# Recursive Programming in M

*by David Selwood*

One frequent problem with M on some multiuser systems is the lack of stack memory. *Recursion* programming takes place when a function calls itself possibly many times, relies on a lengthy stack, and is computed in exponential time. An example of recursion is shown in figure 1, where the program totals all the numbers between 1 and 100, and displays the result of each calculation. Due to insufficient stack memory, the program crashes as soon as it has called itself 24 times.

To resolve the problem of insufficient stack memory, one option could be to increase the memory, but this is not always viable on some M multiuser systems because all M jobs are allocated the same amount of stack memory. The problem could be resolved by using *iteration* and thus allowing functions to be computed in linear time. That is, instead of using recursion, a programmer could use the FOR command. The problem with the FOR command is that there are functions, such as the *factorial* or the *fibonacci* functions, that can be expressed most naturally (and most easily) via recursion.

To perform recursion without increasing the stack memory, I chose to use the disk drive as a stack but sacrificed the speed of memory in favor of the disk size. Two advantages in this selection were that it would help in debugging and also that any elements on the disk stack can be accessed quite easily without "popping off" the stack as happens in conventional use.

A simple example of this is given in figure 2, where the disk is used as a stack for the labels of subroutine calls. The routine in figure 2 is split into two sections, with the first section simply initializing two variables and calling the code that performs the recursion. The variable Num is used to hold the number of the next addition and also acts as an index into the disk stack. When the routine performs a recursive call, it sets up a temporary global that within the brackets contains the level of recursion, with level 1 (Num=1) being the first. Second, the right side sets up the label that the routine should jump to when falling out of the recursion. It should be noted that the right side also could contain the contents of variables that could be restored when falling out of the recursion. The routine then adds one onto the variable Num and then calls itself with a GOTO. As soon as Num is greater than 100, the routine then starts to fall out of the recursion. The index of the variable Num is checked for on the disk and, if the label exists where the routine should fall to, is read. Again, note that at this time, if the contents of other variables needed to be restored, they could be read in the same read statement.

Since the routine in figure 1 crashed after 24 calls to itself, it would prove difficult to compare for processing times against the example in figure 2. Disk caching was used (as it is on most systems) and the actual performance was more than acceptable.

Although the two examples of recursion are trivial, the method of using the disk as a stack has been used in a far more complex situation for personnel scheduling. The actual algorithm for using the disk as a stack was the same as shown here and the routine for rostering was NP-complete (no known polynomial algorithms and nondeterministic) and would be difficult if not impossible with linear techniques.

Note: This methodology is useful in special cases, but the possibility of error (in logically stacking and unstacking values from disk) is not small. The extra programming required might force the programmer to "beat up" on the M supplier to allow more nesting levels.

```
ALGREC01  KILL
          SET Num=1,Result=0
          DO Check(Num)
          QUIT

Check(Num)      New (Num,Result)
          SET Result=Result+Num
          WRITE !,Num,?10,Result
          IF Num<100 D Check(Num+1)
          QUIT
```

Figure 1. Recursion using memory.

```
ALGREC02  KILL  KILL ^TEMP($j)
     SET Num=1,Result=0
     DO Check
     QUIT
Check    New (Num,Result)
     IF Num>100 G Check01
     SET Result=Result+Num
     WRITE !,Num,?10,Result
     SET ^TEMP($j,Num)
          ="Check01"
     SET Num=Num+1
     Go Check
Check01 S Num=Num-1,Var=$G
               (^TEMP($j,Num))
     IF Var'="" Go @Var
     KILL ^TEMP($j)
     QUIT
```

Figure 2. Recursion using the disk as a stack.

David Selwood has used M for more than four years and works for Oldham NHS Trust where he is the senior analyst for all development platforms. He keenly awaits M products to become more Windows-based. You may contact him at 419 Rochdale Road, Royton, Oldham, Ol15SL, United Kingdom.