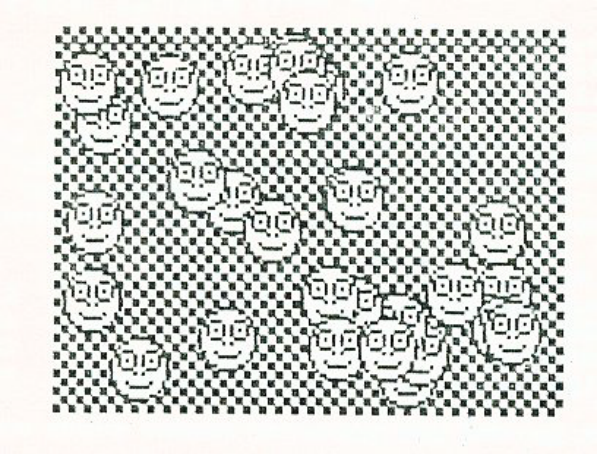
****

PUTTING SHAPES ON A BACKGROUND

Well, of course that's easy! Just PRINT AT or PLOT and put a UDG or a GET block on your background... The only problem is that a rectangular area of background gets over-written even if the shape you are trying to place is irregular; UDGs and GET blocks always have a PAPER "fringe" unless the shape design fibs them completely. You could use OVER 2, so that only the INK in the design is added to the screen. This avoids background being over-written by "fringe", but causes another problem: PAPER details inside your design are not put on the screen either. The background "shows through" your design. The solution used in many commercial games is to hold extra information for each shape about which areas are "fringe" and which are internal detail. This is usually done by having a stored "mask" shape the same size as the original graphic, with the bit pattern specifying "fringe" or "not fringe". Usually this is used to force a "hole" of the desired shape in the background, before the details from the original graphic are added. (This can now be done by a process analogous to OVER 2, since there is no longer any background to "show through".) The method allows irregular shapes to be added to detailed backgrounds.

The program below shows the same idea in Beta Basic; limes 10 to 30 prepare a "mask" graphic in a$, and lines 40 to 100 prepare the actual graphic in bb (a face - I'm sure you can do better!). Lines 110 to 1a0 provide a simple background. Originally I used OVER 2 and the mast: to force a "hole" to INK, then OVER 1 and the mask to reverse the "hole" to PAPER before placing the graphic in the background with OVER 1. Then I realised that you can get the same effect by reversing the INK and PAPER colours and leaving the hole as (white) INK and omitting the reversing step. In the example below, I added extra lines (160 and 210) to store the background and replace it, which might be desirable in some cases - omit them if you like. The PAUSE statements are also optional.



```
1     PAPER 0
      INK 7
10    CIRCLE 7,167,7
20    FILL 7,167
30    GET a$;0,175,2,2
40    CLS
50    CIRCLE 7,167,7
60    CIRCLE 4,169,2
      PLOT 4,169
70    CIRCLE 10,169,2
      PLOT 10,169
80    PLOT 6,166
      PLOT 8,166
90    PLOT 4,164
      DRAW 6,0,1
100   GET b$;0,175,2,2
110   KEYWORDS 0
120   PRINT STRING$(704,CHR$ 134)
130   KEYWORDS 1
140   FOR n=1 TO 50
150      LET x=RNDM(224),y=RNDM(144)+31
160      GET c$;x,y,4,4
170      PLOT CSIZE 16; OVER 2;x,y;a$ 180  PAUSE 0
190      PLOT CSIZE 16; OVER 1;x,y;b$ 200  PAUSE 0
210      PLOT x,y,c$
220   NEXT n
```

****

PROC VDU - handling control codes.

This contribution is from Alan Salmon of Bristol. He says: 'Beta Basic has most of the commands of BBC Basic, except VDU. So here is PROC VDU, which prints a string of characters from the codes given. So VDU 16,2,127 prints a red copyright symbol, equivalent to PRINT CHR$ 16; CHR$ 2; CHR$ 127. In addition, you can put strings after the VDU to display a whole string. Line 20 is obviously shorter than PRINT "123"; CHR$ 10; CHR$ 8; CHR$ 8; "456". Another use is with user-defined shapes, as in line 30.'

Most VDU commands I've seen have been harder to use than plain old PRINT - but the versatility of this version is an advantage. You need to be in a non-zero CSIZE to get the user-defined shapes feature to work - e.g. CSIZE 8.

```
10    vdu 16,2,127
20    vdu "123",10,8,8,"456"
30    vdu 0,128,128,128,128,1:8,128,128,255

1000  DEF PROC vdu DATA
1010     LOCAL a,a$,b$
1020     LET b$=-'
1030     DO UNTIL ITEM()=0
1040        IF ITEM()=2 THEN
               READ a
               LET b$=b$+CHR$ a
            ELSE
               READ a$
               LET b$=b$+a$
1050     LOOP
1060     PRINT b$
1070  END PROC
```

****

PROCs SHOW and SQUARE - scaling the display.

The next two procedures are from Jarmo Salonen (Tampere, Finland). You will need to keep the first PROC for use with the following one. In my previous career as a research scientist, I spent quite a lot of time working out what scale to use for graphs so that my data would just about fill a sheet of paper without falling off the edge. PROC show solves this kind of problem by allowing you to specify the minimum and maximum x and y values you want to plot, plus an optional "safety margin" to give a border round the plotting area. It then sets the special Beta Basic variables XOS, YOS, XRG and YRG accordingly.

```
10    show 0,99,0,49
20    PLOT 0,0
      DRAW 99,0
      DRAW 0,49
      DRAW -99,0
      DRAW 0,-49

100   DEF PROC show xmin,xmax,ymin,ymax,shr
110      LOCAL xrange,yrange
         DEFAULT shr=.01
         IF shr<.01 THEN LET shr=.01
120      LET xrange=xmax-xmin,yrange=ymax-ymin
130      LET xrg=(1+shr)*xrange,yrg=(1+shr)*yrange
140      LET xos=-xmin+shr/2*xrange,yos=-ymin+shr/2*yrange
150   END PROC
```

Sometimes you might have an additional requirement for plotting: that not only must all the data points fit on the screen, but that the horizontal and vertical scales should be equal. An example would be a floor plan or drawing in which square objects in the real world should look square in diagramatic form on the screen. PROC square takes care of this by fiddling with your supplied xmin, xmax, ymin and ymax values before passing them to PROC show. Try the lines below with and without the call to PROC square.

```
30    LET xmin=-8,xmax=2,ymin=1,ymax=11
40    square xmin,xmax,ymin,ymax
50    show xmin,xmax,ymin,ymax,.5
60    PLOT -8,1
      DRAW TO 2,1
      DRAW TO 2,11
      DRAW TO -8,11
      DRAW TO -8,1

200   DEF PROC square REF xmin, REF xmax., REF ymin, REF ymax
         LOCAL xrange,yrange,delta
         LET xrange=xmax-xmin,yrange=ymax-ymin
210      IF xrange>256/176*yrange THEN
            LET yrange=176/256*xrange,delta=(yrange-(ymax-ymin))
            /2,ymin=ymin-delta,ymax=ymax+delta
         ELSE
            LET xrange=256/176*yrange,delta= (xrange-(xmax-xmin))
            /2,xmin=xmin-delta,xmax=xmax+delta
220   END PROC
```

****

CATALOGUING TO A STRING

Beta Basic 4.0 has a function, CAT$, which returns the RAM disc catalogue, including type and length information, as a string. This makes it simple to write improved catalogues, or to check if a program name exists before saving or loading to RAM disc. Some people have asked why I didn't include something similar for Microdrives or disc drives. The answer is that experience has taught me that device-specific programming (a Microdrive version, a Discovery version, a Disciple version...) is difficult and time-consuming. On the other hand, if we are less ambitious and decide to forget about type information, file size and so on, it is possible to write a general-purpose routine which will divert the characters that normally appear on the screen during a CAT, and place them in a string instead. (Of course, you can always read them off the screen using SCREEN$, but this is slow and clumsy and requires you use the screen.) The ideas used here may still be useful-even without Beta Basic - you would have to replace the DPOKEs and DPEEKs with the longer ZX Basic form, and replace LENGTH (0,"a$") with a suitable address somewhere safe above RAMTOP. The characters placed at that address would have to be PEEKed after the CAT was finished.

The CAT_TO procedure makes use of a little-known channel that is not available to Basic, channel R. Normally, it is used to "print" characters entered by the user to an internal buffer during line entry, at a position indicated by the system variable K CUR. As each character is added, the buffer is expanded and K CUR is incremented. (This is how an input line is built up ready for syntax: checking and execution.) If we POKE the address of this routine into the channel P (LPRINT) area, any LPRINTed or LLISTed characters will be sent to K CUR instead of the printer. Line 1020 does just this, using the CHANS system variable to find the start of the channels, and then adding 15 to get the start of the P channel. (There are 3 preceding channels of 5 bytes each - K, S and R.) The ROM address of the R channel "add character" routine is 3973.

We don't actually want characters to go into a line entry buffer, so line 1030 pokes K CUR to point to a string instead. Now anything that would normally go to the printer, e.g. after CAT #3;1, will go into the string. The text of the string is expanded as needed, but the 2 bytes before the start of the string that store its length (used by the ROM) will not be updated; line 1050 does the required updating, by looking at how far K CUR has advanced from the start of the string. Finally, line 1060 returns the P channel to normal (for Beta Basic).

Now look at the demo lines starting at 10; they are a bit of a hatch-patch to give you some ideas. Because the procedure uses REF, the catalogue can be sent to any string - in this case, c$. If you simply PRINT c$, you will see the catalogue, but I have used a FOR-NEXT loop here to print each filename in turn using two columns; an array of names is set up at the same time. Microdrive and Discovery CATS have a 10-character cartridge/disc name, then 2 carriage returns, then a series of names padded with spaces to 10 characters, each followed by a carriage return, and finally a carriage return followed by the free space. Other storage systems will probably differ. Names in an array are more convenient in some ways - several such arrays could be JOINed together and SORTed, for example. The final demo lines show how to check if a file exists before doing a SAVE.

```
10   cat_ to c$
20   DIM r$(INT (LEN c$-12)/11,10)
30   FOR n=13 TO LEN c$-10 STEP 11
40      PRINT c$(n TO n+9),
50      LET r$(n/11)=c$(n TO n+9)
60   NEXT n
70   DIM n$(10)
80   LET n$="test"
90   IF INSTRING(13,c$,n$) THEN PRINT "file exists!"
        STOP
100  SAVE 1;n$

1000 DEF PROC cat-to REF a$
1010    LET a3=""
1020    DPOKE DPEEK(23631)+15,3973
1030    DPOKE 23643,LENGTH(0,"a$")
1040    CAT #3;1
1050    DPOKE LENGTH(0,"a$")-2,DPEEK(23643)-LENGTH(0,"a$")
1060    DPOKE DPEEK(23631)+15,64423
1070 END PROC
```

****

PROC Test_key

This is from Michael Field (Episkopi, BFPO 53). He says: "I originally wrote this procedure to test for the common Y/N response but ended up with something a bit better. The procedure waits for specific keys to be pressed then returns the key pressed to the main program in upper case in the string specified. Note: the LOOP: END PROC must be on a separate line."

```
10   Test_key "AB12",a$
20   PRINT a$

100  DEF PROC Test-key t$, REF q$
        LOCAL test
        DO
           GET q,$
           LET q$=SHIF$(1,q$)
           LET test=INSTRING(I,t$,q$)
           IF test THEN
                LET q$=t$(test)
           EXIT IF 1
110     LOOP
     END PROC
```

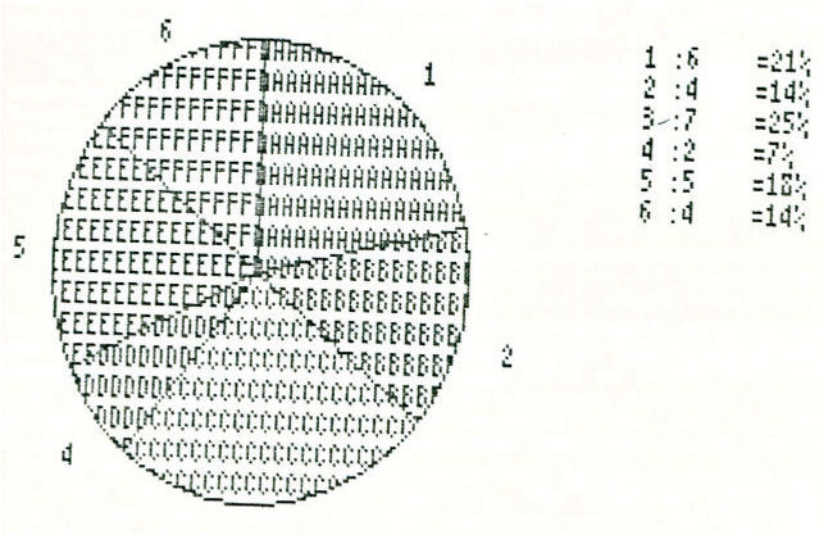
t$:= Key(s) to be tested for. Note: must be in upper case.
q$:= String that the key pressed is to be returned in.

The example tests for keys aAbB12 and returns the trey pressed in a$ in upper case.

****

PROC PIE - making pie charts.

This method of presenting information is very popular these days. The program was sent in by Alain Vezes (Albi, France); it makes it very easy to convert numbers into a display like the one illustrated. It was written with Beta Basic 4.0, but it will run under BB 3.0 and give a "skeleton" pie chart if you omit the FILL USING statement in line 170. (You could also omit line 70.) Line 70 sets up the required number of FILL patterns, using letters, but you could also use checks, stripes or other patterns, as explained in the BB 4.0 manual, provided you design enough unique types.




```
10    INPUT "number of items?";nb
50    CSIZE 4,8
60    DIM i(nb)
      DIM a(nb)
      DIM a$(nb,39)
70    FOR n=1 TO nb
         CLS
         LET c$=CHR$ (n+64)
         PRINT CSIZE 4,9;c$;c$;c$;c$`c$;c$;c$;c$
         GET a$(n);0,175,2,2
      NEXT n
      CLS
80    FOR n=1 TO nb
         INPUT "value ";(n);"? ";i(n)
      NEXT n
90    LET tot=0
100   FOR n=1 TO nb
         LET tot=tot+i(n)
      NEXT n
110   FOR n=1 TO nb
         LET a(n)=i(n)/tot*100
         PRINT TAB 49;n;TAB 51;":";i(n);TAB 57;"=";
         INT (a(n)+.5 );"%"
      NEXT n
120   pie 90,90,60,nb
```

```
140   DEF PROC pie x0,y0,r,nb
150     CIRCLE x0,y0,r
        PLOT x0,y0
        DRAW 0,r
160     LET a0=PI/2
170     FOR n=1 TO nb
          LET c$=STR$ n
          LET a=a0-2*PI*a(n)/100
          LET b=a0-PI*a(n)/100
          PLOT x0,y0
          DRAW TO x0+r*COS a,y0+r*SIN a
          FILL USING a$(n),x0+3*r/4*COS b,y0+3*r/4*SIN b
          PLOT x0+6*r/5*COS b,y0+6*r/5*SIN b;c$
          LET a0=a
        NEXT n
180   END PROC
```

****

PROC REMKILL - Removing REM statements.

This procedure gives a facility I never got round to including in BB's "toolkit" commands. It uses the system variables PROG (23635) and VARS (23627) which point to the start and end of a Basic program, to guide a search for REM lines. We cannot simply search for 'REM' (CHR$ 234) by itself, because the invisible form of numbers might contain one; e.g. LET z=234 is followed by an invisible CHR$ 234. Instead we look for an end-of-program-line marker (CHR3 13), don't worry about the next 3 bytes (which will be a line number and the least significant byte of a line length), then insist on a CHR$ 0 (which will be the most significant byte of a line length) before the REM. All this reduces any ambiguity and allows REMs at the start of a line (only) to be found reliably. The search will fail to find REM lines longer than 255 characters, but this shouldn't matter.

Once the REM has been found, the preceding line number can be PEEKed and DELETE can be used to remove the line.

```
10    DEF PROC remkill
        LET x=DPEEK(23635)
        DO
          LET x=INSTRING(x,MEMORY$()( TO DPEEK(23627)),
          CHR3 13+"###"+CHR$ 0+CHR$ 234)
        EXIT IF x=0
          LET lnum=PEEK (x+1)*256+PEEK (x+2)
          PRINT "deleting ";lnum
          DELETE Inum TO lnum
        LOOP
      END PROC
```

****

PRINTERS

Reader John Luby asks if anyone can recommend a printer interface to work with a Spectrum 128/Seikosha GP100A/Beta Basic combination. He doesn't like his current Dk'tronics interface; COPY doesn't work properly. (Incidentally, I think I've sorted out some problems John had with BB 4.0 and this interface, if anyone is interested.)

****

PROC PULL - a flexible pull-down menu procedure.

This is one of a set of procedures concerned with input checking, windows and elegant menu handling, sent in by Paul Field of Halsham, N. Humberside. There isn't space for them all, but I give Paul's favourite. PROC PULL uses a list of menu options passed via DATA. An area of screen is stored and a window big enough to take the number of options is generated, with a shadow frame round it. Any key except ENTER advances the selected option cursor; ENTER causes the original screen area to be restored. The selected option number is POKEd to 65535, from where it can be PEEKed. (I think Paul did it this way because you cannot mix, a DATA list and a REF parameter easily. However, he also suggested that "choice" could be made a global variable by removing it from the LOCAL list.) My line 10 does a LIST simply to provide a background to "pull down" onto. Usually, you would use a menu like this to control something like a graphics program, where you don't want to mess up the display. The global variable AA determines how far right the menu appears - try setting it to 2, 3 or 4 before calling the procedure.

```
10    LIST TO 110
20    pull top item,second,third,fourth

100   DEF PROC pull DATA
         LOCAL options,a$,q$,b$,p$,a,b,c,d,choice
         DEFAULT aa=1
110      DIM q$(10)
         LET a$="",options=0
         DO
            READ LINE q$
            LET q$(1)=SHIFT$(1,q$(1))
            LET a$=a$+q$+CHR$ 13,options=options+1
         LOOP UNTIL NOT ITEM()
         LET a=(aa-1)*56+8,b=options*8,c=167,d=72
120      GET p$,a-1,c,10,options+1;1
         WINDOW 2,a,c,d,b
         LET a=a-1,d=d+1
         PLOT a,c
         DRAW 0,-b
         DRAW d,0
         PLOT a+1,c-b-1
         DRAW d,0
         DRAW 0,b
         WINDOW 2
         BRIGHT 1
         CSIZE 7,8
         OVER 2
         CLS 2
         PRINT a$( TO LEN a$-1);
            AT 0,0; PAPER 2; INK 7; OVER 2;" "
130      LET choice=1
         DO
            GET b$
         EXIT IF b$=CHR$ 13
            ROLL 2
            LET choice=1+choice*(choice<options)
         LOOP
         WINDOW 0
         CLS 2
         PLOT OVER 0;a,c;p$
         POKE 65535,choice
      END PROC
```

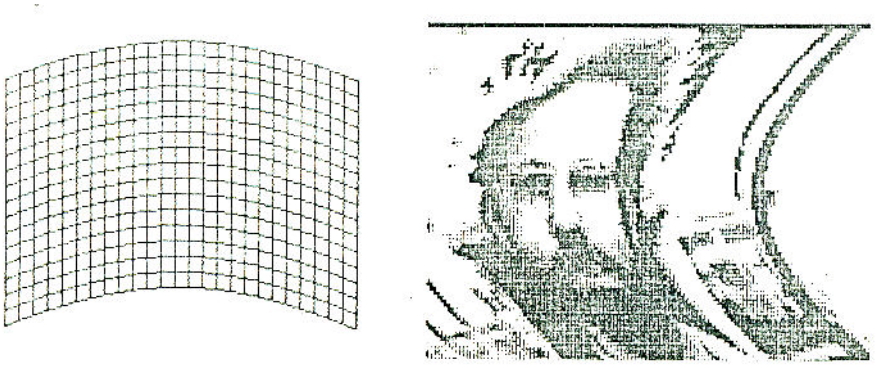
Charles Buszard also sent a worthy contribution on a similar theme. Some of his ideas could be useful in adapting PROC PULL. For example, you could use up and down cursors to give control of the selection cursor in both directions, and the cursor could be a bar of colour by matting line 120 finish with more spaces inside the quotes. Charles also noted that READ LINE has a problem - it allows menu options to be passed without quotes, but an option like 'save file' will turn into ' SAVE file' and cannot be entered without quotes. The solution is either to put all options in quotes and just use READ, or put quotes around problem options only, and strip them off after READ LINE if needed.

****

PROC DISTORT - stretching screen pictures.

Some time ago I saw a review of a flashy graphics system in Personal Computer World magazine. One of the facilities allowed the user to stretch the image in any direction. I wondered if this would be possible in Beta Basic, using ROLL or SCROLL, and set to work, with enthusiasm. It turns out to be fairly easy to achieve a left or right distortion, but harder in the vertical dimension, since ROLL will not handle a single column of pixels. However, it proved possible to use one character column as a working area, rolling a column of pixels into it, and then moving the character column up or down. It looks odd, but it works! I wrote a PROC GRID to give a screen picture for DISTORT to work on, but you could just do a LIST or load a screen picture (as I did for the illustration). The variable f in lime ?0 controls the degree of distortion. Of course, you can distort the same picture in several directions. The method of specifying left, right, up or down using DATA in the DEF PROC and a control words list in a string searched by INSTRING might be of interest. The words are arranged at regular locations (1,5,9 and 13) in the string so that the relevant ROLL code can be easily derived.

What would you want to use this PROC for? Well, I looked back at the PCW article and it didn't say... but having written it, I'll publish it anyway! (I get mail saying things like "I liked the item on ----, very clever. Like me, your contributors seem to like mucking about" but also, "I think some of your contributors are so carried away by their own cleverness that they forget what they are doing it for!" which may be the same message expressed in different words.)



```
10    grid
20    distort left
30    PAUSE 50
40    distort right
50    PAUSE 50
60    distort up
      PAUSE 50
      distort down

70    DEF PROC distort DATA
80      LOCAL a$,d,f,n
90      LET f=50
100     READ LINE a$
        LET a$=SHIFT$(2,a$)
110     LET d=(INSTRING(1,"leftdownup right",a$)+19)/4
120     IF d=5 OR d=8 THEN
          FOR n=0 TO 175
            ROLL d,SINE(n/175*PI)*f+1;0,n;32,1
          NEXT n
130     IF d=6 OR d=7 THEN
          FOR n=0 TO 247
            ROLL 5
            ROLL d,SINE(n/247*PI)*f+1;248,175;1,176
            ROLL 5,7;248,175;1,176
          NEXT n
          ROLL 5
140   END PROC

150   DEF PROC grid
160     LOCAL x,y
170     FOR x=0 TO 255 STEP 8
180       PLOT x,0
          DRAW 0,175
190     NEXT x
200     FOR y=0 TO 175 STEP 8
210       PLOT 0,y
          DRAW 255,0
220     NEXT y
230   END PROC
```

****

DISC NEWS

The Beta Disc Users Club that I mentioned in a previous issue (for Technology Research Ltd. disc interface users) has closed down due to lack of support. I suspect that many former users have connected their drives to the newer and more friendly Disciple or PLUS D interfaces. I have spent some time further adapting BB 3.0 and BB 4.0 for these interfaces. Beta Basic is no longer turned off by disc errors, LLISTing works normally, and disc commands can be used quite normally in a program. Not implemented are DEFAULT (device), slicer SAVES to disc, EOF and ON ERROR with disc errors. The required alterations have grown too big to publish here; if you want a Disciple/PLUS D version on tape, return your old tape and £ 2.00 and I'll exchange it.

****

BETA BASIC 4.0 - SORT bugs and a CSIZE bug.

H.N.S. Wijegoonawardena (Edgware, Middx.) reported that Beta Basic 4.0 did not SORT INVERSE properly on strings starting with CHR$ 0. Such strings sound a bit obscure, but will often be produced if you are coding numbers as strings using CHAR$. The problem resulted from an "improvement" I made to the sort routine to keep the speed as high as possible despite additions to cope with RAM disc sorts. John Luby (Duns, Berwickshire) reported other problems when sorting big RAM disc files (over 16K). These may not be obvious (depending on exactly how the strings cross 16K page boundaries) but can be severe (reset!). If you have BB 4.0 version 3 or less (PEEK 47272) and want to correct these SORT problems, then write for a POKE sheet.

If you have BB 4.0 version 1, and you enter:

```
CSIZE 4,9: FOR n=1 TO 30: PRINT n
```

you will find the value of N looks odd after the screen scrolls. To my initial surprise, this problem can be solved by entering the "patch" published in issue 7, page 3. (Note: Once you have used such a "patch" program, NEW, then MERGE Beta Basic's loader, and RUN to save the modified program. There is no need to include the Basic lines of the "patch" in your programs.)

****

MEMORY USAGE

Here's a little routine to give a variety of useful or interesting details about memory usage, using data from system variables. The smallest value for variables length you will see is 36, due to XOS, YOS, XRG and YRG, and 4 bytes used by the procedure. The size of BASIC's GOSUB/DO/PROC stack will be at least 3, due to PROC status's return address.

```
100  DEF PROC status
110     PRINT "Program length: ";DPEEK;(23627)-DPEEK:(23635)
120     PRINT "Variables length:";DPEEK(23641)-DPEEK(23627)-1
130     PRINT "Free memory:";MEM ()
140     PRINT "BASIC stack:";DPEEK(23730)-DPEEK(23613)-3
150     PRINT "RAMTOP:";DPEEK(23730)
160  END PROC
```

****

TEXT JUSTIFICATION

Ettrick Thomson (Aldeburgh) sent in a text justification procedure that improves on Mike Eida's contribution in issue 8 by distributing the "padding" spaces as evenly as possible. I had had similar thoughts, but worked on a routine that is only applicable to dot-matrix printers. This allows inter-word spaces to be, say, 1.5 spaces in width, at the cost of much slower printing. I have used it on my Tasword files for this Newsletter - what do you think? I may publish it in the next issue.

****

SHORT TIP

From Ettrick Thomson: If you want to alter a program without disturbing a laboriously built-up display, you can KEYIN the new line without an automatic listing being produced. (Better have a printout of the program handy, as a guide - Ed.)

****
MATHS OPERATIONS ON ARRAYS

A large part of the time taken by many computer programs in business and science is due to repeated identical operations on each number in an array or set of arrays. Math's chips, or parallel processing, can be used to reduce the time taken. Well, we can't use that on the Spectrum, but improved software can increase the speed of array operations considerably; we can get quite a good increase in speed simply by using machine code for the main loop that deals with each number in the array in turn. I know that at least some readers are interested in handling numeric arrays, so I am providing a basis for experimentation which gives about a 10-fold speed increase.

To help the explanation, I give below the BASIC that this machine code is equivalent to. There is no need to type these lines in! We might add together the contents of two arrays (let's call them the first and second arrays) and put the results into a third array. If each array has *size* numbers in it, we might use something like this to do it:

```
10 FOR n=1 TO size
20   LET c(n)=a(n)+b(n)
30 NEXT n
```

Obviously we can alter the "+" to or "/" if we prefer. If we wanted to add a constant in place of the numbers from the second array, then line 20 could be:

```
20 LET c(n)=a(n)+b(1)
```

The first number in b() would be preset to the constant we want. (I know you would normally use e.g. "x" instead of "b(1)" but the latter is closer to the machine code.) If we want the results to be placed into the first array instead of the third array, we just alter "c(n)" to "a(n)" and we don't need a third array anymore. I hope this is simple - now let's look at the machine code version.

PROC setup POKEs the machine code into the UDG area; it only needs to be called once, but in any case it does nothing if it decides the code has been POKEd already. Line 10 to 70 create some arrays and demonstrate the procedure. The procedure obtains the addresses of the arrays from the array names, which are passed by an unusual method; a DATA list is used instead of conventional parameters. This is neccessary because of a problem with passing arrays by reference - you cannot pass the same array twice. If you do, Beta Basic will try to "rename" the same passed array with two different names (which would be specified in the DEF PROC statement) and you will get odd results. For the present application, we might well want to pass the same array several times; for example, when adding an array to itself. Besides, we don't need to do any conventional array handling inside the procedure - all we need is the start addresses of the arrays. LENGTH can obtain these from a string variable containing the array name, plus "()".

The machine code is POKEd with the three start addresses which tell it where to get the first number and the second number, and where to put the result after adding, subtracting, multiplying or dividing them. Some or all of these addresses may be the same. (You can work out the address of an individual number if you wish from the length of a number, which is 5

bytes. If you want to use two-dimensional arrays, it will be useful to know that the array elements occur in the order 1,1 1,2 1,3... etc. 2,1 2,2 2,3 . . . etc. in memory.)

The POKE in line 1070 gives the size of the array containing the "first" numbers to the machine code. This determines how many numbers to deal with. WARNING!! The array to contain the results should be at least as long as the "first" array! Normally, all three arrays (if different arrays are used) will be the same length. An exception occurs when you want to use a constant number in place of the "second" array - in this case, you still need an array to allow LENGTH to produce an address, but the array need be only 1 number long. You could DIM b(1): LET b (1)=10 and use the code to add 10 to every number in an array a().

The POKE in line 1080 determines which operation to perform. (The number is a code used by the ROM's calculator.)

```
  10    setup
  20    LET s=50
        DIM a(s)
        DIM b(s)
        DIM c(s)
  30    FOR n=1 TO s
           LET a(n)=n
        NEXT n
  40    FOR n=1 TO s
           LET b(n)=n*2
        NEXT n
  50    PRINT "working..."
  60    matrix a,b,c
  70    FOR n=1 TO s
           PRINT c(n)
        NEXT n

1000    DEF PROC MATRIX DATA
1010       LOCAL a$,b$,c$,c
1020       READ LINE a$, LINE b$, LINE c$
1030       LET c=USR "a"
1040       DPOKE c+1,LENGTH(0,a$+"()")
1050       DPOKE c+4,LENGTH(0,b$+"()")
1060       DPOKE c+8,LENGTH(0,c$+"()")
1070       DPOKE c+11,LENGTH(1,a$+"()")
1080       POKE c+28,15
           REM l5=ADD,3=sub,4=mult,5=divide
1090       DPOKE c+23,0
           REM 65531 for consts
1100       RANDOMIZE USR c
1110    END PROC

1120    DEF PROC setup
1130    IF PEEK USR "a"<>33 THEN
           RESTORE
1140       FOR n=USR "a" TO USR "a"+50
              READ a
              POKE n,a
           NEXT n
1140    DATA 33,0,0,17,0,0,221,33,0,0,1,0,0,197,213,205,180,51
        ,227,205,180,51,1,251,255,9,229,239,15,2,56,235,221,22
        9,209,1,5,0,221,9,237,176,209,225,193,11,120,177,32,21
        9,201
1150 END PROC
```

****
BETA BASIC AND THE PLUS 3

I have spent some time recently pulling the +3's ROMs (about 60K of them!) to bits - my printer got pretty hot doing disassemblies. It doesn't look too difficult to get Beta Basic working on it. It has a nice operating system to make life easier for me - adding serial and possibly random files (left out of the Basic!) should be fairly simple. As for the machine itself, the disc drive takes about 9 seconds to load 16K, which is a bit faster than an OPUS Discovery but less than a quarter of the speed of a Disciple or a +D interface. The drive seems reliable, the keyboard is fine, and it is nice to have RS232 and Centronics built in. Unfortunately, the editor gives me high blood pressure! I don't think the machine has sold very well so far. Personally, I'd hang on for a price fall if you can. I am sure the +3 could be sold for £160 or less if need be. At that price, it would be a good buy.

The ROM error messages include "+2A does not support format" and I think a +2A may appear, with a built-in tape recorder, but the ability to add an external disc drive.

****
READERS' LETTERS

Dear Andy,

I have discovered 2 bugs (I hope). (A listing was enclosed that used INKEY$ and PAUSE, amongst other commands. It didn't work reliably. Ed.) When LIST FORMAT 2 is in operation long lines are not correctly indented when CSIZE 5,8 is being used.

Paul Field, Halsham, N. Humberside.

There is a ROM bug in PAUSE that may be important; try the example below

```
10   IF INKEY$<>”" THEN GOTO 10
20   PRINT LEN INKEY$
30   PAUSE 200
```

If you enter RUN (ENTER) and keep your finger on the ENTER key, the program will wait for you to remove your finger. Line 20 will print 0, because no key is being pressed. Then you expect a 4 second delay, but in fact there is none. This is because PAUSE warts for a "key has been pressed" flag in the system variables to be set. In some circumstances, the PAUSE command is entered with the flag already set, so no PAUSE actually happens. PAUSE should reset the flag to "no key pressed" initially. He can fix this with the "patch" below which does the resetting before the ROM routine is entered. I just put the code in the UDG area, but you could incorporate it permanently into BB's code if you like.

```
10   LET X=USR "A"
20   DPOKE 64832,X
30   FOR N=X TO X+9
40      READ A
        POKE N,A
     NEXT N
50   DATA 6,0,0,0,205,79,31,195,58,31
60   DPOKE X+2,X+4
```

The lack of indentation with CSIZE 5,3 I don't understand, but I guessed it might be something to do with odd pixels "left over" after the screen width of 256 is divided by 5. So I tried:

```
WINDOW 1;1,175,255,176: WINDOW 1
```

and it cured the problem.

Dear Dr. Wright,

The otherwise admirable procedure VKILL (no. 8) has a bug - try to vkill the variable S! The trouble is that when *posh* in line 1080 looks for the FOR-NEXT version of S it finds it among the first seven bytes of the variables area where it is part of Beta Basic's variable XOS with value 0, so the POKE of line 1090 creates chaos. A solution is to skip BB's use of the variables area by inserting +32 after the DPEEK(23627) of line 1080. It is to my liking also to replace the GOTOs of lines 1030 and 1040 with ENDPROC. (I am not sure if it is always safe to do that in such circumstances but is seems so in this instance.)

G. Jackson, Cardiff

Thanks for pointing this out! Your cure will fail, though, in any case where a multi-letter numeric variable has a value of zero and ends in a letter which matches the single-letter variable you are trying to delete. It would be better to make the value we search for much more unusual than zero - how about 2.961784785E-39? That is coded in the variables area as 5 successive CHRS 1's. (How do I know such exotica? Well, I used LENGTH (0,"x()") to find a numeric array and then PEEKed and POKed a bit, in combination with altering the value of x(1) and printing it.) If you alter the two KEYINs in line 1060 so that "0 to 0" becomes "2.961784785E-39 to 1" and "=0" becomes "=2.961784785E-39", and replace the five CHRS 0's in line 1080 with STRIHGS(5,CHRS 1) all will be well.

Multiple END PROCs will work O.K. in all cases. The only problem is that Beta Basic will no longer be able to "skip over" procedure definitions when it comes to them; you will have to put a STOP before your procedures, or jump round them. (BBC Basic always requires this, but Beta Basic has the sense to skip to the next END PROC if it runs into a DEF PROC when the procedure hasn't been called.)

Dear Andy,

You may think it worth while to include the results of my delving into the Kempston E Centronics interface to round out the last paragraph of the BB 4.0 manual. The POKES to control the status of the interface (without using the COPY: REM.. system) from within a program are as follows:

```
POKE 23729,0    CR To reset as at power-up.
          ,10   +LF  To initiate line-feed at each CR.
          ,16   ESC  Equivalent to doing LPRINT CHR$ 27;
          ,32   ↑ To give double-height print.
          ,64   TOKENS OFF
```

Any sum of the above values provides any needed combination.

S. Charles Bustard, Chorleywood, Herts.

The list below was kindly prepared by Charles Buszard. It is a listing of the procedures in Newsletter issues 1 to 7; the REF column gives issue/page number.

****

BETA BASIC NEWS-LETTER PROCEDURES; ALPHABETICAL ORDER.

PROCNAME	REF.	CATEGORY	PROCNAME	REF.	CATEGORY
G & K	7/05	KEYWORD control	keyprint	1/14	INPUT routine
adr	4/11	Screen pixel	kill	4/08	DELETE block.
axes	3/05	Scaled graph	leave	5/03	(See smenu)
binar	1/02	BIN/DEC convert	locate	5/09	(See errmes)
bold	5/07	Printing, bold.	lplot	4/13	PLOT, lo res.
bplot	1/04	PLOT bold	pixel m/c	4/13	CODE, storage.
byte	7/05	Hi/Lo split.	maxmin	2/11	(See ends)
c	3/01	CAT in WINDOW.	md	2/08	M/drive command
call	2/04	CALL for USR.	menu	5/04	Menu themes.
case	3/04	In CASE of - -	mirror	7/04	Mirror image.
centre	3/02	Centre a string	off	1/05	Andy's joke.
centre	6/08	Centre improved	os-num	5/11	Old-style Nos.
chardes +	4/10	Char. designer.	overwindow	1/02	WINDOW command.
countlines	2/15	(See slist)	paglist	4/12	LIST, by pages.
cpy	4/04	COPY at 64 cpl.	part copy	2/15	(See slist)
cursor	6/11	Cursor, j-stick	perm, etc	6/03	Permutations.
decolor	5/15	Remove controls	plist	4/05	LIST, PROCs.
dir	7/02	Disciple disc.	poly	2/06	(See ellipse)
display	4/10	(See chardes)	polygon	2/06	(See ellipse)
down	3/06	PRINT downwards	pplot	1/08	CSIZE PLOT.ting.
dump	4/07	COPY, graphics	prtstr	3/02	PRINT strings.
ed	4/10	(See chardes)	ptime	1/03	Timer, print.
edge	7/05	BORDER designer	readnumber	3/08	INPUT command
ellipse	2/06	DRAW ellipse.	readstring	3/09	(See readnumb.)
end2	5/14	CLEAR stack.	repeat	5/08	LOGO-style;
end-repeat	5/08	(See repeat)	rotate	4/06	Chars. command.
ends	2/10	Mark an array.	seqa, etc	6/06	Sierspinski;
errmes	5/09	Error message.	set+reset	6/07	Bit handling.
evolution+	6/05	Evolution model	short	5/02	String handling
fatchar	5/07	Characters, fat	short	6/08	Improved short.
gdraw	3/06	(See axes)	show	2/02	(See hide)
gplot	3/06	(See axes)	shrink	1/05	Screen handling
grab & put	1/07	Screen control.	sketch	1/04	Cursor control.
gtime	1/03	Timer, get val.	slist	2/14	CSIZE LLISTing.
hanoi	4/14	More on Hanoi.	smenu	5/03	Menu theme.
hanoi	3/03	Recursivity.	sound	2/05	Sound effects.
hide	2/02	PROCS, hidden.	splay	7/06	Music, 128k.
hilite	5/08	Hilite on PLOT.	strip	5/15	(See decolor)
import	5/06	Tasword 3 usage	sv	2/05	SAVE, Beta disc
invert	2/02	(See hide)	torus	7/11	Visual display.
italics	5/07	PRINT, italics	udg	1/03	Udg handling.
jump	7/04	To statement.	wdraw	4/03	(See windo)
key	7/05	Any key continu	windo etc	4/03	WINDOW design.
keyboard	7/12	Music, 48k.	ztime	1/03	Timer, zero.

****

Send all contributions, questions and correspondence to:

BB NEWSLETTER, 24 WYCHE AVE., KINGS HEATH, BIRMINGHAM B14 6LQ

Scanned, Typed, OCR-ed, and PDF by
Steve Parry-Thomas 28th October 2004.
This PDF was created to preserve this
Newsletter for the future.
For all ZX Spectrum, Beta Basic
And www.worldofspectrum.org users
(PDF for Michael & Joshua)