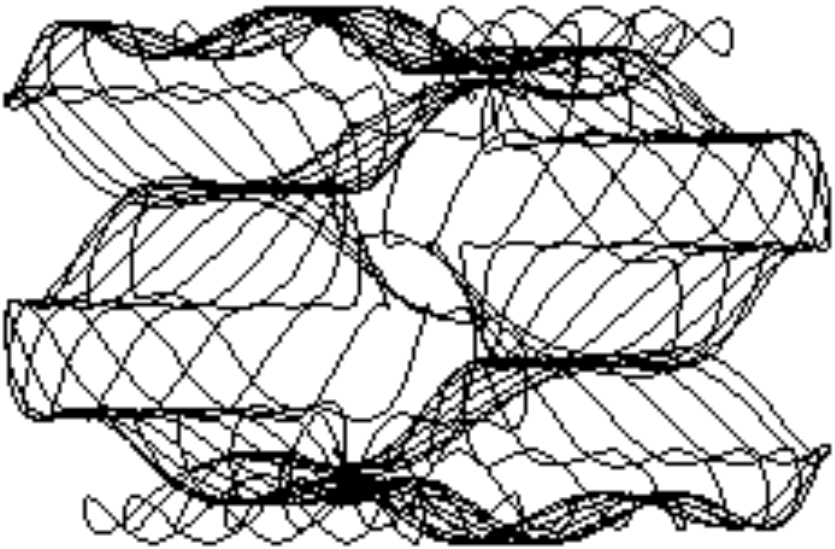
****

"MADNESS"

This routine is either a graphical doodle or a plot of a complex mathematical curve, depending on your prejudices! It was inspired by an article in *Scientific American's* Computer Recreations column; the expressions the program uses in drawing are similar to some discovered accidentally by an American, Stanley S. Miller. I have used DRAW TO, rather than PLOT; the display is therefore made up of short straight lines. This improves speed no end, while still giving very acceptable results.

```
10 LET s=50
20 LET XRG=3.8,YRG=2.6,XOS=XRG/2,YOS=YRG/2
30 LET T=-s
40 PLOT SINE(.98*T)-.7*COS (3.05*T),COSE(1*T)+.1*SINE(13*T)
50 FOR T=-s TO +s STEP .05
60  DRAW TO SINE(.98*T)-.7*COS (3.05*T),COSE(1*T)+.1*SINE(
      13*T)
70 NEXT T
```



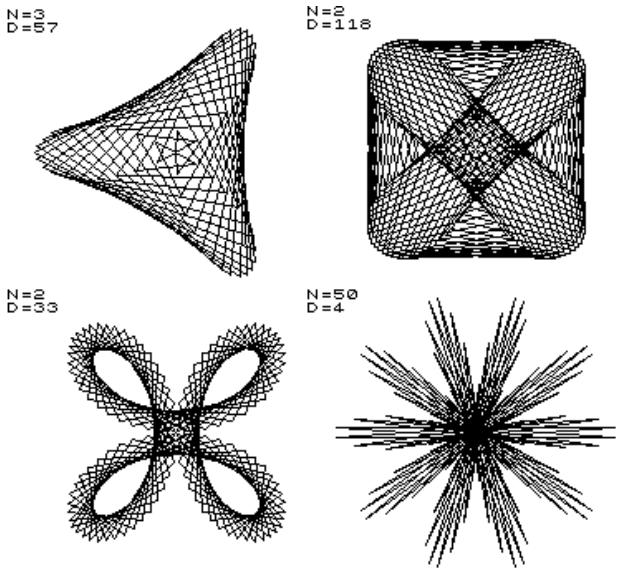
****

"THE ROSE" - a pattern making program.

The algorithm (basic method) embodied in this program was recently made public by mathematician Peter Maurer; I got hold of it via Keith Devlin's *Micromaths* column in Computer Guardian. The algorithm is called "The Rose" because many of the patterns it produces have a flower-like appearance. It has been used in graphics demos on high powered machines. My adaptation uses a LIM variable to reduce time-wasting re-drawing of the same lines due to rotational symmetry. N and D should both be whole numbers between 1 and 359.

```
 10 LET XRG=3,YRG=XRG*176/256
 20 LET XOS=XRG/2.5,YOS=YRG/2
 30 DO
 40   INPUT "N:";N
 50    INPUT "D:";D
 60   PRINT "N=";N'"D=";D
 70   PLOT 0,0
 80   LET A=0,OX=0,OY=0,LIM=0
 90   IF MOD(N,2)=1 THEN LET LIM=180
100   DO
110     LET A=MOD(A+D,360)
120     LET R=SIN (MOD(N*A,360)*.01745)
130     LET T=A*.01745
140     LET NX=R*SIN (T),NY=R*COS (T)
150     DRAW TO NX,NY
160     LET OX=NX,OY=NY
170   LOOP UNTIL A=LIM
180   BEEP .1,0
      PAUSE 0
190   CLS
200 LOOP
```

(The triangular pattern rubs in the fact that my printer dumps are slightly distorted - it should be completely symmetrical!)



****

PROC TASLIST - converting programs to Tasword 3 files.

This contribution from G.R.J. Baldwin is an improvement to one published in Newsletter 5. The author writes:

"The problems [with the original] were, firstly that we need to use JOIN instead of LET a$=a$+... otherwise long programs cannot be converted as there is not enough memory (this is explained on page 36 of the Beta Basic 3.0 manual). Secondly, when the tokens are expanded, the resulting line can be quite a bit longer than the normal 64 character screen-width in Tasword, which makes editing tedious. This can be corrected by chopping the expanded line into 64-character chunks, each with its own carriage return/line feed, before adding it to a$. I've added the little luxury of asking for a filename for the saved file, and removing the ERASE, assuming that the file is being saved for the first time."

```
9980 DEF PROC taslist
9981   CLEAR
       INPUT "Tasfile name? ";n$
9982   LIST FORMAT 2
9983   OPEN #5;"m";1;"temp"
       LIST #5; TO 9979
       CLOSE #5
9984   OPEN #5;"m";1;"temp"
       LET a$=""
9985   DO UNTIL EOF(5)
9986     INPUT #5; LINE z$
9987     LET z$=SHIFT$(7,z$)
9988     IF LEN z$>64 THEN LET y$=z$( TO 64)
           LET z$=z$(65 TO )
           LET b$=y$+CHR$ 13+CHR$ 10
           JOIN b$ TO a$
           GO TO 9988
9989     LET z$=z$+CHR$ 13+CHR$ 10
         JOIN z$ TO a$
9990   LOOP
9991   CLOSE #5
9992   ERASE "temp"
       OPEN #5;"m";1;n$
       PRINT #5;a$
       CLOSE #5
9993 END PROC
```

Comments: I had also run across the need for dealing with long lines. These occur because when the program is listed to a file, there is no "screen width" and so long lines (e.g. DATA statements) never have carriage returns inserted. The use of JOIN is a good idea for longer programs. The CLEAR in line 9981 results in a "Missing DEF PROC" report when line 9993 is reached - but Mr. Baldwin must have needed all the memory he could get! I use a version that indents most program lines; the line equivalent to 9987 is altered slightly to achieve this:

```
LET z$="   "+SHIFT$(7,z$)
```

****

PARAMETER PASSING USING REF

Judging by my correspondence, quite a few of you are sometimes confused by parameter passing, particularly passing by reference. This is natural, I think; some difficult concepts are involved, and they are not easy to explain. However, I will try to clarify a few points.

Why use REF parameters?

If a procedure is to calculate a result, you can just use e.g. LET result=(something) inside the procedure, and then use result outside the procedure without any problems. No REF parameters are needed. The only problem is that you cannot use that procedure in any future program without looking at it and realising that the answer is going to be in the variable result - which you might have used already. However, if you use REF in the DEF PROC, you can "tell" the procedure that the main program will supply the name to be used; this allows you to write a procedure to deal with any numeric or string variable. For example, a procedure could convert a Fahrenheit temperature held in a variable to a Celsius temperature in the same variable, whether it was called temp, x, fh or zzz:

```
 10 LET zzz=98.4
 20 FCconvert zzz
 30 PRINT zzz

100 DEF PROC FCconvert REF temp
      LET temp=(temp-32)/1.8
    END PROC
```

Of course, the main program *must* supply a name - if e.g. 123 is given in the procedure call, there is no variable to put the result in!

It doesn't matter if the name after REF is used in the main program - it is automatically LOCAL to the procedure. (In other words, if the variable is in use, its value will be saved before the procedure is used, and restored afterwards.) If you want a procedure to "plug in" to any program, you should make sure that all variables are LOCAL. Parameters are automatically LOCAL; other variables need a LOCAL statement naming them. If you follow these rules, it doesn't matter what variable names the procedure uses; there will be no conflict with the rest of the program.

You may notice that not everything I write obeys these guidelines! The truth is, I often don't need a procedure to act as a plug-in unit; once it works, I might not always bother to tidy it up. (Also, parameter passing and LOCAL do slow execution, and you might not always want to use them for that reason.) Perhaps it is better that I produce another imperfect idea rather than spend a long time polishing the current one... Besides, it is more fun! If you readers are sparked to modify or improve things in the Newsletter, I am well pleased.

****
PROC FACT - calculating factorials.

Dave Swales (Gainsborough, Lincs.) writes:

"The enclosed... finds the factorial of a number. Briefly, a factorial is the product of all the positive integers from 1 to a given number. E.g. 4 factorial is the product 1*2*3*4=24"

(Such things are involved in calculating, say, how many 7-letter arrangements you can make with a set of 7 Scrabble tiles. For the first letter you have 7 choices, for the second 6, etc. The number of arrangements is 7*6*5*4*3*2*1=5040.)

```
9999 DEF PROC fact x
       LOCAL a(),b,c
       DIM a(x)
       LET b=1
       FOR c=1 TO x
         LET b=b*c,a(c)=b
       NEXT c
       PRINT x;" Factorial = ";a(x)
     END PROC
```

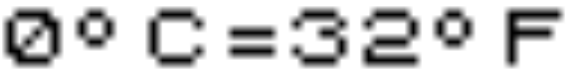
I seem to remember there is a non-BB way to do this using a cunning recursive function definition, but I can't quite reconstruct it!

****
PROC DEGREE - defining a degree symbol

This contribution from A.E. Legat configures any specified UDG as a useful degree symbol. My demonstration lines show UDG "A" being altered. You could also have used: 10 DEGREE" ALTER" with ALTER got by typing GRAPHICS-A. The ALTERs in line 30 need to be typed as GRAPHIC-A. Below the listing is a "blown-up" copy of the output.

```
 10 DEGREE "A"
 20 KEYWORDS 0
 30 PRINT "0 ALTER C=32 ALTER F"
 40 KEYWORDS 1

100 DEF PROC DEGREE X$
      LOCAL A
      LET A=USR X$
      POKE A,STRING$(8,CHR$ 0)
      DPOKE A+1,18480
      DPOKE A+3,12360
    END PROC
```



****

PROC WIND0 - switching between multiple windows

Contributor David Swales (Gainsborough, Lincs.) writes:

"I chose the title wind0 to distinguish it from PROC windo in Newsletter 4. One of the problems when debugging programs is having to scroll up and down the listing. With PROC wind0 it is possible to show parts of a listing in any of four windows. It is then fairly easy to follow the flow of the program."

"The definitions of keys 1 to 0 are displayed in the header. Part of a program can be listed in one window, and the program run in another. Here is an explanation of the keys:"

```
 1 w1.........Window 1 current window (LH 1/2 of screen)
 2 w2.........Window 2 current window (Top RH 1/4 of screen)
 3 w3.........Window 3 current window (Bottom RH 1/4 of screen)
 4 w4.........Window 4 current window (Full screen - 20 lines)
 5 CLS........Clear current window
 6 LIST.......List to current window (Scroll? appears as normal)
 7 LIST x,y...List specified lines to current window
 8 RUN x......Run program from a specified line
 9 INIT.......Call PROC wind0 (only after keys are defined)
 0 NORM.......Return to normal (Window 0 - CLS - CSIZE 8)
```

The key definitions certainly make the windows easy to switch between. Some may feel that key 8 is not really necessary. The seemingly pointless key 9 actually is needed if the program you are debugging alters the windows. The WINDOW ERASE associated with key 0 could be changed to WINDOW 0 - this would have the advantage that· you could switch back to say, window 1 without having to use key 9 first. The possible disadvantage is that the existence of 4 windows might alter the operation of the program you are debugging. (Note: For some reason, WINDOW ERASE requires a CLS before the next PRINT to avoid a spurious error message... I must have goofed up somewhere!)

```
9000 DEF PROC wind0
       LOCAL x,y
       DEF KEY "1";" WINDOW 1: CSIZE 4,8"
       DEF KEY "2";" WINDOW 2: CSIZE 4,8"
9010   DEF KEY "3";" WINDOW 3: CSIZE 4,8"
       DEF KEY "4";" WINDOW 4: CSIZE 4,8"
       DEF KEY "5";" CLS "
       DEF KEY "6";" LIST "
9020   DEF KEY "7";" INPUT x,y: LIST x TO y"
       DEF KEY "8";" INPUT x: RUN x"
       DEF KEY "9";"wind0"
       DEF KEY "0";" WINDOW ERASE : CLS "
9030   WINDOW 1,0,165,122,165
       WINDOW 2,130,165,122,82
       WINDOW 3,130,83,122,83
       WINDOW 4,0,165,255,160
9040   CSIZE 4,8
       CLS
       PRINT "1 w1:2 w2:3 w3:4 w4:5 CLS:6 LIST xy:8 RUN x
       9:INIT 0:NORM"
       PLOT 0,166
       DRAW TO 255,166
       WINDOW 4
9050 END PROC
```

****
ORBITS - simulating spacecraft behaviour.

I had been vaguely aware for some years that spacecraft behaved in a non-obvious way; for example, if a rocket orbiting the Earth (or anything else, for that matter) fires against its direction of movement, it slows down briefly, but then drops into a lower, faster orbit. And firing the rocket straight towards the planet in an attempt to reach a higher orbit is the wrong thing to do - you need to fire it behind you. Anyway, recently I decided to try to write an orbit display program to help give some sort of "feel" to these ideas. The result is no arcade game, but it is reasonably accurate, I think.

Line 10 sets the graphics origin to the centre of the screen; the "planet" will be placed here. (Actually, the small "planet" size I have used means that the orbits are very distant ones compared to real satellite orbits - better pretend it is the sun we are orbiting!) Lines 20 and 30 set the spacecraft's initial position, and velocity in the horizontal and vertical directions.

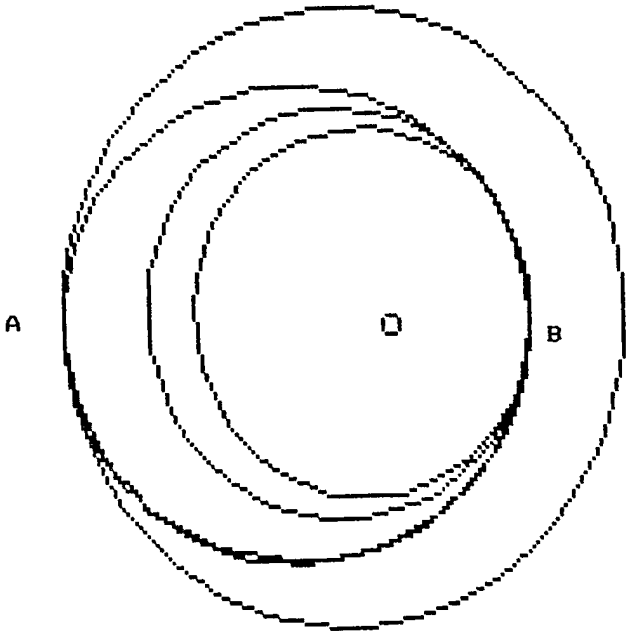
Lines 70 and 160 provide a primitive manoeuvring system; if you press a cursor key, your rocket fires in the direction of the cursor arrow. (So you will have to press a different key to brake if you are in a different part of the orbit - sorry about that...I got fed up with the trigonometry....)

Line 130 draws the orbit as a series of straight lines; alternatively, use line 140 to plot a series of points (this shows up speed changes nicely), or line 150 to plot one moving point. You might also like to duplicate the general form of lines 130 and 160 as lines 135 and 165, but using different variables. With a few other changes, this will give you two orbiting objects. It is surprisingly hard to manage an interception of the second object if you start in a different orbit! (I am quite interested in simulating two orbiting spacecraft with a few tens of miles of wire connecting them together - which is an actual proposal - if you get this working, please send it to me! Though not for publication.)

The calculations are crammed into line 170 to keep things as fast as possible. First we get distance from the planet (d) using Pythagoras' Theorem (sum of the squares of the other two sides and all that). Then the angle that gravity will pull from (i.e. the direction of the planet) is obtained by a bit of trigonometry that took me a lot of head-scratching and trial-and-error. The gravitational force on the spacecraft (f) is inversely related to distance from the planet, squared. That force will have an effect on the horizontal velocity (hv) and vertical velocity (vv) of the spacecraft that depends on both the strength of the force and its angle. From the two velocity variables it is pretty easy to calculate new coordinates (x and y) after a certain time. The calculation assumes that all the factors like distance, angle and force can be considered to be constant over the small section of the orbit that is calculated at one time. In real life, this isn't true, but the error isn't serious unless your orbit goes quite close to the planet. (I have not provided any check for a collision with the planet, but you could add this by checking the value of d.) The variable s sets the size of each calculation step, allowing you to obtain greater precision at the expense of speed.

```
 10 LET xos=128,yos=88
 20 LET x=-80,y=0
 30 LET hv=0,vv=2.5
 40 LET s=.5
 50 LET g=600/s
 60 LET ox=x,oy=y
 70 LET a$=CHR$ 8+CHR$ 9+CHRS 10+CHR$ 11
 80 LET p=.1
 90 BORDER 6
100 CIRCLE 0,0,2
110 PLOT x,y
120 DO
130   DRAW TO x,y
140   REM PLOT x,y
150   REM
      PLOT OVER 1;ox,oy
      PLOT x,y
      LET ox=x,oy=y
160   ON INSTRING(1,aS,INKEY$)
        LET hv=hv-p
        LET hv=hv+p
        LET vv=vv-p
        LET vv=vv+p
170   LET d=SQR (x*x+y*y),a=PI*(x<0)+PI/2-ATN (y/x),f=g/(d*d
       ),hv=hv+f*SIN a,vv=vv+f*COS a,x=x-hv/s,y=y-vv/s
180 LOOP
```

In the illustration, I was initially in the outer orbit, then I fired a braking burst at A to drop to a lower orbit. Note that only the other side of the orbit moves. At point B, I fired two more braking bursts to make the orbit progressively more circular.



****

PROC JUST - Proportional text justification on the screen.

This contribution is from Alan Salmon of Bristol. He writes:

"The advantage of this routine is that, instead of merely expanding the spaces between words, it expands each letter into the correct size to fill the line. The syntax of the command is JUST t$,cpl, where t$ is the text to be justified, and cpl is the required number of characters per line. Note: the last line of the text is not justified, as this delineates the end of a paragraph."

"Notes: Lines 1030-1050 choose a suitable height for the characters. Line 1060 gets the current CSIZE. Line 1130 restores the CSIZE. PROC CHOP chops off <= cpl characters from a$ to b$. PROC DISP displays a line of text, justified."

Sometimes, if the number of characters per line is small, the font size will vary within the text, because of large changes in the required character spacing. However, the routine would be useful for outputting information or instructions neatly.

```
  10 OVER 2
  20 INPUT "Characters per line:";cpl
  30 just "This procedure is used to justify text and make it
     look generally neater. Its advantage over similar
     procedures is that it uses fractional spaces, as you can
     probably see.",cpl

1000 DEF PROC just a$,cpl
1010   LOCAL csiz1,csiz2,h,b$
1020   DEFAULT cpl=32
1030   LET h=INT (256/cpl)-2
1040   DO UNTIL h/8=INT (h/8)
         LET h=h+1
       LOOP
1050   LET h=h+2
1060   LET csiz1=PEEK 57370,csiz2=PEEK 57371
1070   CSIZE h
1080   DO UNTIL LEN a$<=cpl
1090     chop a$,b$,cpl
1100     disp b$,h
1110   LOOP
1120   PRINT CSIZE INT (256/cpl),h;a$
1130   CSIZE csiz1,csiz2
1140 END PROC

2000 DEF PROC chop REF a$, REF b$,cpl
2010   LOCAL f
2020   LET b$=""
2030   FOR f=cpl+1 TO 1 STEP -1
2040     IF a$(f)<>" " THEN
           NEXT f
2050   IF f=0 THEN
         LET f=cpl
2060   JOIN a$( TO f-1) TO b$
2070   IF a$(1)=" " THEN DELETE a$(1)
2080 END PROC
```

```
3000 DEF PROC disp x$,h
3010   LOCAL w,x
3020   LET x=256
3030   DO UNTIL LEN x$<=1
3040     LET w=INT (x/LEN x$)
3050     PRINT CSIZE w,h;x$(1);
3060     LET x=x-w
3070     DELETE x$(1)
3080   LOOP
3090   PRINT CSIZE w,h;x$
3100 END PROC
```

****

PROC ERASEY/N - optional file erase procedure

This contribution comes from Robert Dickson (London). It requires the CAT TO procedure from issue 9. Robert says:

"It runs through the catalogue, printing each program name, one at a time, asking whether the file is to be erased or not. This procedure is useful if you have lots of files on a cartridge that need to be erased, it saves having to type lots of ERASE commands. The procedure uses the DEFAULT drive, whereas the CAT_TO procedure uses drive 1. I suggest the following alteration to PROC CAT_TO, so that it uses the default drive:"

```
1040 CAT #3;PEEK 57395
```

(That suggestion, and Robert's procedure, should work OK with Microdrive or Discovery disc. PROC ERASEY/N could also be modified to work with the Disciple or PLUS D, if allowance is made for the different catalogue format.)

```
 10 erasey/n

100 DEF PROC erasey/n
      LOCAL f,n,p$,c$,a$,b$
      GET p$,0,175,32,6;1
      WINDOW 2,0,175,256,48
      WINDOW 2
      BRIGHT 1
      CLS
        cat_to a$
      CLS
      PRINT "Cartridge name: ";a$( TO 10)
      PRINT "Free memory: ";a$(LEN a$-2 TO )
120   LET a$=a$(13 TO LEN a$-4)
130   FOR n=1 TO LEN a$ STEP 11
        LET c$=a$(n TO n+9)
        PRINT AT 3,0; "Current program: ";c$
        PRINT "Erase? (Y/N)"
140     IF SHIFT$(1,INKEY$)="N" THEN GO TO 170
150     IF SHIFT$(1,INKEY$)<>"Y" THEN GO TO 140
160     ERASE c$
        GO TO 180
170     FOR f=0 TO 50
        NEXT f
180   NEXT n
190   WINDOW 0
      PLOT 0,175;p:$
    END PROC
```

****

POPULATION FLUCTUATIONS

This section was stimulated by a book review; actually, three separate reviews of "CHAOS: MAKING A NEW SCIENCE" by James Gleick. The book is about the mathematics of chaotic systems. I'll probably never read it, but the reviews were interesting. They made the point that the behaviour of some systems, such as those controlling the weather, are very sensitive to initial conditions; a tiny change in some input can cause large changes to develop. If turbulence from your cough can alter next month's weather, long-term forecasting is a difficult art! The system modelled below is much simpler than weather, but it too can exhibit this "chaotic" behaviour. The program simulates population changes with a fixed food supply (e.g. on an island). The population is X, where 0=extinction and 1=maximum. R is the rate of growth. The first statement in line 110 is the heart of the simulation; it states that the next population size depends on the current population size times the rate of growth, times (1-X), which is a measure of how close the population is to maximum. (The equation was invented by a chap called Verhulst in 1845.) As the population increases, it gets harder to reproduce because of shortage of resources. This normally leads to same stable population size being reached; with growth rates between about 1.1 and 2.9, al most any initial population will reach a stable state quite quickly. But try a growth rate of 3.2 - this gives two alternating population levels. A growth rate of 3.5 gives 4 levels in regular succession. The illustration shows that complex, unpredictable behaviour occurs with some growth rates. This is not at all that we might expect from our simple equation! The picture is reminiscent of data I've seen from some real-life populations - and looks rather like some climate records, too. (The people working on "chaos" have found some interesting rules that work for very varied phenomena.)

```
 20 INPUT "GROWTH RATE:";r
 30 INPUT "INITIAL POPULATION:";x
 40 PRINT AT 6,0; "GROWTH RATE:";r
 50 PRINT "INITIAL POPULATION: ";x
 60 LET mag=5
 70 PLOT 0,100
    DRAW 255,0
 80 PLOT 0,0
    DRAW 255,0
 90 PLOT 255-mag,x*100
100 DO
110   LET x=x*r*(1-x)
      DRAW TO 255,x*100
120   SCROLL 5,mag;0,99;32,99
130   PLOT 255-mag,x*100
140 LOOP
```

****

STRING INSERTION

Dave Swales (Gainsborough, Lines.) has been trying to write a dictionary program in BB to aid in solving crossword puzzles. (See issue 8, page 15.) It sounded like a lot of typing to me, but David has gone ahead and got most of it working. He uses long strings to store the dictionary information, with the words packed together to save space. Each one starts with a capital letter and ends in "*". (This convention ensures that when you look for "Ban*" with INSTRING you won' t find it in "Abandoned*".) Dave wrote recently for help in adding extra words to the existing data, preferably keeping correct alphabetical order. I came up with the following bit of program; as an illustration, it inserts "abalone" into the right place in a fragment of the dictionary. The SHIFT$ function is used so that the user doesn't have to remember to add words with a leading capital letter. Line 110 deal with the case where the new word has to be "inserted" at the end of the dictionary.

A faster method would be to do a "binary search" for the right position, which is more the approach used by a human looking for a telephone number, rather than a simple forward search from the start of the dictionary. However, if I get too fussy, I never finish answering my morning post!

```
 10 LET a$="abalone"
 20 LET a$(1)=SHIFT$(1,a$(1))
 30 LET a$(2 TO )=SHIFT$(2,a$(2 TO ))
 40 LET a$=a$+"*"
 60 LET t$="Aardvark*Aback*Abandon*Abandoned*Abase*Abash*"
 70 LET p=1
 80 DO
 90   LET s=p
100   LET p=INSTRING(p,t$,"*")
110   IF p=0 THEN JOIN a$ TO t$
      EXIT IF 1
120   LET c$=t$(s TO p)
130   IF a$<c$ THEN JOIN a$ TO t$(s)
      EXIT IF 1
140   LET p=p+1
150 LOOP
160 PRINT t$
```

****

DISC BITS

Unfortunately, most of this section is devoted to errors that have somehow crept into the disc versions of Beta Basic. (Disciple/Plus D users please remember when making new saved versions of BB 4.0 to erase "bbc2" from the destination disc beforehand.) The first problem applies to some OPUS and Disciple/+D versions of BB 4.0. A "fix" for a RAMdisc SORT problem corrupted the code for normal SORTs, so that e.g. LET a$="Fred Bloggs": SORT a$ doesn't work. If you observe this symptom, the program below should cure it. First, load BB 4.0, then type in the program and run it.

```
10 LOAD "bbc2"CODE 32000
20 FOR n=37808 TO 37821: REM OPUS version
20 FOR n=37821 to 37834: REM Disciple version
30 READ a: POKE n,a: NEXT n
40 DATA 191,195,115,191,221,37,200,24,235,201,84,93,26,9
50 RANDOMIZE USR 32256
60 PRINT "Now NEW and MERGE Beta Basic's loader. RUN to
   SAVE modified version."
```

The next problem applies only to some Disciple versions of BB 4.0. Alan Purdom (Reading, Berks.) wrote to say that he could not get FORMAT "p";n to work (or even enter). This stopped him directing printed output to his serial printer, as the PLUS D/Disciple version of BB, as supplied, uses the built-in parallel port. I found that 4 bytes of the FORMAT command had been corrupted somehow, in my master copy. This can be corrected by loading BB 4.0, entering the following, and then using the usual line 1 SAVE routine in BB's loader.

```
POKE 55940,205: POKE 55941,63: POKE 55942,189: POKE 55943,11
```

This will allow you to use FORMAT "p";0 or FORMAT "p";1 to select RS232 text or "binary" modes, or e. g. FORMAT "p"; 1200 to set the baud rate. You might also have to POKE @11,1 to keep the disc system from taking control.

The following problem is a nasty one involving BB 3.0 and 4.0 on the Disciple/Plus D. It arises because a "cure" for this interface's tendency to delete invisible forms of numbers from BB programs goes wrong with floating-point numbers. For example, 10 DATA 1.3 will have extra bytes inserted into the line, which appear as odd characters or keywords. I don't know how I missed this problem, but I didn't get any immediate complaints, either! If you observe this symptom, use the following program, in either 3.0 or 4.0, form. As usual, use the line 1 SAVE routine once the changes are made. Many apologies to those inconvenienced by these problems. (Any user who wrote to complain has naturally had a personal reply already.)

```
10 FOR n=50236 TO 50251
20 READ a: POKE n,a: NEXT n
30 FOR n=50287 TO 50302: REM BB 3.0 version
30 FOR n=50461 TO 50476: REM BB 4.0 version
40 READ a: POKE n,a: NEXT n
50 FOR n=64239 TO 64242
60 READ a: POKE n,a: NEXT n
70 POKE 50192,48
80 DATA 205,111,196: REM BB 3.0 version
80 DATA 205,29,197: REM BB 4.0 version
85 DATA 48,249,205,182,24,40,195,229,223,205,155,44,225
```

```
 90 DATA 35,126,205,27,45,208,254,46,200,230,223,254,69,200,
    55,201
100 DATA 223,205,1,196
```

Lou Oliver (Perth) had troubles with an auto-running BB program on his PLUS D. As soon as the first BB command was hit, a "Nonsense in Basic" report stopped the program. Auto-running programs can be a problem on various systems; they tend to turn Beta Basic off. One solution can be to turn BB on again using RANDOMIZE USR 58419; on other systems, making the first line of the program a non-functional BB command (e.g. 10 ELSE) will be sufficient.

Lou also discovered a slightly roundabout way of saving program "slicers" to disc (not implemented for the Disciple/+D) using BB 4.0+D. He writes:

"First, the procedure, program lines or set of variables is SAVE!d to RAM disc in the normal way, but using a filename which MUST end with a "$" sign. Then it's just a case of using LIST and INPUT as you would with a normal array file. To get the data back into the program use MERGE!."

****

PRINTERS AND THE PLUS THREE

Stephen Folly (Harmondsworth, Middx.) is using BB 3.0 on the PLUS 3 in 48K mode. The normal LPRINT will not work with full-size printers. However, Stephen's procedure below makes this possible. He writes:

"The variable *p$* is the string to be printed. The variable *flag* is to determine whether characters with ASCII codes above 127 are to be sent to the printer or changed into normal characters (ASCII below 128). This is rather like the +3 command FORMAT LPRINT "E" or "U". If this value is zero, or not included, then the string will be converted using SHIFT$ with a value of 7. Then we enter the loop to print the characters. The DO..WHILE loop prevents characters being sent while the printer is off line. Then OUT the character, take the STROBE bit in port 8189 high, then low again. This routine will only work if you have not used OUT 32765,48 to disable paging routines."

```
1000 DEF PROC printer p$,flag
       LOCAL x
       DEFAULT p$=CHR$ 13,flag=0
       IF flag=0 THEN
         LET p$=SHIFT$(p$,7)
1010   FOR x=1 TO LEN p$
         DO WHILE AND(IN 4093,1)
         LOOP
         OUT 4093,CODE p$(x)
         OUT 8189,20
         OUT 8189,4
       NEXT x
     END PROC
```

****
READERS' LETTERS

Dear Andy.
NL 10 received and the fractal program was very interesting... Please tell us how the fractal fern works. Why should five data statements make such a thing and how are different shapes made? I know it can't be trial and error.

Mike Eida. Crawley. Sussex

Like I said, I don't really have the expertise to explain that program properly. It wasn't my technique. But I did mean to give some more background. Simple shapes can be built up by a combination of calculation and trial and error - try fiddling with the data statements I gave - but this is rather laborious, and I haven't done much along such lines. The BYTE article I dredged the idea from showed complex colour pictures that had been reduced to a few hundred or thousand bytes, by calculating something equivalent to the DATA statements in the fractals program. This is compression of over 10,000 to 1 compared to the conventional memory requirement. I would have guessed that was impossible, but I suppose coding a fern in 29 numbers is pretty amazing too. For a while I grew very excited about the great games graphics this would open up - then I read that the compression of the images takes about 100 hours, and the expansion about 39 minutes, even on a very fast micro! In the future, though, custom hardware should allow multiple frames per second to be decoded - allowing complex full-colour animation to be stored very compactly, or transmitted easily. But don't expect this next year!

I think some readers may be a bit intimidated by complicated articles in the Newsletter. They should always remember that programs written by someone else, who perhaps sweated over the thing for weeks, are generally harder to understand than one's own efforts. Many programs include neat "tricks" or methods that are non-obvious unless they are the kind you are used to. Besides, sometimes I include things I have stumbled across elsewhere that struck me as interesting, even if I didn't understand them very well - like the fern!

Some contributions require a knowledge of the internal structure of a BASIC program, or of machine code, for full understanding. However, I always hope that such stuff will be useful in any case. After all, most assembly-language programmers (often called whizz-kids, until they get old like me) are quite happy to call a handy ROM routine, as long as they know what it does. How it does it doesn't matter very much.

Dear Andy.

I was trying to check a BB 3.0 program after altering an array with the following command: ALTER a$ TO "!a$". For some reason the program did not work properly! I had to perform other corrections to the program, like:

```
IF !a$(5,I)="a" OR !a$(5,I)="A"...
```

won't work. Also, after LET !a$(I)="", the string !a$(I) stayed intact! Is this a bug. or is it intentional? LET !a$(I)=" " worked! BB 4.0 users should also be warned that there is a difference between !A$() and !a$() - unlike normal arrays.

Is there any way to PEEK a location to know if the program is running on a 48K Spectrum or a 128K machine? I would like to be able to use a line like the following:

```
IF PEEK (version)=128 THEN LOAD !"PICT"SCREEN$
```

so that a program could run on both machines, having special features on the 128K one.

I am having difficulties arranging for a "customized" version of BB 4.0, like I have for BB 3.0, with LIST FORMAT, CSIZE, DEF KEYs arranged to my liking, including lowering RAMTOP to accomodate a Hebrew character set... Am I doing something wrong?

A few questions about RAM disc arrays: With memory arrays there is a big difference in access speed if they are declared early on before other variables. Is this similar for RAM disk arrays?

I again must congratulate you on an excellent program. You have again improved a computer to the best of its capabilities!

Daniel Ben-Sefer, Degania B, Israel

Using ALTER on array commands in a BB 3.0 program may well not produce a usable BB 4.0 RAM disc version. This is because (see p13, BB 4.0 manual) references to RAM disc arrays must be at the start of an expression, because of restrictions imposed by the ROM expression evaluator. What is an expression? Well, to the computer, anything that evaluates into a number or a string is an expression. so your example is seen as a single expression (not two with an OR in the middle) and breaks the rules. You need to use e.g.:

```
LET t$=!a$(5, I): IF t$="a" OR t$="A" THEN...
```

The inconsistency you mention between normal and RAM disc arrays when a null string assignment is made is my mistake. Assembly language programmers always have to be careful when dealing with zero lengths, because e.g. a block move of zero bytes will work on 64K bytes instead. I made sure not to do anything when a null string assignment was made, but I should have "padded" the destination string to all spaces, as you point out.

You can tell a 48K Spectrum from later versions by peeking the ROM. (PEEK always works on the 48K-mode part of the ROM in 128K machines, but this is not an exact match for the original 48K ROM.) PEEK 76 will give 2 only on a 48K Spectrum. A related problem is to detect what mode a 128K Spectrum is running in; Charles Buszard (Chorleywood, Herts.) has suggested PEEK 23681. This gives 91 in Spectrum 48K mode after a reset (or in a real 48K machine) or zero in 128K mode, Whether or not BB 3.0 or BB 4.0 are resident. (This is the most-significant byte of the printer buffer address - in 48K mode, try POKE 23681,64: LPRINT "qwertyuiop".) The only possible problem is that PEEK 23681 remains zero if you go into 48K mode via the SPECTRUM command.

You shouldn't have any problems making a customized BB 4.0. You didn't say exactly what you did, so I can't really help.

Just as with memory variables, RAM disc arrays declared first will be found fastest. This probably won't be very significant unless you have quite a few RAM disc files.