out for yourself.

The program listed here provides an easy way to create and use dichotomous keys of this type. Once the key has been set up it may be used by anyone, regardless of their expertise with computers. Part 1 of the program is published this month; this allows you to create and run simple keys. Part 2 next month will provide more complex editing techniques, such as chaining and redirection of questions.

USING THE PROGRAM

Type in this month's listing, being very careful to keep to the line numbers given, so that next month's listing will dovetail with it. Then save the program and run it.



```
                 Dichotomous Keys : ANIMALS (5)

This Level - Question 1 (Root Level)

Qstn : Does it have a trunk

Yes - an elephant                     No  - 2

      Please enter the above information. Press RETURN at each field.
```

Editing a dichotomous key

You will first of all be presented with a menu containing four options: Run Key, Edit Key, OS Command and Exit. To move around the menu, use the up and down cursor keys, and to select an item press the space bar when the cursor is on that line. The option OS Command lets you type in star commands (but beware of *COMPACT, *COPY and *BACKUP - they corrupt the RAM and stop the program from working), whilst Exit leaves the program.

To start with, select the Edit Key option. You will then be asked for a filename for the key. If this file does not exist you will be offered the option of creating it, so do this now. The program then asks for the number of items for identification and calculates the number of questions to be asked. A dichotomous key to identify n objects requires n-1 questions, but the program does in fact create one spare question, the reason for which will become apparent in Part 2.

Once your key has been created you are in Edit mode. A horizontal menu gives access to the editing functions, some of which will not be usable until you have Part 2.

You can move between the questions in the key in two ways. Prev Q and Next Q move backwards and forwards one question at a time. Larger jumps to any question in the file can be made with Go To. A further facility will be added next month.

Select Edit to define a new question, or edit an existing one. Three pieces of information (or fields) are required:

(i) A QUESTION for the computer to ask when the key is run;

(ii) A YES POINTER which tells the computer what to do if the question is answered 'Yes';

(iii) A NO POINTER, similar to the Yes Pointer, but for the answer 'No'.

The two pointers may be either numbers or words. If they are numbers, the computer will subsequently ask the question whose number is given, otherwise it will treat the word as the name of the object (or whatever) being identified. Consider this question definition:

```
Does the animal's name begin with the letter 'A'?

YES - an AARDVARK     NO - 2
```

In this case, answering 'No' will make the computer go on to ask Question 2, whilst 'Yes' will provoke this response:

```
It is an AARDVARK
```

Note that the words 'It is' are printed automatically. The pointers should contain only the actual name and any article with which you wish to introduce it, such as 'an', 'a' or 'the'.

When all three fields have been entered, you will be asked whether to save the question to disc.

When you have finished defining the key, select Stop to return to the Options Menu. Now select Run Key to use your new key. Simply answer each question by pressing Y for YES or N for NO. When your object is finally identified you will be informed of this and told what the object is.

Finally this month, here are a few notes about making up keys.

(i) Each question should expect the answer 'Yes' or 'No' only.

(ii) Each question should be framed in such a way that one or more possibilities are discarded. Remember that you only have a finite number of questions to ask, so every question must help in the identification process.

(iii) Where a question points to a further question, the latter should be a new question. It is neither possible nor desirable for any question to be referred to by more than one previous question.

(iv) It is possible for a question to refer backwards in the file, although on stylistic grounds it should be avoided. In other words, question 3 should not cause question 2 to be asked in response to an answer.

You might like to computerise the key to shapes given above.

Part 2 will cover additions to the program to implement the chaining of questions, restructuring and a hard-copy facility.

```
   10 REM Program Dichl
   20 REM Version B1.0
   30 REM Author  Rupert Thompson
   40 REM BEEBUG  Jan/Feb 1990
   50 REM Program subject to copyright
   60 :
  100 MODE3:ON ERROR GOTO150
  110 PROCvar:PROCsetup:exit%=FALSE
  120 REPEAT:PROCmenu:UNTIL exit%
  130 MODE7:CLOSE#0:END
  140 :
  150 MODE7:REPORT:PRINT" at line ";ERL
  160 END
  170 :
 1000 DEF PROCsetup
 1010 PROCtitle("Dichotomous Keys",0)
 1020 PRINTTAB(0,1);STRING$(80,"_")
 1030 PRINTTAB(0,21);STRING$(80,"_")
 1040 ENDPROC
 1045 :
 1050 DEF PROCvar:DIMmenu$(7)
 1070 open%=0:menu%=-1:ENDPROC
 1110 DEF PROCwindow(w,c)
 1120 IF w=1 THEN VDU28,0,0,79,0
 1130 IF w=2 THEN VDU28,0,2,79,2
 1140 IF w=3 THEN VDU28,0,20,79,3
 1150 IF w=6 THEN VDU28,0,24,79,23
 1160 IF c THEN CLS
 1170 ENDPROC
 1175 :
 1180 DEF FNsure(m$):PROCwindow(6,-1):PR
OCtitle(m$+" Are you sure? (Y/N)",0):VDU
7:G=FNgetcha("YyNn"):CLS:=INSTR("Yy",CHR
$G)>0
 1190 :
 1310 DEF PROCtitle(m$,n)
 1320 PRINTTAB((80-LENm$)/2,n);m$;:ENDPR
OC
 1325 :
 1330 DEF PROCinv(n):IF n THEN COLOUR129
:COLOUR0:ELSE COLOUR1:COLOUR128
 1340 ENDPROC
 1345 :
 1350 DEF FNbar(x,y,n,m$,f,l):LOCAL I,t,
c$
 1360 *FX4,1
 1370 PROCextract(m$,n):IF f THEN PROCli
st
 1390 Z=0:nv=0:REPEAT:COLOUR129:COLOUR0:
PRINTTAB(x,y+Z);" ";menu$(Z);SPC(l-LENme
nu$(Z));
 1400 REPEAT:G=GET:IF G=&8B THEN nv=Z-1
 1420 IF G=&8A THEN nv=Z+1
 1430 IF nv<0 THEN nv=0 ELSE IF nv>n THE
N nv=n
 1440 UNTIL G=&8A OR G=&8B OR G=32
 1450 COLOUR128:COLOUR1:PRINTTAB(x,y+Z);
" ";menu$(Z);SPC(l-LENmenu$(Z)):Z=nv:UNT
IL G=32
 1460 *FX4
 1470 =Z
 1475 :
 1480 DEF PROClist:COLOUR1:COLOUR128
 1490 LOCAL I:FOR I=0 TO n:PRINTTAB(x+1,
y+I);menu$(I):NEXT I:ENDPROC
 1500 :
 1510 DEF FNgetcha(v$):REPEAT:G=GET:UNTI
L INSTR(v$,CHR$G)>0 OR v$="" OR INSTR(c$
,CHR$G)>0:=G
 1515 :
 1520 DEF FNenter(x,y,l,o$,v$,p$)
 1530 LOCAL i$:i$=o$
 1535 c$=CHR$13+CHR$127+CHR$9+CHR$135
 1540 *FX4,1
 1550 COLOUR0:COLOUR129:PRINTTAB(x,y);"
";p$;SPC(l+2)
 1560 x=x+LENp$+2:PRINTTAB(x,y);o$;
 1570 REPEAT:G=FNgetcha(v$)
 1580 IF G=127 AND LENo$>0 THEN o$=LEFT$
(o$,LENo$-1):PRINTTAB(x+LENo$+1,y);CHR$1
27;
 1590 IF G>31 AND G<127 AND LENo$<l THEN
 o$=o$+CHR$G:PRINTCHR$G;
 1600 IF G=&87 THEN o$="":PRINTTAB(x,y);
SPC(l);TAB(x,y);
 1610 IF G=9 AND o$="" THEN o$=i$:PRINTT
AB(x,y);o$;
 1620 UNTIL G=13 OR (G>&87 AND G<&8C)
 1630 flag=1:IF G=13 OR G=&8A THEN flag=
-1 ELSE IF G=&89 THEN flag=-100
 1640 *FX4
 1650 COLOUR1:COLOUR128:PRINTTAB(x-LENp$
-2,y);" ";p$;" ";o$;SPC(l+1-LENo$)
 1660 =o$
 1665 :
 1670 DEF PROCextract(m$,n)
 1680 LOCAL t,I:t=0:FOR I=0TO n:menu$(I)
="":REPEAT:t=t+1:c$=MID$(m$,t,1):menu$(I
)=menu$(I)+c$:UNTIL c$="*":menu$(I)=LEFT
$(menu$(I),LENmenu$(I)-1):NEXT I:ENDPROC
 1685 :
 1690 DEF FNlotus(m$,n,e,W,Z):LOCAL I
 1710 *FX229,255
 1720 PROCwindow(2,W):*FX4,1
 1740 PROCextract(m$,n):IF W THEN FOR I=
0 TO n:PRINTTAB(I*10,0);menu$(I);:NEXT I
 1760 REPEAT:REPEAT:COLOUR0:COLOUR129:PR
INTTAB(Z*10,0);menu$(Z);SPC(9-LENmenu$(Z
));:G=GET:UNTIL G=&88 OR G=&89 OR G=32 O
R G=27
 1770 COLOUR1:COLOUR128:PRINTTAB(Z*10,0)
;menu$(Z);SPC(9-LENmenu$(Z));
 1780 IF G=&88 AND Z>0 THEN Z=Z-1 ELSE I
F G=&89 AND Z<n THEN Z=Z+1
 1790 UNTIL G=32 OR (G=27 AND e)
 1800 *FX4
 1810 *FX229
 1820 IF G=32 THEN =Z ELSE =-1
 1830 :
 1910 DEF PROCmenu
 1920 LOCAL Z:PROCwindow(3,menu%)
 1940 IF menu% THEN PROCtitle("OPTIONS M
ENU",5)
 1950 Z=FNbar(34,7,3,"Run Key*Edit Key*O
S Command*Exit*",menu%,11):menu%=0
 1960 IF Z=0 THEN PROCrunkey
 1970 IF Z=1 THEN PROCeditkey
 1980 IF Z=2 THEN PROCstar
 1990 IF Z=3 THEN exit%=FNsure("Exit pro
gram.")
 2000 ENDPROC
 2005 :
 2010 DEF FNfilename(c)
 2020 LOCAL F:CLOSE#0:open%=0:PROChead
 2040 PROCwindow(3,-1)
 2050 file$=FNenter(30,8,7,file$,"","Nam
e of key:")
 2060 F=OPENUPfile$:IF F>0 THEN CLS:open
%=-1:PTR#F=0:INPUT#F,ite%:PROChead:=F
 2070 IF NOT c THEN PROCtitle("File does
 not exist. Press any key to continue",9
):VDU7:G=GET:CLS:=0
 2080 PROCtitle("File does not exist. Do
 you wish to create it? (Y/N)",9)
 2090 IF NOT FNyn THEN CLS:=0
 2100 PROCcreate(file$):CLS:PROChead
 2110 =F
 2115 :
 2120 DEF FNyn:=INSTR("Yy",CHR$FNgetcha(
"YyNn"))>0
 2122 :
 2125 DEF PROChead:PROCwindow(1,-1):IF o
pen% THEN PROCtitle("Dichotomous Keys :
"+file$+" ("+STR$ite%+")",0) ELSE PROCti
tle("Dichotomous Keys",0)
 2126 ENDPROC
 2128 :
 2130 DEF PROCeditkey:menu%=-1
 2140 F=FNfilename(-1):IF F=0 THEN ENDPR
OC
 2150 LOCAL qstn,I,Z,lotus%
 2160 qstn=1:lotus%=-1:ptr=0
 2170 REPEAT:PROCedrd:Z=FNlotus("Prev Q*
Next Q*Up Lev*Edit*Go to*Print*Routing*S
top*",7,0,lotus%,Z):lotus%=0
 2180 IF Z=0 AND qstn>1 THEN qstn=qstn-1
 2190 IF Z=1 AND qstn<ite% THEN qstn=qst
n+1
 2210 IF Z=3 THEN PROCedit
 2220 IF Z=4 THEN qstn=FNgo
 2260 UNTIL Z=7:CLS:ENDPROC
 2270 :
 2410 DEF PROCcreate(o$):CLS
 2430 PROCtitle("Create Key : "+file$,7)
 2440 REPEAT:ite%=VALFNenter(20,9,3,"","
0123456789","How many objects to identif
y? "):UNTIL ite%>0
 2450 OSCLI("SAVE "+file$+" 0 +"+STR$~((
126*ite%)+10))
 2460 PROCinitfile:CLS:ENDPROC
 2470 :
 2480 DEF PROCinitfile:F=OPENUPfile$
 2490 open%=-1:PROChead:PROCwindow(3,-1)
 2500 q$="":yes$="":no$="":ptr=0
 2510 PROCtitle("Initialising File - Ple
ase Wait",5)
 2520 PRINTTAB(27,7);"Initialising Quest
ion : "
 2530 FOR I=1 TO ite%:PTR#F=(126*(I-1))+
10
 2550 PRINTTAB(51,7);I
 2560 PRINT#F,ptr,q$,yes$,no$:NEXT I
 2575 PTR#F=0:PRINT#F,ite%:ENDPROC
 2580 :
 2590 DEF PROCreadrec(I)
 2600 PTR#F=((I-1)*126)+10
 2610 INPUT#F,ptr,q$,yes$,no$:ENDPROC
 2620 :
 2630 DEF PROCwriterec(I)
 2640 PTR#F=((I-1)*126)+10
 2650 PRINT#F,ptr,q$,yes$,no$:ENDPROC
 2660 :
 2670 DEF PROCedrd:PROCwindow(3,-1)
 2690 PROCreadrec(qstn):p=ptr:IF p>0 THE
N PROCreadrec(p):PRINTTAB(1,1);"One Leve
l Up - Question ";p;:PROCprintrec(3)
 2700 PRINTTAB(0,8);STRING$(80,"_")
 2710 PROCreadrec(qstn):PRINTTAB(1,10);"
This Level - Question ";qstn;:PROCprintr
ec(12)
 2720 ENDPROC
 2725 :
 2730 DEF PROCprintrec(n)
 2740 IF ptr=0 THEN PRINT" (Root Level)"
 ELSE PRINT" (Up to Question ";ptr;")"
```

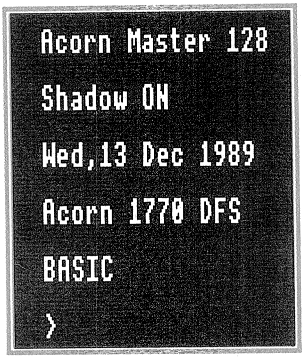
Continued on page 28

Master Display ROM (Part 1)

Derek Gibbons describes a utility which allows you to configure your Master 128 to use Shadow RAM at all times regardless of mode, and display a new start-up message.

One of the most serious limitations of the Model B was the large amount of memory used by the screen display in modes 0-6. The Master 128 partly overcame this by including a separate bank of RAM which could be used for the screen, the so-called shadow RAM. However, some early reviews of the Master were highly critical of the fact that it is not possible to configure the standard machine to power-up with shadow permanently on so that all subsequent mode changes still use shadow RAM.

The program listed here assembles a ROM image, referred to as the Master Display ROM, which overcomes this limitation, and carries out some other useful actions as well.



```
Acorn Master 128

Shadow ON

Wed,13 Dec 1989

Acorn 1770 DFS

BASIC

>
```

The new Master 128 start-up display

In the standard machine, shadow screen operation can be invoked in two different ways (ignoring all the various *FX options). It can be selected explicitly by *SHADOW, followed by a mode change (even re-selection of the current mode), after which, ANY mode will use shadow RAM for the display, setting HIMEM at &8000. It can also be selected implicitly simply by changing to mode n+128 (where n has the value 0 - 7); however, subsequent selection of mode 7 or less will then revert to non-shadow operation.

Secondly, with the configuration command, it is possible to define an implicit shadow mode for use at power-up or Ctrl-Break, by issuing:

```
*CONFIGURE MODE n
```

where 'n' is in the range 128-135, but once again the shadow operation is lost if a non-shadow mode is subsequently selected.

As explained above, The Master Display ROM overcomes this problem, by detecting hard Breaks, finding the configured mode from the CMOS RAM, and then effectively issuing the appropriate *FX114 call. This call is the exact equivalent of the *SHADOW command; *FX114,0 selects shadow operation regardless of mode, while *FX114,1 selects main or shadow memory depending on the mode. Although a subsequent mode change is then required, this is automatically provided by the MOS as part of the hard Break reset sequence. The ROM also incorporates the message 'Shadow ON' or 'Shadow OFF', as appropriate, in the display at the top of the screen, but this action does not occur on a simple Break as such a Break is not sufficient to destroy the shadow setting.

As an extra feature, the ROM adds a further line to the hard Break message, to show the current date from the Master's real time clock, and modifies all Break messages to display 'Acorn Master 128' instead of just 'Acorn MOS' (if you've got it, why not flaunt it!). An extended version of the ROM, to be described in Part 2, allows the foreground and background start-up colours to be configured for modes other than 7 and 135. Even the extended ROM image is less than &400 bytes long and provides ample opportunity for the inclusion of additional routines.

Either version of the ROM can be blown into an EPROM and installed in any cartridge socket (0 - 3), or in internal socket 8. The ROM image can also be loaded directly into, and operated from, sideways RAM, but this limits its real value as it is intended to function at power-up and not just on hard or soft Breaks. However, this is useful to test that everything is working correctly if you have typed the program in from the magazine listing. Once the ROM has been installed, no user interaction other than *HELP is involved. The ROM then displays its title string in the normal way, and *HELP MDR displays brief information on the ROM's functions. Note that MDR must be used in full; it is sufficiently short not to be worth considering the use of an abbreviation.

Listing 1 is the source code for the initial version of the ROM. It should be entered exactly as printed and should NOT be renumbered in case it is decided to extend the ROM image as described later. When run in mode 7, there is plenty of room to assemble the ROM image at &5000 as the program has been kept compact by omitting the majority of unnecessary spaces (but note that the space in line 1100 is essential). For the same reason, there are virtually no REM comments, but for those who may be interested, a description of the action of the various stages follows.

PROGRAM NOTES

Line 1240 checks for service call 39 (reset); lines 1280-1290 read the screen cursor position with OSBYTE 134; lines 1320-1330 change the 'Acorn MOS' message to 'Acorn Master 128' by overwriting, starting with the 'M'; and lines 1340-1360 restore the cursor with VDU31.

Lines 1370-1380 use OSBYTE 253 to establish the type of Break. For a hard Break (power-up or Ctrl-Break), lines 1410-1420 display 'Shadow O'; lines 1430-1450 read byte 10 of the CMOS RAM, and extract and check the mode number and shadow flag; lines 1450-1480 add 'N' or 'FF' to the display message as appropriate; and line 1500 uses OSBYTE 114 to set, or reset, the permanent use of shadow RAM. Line 1530 *then reads the CMOS clock RAM using* OSWORD 14 and lines 1560-1580 extract and display the date part of the time string as 'Ddd,nn Mmm yyyy'. This could be shortened to cut out the day of the week, or lengthened to include the time of day, according to personal preference.

If line 1240 did not detect a reset, line 1660 checks for service call 9 (HELP), and lines 1670-1680 and 1760-1800 check the rest of the command line. If there is no more text on the command line, 'MDR' is printed below the ROM title string (which the MOS will have already printed), but if 'MDR' is itself detected, lines 1930-1970 display the extra help information.

Note that extensive use is made of memory locations &90-&92 in page zero. These are part of the Econet work space, but can normally be used safely if no Econet upgrade is fitted. When the CMOS clock time string is read and stored starting at &90, memory usage spills over into the NMI work space at &A0, but no problems have been experienced in this respect so far, presumably because the string is only stored there on a hard break. If either of these areas is needed for other purposes, the normal user work space at &70-&8F could be used by changing all instances of &9n to &7n, but otherwise it is preferable to keep this area free as users' own programs may require it.

Listing 1

```
  10 REM Program MDR1_S
  20 REM Version B1.1
  30 REM Author  D. Gibbons
  40 REM BEEBUG  Jan/Feb 1990
  50 REM Program subject to copyright
  60 :
 100 MODE 7
 110 PROC assemble
 120 *SAVE MDR1 5000+400 8000 8000
 130 END
 140 :
1000 DEFPROCassemble
1010 osbyte=&FFF4:osasci=&FFE3
1020 oswrch=&FFEE:osword=&FFF1
1030 osnewl=&FFE7:comline=&F2
1040 romnum=&F4
1050 FORPASS=4TO7STEP3
1060 P%=&8000:O%=&5000
1070 [
1080 OPT PASS
1090 EQUB0:EQUW0:JMPservice
1100 EQUB&82:EQUBoffset MOD256
```

```
1110 EQUB1
1120 .title
1130 EQUS"Master Display ROM"
1140 EQUB0
1150 .version
1160 EQUS" v1.1"
1170 EQUB0
1180 .offset
1190 EQUB0
1200 EQUS"(C) BEEBUG"
1210 EQUB0
1220 .service
1230 PHP:PHA:PHX:PHY
1240 CMP#&27:BEQbreak:JMPtryhelp
1270 .break
1280 LDA#&86:JSRosbyte
1290 STX&90:STY&91
1300 LDX#0
1310 .onscrn
1320 LDAsize,X:JSRosasci
1330 INX:CPX#13:BCConscrn
1340 LDA#&1F:JSRoswrch
1350 LDA&90:JSRoswrch
1360 LDA&91:JSRoswrch
1370 LDX#0:LDY#&FF:LDA#&FD:JSRosbyte
1380 TXA:BEQnothard
1390 LDX#0
1400 .pshadow
1410 LDAtshadow,X:JSRosasci
1420 INX:CPX#8:BCCpshadow
1430 LDX#10:LDY#0:LDA#161:JSRosbyte
1440 LDX#0:TYA:AND#8:BNEshadow
1450 LDA#ASC("F"):JSRosasci:JSRosasci
1460 LDX#1:JMPshadoff
1470 .shadow
1480 LDA#ASC("N"):JSRosasci
1490 .shadoff
1500 LDY#0:LDA#114:JSRosbyte
1510 LDA#13:JSRosasci:JSRosasci
1520 LDA#0:STA&90
1530 LDA#14:LDX#&90:LDY#0:JSRosword
1540 LDY#0
1550 .date
1560 LDA&90,Y:JSRosasci
1570 INY:CPY#15:BNEdate
1580 JSRosnewl:JSRosnewl
1590 .nothard
1600 JMPrestore
1610 .size
1620 EQUB&1F:EQUB&06:EQUB&01:EQUS"Maste
r 128":EQUB&0D
1630 .tshadow
1640 EQUS"Shadow O"
```

```
1650 .tryhelp
1660 CMP#9:BNEnothelp:JSRhelp
1670 LDA(comline),Y
1680 CMP#13:BNEcheck
1690 LDX#255
1700 .details
1710 INX:LDAcommand,X:BEQdonecommand
1720 JSRosasci:BRAdetails
1730 .donecommand
1740 JSRosnewl:BRArestore
1750 .check
1760 LDX#1:DEY
1770 .again
1780 INX:INY:LDA(comline),Y
1790 AND#&DF:CMPcommand,X:BEQagain
1800 LDAcommand,X:BEQmine
1810 .restore
1820 PLY:PLX:PLA:PLP:RTS
1830 .nothelp
1840 JMPrestore
1850 .help
1860 JSRosnewl:LDX#&FF
1870 JSRhelploop:JSRhelploop
1880 JSRosnewl:RTS
1890 .helploop
1900 INX:LDAtitle,X:BNEhmore:RTS
1910 .hmore
1920 JSRosasci:BRAhelploop
1930 .mine
1940 LDX#255
1950 .more
1960 INX:LDAlist,X:BMIalldone
1970 JSRosasci:BRAmore
1980 .alldone
1990 PLY:PLX:PLA:PLP:LDA#0:RTS
2000 .command
2010 EQUS"  MDR"
2020 EQUB0
2030 .list
2040 EQUB13
2050 EQUS"  Ctrl-Break displays CMOS Da
te"
2060 EQUB13
2070 EQUS"  and implied state of Shadow
"
2080 EQUW&D0D
2110 EQUS"  Shadow mode via *CO. MODE n
"
2160 EQUW&D0D
2170 EQUB&FF
3340 ]:NEXT
3350 ENDPROC
3360 END
```

Exploring Communications

Alan Wrigley explains some of the attractions of using your micro to access comms services.

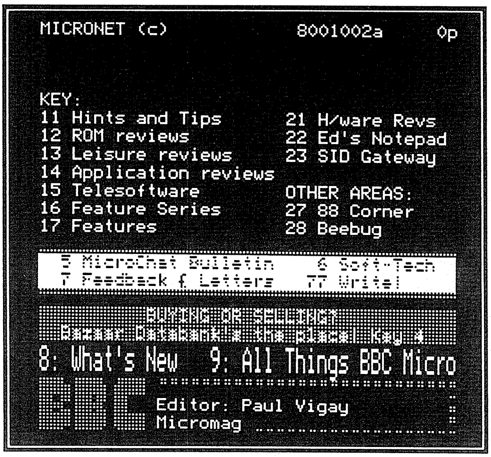
One fact which emerged strongly from our Readers' Survey last year was that a considerable number of BEEBUG readers would be quite happy to see the demise of The Comms Spot. The particular irony of this is that, in part at least, the BBC Micro owes its very existence to comms. The original specification laid down by the BBC for "their" micro was that it should be capable of downloading software from Ceefax as part of the Computer Literacy Project, hence the inclusion of a teletext emulation screen mode, i.e. mode 7.

In fact, the BBC still remains one of the best micros for comms use, and there are very many comms enthusiasts all over the country who would not swap it for anything else.

In my own, purely personal, view, it is a pity that the subject of comms attracts such antipathy, as I feel that it offers a window onto one of the most exciting aspects of current and future technology. While the "paperless office" may or may not ever be accepted to the degree that its supporters would like, it is a fact that telecommunications have already revolutionised many areas of modern life and will continue to do so at an ever-increasing rate.

Perhaps many of those BEEBUG readers who dismiss the subject are unaware of what it has to offer. First and foremost, accessing a comms service or a bulletin board with your computer provides instant communication with large numbers of like-minded people. It is like having the facilities of a club and a telephone hotline rolled into one, with the added advantage that you can access it at times which suit yourself. If you have a technical problem, you can air it instantly, and almost certainly someone who knows the answer will log on before long.

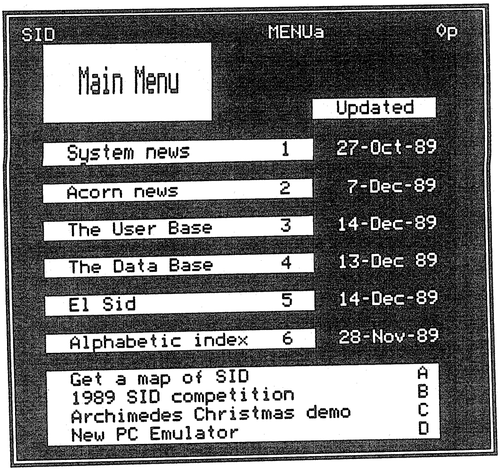
Magazines have fixed publishing deadlines and a finite amount of space per issue, which makes it difficult for them to react instantly to news or to include all the things they might like to publish. Bulletin boards, on the other hand, together with the various other micro-specific services such as Micronet (which is part of Prestel), and Acorn's own bulletin board SID (Support Information Database), have no such constraints, and are invariably run by knowledgeable enthusiasts who will often have access to information that is difficult for the average user to obtain. They also provide what, for some users, is the single most important aspect of comms - free downloadable software. For the cost of a 10-minute phone call, some of this software is a real bargain.



```
MICRONET (c)                 8001002a     0p

KEY:
11 Hints and Tips           21 H/ware Revs
12 ROM reviews              22 Ed's Notepad
13 Leisure reviews          23 SID Gateway
14 Application reviews
15 Telesoftware            OTHER AREAS:
16 Feature Series           27 88 Corner
17 Features                 28 Beebug

  5 MicroChat Bulletin      6 Soft-Tech
  7 Feedback & Letters     77 Write!

             BUYING OR SELLING?
   Bazaar Databank's the place! Key 4
8: What's New   9: All Things BBC Micro
BBC
        Editor: Paul Vigay
        Micromag
```

The BBC area on Micronet

In addition to the purely computer-oriented functions, the larger systems such as Micronet provide a meeting-place for people of widely-differing backgrounds without having to leave the comfort of their own homes. It is a fallacy that comms is the preserve of the younger generation; people of all ages enjoy exchanging views on-line, and one of the features which attracted me to Micronet in the first place is that, on screen, all people are equal. Age, appearance, background are all irrelevant. Indeed, for some people, particularly those with a physical disability, comms provides them with their only chance to integrate themselves fully into an able-bodied society without the risk of feeling disadvantaged.



```
SID                         MENUa          0p

  Main Menu
                                  Updated

  System news           1       27-Oct-89
  Acorn news            2        7-Dec-89
  The User Base         3       14-Dec-89
  The Data Base         4       13-Dec 89
  El Sid                5       14-Dec-89
  Alphabetic index      6       28-Nov-89

  Get a map of SID                    A
  1989 SID competition                B
  Archimedes Christmas demo           C
  New PC Emulator                     D
```

A sample frame from SID

Multi-user adventures were looked at briefly in the last issue (A MUG's Game, Vol.8 No.7), and while they may not be everyone's cup of tea, it is also wrong to assume that they are not the place for adults. Meeting other adventurers from time to time, I am constantly amazed at how many players are in their thirties, forties and even older; people who, by the standards normally imposed by society, should "know better" than to be playing computer games. In the best games, there is a strong element of role-play. Business schools know the value of role-play in teaching management techniques and developing character, but so far the message seems to have been largely lost on the critics of computer games. Besides which, multi-user adventures are a lot of fun!

An annual subscription to Micronet currently costs around £92, including a free modem, with comms software available for a small extra charge. This also covers access to all the other public services on Prestel, i.e. those for which an extra subscription is not required. Non-subscribers can get a taste of what is on offer by logging on to the Prestel demonstration service on 0272 250000, then entering 4444444444 (ten 4's) as their ID number and 4444 as the password. This allows you to see some demonstration screens.

SID can also be accessed from Micronet for a small extra charge (during off-peak hours it is 1p per minute on top of Prestel charges), or alternatively you can dial SID in Cambridge direct, on 0223 243642.

Bulletin boards have been covered from time to time in the Comms Spot, most recently in Vol.8 No.3. Lists of boards and their telephone numbers are sometimes published in other magazines, but there is no guarantee that they will still exist when you try to access them. The smaller ones come and go, and so definitive lists are hard to come by.

So why not join the future now, and explore the world of comms? You may be surprised at what you find.

Dual Column Listings Updated

by Paul Pibworth

The following changes allow this utility (see BEEBUG Vol.7 No.9) to print hash and pound signs correctly, assuming that these symbols are otherwise printed wrongly.

Split line 1980, add lines 1981 to 1983, 2041, and 2151 to 2159 as shown:

```
1980 LDA (prinbuf),Y
1981 CMP #35:BEQ hash:CMP #96:BNE not£
1982 LDA #35
1983 .not£ CMP #0:BNE char1
2041 .donehash
2151 .hash \ switch to USA, send #,
switch back to UK
2152 LDA #1:JSR &FFEE:LDA #27:JSR &FFEE
2153 LDA #1:JSR &FFEE:LDA #82:JSR &FFEE
2154 LDA #1:JSR &FFEE:LDA #0:JSR &FFEE
2155 LDA #35:JSR &FFEE
2156 LDA #1:JSR &FFEE:LDA #27:JSR &FFEE
2157 LDA #1:JSR &FFEE:LDA #82:JSR &FFEE
2158 LDA #1:JSR &FFEE:LDA #3:JSR &FFEE
2159 JMP donehash
```

A complete and revised copy of this program is included on this month's magazine disc.

EdiKit (Part 2)

Bill Hine presents the second part of the EdiKit ROM for the BBC B, B+ and Master.

EdiKit1's *FTEXT command described last month.

ENTERING THE PROGRAM

You should first load the ROM image Edirom1, assembled from the program given in part one of this series (BEEBUG Vol.8 No.7), into a sideways RAM slot using the appropriate command. Owners of a B+128 or Master, for example, would need to enter:

```
        *SRLOAD Edirom1 8000 X
```

where the 'X' is a vacant RAM slot. Alternatively, you could re-run EdiKit1 allowing the assembled ROM to be loaded from that program.

Now type in this month's program and save it as EdiKit2. There are no variations in the code between the Model B, B+ and Master, but you may need to substitute your own version of Acorn's *SRSAVE and *SRLOAD commands in lines 170 and 180. You should also replace 'X' in lines 170 and 180 with the slot number which you are currently using for your version of Edirom. Run the new program and answer 'Y' to the prompt:

```
        "SAVE and LOAD ROM?"
```

Part 2 of the ROM will be saved as Edirom2 and reloaded into SWR slot 'X' from &8C00 to &9240. The whole ROM (Edirom1 + Edirom2, &8000 to &9240) will then automatically be saved as Edirom.

Next you will need to initialise the ROM by pressing Ctrl-Break. Typing *HELP EdiKit should now print out a list of commands. You should find that *FTEXT, *FBASIC and *FPROCFN are working (don't forget to enter OLD before trying them, as you will have pressed Ctrl-Break), but calls to the remaining commands will produce the message "Not implemented".

USING THE NEW COMMANDS

The format of FBASIC is:

```
        *FBASIC <element>
or:     *FB.<element>
```

parameter; this is not necessary with the abbreviated form. Lower case letters (or any mix of cases) may also be used. The command will cause all lines of a Basic program containing the specified element to be listed out.

An element of Basic may be defined as:

1. The name of a variable, procedure or function (e.g. bptr, str$, P%, PROCA).
2. A Basic keyword (e.g. FOR, PRINT, CLS) as listed in the User Guide.
3. A numeric value (e.g. 397, &8000).
4. A string (e.g. "Press any Key").
5. An operating system or "star" command (e.g. *LOAD, *FX).
6. A line number following GOTO, GOSUB or RESTORE (e.g. GOTO 350).
7. An assembler command (e.g. LDA, JSR).
8. An assembler label beginning with a full stop (e.g. .findbasic).
9. The operand of an assembler command (or a significant part of it (e.g. #&100 or just &100, #ttable DIV256 or #ttable or ttable or DIV256).

Wildcards may be used in names, numeric values and assembler labels. Thus *fb.&???? will find all hex numeric values of four digits. *FB.&* will find all hex values in the program. When used on EdiKit2 itself, *FBASIC ?ptr will find bptr, pptr and tptr. *FB..* will list all the assembler labels. The '?' wildcard is used to stand in for any single letter. The '*' is used to mean "etcetera", and can only be used in the second or subsequent character position in a word. In the initial position, of course, it signals an OS command. No wildcards are allowed in strings, but they need not be complete. Thus *FB."E will find all lines containing a string beginning with E, while *FB." will list all the strings in the program. The initial quotes sign must be included as this tells EdiKit to search for a string.

Keywords may be abbreviated in the same way as when inputting a program. E.g. *FB.P. will

GOSUB, RESTORE) must be signalled to EdiKit by preceding them with the up-arrow character "^". To find a program line containing GOTO 1670, for example, you should type *FB.^1670.

Assembler instructions may be entered in upper or lower case, and occurrences of either will be found. Thus both *fb.iny and *FB.INY will find lines of assembler containing either INY or iny. EdiKit will also find assembler labels. Just enter the label name including the initial stop, e.g. *FBASIC .label or *fb..label. However, if you enter the label name without the stop, EdiKit will list out both the line containing the label itself AND all those with references to it.

*FBASIC differs from *FTEXT (from part one) in being much more selective. If you are not sure of a name but know roughly what you are looking for, *FBASIC with wildcards is the best bet; it will prevent a lot of irrelevant lines being printed out. If not, use *FTEXT with those characters you can remember. Note that, with the exception of assembler operands, *FBASIC can only find single elements. It cannot find Basic expressions (e.g. 100*newprice/oldprice) made up of elements linked by the mathematical or indirection operators (+ - * / ^ = < > ? ! $ etc.). It uses these symbols as delimiters marking the boundaries between elements. Nor does *FBASIC find the contents of REM statements or assembler comments as they are not a true part of a Basic program. In these cases, use *FTEXT.

The other new command, *FPROCFN, has no parameters and may be abbreviated to *FP. or *fp. It provides you with a list of all the procedures and functions in a program by listing the lines containing their definitions.

For convenience, the magazine disc/tape contains a ready assembled copy of Edirom (as well as this month's program). Next month in this series we shall add five more useful star commands to EdiKit.

```
   10 REM Program EDIKIT2
   20 REM Version 1.00
   30 REM Author  W.D.Hine
   40 REM BEEBUG  Jan/Feb 1990
   50 REM Program subject to copyright
   60 :
```

```
  100 MODE 7:ON ERROR GOTO220
  110 start=&8C00:size=&600:DIMcode &640
  120 PROCinitvars:PROCassemble
  130 PROClinks
  140 PRINT"SAVE and LOAD ROM? Y/N"
  150 A=GET AND&DF:IF A<>ASC"Y" THEN END
  160 OSCLI("SAVE edirom2 "+STR$~code+"+
"+STR$~(O%-code))
  170 *SRLOAD edirom2 8C00 X
  180 *SRSAVE edirom 8000 9240 X
  190 PRINT"Press Ctl-Brk to initialise"
  200 END
  210 :
  220 MODE7:REPORT:PRINT" at line ";ERL
  230 END
  240 :
 1000 DEF PROCinitvars
 1010 ipbuff=&600:buffer=&700:page=&18
 1020 top=&12:commandln=&F2:nl=&0D:cr=nl
 1030 rem=&F4:def=&DD:quote=ASC""""
 1040 numtoken=&8D:to=&B8:space=ASC" "
 1050 star=ASC"*":query=ASC"?"
 1060 oswrch=&FFEE:osnewl=&FFE7
 1070 osbyte=&FFF4:osword=&FFF1
 1080 osrdch=&FFE0:osasci=&FFE3
 1090 romexit=&8803:escexit=&8806
 1100 initscanft=&8809:scanprog=&880C
 1110 detokline=&880F:prntline=&8812
 1120 prnttext=&8815:notimp=&8818
 1130 initbptr=&881B:initpptr=&881E
 1140 inittptr=&8821:reinitbptr=8824
 1150 initvector=&8827:prnteoprog=&882A
 1160 flags=&70
 1170 REM 0=print leading spaces
 1180 REM 1=assembler opcode
 1190 REM 2=find tokenised line no
 1200 REM 6=first signif digit found
 1210 REM 7=assembler on/off
 1220 ay=&71:ex=&72:wy=&73:strlen=&74
 1230 str2len=&75:token=&76:tokn=&77
 1240 hitokn=&78:bptr=&80:tptr=&82
 1250 pptr=&84:action=&8A:lo=&86:hi=&87
 1260 scantype=&88:ipdelim=&8C
 1270 ENDPROC
 1280 :
 1290 DEF PROCassemble
 1300 FOR pass=4 TO7 STEP3
 1310 P%=start:O%=code:[ OPT pass
 1320 .proccalls
 1330  JMP findbasic:JMP findprocfn
 1340 .lfrom JMP &9200
 1350 .lproc JMP &9203
 1360 .lfn JMP &9206
 1370 .rbasic JMP &9209
 1380 .rtext JMP &920C
```

```
1390 .sysinf JMP &920F
1400 .varlist JMP &9212
1410 .fkdefs JMP &9215
1420 .pcomm JMP &9218
1430 .qcomm JMP &921B
1440 .rcomm JMP &921E
1450 .scomm JMP &9221
1460 .tcomm JMP &9224
1470 .ucomm JMP &9227
1480 .vcomm JMP &922A
1490 .wcomm JMP &922D
1500 .xcomm JMP &9230
1510 .ycomm JMP &9233
1520 .zcomm JMP &9236
1530 .rtncalls
1540 JMP initscanfb:JMP initscanfkw
1550 JMP ipbasparam:JMP testdelim
1560 JMP testopcode
1570 .findbasic
1580 LDA#0:STAflags:JSR ipbasparam
1590 JSR initbptr:JSR initvector
1600 JSR initscanfb:JSR testopcode
1610 JSR scanprog
1620 JSR prnteoprog:JMP romexit
1630 .initscanfb
1640 LDA#scanlfnameorval MOD256:STA lo
1650 LDA#scanlfnameorval DIV256:STA hi
1660 LDA#4:BITflags:BNE initscanlnno
1670 JSR initpptr:JSR testkeywd
1680 BEQ testquotes:JMP initscanfkw
1690 .initscanlnno
1700 LDA#&8D:STA tokn:LDA#0:STA hitokn
1710 .initscanfkw
1720 LDA#scanlineforkw MOD256:STA lo
1730 LDA#scanlineforkw DIV256:STA hi
1740 JMP initsfbexit
1750 .testquotes
1760 LDAipbuff:CMP#quote:BNEinitsfbexit
1770 LDA#scanlineforqstr MOD256:STA lo
1780 LDA#scanlineforqstr DIV256:STA hi
1790 .initsfbexit LDA lo:STA scantype
1800 LDA hi:STA scantype+1:RTS
1810 .testopcode
1820 LDAstrlen:CMP#3:BEQ cdbeopcode
1830 CMP#4:BNE notopcode:LDX#2
1840 LDAipbuff,X:AND#&DF
1850 CMP#ASC"U":bne notopcode
1860 .cdbeopcode
1870 LDAflags:ORA#2:STAflags
1880 .notopcode RTS
1890 .scanlfnameorval
1900 CMP#nl:BEQscannvexit
1910 CMP#rem:BEQ scannvexit
1920 LDAflags:BPL scanlforelem
1930 JSR scanassem
```

```
1940 JMPscanlfnameorval
1950 .scanlforelem JSR elemloop
1960 JMPscanlfnameorval
1970 .scannvexit RTS
1980 .scanassem
1990 JSR detokline:INY
2000 .assemloop
2010 LDAbuffer,Y:CMP#nl:BEQ asslpexit
2020 CMP#ASC":":BEQ nextasslp
2030 CMP#quote:BEQ skipq
2040 CMP#space:BEQ nextasslp
2050 CMP#ASC"]":BEQ quitassem
2060 CMP#ASC".":BEQ label
2070 CMP#ASC"\":BEQ nextstmt
2080 CMP#ASC";":BEQ nextstmt
2090 JMP sostmt
2100 .nextasslp INY:JMP assemloop
2110 .asslpexit RTS
2120 .nextstmt JMP skiptostmt
2130 .label
2140 LDX#0:LDA buffer,Y:CMP ipbuff
2150 BEQ labelmatchloop:CMP#ASC"."
2160 BEQ nextlabel:LDA ipbuff
2170 CMP#query:BEQ labelmatchloop
2180 JMP notlabelmatch
2190 .nextlabel INY:JMP label
2200 .labelmatchloop INX:INY
2210 LDA ipbuff,X:CMP#cr:BEQ eolabel
2220 CMP buffer,Y:BEQ labelmatchloop
2230 CMP#query:BEQ labelmatchloop
2240 CMP#star:BEQ labelmatch
2250 JMP notlabelmatch
2260 .eolabel LDA buffer,Y:CMP#space
2270 BEQ labelmatch:CMP#ASC":"
2280 BEQlabelmatch:CMP#nl:BEQlabelmatch
2290 .notlabelmatch
2300 JSR skiptodelim:JMP assemloop
2310 .labelmatch:JMP assemmatch
2320 .skipq JSRskipqloop:JMPassemloop
2330 .skipqloop INY:LDAbuffer,Y
2340 CMP#nl:BEQ sqexit:CMP#quote
2350 BNE skipqloop:INY:.sqexit RTS
2360 .quitassem
2370 LDA flags:AND#&7F:STA flags:LDY#4
2380 .qaloop LDA(bptr),Y:INY:CMP#ASC"]"
2390 BNE qaloop:CMP ipbuff:BNE saexit
2400 JMP assemmatch
2410 .sostmt LDX#0:STYwy
2420 LDA#2:BITflags:BEQskipopcode
2430 .opcodeloop
2440 LDA buffer,Y:AND#&DF:STA ay
2450 LDA ipbuff,X:AND#&DF:CMP ay
2460 BNE skipopcode:INY:INX:CPX strlen
2470 BEQ assemmatch:JMP opcodeloop
2480 .saexit RTS
```

```
2490 .nextassem INY
2500 .newstmt JMP assemloop
2510 .skipopcode LDYwy:INY:INY
2520 .testu
2530 LDA buffer,Y:INY:AND#&DF
2540 CMP#ASC"U":BNE movetooperand:INY
2550 .movetooperand
2560 LDX#0:LDAbuffer,Y:CMP#nl:BEQsaexit
2570 CMP#ASC":":BEQ newstmt
2580 CMP ipbuff:BEQ opmatchloop
2590 CMP#ASC"\":BEQ skiptostmt
2600 CMP#ASC";":BEQ skiptostmt
2610 CMP#quote:BEQ skipq2
2620 JSR testdelim:BEQ nextmovetoop
2630 LDAipbuff:CMP#query:BEQopmatchloop
2640 BNEnoopmatch
2650 .nextmovetoop INY:JMPmovetooperand
2660 .skipq2
2670 JSRskipqloop:JMPmovetooperand
2680 .opmatchloop
2690 INY:INX:LDA buffer,Y:JSR testdelim
2700 BEQ testopmatch:CMP ipbuff,X
2710 BEQopmatchloop:LDAipbuff,X
2720 CMP#query:BEQ opmatchloop:CMP#star
2730 BEQ assemmatch:JMP noopmatch
2740 .testopmatch
2750 LDAipbuff,X:CMP#cr
2760 BEQassemmatch:CMP buffer,Y
2770 BEQ opmatchloop:JMP noopmatch
2780 .assemmatch
2790 LDA#(skiptostmt-1)DIV256:PHA
2800 LDA#(skiptostmt-1)MOD256:PHA
2810 STYex:JMP(action)
2820 .skiptostmt
2830 LDA buffer,Y:CMP#nl:BEQ saexit2
2840 CMP#ASC":":BEQ newstmt2:INY
2850 JMP skiptostmt
2860 .newstmt2 JMP assemloop
2870 .saexit2 RTS
2880 .noopmatch
2890 JSR skiptodelim:JMPmovetooperand
2900 .skiptodelim LDA buffer,Y
2910 JSR testdelim:BEQ stdexit
2920 INY:JMP skiptodelim
2930 .stdexit RTS
2940 .elemloop
2950 LDX#0:JSR movetoelement
2960 CMP#nl:BEQ elemlpexit
2970 CMP#rem:BEQ elemlpexit
2980 CMP#star:BEQ testinitstar
2990 CMP#quote:BNE elemloopcont
3000 JSR skipstrloop:JMP elemloop
3010 .elemlpexit JMPelemexit
3020 .testinitstar
3030 CMP ipbuff:BEQ matchloop
```

```
3040 INY:JSR movetodelim:JMP elemloop
3050 .elemloopcont
3060 CMP#ASC"[":BEQ assemon
3070 CMP#ASC"$":BEQ nextelemloop
3080 CMP ipbuff:BEQ matchloop
3090 LDAipbuff,X:CMP#query:BEQmatchloop
3100 JSR movetodelim:JMP elemloop
3110 .nextelemloop
3120 INY:LDA(bptr),Y:JMP elemloop
3130 .assemon LDAflags:ORA#&80
3140 STA flags:JSR detokline:LDY#4
3150 .aonlp INY:LDA buffer,Y:CMP#ASC"["
3160 BNE aonlp:CMP ipbuff:BNE elemexit
3170 JMP nameorvalmatch
3180 .matchloop INY:INX
3190 LDA(bptr),Y:JSR testdelim
3200 BEQ testeobasicstr:LDA ipbuff,X
3210 CMP#cr:BEQ testeobasicstr
3220 CMP(bptr),Y:BEQ matchloop
3230 LDA ipbuff,X:CMP#query
3240 BEQ matchloop:CMP#star
3250 BEQ starmatch
3260 .testeobasicstr LDA(bptr),Y
3270 JSR testdelim:BEQ testeomatchstr
3280 JSR movetodelim:JMP nextelem
3290 .starmatch JSR movetodelim
3300 JMP nameorvalmatch
3310 .testeomatchstr LDA ipbuff,X
3320 CMP#cr:BNE nextelem
3330 .nameorvalmatch
3340 LDA#(nextelem-1)DIV256:PHA
3350 LDA#(nextelem-1)MOD256:PHA
3360 JSRdetokline:JMP(action)
3370 .nextelem JMP elemloop
3380 .elemexit RTS
3390 .movetoelement
3400 LDA(bptr),Y:CMP#nl:BEQ mteexit
3410 CMP#rem:BEQ mteexit:CMP#quote
3420 BEQ mteexit:CMP#star:BEQ mteexit
3430 JSR testdelim:BNE mteexit
3440 INY:JMP movetoelement
3450 .mteexit RTS
3460 .movetodelim LDA(bptr),Y
3470 JSR testdelim:BEQ mtdexit:INY
3480 JMP movetodelim:.mtdexit RTS
3490 .skipstrloop INY:LDA(bptr),Y
3500 CMP#nl:BEQ sslexit:CMP#quote
3510 BNE skipstrloop:INY:.sslexit RTS
3520 .scanlineforkw LDA(bptr),Y
3530 CMP#nl:BNE scanfortokn:RTS
3540 .scanfortokn
3550 CMP tokn:BEQ tokenmatch
3560 CMP hitokn:BEQ tokenmatch
3570 INY:JMP scanlineforkw
3580 .tokenmatch
```

```
3590 LDA#4:BITflags:BNE testlineno
3600 LDAtokn:CMP#&B8:BNE tokenmatch2:
3610 LDA#ASC"P":CMPipbuff+2:BEQ testtop
3620 INY:CMP(bptr),Y:BEQscanlinekwnext
3630 DEY:JMP tokenmatch2
3640 .testtop INY:CMP(bptr),Y
3650 BNE scanlinekwnext
3660 .tokenmatch2
3670 LDA#(scanlinekwnext-1)DIV256:PHA
3680 LDA#(scanlinekwnext-1)MOD256:PHA
3690 INY:JSR detokline:JMP(action)
3700 .scanlinekwnext
3710 JMP scanlineforkw
3720 .testlineno
3730 STYwy:JSR detokline:LDYex:LDX#0
3740 .lnnomatchloop LDA buffer,Y
3750 JSR testdelim:BEQ checklnnomatch
3760 CMP ipbuff,X:BNE nolnnomatch
3770 INX:INY:JMP lnnomatchloop
3780 .checklnnomatch LDAipbuff,X:CMP#cr
3790 BNE nolnnomatch:JSR resety
3800 JMP tokenmatch2
3810 .nolnnomatch
3820 JSR resety:JMP scanlineforkw
3830 .resety LDYwy:INY:INY:INY:RTS
3840 .scanlineforqstr
3850 LDA(bptr),Y:CMP#nl:BNE scanq1
3860 .slfqstrexit RTS
3870 .scanq1 CMP#quote:BEQ qmatch
3880 INY:JMP scanlineforqstr
3890 .qmatch LDX#0
3900 .qmatchloop INY:INX:LDA ipbuff,X
3910 CMP#cr:BEQ qstrmatch
3920 LDA(bptr),Y:CMP#nl:BEQ slfqstrexit
3930 CMP ipbuff,X:BEQ qmatchloop
3940 JMP scanq2
3950 .qstrmatch
3960 LDA#(scanq2-1)DIV256:PHA
3970 LDA#(scanq2-1)MOD256:PHA
3980 JSRdetokline:JMP(action)
3990 .scanq2
4000 .scanq2loop LDA(bptr),Y:CMP#nl
4010 BEQ slfqstrexit:INY:CMP#quote
4020 BEQ scanlineforqstr:JMP scanq2loop
4030 .testkeywd
4040 JSR inittptr
4050 .testkw2
4060 LDY#0:STY tokn:STY hitokn
4070 LDA(pptr),Y:CMP(tptr),Y:BNE nextkw
4080 INY
4090 .kwlp
4100 LDA (pptr),Y:CMP#ASC"."
4110 BEQ keywdspot:CMP#cr:BEQ kwmtch
4120 CMP(tptr),Y:BEQnextkwlp
4130 LDA(tptr),Y:CMP#&B8:BNE nextkw
```

```
4140 LDA(pptr),Y:CMP#ASC"P":BEQ kwmtch
4150 .nextkwlp INY:JMP kwlp
4160 .nextkw
4170 INY:LDA(tptr),Y:BPL nextkw:PHA
4180 INY:INY:TYA:CLC:ADC tptr:STA tptr
4190 LDA#0:ADC tptr+1:STA tptr+1:PLA
4200 CMP#&D3:BNE testkw2:RTS
4210 .keywdspot
4220 LDA(tptr),Y:STA(pptr),Y:INY
4230 CMP#0:BPLkeywdspot:DEY:STYstrlen
4240 LDA#nl:STA(pptr),Y
4250 .kwmtch LDA(tptr),Y:BPL nextkw
4260 STAtokn:CMP#&8F
4270 BCCfkwdtknexit:CMP#&94:BCCsettokn2
4280 RTS
4290 .settokn2
4300 CLC:ADC#&40:STA hitokn
4310 .fkwdtknexit RTS
4320 .testdelim
4330 CMP#ASC"#"+1:BCC delim
4340 CMP#ASC"'":BCC char
4350 CMP#ASC"&"+1:BCC delim
4360 CMP#ASC".":BEQ char
4370 CMP#ASC"0":BCC delim
4380 CMP#ASC"9"+1:BCC char
4390 CMP#ASC"@":BCC delim
4400 CMP#ASC"^":BEQ delim
4410 CMP#ASC"z"+1:BCC char
4420 .delim LDA#0:.char CMP#0:RTS
4430 .ipbasparam
4440 LDA(commandln),Y:INY:CMP#space
4450 BEQ ipbasparam:DEY:DEY:LDX#&FF
4460 .bparamlp INY:INX:LDA(commandln),Y
4470 CMP#ASC"^":BEQ initlinenoflag
4480 STA ipbuff,X:CMP#nl
4490 BNE bparamlp:STX strlen
4500 .trsplp
4510 CPX#0:BEQ ippxit:DEX
4520 LDAipbuff,X:CMP#space:BEQ trsplp
4530 INX:LDA#nl:STA ipbuff,X:STX strlen
4540 .ippxit RTS
4550 .initlinenoflag LDA flags:ORA#4
4560 STA flags:LDX#&FF:JMP bparamlp
4570 .findprocfn
4580 LDA#def:STA tokn:LDA#0:STA hitokn
4590 JSR initbptr:JSR initvector
4600 JSR initscanfkw:JSR scanprog
4610 JSR prnteoprog:JMP romexit
4620 ]NEXT:ENDPROC
4630 :
4640 DEF PROClinks
4650 P%=start+size:O%=code+size
4660 FORI%=0 TO20
4670 [OPT7:JMP notimp:]
4680 NEXT:ENDPROC
```

1st course

Improving Basic programs

by Mike Williams

In this month's first course I want to try and deal with a number of ideas and techniques with the aim of helping you to improve the quality of your programs, both in the way in which they are written, and in the way in which they interact with the user. I have not infrequently come across parts of programs which would have benefitted from some fine tuning, and it is on some of the points raised in this way that this article is based.

Before going into any detail, let us be clear, though, what we are talking about. First of all there is the matter of program style. I can tell you without any doubt, that programs which make frequent use of GOTO are far more difficult to understand, and thus to modify. As a result it is easier to overlook some small bug in a program. How can such programs be improved? The answer lies in the frequent use of procedures and functions, and REPEAT-UNTIL loops where these are appropriate. We will come back to this point later.

A further point on which we can judge a program is its conciseness. Why have 100 lines in a program if 50 will do the same job. In some cases, particularly on a model B, the amount of memory used by the program can become quite critical, so any opportunity to reduce the memory needed can be very useful. However, highly condensed programs can be unreadable, contrary to my earlier point, so conciseness should not be overdone, except in extremis.

To achieve any reduction in the size of a program, what we are usually looking for are examples of redundancy. For example if the same task (more or less) occurs more than once in a program, then we should be looking for ways of coding this as a function or procedure and then just calling that when required. There are other examples of redundancy which can also be identified and dealt with.

So far what I have referred to will largely go unseen by the user of the program, and you may ask if that is the case what does the nature of the coding matter. Well I have already given some reasons to justify this, and there are others. However, the other aspect of programming that I want to deal with is the user interface, that is what the user sees on the screen as he runs the program, and particularly how the user communicates or interacts with the program. It is this subject that I want to deal with first.

HANDLING USER INPUT

Many programs, particularly at the outset, require some dialogue with the user. The program needs a number of pieces of information before it can proceed. Often it is desirable for the program to check the validity of the information input before attempting to use it. And that is a point in itself. It is much better for a program to handle any potential error, rather than face the user with some meaningless error message from the operating system or filing system.

But the crux of the matter is what the user sees on the screen. The simplest solution, and one often adopted is to go on repeating a question until an acceptable answer is typed in (which is fair enough), but to do this in such a way that the repeated prompts and responses just scroll down the screen. In my view this leads to an untidy display, it doesn't look very slick, and worst of all other perfect valid information already on the screen may be lost from view.

I don't like scrolling screens, and it's easy to achieve something which I think is much better. First of all, I have thrown my usual programming principles to the wind and written something which is fairly typical of the usual approach to this task (we assume we are using a DFS system where file names are limited to 7 characters):

```
100 PRINT"File name: ";
110 INPUT fname$
120 IF LEN(fname$)>7 THEN GOTO 100
```

This could be reduce to a mere two lines by combining the first two lines:

```
100 INPUT"File name: " fname$
110 IF LEN(fname$)>7 THEN GOTO 100
```

but this runs into other problems (see later) so I have kept things simple.

I dislike this for two reasons: one is its use of GOTO, which may not seem bad in what is quite a simple example, while the other is the scrolling effect produced on the screen by a sequence of invalid inputs. The first point is easily dealt with by rewriting the three lines as:

```
100 REPEAT
110 PRINT"File name: ";
120 INPUT fname$
130 UNTIL LEN(fname$)<8
```

Note how the condition has become reversed, because we now repeat the loop until the answer is valid, whereas before the loop was repeated if the answer were unacceptable. In addition, our original three lines have now become four. You will often find that in small scale examples like this, the more structured approach, which I advocate, appears to increase the length of the program. Sometimes this is so, but overall I do not believe there is much either way, and the above example is actually five bytes shorter than the previous version.

Now we can consider the second point about the original code. A succession of incorrect responses will cause the question (and answer) to repeat down the screen, eventually scrolling the screen. How can we prevent this? The first step is to decide just what it is we really want the program to do. I suggest that the program should display the prompt, and wait for the input. If the response is not valid, it should be wiped from the screen, repositioning the cursor at the end of the prompt ready for re-input.

As with any piece of programming there is more than one way of achieving this result. Here is one possibility:

```
100 REPEAT
110 PRINT TAB(0,n)"File name:";SPC20
120 INPUTTAB(12,n) fname$
130 UNTIL LEN(fname$)<8
```

This method uses the TAB function to position both prompt and response on line 'n'. If the question has to be repeated after an invalid input, the cursor is moved back to the original screen line before repeating the prompt. Thus prompt and input both remain on the same line until a correct response is received.

Try it by typing in the four lines and then RUN. The 'n' will need to be replaced by a suitable value for this to work correctly.

Other points about this example are as follows. The TAB function also requires an 'x' value to indicate where on the line text or input is to appear. I assume our prompt starts at the left of the screen. The value of 12 in line 120 is determined by the length of the text prompt and indicates where the flashing cursor will be displayed at the start of input.

Incorrect responses are wiped out by simply 'printing' 20 spaces after the prompt (if not 20, whatever is appropriate for the screen mode and likely input).

One minor criticism of the above code might be that the prompt is unnecessarily redisplayed whenever incorrect information is received. This happens so quickly that it is hardly discernible, but that too could be avoided by writing:

```
100 PRINT(0,n)"File name:"
110 REPEAT
120 PRINTTAB(12,n) SPC20
130 INPUTTAB(12,n) fname$
140 UNTIL LEN(fname$)<8
```

But this now takes up an extra line of code. Unfortunately, although an INPUT statement can include text in quotes which is to be displayed as a prompt, you cannot then use the SPC function. The 20 spaces would have to be explicitly written as such within quotes. The version above achieves a marginal increase in speed at the expense of a longer program. In such situations you must decide what is more important for your needs.

All this has taken much longer to explain than to do. I hope you have followed my example sufficiently that you will be able to use the same technique in your own programs wherever a prompt for input needs to be checked. If the check is a complex one then write this as a function which returns the value TRUE or FALSE depending on the result of the check. The last line then becomes:

```
130 UNTIL FNcheck(fname$)
```

For example, the function could check both that the filename is itself valid, and that, if so, the file specified does exist.

AN EXAMPLE OF REDUNDANCY

I will deal with one other example this month, this time one concerning redundancy of code within the program, rather than how the screen appears to the user. The following is a slightly artificial example, but it will serve my purpose:

```
100 IF x=1 THEN y=y+1
110 IF x=2 THEN y=y+4
120 IF x=3 THEN y=y+9
```

I have seen examples similar to this in a good few programs. Now you might guess that the different values to be added onto 'y' bear some relationship to the value of x, and in this example it is not too difficult to see what that is. Each value is simply the square of the value of x. Thus the three lines above could be replaced by the one line:

```
100 y=y+x^2
```

and we don't even need to use an IF statement at all.

In other cases, deducing a suitable relationship can be more difficult. Unfortunately, there is no one single answer, but it is surprising how often the relationship is linear. Now this is a mathematical term, which means that the value to be added on (which I will represent as 'v') is related to the corresponding value of x by a simple formula:

```
      v = ax + b
```

where 'a' and 'b' are two constants (or numbers) whose value we need to determine.

For example, suppose we know that:

```
      IF x=1 THEN y=y-1
      IF x=2 THEN y=y+4
      IF x=3 THEN y=y+9
```

By using the values we know, (that v=-1 when x=1, v=4 when x=2 and that v=9 when x=3), we can substitute these values in turn giving:

```
      -1 =  a + b
       4 = 2a + b
       9 = 3a + b
```

Subtracting the first equation from the second gives:

```
       5 = a[4 - -1 giving 5]
```

Putting a=5 in the first equation gives:

```
       b = -6      [-1 = 5 + b]
```

We haven't used the third equation (and there might be others) at all, but it is important to check that the values we have calculated are valid in all the possible equations, and we can see here that:

```
      9 = 3*5 + -6[i.e. 9 = 15 - 6]
```

This means that the three IF statements can now be replaced by the single line:

```
      y=y+(5*x-6)
```

The saving may seem small here, but in practice can be very worthwhile. If the values obtained from your first two equations do not work with ALL the other equations you produce, then you have a problem - there is no linear relationship which applies in all cases. If there is just one exception then this can be catered for as such, viz:

```
100 IF x=18 y=y+23 ELSE y=y+(3*x-2)
```

If the equation fits hardly any of the possibilities, then it must be abandoned. The whole point is to find one format which works with the vast majority of cases. Then it is a case of resorting to some trial and error. For example, the sequence 5, 11, 21 is related to 1, 2, 3 by the relation:

```
      y = 2*x^2 +3
```

If the basic sequence is 1, 2, 3 etc it is often worth seeing if the squares of these numbers fit in at all (1, 4, 9 etc). As I say, it is all very much a case of trial and error.

The principle behind all this, as with so much else, is that realising that an alternative approach may be possible is half the battle to finding it, otherwise you won't even know to start looking. At this point I must stop, but I hope to cover more ideas along the same lines next time around. In the meantime I hope you have found this discussion to be of some use.

Using the ROM Filing System (Part 1)

Jon Keates provides a useful introduction to the little used ROM Filing System (RFS).

Many BBC users may be aware of the existence of the resident ROM Filing System, the RFS, but very few people seem to make any active use of it. This is usually because the RFS, rather oddly, makes no direct provision for allowing programs to be saved easily to the RFS, a crucial requirement, and because of the difficulty of assembling any RFS header in the first place. However, the RFS is a filing system which has its merits like any other, and which offers an alternative to cassette or disc filing systems on any machine with sideways RAM.

This month, in the first of a two part article, we publish a program called RFShead. This will assemble the necessary ROM image header which at least allows the RFS to function. In the next issue we shall provide a program that will load any suitable files from disc, and write them to a RAM bank correctly formatted for the RFS.

For those of you who are not familiar with the ROM Filing System, some words of explanation are necessary. It is entered by using the *ROM command, which then allows the use of the RFS commands, provided a suitable ROM or RAM image is already installed. The basic commands associated with the RFS (and listed in the User Guide) allow the use of Basic, Exec, and Machine Code files for input only from the RFS, but not for output, i.e. Saving, Spooling etc. This is why the RFS has to be assembled as a ROM image and then installed in a RAM bank, or blown in to an EPROM, before it can be used.

By its nature the RFS is not as versatile as the cassette filing system (CFS), but it is faster and less troublesome to use. It is slower than the disc filing systems, and through not having any facility to save data, it is much more limited in its application and use. But when it is used in a sensible way, considering the limitations, it is a very useful, cheap and always available alternative filing system.

This month's program is intended to make it as simple as possible to assemble and install an RFS ROM image header in a RAM bank. This ROM image will be required for next month's program which will load and update any already existing RFS images with new files.

PROGRAM INFORMATION

First type in and then save the program listed here. When RFShead is run, the program prompts for a title for the ROM image, a version number, and an indication of which RAM bank to use for the RFS. The machine code ROM image header is then assembled with O% set to &6000 and P% to &8000. After the code has been assembled, the the ROM image may be written to sideways RAM, or saved to disc for future use.

Once the ROM image header has been installed in a bank of sideways RAM, press Ctrl-Break to initialise it, and then check the header by using the command:

```
  *HELP RFS
```

This should produce a list of all the star commands recognised by the ROM image, plus the load and chain commands. The title and version number should also appear in the ROM table, when using the *ROMS command.

Both of the programs have been written on a Master 128, but should work on any BBC micro with sideways RAM and Basic II (or later). A few minimal alterations to the programs may be necessary to suit the different makes of RAM. The *SRWRITE command which is used on the Master will have to be replaced with the command that is relevant to your type of RAM.

TECHNICAL DETAILS

The RFS ROM header is in the same format as any other ROM image header, and requires the service entry point, where it has to recognise and support the necessary ROM service calls 13 and 14. These are needed for the Operating System to make use of the Rom Filling System.

When the MOS issues a ROM service call, the service call number is in the accumulator, and the Y register contains 15 minus the number of the next ROM to be scanned.

Service call 13 is required to initialise the RFS. It first checks the value in the Y register; if this is less than the current ROM being investigated then the call should be ignored. If not, the routine will then return the start address of the beginning of the file data in sideways RAM, in locations &F6/&F7. Then it will read the current ROM number from location &F4 and place it in location &F5. This will indicate to the MOS that RFS formatted ROMs are present in the machine and that the current ROM is active, allowing the use of service call 14.

Service call 14 is then used to retrieve the data a byte at a time. This is done using (&F6) as a pointer to the data, which is then returned in the Y register, with the accumulator set to zero. The pointer to the RFS image at location (&F6) is then automatically incremented, ready for the next call.

The only other ROM Service call supported is number 9, for HELP information. This is included for compatibility, and like other filing systems will respond to *HELP or *HELP RFS.

Next month we shall conclude the discussion of the RFS by showing how files can be loaded from disc and stored in this format in sideways RAM.

```
   10 REM  Program RFShead
   20 REM  Version 3.01
   30 REM  Author  Jon Keates
   40 REM  BEEBUG  Jan/Feb 1990
   50 REM  Program subject to copyright
   60 :
  100 MODE7
  110 ON ERROR REPORT:PRINT;" at Line ";
ERL:END
  120 PRINT''
  130 PRINT CHR$141"   RFS ROM Image Hea
der"
  140 PRINT CHR$141"   RFS ROM Image Hea
der"
  150 VDU28,2,23,37,8
  160 REPEAT VDU12
  170 INPUTLINE''"Title for RFS Image ..
  "title$
  180 REMINPUTLINE'"....   "title$
  190 INPUT''"Version Number       ... "v
er%
  200 INPUT''"RAM Bank Number      ... &"
bank$
  210 ramblok%=EVAL("&"+bank$)
  220 UNTIL LEN(title$)>0
  230 PROCassemble
  240 VDU12
  250 PRINT''"To Save RFS Image use:"
  260 PRINT''" *SAVE ";title$;" 6000 "+S
TR$~O%+" 8000 8000"
  270 PRINT''"To Install in SRAM bank ";
ramblok%;" use:"
  280 PRINT''" *SRWRITE 6000 "+STR$~O%+"
 8000 ";ramblok%
  290 END
  300 :
 1000 DEF PROCassemble
 1010 osasci%=&FFE3
 1020 FOR pas%=4 TO 6 STEP 2
 1030 O%=&6000:P%=&8000
 1040 [
 1050 OPT pas%
 1060 EQUW &0000:EQUB &00:JMP entry
 1070 EQUB &82:EQUB LEN(title$)+9
 1080 EQUB ver%:EQUS title$:EQUB &0
 1090 EQUS "(C) BEEBUG 90":EQUB &0
 1100 .entry
 1110 CMP #13:BNE ent_2:JMP file
 1120 .ent_2
 1130 CMP #14:BNE ent_3:JMP byte
 1140 .ent_3
 1150 CMP #9:BNE noclaim:JMP help
 1160 .noclaim
 1170 RTS
 1180 .byte    \service call 14
 1190 PHA:LDA &F5:EOR #&F:CMP &F4
 1200 BNE out:LDY #0:LDA (&F6),Y
 1210 TAY:INC &F6:BNE claimed
 1220 INC &F7:JMP claimed
 1230 .file    \service call 13
 1240 PHA:TYA:EOR #&F
 1250 CMP &F4:BCC out
 1260 LDA #((data) MOD 256):STA &F6
 1270 LDA #((data) DIV 256):STA &F7
 1280 LDA &F4:EOR #&F:STA &F5
 1290 .claimed
 1300 PLA:LDA #0:RTS
```

```
1310 .out
1320 PLA:RTS
1330 .help    \service call 9
1340 PHA:LDA (&F2),Y:CMP #13
1350 BNE help_2:LDX #0
1360 .loopA
1370 LDA message_1,X:BEQ exitA
1380 JSR osasci%:INX:JMP loopA
1390 .exitA
1400 PLA:RTS
1410 .message_1
1420 EQUB &0D:EQUS "RAM Filing System"
1430 EQUW &0D0D:EQUS "  RFS"
1440 EQUW &000D
1450 .help_2
1460 TYA:PHA:LDX #0
1470 .loopB
1480 LDA (&F2),Y:AND #&DF
1490 CMP #&21:BCC skipB
1500 CMP name,X:BNE exitB
1510 INX:INY:JMP loopB
1520 .skipB
1530 LDA name,X:BNE exitB:JMP help_3
```

```
1540 .exitB
1550 PLA:TAY:PLA:RTS
1560 .name
1570 EQUS "RFS":EQUB &0
1580 .help_3:LDX #0
1590 .loopC
1600 LDA message_2,X:BEQ exitB
1610 JSR osasci%:INX:JMP loopC
1620 .message_2
1630 EQUB &0D:EQUS " RFS  * Commands  :
"
1640 EQUW &0D0D:EQUS "  EXEC  LOAD  RUN
"
1650 EQUB &0D:EQUS "  DUMP  CAT   EX"
1660 EQUB &0D:EQUS "  LIST  TYPE  PRINT
"
1670 EQUW &0D0D
1680 EQUS "  LOAD <fsp>  CHAIN <fsp>"
1690 EQUW &000D
1700 .data EQUB &2B
1710 ]
1720 NEXT
1730 ENDPROC
```

Dichotomous Keys (continued from page 12)

```
 2750 PRINTTAB(1,n);"Qstn : ";q$
 2760 PRINTTAB(1,n+2);"Yes - ";yes$;TAB(
41,n+2);"No  - ";no$:ENDPROC
 2770 :
 2780 DEF PROCedit
 2790 PROCwindow(6,-1):PROCtitle("Please
 enter the above information. Press RETU
RN at each field.",0):PROCwindow(3,0)
 2800 q$=FNenter(0,12,70,q$,"","Qstn :")
 2810 yes$=FNenter(0,14,20,yes$,"","Yes
-")
 2820 no$=FNenter(40,14,20,no$,"","No  -
")
 2825 PROCwindow(6,-1):IF FNsure("Confir
m this information.") THEN PROCwriterec(
qstn)
 2840 CLS:ENDPROC
 2845 :
 2990 DEF FNgo:LOCAL I
 3000 PROCwindow(6,-1):I=VALFNenter(30,0
,3,"","0123456789","Question number?")
 3010 CLS:IF I=0 OR I>ite% THEN =qstn EL
SE =I
 3015 :
 3020 DEF PROCrunkey
 3025 LOCAL I:menu%=-1:F=FNfilename(0)
 3040 IF F=0 THEN ENDPROC
```

```
 3050 REPEAT:done%=0:qstn=1
 3060 REPEAT:PROCrunrd
 3070 I=FNyn:IF I THEN done%=VALyes$ ELS
E done%=VALno$
 3075 IF done%>0 THEN qstn=done%
 3080 UNTIL done%=0:IF I THEN ite$=yes$
ELSE ite$=no$
 3090 PROCwindow(3,-1):PROCtitle("YOUR S
UBJECT HAS BEEN IDENTIFIED:",7)
 3100 PROCtitle("It is "+ite$,9)
 3110 PROCwindow(6,-1):PROCtitle("Do you
 wish to use this Key again? (Y/N)",0)
 3120 I=FNyn:CLS:UNTIL NOT I:ENDPROC
 3125 :
 3130 DEF PROCrunrd:PROCwindow(3,-1)
 3150 PROCreadrec(qstn):PROCtitle(q$,7)
 3160 PROCtitle("Press Y for YES",9):PRO
Ctitle("Press N for NO ",10):ENDPROC
 3170 :
 3350 DEF PROCstar:menu%=-1
 3360 CLS:PROCtitle("Enter your Star Com
mands",0):PRINTTAB(0,0);
 3370 REPEAT:PRINT':INPUT"*"I$:PRINT:OSC
LI(I$)
 3390 PRINT'"Press any key to continue";
 3400 UNTIL GET<>ASC("*"):CLS:ENDPROC
```

Procedure/Function Appender

Paul Pibworth describes a very useful programming utility which locates procedures and functions within programs on disc, and appends them to the current program in memory.

Program writers who are highly organised have discs packed with routines, in the form of procedures or functions which they can select and use with apparent ease, by *EXECuting them in as required. Those who are disorganised, on the other hand, have difficulty finding their discs, let alone the routines they want!

Somewhere between the two are programmers who use the same routines more than once, and know where they can be found. Normally these routines would be used either by typing them in from a listing (if you have a printer), or by merging the whole of the program containing the routine, and then deleting all the unwanted lines.

The program described in this article offers a third possibility, based on the concept that it should be possible for the computer to do the work (isn't that what they are for?). In order to be of use, any routine must work without losing the program which is in memory. It also needs to be fast. All this suggests a machine code routine, which can move through a program on disc, looking for a given procedure or function, and when found, add it to the end of the program in memory.

Type in the listing, being very careful to distinguish between 1 (one) and l (ell), particularly with label names such as .loop1, .loop2 etc. Having typed it in, save the source program and then run it. The code is assembled at &6000 to allow you to have another machine code utility sitting above, but you could raise this if you wish, by altering line 100. Once it has been assembled, you will be prompted to save the routine to disc with the suggested name of GETDEF. The routine can then be installed by typing *RUN GETDEF, or just *GETDEF.

Once loaded, the routine can be called at any time with the function key f0. A different function key could be used, by changing line 2910. The on-screen directions are given in sequence, and are easy to follow.

The outline of the routine is as follows:

1. Request program name.
2. Check that it exists, and open it as a file.
3. Request whether procedure or function.
4. Request procedure/function name.
5. Search for the given procedure.
6. When found, display directions.
7. Display the procedure, line by line on the screen, with the option to append or exit.
8. Close the file.
9. Renumber the program and return to Basic.

As written, the routine runs in mode 7, which gives the advantages of clarity and extra memory. If you normally work in another mode, then add the line:

```
LDA#22:JSRosa:LDA#7:JSRosa
```

immediately before the labels .title and .col (lines 260 and 360). This will cause the routine to switch to mode 7 before running, and the colour codes and windows which are then set up need not be changed.

Program names have been limited to 12 characters (CPX#12 in subroutine .prog3). This allows you to use a name such as:

```
:1.B.PROGRAM
```

which is useful if you have a double drive and DFS. The utility does work with ADFS, but you would need to increase this limit when accessing sub-directories with long path names, or alternatively use *DIR first to set the correct directory.

The program to be searched is treated as a file, and opened using OSFIND (details in the Advanced Disc User Guide). The file handle is returned in the accumulator, checked for 0 (file does not exist), and stored for use later.

Because the same name can be used for both procedures and functions, you must first select P or F (upper case only), and then enter the program name. Since many programmers use lower case names, caps lock is turned off at this point (line 1020). It can be turned on again with the Caps Lock key, or this line could be omitted. Again, a limit has been set for name length, but this is quite arbitrary. Do not include parameters (brackets) in your procedure name, even if they exist.

The open file is now read, using OSBGET, where Y = the filehandle (from the store). The first byte (13) is ignored, and the rest of the line is stored in a buffer at &900. If the second byte in the line is &FF, the end of the file has been reached.

The stored line is examined in turn for DEF, then PROC or FN, then the specified name. Unless all three are found, the next line is read in and examined. Although transferring a whole line wastes a little time, the disc sector is already in the official disc buffer anyway, and the position of the file pointer is maintained. Once the correct procedure or function has been found, a flag is set to skip the examination process.

It might be thought that the simplest course of action would be to append lines automatically until ENDPROC is found. However, there can be more than one ENDPROC, (and more than one "=" terminator in a function), and it would require a considerable amount of extra programming to cater for this. Also, there are many instances where a group of procedures belongs together.

Each line is therefore displayed for confirmation in turn. enabling you to keep a visual check on what you are appending, and to decide when to stop. The Basic ROM is used to detokenise the line in order to display it (see *Basic Line Editor*, BEEBUG Vol.8 No.2).

The Basic ROM addresses used are those for the Master. BBC B (Basic II) and Compact users should change the values in line 150 as shown:

BBC B	`rom1=&67, rom2=&80`
Compact	`rom1=&23`

Tokenised line numbers are displayed as two question marks, since they would be inaccurate anyway when renumbered.

The .append routine copies the value of TOP minus one into pointers, then uses the line length byte to update the value of TOP. When all the bytes in the line have been appended, from the buffer in &900, the value &FF (which signifies end of program) is added. Since line numbers in procedure libraries are usually kept high, the original line numbers are retained while being appended, and then REN. is placed into the keyboard buffer, which automatically renumbers the whole program when exiting from the routine. Any tokenised numbers will then need to be adjusted. This should only be a problem with lines containing RESTORE, which can easily be altered by hand.

Finally, the program closes the file, again using OSFIND, and turns Caps Lock back on.

I use this utility in conjunction with another routine which then moves the procedure into a new position within the Basic program. You may wish to refer to the program *Resequencer* by David Spencer in BEEBUG Vol.7 No.6.

This is not a short utility, but it is extremely useful. I only wish I had written it earlier!

```
  10 REM Program ProcApp
  20 REM Version B1.0
  30 REM Author  Paul Pibworth
  40 REM BEEBUG  Jan/Feb 1990
  50 REM Program subject to copyright
  60 :
 100 code=&6000
 110 top=&12
 120 osw=&FFEE:osn=&FFE7:osa=&FFE3
 130 osf=&FFCE:osb=&FFF4:osc=&FFF7
 140 osbg=&FFD7:osr=&FFE0
```

```
  150 rom1=&C:rom2=&84:and=&81:REM refer
ences in BASIC 4 ROM
  160 ptr=&70
  170 store1=&72:store2=&73
  180 store3=&74:store4=&75
  190 proclen=&76:linelen=&77
  200 found=&78:REM proc found flag
  210 appended=&79:REM appended flag
  220 rem=&7A:quote=&7B:REM Flags for RE
Ms and Quotes
  230 FOR pass=0 TO 2 STEP 2:P%=code
  240 [OPT pass
  250 LDY #255
  260 .title
  270 INY:LDA intro,Y:JSR osa
  280 CPY #36:BNE title
  290 JSRosn
  300 LDX #fkey MOD256:LDY #fkey DIV256
  310 JMPosc
  320 .start
  330 LDA #0:STA found:STA appended
  340 STA rem:STA quote
  350 LDY #255
  360 .col
  370 INY:LDA colour,Y:JSR osw:JSR osn
  380 CPY #13:BNE col
  390 LDY #255
  400 .window1
  410 INY:LDA win1,Y:JSR osw
  420 CPY #4:BNE window1
  430 LDX #0:LDY #0:JSR tab
  440 LDX #mess1 MOD256
  450 LDY #mess1 DIV256
  460 JSR display
  470 .prog1
  480 LDX #20:LDY #0:JSR tab
  490 LDX #blank MOD256
  500 LDY #blank DIV256
  510 JSR display
  520 LDX #0:LDA #0
  530 .prog2
  540 STA &A00,X:INX:CPX #11
  550 BNE prog2
  560 LDX #20:LDY #0:JSR tab
  570 LDX #0
  580 .prog3
  590 JSR osr
  600 CMP #13:BEQ prog4
  610 CMP #127:BEQ prog1
  620 CMP #27:BEQ prog4
  630 STA &A00,X:JSR osa
  640 INX:CPX #12:BEQ prog5
  650 JMPprog3
```

```
  660 .prog4
  670 CMP #27:BNE prog5
  680 LDA #12:JSR osa:JMP end
  690 .prog5
  700 LDA #13:STA &A00,X
  710 LDX #0:LDY #&A:LDA #&40
  720 JSR osf:CMP #0:BEQ nofile
  730 STA store2:JMP proc1
  740 .nofile
  750 LDX #20:LDY #0:JSR tab
  760 LDX #mess2 MOD256
  770 LDY #mess2 DIV256
  780 JSR display:JSRosr:JMP end
  790 .proc1
  800 LDX #0:LDY #1:JSR tab
  810 LDX #mess3 MOD256
  820 LDY #mess3 DIV256
  830 JSR display
  840 .proc2
  850 JSR osr
  860 CMP #70:BEQ procF
  870 CMP #80:BNE proc2
  880 JSR osw
  890 LDA #&F2:STA store1
  900 LDX #0:LDY #2:JSR tab
  910 LDX #messP MOD256
  920 LDY #messP DIV256
  930 JSR display:JMP proc3
  940 .procF
  950 JSR osw
  960 LDA #&A4:STA store1
  970 LDX #0:LDY #2:JSR tab
  980 LDX #messF MOD256
  990 LDY #messF DIV256
 1000 JSR display
 1010 .proc3
 1020 LDA #202:LDX #48:JSR osb
 1030 .proc4
 1040 LDX #16:LDY #2:JSR tab
 1050 LDX #blank MOD256
 1060 LDY #blank DIV256
 1070 JSR display:LDX #0:LDA #0
 1080 .proc5
 1090 STA &A00,X:INX
 1100 CPX #19:BNE proc5
 1110 .proc6
 1120 LDX #16:LDY #2:JSR tab:LDX #0
 1130 .proc7
 1140 JSR osr
 1150 CMP #13:BEQ proc8
 1160 CMP #127:BEQ proc4
 1170 CMP #27:BEQ proc8
 1180 STA &A00,X:JSR osa
```

```
1190 INX:CPX #13:BNE proc7
1200 .proc8
1210 CMP #27:BNE proc9
1220 LDA #12:JSRosa:JMP end
1230 .proc9
1240 LDA #13:STA &A00,X
1250 STX proclen:JMP readdisc
1260 .eof
1270 LDY #255
1280 .eof2
1290 INY:LDA mess5,Y:JSR osw
1300 CPY #23:BNE eof2
1310 JSR osr:JMP end
1320 .readdisc
1330 LDA #0:STA rem:STA quote
1340 LDY store2
1350 .loop1
1360 JSR osbg
1370 JSR osbg:STA &900
1380 CMP #&FF:BEQ eof
1390 JSR osbg:STA &901
1400 JSR osbg:STA &902:LDX #4
1410 .loop2
1420 JSR osbg
1430 DEX:STA &900,X:INX:INX
1440 CPX &902:BNE loop2
1450 DEX:LDA #13:STA &900,X
1460 LDA found:CMP #0:BNE det1
1470 .isitdef
1480 LDY #0
1490 .isitdef2
1500 INY:LDA &902,Y
1510 CMP #13:BEQ notdef
1520 CMP #32:BEQ isitdef2
1530 CMP #&DD:BEQ isdef
1540 .notdef
1550 JMP readdisc
1560 .isdef
1570 INY:LDA &902,Y
1580 CMP #32:BEQ isdef
1590 CMP store1:BNE notdef
1600 LDX #0
1610 .isdef2
1620 INY:LDA &902,Y
1630 CMP &A00,X:BNE wrong
1640 INX:LDA #13
1650 CMP &A00,X:BNE isdef2
1660 LDA &903,Y
1670 CMP #40:BEQmatch
1680 CMP #58:BEQmatch
1690 CMP #32:BEQmatch
1700 CMP #13:BEQmatch
1710 .wrong
```

```
1720 JMP readdisc
1730 .match
1740 LDA #1:STA found:LDY #255
1750 .window2
1760 INY:LDA win2,Y:JSR osw
1770 CPY #5:BNE window2
1780 JSR instruct:JSR osn:JSR instruct
1790 .det1
1800 LDY #255
1810 .window3
1820 INY:LDA win3,Y:JSR osw
1830 CPY #5:BNE window3
1840 LDX#0:LDY#0
1850 .det2
1860 INY:STY store4
1870 LDA &902,Y:STA store1
1880 CMP #13:BEQ det6
1890 CMP #34:BNE det3
1900 LDA #255:EOR quote:STA quote
1910 LDA store1
1920 .det3
1930 CLC:CMP #&80:BCS det7
1940 .det4
1950 STA &A00,X:INX
1960 .det5
1970 LDY store4:JMPdet2
1980 .det6
1990 JMP show
2000 .det7
2010 LDA #0:CMP rem:BEQ det8
2020 LDA store1:JMP det4
2030 .det8
2040 CMP quote:BEQ det9
2050 LDA store1:JMP det4
2060 .det9
2070 LDA store1:CMP #&8D:BEQ numtoken
2080 LDA #rom1:STA ptr
2090 LDA #rom2:STA ptr+1
2100 .tok3
2110 LDA store1:CMP #&F4:BNE det10
2120 LDA #1:STA rem
2130 .det10
2140 LDY#10
2150 .det11
2160 LDA (ptr),Y:CMP store1:BEQ det12
2170 CLC:LDA ptr:ADC #1:STA ptr
2180 LDA ptr+1:ADC #0:STA ptr+1
2190 JMP det11
2200 .det12
2210 DEY:LDA (ptr),Y:CLC
2220 CMP #&70:BCC det12
2230 CLC:CMP #and:BEQ det13:INY
2240 .det13
```

```
2250 INY:LDA (ptr),Y:CLC
2260 CMP #&7F:BCS det14
2270 STA &A00,X:INX
2280 JMP det13
2290 .det14
2300 JMP det5
2310 .numtoken
2320 LDA #63:STA &A00,X
2330 INX:STA &A00,X
2340 INX:INY:LDY store4
2350 INY:INY:INY:JMP det2
2360 .show
2370 STX linelen
2380 LDA #13:STA &A00,X
2390 LDX #0:LDY #0:JSR tab
2400 .show2
2410 LDA &A00,X:JSR osw
2420 INX:CPX linelen:BNE show2
2430 JSR osr:CMP #13:BEQ append
2440 .end
2450 JSR reset:RTS
2460 .append
2470 SEC:LDA top:SBC #1:STA ptr
2480 LDA top+1:SBC #0:STA ptr+1
2490 CLC:LDA top:ADC &902:STA top
2500 LDA top+1:ADC #0:STA top+1
2510 LDY #0:LDX &902
2520 .append2
2530 LDA &900,Y:STA (ptr),Y
2540 DEX:INY:CPX #0:BNE append2
2550 LDA#255:STA (ptr),Y
2560 LDA #1:STA appended
2570 JMP readdisc
2580 .tab
2590 LDA #31:JSR osw
2600 TXA:JSR osw:TYA:JSR osw
2610 RTS
2620 .instruct
2630 LDY #255
2640 .instruct2
2650 INY:LDA mess4,Y:JSR osw
2660 CPY #29:BNE instruct2
2670 RTS
2680 .reset
2690 LDA #0:LDY #0:JSR osf
2700 LDA #202:LDX #32:JSR osb
2710 LDA #26:JSR osw:LDA #12:JSR osw
2720 LDA appended:CMP #1:BEQ renumber
2730 RTS
2740 .renumber
2750 LDA #&99:LDX #0:LDY #82:JSR osb
2760 LDY #69:JSR osb:LDY #78:JSR osb
2770 LDY #46:JSR osb:LDY #13:JSR osb
```

```
2780 RTS
2790 .display
2800 STX ptr:STY ptr+1:LDY #255
2810 .display2
2820 INY:LDA (ptr),Y
2830 CMP #255:BEQ display3
2840 JSR osw:JMPdisplay2
2850 .display3
2860 RTS
2870 .intro
2880 EQUD&0A0A1F0C:EQUB&82:EQUS"APPEND
BY P.PIBWORTH"
2890 EQUW&0F1F:EQUW&830C:EQUS"PRESS f0"
2900 .fkey
2910 EQUS "KEY0 CALL &"+STR$~start+"|M"
:EQUB13
2920 .colour
2930 EQUW &830C:EQUW&8383:EQUW&8500:EQU
W&085
2940 EQUW &8282:EQUW &8282:EQUW &8282
2950 .win1
2960 EQUB28:EQUB1:EQUB3:EQUB39:EQUB1
2970 .win2
2980 EQUB28:EQUB1:EQUB6:EQUB39:EQUB5:EQ
UB12
2990 .win3
3000 EQUB28:EQUB1:EQUB13:EQUB39:EQUB8:E
QUB12
3010 .blank
3020 EQUS STRING$(18," "):EQUB255
3030 .mess1
3040 EQUS"Enter program name: ":EQUB255
3050 .mess2
3060 EQUB13:EQUS"NO SUCH PROGRAM !....T
AP A KEY":EQUB255
3070 .mess3
3080 EQUS"Procedure or Function (P/F) "
:EQUB255
3090 .messP
3100 EQUS"Enter name: PROC":EQUB255
3110 .messF
3120 EQUS"Enter name:   FN":EQUB255
3130 .mess4
3140 EQUB141:EQUS"RET to APPEND   SPACE
to EXIT "
3150 .mess5
3160 EQUW&850C:EQUS"End of File, Tap a
key"
3170 ]
3180 NEXT
3190 PRINT"TO SAVE, TYPE *Save GetDef "
;~code;" +";~(P%-code)
3200 END
```

Writing a Compiler (Part 3)

David Spencer continues his discussion of how to write a compiler by considering the code generator.

In the previous two workshops in this series we have developed a routine for the lexical analyser of our compiler, and presented a grammar that represents the language. This month we will show how to build a syntax analyser around this grammar, and introduce a technique to generate the compiled code.

THE SYNTAX ANALYSER

As explained last month, we are going to implement the syntax analyser using a method known as *recursive descent parsing*. This works by taking the input string (the expression in our case) one token at a time, and using it to identify the structure of the input. Our starting point is the grammar from last month, which is reproduced in figure 1 for reference.

expr := *term expr2*

expr2 := + *term expr2* | - *term expr2* | ε

term := *factor term2*

term2 := * *factor term2* | / *factor term2* | ε

Figure 1. The augmented grammar for simple numerical expressions

Before implementing the syntax analyser, we need to define two procedures to go with our lexical analyser. The first of these is called *match*, and its purpose is to match the current input token (as returned by the lexical analyser) with a specified token. For example:

```
PROCmatch(ASC"+")
```

will attempt to match the current token with the token for an addition operation (refer to the first part of this series for a list of allowable tokens). If the matching succeeds, then *match* calls the lexical analyser to read the next input token, and exits. On the other hand, if a match is not found, then our second new procedure, error, will be called. This simply prints an error message and terminates the program.

We can now get down to the bones of our syntax analyser. The basic idea is that for each non-terminal in the grammar (i.e. each line in figure 1), we will define a procedure that attempts to apply one of the productions for that non-terminal. It does this by working through the production, calling the appropriate procedure for each non-terminal that appears in it, and calling *match* to match terminals against the expected input. In the case where there are several productions that can be applied to a particular non-terminal, the next input token is used to determine the correct one. This is possible because as mentioned last month, all the productions for each non-terminal start with a different symbol, and hence the one to use can be deduced without having to read additional input tokens and possibly backtrack and try another if we come up against a dead-end. As an example of this, consider the procedure to parse the non-terminal *factor*. The next input token should either be that representing a number, or that for a left hand bracket, or the token for negation. On the basis of this, the appropriate production is chosen. If none of the possibilities match, then a syntax error exists and hence *error* is called to report it.

Figure 2 shows the implementation of the entire syntax analyser in pseudo code. You will notice that the main program inputs an expression as a string, calls the *expr* procedure to parse the expression and then matches the token 0. This is the token used to represent the end of the input, and is used to ensure that an otherwise valid expression isn't followed by complete garbage. For example, "1+2asd" should cause a syntax error. Incidentally, the statement to set *minusflag* to FALSE is needed to initialise the lexical analyser, as explained in the first part of this series.

A point to note is the way in which null productions (also called epsilon productions) are handled. Such productions occur for the non-terminals *expr2* and *term2*. Such productions are only applied as a last resort, when none of the other productions can be applied. If you look at the procedure for *expr2*, you will see that it first tries to apply the productions starting with the tokens '+' and '-'. If neither of these fits (in other words if the next token isn't a '+' or '-'), then the procedure exits without any further action. As *match* has not been called at all, the input token will still be the same as when the procedure was called, and hence we have achieved a result of matching a null token. A similar argument applies to the productions for *term2*. On the other hand, if the procedure for *factor* fails to match any of the possible tokens, then a syntax error is generated, as an epsilon production isn't a valid option for *factor*.

```
minusflag=FALSE
INPUT expression
PROCexpr
PROCmatch(0)
END

DEF PROCexpr
    PROCterm
    PROCexpr2

DEF PROCexpr2
    IF token='+'
           PROCmatch(+)
           PROCterm
           PROCexpr2
    IF token='-'
           PROCmatch(-)
           PROCterm
           PROCexpr2

DEF PROCterm
    PROCfactor
    PROCterm2

DEF PROCterm2
    IF token='*'
           PROCmatch(*)
           PROCfactor
           PROCterm2
    IF token='/'
           PROCmatch(/)
           PROCfactor
           PROCterm2

DEF PROCfactor
    IF token=number
           PROCmatch(number)
    IF token='('
           PROCmatch(()
           PROCexpr
           PROCmatch())
    IF token='_'
           PROCmatch(_)
           PROCfactor
    ELSE
           PROCerror
```

Figure 2. Pseudo code for syntax analyser

SYNTAX DIRECTED TRANSLATION

Having all but implemented the syntax analyser, you may be wondering what on earth its use is. True enough, it will detect any syntax errors in the input expression, but it will not perform the crucial function of actually compiling the expression into machine code. However, the syntax analyser does recognise the structure of the input, and therefore you might intuitively think that this information could be used to perform the translation. Indeed, this is the case, and the translation can be handled using a technique known as *syntax directed translation*. The idea is that each time a production is applied, the syntax analyser generates the necessary code to implement that part of the input.

Before applying this in practice, we need to consider the method by which we wish to evaluate an expression. We shall use a technique known as *Reverse Polish Notation (RPN)*, which is also called *Postfix Notation*. With this method, all the arithmetic is performed using a stack. Operands are pushed onto the stack, and operators pull the appropriate number of items off the stack, perform the operation, and push the result back onto the stack. Assuming that the stack is empty to start with, then after an expression has been evaluated, the stack holds a single value representing the result of that expression. As an example, the expression 1+2 is evaluated by pushing the values 1 and 2 onto the stack. The '+' operator then pulls these two values off the stack, performs the addition, and pushes the result back. More complex expressions can be evaluated by simply pushing values on to the stack and executing operators as appropriate.

The operands are always stacked in the order they are encountered, but the execution of the operators is controlled by their relative precedences. For example, consider the two expressions 1+2*3 (1+2)*3. The first is executed by stacking all three operands and then performing the multiplication followed by the addition. In the case of the second expression, the 1 and 2 are stacked, the addition performed, the 3 stacked and finally the multiplication performed. For more details of RPN refer to the article *Introducing Postscript (Part 2)* in BEEBUG Vol.8 No.5.

At first sight, it might seem to be a tricky task to convert our expression into an executable form. But remember that our grammar was designed to encapsulate the ideas of operator precedence, and it therefore has built into it the information necessary to control the conversion to RPN. What we have to do for each production in our grammar, is to add a *semantic rule* which determines the instructions that should be generated each time a production is applied.

At this point it is decision time again. Clearly, our final aim is to produce an executable 6502 machine code program. However, if we were to try and do this directly using *syntax directed translation*, we would be biting off more than we could chew. Instead, we will convert our expression into an assembly language program, rather than machine code. This removes from us the burden of having to know about things such as individual 6502 opcodes. Another major simplification we can make is to remove particular operations, such as the arithmetic functions themselves, and make them into subroutines. All the compiler needs to know about is how to call these routines, and not how the routines themselves are actually implemented.

We will extend the subroutine idea further to include a subroutine to push a value on to the stack, and another one to print out the final result of the expression. The stacking subroutine obviously has to be passed the value to be stacked, and as this will be a 32-bit number, it cannot be held in the 6502's registers. Instead, the compiler will store the value elsewhere in memory, and pass a pointer to it in the processor's X and Y registers (as used for passing addresses to OSWORD calls).

As an example, the assembly language program produced by the compiler for the expression (1+2)*3 looks something like:

```
      LDX #op1 MOD &100
      LDY #op1 DIV &100
      JSR stack
      LDX #op2 MOD &100
      LDY #op2 DIV &100
      JSR stack
      JSR add
      LDX #op3 MOD &100
      LDY #op3 DIV &100
      JSR stack
      JSR multiply
      JSR print
      RTS
.op1  EQUD 1
.op2  EQUD 2
.op3  EQUD 3
```

All that remains for us now is to define the semantic rules for our grammar. Figure 3 shows these rules for each production (in brackets). The notation used is similar to that for expressing the grammar itself, with the addition of the symbol '| |' which means concatenate. For example:

factor | | **mult** | | *term2*

means that the output produced should be that produced by the semantic rules for *factor*, followed by **mult**, followed by the output from *term2*. (We have used **value**, **add**, **sub**, **mult**, **div** and **neg** to represent the stacking and arithmetic calls simply to give a more succinct notation.) It is left as an exercise to verify that the semantic rules of figure 3 are indeed correct.

```
expr := term expr2 (term | | expr2)
expr2 := + term expr2 (term | | add | | expr2)
      | - term expr2 (term | | sub | | expr2)
      | ε
term := factor term2 (factor | | term2)
term2 := * factor term2 (factor | | mult | | term2)
      | / factor term2 (factor | | div | | term2)
      | ε
factor := number (value)
      | ( expr ) (expr)
      | - factor (factor | | neg)
```

Figure 3. The semantic rules for each grammar production

In next month's Workshop we will present the complete compiler program and move on to look at run-time libraries, linkers and loaders.

Psion Chess for the 512

Reviewed by Bernard Hill

Back in BEEBUG Vol.5 No.9 I reviewed Martin Bryant's Colossus 4 Chess, which has clearly become the best of the chess programs for the BBC micro. This position is still in no doubt - unless you own a 512 co-processor. The IBM PC range is surprisingly low in decent chess-playing programs, probably due to the Intel 8086 processor used: the 6502 and 68000 are more widely chosen for chess software. But the one exception to this is Psion Chess by Richard Lang (a very well-known name in computer chess, being responsible for the top end Mephisto machines as well as QL Chess for the Sinclair QL) - and it runs on the 512 co-processor.

The whole IBM clone range runs at an astonishing range of speeds, from the slowest IBM PC/XT to the top-end 386 machines offering an effective speed increase of over 50 times, and the 10 MHz 80186 processor in the 512 rates about 2.5 times faster than the basic XT when running Psion Chess. Psion Chess has a chess grade of 2085 in the USA on a PC/AT, so making allowances for speed and Chess Federation changes should give a British Chess Federation grade of 167 on the 512. The improvement over Colossus' 140-ish grade is verified by my tests when a 10-game match went Psion 9.5 : Colossus 0.5.

But what of the features? In many ways Psion Chess matches Colossus' features very closely. Missing are some which I never used on Colossus: legal moves displayed; invisible pieces; screen colour changes and self-mate problem mode. Colossus was rich in timing modes however, and I particularly miss the count-down and move-control modes because Psion merely has 11 normal 'average response time' levels of play, together with a 'same-as-you', 'infinite' and 8 'Mate-in-n' levels.

But loss of all these features is to my mind completely compensated not only by the much greater program strength but also the superb graphics. The partial mode 4 three-dimensional screen of Colossus is replaced with essentially a full mode 0 view of the pieces (design as in QL Chess if you've seen it). Very good, but the really impressive thing is the way the pieces SLIDE when moving (or even resetting the board) for complete realism in playing. Game replay just has to be seen to be believed!

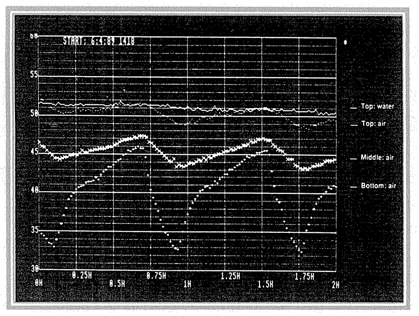
A press of the f2 key gives the 2D screen together with Help, optional Analysis Window and Move history. Pressing f1 gives access to 6 more help screens, and there is a facility to print the moves or the board on a printer. Key f4 even changes language to French or German! Other extra facilities include Next Best, to force the computer to think again, and Wait, which pauses the whole program.

There is just one sad thing however. Because of the way the program accesses the IBM sound chip, playing on the 512 processor leaves the game silent, so there is no audible beep when it moves, and you must keep an eye on the board because the 3D view has no supporting text to indicate whose move it is. In slow time levels I have to resort to enabling the move-print option so that I can be alerted by the printer when Psion moves, or else you could press B, which when it's the computer's move indicates (SLIDES!) its Best-so-far move and retracts it. When it's my move, B does nothing (but H for Hint would do the same for my pieces).

So let me heartily recommend Psion Chess to novice and expert alike. The former will revel in the graphics and the latter in the strength of the program. The program is available for around £20 mail-order, and if you ever get the chance to run the disc on a PC system with Hercules monitor, be prepared for a breathtaking graphic display.

Psion Chess is readily available through shops and dealers catering for the PC games market, and at a range of prices. For example, RSC Ltd, 75 Queens Road, Watford WD1 2QN, tel. (0923) 243301 can supply it for £21.28 inc. VAT. You may be able to find other suppliers who will charge marginally less.

The Best of BEEBUG



Applications II Disc

Share Investor
A program which assists decision making when buying and selling stocks and shares.

Real Time Clock
A real time digital alarm clock displayed for all BBC micros.

Running Four Temperatures
A program for calibrating and plotting up to four temperatures.

Crossword Editor
Design, edit and solve crosswords with this program.

Label Processor
Design, save and print labels at any size on an Epson compatible printer.

Monthly Desk Diary
A month-to-view calendar which can be used on-screen or printed out.

3D Landscapes
Create computer-generated three dimensional landscapes with this program.

Foreign Language Tester
Define foreign characters and test your knowledge of foreign languages.

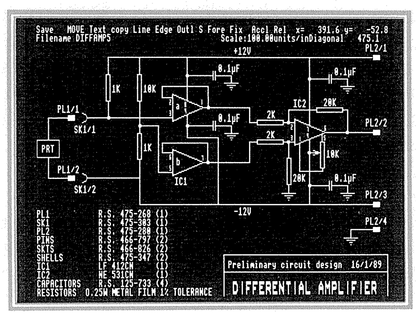
Julia Sets
Fascinating displays of Julia sets, the extensions of the Mandelbrot set.

Applications I Disc

✸ Business Graphics ✸ Video Cataloguer
✸ World by Night and Day ✸ Phone Book
✸ Page Designer ✸ Personalised Letter-Heads
✸ Mapping the British Isles ✸ Selective Breeding
✸ Appointments Diary ✸ The Earth from Space
✸ Personalised Address Book

Basic Booster ROM

Super Squeeze	A program compressor.
Partial Renumber	A very useful utility which renumbers a selected block of lines.
Program Lister	List any program direct from a file.
Resequencer	Rearrange the lines in a Basic program - line numbering is automatically adjusted.
Smart Renumber	Renumber a program so that procedures start at a particular line number.
Textload and Textsave	Save and load a Basic program as text.


Save MOVE Text copy Line Edge Outl S Fore Fix Accl Rel x= 391.6 y= -52.8
Filename DIFFAMP5 Scale:100.00units/inDiagonal 475.1
+12V
PL2/1
0.1µF
1K
10K
a
IC2
20K
PL1/1
SK1/1
2K
2K
PL2/2
PRT
1K
b
10K
PL1/2
SK1/2
IC1
20K
0.1µF
PL2/3
-12V
PL2/4

```
PL1         R.S. 475-268 (1)
SK1         R.S. 475-303 (1)
PL2         R.S. 475-280 (1)
PINS        R.S. 466-797 (2)
SKTS        R.S. 466-826 (2)
SHELLS      R.S. 475-347 (2)
IC1         LF 412CN     (1)
IC2         NE 531CN     (1)
CAPACITORS  R.S. 125-733 (4)
RESISTORS  0.25W METAL FILM 1% TOLERANCE
```

Preliminary circuit design 16/1/89

DIFFERENTIAL AMPLIFIER

ASTAAD

Enhanced ASTAAD CAD program with

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

General Utilities Disc

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

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

Please rush me my Best of BEEBUG disc at the members price of £5.75 (non-members price £15)
(ASTAAD disc - members price of £9.95 , non-members price £19.95)

Applications II *Code 1411A (80 track DFS)* □
Basic Booster ROM *Code 1403A* □
ASTAAD Code 1407A (80 track DFS) □
General Utilities Disc *Code 1405A (80 track DFS)* □
Applications I Disc *Code 1404A (80 track DFS)* □

Applications II *Code 1412A (3.5" ADFS)* □
Basic Booster Disc *Code 1402A* □
ASTAAD Code 1408A (3.5" ADFS) □
General Utilities Disc *Code 1413A (3.5" ADFS)* □
Applications I Disc *Code 1409A (3.5" ADFS)* □

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

Phone your order now on (0727) 40303

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

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

Amateur Research (Part 4)

John Belcher investigates megalithic physics.

CIVILISATION - ANCIENT OR MODERN?

Despite all the claims of natural selection, evolution, and education, the single feature that really separates early man from his modern counterpart is technology. It is technology that enables a civilization to emerge from obscurity and flourish, and paradoxically it is the same technology that hastens its decline and fall. Thus it is that civilizations come and go in cycles.

Primitive man had a great understanding of natural events, an intuitive instinct, so to speak. Modern man has lost this instinct, this faculty for understanding, but has replaced it with knowledge, with information, facts and figures. He no longer needs to think; all the thinking has been done for him. It is only when modern man is isolated from this pool of information - and misinformation - that his creative faculties are given a chance to emerge.

Isaac Newton laid the foundation of his research when he took to rural Lincolnshire to escape the London plague. Benjamin Franklin demonstrated his skill as an amateur researcher and inventor when confined to backwoods America, but lost his creative genius when his visit to Europe exposed him to the opinions of the scientific intelligentsia.

The general situation is best expressed in terms of the City Slicker in his flashy motor-car asking directions from the Country Bumpkin. Failing in this attempt, the said City Slicker remarks, "You don't know much, do you?". To which our bucolic friend replies, "No, maister, Oi doan't, but Oi bain't lorst, neither!" Which perhaps explains why civilizations become lost and die away.

Historians and archaeologists alike would shrink with horror at the mere suggestion that stone-age man, indeed pre-stone-age man, enjoyed a more advanced culture than we do today, conveniently overlooking the existence of those stone relics of a bygone age, the pyramids of Egypt and Central America, and the megalithic stone circles that abound in Europe. For us to look for signs of some super intelligence in mere heaps of stones, however, would appear to be a mission doomed to failure. Moreover, even if we were successful and could prove the existence of some long-gone super race, could we - from our limited intelligence - understand the significance of our discoveries?

PROFESSOR THOM'S MEGALITHIC YARD

In time to come, the field work of a professor of engineering may become a turning point in our understanding of past civilizations, and place our own modern culture in a somewhat better perspective.

Over a period of time, one Prof. Alexander Thom carried out extensive surveys of some 300 megalithic stone-circles. From an analysis of the resulting data he discovered evidence which suggested that the builders of these rings of stone were capable engineers in their own right. That is to say, they were obviously aware of the first of my three ubiquitous features in nature, the constant PI. This, of course, should become common knowledge to anyone inventing, or re-inventing, the wheel. The builders of the stone circles, however, went much further than that. They appeared to be obsessed with proving(?) that the ratio of the circumference of a circle to its radius was exactly 3, rather than 3.14159.... This they achieved by using intricate forms of geometric construction which produced stone circles having an ovoid or egg-shape layout.

Moreover, once a proposed design was seen to be suitable, the ensuing construction of the stone circle was undertaken with extreme care, showing a degree of accuracy that would be acceptable even today.

The real breakthrough, however, came with the discovery of what Prof. Thom calls the megalithic yard, MY, where:

MY = 2.72-ft (1)

Popular commentators, in attempts to show their cleverness, have pointed out that there is a possible connection between MY = 2.72-ft and MY = EXP(1)-ft. More cautiously, Thom does not draw attention to this, beyond hinting at a 'hidden significance'.

The argument against this, however, is overwhelming. The exponential series is a discovery of comparatively recent times, arising in turn from the discovery of the binomial theorem. That said, the value of e = 2.7182818... etc is not a mathematical constant readily encountered in the normal course of events, unlike PI. Any attempt to prove the connection between the megalithic yard and the exponential series, therefore, would be to postulate the existence of a pre-stone-age civilization having a scientific and technological development far in advance of our own! Perish the thought!

What Thom did emphasize was that the megalithic yard was a measure of the radius of a stone circle. And it is at this point that you may be recalling your own efforts last month to analyse the force characteristics of a ring array of elements, during which you discovered that RE = EXP(1), using a somewhat rough and ready simulation!

FORCE CHARACTERISTICS OF THE RING ARRAY

This month we are able to undertake a more accurate simulation of the ring array of elements, and thus arrive at more accurate - and more convincing - results. Not forgetting that such an array also simulates one of Thom's stone circles, which is to say, a ring of stones!

Program BPROG4, although short, is fairly powerful, both simulating a '100-element' ring array, and analysing its force characteristics. You can, if you so wish, examine these characteristics over any range of distance that catches your fancy. But our immediate interest is in the internal force characteristic of the array. Run the program and enter the distance, d = 0.001, a value which is critical to our purpose, a smaller value introducing computer error, and a larger one introducing proximity error.

The values of the force characteristic at this distance are as follows:

```
    AE = 5.00000035E-2
    RE = 2.71828115
    FN = 6.76676802E-3
```

A nice feature of this result is that the zeros in the value of AE indicate the accuracy of the value of RE, i.e. RE = 2.718281 to seven significant figures. In other words, RE = EXP(1).

In one short computer program, therefore, the original argument against any independent derivation of e = 2.7182818... etc is dashed to the ground. One merely has to understand the secrets of a ring of stones!

THE YARD AS A UNIT OF MEASUREMENT

We are now leaving Prof. Thom, and are going into unknown territory! Why is the unit of length, the yard, so called? I'm going to suggest that it was originally used to measure the area of another type of 'yard', a churchyard, a farmyard, a courtyard, and so on. In other words, the megalithic yard is a measure of an enclosed area of space.

THE CUBIT AS A UNIT OF MEASUREMENT

Certain stone circles would appear to have their measurements based on the cubit, and the reason for this has not been discovered (c/f Stonehenge).

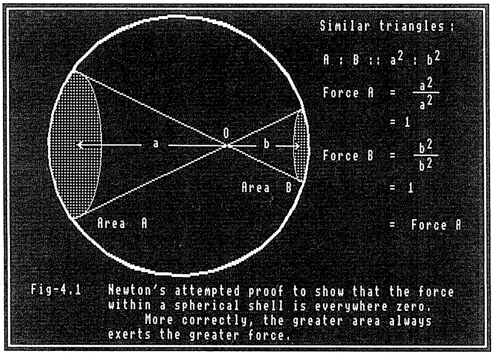
Two somewhat archaic words which describe an enclosed space are cubby-hole and cubicle, respectively. Applying that same convoluted recursive logic that you have seen me use in the past I therefore suggest that the megalithic cubit is a measure of an enclosed volume of space!

One dictionary definition of the cubit is 'a length in the range 17"-22"'. All we have to do now is to find the internal force characteristics of an array which will define its RE in terms of the cubit! Fat chance we have of doing that!

FORCE CHARACTERISTICS OF THE SHELL ARRAY

The nearest approach to an enclosed volume of space is a spherical shell. Some of you may recognise it as an equipotential surface. But this presents a difficulty.

Many years ago, Isaac Newton attempted to demonstrate geometrically that a uniform spherical shell of matter will exert no gravitational forces on objects inside it, provided that the gravitational force declines as the square of the distance. The situation is shown in Fig-4.1. Of course, everybody has agreed with Newton's hypothesis since that time, all conveniently overlooking the fact that the demonstration effectively involves the Three Body Problem! That said, today it is accepted that the force within an enclosed shell is everywhere zero!



Similar triangles :

$A : B :: a^2 : b^2$

$\text{Force A} = \frac{a^2}{a^2}$

$= 1$

$\text{Force B} = \frac{b^2}{b^2}$

$= 1$

$= \text{Force A}$

Fig-4.1 Newton's attempted proof to show that the force within a spherical shell is everywhere zero. More correctly, the greater area always exerts the greater force.

Fig.4.1

Program BPROG5 does for the shell array of elements what BPROG4 did for the ring array. The simulation of the shell is a poor one though seemingly accurate. What is required, of course, is for some resourceful reader out there to devise a simulation for such an array, based somewhat on the principle of dimples on golf-balls. But my 32-element array should suffice for our immediate purpose.

Run the program, and enter the same value of distance, d = 0.001. We now find the internal characteristics of the 32-element shell array to be:

AE = 3.33333388E-4
RE = 1.64872865
FN = 1.22626592E-4

This time the value of:

RE = 1.64872 = EXP(0.5)

a not insignificant result! Sad to say, the force within the shell is not zero, not unless we put the reference element at the centre of the shell.

So what then was the length of the megalithic cubit? Well if pre-stone-age man was as clever as my Master 128, the answer is glaringly obvious. The megalithic cubit, MC, has a length:

MC = EXP(0.5)-ft = 1.64872127-ft

which is some 19.78 inches!

MEGALITHIC INTELLIGENCE

The alternative definition of 'cubit' is 'the length of the forearm', from the point of the elbow to the tip of the middle finger. Thus the human frame possesses two of these units of length, the foot and the cubit. And I ask myself, is this coincidence, or is there a basis for such a relationship, and if the latter, which came first, the units of length or the body's proportions? If there is such a relationship, then we have a means of assessing stone-age man's intelligence.

In my own case, my height is 72.5", the length of my forearm is 20.1", and the length of my foot - assuming I could straighten out my big toe - is 11.6". So my cubit/foot ratio is 1.73:1, some 5% greater than the ideal 1.65:1. Does this make me a near-genius, despite being a duffer at intelligence tests? If so, then I'm all for the stone-age criterion of mind-power!

That said, apart from being a stone-age type, I am also what is known as a hyper-pituitary type, as evidenced by long feet and arms. This suggests that thousands of years ago, the hyper-pituitary type was the norm. To stone-age man, therefore, a hyper-pituitary type would be a giant, and legend has it that they abounded in those days!

How about modern man? Well, by comparison, he would be a hypo-pituitary type, showing a deficiency of the pituitary gland, the 'leader of the endocrine orchestra', in physical terms, a shortness of stature, a tendency to put on fat, and a lethargic dislike of exercise. He would have a noticeably negative attitude to life in emotional terms, leading to a weakness of character and a lack of harmony with the environment. Finally - with a natural deductive method of thinking giving way to an artificial and unnatural mode of logic - he would exhibit a diminished mental ability, characterized by bouts of paranoia and boredom!

Not such an unreasonable assessment, perhaps, but not much hope for Bacon's 'New Atlantis'!

RETROSPECT AND PROSPECT

Prof. Alexander Thom's discovery has given us much to think about concerning past and present civilizations, and in the realm of archaeo-physics - my term - this is only an introduction!

That concludes our series on Amateur Research. I hope you have found the articles both stimulating and interesting.

REFERENCES

(1) THOM, A., Rings and Menhirs - Geometry and Astronomy in the Neolithic Age, 'In Search of Ancient Astronomies', Edited by E.C. Krupp, Penguin Books (1984), pp39-76.

Listing 1

```
   10 REM Program .>BPROG4
   20 REM Version B1.0
   30 REM Author  J.C. Belcher
   40 REM BEEBUG  Jan/Feb 1990
   50 REM Program subject to copyright
   60 :
  100 MODE6
  110 PRINT"RING ARRAY OF ELEMENTS"
  120 N=100:N=N DIV4:theta=90/N
  130 REPEAT:AE=0:RE=1
  140 INPUT'"Distance = "d
  150 FOR angle=(theta/2) TO (90-theta/2
) STEP theta
  160 A=1:X=COS(RAD(angle))
  170 Y=SIN(RAD(angle))
  180 PROCcalc1(d,A,-X,Y)
  190 PROCcalc1(d,A,-X,-Y)
  200 PROCcalc1(d,A,X,Y)
  210 PROCcalc1(d,A,X,-Y)
  220 NEXT
  230 FOR angle=(theta/2) TO (90-theta/2
) STEP theta
  240 A=1:X=COS(RAD(angle))
  250 Y=SIN(RAD(angle))
  260 PROCcalc2(d,A,-X,Y)
  270 PROCcalc2(d,A,-X,-Y)
  280 PROCcalc2(d,A,X,Y)
  290 PROCcalc2(d,A,X,-Y)
  300 NEXT
  310 PRINT"AE = "AE'"RE = "RE'"FN = "AE
/RE^2
  320 UNTIL FALSE
  330 END
  340 :
 1000 DEF PROCcalc1(d,A,X,Y)
 1010 x=d+X:R=SQR(x^2+Y^2)
 1020 a=A*(x/R):AE=AE+a
 1030 ENDPROC
 1040 :
 1050 DEF PROCcalc2(d,A,X,Y)
 1060 x=d+X:R=SQR(x^2+Y^2):a=A*(x/R)
 1070 R=R^(a/AE):RE=RE*R
 1080 ENDPROC
```

Listing 2

```
   10 REM Program .>BPROG5
   20 REM Version B1.0
   30 REM Author  J.C. Belcher
   40 REM BEEBUG  Jan/Feb 1990
   50 REM Program subject to copyright
   60 :
  100 MODE6
  102 PRINT"SHELL ARRAY OF ELEMENTS"
  104 REPEAT:A=1:AE=0:RE=1
  106 INPUT'"Distance  = "d
  110 FOR I%=1 TO 2:RESTORE 10000
  120 FOR J%=1 TO 4:READ A,X,Y,Z
  130 IF I%=1 THEN PROCcalc1(d,A,-X,Y,Z)
:PROCcalc1(d,A,-X,-Y,Z)
  140 IF I%=1 THEN PROCcalc1(d,A,X,Y,Z):
PROCcalc1(d,A,X,-Y,Z)
  150 IF I%=2 THEN PROCcalc2(d,A,-X,Y,Z)
:PROCcalc2(d,A,-X,-Y,Z)
  160 IF I%=2 THEN PROCcalc2(d,A,X,Y,Z):
PROCcalc2(d,A,X,-Y,Z)
  170 NEXT J%,I%
  172 PRINT"AE = "AE'"RE = "RE'"FN = "AE
/RE^2
  180 UNTIL FALSE
  182 END
  190 :
 1000 DEF PROCcalc1(d,A,X,Y,Z)
 1010 x=d+X:R=SQR(x^2+Y^2+Z^2)
 1020 a=A*(x/R):AE=AE+a
 1030 ENDPROC
 1040 :
 1050 DEF PROCcalc2(d,A,X,Y,Z)
 1060 x=d+X:R=SQR(x^2+Y^2+Z^2)
 1070 a=A*(x/R):R=R^(a/AE):RE=RE*R
 1080 ENDPROC
 1090 :
10000 DATA 0.036611652,0.853553391,0.368
406439,0.368406439
10010 DATA 0.036611652,0.368406439,0.368
406439,0.853553391
10020 DATA 0.01516504325,0.577350269,0.5
77350269,0.577350269
10030 DATA 0.036611652,0.368406439,0.853
553391,0.368406439
```

512 Forum

by Robin Burton

It's the start of a new decade so I'll first wish you all a happy new year. I also wonder what's in store in the world of computing, both for Acorn and in DOS.

CHANGING TIMES

Both the Beeb and DOS are products of the 80s and both will soon be 10 years old. BEEBUG was on the Acorn scene very early, (note the fact that we're almost into volume 9), and while some Acorn publications have come and gone BEEBUG not only flourishes but remains faithful to our trusty 6502 machines. In fact it seems likely BEEBUG will soon be the only magazine catering for the 8 bit BBC micro at all, let alone exclusively.

I know for certain that one 'BBC micro' publication will be wholly Archimedes based very shortly. The rest may not be so formal or so forthright about their intentions, but it's obvious from their contents that only lip-service is now being paid to to the trusty Beeb (I, at least, will continue to refuse to call the A3000 a BBC micro).

"What's he going on about?" you're wondering. Advertisements actually. Many of your letters are complimentary and encouraging about the Forum, but at the same time quite a few express concern that 512 Forum may not continue for much longer.

Let me put your mind at rest. Mike Williams, in 'Editor's Jottings' in the last issue, said BEEBUG magazine intends to continue so long as there's a demand. Let me add to that by saying there are no plans to end 512 Forum, and as far as I'm concerned it will continue as long as there's a BEEBUG magazine. Given that other magazines are catering less and less for our machine, I'd hope that the future is secure for the faithful for some time and that brings me to my second point: some of you can help.

Recent issues have, as usual, carried readers' hardware ads., most of it sold, presumably, as users move to the Archimedes. Of more concern to me, however, is that there seem to have been rather more 512s on offer lately too. Perhaps some of them were purchased in Acorn's 'too good to miss' sale a year ago and their owners failed to get to grips with DOS, but that can't account for all of them.

My point is, if you sell your hardware make sure you remind its new owner about BEEBUG and 512 Forum, and definitely do so if the sale isn't through BEEBUG's Personal Ads. It might seem obvious to you, but gone are the days when you could walk into a newsagent's and pick up any of several magazines containing interesting or helpful articles and programs. Bear in mind that Beebug is on subscription, so if new (to Acorn) users receive no guidance from those in the know they might soon wonder what they've let themselves in for these days.

Surely all 512 users remember what it's like to be without support and information, so if do you sell your hardware, do the purchaser (and the remaining Beeb and 512 users) a favour and spread the word.

DON'T DO IT!!

I'm indebted to Cliff Bowman of TCS (see 512 Forum in BEEBUG Vol. 8 No.7) for bringing my attention to the following. Until he phoned me I hadn't seen the article myself.

You may have noticed the appearance of that extremely rare creature, (previously thought extinct) a 512 item in another magazine. Dave Atherton of Dabs Press (who else?) included an item in his column recently about changing the size of DOS hard disc partitions. I trust that after reading the following you'll excuse this Forum being given over to hard disc owners, even if you aren't one of them.

Most ideas in Dave's column are supplied by readers, and I'm not being condescending when

I say that I sympathise with anyone who writes a specialised monthly column. It's utterly impossible to test or verify every idea or program submitted. There simply isn't time, even if the correct hardware and software combination can be produced, which of course is often impossible. I have the same difficulties with suggestions or queries about DOS software - there's just no way that I can lay my hands on every package, much less be familiar with them all even if I do have a copy. Some things have to be taken on trust.

Now to the important point. The procedure outlined is fairly involved and there's plenty of opportunity to get it wrong since it involves editing your hard disc at sector level. Assuming that you get it all right your problems aren't over. The method outlined might work for some combinations of sizes (of the hard disc total capacity and both the initial and resulting partitions), but for most you'll be heading for disaster. Unhappily it's also a time bomb, in some cases it will be sooner, in others later!

By the way, Cliff warns that you MUST NOT use 'CHKDSK' (and probably many other DOS disc utilities), or disaster might not be sooner or later, it could be immediate.

If you tried the exercise and are using a modified partition I suggest you read the rest of the Forum, then ensure that all your hard disc files, ADFS and DOS Plus, are safe before you do anything else. If you back up any more files check their contents first, then copy the files separately. Even if you have the tools, don't copy entire partitions or even entire directories, you'll see why shortly.

THE RISKS

Danger lurks in several areas. To understand, bear in mind that the DOS partition isn't just seen by DOS as a hard disc, it's also seen by ADFS as a file, and that's part, though not all of the problem. If you read the previous Forum about how DOS files are stored on disc you will I hope, excuse this repetition for any who might have missed it.

The first point is that the size of a DOS partition relies on two separate pieces of data, not just one as suggested in the article. This means that inconsistencies can arise where ADFS thinks its usable disc area is of one size, while DOS disagrees. Which way round this argument occurs depends on the total size of your Winchester, the original (standard) partition size and the new size produced by the editing exercise.

The second point is that in DOS, files become fragmented as mentioned a few issues ago. While a file written to an empty disc will occupy a contiguous area (i.e. sequential sectors and clusters), over time, as files are deleted or re-written with new lengths and new files are added, the physical areas of disc used by any file (i.e. the clusters) can become literally scattered all over the disc. Unlike DFS or ADFS, DOS doesn't delete a file and re-write it in one piece when it expands, even if the existing location is no longer big enough (which is why there's no 'COMPACT' command).

DOS uses the existing area, then adds extra (unused) clusters as required at the logical 'end' of the old file area, but from wherever it happens to physically find clusters on the disc. Conversely, when a file contracts or is deleted newly freed clusters are marked available, and may be used immediately by anything else. When looking for empty clusters, whether for a new directory, a new file, or to extend an existing one, DOS simply starts at the beginning of the disc area and uses extra clusters in the order it finds them.

You can envisage that, when a disc has been used for some time any file can consist of a collection of clusters, each physically separated from its logical neighbour. Although processed in a logical sequence when you access the file, the clusters needn't be in any kind of sequence, physical, logical or otherwise.

This happens on all DOS discs except RAM discs which are freshly created every time you use them anyway. Floppies are easy to sort out because they're of limited size. Although smaller than a Winchester, they're also much slower, so you tend to notice extended access times more. It's then time to re-organise the files by backing up (at file level, i.e. file by file). Do this to a completely empty disc, then copy the resulting disc back to an empty working disc. This ensures that all the files occupy physically (as well as logically) sequential clusters on disc. Of course you'll need to deal with directories one at a time.

Hard discs suffer the same problem, but as they're very much larger there are more files and there's more scope for fragmenting. Unfortunately it's also less noticeable because of the higher speed, so re-organising tends to be overlooked. Remember that the intermingling of logical sections of data isn't limited to files in one directory, it applies to everything on the disc, including the actual directories themselves.

Without getting too involved in details you can see that if DOS and ADFS disagree over precisely where their usable areas of disc start and finish an extremely serious problem is virtually unavoidable. Of course DOS and ADFS will disagree on their areas if the data on which they base decisions has been changed in any way inconsistently. The result can be that ADFS files overwrite areas which DOS thinks it owns, or vice-versa, and it doesn't only depend on which partition is extended.

I should warn you that if you amended your hard disc in the manner suggested in the Acorn User article and think it's OK, you might not discover problems immediately. You'll probably only become aware of it when EITHER of the two areas, DOS or ADFS, has been used for some time, in other words, when you've lots of data to lose. Bear in mind that neither the ADFS command '*FREE' nor the DOS command 'DIR' helps, because the free space given by both is the total, and the free space can be anywhere, like the files. Although both partitions may show lots of free space, it's quite possible that areas towards the end of the allocation have been used, and that's where disaster awaits.

When it happens there are two possible results. The first and least harmful is that you'll find that some DOS files actually contain data from ADFS, or that ADFS files contain data from DOS.

The second, and worse, possibility is that if the 'other' filing system has overwritten either a directory or, for DOS, any part of the free space data for the partition, whole directories, or even the whole partition, could be totally inaccessible. In this case the data is completely irrecoverable, except perhaps by editing and recovering sectors. Bearing in mind file fragmentation and that only text is readily identifiable, I'd rather look for a needle in a haystack!

As I said, which way round things happen depends on several factors, but unless the invading data is easily identifiable you'll probably never know. What turns disaster into catastrophe, especially if you're diligent about backups, is the fact that when you discover it you may also find that your backups are in the same state. If so the data in the affected areas is gone forever.

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

A SELF-HELP UTILITY (BEEBUG VOL.8 NO.4)

When searching for a string this useful program converts all alphabetic characters to upper case. If this is applied to other characters (such as digits) in the range 32 to 63, an incorrect code is generated, resulting in a failure to find the string specified (the example quoted of a search string of '*FX5' fails for this reason). The problem can be cured by changing and/or adding the following lines:

```
   450 LDA (&F2),Y:CMP #61:BMI noconv
   451 CMP #&7B:BPL noconv
   452 AND #&DF:.noconv CMP (&A8),Y
```

Thanks to Peter Coaker for this information.

THE GAME OF CRIBBAGE (BEEBUG VOL.8 NO.2)

In the unlikely event that a Jack cut gives a winning score, the wrong player is credited with the game. Incorporating an additional check at the end of line 1980 will compensate:

```
     ...:PRINT:PROCsc:IFsc%?go%<121 THEN
go%=ABS(box%-1)
```

A Postscript Dump for Mode 7

In his last article on the use of PostScript, Willem van Schaik describes a mode 7 screen dump.

INTRODUCTION

After the two previous programs that provide PostScript output from BBC Basic programs, we have met both the bit-mapped and the vectorised way of defining graphics. In this third and last article we will use a combination of both techniques to create PostScript screen dumps of mode 7 teletext screens.

Unlike the other modes, a mode 7 screen is wholly character oriented. As every regular BEEBUG reader will know, graphics are only possible in mode 7 by using control codes to switch between alphanumeric and graphic character sets. The program CharSet (see listing 1) will display both of these two character sets on screen. Because this is done in a hexadecimal numbered matrix of 16 rows by 16 columns, a much better insight into the structure of the character set is possible than with the tables given in the User Guide. Simply type in the program, save it and then run it. Listing 2 consists of the mode 7 screen dump program itself, MO7POST, and this is described in the rest of this article.

In mode 7, an instruction such as:

```
   PRINTTAB(x,y) CHR$(n)
```

will normally display a character corresponding to the byte in screen memory which contains the value n. However, for the ASCII codes &23, &5F and &60 (decimal 35, 95 and 96) this is not true. If we look at the graphics character set it will be clear that the representations of these values have changed places. But when we inspect what is stored in *screen memory - with:*

```
   PRINT ~(&7C00+40*y*x)
```

this irregularity has disappeared. This may not seem very important, but it can be the cause of a good many headaches when trying to create a mode 7 screen dump program which is 100% correct.

CHARACTER DEFINITIONS

Now let's get back to PostScript. When you want to print a screen where only ASCII codes are stored, and where a dedicated hardware chip generates the pixels, the first task is to make your own character definitions. Most mode 7 screen dumps evade this using the standard character set of the printer, but I have included in the program a dedicated character set that matches the characters on the screen as much as possible. These definitions are stored in DATA statements in a format that is suitable for the PostScript image command.

In the PSDUMP program (see the article on PostScript screen dumps in BEEBUG Vol.8 No.6) a bit-mapped image of the whole screen was generated. For a mode 7 screen dump, we must use another technique by generating a separate image for each character. A mode 7 character consists of 10 rows of 6 pixels. The postscript image command demands that a new row starts on a character boundary; therefore to represent 6 pixels we need 8 bits with the last two dummy. These 8 bits will be represented by two hexadecimal digits. Hence, for the 60 pixels making up one screen character we need 10x2, that is 20 characters in PostScript (no wonder PostScript files can get to be so large).

For the DATA statements in the program MO7POST a compression technique is used. To decompress the character definitions:

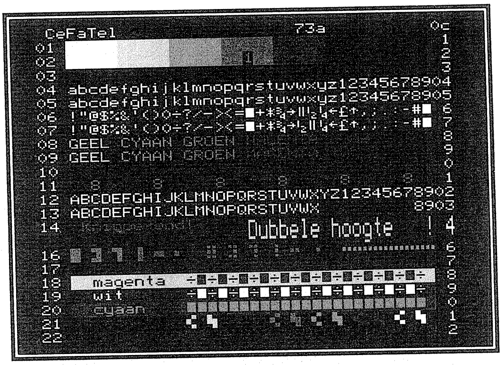
- take the first digit, which represents the number of leading zeros,

- then take the remaining (hex) digits,

- then add as many zeros as necessary to make a string of 20 characters.

The decompression is handled by the procedure PROCinitpixels, resulting in the array code$ that will contain the alphanumeric character definitions of ASCII codes 32 to 127 (stored in the array code$). Incidentally, these definitions can be used in any other program which needs a mode 7 character set.

The graphic character definitions do not have to be read from DATA statements, but can be calculated when needed. FNsix does this job on receipt of two parameters. The relevant ASCII value must be passed in the first parameter n%, while the second parameter c% is a boolean that determines if contiguous or separated graphics are desired.


CeFaTel
73a
0c
04 abcdefghijklmnopqrstuvwxyz12345678904
05 abcdefghijklmnopqrstuvwxyz12345678905
08 GEEL CYAAN GROEN
09 GEEL CYAAN GROEN
12 ABCDEFGHIJKLMNOPQRSTUVWXYZ12345678902
13 ABCDEFGHIJKLMNOPQRSTUVWX 8903
Dubbele hoogte
magenta
wit
cyaan

Sample laser printed teletext screen showing both text and graphics characters

The biggest difficulty is caused by the "hold graphics" feature. This attribute ensures that a subsequent control character does not appear as a space, but as a repeat of the last graphics character displayed (to the left). Thus, in PROCprint, the last printed graphics character is stored in im$ each time.

IMAGEMASK

In the PSDUMP program we used the image PostScript command to define a picture based on a bit-map. The colour or grey scale of the pixel can be selected with 1, 2, 4 or 8 bits/pixels.. In this program we are using a similar command, which is called imagemask. With imagemask a figure 1 in the bit-map defines where a pixel will appear in the current colour previously set with setgray or setrgbcolor. This method is used here because it allows the possibility of putting colours on top of others already there. This is needed to implement coloured backgrounds.

Imagemask, as used here, prints only one character at a time at the PostScript graphics origin. Therefore we advance the origin one character to the right after each imagemask. After finishing a line we don't have to move left again, because we just restore the graphics state as it was saved at the beginning of the line. PROCnewline takes care of that. It also resets all the attributes such as colour or double-height.

POSTSCRIPT PROCEDURES

In PostScript files such as those generated with the program listed here, extensive use is made of just a small number of PostScript commands. To reduce the size of these files, it is normal PostScript programming to define abbreviations that can be used instead of the full command. An example is the line:

```
    /im {imagemask} def
```

Thereafter, we can just program "im" instead of the much longer "imagemask".

USING THE PROGRAM

The use of MO7POST is very similar to that of PSDUMP (as it should be of course). The main difference is that you must save the screen beforehand with:

```
    *SAVE <name> 7C00+400 0000 7C00
```

where name is the file name to be used. For further information you should refer to the instructions for using PSDump.

This is the last in a series of three on creating PostScript screen dumps. With these programs you can make screen dumps of all BBC micro graphics modes, and mode 7. And even if you haven't got your own PostScript printer (they are very pricey), it should not be too difficult to transfer a PostScript file created on a BBC micro to any other system supporting a PostScript printer.

Listing 1

```
   10 REM Program CharSet (mode 7)
   20 REM Version 1.1
   30 REM Author  Willem van Schaik
   40 REM BEEBUG  Jan/Feb 1990
   50 REM Program subject to copyright
   60 :
  100 MODE 7:VDU23,1,0;0;0;0;
  110 ON ERROR MODE 3:REPORT:PRINT " at
line ";ERL:END
  120 M%=0
  130 PRINTTAB(0,1) STRING$(39,"-")
  140 PRINTTAB(0,23) STRING$(39,"-")
  150 FOR I%=0 TO 15
  160 PRINTTAB(I%*2+4,3) ;~I%
  170 PRINTTAB(0,I%+5) ;~I%
  180 PRINTTAB(37,I%+5) CHR$(135);~I%
  190 FOR J%=0 TO 15
  200 C%=16*I%+J%
  210 IF C%MOD128<32 THEN C%=32
  220 PRINTTAB(I%*2+4,J%+5) CHR$(C%)
  230 NEXT J%,I%
  240 PRINTTAB(13,24) "Press any key";
  250 :
  260 REPEAT
  270 IF M%=135 THEN M%=151 ELSE M%=135
  280 IF M%=135 THEN PRINTTAB(5,0) "Mode
 7 alphanumeric characters"
  290 IF M%=151 THEN PRINTTAB(5,0) "  Mo
de 7 graphic characters    "
  300 FOR I%=0 TO 15
  310 PRINTTAB(2,I%+5) CHR$(M%)
  320 NEXT I%
  330 A=GET
  340 UNTIL FALSE
  350 END
```

Listing 2

```
   10 REM Program MO7POST
   20 REM Version B 3.8
   30 REM Author  Willem van Schaik
   40 REM BEEBUG  Jan/Feb 1990
   50 REM Program subject to copyright
   60 :
  100 ON ERROR OSCLI("SPOOL"):REPORT:PRI
NT" at line ";ERL:END
  110 MODE 7
  120 base%=&7800:HIMEM=&7800
  130 DIM pix$(3),code$(95),rgb$(7)
  140 PROCinput:PROCinitpixels
  150 PROCinitrgb
  160 OSCLI("LOAD "+filename$+" "+STR$~(
base%))
  170 OSCLI("SPOOL "+postfile$)
  180 PROCprolog
  190 :
  200 PROCnewscreen
  210 FOR y%=0 TO 24
  220 PROCnewline(y%)
  230 FOR x%=0 TO 39
  240 IF x%<>0 THEN PROCnewchar
  250 cell%=FNcell(x%,y%)
  260 IF cell%<=&9F THEN PROCattribute(c
ell%)
  270 IF cell%>=&A0 THEN PROCprint(cell%
)
  280 NEXT x%,y%
  290 :
  300 PROCepilog
  310 OSCLI("SPOOL")
  320 END
  330 :
 1000 DEF FNcell(cx%,cy%)
 1010 cc%=?(base%+cx%+40*cy%)
 1020 IF cc%=&00 OR cc%=&7F OR cc%=&80 T
HEN cc%=&A0
 1030 IF cc%>&7F THEN =cc% ELSE =cc%+&80
 1040 :
 1050 DEF PROCinitpixels:LOCAL i%,c$
 1060 FOR i%=0 TO 95
 1070 READ c$
 1080 code$(i%)=LEFT$(STRING$(VAL(LEFT$(
c$,1)),"0")+MID$(c$,2)+STRING$(20,"0"),2
0)
 1090 NEXT
 1100 ENDPROC
 1110 :
 1120 DEF PROCinitrgb:LOCAL i%,swap$
 1130 FOR i%=0 TO 7
 1140 READ rgb$(i%)
 1150 NEXT
 1160 IF inv% THEN swap$=rgb$(0):rgb$(0)
=rgb$(7):rgb$(7)=swap$:REM swap black &
white
 1170 ENDPROC
 1180 :
 1190 DEF PROCinput:LOCAL sc%,ps%,rpl%
 1200 PRINT "---- BEEBUG MODE 7 PostScri
pt dump ---- "
 1210 REPEAT
 1220 INPUT "Screendump filename: " file
name$
 1230 sc%=OPENIN(filename$)
 1240 UNTIL sc%<>0
 1250 CLOSE# sc%
 1260 REPEAT
 1270 INPUT "Postscript filename: " post
file$
 1280 ps%=OPENIN(postfile$)
 1290 CLOSE# ps%
 1300 IF ps%<>0 THEN INPUT"Replace file
 <y/N>: " yn$:rpl%=FNyn(yn$) ELSE rpl%=F
ALSE
 1310 UNTIL ps%=0 OR rpl%
 1320 INPUT"Encapsulated  <y/N>: " yn$:e
ps%=FNyn(yn$)
 1330 INPUT'"Rotate dump   <y/N>: " yn$:
```

```
rot%=FNyn(yn$)
 1340 INPUT"Width dump in mm.   : " width
 1350 INPUT"Dump inversed <y/N>: " yn$:i
nv%=FNyn(yn$)
 1360 INPUT"Add border     <y/N>: " yn$:b
or%=FNyn(yn$)
 1370 PRINT'STRING$(39,"-"):PRINT
 1380 wix%=(width*9DIV127)*20
 1390 wiy%=((width*288/3175)DIV2)*25
 1400 ENDPROC
 1410 :
 1420 DEF FNyn(answer$)
 1430 IF INSTR(" Yy",answer$)>1 THEN =TR
UE ELSE =FALSE
 1440 :
 1450 DEF PROCprolog:LOCAL b%
 1460 PRINT"%!PS-Adobe-1.0"
 1470 PRINT"%%DocumentFonts:"
 1480 PRINT"%%Title: ";postfile$
 1490 PRINT"%%Creator: Acorn BBC MODE 7
screendump"
 1500 PRINT"%%Pages: ";
 1510 IF NOT eps% THEN PRINT"1" ELSE PRI
NT"0"
 1520 IF bor% THEN b%=1 ELSE b%=0
 1530 PRINT"%%BoundingBox: ";
 1540 IF rot% THEN PRINT ;300+wiy%+b%;"
"; 420-wix%-b%;" ";300-wiy%-b%;" ";420+w
ix%+b%
 1550 IF NOT rot% THEN PRINT ;300-wix%-b
%;" "; 420-wiy%-b%;" ";300+wix%+b%;" ";4
20+wiy%+b%
 1560 PRINT"%%EndComments"
 1570 PRINT"/cl {clip} def"
 1580 PRINT"/cp {closepath} def"
 1590 PRINT"/fi {fill} def"
 1600 PRINT"/gr {grestore} def"
 1610 PRINT"/gs {gsave} def"
 1620 PRINT"/im {imagemask} def"
 1630 PRINT"/lt {lineto} def"
 1640 PRINT"/mt {moveto} def"
 1650 PRINT"/np {newpath} def"
 1660 PRINT"/ro {rotate} def"
 1670 PRINT"/sc {scale} def"
 1680 PRINT"/sr {setrgbcolor} def"
 1690 PRINT"/tr {translate} def"
 1700 PRINT"/t {1 0 tr} def"
 1710 PRINT"/m {6 10 true [6 0 0 10 0 0]
} def"
 1720 PRINT"/c {np 0 0 mt 40 0 lt 40 1 l
t 0 1 lt cp cl} def"
 1730 PRINT"/b {np 0 -1 mt 41 -1 lt 41 2
 lt 0 2 lt cp fi} def"
 1740 PRINT"%%EndProlog"
 1750 PRINT"300 420 tr"
 1760 IF rot% THEN PRINT"90 ro"
 1770 IF bor% THEN PROCborder
 1780 PRINT ;wix%;" neg ";wiy%;" tr"
 1790 PRINT ;wix%/20;" ";wiy%/12.5;" sc"
 1800 PRINT rgb$(0);" sr"
```

```
 1810 PRINT"np 0 0 mt 40 0 lt 40 -25 lt
0 -25 lt cp fi"
 1820 PRINT"gs"
 1830 ENDPROC
 1840 :
 1850 DEF PROCborder
 1860 PRINT"0 0 0 sr np ";wix%+1;" neg "
;wiy%+1;" neg mt ";wix%+1;" ";wiy%+1;" n
eg lt ";
 1870 PRINT ;wix%+1;" ";wiy%+1;" lt ";wi
x%+1;" neg ";wiy%+1;" lt cp fi"
 1880 PRINT"1 1 1 sr np ";wix%;" neg ";w
iy%;" neg mt ";wix%;" ";wiy%;" neg lt ";
 1890 PRINT ;wix%;" ";wiy%;" lt ";wix%;"
 neg ";wiy%;" lt cp fi"
 1900 ENDPROC
 1910 :
 1920 DEF PROCepilog
 1930 PRINT"gr"
 1940 IF NOT eps% THEN PRINT"showpage"
 1950 PRINT"%%Trailer"
 1960 ENDPROC
 1970 :
 1980 DEF PROCnewscreen
 1990 double%=FALSE
 2000 double$="up"
 2010 ENDPROC
 2020 :
 2030 DEF PROCnewline(y%)
 2040 IF double% AND double$="up" THEN d
ouble%=FALSE:double$="low"
 2050 IF double% AND double$="low" THEN
double%=FALSE:double$=""
 2060 alfa%=TRUE:colour%=7:conceal%=FALS
E:contig%=TRUE:hold%=FALSE
 2070 im$=STRING$(20,"0")
 2080 PRINT"gr 0 -1 tr gs"
 2090 PRINT"% Line: ";y%
 2100 PRINT"c ";rgb$(7);" sr"
 2110 ENDPROC
 2120 :
 2130 DEF PROCnewchar
 2140 PRINT"t ";
 2150 ENDPROC
 2160 :
 2170 DEF PROCattribute(c%)
 2180 IF c%=&9C THEN PRINT rgb$(0);" sr
b ";rgb$(colour%);" sr"
 2190 IF c%=&9D THEN PRINT "b"
 2200 IF c%=&9E THEN hold%=TRUE
 2210 IF hold% AND NOT alfa% THEN PRINT
"m {<";im$;">} im"
 2220 IF c%>=&81 AND c%<=&87 THEN colour
%=c%-&80:alfa%=TRUE:PRINT rgb$(colour%);
" sr"
 2230 IF c%>=&91 AND c%<=&97 THEN colour
%=c%-&90:alfa%=FALSE:PRINT rgb$(colour%)
;" sr"
 2240 IF c%=&8C AND double% THEN PROCsin
gle
```

```
 2250 IF c%=&8D AND NOT double% THEN PRO
Cdouble
 2260 IF c%=&98 THEN conceal%=TRUE
 2270 IF c%=&99 THEN contig%=TRUE
 2280 IF c%=&9A THEN contig%=FALSE
 2290 IF c%=&9F THEN hold%=FALSE
 2300 ENDPROC
 2310 :
 2320 DEF PROCdouble
 2330 double%=TRUE
 2340 IF double$="" THEN double$="up"
 2350 IF double$="up" THEN PRINT "0 -1 t
r 1 2 sc"
 2360 IF double$="low" THEN PRINT "1 2 s
c"
 2370 ENDPROC
 2380 :
 2390 DEF PROCsingle
 2400 double%=FALSE
 2410 IF double$="up" THEN PRINT "1 0.5
sc 0 1 tr"
 2420 IF double$="low" THEN PRINT "1 0.5
 sc"
 2430 ENDPROC
 2440 :
 2450 DEF PROCprint(c%)
 2460 IF c%=&A0 AND NOT alfa% THEN im$=F
Nsix(0,contig%)
 2470 IF c%=&A0 THEN ENDPROC
 2480 IF conceal% THEN ENDPROC
 2490 IF NOT double% AND double$="low" T
HEN ENDPROC
 2500 PRINT "m {<";
 2510 IF c%>=&A1 AND c%<=&BF AND alfa% T
HEN PRINT code$(c%-&A0);
 2520 IF c%>=&A1 AND c%<=&BF AND NOT alf
a% THEN im$=FNsix(c%-&A0,contig%):PRINT
im$;
 2530 IF c%>=&C0 AND c%<=&DF THEN PRINT
code$(c%-&A0);
 2540 IF c%>=&E0 AND c%<=&FF AND alfa% T
HEN PRINT code$(c%-&A0);
 2550 IF c%>=&E0 AND c%<=&FF AND NOT alf
a% THEN im$=FNsix(c%-&C0,contig%):PRINT
im$;
 2560 PRINT">} im"
 2570 ENDPROC
 2580 :
 2590 DEF FNsix(n%,c%):LOCAL i%,l%
 2600 pix$(0)="00"
 2610 IF c% THEN pix$(1)="E0":pix$(2)="1
C":pix$(3)="FC" ELSE pix$(1)="60":pix$(2
)="0C":pix$(3)="6C"
 2620 IF c% THEN l%=3:c$="" ELSE l%=2:c$
="00"
 2630 FOR i%=1 TO l%
 2640 c$=c$+pix$(n%DIV16)
 2650 NEXT i%
 2660 IF c% THEN l%=4 ELSE l%=3:c$=c$+"0
0"
```

```
 2670 FOR i%=1 TO l%
 2680 c$=c$+pix$(n%MOD16DIV4)
 2690 NEXT i%
 2700 IF c% THEN l%=3 ELSE l%=2:c$=c$+"0
0"
 2710 FOR i%=1 TO l%
 2720 c$=c$+pix$(n%MOD4)
 2730 NEXT i%
 2740 =c$
 2750 :
10000 REM character defns. &20 to &7F
10010 DATA 1,41000101010101,9000282828,2
7C20207020202418,438541438505438,5C4C201
008646,43448542050502,900010101
10020 DATA 58102020201008,42010080808102
,41054381038541,610107C101,220101,9038,4
1,64020100804,41028444444281
10030 DATA 43810101010301,47C40201804443
8,43844041808047C,58087C48281808,4384404
0478407C,438444478402018
10040 DATA 42020201008047C, 438444438444
438,43008043C444438,61000001,22010100000
1,58102040201008,87C007C
10050 DATA 42010080408102,41000101008443
8, 438405C545C4438,444447C4444281,478444
478444478,43844040404438
10060 DATA 478444444444478,47C4040784040
7C,44040407840407C, 43C444C40404438,4444
4447C444444,43810101010l038
10070 DATA 438440404040404,4444850605048
44,47C40404040404,444444444546C44, 44444
4C54644444,438444444444438
10080 DATA 440404078444478,4344854444444
38,444485078444478,438440438404438,41010
101010107C, 438444444444444
10090 DATA 410102828444444,4285454544444
44,444442810284444,410101010284444,47C40
201008047C,610207C201
10100 DATA 01C100804584040404,610087C081
,6101054381,428287C287C2828,907C,43C443C
0438,47844444478404,43C4040403C
10110 DATA 43C4444443C0404,438407C4438,4
10101038101008,038043C44444438,444444444
78404,43810101030001,0201010101010010001
10120 DATA 42428302824202,43810101010103
,45454545468,44444444478,43844444438,040
407844444478,14043C4444443C
10130 DATA 4202020302C, 4780438403C,5810
101038101,43C44444444,41028284444,428545
44444,44428102844,038043C44444444
10140 DATA 47C2010087C, 141C140C24202020
2,428282828282828,141C140C641060106,6100
07C001,47C7C7C7C7C7C7C
10150 :
10200 REM parameters for PostScript setr
gbcolor command
10210 DATA 0 0 0,1 0 0,0 1 0,1 1 0,0 0 1
,1 0 1,0 1 1,1 1 1
10220 :
```

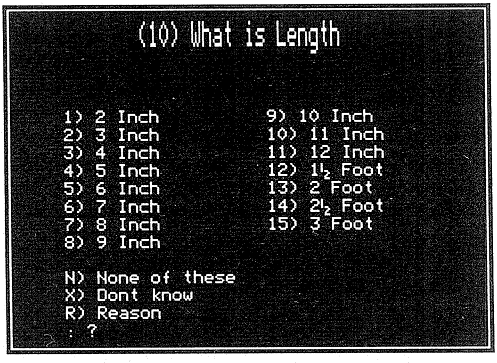
An Expert System for the BBC

Alan Wrigley reviews a new package from Pineapple Software.

Product	BESS
Supplier	Pineapple Software 39 Brownlee Gardens, Seven Kings, Ilford IG3 9NL. Tel. 01-599 1476
Price	£28.75 (inc. VAT)

If the article elsewhere in this issue on dichotomous keys has whetted your appetite for exploring the implementation of expert systems, the BBC micro Expert System Shell (BESS) may be of interest to you.

An expert system is a piece of software which essentially makes expert knowledge accessible to non-experts. Such a system will contain a knowledge base, compiled by an expert in the field, and an interface which enables others to diagnose, forecast or identify some key fact by answering a set of questions. For example, it would be possible, though not necessarily desirable, for a layman to diagnose an illness by using an expert system containing data compiled by a doctor.



```
        (10) What is Length

1) 2 Inch           9) 10 Inch
2) 3 Inch          10) 11 Inch
3) 4 Inch          11) 12 Inch
4) 5 Inch          12) 1½ Foot
5) 6 Inch          13) 2 Foot
6) 7 Inch          14) 2½ Foot
7) 8 Inch          15) 3 Foot
8) 9 Inch

N) None of these
X) Dont know
R) Reason
 : ?
```

Using the BESS Inference Engine

The term 'shell' which is applied to BESS signifies that the software contains only the interface and the capability to solve problems, but the knowledge must be provided by the user. Unlike the simple binary tree structure of a dichotomous key, BESS provides a framework for multiple choice questions, based on the information which is put in when the knowledge base is set up. It is possible, therefore, to build quite complex systems with such software, although in practice the memory limitations of the BBC will restrict the size of the database.

The package consists of a master disc (40 track DFS one side, 80 track DFS the other) and a 27-page A5 photocopied manual. The software allows you to create a work disc, but the master disc must always be inserted in order to start BESS. The disc contains a sample knowledge base for you to experiment with, which will identify common British mammals.

THE EDITOR

There are three main parts to BESS - the *Editor*, the *Compiler* and the *Inference Engine*. The editor is basically a simple word processor which is used to create a text file containing the data which makes up the knowledge base. A text file consists of a series of statements of the form:

```
IF <condition> THEN <fact>
```

built around the six keywords understood by BESS (IF, THEN, NOT, AND, OR, GOAL). For example:

```
IF SKY BLUE
THEN FORECAST SUNNY

IF BITS 8
AND NOT PROCESSOR 6512
THEN GOAL COMPUTER "MODEL B"
```

As you can see, each fact or condition must contain a variable (e.g. SKY) and a value which is assigned to it (e.g. BLUE). Prefixing a fact with the keyword GOAL tells the computer that this is to be a "goal variable", i.e. one which it is ultimately the purpose of the knowledge base to identify, such as ILLNESS in the hypothetical example quoted above, or ANIMAL in the sample database supplied. It is quite possible to have several goal variables in one knowledge base.

Clearly the most difficult task in setting up a knowledge base is compiling the data itself. This is no simple matter, since however expert your knowledge of a subject, translating it into a set of foolproof rules suitable for an expert system takes a great deal of time and patience. Ruling out all possible incorrect identifications is just as important as making the correct ones. For example, using the sample database supplied with the software, I discovered that a hedgehog is long and thin, with a bushy tail, and lives in trees, while a pig is 5 feet tall and has webbed feet!

The editor contained within BESS seemed rather quirky, and frankly I did not find it of much use. Whatever commands I used, the text simply scrolled straight to the end, which made it impossible to edit text of more than one screen in length. Eventually the editor stopped working altogether and the screen went haywire every time I tried to input text. Fortunately, text files can be imported from any word processor provided that the file is in ASCII format, so I quickly switched to using View to write my knowledge base.

THE COMPILER

Having written the text file, the next stage is to select the compiler from the BESS main menu. This takes the text file and converts the information into a form suitable for the inference engine to work on. At this stage any errors in the text are reported, for example if *there is no value following* a variable, or if a keyword is missing where it should have been expected. Unfortunately, when I tried it out, the compiler rejected a number of my text statements for reasons which did not seem very clear, even though, as far as I could see, they complied with the rules. When all the errors have been reported, the program returns you to the editor to amend the text, but because of the deficiencies in the editor already mentioned, I had to switch from BESS to View and back again each time. I found this extremely annoying, and eventually I gave up and removed the offending rules from the text in order to complete the compilation process.

THE INFERENCE ENGINE

Once the database has been compiled you can run the inference engine. The goal variables are presented as a menu and you must choose one to identify. The software will now try to complete the identification, working out which questions to ask as it goes along. For example, you may be asked "What is SKY?" and asked to choose from a list of all the values which have been assigned to SKY in the database, or to select "none of these" or "don't know". A further useful option at this stage is to ask the computer to give the reason why it has asked this question, whereupon it will display the rule which is being processed. The computer can be interrogated further to say why the rule needs to be solved, or how a previous fact in the rule has been solved. This is useful for locating logic errors in the knowledge base, or just to investigate the reasoning process.

Eventually the correct answer will be identified, assuming of course that the right information has been supplied in the first place.

PRINTER OUTPUT

An option on the main menu allows you to output the contents of a knowledge base to a printer. Alternatively, you can load the text file into a word processor and add headings etc. before printing. The printer can also be used when compiling, to list the errors which are reported, or to print out a list of the variables and their possible contents.

Continued on page 56

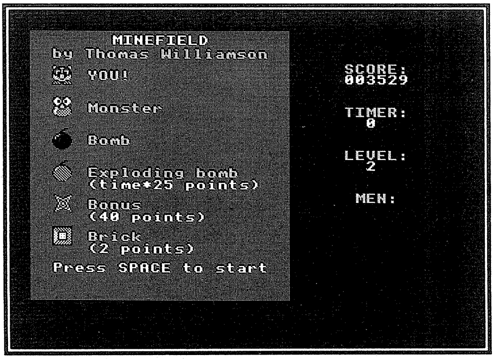
Minefield

Find your way through the minefield with Thomas Williamson's colourful and explosive game.

Minefield is an addictive game similar to the popular arcade game UXB. The rules are very simple. The game is played on a square grid of blocks. Scattered around the grid are unexploded mines, each of which will start flashing in turn. When a mine flashes, you have 10 seconds to reach and defuse it before it explodes. However, once you have passed over any block, it turns green to indicate that you may not pass that way again. Defusing all the mines allows you to progress to the next level, but as the levels rise, you start to meet monsters which make a bee-line for you.

The program is self-contained, so just type in the listing, save it, then run it. Use the Z, X, : and / keys to move left, right, up and down respectively. Pressing C or V will rotate the row you are on to the left or right respectively (i.e. each block in the row moves one space left or right and the block lost from the end is replaced at the start of the row). This is useful if you are surrounded by green squares and cannot otherwise move, but you cannot use it if there is a monster on the same row as yourself.

Be careful when typing in the listing, particularly with the DATA statements at the end which contain the data for the machine code routines used.

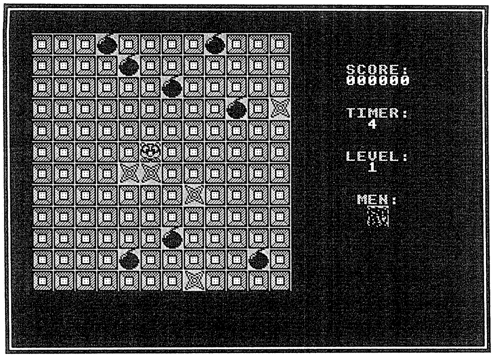


MINEFIELD
by Thomas Williamson
YOU!
Monster
Bomb
Exploding bomb
(time*25 points)
Bonus
(40 points)
Brick
(2 points)
Press SPACE to start

SCORE:
003529
TIMER:
0
LEVEL:
2
MEN:

Minefield help screen showing sprites used

On a model B, this program will only run with PAGE at &E00. See this month's Hints and Tips for a routine to do this automatically.



SCORE:
000000
TIMER:
4
LEVEL:
1
MEN:

The game in progress

```
   10 REM Program Minefield
   20 REM Version B1.0
   30 REM Author  Thomas Williamson
   40 REM BEEBUG  Jan/Feb 1990
   50 REM Program subject to copyright
   60 REM:
  100 ON ERROR GOTO140
  110 PROCcode
  120 MODE1:PROCset
  130 REPEAT PROCttl:VDU26:?&2B2=0:PROCs
et2
  140 REPEAT PROCset3:PROCscrn:TIME=0
  150 REPEAT ?&2B1=14:PROCman:IF M%>0 PR
OCmon
  160 F%=F%EOR1:IF scr?U%=2 PROCpri(U%,F
%)
  170 PRINTTAB(32,9);(1100-TIME)DIV100"
":IF TIME>1010 E%=1
  180 REPEAT UNTIL ?&2B1=0
  190 UNTIL N%=0 OR E%:IF N%=0 PROCnwl E
LSE VDU19;8;0;28,1,25,24,1:SOUND0,1,6,1:
CALLpow:VDU19;2;0;26:MN%=MN%-1:TIME=0:RE
PEAT UNTIL TIME>200
  200 UNTIL MN%<0
  210 PROChi:UNTIL 0
  220 :
  230 VDU4:IF ERR=17 GOTO40 ELSE MODE7:P
RINT'':REPORT:PRINT" at line ";ERL:END
  240 :
 1000 DEF PROCscrn
 1010 COLOUR131:CLS
 1020 FOR A%=0 TO &8F STEP4:A%!scr=0:NEX
```

```
T:scr?65=5:P%=65
 1030 IF M%>0 RESTORE1100:FOR A%=0 TO M%
-1:READ M%(A%):scr?M%(A%)=4:O%(A%)=0:NEX
T
 1040 FOR A%=1 TO N%:REPEAT B%=RND(&90)-
1:UNTIL scr?B%=0:scr?B%=3:NEXT:N%=N%+1:P
ROCnwb
 1050 FOR A%=0 TO 4:REPEAT B%=RND(&90)-1
:UNTIL scr?B%=0:scr?B%=1:NEXT:IF LV%>5 F
OR A%=0 TO LV%:REPEAT B%=RND(&90)-1:UNTI
L scr?B%=0:scr?B%=9:NEXT
 1060 COLOUR1:PRINTTAB(30,4)"SCORE:"'''
'TAB(30)"TIMER:"''''TAB(30)"LEVEL:"''''TA
B(31)"MEN:"
 1070 COLOUR2:PROCscr(0):PRINTTAB(32,9)"
10"''''TAB(32);LV%:IF MN%>0 VDU5:GCOL0,3
:B%=&5C90-MN%*16:A%=5:MOVE1056-MN%*32,47
6:FOR C%=1 TO MN%:CALLpri:B%=B%+32:VDU22
4,225,10,8,8,226,227,11:NEXT:VDU4
 1080 B%=stt:FOR C%=0 TO &8F:A%=C%?scr:C
ALLpri:B%=B%+32:IF (B%-stt)MOD640=&180 B
%=B%+&380
 1090 NEXT:ENDPROC
 1100 DATA0,&8F,&B,&84
 1110 :
 1120 DEF PROClft(R%)B%=stt+R%*&500:R%=R
%*12:S%=R%?scr:FOR C%=R% TO R%+10:C%?scr
=C%?(scr+1):A%=C%?scr:CALLpri:B%=B%+32:N
EXT:R%?(scr+11)=S%:A%=S%:CALLpri:P%=P%-1
:IF P%MOD12=11 OR P%<0 P%=P%+12
 1130 IF U%DIV12=R%DIV12 U%=U%-1:IF U%MO
D12=11 OR U%<0 U%=U%+12
 1140 ENDPROC
 1150 :
 1160 DEF PROCrgt(R%)B%=stt+&160+R%*&500
:R%=R%*12:S%=R%?(scr+11):FOR C%=R%+10 TO
 R% STEP-1:C%?(scr+1)=C%?scr:A%=C%?scr:C
ALLpri:B%=B%-32:NEXT:R%?scr=S%:A%=S%:CAL
Lpri:P%=P%+1:IF P%MOD12=0 P%=P%-12
 1170 IF U%DIV12=R%DIV12 U%=U%+1:IF U%MO
D12=0 U%=U%-12
 1180 ENDPROC
 1190 :
 1200 DEF PROCman
 1210 IF INKEY-90 PROCpse
 1220 O%=P%
 1230 IF INKEY-98 P%=P%-1:IF P%MOD12=11
OR P%<0 P%=P%+12
 1240 IF INKEY-67 P%=P%+1:IF P%MOD12=0 P
%=P%-12
 1250 P%=FNink:IF P%<>O% Q%=P%?scr:IF Q%
=9 P%=O%
 1260 IF P%=O% PROCman1:ENDPROC
 1270 O%?scr=9:P%?scr=5:PROCpri(O%,9):PR
OCpri(P%,5)
```

```
 1280 IF Q%=0 SOUND2,-5,0,1:PROCscr(2):E
NDPROC
 1290 IF Q%=1 SOUND1,4,200,1:PROCscr(40)
:ENDPROC
 1300 IF Q%=3 OR Q%=4 E%=1:ENDPROC
 1310 IF Q%=2 SOUND3,3,40,1:PROCscr(25*(
(1100-TIME)DIV100)):PROCnwb
 1320 ENDPROC
 1330 :
 1340 DEF PROCman1:X%=P%DIV12:IF M% Y%=1
:FOR A%=0 TO M%-1:Y%=Y%+Y%*(M%(A%)DIV12=
X%):NEXT:IF Y%=0 ENDPROC
 1350 IF INKEY-83 PROClft(X%) ELSE IF IN
KEY-100 PROCrgt(X%)
 1360 ENDPROC
 1370 :
 1380 DEF PROCpri(R%,A%)B%=stt+32*(R%MOD
12)+&500*(R%DIV12):CALLpri:ENDPROC
 1390 :
 1400 DEF PROCmon V%=(V%+1)MODW%:IF V%>0
 ENDPROC
 1410 FOR C%=0 TO M%-1:X%=M%(C%)DIV12:Y%
=M%(C%)MOD12:O%=M%(C%)
 1420 IF X%=P%DIV12 PROCh ELSE IF Y%=P%M
OD12 PROCv ELSE IF RND(2)=1PROCv ELSE PR
OCh
 1430 NEXT:SOUND3,2,236,2:ENDPROC
 1440 :
 1450 DEF PROCv
 1460 IF X%<P%DIV12 M%(C%)=M%(C%)+12 ELS
E M%(C%)=M%(C%)-12
 1470 Q%=scr?M%(C%):IF Q%=4 M%(C%)=O%:EN
DPROC
 1480 PROCpri(O%,O%(C%)):PROCpri(M%(C%),
4):scr?M%(C%)=4:scr?O%=O%(C%):O%(C%)=Q%:
IF Q%=5 E%=1
 1490 ENDPROC
 1500 :
 1510 DEF PROCh
 1520 IF Y%<P%MOD12 M%(C%)=M%(C%)+1 ELSE
 M%(C%)=M%(C%)-1
 1530 Q%=scr?M%(C%):IF Q%=4 M%(C%)=O%:EN
DPROC
 1540 PROCpri(O%,O%(C%)):PROCpri(M%(C%),
4):scr?M%(C%)=4:scr?O%=O%(C%):O%(C%)=Q%:
IF Q%=5 E%=1
 1550 ENDPROC
 1560 :
 1570 DEF PROCnwb:N%=N%-1:IF N%=0 ENDPRO
C
 1580 A%=0:IF N%>1 A%=RND(N%)-1
 1590 U%=-1:FOR B%=0 TO A%:REPEAT U%=U%+
1:UNTIL scr?U%=3 OR U%=&90:NEXT:IF U%=&9
0 A%=-1:REPEAT A%=A%+1:UNTIL O%(A%)=3:U%
=M%(A%):O%(A%)=2 ELSE scr?U%=2
```

```
 1600 TIME=0:ENDPROC
 1610 :
 1620 DEF PROCset
 1630 *FX9 1
 1640 *FX10 1
 1650 *FX210
 1660 VDU23;8202;0;0;0;12,19;2;0;19,1,5;
0;19,3;0;0:COLOUR131:CLS
 1670 pri=&900:pow=&947:scr=&C70:stt=&32
90
 1680 DIMH%(4),H$(4),M%(3),O%(3)
 1690 FOR A%=0 TO 4:H%(A%)=20000-A%*4000
:H$(A%)="T.W.":NEXT
 1700 ENDPROC
 1710 :
 1720 DEF PROCset2
 1730 SC%=0:LV%=1:MN%=3:M%=0:W%=11:N%=8
 1740 ENDPROC
 1750 :
 1760 DEF PROCset3
 1770 E%=0:V%=0:F%=3
 1780 ENDPROC
 1790 :
 1800 DEF PROCnwl
 1810 FOR A%=1 TO LV%*10:PROCscr(25):SOU
ND0,-12,5,1:SOUND0,0,0,0:NEXT:N%=8:LV%=L
V%+1:M%=1+M%MOD4:IF M%=1 AND LV%>2 MN%=M
N%+1:IF W%>2 W%=W%-3
 1820 ENDPROC
 1830 :
 1840 DEF PROCscr(S%)SC%=SC%+S%:PRINTTAB
(30,5)MID$("00000",LENSTR$SC%);SC%:ENDPR
OC
 1850 :
 1860 DEF PROCpse:T%=TIME:PRINTTAB(30,9)
"Paused":REPEAT IF INKEY-17 COLOUR1:PRIN
TTAB(28,21)"Sound  Off":COLOUR2:*FX210 1
 1870 IF INKEY-82 PRINTTAB(28,21)SPC10:*
FX210
 1880 UNTIL INKEY-106:PRINTTAB(30,9)SPC2
;(1100-T%)DIV100SPC3:TIME=T%:ENDPROC
 1890 :
 1900 DEF PROCttl
 1910 VDU28,1,25,24,1,17,128,12
 1920 PROCotl("MINEFIELD",272,976,2):PRO
Cotl("by Thomas Williamson",96,936,1)
 1930 REPEAT VDU28,1,25,24,4,12
 1940 PROCotl("HISCORES",288,875,1)
 1950 FOR A%=0 TO 4:PROCotl(STR$(A%+1)+"
)",96,800-A%*88,1):PROCotl(MID$("00000",
LENSTR$H%(A%))+STR$H%(A%)+"  "+H$(A%),19
2,800-A%*88,2):NEXT
 1960 PROCget:IF Y%<>1 PROCmonster
 1970 UNTIL Y%=1:ENDPROC
 1980 :
```

```
 1990 DEF PROCmonster:CLS:RESTORE2030:B%
=&3A30:Y%=875:FOR A%=5 TO 0 STEP-1:CALLp
ri:READ A$:PROCotl(A$,200,Y%,1):IF A%<3
READ A$:PROCotl(A$,200,Y%-36,2)
 2000 Y%=Y%-96:B%=B%+&780:NEXT
 2010 PROCget:ENDPROC
 2020 :
 2030 DATAYOU!,Monster,Bomb,Exploding bo
mb,(time*25 points),Bonus,(40 points),Br
ick,(2 points)
 2040 :
 2050 DEF PROCget:*FX15
 2060 PROCotl("Press SPACE to start",96,
300,2):I%=INKEY999:IF I%=32 Y%=1 ELSE Y%
=0
 2070 ENDPROC
 2080 :
 2090 DEF PROCotl(T$,X%,Y%,C%)VDU5:GCOL0
,3:MOVEX%-4,Y%:PRINTT$:MOVEX%,Y%+4:PRINT
T$:MOVEX%+4,Y%:PRINTT$:MOVEX%,Y%-4:PRINT
T$:GCOL0,C%:MOVEX%,Y%:PRINTT$:VDU4:ENDPR
OC
 2100 :
 2110 DEF PROChi IF SC%<H%(4) ENDPROC
 2120 *FX15
 2130 VDU28,1,25,24,1,12:PROCotl("CONGRA
TULATIONS!",160,960,2):PROCotl("You are
in the top 5",96,900,1):PROCotl("Please
enter your name:",48,856,1)
 2140 COLOUR3:I$="":REPEAT REPEAT G%=GET
:UNTIL G%>31 AND G%<128 OR G%=13:IF G%=1
27 AND I$>"" I$=LEFT$(I$,LENI$-1) ELSE I
F G%>13 AND LENI$<11 I$=I$+CHR$G%
 2150 SOUND1,-10,210,1:PRINTTAB(16-LENI$
,9)"  "I$:UNTIL G%=13
 2160 P%=-1:REPEAT P%=P%+1:UNTIL SC%>=H%
(P%):IF P%<4 FOR A%=3 TO P% STEP-1:H%(A%
+1)=H%(A%):H$(A%+1)=H$(A%):NEXT
 2170 H%(P%)=SC%:H$(P%)=I$:ENDPROC
 2180 :
 2190 DEF FNink:IF P%<>O% =P%
 2200 IF INKEY-105 P%=P%+12:IF P%>&8F P%
=P%-&90
 2210 IF INKEY-73 P%=P%-12:IF P%<0 P%=P%
+&90
 2220 =P%
 2230 DEF PROCcode
 2240 FOR A%=&B00 TO &BFF STEP4:!A%=0:NE
XT
 2250 DIML%(10),C%(10)
 2260 RESTORE2320:FOR A%=0TO10:READ L%(A
%),C%(A%):NEXT
 2270 G%=&8C0:FOR A%=0 TO 10:READ A$:C%=
0:IF A%=10 G%=&C00
 2280 FOR B%=1 TO LENA$ STEP2:?G%=EVAL("
&"+MID$(A$,B%,2)):C%=C%+?G%:G%=G%+1:NEXT
 2290 IF LENA$<>L%(A%) OR C%<>C%(A%) $&9
00=CHR$0+"~Error in DATA line "+STR$(223
0+A%*10)+CHR$0:CALL&900
 2300 NEXT:READA$:FORA%=0TO3:VDU23,224+A
%:FORB%=0TO7:VDU EVAL("&"+MID$(A$,A%*16+
B%*2+1,2)):NEXT:ENDPROC
 2310 :
 2320 DATA128,&DC2,230,&367D,128,&2888,1
28,&1C32,128,&1A31,128,&2552,128,&2015,1
28,&287C,160,&278F,128,&2058,64,&69D
 2330 DATA0100000000000028FF00FF78000000
0001140000FF00003C0000F43C0000000002FD01
02080CFF50FF00FD780000000003FF01FF010101
640000FD6400000000
 2340 DATA8570A9008571AD08048D3809186980
8D3E09AD09048D390969028D3F09A20606702671
CAD0F918A57069738570A57169098571A000B170
9DCDABC8B1709DEFCDC8E8E020D0EF60A200A004
BDF30A8570A91120EEFFA57029031869802 0EEFF
A92020EEFFA5704A4A857088D0E3E8E090D0D760
 2350 DATA00ADFFDADFADEBDABDBDDAFAADECDA
FF00F8FFDA5FF8AFFF5FA0FF50F8A0DAFF00E2FF
6B5FE2AFEF5FB1EF50E2A06BFF002AEE266E2AAE
262A2A26AE2AE626EE
 2360 DATA00006611751132323232117511660 0
0000EB00D788F6E6F9F9E6F688D700EB00007D00
BE11F676F9F976F611BE007D0000006688EA88C4
C4C4C488EA88660000
 2370 DATA00AF00DF00AF0057116732336511 75
0000AF115F33AF775FEBAFD75FAFAF5F7777AFCC
5F00AFCC5FBFAF5F5FAFBF5FCC00AE006E00AE00
4C00CC8888CC004C00
 2380 DATA00FF00FF00FF0077117723335711 57
0000FF11FF33FF77FFBFFF7FFFFFFFFF7777FFCC
FF00FFCCFFFFFFFFFFFFFFFFCC00EE00EE00EE00
CC00CC8888CC00CC00
 2390 DATA1174765674ADF8E8F8AD7446763333
008847E63BE205FD0AFD05F30AF605EDFF334DFC
8AF805F60AF605F80AFC157FEE0044CC4CC426E2
2AE226C44CCC888800
 2400 DATA00BF66DAF9ADFB5666335457DCAFFE
FF00FC00ED77C7DE7B67ADF27FEFBFFFEE007F00
7EDD6D7FDACCB7F8DFFEAFFEFF00AECC6AE2A6EA
4CCC88444C66AEEEEE
 2410 DATAFC3F003F000057D5F0EA0300A7AAAF
A60E00A7AA03563A00A7A7FF5B3900A7A7574DE9
03A7A5A7BAA50EA7AAA7BA553AA7EA79BA5539A7
7F7ABA55E9A7737ABA55E5A7737ABA29EAFC785A
BAFA3B0B79
 2420 DATAAABA5C3D6BE5AACE9FEE6C15FF779D
DEACA97A799EEEB0BA797A9E3EC0AF79AAAA0E00
B071AAAA0E00B0C6EAAB3700C02A7FFC390000AF
C36A0F0000F000FF00
 2430 DATAFF9F080080800000FFFB2101030301
0100000080C0800001010101030703 0101
```

An Expert System for the BBC (continued from page 52)

DOCUMENTATION

The manual supplied with BESS is fairly short and simply produced, but clearly written, and it covers all the aspects of using the software in a logical fashion. However, I found it unable to explain why my own attempts at compiling a knowledge base generated so many errors, as described above.

CONCLUSIONS

The implementation of an expert system on the BBC micro is an interesting idea, and a package of this kind could be extremely useful in the educational field, both for making specialised knowledge available to a wider audience, and for teaching the concept of expert systems. The creation of a knowledge base could form the basis of an interesting class project, for instance. Although BESS is perfectly adequate for this purpose (with some serious reservations about the editor), I felt that, as a whole, the software lacked that professional feel. Apart from the problems I described earlier, the screen displays and the sample database are peppered with spelling mistakes and dubious usage of the English language - one particular example which made me cringe was the phrase "not exists" which appears at the top of the screen when a file has not been created.

In conclusion, then, while accepting that BESS is an interesting concept that could be quite useful in schools and other situations where a straightforward expert system is required, such as counselling services, I would have expected to see a more professional look from a piece of software at this price.

POSTBAG

POSTBAG

UPDATED SPRITE EDITOR

I have recently been using the Sprite Editor (BEEBUG Vol.6 No.8) and wanted to change mode 1 colour 2 (yellow) to colour 4 (blue). At first I added a couple of lines to make the change permanent but after studying the program for a while I realised it would be a simple job to add a new function which allows you to assign any physical colour to a chosen logical colour. These lines are listed below:

```
  245 IF funckey=209 PROCcolours
 1741:
 1742 DEF PROCcolours
 1743 LOCAL c1,c2
 1744 c1=FNc:c2=FNc
 1745 VDU19,c1,c2;0;
 1746 ENDPROC
 1750:
 1751 DEF FNc
 1752 LOCAL k:*FX21
 1753 REPEAT:k=GET
 1754 UNTIL k>199 AND k<208 OR k>209 AND
k<218
 1755 IF k<208 =k-200 ELSE =k-202
```

Now, to change mode 1 yellow to blue, just press function keys f9, f2, f4 (f9 selects the colour change function, f2 yellow, f4 blue). I hope others find this useful.

Graham Crossley

READERS' REQUESTS

I would like to echo Mr King's suggestion/request (see Postbag Vol.8 No.6) for articles on sideways RAM. Some new articles on printer codes for text and graphics would be welcome, as would some information on power supply problems with many add-ons. With the flourishing secondhand market, many people are buying such systems and unwittingly encountering problems when trying to run too many devices off the Beeb's own low voltage power supply.

T.J.Sinclair

I wish to add my support to Mr King's letter regarding the use of sideways RAM. Writers of most articles on this subject seem to delight in confusing matters with their expertise in programming. Could we not have 'step-by-step' instructions for saving and loading SWR's, and all similar problems?

My ATPL board is fitted with 6264 RAM chips as recommended, but the manual is vague on their use. Could an everyday 'translation' be provided?

Thirdly, I have a small urge to update my model B for an Archimedes A3000, but I need to know more about the differences. Can I use my 20 or so ROMs, what about my ATPL board, my shadow RAM board, teletext adaptor and 40/80 track switchable drives? Incidentally, I note that Arcs are now supplied with a Philips CM8833 monitor. Have I missed something?

L.W.Batts

Mr.King's letter provoked a number of interesting replies all of which we shall be taking into account in planning the content of future magazines. Certainly the use of sideways RAM seems to be a high priority for many readers, and we will give some emphasis to that topic, though we have already published a good many articles already on this subject (see Vol.8 No.6 p10 for example).

We will also be pursuing the idea of an article to help would-be Archimedes owners. Many Basic programs will run, usually much faster than before, but there are also incompatibilities particular where ROMs and hardware add-ons are concerned. On the last point, the Philips monitor incorporates stereo loud speakers which can be directly connected to the stereo sound output of the Archimedes. The standard Acorn monitor, also made by Philips, does not have this facility, restricting sound output to the Arc's own quite tiny speakers. The Philips brand monitor is offered as an option by some dealers.

CAN DO BETTER

Andrew Roland's article *A Good Report* (see BEEBUG Vol.8 No.7) was very interesting, but I have developed a quite simple reporting method which suits my needs. If an error causes a crash during a 'write to disc' operation, I want my users to contact me immediately. If they don't they often compound the problem and corrupt all their discs. I therefore keep all disc writing lines to odd numbers, and all else to even numbers. My ON ERROR routine is then as follows:

```
   IF ERL MOD2=1 PRINT"FATAL ERROR - RING
GEOFFREY NOW" ELSE REPORT:PRINT" at line "
;ERL
```
etc. etc.

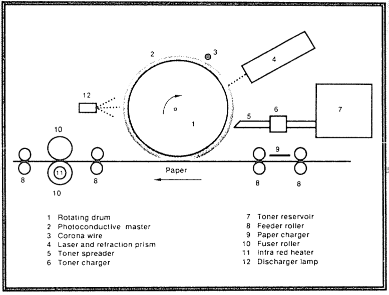
Geoffrey Shalet

It is only too easy to overlook the fact that pseudo variables like ERR and ERL can be processed numerically in a wide variety of ways. Mr.Shalet's routine is a simple but effective idea.

RISC USER

The Archimedes Magazine & Support Group

Risc User is enjoying the largest circulation of any magazine devoted solely to the Archimedes range of computers. Now in its third year of publication, it provides support to schools, colleges, universities, industry, government establishments and private individuals. Existing Beebug members, interested in the new range of Acorn micros, may either transfer their membership to the new magazine or extend their subscription to include both magazines.
A joint subscription will enable you to keep completely up-to-date with all innovations and the latest information from Acorn and other suppliers on the complete range of BBC micros. RISC User has a massive amount to offer to enthusiasts and professionals at all levels.



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

DESKTOP PUBLISHING WITH DRAW

How to use the Draw application for simple desktop publishing.

OUTLINE FONTS

A survey of outline fonts for the Archimedes.

THE ARM 3 RISC PROCESSOR

An in-depth look by the designer of this exciting development.

SCREEN COMPRESSOR

A module for the speedy and efficient compression and expansion of screen displays.

INTRODUCING LASER PRINTERS

A look at the workings of laser printers and Page Description Languages.

MASTERING THE WIMP

A major series for beginners to the WIMP programming environment. This month - Responding to Redraw requests.

A VOICE MODULE GENERATOR

Use the mouse to design waveforms and automatically generate voice modules from them.

A CHECK SAVE MODULE

A short utility which prevents the accidental overwriting of files.

UNDER THE LID

A major series explaining the hardware that makes up the Archimedes.

PROBING PASCAL

A brief overview of the Pascal language and the two compilers available for the Archimedes.

SCSI INTERFACES

What is a SCSI interface ? A comparative review of SCSI interfaces available for the Archimedes and A3000.

COLOURED TEXT PANELS

A module for creating three dimensional coloured slabs for text displays.

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

As a member of BEEBUG you may extend your subscription to include RISC User for only £8.10 (overseas see below).

Name:...

Memb No:...

Address: ....

....

....

SUBSCRIPTION DETAILS

Destination	Additional Cost
UK,BFPO &Ch Is	£ 8.10
Rest of Europe and Eire	£12.00
Middle East	£14.00
Americas and Africa	£15.00
Elsewhere	£17.00

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

I authorise you to debit my ACCESS/Visa/Connect account: /______/ ______/ ______/ ______/

Signed: .... *Card Expiry Date: /______/ ______/*

Send to: RISC User, 117 Hatfield Road, St Albans, Herts AL1 4JS, or telephone (0727) 40303, or FAX (0727) 60263

HINTS *and tips* HINTS *and tips* HINTS *and tips* HINTS *and tips* HINTS *and tips*

MOVEDOWN ROUTINE

Alan Wrigley

One of the commonest problems with the model B is caused by programs which require more memory to run than is normally available on a disc-based machine. It is worthwhile, therefore, to repeat a hint which has been published in the past but which is always useful.

Where a program requires every spare byte of memory, it can be made to move itself down in memory when run. Simply add the following lines:

```
1IF PAGE<&E01 THEN 10
2*KEY0 *T.|MFORA%=0TO(TOP-PAGE)STEP4:
A%!&E00=A%!PAGE:NEXT|MPAGE=&E00|MOLD|
MRUN|M
3*FX138,0,128
4END
```

This assumes that the first line of the program proper is line 10 - if not you should alter line 1 accordingly. The routine programs function key f0 to select the tape filing system, move the program down, reset PAGE, and run. Line 3 then simulates the pressing of f0, which causes it to be executed immediately.

FLAGGING EVENTS

E.N.Bramley

One way of flagging the occurrence of particular events within a program is to set up an array in which the elements are initially set to zero, and changed to one when the corresponding event has taken place. However, if large numbers of events are involved, the array may take up an unacceptably large amount of memory. Since a flag (TRUE or FALSE) really only requires one bit of memory, the following procedure may be useful, allowing eight separate flags per byte.

Taking M as any convenient location in memory not used by the program, e.g.:

```
DIM M 100
```

the contents of M to M+N/8 are first set to zero:

```
FOR I=0 TO N/8-1:?(M+I)=0:NEXT
```

where N is the number of flags required, and the amount of memory allocated (100 above) is at least equal to N/8. Then the flag for position X is tested and set, if not already set, by:

```
A=X DIV8:B=X MOD8
IF (M?A AND 2^B)=0 THEN M?A=2^B
```

Similarly the setting of flag X can be determined from:

```
A=X DIV8:B=X MOD8
Z=(M?A AND 2^B)
```

where the Z is zero or non-zero depending on the setting of the flag (zero meaning the flag is unset).

ANTI-LIST PROTECTION ON A MASTER

Andrew Ansell

It is possible to make a Basic program clear the screen and appear not to respond when any attempt is made to list it by adding the following line to the program:

```
REM |L <any text message you want to
see displayed on the screen> |U
```

This REM can also be added to the end of each line (with a few exceptions) by using colon followed by REM. This takes longer to enter, but will stop a hacker from listing lines either side of a single protected line.

The protected lines should be entered using the Master's Edit, and then saved using Shift-f4 followed by B. (for Basic). To enter | followed by a letter, hold down Ctrl and press the corresponding letter (this gives an inverted letter on the screen).

The |L will clear the screen and the |U will switch off the VDU drivers. It may be advisable to place |F at the end of the last instruction to switch them back on again. In response to LIST, the program is listed but nothing appears on the screen.

RS423 AS A PRINTER PORT

Bernard Hill

The following routine can be used to turn your RS423 (serial) interface into an output port. This can be used to connect one machine to another, for example, and can be useful where you have a printer connected to the second computer.

```
 10 ON ERROR GOTO 100
 20 *FX2,1
 30 CLS:PRINT"Waiting"
 40 REPEAT UNTIL ADVAL-2>0
 50 VDU11:PRINT"Printing"
 60 REPEAT
 70 VDU2,1,GET,3
 80 UNTIL ADVAL-2=0
 90 RUN
100 *FX2
110 IF ERR=17 THEN END
120 REPORT:PRINT" at line ";ERL
```

Personal Ads

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

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

Epson RX80FT+ printer with BBC cable £90 o.n.o. 16k EPROMs (27128 21v) £3 o.n.o. WANTED: Interword for the BBC B. Tel. (0332) 556381.

Master 512 in Viglen case and detached keyboard, with Z80 co-pro. Separate 40/80 twin disc plinth with own PSU. Mono monitor, joysticks, leads. Full set manuals, books, original software, complete set BEEBUG magazines, extras, all in excellent condition £550 o.n.o. (owner upgrading). Tel. 01-997 1218.

JUKI 5510 fast dot matrix printer V.G.C. Epson compatible £99 o.n.o. Arc games complete £8.50 each Startrader, Holed Out, Quazer, Clares Arcade, Orion, Minotaur, Terramex, Word Up Word Down, Pacmania. Tel. (08043) 2580.

Seven complete vols (vols 1-7) of BEEBUG magazine in binders in excellent condition, reasonable offers. Tel. (0623) 552530.

Magazines for sale; 100+ inc (45 Acorn User Dec '82-Sept '87); (35 Micro User Feb '85-Dec '87); (24 A&B Computing Mar '84-Dec '86). Offers? Tel. (0268) 694220.

Acorn Teletext Adaptor with PSU £40, Viewstore ROM £35, Viewspell ROM £20, Morley Master AA ROM board £30, Advanced Disc Investigator £18, 8 Master ROM cartridges various makes £7 each or 8 for £40, Voltmace 3B joystick unused £6, Colossus Chess £8, Superior SPEECH £5, AMX Super Art and latest mouse £30, Holed Out golf £7, Monitor swivel base £8, BEEBUG Teletext Editor £6, Advanced Reference Book for Master £12, Sideways RAM book and disc by Bruce Smith £10. Write to; 9 Loxwood Close, Little Common, Bexhill on Sea, East Sussex, TN39 4LX.

Master 128 in Viglen console, remote keyboard, 2x40/80T drives £480. BBC issue 7 £300. Aries 20 and B32 boards £85. 2 x 40/80 Technomatic drives in monitor plinth £150. Interword £30. Spellcheck £35. Microvitec Cub med. res. monitor £150. Philips 8833 med. res. monitor £185. Morley Electronics V2 EPROM blower £25. Ground Control UVIPAC EPROM eraser £20. Tel. (0846) 609473 office hours or (0846) 609284 after office hours.

M128 with Viglen 28Mb hard disc, Cumana CD800S dual 40/80T DS DD drives, 65C102 co-processor, almost new Philips green screen monitor, all in perfect condition, will sacrifice at £550 for quick sale. Tel. 01-405 9966.

WANTED URGENTLY: AMX mouse + Super Art for M128, write to; Ben Carr, Flat B4/3 Floor Greenville gardens, Hong Kong, Fax: 7956000 quote price including Airmail to H.K.

Master Compact + housed 3.5" disc drives + various ROMs £300. BBC B issue 7/Cumana disc drive + joysticks + ROMs + AMX mouse £275 Acorn RGB monitor as new £110, Breaking Commodore system. Offers? Tel. (04867) 80632.

WANTED: Handbook for Seikosha GP-80 printer. Tel. (0245) 352714.

M128 complete as new, Microvitec 452 Cub monitor, Cumana DS DD dual 80T disc drives with PSU, Star NL10 printer, Acorn Teletext adaptor, Marconi RB2 Trackerball, Twin BBC joysticks, Olivetti ope JP101 ink jet printer, all half price. Also volumes 1-7 of BEEBUG, many BEEBUG discs and a whole range of books, manuals leads and software on ROM and disc. Tel. for list (0622) 670714.

WANTED: 512k co-processor with DOS v2.1 also BEEBUG vol. 1 and 5. Tel. (0902) 783299.

80T 5.25" disc drive with leads for second drive £50, Acorn Teletext adaptor with ROM £40, Internal modem for Master with ROM, replacement battery pack, leads, software, books £50, Plotmate A4 plotter with ROM, pens, software £150 o.n.o. Wordwise Plus £15, Master cartridge £4, Two button mouse (Master 512) £15, Disc Spellchecker for Wordwise £5, 128k Solidisk Sideways RAM £15, Beta Base Utilities £3, Beta Base Accounts £3, CC Graphics ROM £15, Disc games £2 each. Tel. (0652) 650324.

BEEBUG magazine tapes; 41 assorted magazine tapes (vols. 2-8) including most of BEEBUG's best programs (ASTAAD 3, Filer, Business Graphics, ADFS Sector Editor etc) all originals, cased as new. Original cost over £100 will accept offers around £40 for the lot. Tel. (0279) 813463 after 6pm.

WANTED: Printwise on 40/80 disc for BBC B. Tel. (0823) 289378.

Music software for sale; complete music performing/printing package for BBC B, Master. This hardly used package includes EMR Miditrack Editor, Performer, Scorewriter, Utilities, Midi Interface £250 o.n.o. Upgrade of system to Archimedes is the reason for sale. Contact the Music dept. Tel. (0602) 417663.

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

Epson FX80 printer little used £65. Tel. (0483) 573688.

BBC Master Compact with Acorn AKFII colour monitor (boxed), dual disc drives, modem, ROMs, discs £475 o.n.o. Commodore C128, cassette player, comms pack, lots of cassettes and discs and other features £100 o.n.o. Archimedes RISC OS 440 with Acorn AKFII colour monitor (boxed). Hardly used. Some software £1550 o.n.o. Tel. (04867) 80632 eves and weekends.

BBC with Torch Z80 2nd processor and disc pack (runs standard BBC and CPM software). Twin D/S 80T disc drives, Microvitec 1451 med. res. monitor, Computer Village sideways RAM/ROM board, Star DP515 printer, Watford mouse, Voltmace joystick, META editor and assembler, AMX Stop Press, GDUMP, Watford Beebmon, Termulator. MCP Torch Perfect Writer, Speller, Filer, Calc. plus books and other software £550 o.n.o. Tel. (0258) 840038 eves.

A410/1 + extra 1Mb memory with 50Mb Hard Drive Watford Electronic Digitiser CC ROM podule, PC Emulator + lots of software £1,650. Tel. (0224) 535204 after 6pm.

BBC B issue 7 Acorn 1770 DFS and ADFS, Solidisk 256k board £200, View Professional £30 AMX Pagemaker £20 (0276) 23124.

BBC issue 7, Data recorder, Watford Speech Synthesiser, light pen, Joystick, Input magazines & £450 of games. Worth £1,000, sell for £350 or may split. Tel 051-480 3212 after 5pm.

WANTED: Quinkey five finger keyboard. Tel. 01-858 6086.

FX80 with additional tractor feed, BBC lead, handbook and spare ribbon £120. Care Quad Cartridge for Master £7, STARdataBASE ROM with handbook and utilities disc £19. Tel. 051-677 1518.

M128, Microvitec med. res. monitor in metal case, 40/80t DS DD drive, Master ref. manuals 1&2 plus misc. software on disc and in ROM cartridges £600 o.n.o. Tel. (0273) 557995 after 6pm.

BBC Master 512 (IBM compatible) and DOS+ problem solver; Sanyo colour monitor, Genie cartridge, Vine Micros' internal ROMboard, OverView package, Voltmace joystick, AMX mouse and Super Art, Viglen ROM cartridge, Interword, InterChart and Sciways ROM's, MOS+ ROM & BEEBUG Master ROM + over £200 worth of software/manuals. Worth over £1500, accept £500. Also Technomatic PD800P dual 5.25" disc drive (plinth mounted) with PSU, worth £215, accept £100. All in excellent condition. Tel. 01-455 9033 after 7pm and ask for flat 7/54.

Z80 second processor BBC/Master, v.g.c. including software £100, CC ROMs for sale; Interbase £30, Printmaster £10, Disc Doctor £10, Near Letter Quality £10, Graphics Tablet £10. Tel. (0604) 405184 eves.

M512 80186 co-processor, with DOS+ v2.1. Includes mouse, manual and all GEM software as standard £115, Dabs Press Master 512 User Guide and program disc £10, Dabs Press Shareware Collection £15, BEEBUG C + Stand Alone Generator £30, Technomatic 3.5" double sided 80T disc drive without PSU £45. Tel. (0924) 826483 after 5pm.

512 board, mouse, Dabs User Guide, Utils disc, Shareware I £160, AMX Stop Press + cartridge £28, AMX Super Art (Master) £20, AMX 3D Zicon £12, Nidd Valley mouse and Chauffeur + Grafik £35, Dumpmaster II ROM £20, System Delta + manual £45, Viewstore + cartridge £35, View and Viewsheet Guides £5 each, Dabs View Guide + disc £10, Dabs MOS+ disc £7, Dabs Sidewriter disc £8, Windomatic for Viewsheet £8, Elite disc £7, Gemini Life and Business Organiser £10, BBC Assembler kit-2 books, 3 discs £10, BEEBUG members disc + vol 6 no 1 disc £5, Acorn User 1987 compilation disc + May 88 disc £5. Offers for 65+ BEEBUG's from issue 1? all prices negotiable. Tel. (045527) 4018 eves.

M128 with View, Philips monitor, Viglen/Teac 5.25" dual disc drive, switchable 40/80 with integral power supply, all hardly used. Also, CC Mega 3 ROM in Care Master Quad ROM cartridge, unused. Also, Pace Linnett 1200 Modem and Commstar II communications software and PSU, unused. All complete with manuals and cables. Tel. (0635) 47082.

WANTED: Multisync monitor in good condition. Tel. (0324) 558692 with price.

Archimedes 310, mono monitor, PC Emulator, external disc drive interface £650. Tel. 01-986 4442.

BBC B 32k issue 7, Data recorder, many games £175 o.n.o. Tel. (0993) 776066 after 3.30pm.

BEEBUG MEMBERSHIP

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

BEEBUG SUBSCRIPTION RATES

BEEBUG		BEEBUG & RISC USER
£16.90	1 year (10 issues) UK, BFPO, Ch.I	£25.00
£24.00	Rest of Europe & Eire	£36.00
£29.00	Middle East	£43.00
£31.00	Americas & Africa	£46.00
£34.00	Elsewhere	£51.00

BACK ISSUE PRICES (per issue)

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

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

POST AND PACKING

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

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

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

BEEBUG MAGAZINE is produced by BEEBUG Ltd.

Editor: Mike Williams
Assistant Editor: Kristina Lucas
Technical Editor: Alan Wrigley
Technical Assistant: Glynn Clements
Production Assistant: Sheila Stoneman
Advertising: Sarah Shrive
Managing Editor: Sheridan Williams

CONTRIBUTING TO BEEBUG PROGRAMS AND ARTICLES

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

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

In all communication, please quote your membership number.

BEEBUG Ltd (c) 1990
ISSN - 0263 - 7561
Printed by Newnorth-Burt Ltd (0234) 41111

Magazine Disc/Cassette

JAN/FEB 1990 DISC/CASSETTE CONTENTS

DICHOTOMOUS KEYS (Part 1) - A program for designing expert systems where the information follows a tree structure with 'dichotomous' branching.

MASTER DISPLAY ROM (Part 1) - A useful utility which allows you to configure your Master 128 with Shadow RAM permanently on, regardless of mode.

* DUAL COLUMN LISTINGS - A complete copy of the most useful program updated as described in this issue of BEEBUG.

EDIKIT (Part 2) - The second part of this ROM based utility adds the commands *FBASIC and *FPROCFN to your EdiKit.

USING THE ROM FILING SYSTEM - The first of two programs which facilitate the use of ROM filing system - this one assembles ROM image headers.

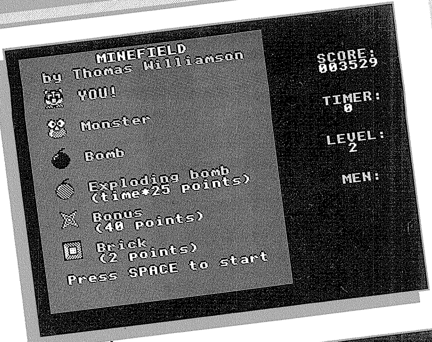
PROCEDURE/FUNCTION APPENDER - A very useful utility which locates procedures and functions on disc, appends them to a program in memory and renumbers the program.

AMATEUR RESEARCH (Part 4) - Two programs experimenting with the elements of ring array and shell array and proving (or disproving) megalithic intelligence.

A POSTSCRIPT DUMP FOR MODE 7 - The last program of this series allows you to dump a Mode 7 screen using a PostScript printer.

MINEFIELD - A colourful and addictive arcade-type game.

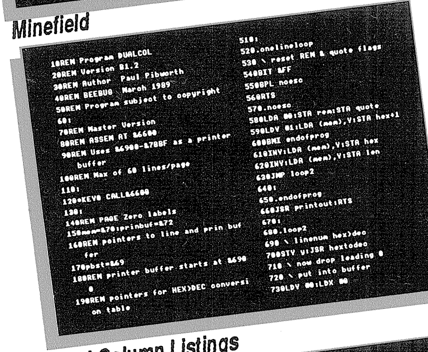
* - *Program on disc only*



```
MINEFIELD
by Thomas Williamson
YOU!
Monster
Bomb
Exploding bomb
(time*25 points)
Bonus
(40 points)
Brick
(2 points)
Press SPACE to start
```

```
SCORE:
003529
TIMER:
0
LEVEL:
2
MEN:
```

Minefield



```
10REM Program DUALCOL
20REM Version B1.2
30REM Author  Paul Pibworth
40REM BEEBUG  March 1989
50REM Program subject to copyright
60:
70REM Master Version
80REM ASSEM AT &6600
90REM Uses &6900-&7BBF as a printer
   buffer
100REM Max of 60 lines/page
110:
120*KEY0 CALL&6600
130:
140REM PAGE Zero labels
150mem=&70:prinbuf=&72
160REM pointers to line and prin buf
   fer
170pbst=&69
180REM printer buffer starts at &690
   0
190REM pointers for HEX>DEC conversi
   on table
```

```
510:
520.onelineloop
530 \ reset REM & quote flags
540BIT &FF
550BPL noesc
560RTS
570.noesc
580LDA #0:STA rem:STA quote
590LDY #1:LDA (mem),Y:STA hex+1
600BMI endofprog
610INY:LDA (mem),Y:STA hex
620INY:LDA (mem),Y:STA len
630JMP loop2
640:
650.endofprog
660JSR printout:RTS
670:
680.loop2
690 \ linenum hex>dec
700STY y:JSR hextodec
710 \ now drop leading 0
720 \ put into buffer
730LDY #0:LDX #0
```

Dual Column Listings

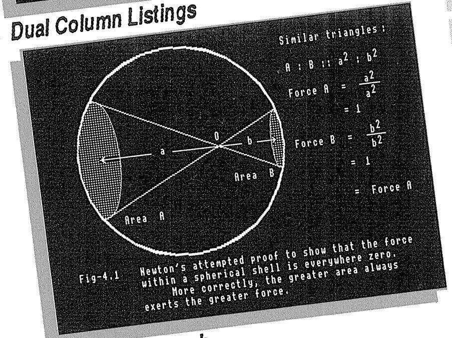


Fig-4.1 Newton's attempted proof to show that the force within a spherical shell is everywhere zero. More correctly, the greater area always exerts the greater force.

Amateur Research

ALL THIS FOR £3. 50 (CASSETTE), £4.75 (5" & 3.5" DISC) + 60P P&P (30P FOR EACH ADDITIONAL ITEM)

Back issues (5.25" disc since Vol.3 No.1, 3.5" disc since Vol.5 No.1, tapes since Vol.1 No.10) available at the same prices.

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

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

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

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

BEEBUG - THE ARCHIMEDES SPECIALIST

The NEW BBC Micro - The A3000

Ideal for home, education and business use, this powerful computer includes the multi-tasking Operating System - RISC OS; BBC Basic V and a BBC B emulator are supplied as standard. An optional PC emulator disc allows over 90% of DOS software to be used including Lotus, dBase III, MS Word and Wordperfect.

ACORN A3000 SERIES

0225G	A3000 Entry System	649.00
0256G	A3000 Colour System	838.00

ARCHIMEDES 400 SERIES

0260G	410/1 Entry System	1199.00
0261G	410/1 Colour system	1388.00
0262G	420/1 Entry System	1699.00
0264G	420/1 Colour System	1786.00
0275G	440/1 Entry System	2499.00
0276G	440/1 Colour System	2586.00

THESE PRICES EXCLUDE VAT

CHOICE OF MONITOR

Choose between the standard Acorn RGB monitor dressed in Acorn livery with an audio input or the Philips 8833 monitor with stereo sound and a video input. Both monitors offer similar picture quality.

FREE ON-SITE MAINTENANCE

All new 400 series and A3000 computers purchased through Beebug include one years On-Site Maintenance. Should your machine fail an engineer will call at your home or place of work within 24 hours to repair the machine.

9 Months 0% Finance

Under this scheme you may purchase an Archimedes A3000 or 400 Series computer (base or colour) by an initial deposit and 9 payments at monthly intervals. There is no charge to you for this service.

The table below shows some examples of the repayments on various machines.

Machine	Deposit	9 Payments
A3000	71.35	75.00
A3000 Colour	99.70	96.00
410/1	145.85	137.00
410/1 Colour	155.85	147.00
420/1	198.85	195.00
420/1 Colour	208.85	205.00
440/1	290.85	287.00
440/1 Colour	300.85	297.00

THESE PRICES INCLUDE VAT

If you wish to take advantage of this scheme please telephone us and we will send you further details. Finance over 12/24/36 months is also available.

FREE 40Mb HARD DISC DRIVE

Purchase a 410 with Acorn colour monitor and we will supply a free, high quality 40Mb Hard Disc with 28ms access time and autopark.The drive will be fitted and tested before dispatch. Please mention this offer when ordering.

For a limited period only

This offer is not available on 0% finance and is an alternative to the members offer listed below.

SPECIAL DISCOUNTS FOR TEACHERS

Please phone for further information.

SPECIAL PRICES FOR EDUCATION

Please contact us for all your requirements.

RISC USER & BEEBUG MEMBERS BUYING AN ARCHIMEDES SYSTEM WILL RECEIVE FREE:

A3000 - On Site Maintenance, Pacmania, printer lead, 10 x 3.5" discs & lockable disc box. Members using the 0% finance offer receive On-Site Maintenance only.

410/1 - On-Site Maintenance, and items to the value of £110 (Entry system) or £123 (Colour system). Members using the 0% finance offer receive free On-Site Maintenance only.

420/1 - On-Site Maintenance, and items to the value of £167 (Entry system) or £180 (Colour system). Members using the 0% finance offer receive On-Site Maintenance,10 x 3.5" discs.

440/1 - On-Site Maintenance and items to the value of £259 (Entry system) or £272 (Colour system). Members using 0% finance receive free On-Site Maintenance,10 discs & box plus either £45 (Entry system) or £65 (Colour system).

Please phone or write to receive a copy of our new FREE 54 page catalogue.

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