BETA BASIC NEWSLETTER NO. 4

Hello - bit late again, I fear, but at least each issue is pretty "meaty" - I initially envisaged something like 4 pages per Newsletter!

Chris Nash (Bracknell, Berks.) and Daniel Ben-Sefer (Degania B., Israel) report that the readnumber procedure in issue 3 is not completely "idiot-proof", since it is possible to enter two decimal points. Daniel's solution is the shortest: change line 1090 to:

```
   1090 IF (a$>="0" AND a$<="9") OR (decimal AND a$="." AND
        INSTRING(1,b$,".")=0) THEN
           LET b$=b$+a$
```

He also sent a correction to PROC and from issue 2:

```
   7 LOOP UNTIL 1
      LET q=LEN f$-(LEN f$-q)*(0<INSTRING... etc.
```

Michael Charlton (who is a retired physics teacher from Pontefract) reported that PROC axes in issue 3 fails with certain values with Statement lost, 100:24. This is because BB with a version number less than 22 will not handle a FOR-NEXT loop of zero times, if the NEXT is on the same line as the FOR. The solution in this case is to put the NEXTs on separate lines.

Jesper Hertel (Vaerøse, Denmark) (by the way, I have sworn off my previous habit of referring to non-U.K. subscribers as living in e.g. (Norway) while U.K. subscribers live in places like (Milton Keynes) - it smacks of Anglo-centrism!) where was I... yes, he reported that FILL has no check on its Y coordinate, and that EDIT and INPUT do not respond correctly to the up and down cursor keys after ON ERROR has been active earlier in the same program run. So watch out!

Manlie Reeve is a 25 year old factory worker, making brass ware. He says he says less for each computer he buys, and he wonders how cheap they will get. I guess that from now on computers won't get much cheaper - they will just get better for the money - look at Amstrad's new IBM beater.

Bo Nordstrom.(Siriusgatan 66, S-415 22 Gothenburgh, Sweden) is a 31 year old Polytechnic student. He doesn't know anyone who is interested in computers, and he would like to correspond with anyone who is.

One thing that disappoints me is that despite the great variety of BB users, only a tiny number are female - I wonder why?

A number of subscribers have asked for news of a Spectrum 128K Beta Basic. Unfortunately, nothing much is happening on this front. This is for several reasons. First, the RAM paging on the 128 is very limited - it is only possible to switch the top 16K of memory. (In contrast, a PCW 256 can switch any of its 16K blocks into any quarter of the memory.) This is no great problem for a game, since the actual program is likely to be only about 8 to 16K long. It can sit in memory all the time, and switch in a new block of data when it is needed; for example, when a new room or level is reached. Now consider a Basic interpreter - ideally, one wants to give the user the maximum program and data area, so the best place to switch would be the lower 16K. You still have to do a bit of fiddling to call routines in the main ROM, but this is quite possible, as shown by Interface 1. However, if the Basic interpreter has to overlie the Basic program and data area in the top 16K, you've got problem. Every routine that reads or writes to part of the program or variable area will have to do something clever to allow it to work when it is switched out of memory (or appear to do so) his is quite possible, but it needs thousands of program hangs and a lot of tricky new programming, and the result will run rather slower than the original - in some cases, much slower Certainly it would be possible to squeeze BB into one 16K block by hiding some routines in switched RAM - but the improvement in available RAM isn't large.

In addition to feeling that the 123K machine is poorly designed, I see that the magazines for fairly cheap computers are concentrating more and more on games. More powerful computers are now quite affordable. So were does this leave a 128K Beta Basic? Selling mostly as an upgrade to a smallish pool of Spectrum users who are interested in programming and have bought a 128+ Spectrum I fear. Even a small ad costs about £280, so one wonders if the programming effort is worthwhile. I leave open one possibility - a version that is similar to BB 3.0 but allows use of the RAMdisc and perhaps sound.

In case you are wondering, Betasoft is still going strong, mostly as a programming subcontractor for larger companies with more marketing skill and less programming talent.

****

FAT CHARACTERS

No no, not fat people! I mean those chunky letters that many other computers use. Some of you might like to use such characters but don't fancy designing a new character set. In fact chunky characters can be produced by a simple doubling of eachpixel in the normal Spectrum character set. Below, the method is applied to the entire screen by line 50; the principle might be better applied to UDGs or GET screen blocks, but I leave that up to you.

```
   10 CSIZE 0
   20 FOR n=32 TO 127
        PRINT CHRS n;
      NEXT n
   30 PRINT
      LIST

   50 FOR n=16384 TO 22527
        POKE n,OR(PEEK n,PEEK n/2)
      NEXT  n
```

****
INTERACTIVE WINDOW DESIGN

This contribution from Dave Trebilcock (Cheshire) makes it much easier to work out WINDOW co-ordinates - it even create the required program line. Here's Dave's explanation:

The routine consists of three procedures:

WINDO	which does the main work
WDRAW	to draw an outline of the window
SETUP	to declare the finished window and add the declaration into the current program if desired.

A window is drawn in the middle of the screen which can then be manipulated using the cursor controls 5-8. Without CAPS LOCK the <u>size</u> of the window is adjusted, with CAPS LOCK the <u>position</u> of the window is adjusted.

The relevant co-ordinates are constantly displayed on the INFUT line and the current screen is unaffected as the WINDOW outline is drawn using OVER (PAPER colours may be affected though.)

PROC WINDO has 2 parameters -
NUM defines the WINDOW to be declared
PLIN is the program line to be declared (ignored if ZERO)

After positioning the window, pressing "E" will declare the window. The window declaration can be abandoned at any time by pressing "Q".

```
   20   DEF PROC windo num,plin
   30   LET x=88,y=119,w=40,L=40,z$="E5678"+CHR$ B+CHR$ 10+CHR$
        11+CHR$ 9+"Q",key=O
        DEFAULT num=l,plin=0
   40   DO UNTIL key=1 OR key=10
   50      wdraw
           PRINT #1;AT 0,0;"x = ";x;" y = ";y,"width = ";w
           ;" length = ";L;" "
   60      DO
              GET k$
              LET key=INSTRING(1,z$,SHIFT$(1,k$))
           LOOP UNTIL key
   70      wdraw
   80      IF key=2 AND # THEN LET w=w-B
   90      IF key=5 AND x+w<256 THEN LET w=w+8
   100     IF key=4 AND y+L-2>y+7 THEN LET L=L-8
   110     IF key=3 AND y-L>6 THEN LET L=L+8
   120     IF key=6 AND x>7 THEN LET x=x-8
   130     IF key=9 AND x+w < 256 THEN LET x=x+8
   140     IF key=7 AND y-L>6 THEN LET y=y-B
   150     IF key=8 AND y<168 THEN LET y=y+8
   160  LOOP
   170  IF key=1 THEN setup
   180 END PROC

   190  DEF PROC wdraw
   200     OVER 1
           PLOT x,y
           DRAW TO x+w-1,y
           DRAW TO x+w-1,y-L+1
           DRAW TO x,y-L+1
           DRAW TO x,y
           PLOT x,y
           OVER 0
   210 END PROC
```

```
  220  DEF PROC setup
  230     WINDOW num,x,y,w,L
          WINDOW num
  240     IF plin THEN KEYIN STR$ plin+" WINDOW "+STR$ num+","+ST
          R$ x+","+STR$ y+" , "+STR$ w+","+STR$ L
  250  END PROC
```

****
PROC cpy - 64 column text COPY

I mentioned in an earlier Newsletter that a SCREEN$ that recognised 4*8 pixel characters would be quite easy to write, and it would allow text screen dumps to a full-width printer. I actually meant that this would be easy to do in assembly language, but prodding from subscribers has induced me to write a Beta Basic version. The procedure assumes you are in CSIZE 4,8 to start with.

I used a slightly inelegant method that prints each of the 4X6 characters with a leading space at a (hopefully) convenient screen location. GET is used to pick up the data for this 8*8 block, and C$ accumulates the results. In the end C$ holds a complete character set, in the normal 8 bytes/character format. Now we need to alter the system variable CHARS so that SCREEN$ will look at the character set in C$ rather than the one in ROM - see line 1040. Note, however, that only 4*8 characters after a leading 4*8 space will be recognised. The loops starting at 1060 GET each of the characters on the screen and print them so that only one required half of the GET block falls into the position looked at by SCREEN$. The ON ERROR statement is s intended to return you to the normal character set if you press BREAK. Charles Buszard suggests that the 8 in line 1060 could be a variable allowing copying to cease at various points down the screen.

```
  1000 DEF PROC cpy
          LOCAL c$,g$,x,y,n
  1010    LET c$=""
  1020    FOR n=32 TO 127
             PRINT AT 21,0;" ";CHR$ n
             GET g$,0,7
             LET c$=c$+g$(2 TO )
          NEXT n
  1030    DELETE g$
  1040    DPOKE 23606,LENGTH(0,"c$")-256
  1050    ON ERROR 1120
  1060    FOR y=175 TO O STEP -8
  1070       FOR x=0 TO 252 STEP 4
  1080          GET g$,x,y
  1090          PLOT CSIZE 8;4,7;g$
  1100          LPRINT SCREEN$ (21,0);
  1110       NEXT x
             LPRINT
          NEXT y
  1120    DPOKE 23606,15360
  1130 END PROC
```

****

PROC plist - listing the names and locations of all procedures.

I have received a number of requests for a procedure (or new command) to do this, plus contributions from M.J. Smith (Lancs.) and A.M. Yarnell (West Midlands). Here is my own attempt, which I hope will prove useful.

The general idea is to look through the program (using MEMORY$, INSTRING$, and some DPEEKed system variables) for CHR$ 13's (the end of line marker) followed a bit later by DEF PROC (make sure you enter this as a keyword!). You cannot just look for DEF PROC alone, or you will find lines like LET x=129; the invisible five-byte form of 129 contains CHR$ 129, which is the token for DEF PROC... Having found a DEF PROC, get the line number out of the line header information, and read the procedure name from after the DEF PROC.

```
10   DEF PROC PLIST
        CLS
        PRINT "PROCEDURES:"
        PRINT
        LET A=DPEEK(23635)
20      DO
           LET A=INSTRING(A,MEMORY$()( TO DPEEK(23627)),CHR$ 13+
           "####"+" DEF PROC ")
        EXIT IF A=0
           LET N$=""
30         LET lnum=PEEK (a+l)#256+PEEK (a+2)
           LET a=a+5
40         DO
              LET A=A+l
              LET C$=CHRS PEEK A
           EXIT IF C$=CHR$ 13 OR CS-":"
              LET N$-N$+C$
           LOOP
50         PRINT NS;TAB 22; USING "####";lnum
60      LOOP
70   END PROC
```

****

BETA BASIC AND DISC SYSTEMS

Our sales and the postbag indicate that the OPUS Discovery disc drive is selling well to BB users - deservedly so, in my opinion. The standard BB can be transfered to OPUS disc and mostly works, but error handling, EOF, simplified LOAD/SAVE etc. and LPRINT will be faulty. For those who don't know, there is a special version, Beta Basic 3.0D, for this system, which solves the problems. It comes on tape, with instructions, for £2.00. (£3.00 outside Europe.)

We have some interesting OPUS-specific contributions but they may be too specialised to publish.

Francis Glassborow of Southfield Software has managed to get BB working with the SPDOS disc system. You can get the required paging program from him for (I think) £1.50. The address is:

SOUTHFIELD SOFTWARE, 64 Southfield Road, OXFORD OX4 1PA

****

PROC rotake - rotating characters by 90 degrees.

Michael Charlton (Pontefract, W. Yorks.) sent a sent of character manipulation procedures that included a "character rotate" facility - used twice, it gave an "Australian" character set, making for very odd listings! This worked fine but was rather Basic is not a fast language where bit manipulation of this sort is required. I was stimulated to write a machine code routine to turn characters by 90 degrees, which turns out to be useful U1 for a variety of purposes. (See PROC dump below.)

The source location of the 8 bytes to be rotated is determined by the first parameter. Normally, the results are put back into this location after the rotate, but a second parameter can be used to specify another destination. This is particularly useful if the source is the ROM character generator! (Note: this starts at 15616). The routine is automatically modified to work correctly if either source or destination is in screen memory, provided you do not straddle a character square boundary. Line 10 demonstrates rotation on of screen memory whereas line 20 prints all the UDG`s, rotate their definitions (not screen memory), prints them etc. till they are right way up again. Note that PROC rotate is an anti-clockwise rotate - the fastest way to get a clockwise version is to alter line 110 to contain three RANDOMIZE USR cd`s. Line 30 uses the ROM as a source and the screen as a destination for the rotated result.

The machine code is poked into the locations beginning at "cd" on the first call only, using the assumption that if the first byte is as expected, the poke must have been done already ! I have used the printer buffer, but you may prefer to locate the code above RAMTOP - the procedure "m/c" in this issue may be useful

```
10   PRINT "HELLO WORLD !"
     FOR n=16384 70 16400
        rotate n
     NEXT n

20   KEYWORDS 0
     FOR R=1 TO 4
        FOR n=144 TO 164
           PRINT CHR$ n;
        NEXT n
        PRINT
        FOR c=USR "a" TO USR "u" STEP 8
           rotate c
        NEXT c
     NEXT r
     KEYWORDS 1

30   rotate 15624,16384

60   DEF PROC rotate src,dest
        DEFAULT dest=src
        LOCAL n,a,cd
        LET cd=23296

70   IF PEEK cd<>33 THEN
        RESTORE 12O
        FOR n=cd TO cd+37
           READ a
           POKE n,a
        NEXT n
     80   DPOKE cd+1,scr
          DPOKE cd+29,dest
     90   IF src,16080 AND src <22528 THEN
             POKE cd+16,36
          ELSE POKE cd+16,35
     100  IF dest>16383 AND dest <22529 THEN
             POKE cd+34,36
          ELSE POKE cd+34,35
     110  RANDCMIZE USR cd
     120  DATA 33,0,0,30,128,14,8,6,8, 229,126,163,254,1,203,18,35
          ,16,247,225,213,203,11,13,32,237,6,8,33,144,128,241,47,
          119,35,16,250,201
     130  END PROC
```

****
PROC dump - graphics COPY to a dot-matrix printer

Even if you already have a printer interface that allows screen dumps of graphics, this procedure may be useful, since it can easily be modified to copy any part of the screen. You will need PROC rotate given earlier, as a sub-procedure. Make sure the location of the machine code doesn't conflict with your printer driving software. The main reason that screen dumps are usually done in machine code is that the printer requires the state of all the bit 7's of the first character to be sent as one byte, then-, all the bit 6's, etc. This is slow - but if you rotate the data for the first character anti-clockwise, all the bit 7's will fall into the bottom byte of data and it can be sent easily, followed by the bit 6's in the second from bottom byte, and so on.

Printer output should be going to a "b" type stream that will accept all byte values. Line 20 should set up the printer to advance by the length of eight dots at each line feed. Line 30 sets B$ to the characters needed to say: "256 bytes of bit-image data follows". I am using an EPSON RXSO - you may need to use different control codes for another printer. Usually, the last character in b$ will be "number of 256's" and the second to last will be "number in addition to the 256's" for the bit-image byte count. R and C in line 40 decide the row and column areas to be dumped. The expression after GET converts R and C to the co-ordinates that are required by the command. Line 5O should reset the line feed distance back to normal.

```
     10   DEF PROC dump
     15      LOCAL a$,b$,r,c,b
     20      LPRINT CHR$ 27;"A";CHR$ 8;
     30      LET b$=CHRS 27+"K"+CHR$ 0+CHR$ 1
     40      FOR r=0 TO 21
                LPRINT b$;
                FOR c=0 TO 31
                   GET a$,c*8,(21-r)*8+7
                    rotate LENGTH(0,"a$")+l
                   FOR b=9 TO 2 STEP -1
                      LPRINT a$(b);
                   NEXT b
                NEXT c
                LPRINT CHR$ 10
             NEXT r
     50      LPRINT CHRS 27;"A";CHRS 11;
          END PROC
```

The procedure takes about 9 seconds per line on the RX-80, or about 3 and a half minutes per screen.

****
PROC kill - unproved block delete.

Beta Basic DELETE command makes sure you know exactly what you are doing by checking that any line numbers you specify really exist Sometimes this can be inconvenient - for example, if you are loading "overlay" parts of a program at lines between 5000 and 7999, you might want to DELETE 5000 TO 7999 before MERGEing the new section. (This "overlay" technique allows a program to be cut into chunks and it is common in large business program - BB can use the memory saving too! Subscriber Charles Buszard has been exploiting it the method.)

To avoid – "No such line" errors, use this simple procedure:

```
100  DEF PROC kill f,l
        KEYIN STR$ f+" REM "
        KEYIN STR$ l+" REM "
        DELETE f to l
     END PROC
```

Note: The kill procedure, or a DELETE command, should be higher in the listing (at smaller line numbers) than the lines it deletes.

****
REMOVING EXTRA XOS, YOS, XRG AND YRG COPIES.

Users of the TRL Disc Interface and some other users may have to "turn on" Beta Basic more than once, because their interface causes Beta Basic to lose control of the system after LOAD and MERGE commands. This can be done by RANDOMIZE USR 58419. This USR routine sets up the special variables XOS, YOS, XRG and YRG each time it is used. This can create multiple copies which accumulate if CLEAR or RUN are never used. (LIST DATA will show if this has happened.) They can be eliminated by the routine below, without loss of any other variables.

```
10   REM get rid of extra xos, etc. A$ can be in use already
20   IF PEEK (DPEEK(23627)+32)<>184 THEN STOP
30   POKE DPEEK(23627)+32,65
40   POKE DPEEK(23627)+33,29
50   POKE DPEEK(23627)+34,0
60   DELETE a$
70   GO TO 20
```

NOTE: You can use POKE 53754,201 to prevent RANDOMIZE USR 58419 (and RUN and CLEAR) from creating the special variables - but you may have problems if you later use RUN or CLEAR; you should keep at least one copy of the special variables.

****
LLISTINGS.

BB versions after 20 (PEEK 47272 for version no.) allow BB to control line length and indent long program lines correctly to (I hope) any printer interface. Location 57503 controls line length and it should be poked to a value less than or-equal to the line length Your printer or interface would otherwise produce. The initial value is 80 - if you are using a ZX-type printer, POKE 57500,32.

****

PROC chardes - character designer

In Newsletter 1 I suggested that a Beta Basic character designer would be useful, and several subscribers sent creditable contributions, for example, Bo Nordstrom (Gothenburgh, Sweden). I also wrote one myself, which is slightly better in some ways; it is listed below. It is not completely error-trapped, and the various component procedures could be usefully made more independent by using more parameter passing, but I hope it will form a useful source of ideas.

The main procedure, CHARDES, accepts input and calculates the location of the desired character pattern (adr) and the number of bytes in it. It copes with either UDG's or BB's small character set. Next the current data is displayed as black or yellow spaces, using PROC display. PROC ed lets you move a cursor around the data, using the cursor keys. The DELETE key reverses the colour at the cursor position. ENTER when you are finished. Now PROC scread reads the data from the screen and places it as binary into a string array, and finally PROC pokeback replaces the modified data at the original location.. The slightly unconventional approach of reading the data from the screen means that PROC ed does not have to worry about changing both screen and data, which makes it faster.

PROC ed may have general uses, so I will describe it in some detail. The parameters ll,rl,tl and bl are Left Limit, Right Limit, Top Limit and Bottom Limit for cursor travel. (I actually made no use of the variability of these parameters, and the other bits of PROC chardes are less flexible, but it should simplify other uses of the procedure.)

The procedure uses PAPER 8 so that printing does not alter the paper colour, and INK 9 to ensure the cursor shows up.

The ON section is a fast way of responding to various keys, and needs some explanation. What's this ON k-b OR k<B OR k>13? Well, an expression like k>13 is evaluated as either 1 (true) or 0 (false). A number OR 1 gives 1; a number OR 0 gives the original number. (This is slightly peculiar to my machine codeish brain - the BB bit-by-bit OR seems more obvious - but that's the way it is.) The overall effect is that if k is outside the limits 8 to 13, the expression after ON evaluates to 1, whereas if k is inside the limits, the result is k-b - namely, 2 to 7. Therefore an illegal value for k always goes to the first statement in the ON list, which could be just a colon if you like. The other statements correspond to characters 8,9, 10,11,12 and 13, which are left, right, down and up cursors, DELETE and ENTER. Different keys would be more convenient if you have a rubber key Spectrum.

The logical expressions in the LET statements after ON give a move of one square if "limit not exceeded" is TRUE. The PRINT expression corresponding to DELETE uses a bit of fiddling to get the PAPER attribute for the square with the cursor. If it is e already, it will be 0 next time, and vice versa. The EXIT IF 1 corresponds to the ENTER key, and it will cause an unconditional jump out of the DO-loop.

```
10   DEF PROC chardes
       LOCAL a$,c$,n,b,adr,c,r,s
20     PRINT "UDG or Small characters? (U/S)"
       DO
         GET a$
         LET a$=SHIFT$(1,a$)
       LOOP UNTIL a$="U" OR a$="S"
30     PRINT "Character?"
       GET c$
40     IF a$="U" THEN LET c$=SHIFT$(1,c$)
50     IF a$="U" THEN LET s=8
         LET adr=65368+(CODE c$-65)*s
60     IF a$="S" THEN LET s=7
         LET adr=51291+INT (CODE c$/2-16)*s
70     display
80     ed
90     DIM a$(8,8)
100    scread a$
110    poKeback
120   END PROC

130  DEF PROC display
       CLS
       FOR n=adr TO adr+s-1
         FOR b=1 TO 8
           PRINT PAPER 6-VAL (BIN$(PEEK n)(b))*6;" ";
         NEXT b
         PRINT
       NEXT n
     END PROC

140  DEF PROC ed ll,rl,tl,bl
150    DEFAULT 11=0,rl=7,tl=0,bl=7
160    LOCAL r,c,k$,k
170    LET c=ll,r=tl
       PRINT AT r,c; PAPER 8; INK 9;"*"
180    DO
190      GET k$
         LET k=CODE k$
         PRINT AT r,c; PAPER 8;" "
         ON k-6 OR k<8 OR k>13
           BEEP .1,1
           LET c=c-(c>ll)
           LET c=c+(c<rl)
           LET r=r+(r<bl)
           LET r=r-(r>tl)
           PRINT AT r,c; PAPER 6-AND(BIN 111000,ATTR (r,c))/8;
           " "
         EXIT IF 1
200      PRINT AT r,c; PAPER 8; INK. 9;"*"
       LOOP
     END PROC

210  DEF PROC scread REF a$
       FOR r=0 TO 7
         FOR c=0 TO 7
           IF AND(BIN 111000,ATTR (r,c))=0 THEN
             LET a$(r+l,c+l )="1"
           ELSE LET a$(r+1,c+1)="0"
220      NEXT c
       NEXT r
     END PROC
230  DEF PROC pokeback
       FOR n=1 TO s
         POKE adr+n-1,VAL ("BIN "+a$(n))
       NEXT n
     END PROC
```

****
THE SPECTRUM BASIC MERGE BUG.

Yes, there is one! BB users now and then mention crashes when MERGEing large programs when memory is tight. This is due to a fault in a ROM routine called (by Dr. Ian Logan) RECLAIM. It shuffles memory about as variables and program lines are moved from a temporary storage area into the program area. (This can cause a delay of many seconds sometimes.) Unfortunately, RECLAIM shuffles a little too much memory, and this can cause the stack to be over-written if a large line or variable is moved and memory is tight. This has nothing to do with Beta Basic, apart from its memory consumption, but be warned! (I know of no way to check if a crash will occur in a particular case.) This bug seems to be little known - I only found it myself when I did a byte-by-byte comparision of the debugged Scandinavian Spectrum ROM and the normal ROM.

****
PROC adr - getting the screen address of a pixel

Sometimes one wants to know the address in memory of a particular point on the screen. This is easy to do in machine code, using a ROM subroutine, but difficult and slow in Basic. PROC adr constructs a short machine code routine in a$ which consists of: "put co-ordinates in the correct register: call ROM routine: put screen address in a convenient register: return" The code in the string can itself be called, using:

```
        LENGTH (0,"a$")
```

to find its location. The screen address comes back, from the procedure in a variable you specify, such as "slot". You might find this PROC useful (after modification to work with rows and columns) with PROC rotate in this issue.

```
     10   adr 100,100,sloc
          PLOT 100,100
          PAUSE 100
          POKE sloc,255

     20   DEF PROC adr x,y, REF a
            LOCAL a$
            LET a$=CHRS 1+CHR$ x+CHR$ y+CHR$ 205+CHR$ 170+CHRS 34+C
            HR$ 68+CHR$ 77+CHR$ 201
            LET a=USR LENGTH(0,"a$")
          END PROC
```

****

PROC paglist - paginated listings

G.J. Doodson (Telford, Shropshire) wrote to see if I could suggest a method of paginating listings at desired points. I tried various control codes inserted into the listing in an effort to force pagination, but had no success. Instead, I modified PROC slist from issue 2. The resulting PROC paglist looks through the listing and paginates if it finds "page" at the start of a line. (INSTRING is used so that you can use a character before "page", or not.) Page is a dummy PROC that simply acts as a marker. Alternatively, you could use:

```
        REM page
```

PROC paglist comes in two forms, depending if you want to list to a printer or the screen. The version below is for printers that respond to CHR$ 12 by doing a form feed. Stream 3 should be open for text output, and stream 4 for "b" type (any codes) output. To work with the screen, alter line 20 to:

```
        IF etc............ THEN
           PAUSE 0
           CLS
        ELSE LIST lnum-1 TO lnum
```

```
   10   DEF PROC paglist st,end
           DEFAULT st=1,end=9999
           LET x=DPEEK(23635)
           DO
              LET x=x+DPEEK(x+2)+4
              LET lnum=PEEK x*256+PEEK (x+1)
           LOOP UNTIL lnum>=st
   20      DO UNTIL lnum>end
              IF INSTRING(1,MEMORY$()(x+4 TO x+8),"page") THEN
                 PRINT #4;CHR$ 12;
              ELSE LLIST lnum-1 TO lnum
   30         LET x=x+DPEEK(x+2)+4
              LET lnum=PEEK, x*256+PEEK (x+1)
           LOOP
        END PROC

   40   DEF PROC   page
        END PRCC
   70   page
   80   REM PAGEFUL OF STUFF
   130  page
   140  REM NEXT PAGE OF STUFF
   150  PRINT
```

Note the spaces between DEF PROC and page in line 40 - these prevent this instance of "page" being found, as it will be too far from the start of the line.

****

PRCC end - for those who hate STOP

Dr Alain Vezes (Albi, France) sent a collection of procedures. He doesn't like STOP statements in programs, he says. PROC end can be used instead of STOP. It will cause a "Program finished" or an "OK" message.
Comment: you had better use a CLEAR or a RUN now and then, or the stack will get cluttered with PROC return addresses.

```
   100 DEF PPOC end: GO TO 1e4: END PROC
```

****
PROC lplot -- low resolution PLOT

Dr. Vezes sent two procedures to emulate the PLOT and UNPLOT of the ZX81, for nostalgics! The smaller number of points also means crude shapes can be drawn quite fast. Dr. Vezes used, repeated DRAWS to make the large pixels, but the procedure below uses UDGs instead, as this is faster. UNPLOT can be emulated b'1 makinq the IN" colour the same .is the background colour when the PROC is called. "ALTER" is entered as a keyword.

```
     10   FOR t=0 TO 42
             lplot t,t
          NEXT t

     20   DEF PROC lplot x,y
     30      IF PEEK USR "a"<>240 THEN
                LOCAL n,x
                RESTORE 30
                FOR n=USR "a" TO USR "a"+7
                   READ x
                   POKE n,x
                NEXT n
                DATA 240,240,240,240,0,0,0,0
     40      KEYWORDS 0
             PLOT OVER 2;4*x,4*y+3;" ALTER "
             KEYWORDS 1
     50   END PROC
```

****
PROC m/c - storing machine code above RAMTOP

This contribution is from Ettrick Thomson (Aldeburgh, Suffolk). We have corresponded quite a bit about the problem of loading machine code above a RAMTOP that differs according to the DEF KEYS and WINDOWS you have created. Although CLEAR with a number less than 768 will create space above the KEY/WINDOW definitions by moving them (and RAMTOP) down, the location of the space will still vary according to which version of BB you have, and it will be different if you have already placed some code in this area. The solution is to create space with CLEAR, and then look through the KEY/WINDOW data, starting just above RAMTOP, for the zero end marker. The next location is the start of the space you have just created.

Ettrick has incorporated this method into a procedure that takes a variable name and the space required as its parameters. The space is created, found, and the code is READ and POKEd into it. The variable name you give is set equal to the address of the code, so that e.g. RANDOMIZE USR *name* can be used to excecute the code. Remember that RUN will clear this value - you might rant to print it and then use it in a LET statement. Note also that running this example multiple times will insert multiple copies of two code routines above RAMTOP. This will work, but it may not be your intention.

The example prints RAMTOP i.e. DPEEK(23730) as two code routines are poked into place. The first one is Ettrick's fast square root routine - it is about seven times faster than the normal SQR and more accurate. It is demonstrated at line 100. The other routine is just a dummy that returns 123 - it is executed at line 90 to prove that several routines can be poked into place and still work!

```
  10   RESTORE
  20   PRINT DPEEK(23730)

  30   m/c sq,33
  40   DATA 239,61,192,56,6,5,126,167,200,198,128,31,119,35,126,
       23,48,2,207,9,54,127,239,49,224,1,5,15,56,53,16,246,201

  50   PRINT DPEEK(23730)

  60   m/c dy,4
  70   DATA 1,123,0,201
  8O   PRINT DPEEK(23730)
  90   PRINT dy,USR dy
  100  INPUT x
  110  PRINT x,x AND USR sq
  120  GO TO 100

 9900  DEF PRROC m/c REF s,l
          LOCAL a,b
          CLEAR l
          LET s=DPEEK (23730) +2
          DO UNTIL PEEP (s-1)=0
             LET s=s+3+DPEEK(s)
          LOOP
          FOR a=s TO s+l-1
             READ b
             POKE a,b
          NEXT a
9910   END PROC
```

Ettrick suggested that the procedure could be easily modified to LOAD the code from tape or Microdrive, rather than from a DATA statement. If you stick to DATA statements, you might be able to use ITEM to detect the end of the data and avoid the need for knowing the number of bytes.... hmmm, you wouldn't know the space to CLEAR until you had read all the data... perhaps you could accumulate the bytes in a string, then do the CLEAR required for its length, and POKE the string into the correct place...

****

TOWERS OF HANOI REVISITED

Apart from PROC m/c, Ettrick Thomson also sent a graphically pleasing version of "Towers of Hanoi" (see issue 3) and suggested a faster version of PROC hanoi which has relevance to recursion in general. This part is listed below it won't work on its own). Ettrick points out that the faster recursion is stopped, the better; the version in issue 3 stopped at n=0, whereas the version below stops at n=1. Calls to hanoi with n=0 do nothing but add to the running time, so it is better to take special action when n=1 (the ELSE part) and avoid further recursion.

```
9000 DEF PROC hanoi A,B,C,n
        IF n>1 THEN
           hanoi A,C,B,n-1
           move_ring n,A,B
           hanoi C,B,A,n-1
        ELSE
           move-ring 1,A,B
9010 END PROC
```

****

READERS' LETTERS ETC.

Everyone who reads the computer magazines knows that it isn't sufficient just to answer letters and confuse a lot of people to be considered a computer expert - you need a high-contrast picture of yourself at the top of your column, preferably one that makes you look like a zombie. So, using a screen image transfered from a SEC and PROC dump from this issue, I've prepared something suitable. From now on, I'll be a proper expert - if I don't know something, I'll make it up! (Well I think some of them do...)

****

OCP's +80 ADDRESS MANAGER

Charles Bustard has come up with some improvements on the above program which he offers to anyone interested in using it with Microdrive. Contact him at:

"Thirteen" Grove Wood Close, Chorleywood, Herts. WD3 5PU

****

CAPS LOCK DETECTION

Ian Flinn (Purley, Surrey) wanted to know how to detect the caps lock state from inside a program (allowing a word-processing routine to display the status - Ian varied the BORDER colour.) I suggested using AND PEEK(23658,8) which returns 0 for lower case and 8 for capitals.

****

CHR$ 15 (ENTER-like control code)

Dr. Alain Votes (Albi, France) has some problems with printing strings containing CHR$ 15 because a carriage return +TAB 5 is produced. There is some conflict between what it is desirable for CHR$ 15 to do in listings and in a PRINT statement - the former needs a TAB for correct indentation. However, if you want to yet just the carriage return, make these POKES:

```
POKE 47706,62: POKE 47707,13: POKE 47709,201
```

(Why not use a CHR$ 13 - return - in the first place? Well, you can't enter that in the middle of a string with INPUT or EDIT.)

****

David Warne of Chester wrote to complain that the POKEs on page 1 of Newsletter 3 cause more problems than they correct. (Thanks David!) An improved version is:

```
DFOKE 60082,63676: DPOKE 60088,63678
DPOKE 56097,63676: DPOKE 56100,63678
```

****

Dear Dr. Wright,

I should like to be able to send the real-time clock display elsewhere, rather than the top left hand corner of the screen... when editing and debugging a program I find it difficult to read the clock as there is no border between the clod: and the listing. Also the clock tends to scroll down the screen with the listing, corrupting it.

Does anyone have a PROC for editing large strings? With normal edit there is a warning "rasp" and it is very slow as it rebuilds the screen. Any ideas? I want to generate a simple word processor, or text editor.

I feel that going from A4 size was a mistake as it is now more awkward to file the Newsletters. Not only that the print is half size, is this a cost cutting exercise? No matter thanks for Beta Basic, you've done many Spectrum owners a big favour. Keep up the good work.

Martin Reed, Gloucester

Thanks for the encouragement! You can move the CLOCK display by poking some locations: 61314 and 61318 contain the least and most significant bytes of the screen address of the top right hand pixel of the time display. Because of the complex way the screen is mapped, you may have to think a bit to get the position correct! (well I just corrupted my program...) Location 61314 normally holds 31, and 61318 holds 64. Poking values between 7 and 255 into 61314 will make the display occupy all the main print positions in the top third of the screen. Poking 61318 with 65 or 66 will lower the display be one or two pixels; poking it with 72 or 80 will move the display to the middle or bottom third of the screen. POKE 61314,255: POKE 61318,80 will move the display to the bottom right of the screen, in the INPUT line, which might be a convenient location.

I know several users have written text editors, but I don't seem to have the right scrap of paper to hand... it tends to be slow work inserting or deleting if you have to reprint the entire screen. Perhaps you could use something entirely screen Cased, using cursor control codes as in the GET key example in the manual. You could open up or delete new lines or text spaces with ROLL. When the screen was finished, you could use the slow SCREEN$ method to move it to a string, perhaps doing some processing (get rid of extra spaces?) on the way. You can use the method in PROC cpy (in this issue) to process a 64-column screen. PROC prtstr in issue 3 might come in handy too. All this is just off the top of the head stuff - what do the rest of you think?

Opinions on the AS size Newsletter seem to be split down the middle - as were the printing costs! The Newsletters have been much longer than I anticipated, so this is a valuable saving. I am sorry for the filing problem.

****

LAST WORD

The readers letters got squeezed a bit this issue - I actually had to chop off a couple of prepared pages. (They have already been replied to personally, of course.) More next issue.

BETASOFT 92 OXFORD ROAD, MOSELEY, BIRMINGHAM, B13 9SQ, ENGLAND