PERSONAL COMPUTER NEWS ISSUE 26 1st September 1983

TEXT FROM ARCHIVE.ORG -
Originally full of OCR errors, corrected as far as possible.

TI TRANSFORMATIONS

Stephen Shaw details the pleasures and pitfalls of making your TI programs rewrite themselves.

Self-writing TI

The idea of a program which can change itself is not particularly new... infact, it is one of the oldest concepts around.

However, it has never been exploited to any major extent, largely because the received wisdom in the industry is that self-modifying programs are a bad thing. Lest this should sound autocratic, we should add that there's a very good reason.

Ordinary common-or-garden nonmodifying programs are hard even for the original author to follow or modify. Imagine trying to follow one which changes every time you look at it!

So. enjoy the idea, use it to do some clever tricks, but take care not to include the technique in any programs which you expect to use for a long time and don't wish to be endlessly modifying

==========

When either the Mini Memory Module or the Extended Basic Module plus 32K Extended Memory are used, it is possible for TI99/4A owners to examine the storage of their programs in the computer's memory.

The T199/4A stores program lines on a stack principle. As each line is entered, regardless of its line number, it is placed at the top of the stack. When a program line is edited, the old line is removed, the stack is adjusted, and the new line added to the top, hence the delay before the cursor reappears. The computer is changing the memory locations of every line above the edited line, and changing the line index which it uses to point to the lines, and which is stored at the very top of the program stack in line number order.

If no disk controller is attached, users may find their TI Basic programs in VDP RAM. The first line entered will end at address 16383. and each subsequent line entered will end at a lower address.

With Extended Basic (Vn 100) and 32K RAM. programs are stored from CPU RAM address -25. each subsequent line having a more negative address. A handful of Extended Basic Version 110 modules have been sold in the UK. With these, programs start from CPU address 0 (zero).

Programs are stored in coded format, with single byte codes for the command words, using ASCII codes 129 to 254. This is why users may not define 255 characters. Internally an offset is used to make ASCII 32 (space) appear to be a code 0, and ASCII 159 appear to be a code 127, for screen printing purposes.

In program storage the offset is not used and characters appear as having their proper coding.

Enter this program, in this order:

100 REM PCN
110 A=B+2
120 C$=D$&“E”

In Extended Basic, in command mode, enter the following line (NB: no line number!):

```
FOR T=-25 TO -51 STEP -1
:: CALL PEEK(T,A) ::
PRINT T;A;CHRS(A) :: NEXT
T
[UPDATED: Using XB Vn 110
use for T=0 TO -26 STEP
-1]
```

When you press ENTER the computer will display the short three line program by showing the memory location, the value in that location, and the equivalent character (if appropriate).

Without 32K RAM the program is stored in VDP RAM and Extended Basic does not allow you access to this area of memory.

With mini memory, the command mode section must be added to the program, using locations from 16383 to 16356. Note that in TI Basic the storage

format is slightly different, although the same codes are used, eg in TI Basic a space is inserted on both sides of the REM PCN. For mini memory, use PEEKV instead of PEEK.

A list of the command codes is given in figure I. They are fairly straightforward, except the way in which fixed values are stored. NUMBERS and UNQUOTED STRINGS are identified by code 200. This is followed by the number of digits or characters involved, and then the number or the characters themselves.

An example of an unquoted string is the name given to a subprogram. CALL COLOR for instance uses one byte for CALL but COLOR takes up 7 bytes — 5 for the word and one each to identify the unquoted string and to indicate its length. This is why you cannot use CALL SUB$ : SUB$ is a quoted string. Ouoted strings are identified by code 199. and follow the same format — one byte is used for the length of the string.

LINE NUMBERS when they appear in a program (eg GOTO 123) are identified by code 201. and the actual line number then takes up just two bytes, whatever number it is. If the first byte is A and the second byte is B, the line number is:

LINE NUMBER = A times 255 plus B.
Byte B has a maximum value of 127. and byte A a maximum value of 255, giving a maximum line number of 32767.

It is possible by entering short programs such as the above to obtain a good understanding of how the computer stores its programs.

As you have the capacity to change the contents of CPU RAM with Extended Basic (CALL LOAD) or VDP RAM with Mini Memory (POKEV). it becomes possible for a program to almost completely rewrite itself.

In Extended Basic, add to the short program above the following line:

130 CALL INIT:: CALL LOAD(-28,77,65,71)
(XB Vn 100)

Before you RUN the amended program. LIST it. Now ENTER RUN and LIST again. Notice any change?

When changing a line in a program in this manner, there are two important precautions:

1. The line, and any lines below it in the program stack, must not be edited. Otherwise when you change the contents of memory locations, you won't be changing the line you thought you were! It is possible to look up the line's memory location in the line index before the program rewrites the line, but it is much easier to ensure that the line(s) to be rewritten are at the bottom of the stack. If only one line is to be edited, enter it first With a middle value line number:

10000 REM PCN

Now you may enter lines on either side, and edit them, and the location of that (first entered) line will not alter. You may also RESEQUENCE without causing any problems.

2. The length of the line is the first byte in the line, and it is probably not possible to rewrite a line with a different length.

In Extended Basic this is not too much of a problem: the initial line can terminate with a tail REM (!) and a long false REM. When rewritten you merely ensure that the overwriting terminates with a tail REM (code 131) and a space (code 32). then the rest of the line remains as a dummy REM.

In TI Basic it is usually necessary to keep the line length the same, but some commands do permit dummy endings. This is a matter for experimentation.

What use is this facility? I have programs in TI Basic and Extended Basic which permit pseudo high resolution pictures to be drawn by redefining characters. When completed the computer scans the screen and rewrites the program by dumping the definitions and positions of the characters to defining lines. When the overwritten program is re-run, the sketch appears quite quickly.

Another use is to create commands TI do not give you.

A popular use is to enable a generalised disk directory to be added to each disk.

When Extended Basic is selected, the automatic directory, on the disk as LOAD, is loaded and RUN automatically.

It then reads the disk index and presents you with a menu. The menu selection is then automatically run. Extended Basic will permit the program line:
100 RUN “DSK1.GAME"
but not:
100 RUN “DSK1."&A$
There seems to be no reason for this not to be accepted, except that it gives an error message SYNTAX ERROR

Therefore the rewrite facility is used to CALL LOAD the required line into memory, one byte at a time, so that the computer sees the line as RUN "DSK1.GAME", exactly as it wants to see it. In this case, because the disk file names are not of fixed length, a value of zero was placed in the unused dummy line positions. Zero marks end of line and prevents crashes.

It is possible with this facility to insert your own (if limited) VAL function, to permit for example the INPUT of a fraction in the form 3/4.

First you need a dummy line:
10000 A = AAAAAAAAAA + AAAAA
AAAAA+AAAAAAAAAA
If this is the first line input it is fairly simple to find the locations of each character in the line, as they are stored in memory.

Your input will be to a string variable: INPUT “FRACTION": A$ then you must split this up into its three parts and place them into the DEF line.

Use a loop and SEG$ to determine the location of the oblique "/". This will enable you to determine each part of the string.

Following the equal sign in the DEF statement you will need code 200, then a value equal to the number of digits in the first number (use LEN and SEG$). Then place the number using the ASCII codes for each digit.
Then follows code 196 (/), code 200, and the length of the second number, then the digits in ASCII code.

Finally, so that the excess AAA’s have no effect, in Extended Basic load the aides 131 and 32, or in TI Basic load the codes 193 (+) and two 65’s (A).

Provided your program does not use variables made up of several AAA's, these have a zero value and no effect.
To quickly see a final result, clear your computer and enter:
1000 A-45/788+AA+AAAAAA+AAAAA

Now see how that is stored, using the methods described above.

After you have entered your fraction. and CALL LOADed (orPOKEV'd) it into memory, you may refer to the fraction in your program by GOSUBbing to 1000 to set the value of A.

If the line is to be used more than once, it should be restored to its original state between each use, by CALL LOADing the original values.

The ability to change a program in this way opens a powerful and useful door for TI99/4A owners, who are no longer quite as limited as they may have thought.

It is possible for a 13K program to almost completely overwrite itself — only the last line needs to be unaltered, to prevent a crash during overwriting.

Note the use of CALL LOAD above. You may load a line fully with only one command, and in the correct order. When using Mini Memory CALL INIT is not used.

Figure 1
see image

■ Many codes are not accepted by the TI Basic interpreter,
■ Some code* (mariied NK) an not accepted in Extended Basic.■ Codes may be used slightly differentty by TI Basic to Extended Basic. The computer adjusts storage format If a program saved in TI Basic is loaded with Extended Basic and vice versa.