Fill of Machine Code

This program contains a general purpose routine which will fill any enclosed area of the screen in any chosen colour except the current paper colour. It will work in all screen modes and its operation can be aborted at any time before the area is completely filled.

The routine operates by scanning left and right from the starting point, filling in any unpainted pixels, until it meets a vertical boundary. It then stores its current values on a stack and moves up a line from where it calls itself, recursively, and repeats the process until it meets a horizontal boundary. When this happens the program retraces its steps - by retrieving previous values from the stack and continuing from where they left off -moving downwards if necessary, until all stored values have been removed from the stack and all pixels within the boundary have been filled.

This process, however, has the disadvantage of altering some of the values on the stack as it proceeds so that a complete record of all operations is no longer available.

The effect of this is that it is not easy to reverse the process if anything goes wrong; for instance, if the shape to be filled has a small gap in it then the ink will 'escape' and proceed to fill in the whole screen. For this reason a second stack has been used to store such information as is necessary to enable the process to be aborted at any time and the partly filled area to be returned to normal. This use of two stacks, requires a large amount of memory for storage and in theory this could amount to more than the entire screen itself takes up. In practice though, two stacks of 2K each should be sufficient for all but the most intricate and elaborate shapes.

To see the routine in action type in the BASIC program and run it. If you have made a mistake in entering the data then the program will stop and inform you of the offending line which can then be corrected and the program re-run. Note that you must enter the checksum at the end of each `DATA` statement.

If all is well the screen will clear to mode 0 and an irregular shape will appear on the screen inside a large square. The program will then proceed to fill the area around the shape and when this is completed it will fill the shape itself, but in a different colour. After a short delay the whole process will be repeated using modes 1 and 2 and will then start all over again with mode 0. Note that in the highest resolution mode both colours will be the same since in this mode only two colours can be present on the screen. Pressing any key will abort the current fill stage and 'unfill' the painted area before proceeding to the next stage.

When any section is completely filled there is ample time, especially in the higher resolution modes, to decide whether or not to abort. To stop the program completely press **[ESC]** twice.

The machine code routine can be stored anywhere in memory by changing the value of start in line 70. If you relocate the program in this way and find that it does not function correctly and if it worked before you relocated it then you have almost certainly missed out a '@' somewhere in the `DATA` statements. Note that when relocating the code you must leave at least 4K for the stacks and about 450 bytes for the routine itself so it should not be loaded above about &9900 or 39168 decimal.

The demonstration program can be removed by deleting lines 310 to 440 and 480 to 520 and the remaining lines can be used as part of a larger program. Note that your program must set X and Y to a point inside the shape being filled before calling the routine. Also the required graphics ink must be previously selected by PLOT a,b,n where a,b is an imaginary point off screen - such as 800,800 - and n is the required ink number.

```
10 REM ********************************
   ************************************
20 REM ** AMSTRAD CPC464 User Magazine :
    programs from Issue number 4 *******
30 REM *Basic loader and demonstration p
   rogram for machine code FILL routine*
40 REM ********************************
   ************************************
50 BORDER 13:INK 0,13:INK 1,0:INK 3,24
60 DEF FN hex(h$)=VAL("&"+LEFT$(h$,2))
70 start=30000:REM Change this value to
   relocate routine at a different addre
   ss
80 addr=start:ln=510
90 MEMORY start-1
100 REM Main loop to poke in values from
    data statements
110 total=0
120 READ h$:h$=UPPER$(h$)
130 IF h$="END" THEN 280
140 IF ASC(h$)<>ASC("@") THEN 210
150 h$=RIGHT$(h$,LEN(h$)-1)
160 lb=FN hex(h$):total=total+lb:h$=RIGH
    T$(h$,LEN(h$)-2):IF h$="" THEN 250
170 hb=FN hex(h$):total=total+hb:h$=RIGH
    T$(h$,LEN(h$)-2)
180 w=256*hb+lb-30000+start
190 POKE addr,w-256*INT(w/256):addr=addr
    +1
200 POKE addr,INT(w/256):GOTO 230
210 b=FN hex(h$):total=total+b:h$=RIGHT$
    (h$,LEN(h$)-2)
220 POKE addr,b
230 addr=addr+1:IF h$<>"" THEN 140
240 READ checksum:IF checksum=total THEN
     ln=ln+10:GOTO 110
```

```
250 PRINT"ERROR IN LINE";ln
260 END
270 REM Demonstration program to draw an
    d fill shape in different modes
280 xa=&76DE-30000+start:ya=&76E0-30000+
    start
290 FOR i=0 TO 2
300 MODE i
310 PLOT 220,50,3
320 DRAWR 200,0:DRAWR 0,200
330 DRAWR -200,0:DRAWR 0,-200
340 PLOT 250,100,3
350 DRAWR 20,120:DRAWR 40,10
360 DRAWR 20,-40:DRAWR 10,20
370 DRAWR 60,-60:DRAWR -80,-20
380 DRAWR -20,40:DRAWR -20,-40
390 DRAWR 80,-40:DRAWR -110,10
400 PLOT 800,800,1
410 x=240:y=150
420 FOR j=1 TO 2
430 POKE xa,x-256*INT(x/256):POKE xa+1,I
    NT(x/256)
440 POKE ya,y-256*INT(y/256):POKE ya+1,I
    NT(y/256)
450 CALL start
460 FOR k=1 TO 700:NEXT
470 x=280:y=150:PLOT 800,800,3
480 NEXT j
490 NEXT i
500 GOTO 290
510 REM Hex values for machine code FILL
     routine
520 REM Note that a '@' indicates a valu
    e to be changed during relocation.
530 REM The value at the end of each lin
    e is the sum of all entries on that
    line
540 DATA CDE1BB32@EC7647CDE7BB32@ED76B8C
    8CD@6876C@ED73@EE76CD11BC4704,4316
550 DATA 3E08CB3F10FC32@DC7621@F27E22@F0
    7621@F68622@F47E21000022@E276,3013
560 DATA 22@E47622@E676CD09BBDA@9076ED5B
    @F07621@E776010A00EDB8,3143
570 DATA ED53@F076CD@6876204FCD@E8752A@E
    27622@E6762A@E67622@DE76,3302
580 DATA 2A@F0762323235E23561313ED53@E07
    6CD@6D752A@E67622@DE762A@F076,3015
590 DATA 2323235E23561B1BED53@E076CD@6D7
    52A@E676ED4B@DC76@922@E676,2898
600 DATA ED4B@E476A7ED4238BC28BA11@DE762
    A@F07623010A00EDB02B22@F076C9,3450
610 DATA 2A@DC7622@EA76CD@34762A@E876ED4
    B@DC76A7ED4222@E476EB2A@F47E,3680
620 DATA 722B73E521000ED4B@DC76A7ED4222
    @EA76CD@34762A@E876ED4B@DC76,3457
630 DATA 0922@E276EBE12B722B732BED5B@E07
    6722B732B22@F47EC9ED5B@DE76,3463
640 DATA ED53@E8762A@E076ED5B@E876CDEABB
    2A@E876ED4B@EA760922@E876545D,3984
650 DATA 018002A7ED42D02A@E076CDF0BB473A
    @ED76B828D5C92A@DE76018002AF,3635
660 DATA ED42301A2A@E076018F01ED4230102A
    @E076ED5B@DE76CDF0BB473A@ED76,3441
670 DATA B8C9FE01C92A@F47E11@F6867DBB200
    37CBAC83A@ED76CDDEBB2A@F47E23,3981
680 DATA 5E2356ED53@E076234E2346E5EB5059
    CDC0BBE1235E2356E52A@E076,3400
690 DATA CDF6BBE111@F6867DBB20D97CBA20D5
    3A@EC76CDDEBB2A@EE76F9C90000,4250
700 DATA END
```