Recursion

by Frederick L. Hiltz, Stage Manager

Recursion: a routine calls itself, directly or through intermediate routines. The method has acquired a thoroughly undeserved mystique. Let us de-mystify it because recursion belongs in every programmer's tool kit. Either recursion or iteration can solve many problems. Choose the method to match the structure of the problem, considering the clarity, generality, and (lastly) efficiency of both.

We warm up to the subject with a finger exercise that is always popular in the MTA tutorial on programming with style and discipline: reversing the characters in a string. Spread your fingers in front of you, palm out, and imagine 10 letters pasted on your fingernails –

A B C D E G H I J K

Reverse the left half of the string (palm in). Reverse the right half of the string –

E D C B A K J I H G

Finally, swap left and right halves –

K J I H G E D C B A

Reversing a 10-character string required reversing two 5-character strings, and we don't really know how to do that — "turning the palm in" was a black box operation. Don't curse, recur(s)! Apply the method to each 5-character string and repeat until reaching a 1-character string or a 0-character string. We do know how to reverse those; just return them unchanged. Figure 1 illustrates the method in M with a divide-and-conquer algorithm that often goes nicely with recursion.

How many calls to $$reverse are needed to reverse a string of N characters? Each pass calls $$reverse twice and cuts the length in half, thus the number of calls = 2 $\log_2(N)$. Compare the equivalent iterative function, which is trivial in M. Iteration requires N/2 passes through its loop, but it is simpler than $$reverse and certainly faster for small strings. However, what about reversing a 10-million-character string stored on disk?

Enough play with toys. Figure 2 contains a practical routine that

```
          ; Illustrates recursion by reversing the sequence of characters in a string.
          ;         $$reverse("string") = "gnirts"
          ;
          ; Algorithm: reverse(whole) =
          ;              reverse(right half)_reverse(left half)
          ;
reverse(string)n length s length=$l(string)
          i length<2 q string
          q $$reverse($e(string,length\2+1,length))_
                        $$reverse($e(string,1,length\2))
```

Fig. 1 Reversing a String

visits all the nodes in an M array — a depth-first tree walk. Visiting a node entails three steps:

1. Process the node with a routine supplied by the caller.

2. Visit the node's left-most child, if any.

3. Visit the node's right-hand sibling, if any.

In general, both recursion and iteration require initialization, work to be done on each pass, and a termination condition. Do not be overly concerned with the stack requirements of recursion. The 1995 ANSI standard specifies a minimum of 127 stack levels, which is adequate for most real applications. **M**

Frederick L. Hiltz, Ph.D., develops medical information system software at Brigham and Women's Hospital, Boston, Massachusetts.

fhiltz@bics.bwh.harvard.edu

Do you have a question that deserves discussion? Have you found a good answer to someone else's question that you would like to share? How about a controversial question and a discussion of pros and cons? If you prefer that your name not be published, please say so in your contribution, which should be sent to the Managing Editor at *M Computing*.

Editorial Schedule

September, 1996 Issue
Topic: M and the Internet

December, 1996 Issue
Deadline for submission:
September 15, 1996
Topic: Business & Finance

MTA 1997 Annual Conference in Boston at the Hynes Convention Center.

Week of May 18th

```
            ; Depth-first tree walk of a global or local variable (glvn) and
            ; all its descendants, applying the named function to each node | ;
            that has a value. The function receives glvn = the name of
            ; the node, and may do what it will with the node — even kill
            ; it.
            ;Example:
            ;           d depthlst("^P(101588)","show") q
            ;     show  w I,glvn," = ",@glvn q
            ;
depthlst(glvn,funcname) ;
            i $d(@glvn)#lo d @funcname
            d children(glvn)
            q

children(parent) n lastsub
            s lastsub="" f s lastsub=$o(@parent@(lastsub)) q:lastsub="" d
            . d depthlst($name(@parent@(lastsub)),funcname)
            q
```

Fig. 2 Depth-first tree walk