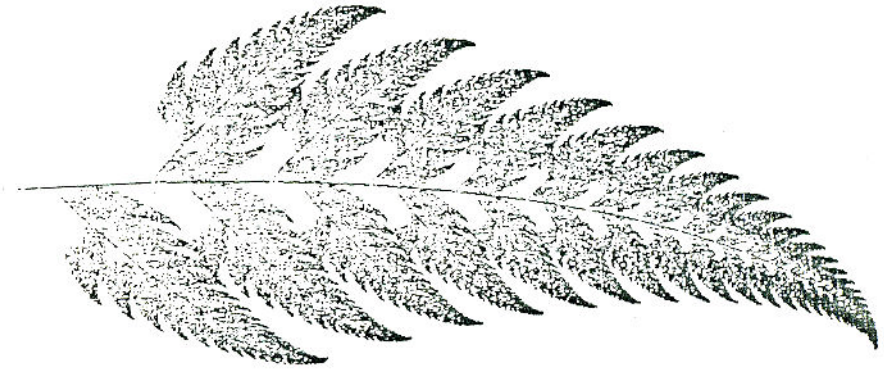
BETA BASIC NEWSLETTER No.10



Fractal fern described in this issue

****

HIGH RESOLUTION SCREEN DUMPS

About 8 months ago I got a letter from Ejgil Hansen (Karlslunde, Denmark). He had had a screen dump routine published in the last issue of ZX Computing, and had later worked out a better 512*350 pixel routine which he thought I might like to look at. Ejgil's idea was to use a reserved area of RAM as a high-resolution "screen" memory on which points could be plotted and lines drawn. This memory could be transferred to the normal screen for examination (at lower resolution) or dumped to a printer. Sparked by this letter, I decided to see what I could cook up along these lines in Beta Basic. (Ejgil used machine code and Spectrum Basic.)

I tried a lot of things - GETS and MEMORY and POKES and giant scrolling screens - which had one disadvantage or another. The arrangement I finally settled on was fairly simple. When deciding how the extra "screen" memory would be organised, I found there was a conflict between a convenient arrangement for printer dumps (which requires a byte to code for a vertical screen slice) and a convenient arrangement for transferring the data to the real screen memory for examination. Eventually I decided just to place any plotted points on both the real screen and the invisible hi-res screen at the same time, without worrying about transfers to and from the real screen. This allowed the data format to be designed for easy screen dumps (on an EPSON-style printer). I used a string array r$(h,w) in which each character codes for a vertical slice of 1*8 pixels. The length of the strings in the array determines the width of the hi-res screen in pixels, and the number of strings in the array determines the height of the hi-res screen in B-pixel units (i.e. character rows). BB 4.0 users could modify the routines to use RAM disc arrays, which would allow 10 times the normal screen area to be coded for. (Of course, it starts to take a long time to fill it up with graphics!)

The array needs to be initialised to CHR$ 0's, rather than spaces. PROC setup_rs sets up an array suitable for a hi-res screen of specified width and height (given in character squares). It also prepares an array of powers of 2 that is used later by PROC plotrs. In contrast to my usual style, there is no - demo program for these procedures here; instead, see the following item on fractals.

```
400  DEF PROC setup_rs w,h
       DIM r$(h,w)
       FOR n=1 TO h
         LET r-$(n)=STRING$(w,CHR$ 0)
       NEXT n
       DIM q(8)
       FOR n=0 TO 7
         LET q(n+1)=2↑n
       NEXT n
     END PROC
```

PLOTing a point on the hi-res screen can be done using the procedure below. The required bit in the array is set by ORing at the binary level (using BB's OR) with an entry from the array q, which will be 1,2,4,8,16,12,64 or 128. This is faster than calculating powers of 2 each time. Another point concerning speed: if you are doing lots of points e.g. drawing a line, it would be faster to incorporate the working part of this procedure into the line-drawing program, rather than use it as a separate PROC. This is what I did in the FRACTALS item below, so the PLOTRS procedure isn't actually used, but it might be useful...

```
500  DEF PROC plotrs x,y
       LOCAL r
       LET r=h-INT (y/8)
       LET r$(r,x+1)=CHR$ (OR(CODE r$(r,x+1),q(y-INT (y/8)*8
       +1)))
     END PROC
```

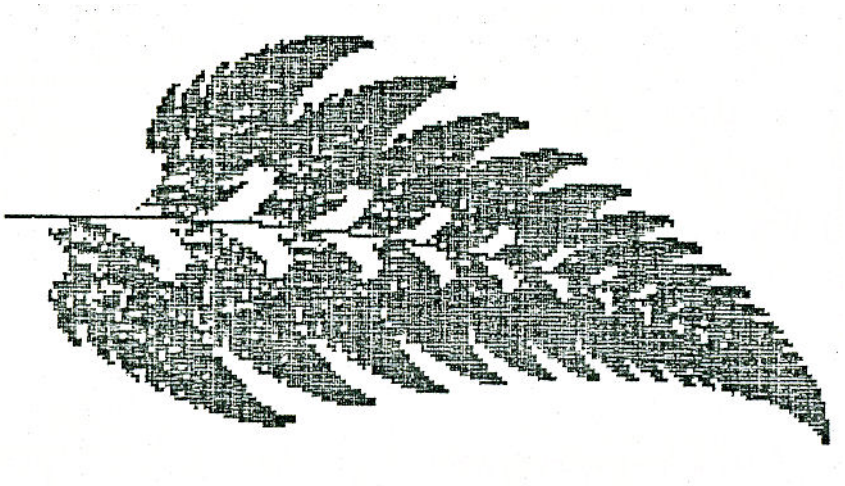
To have a proper loot: at your masterwork, dump it to a printer with something like the procedure below. LPRINT must be directed to a "b" type channel which will let through any character codes. The first LPRINT should set the line spacing, and b$ should contain control codes to tell your printer that W bit-image bytes are being sent. If your printer does an automatic line feed after carriage return you won't need the CHR$ 10 after LPRINT.

```
600  DEF PROC dumprs
       CLOSE #3
       OPEN #3;"b"
       LOCAL a$,b$,r,c,w
       LET w=LENGTH(2,"r$")
       LPRINT CHR$ 27;"A";CHRS 8;
       LET b$=CHR$ 27+"*"+CHR$ 6+CHRS MOD(w,256)+CHR$ INT (w
       /256)
       FOR r=1 TO LENGTH(1,"r$")
         LPRINT b$;
         FOR c=1 TO w
           LPRINT r$(r,c);
         NEXT c
         LPRINT CHR$ 10
       NEXT r
       LPRINT CHR$ 27;"A";CHRS 11;
     END PROC
```

****

FRACTAL FERNS

For some time after devising the hi-res screen system described earlier, I lacked a good use for it. Recently, however, I was playing with a fractal pattern program from BYTE magazine that I thought would benefit from greater resolution than is possible on the Spectrum. However, if you don't have a suitable printer, or just want a screen pattern, rather than a print-out similar to the illustration on the cover, then omit line 20 and the last two statements in line 180. (This gives better speed anyway!) The illustration below is a normal dump from the real Spectrum screen - compare it with the finer detail of the cover illustration.



I don't have the space or the expertise to describe fractals or the program below properly, so I will be very brief. Fractals are patterns which look the same as you magnify them (within the resolution of your system). For example, the fern on the cover has side branches which resemble the main fern, but are smaller, and the side branches have smaller side branches which look like the side branches, only smaller, and so on. A fairly small set of numbers can describe the relative sizes of the main and smaller parts, their angle, and other trey factors, to give a complex pattern. The numbers to generate the fern are given in the DATA statements in lines 40 and 50. The plotted points are distributed in a partly random way, and the fern is built up gradually in better and better detail.

To fit the fern, I set up a hi-res screen of roughly normal width but 3, times normal height. I originally had a PRINT x, y statement in line 180; knowing the likely range of values, I then got "fiddle factors" to put in the PLOT statement to magnify the fern and keep it on-screen, and "fiddle factors" to get j and k from x and y to keep the hi-res plot within the limits of the array. For the cover illustration, I set the FOR loop in line 140 to a very high value and went to bed!

```
10   LET wid=35,hei=64
20   setup_rs wid*B,hei
30   DIM a(4)
     DIM b(4)
     DIM c(4)
     DIM d(4)
     DIM e(4)
     DIM f(4)
     DIM p(4)
40   DATA 4
50   DATA .85,.04,-.04,.85,0,1.6,.85
     DATA -.15,.28,.26,.24,0,.44,.07
     DATA .2,-.26,.23,.22,0,1.6,.07
     DATA 0,0,0,.16,0,0,.01
60   READ t
70   LET pt=0
80   FOR n=1 TO t
90      READ a(n),b(n),c(n),d(n),e(n),f(n),z
100     LET pt=pt+z
110     LET p(n)=pt
120  NEXT n
130  LET x=0,y=0
140  FOR n=1 TO 10000
150     LET z=RNDM(0)
160     IF z<=p(1) THEN LET k=i
        ELSE IF z<=p(2) THEN LET k=2
        ELSE IF z<=p(3) THEN LET k=3
        ELSE LET k=4
170     LET s=a(k)*x+b(k)*y+e(k),t=c(k)*x+d(k)*y+f(k),x=s,y=t
180     IF N>10 THEN
           PLOT x*19+100,y*17
           LET j=x*57+125,k=y*51,r=hei-INT (k/8)
           LET r$(r,j+1)=CHR$ (OR(CODE r$(r,j+1),q(INT k-INT (
           k/8)*8+1)))
190  NEXT n
```

For a pattern of fractal triangles, alter lines 40 and 50 as follows. (The scaling "fiddle factors" and the array size will also need changing.)

```
40   DATA 3
50   DATA .5,0,0,.5,0,0,.34
     DATA .5,0,0,.5,1,0,.33
     DATA .5,0,0,.5,.5,.5,.33
```

****

LISTING BETA BASIC's FUNCTIONS

Beta Basic uses a large number of the available user function letters as a means of adding its own functions. These are transformed by the print routine into long names, so we see e.g. STRING$ instead of FN s$ in our listings. We can exploit this to provide a useful list of all possible functions; any functions not turned into keywords are not used by BB and are still available. (For upwards compatibility with BB 4.0, avoid using FNs Y$, F$ and X$, which give extra functions in this version.) CHR$ 168 is "FN"; it is advisable to use this form, rather than "FN" itself, or the line may look odd as you edit it.

```
10   FOR n=CODE "a" TO CODE "z"
20      PRINT "    ";CHR$ n;"  ";CHR$ 168+CHR$ n+"(";
        TAB 16;CHR$ 168+CHR$ n+"$("
30   NEXT n
```

****

EUROELECTRONICS LPRINT III PRINTER INTERFACE

This is from Iain Rendall of Edinburgh. (His father Ken also subscribes; I've sold quite a few beta Basics by word-of-mouth - even three copies to one family!) Iain writes:

"I am using the LPRINT III with the 48K... Before I had read the bit at the back of the BB 4.0 manual, I made the following changes - DPOKE 47005,23679 and DPOKE 59291,23679. These originally gave references to the 57500 characters/line variable, but as the interface uses 23679, I thought I would make the changes. Also, it is possible to control the interface more or less as normal, using a couple of DPOKEs. DPOKE DPEEK(23631)+15,3836 allows instructions to be sent to the interface as normal and DPOKE DPEEK(23631)+15,64423 returns control to Beta Basic after the job is done. For example, even after BB is loaded and has control of the "p" channel, graphics can be selected by the following procedure: graphic 1 gives small size; graphic 2 gives large size."

```
1000 DEF PROC graphic s
1010 IF s<>1 AND s<>2 THEN RANDOMIZE USR 9465
1020 DPOKE DPEEK (23631)+15,3836
1030 LPRINT CHR$ 13
1040 LPRINT CHR$ 3
1050 LPRINT CHR$ 0:CHR$(7-2*s)
1060 LPRINT CHR$ 13
1070 DPOKE DPEEK (23631)+15,64420
1080 END PROC
```

"However, issuing the COPY command still causes BB to lose control - things can be restored with the next PROC, with line-spacing or other things perhaps altered though:"

```
2000 DEF PROC copyret
2010 LPRINT CHR$ 13 CHR$ 3;CHR$ 5;
2020 COPY: DPOKE DPEEK(23631)+15,64423
2070 END PROC
```

****

USE OF ALTER

Ettrick Thomson (Aldeburgh, Suffolk) sent this tip:

Your manual example 'ALTER 1 TO “23”' points out that such an operation does not produce a usable line, but you don't mention that subsequent 'editing' will put things right.

I recently wanted to make several changes to a program: for one of them the easiest way was first to

```
ALTER " LET s=USR 54321" TO " POKE 16384,s$"
```

which not only fails to provide '16384' with invisible bytes, but leaves in the program area those for '54321'; then

```
REF " POKE 16384,s$"
```

followed by repeated pressings of ENTER until the OK' message, brought the corrupt lines to the editing area, stripped of all invisible bytes, and then returned them to the program area, properly fitted out with invisible bytes.

****

A CALCULATOR

The 128K Spectrum has a calculator option which is hard to get at from Beta Basic. Here is a calculator procedure that will work with both BB 4.0 and BB 3.0. It has a short name so that it can be invoked easily. You might want to use it while you have something important on the screen, so the area over-written by the calculator screen window is saved beforehand and later restored. The PEEKS and POKES in the procedure allow the previous window and CSIZE to be found and restored. (CSIZE is normally changed automatically as you change windows, but unfortunately WINDOW 0 always sets CSIZE 0 - hence the need to handle CSIZE "manually".)

You can use any valid numeric expression as an entry. Also, a "current value" is maintained in v$, which allows entries that normally require a preceding number, like "+5", to be evaluated as "current value+5". Press ENTER on its own to finish using the calculator. A sample use of the calculator might involve typing the following (with ENTER at the end of each line):

```
LIST (optional, to demonstrate window save/restore)
(space)c
16/3
+1234
/14
*23
/2↑10
```

The final display (if you use PRINT USING as I did) will look like the one below. Alternatively, use a simple PRINT, and widen the window width from 88 to 104 to allow for long numbers.

```
    5.3333
 1239.3333
   88.5238
 2036.0476
    1.9883
```

```
  10    DEF PROC c
           LOCAL w,xs,ys,v,a$,t$,v$
           LET w=AND(PEEK 57407,127),xs=PEEK 57370,ys=PEEK 57371
           GET t$;0,175,11,8;1
           WINDOW 2;0,175,88,64
  20       WINDOW 2
           PAPER 6
           INK 1
           CLS
           LET v$="0"
           DO
              INPUT LINE a$
           EXIT IF a$=""
              LET a$=(v$ AND (a$(1)<"0" OR a$(1)>"9"))+a$
  30          LET v=VAL a$
              PRINT USING "######.####";v
              LET v$=STRS v
           LOOP
           PLOT 0,175,t$
           WINDOW w
           CSIZE xs,ys
        END PROC
```

****

LOADING AND EXECUTING A PROCEDURE

Subscriber Richard Edwards (London) wrote:

"I've been racking my brain trying to think of a way to merge a RAM disc procedure and at the same time call it, with all the necessary parameters, all with one statement."

You will gather that Richard is a BB 4.0 user, but his ideas are also relevant to BB 3.0 with disc or Microdrive, provided speed isn't critical. At this point, things were not going very well for Richard - but there was another sheet of paper that continued:

"Luckily, I forgot to post the letter the day I wrote it, and since then I've spent a sleepless night and come up with this other procedure that does everything I wanted except sing and dance. Variables can be passed by reference but care must be taken not to use the same variables as the procedure or they will be corrupted. I've chosen uncommon names for the four variables for this reason."

The basic idea is to fill up your RAM disc, cartridge or disc with procedures that have line numbers in the 9000+ range. These are then MERGEd, executed and deleted by Richard's PRGC. Richard used the tip in Newsletter 8 to allow non-letter PROC names, so he could type e.g.: ! test,1,2 to load and execute the procedure "test" with parameters of 1 and 2.

Perhaps not everyone will want to use the name "!" ,especially if a RAM disc isn't involved, and I'm not sure how to index "!", so I will use the name "lrun" for Load and RUN.

```
10   lrun test,1,2

100  DEF PRGC lrun DATA
110     LOCAL u$,j$,hk,jk
120     LET u$=""
170     DO
           READ LINE j$
140        LET u$=u$+j$+","
150     LOOP UNTIL ITEM()=0
160     LET hk=LEN u$
170     LET jk=INSTRING(1,u$,",")
180     LET u$(jk)=" ",u$(hk)=" "
190     MERGE !u$(1 TO jk)
200     KEYIN u$
210     DELETE 9000 TO
220  END PROC
```

Notes: The DO-LOOP in this example will finish with u$ equal to "test,1,2,". Line 170 finds the first comma so that the first part of u$ can be used as the name to MERGE. (So, obviously, the procedure must have been previously SAVEd with a file name that is the same as the procedure name.) Alter line 190 to MERGE from tape, Microdrive or disc instead of RAM disc. Line 180 removes the first and last commas so that u$ equals "test 1,2 " which is the correct form for KEYIN to "type in". Once the procedure "test" has been called in this way, it is deleted again to reclaim the memory. In this way, programs can be larger than the memory normally available to BASIC.

****

STR$ BUGS

Alan Salmon (Bristol) reported that something like:

```
        PLOT x,y; STR$ a
  or    PLOT INK 3;20,20;STR$ 1
```

plotted the string not in the attributes you would expect, but in those for the lower screen. This turns out to be due to a ROM bug in the STR$ function. STR$ works by saving the current stream, then setting a special output stream which allows the characters of the STR$ to be PRINTed to an internal workspace. (This system was exploited in the CAT_TO procedure in issue 9.) Afterwards, the previous stream is restored, and its permanent attributes are copied to the temporary attributes in a "setting up" operation. This means that e.g.

```
        PRINT INK ;STRS 1
  or    PLOT INK 2;128,88,STR$ 1
```

are ineffective, because the setting of the temporary INK 2 is nullified by the "setting up" done by STR$. The PLOT version will plot in the permanent colours of the last-used stream - this might be the lower screen - e.g. after RUN or INPUT. But if you used PRINT (even PRINT;) more recently then the permanent colours will be those associated with the upper screen. The whole problem can be avoided, as noted by Alan, by e.g.:

```
     LET a$=STR 1: PLOT INK 2;128,88,a$
```

STR$ is also bugged in another, better known way; an extra zero is placed on the floating point calculator stack if you use STR$ with a number less than 1, so that e.g.:

```
     PRINT "xyz"+STR 0.1
```

and

```
   PLOT 128,88,STR$ 0.1
```

behave oddly. The AXES procedure in issue 3 was not protected against this bug, and several users have noticed odd results with certain values. Robert Dickson (Blackheath, London) suggested the following changes:

1. Add LET x$=STR$ x before the two PLOT .... STR$ x statements.
   Alter PLOT .... STR$ x to PLOT .... x$.
2. Add LET x$=STR$ y before the two PLOT .... STR$ y statements.
   Alter PLOT .... STR$ y to PLOT .... x$.
3. Add x$ to the LOCAL statement.
4. A further (unrelated) improvement - Alter the third FOR-NEXT loop to:

```
          FOR y=ystp TO yrg-yos-ystp STEP ystp
```

****

"CLIPPING" LINES

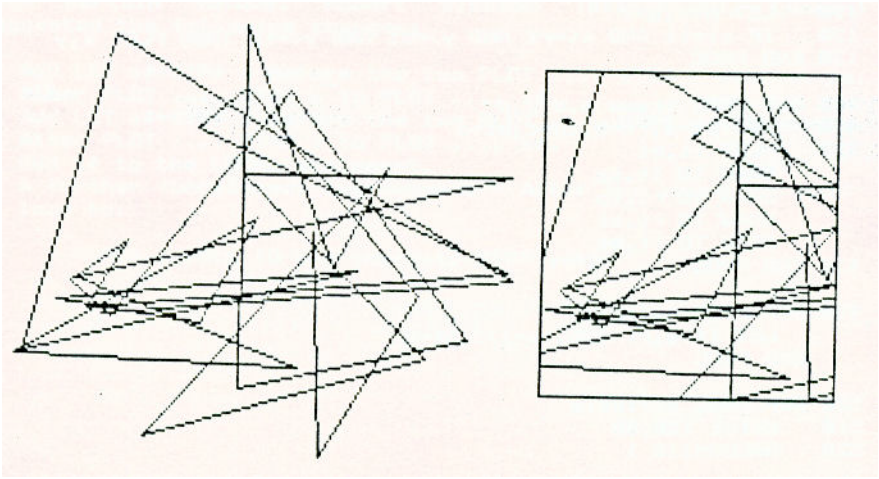
Reader David Reed asked me if I could add a RP command to draw lines in a screen window, by "clipping" the line to fit. This should also allow lines to be "off-screen". The more I thought about it, the nastier the problem seemed, but I got interested and managed to write a suitable procedure. (I think using machine code would be rather harder.) I was afraid that I might have missed some much simpler method that would do the trick, so I sent my first attempt to regular contributor Ettrick Thomson for comments. He suggested some minor changes, and added:

"Quite a problem, this "cdraw_to": saying what it has to do is simple, almost trivial. But turning it into a computer program -!! In confirming your approach, I divided the screen into 9 compartments, a noughts-and-crosses board, the framed window being in the middle. This gives 81 possibilities for pairs of points (p,q), (x,y) but symmetry reduces that to 11; each of these 11 has 2 to 8 variations"

So it seems the procedure CDRAW_TO (clipped DRAW TO) *has* to be quite complicated! I also needed a CPLOT (clipped PLOT); this sets the global variables p and q which record the graphics pen position. The pen position can be anywhere, because p and q can have any values. (CDRAW_TO should show any drawn lines when the pen moves across the graphics window.) The FRAME procedure sets up global variables specifying the left, right, top and bottom of the drawing window, and draws a frame, and the NOFRAME procedure sets the window to the full screen area. SCRIBBLE draws a random pattern for purposes of illustration. (Always the same "random" pattern, to make comparison easier.) It should be quite to add a clipped relative draw command; real experts can try allowing arcs to be clipped! If you like, change line 540 so that the random line goes off the normal Spectrum screen.

```
10   noframe
20   scribble
30   PAUSE 0
40   CLS
50   frame 20,140,150,30
60   scribble

100  DEF PROC cplot x,y
110     LET p=x,q=y
120     IF x>=lt AND x<=rt AND y>=bt AND y<=tp THEN PLOT x,y
130  END PROC

400  DEF PROC frame a,b,c,d
410     LET lt=a,rt=b,tp=c,bt=d
420     PLOT lt,bt
        DRAW TO lt,tp
        DRAW TO rt,tp
        DRAW TO rt,bt
        DRAW TO lt,bt
430  END PROC

450  DEF PROC noframe
        LET lt=0,rt=255,tp=175,bt=0
     END PROC

500  DEF PROC scribble
510     cplot 128,88
520     RANDOMIZE 1
```

```
530     FOR n=1 TO 30
540        cdraw_to RNDM(255),RNDM(175)
550     NEXT n
560   END PROC

600   DEF PROC cdraw_to x,y
610     LOCAL g,tx,ty
620     DO
630   IF p<lt AND x<lt OR p>rt AND x>rt
      OR q<bt AND y<bt OR q>tp AND y>tp THEN
             LET p=x,q=y
          EXIT IF 1
640       REM adjust pen posn p,q to be in the window
650       LET g=(q-y)/(p-x+.000001)
660       IF p<lt THEN
             LET q=q+(lt-p)*g,p+lt
          ELSE
             IF p>rt THEN
                LET q=q+(rt-p)*g,p=rt
670          IF q<bt THEN
                LET p=p+(bt-q)/g,q=bt
          ELSE
             IF q>tp THEN
                LET p=p+(tp-q)/g,q=tp
680       IF p<lt OR rt<p THEN
             LET p=x,q=y
          EXIT IF 1
640       LET tx=x,ty=y
700       REM adjust dest posn x,y to be in the window
710       IF x<lt THEN
             LET y=y+(lt-x)*g,x=lt
          ELSE
             IF x>rt THEN LET y=y+(rt-x)*q,x=rt
720       IF y<bt THEN
             LET x=x+(bt-y)/g,y=bt
          ELSE
             IF y>tp THEN LET x=x+(tp-y)/g,y=tp
730       PLOT p,q
          DRAW TO x,y
740       LET p=tx,q=ty
750     LOOP UNTIL 1
760   END PROC
```



****

PROC TOP - setting the top line for ALTER, REF and LIST REF

I intended at one time to allow REF and ALTER to work on a "slicer" of program lines, but the syntax got pretty nasty and I dropped the idea. However, it is obviously useful to be able to start searching or changing a program somewhere in the middle, so I am giving a DPOKE to allow this. The DPOKE can be conveniently "wrapped up" in a procedure. The first version below is for BB 3.0 users, the second one is a BB 4.0 version. The default line where searches or changes start is the line with the cursor. You can use: TOP 100: REF a$ to look for a$ from line 100 onwards.

```
10   DEF PROC top lin
       DEFAULT lin=DPEEK(23625)
       DPOKE 48804,lin
     END PROC
```

The following BB 4.0 version has to be more complicated because the location we want to DPOKE now sits in paged RAM. Line CO sets up t$ to contain code to: CALL switch page: LET HL = lin: DPOKE (the place),HL: GOTO set normal page. Line 60 finds the string and executes the machine coda it contains.

```
10   DEF PROC top lin
20     DEFAULT lin=DPEEK(23625)
30     LOCAL t$
40     POKE 49146,255
50     LET t$=CHR$ 205+CHR$ 177+CHR$ 185+CHR$ 33+
       CHAR$(lin)(2)+CHAR$(lin)(1)+CHR$ 34+CHR$ 11+
       CHR$ 246+CHR$ 195+CHR$ 193+CHR$ 185
60     RANDONIZE USR LENGTH(0,"t$")
70   END PROC
```

When REF or ALTER are used, the code we altered in paged RAM is copied to non-paged RAM and used - unless Beta Basic "knows" that REF or ALTER were used recently and don’t need to be copied. The POKE in line 40 ensures that BB always decides that the code needs copying. (If this is clear as mud - don t worry')

****

PROCs DEVAL and REVAL - saving memory

Subscriber Michael Williams (West Ealing, London) asked me for procedures to convert all the numbers in a program to their VAL "n" form, and vice versa, The VAL form saves 3 bytes per number, because although you need an extra 3 bytes for the VAL keyword and a pair of quotes, the invisible form of the number is no longer present in the line (saving 6 bytes). However, this form is harder to read, so a procedure to change VALs back to the normal form is desirable.

The procedures below are designed to be MERGEd at the start of a program. Both are fairly fast. Let's go through the REVAL (add VAL forms) procedure first. The DPOKEs to 48804 alter the first line affected by ALTER, as explained in the previous item; BB 4.0 users should substitute ‘TOP’ for ‘DPOKE 48804, ‘. It is necessary to prevent ALTER working on lines 1 to 9, or the procedures will alter themselves while running, causing problems. The sub-procedure GET_P is a method for finding the address of the line after the set of procedures by examining the NXTLINE system variable. In this case, it finds the start of line 10, which allows a search of the program area from line 10

onwards for CHR$ 14+CHR$ 0+CHR$ 0. These characters normally occur after whole numbers (but not floating point numbers - so the PROC won't work on these) in a program line. If there is a successful "find", the loop in line 2 backs through the line, checking for valid digits and building up the complete number in the string t$. Then we can use ALTER to change the number (possibly at multiple locations) to its VAL form. The process continues until all whole numbers have been converted.

```
1  DEF PROC renal
     DPOKE 48604, 10
     LET $t=CHR$ 14+CHR$ 0+CHR$ 0
       get-p
     DO
       LET p=INSTRING(p,MEMORYS()( TO DPEEK(23627)),a$)
     EXIT IF p=0
2      LET t$=" ",t=p
       DO
         LET t=t-1
         LET p$=CHR$ PEEK t
       EXIT IF p$<"0" OR p$>”9”
         LET t$=p$+t$
       LOOP
3      IF LEN t$<>0 THEN PRINT "ALTERiny: ';t$
         LET x=VAL t$
         ALTER(x)  TO "VAL "+CHR$ 34+t$+CHR$ 34
4        LET p=p+1
       LOOP
       DPOKE 48804,1
   END FROC
```

The DEVAL procedure has many similarities to REVAL. A program search is done for VAL "#" (i.e. VAL of any quoted character) ; when this is found ALTER makes the change. This is done for VAL “##”, VAL "###" etc. up to 5 characters, using the outer FOR-NEXT loop.

```
5  DEF PROC deval
     DPOKE 48304,10
     FOR s=1 TO 5
       LET a$=”VAL “+CHr$ 34+STRING$(s,”#”)+CHR$ 34
         get_P
6      DO
         LET p=INSTRING(p,MEMORY$()( TO DPEEK(23627)),a$)
       EXIT IF p=0
         LET t$=””
         FOR n=0 TO S+2
           LET t$=t$+CHR$ PEEK (p+n)
         NEXT n
7        LET X=VAL t$ (3 TO 2+s)
         PRRINT "ALTERinq: ;t$
         ALTER (t$) TO (x)
         LET p=p+1
       LOOP
8    NEXT s
     DPOKE 48804,1
   END PROC

9  DEF PROC get_p
     LET p=DPEEK(23637)
   END PROC
```

While editing these PROCs, I discovered a bug in REF! I did REF " PRINT " and found it worked only once. This turns out to be because the line REF found also had an ALTER (reference) command in it. When I pressed ENTER, the syntax check of ALTER set some system variables up (e.g. a flag saying "ALTER, not REF" was set) and since these are also used by REF, I got a corrupted line where the next PRINT occurred. The routine needs to be altered to avoid touching the system variables during syntax checking, but I'll leave it for the moment. (Well, nobody has mentioned it in 3 years!)

****

DISC NEWS

SWIFT DISC

User Harry Paine (Foleshill, Coventry) reports that BB 3.0 and BB 4.0 seem to be reasonably compatible with the Microdrive Emulator that SixWord market with the Swift Disc. Programs, code and data LOAD, SAVE and VERIFY O.K., programs MERGE O.K. and normal disc PRINT and INPUT statements seem to work. The only thing that fails is the extended MOVE command.

DISCIPLE and PLUS D

Michael Williams (West Ealing, London) received his updated Disciple version of BB and reported no problems, except that he got double-spacing in his listings. This is because I wrote the print routine to do a line feed after carriage return, like the Microdrive "t" channel, whereas Michael's printer does an automatic line feed. POKE 54989,24 stops the Disciple version's line feed; POKE 54989,32 restores it.

The CAT_TO procedure in issue 9 works with the Disciple, provided you have the Disciple/PLUS D version I supply on tape. The catalogue format is more complicated than that for Discovery or Microdrive, and a bit harder to split up into separate file names, since the entries vary in length.

I am sorry to say I discovered a bug in some of the Disciple tapes I've sent out. The tapes incorporate a system to prevent disc errors turning off BB which can cause oddities during INPUT. More recent tapes, incorporating the EOF function, have been fixed. The older tapes can be fixed by the program below:

```
10  FOR n=50781 TO 50793: READ a: POKE n,a: NEXT n
20 DPOKE 54974,50787
30 DATA 207,11,205,185,24,43,254,18,200,225,195,197,214
```

****

"MISER" PROGRAM

Charles Buzard of 13, Grove Wood Close, Chorleywood, Herts. WD3SPU has devised a comprehensive Beta Basic program to update information on his portfolio of Stocks, Shares and Savings. If anyone is interested in having a copy, write to Charles and he will reply with full details about it. It is available only on microdrive cartridge.

****
A STRANGE THING HAPPENED TO ME WHILE BROWSING IN A NEWSAGENTS...

Not long ago I happened to look at (subscriber) Mike Tooley's ON SPEC column in Everyday Electronics magazine. Someone asked him if you could wire a Spectrum ROM into an Amstrad CPC computer. "Ha!" I thought; "no good - you'd have to re-write anything to do with the screen, sound or keyboard. Anyway, CPC's have pageable RAM in the lower 16K of memory, so there is no need for wiring - just load a copy of the Spectrum's ROM from tape!" So I walked off, muttering "Interesting problem..." and falling over waste bins.

Gripped by enthusiasm, I wrote routine to tapes onto the Amstrad, then a routine to "patch" a Spectrum ROM copy loaded in this way so it handled CLS and printing on the Amstrad screen. After a few other changes, when I set the "ROM" going, it set up all its system variables, printed the copyright message and sat there waiting for a keypress. Progress! Next I had to re-write the keyboard scan, which was harder, then the scroll routine... better get BEEP working too, oh yes, PLOT, DRAW, CIRCLE, BORDER. Now Spectrum Basic programs run O.K.! Alright, how about COPY to dot-matrix printers? Why not! Better be able to SAVE Spectrum format tapes - and use the disc system too! The Disc system is tough - it uses memory all over the place, and won't turn the drives off. Finally I get it working.

Next it occurs to me it would be nice if machine code programs that directly change Spectrum screen memory worked too... get *that* working, though with a speed penalty. By this time I have written about 3K of machine code, which means (with the CPC screen memory usage) that a CPC464 has about 20K free. Not really enough for running BB 3.0 - but on a CPC6128, the screen memory can be paged out, so almost the normal 41K could be used, if I re-write things a bit... About this time. I begin to realise, "This is taking a LONG TIME! I'm supposed to be finishing the PLUS 3 version of BB! Why am I writing this oddity? is it hysterical reaction to the PC game conversion I did last month? Can I actually sell this?" I can't sell Spectrum ROMs on tape, for copyright reasons, although all that is needed is a simple SAVE "name" CODE 0,16384. Users would have to provide their own copy. Perhaps some people want to run Spectrum Basic programs on their own or a friend's CPC? Hacker types will enjoy being able to alter the "ROM". Quite a lot of machine code routines should work, too - if I spend a bit more time, even BB 3.0 (on the CPC6128). However, before I do that, I'd better try a little market research - anyone interested? Comments please!

****
Making variables immune to CLEAR in Beta Basic 4.0

This tip is from John Luby of Duns, Berwickshire:
"RAM disc variables are immune from RUN or CLEAR. I find that it is worth-while having RAM disc arrays of one element, say DIM !sortflag(1), to hold important variables that cannot simply be re-initialised. This avoids cautioning database users, etc., not to use BREAK and RUN."

****
ALTERING THE ACTION OF THE PRINT COMMA

Normally, PRINT items separated by commas are positioned 16 characters apart, but sometimes a spacing of 8 would be more convenient. POKE 52185,248: POKE 52187,8 achieve this for any CSIZE *apart from zero*.

****

READERS LETTERS

Dear Andy,

I am using a Euroelectronics LPRINT III... 'The value 0 or 1 in location 60921 determines whether the control codes are sent unmodified or are dealt with by BB... I tried the value 0. Everything was fine until I tried LIST FORMAT 1 or 2. If there was a procedure call after a colon, first, a jump was made in the LLISTING before the procedure name. Second, the value in location 23681 was altered.

lain Rendall, Edinburgh

*(Iain is using location 23681 for a paged memory system.) If there is a leading space to a procedure name, BB does a backspace to delete it and preserve correct line-up in indented listings. This caused the problem, as the backspace character was handled by the ROM, and the ROM LPRINT routine uses 23681. (That location is labelled 'not used' in the Spectrum manual but this is wrong - it is the most-significant byte of the LPRINT ' position. This is usually 91; 91*256=23296, which is the printer buffer start.) Using backspace was a mistake, because many printer interfaces won't deal with this character. (I don't know why I didn't notice this before!) So let's delete the leading space another way by just avoiding printing it; the pokes below will do the trick for all versions of BB 3.0 or 4.0 (I think!).*

```
POKE 58567,19: POKE 58568,51: POKE 58569,51
```

Dear Andy,

I am fourteen years old and computer studies is one of the subjects I study at school. I use Beta Basic 4.0 a lot and it is much more valuable than any other piece of software I have got. One question I have got is how do you print on the bottom line using another character size? I've tried PLOT but that only prints on the second from bottom and PRINT #1 only prints using the normal character set. Also is there any way that I could obtain a copy of 'TURTLE' as there isn't one on my tape.

David Cowell, Romford, Essex.

Quite a lot of people seem to use BB as part of Computer Studies courses, which is very pleasing. Regarding your questions, I decided CSIZE should be inactive in the editing area, to prevent lock-ups when e.g. CSIZE 176 was selected for printed output. Both BB 3.0 and BB 4.0 allow PLOTed strings to "hang down" from a y-coordinate of 0, so that the text is in the top part of the editing area. This allows 64-column output in this area. BB 4.0 has an improved PLOT routine to plot points in the lower part of the screen, but I discovered when you wrote that I forgot to alter the PLOT strings command in a similar way. With Beta Basic 4.0 ONLY, the POKES below will do the job:

```
10 DPOKE 63486,63131
20 DPOKE 52609,63118
30 PLOT CSIZE 4,8;0,-1;"Top line"
40 PLOT CSIZE 4,8;0,-9;"Bottom line"
50 PAUSE 0
```

There should be a copy of "TURTLE" after BB 3.0 on your tape. If not, I'll send you one. I think a few people missed getting instructions for the program, too - if so, just send a S.A.E.

Dear Andy,

I sometimes get odd values for the looping line number of a FOR-NEXT loop when I LIST DATA. Why?

J. Kuiper, Amsterdam

For greater speed, BB doesn't actually use looping line numbers at all. Instead, it keeps a record of the address of the looping line, so that NEXT can loop back immediately, rather than searching down the program for the required line. LIST DATA therefore has to look at the specified address and read the line number from there. It you have altered the program since the FOR-NEXT loop was used, there may no longer be a line number at that address, and you will get odd results. Try entering and running a simple FOR-NEXT loop line; then delete the line and LIST DATA

Dear Andy,

I would like to write a procedure, using the SAVE command with a slicer, to save a program without the variables, but leaving the variables intact so that the program can continue running where it left off. This has led me to notice that it is impossible to save a program, with line 0 intact, using the slicer. Can this be rectified?

Robert Dickson, Blackheath, London

With both BB 3.0 and BB 4.0, POKE 54524,0 will allow slicer SAVES to include line 0. This also makes the default line for LIST line 0. POKE 54524,1 to return to normal.

Dear Dr. Wright,

I am a 17 year old 'A' level student, who uses the Spectrum and BB for programming. Could you recommend a good assembler/monitor program for m/c?

I used the following program:

```
10 INPUT INKEY$
20 GO TO 10
```

The key pressed is printed in the input line. Can you explain?

Kevin Allen, Nottingham

DEVPAC seems to be widely recommended, although I think it is pretty awful. (I use Pyradev on the CPC.) Regarding INPUT, if the command is followed by a simple variable name, the value of the variable can be entered. Anything else is treated like a print command for the lower screen, so that your line 10 acts like PRINT #1; INKEY$. Nothing is actually INPUTed, so INPUT doesn't wait for you. The lower screen is cleared after each "INPUT", so you just get a brief flicker. The command is designed to allow e.g. INPUT "size ";s but you can also use INPUT "" to simply clear the lower screen and reset the "Scroll?" counter. Try a FOR-NEXT loop printing the numbers 1 to 100, with an INPUT "" inside the loop.

****