Program Encapsulation on the HP-42S

Erik Ehrling #1097

Back in Datafile V10N6 in an article called "FOCAL Plus – The Language for Real Handheld Hackers!" Simon Bradshaw speaks about various improvements that he would like to see in a new version of the FOCAL programming language of the HP-41C/V/X and HP-42S calculators. One of these improvements would be the ability to "save the stack at the beginning of a subroutine and recover it at the end" – the purpose of this article is to demonstrate how this by some simple measures can be achieved on the HP-42S.

By using the technique described here it is possible to both save the stack contents and the contents of the numbered registers. Further, by only using numeric/single character labels and numbered registers within a program all internal labels and storage registers can be hidden to the outside. No named variables would then be floating around after the program has finished and only the first program label would be visible in the list of programs (e.g. when pressing XEQ).

As anyone using a HP-42S with 32K (or an emulator for that matter) would have experienced, the memory easily gets cluttered by a lot of variables and program labels – especially as the HP-42S lacks a catalog structure. Using program encapsulation would in this case help to improve the situation.

Saving the stack

The main idea behind this technique is very simple. Just use a set of named variables with reserved names (reserved in the sense that no other program should use them) - here the syntax X_*prgname*, Y_*prgname*, Z_*prgname*, T_*prgname*, L_*prgname*, R_*prgname* is suggested. These variables should then be used for storing the contents of the four stack levels, the LASTX register and the numbered registers (REGS).

For example the start of a program could look like:

```
01 LBL "QPI"              10 STO "L_QPI"
02 STO "X_QPI"            11 RCL "REGS"
03 R↓                     12 STO "R_QPI"
04 STO "Y_QPI"            13 R↑
05 R↓                     14 18
06 STO "Z_QPI"            15 1
07 R↓                     16 NEWMAT
08 STO "T_QPI"            17 STO "REGS"
09 LASTX                  18 R↓
```

The stack contents and the numbered registers are saved. A new set of registers is created (here of size 18, this size should of course be adapted to the individual program). By using NEWMAT instead of SIZE it is ensured that the new registers will not be complex regardless of whether REGS was complex or not upon calling the program.

The real program logic would then start at program line 19. From there and onwards all storage registers that the program manipulates would be the program's own local instances.

The careful reader would have noticed that only the contents of the X register is preserved on the stack when reaching line 19. However, if needed, the contents of the other stack levels could easily be recalled, e.g. by RCL "Y_*prgname*".

The reason for having one named variable for each stack level and not using a 1x5 matrix for holding the contents of the stack levels is that the stack levels could themselves be holding matrices or strings, which in turn are not allowed to be stored in a matrix.

Restoring the stack

When the program eventually has finished the contents of the temporary variables should be recalled to the stack and the variables deleted. For example the program end could look like:

```
375 RCL "R_QPI"           385 CLV "T_QPI"
376 STO "REGS"            386 CLV "Z_QPI"
377 RCL "L_QPI"           387 CLV "Y_QPI"
378 STO ST L              388 CLV "X_QPI"
379 RCL "T_QPI"           389 END
380 RCL "Z_QPI"
381 RCL "Y_QPI"
382 RCL "X_QPI"
383 CLV "R_QPI"
384 CLV "L_QPI"
```

In this specific example all four stack levels are restored. If the program would perform a calculation on X returning a new X value then only T, Z and Y should be retrieved, X set to the new value and LASTX set to the old X value (and so on for variants where more stack levels are involved).

This technique can seem trivial at a first look but actually provides a much more mature feel to programs on the HP-42S when used! Try it out!

Note: An updated copy of the QPI program that was published in V22N4 and which is used as an example in this article can be found at: http://www.hp42s.com/programs/qpi/qpi.html