

Using Recursion in the SAS® System

David Ward, InterNext, Inc., Somerset, NJ

ABSTRACT

Recursion is a friend of most 3rd or 4th generation language programmers. It can be used to slice hundreds of lines of code from complex algorithms, or bring a system to its knees after having consumed all its resources. The concept of recursion and useful applications of it using SAS®/Macro and SAS®/AF software will be presented. You will learn how to traverse directories, trees of information, and other interesting yet challenging topics. After reading this paper you should have a better understanding of when and how to use recursion in the SAS® system.

INTRODUCTION

Recursion refers to the case when a function or sub-routine calls itself. In order for recursion to successfully take place the programming language must provide separate memory areas for the local variables in each function call. Therefore, the data step can not be used to execute a recursive algorithm (although it can be mimicked). Instead, a SAS® programmer must rely on either the SAS® Macro language or SCL to institute recursion. Often the first example of recursion programming students encounter is in the world of sorting. For simplicity's sake, this paper will not examine sorting algorithms in SAS® because simpler examples exist to get you comfortable with recursion.

OUR FIRST ALGORITHM

Let's say you have genealogical data that is updated each month. Each person could potentially have a different number of ancestors, and that number could also change from month to month. Your assignment is to print a report for each person, listing their ancestors (assume fathers).

Name	Father	Mother
====	=====	=====
John Jones	Mark Jones	Mary Fawcett
Sarah White	Henry Little	Jane Smith
Mark Jones	David Jones	Harriet Lawler

From the data above you can see that John has two ancestors listed (Mark and David Jones), and Sarah White only has one ancestor listed. The thought might occur to you to use proc SQL to solve the problem of getting all ancestors for each person, maybe something like:

```
Proc sql noprint;
  Create table ancestors as
  Select d1.name, d2.father as ancestor1 from
  Data as d1, data as d2 on d1.name=d1.father;
Quit;
```

You should quickly see the problem with this approach – you need one join for every ancestor a person has. If a person has 20 ancestors you would need 20 join statements. Instead, using recursion, we do not need to know how deep we must go for each person to obtain their ancestors. A basic outline of the algorithm:

- 1) Find the father of person A
- 2) If person A has no father listed, leave
- 3) Otherwise person A's father becomes person A
- 4) Go to step 1

The algorithm will leave at the appropriate step for each person. The author is aware that this example could be solved with some data step tricks, but it has been chosen for simplicity sake. A macro approach could be:

```
%global ancestors;
%macro getAncestors(person);
  %local father; %let father=;
  data _null_;
    set data(where=(person="&person"));
    call symput('father',father);
```

```
run;
%if %length(&father)>0 %then %do;
  %let ancestors=&ancestors/&father;
  %getAncestors(&father);
%end;
%mend;
%let ancestors=;
%getAncestors(John Jones);
%put NOTE: Ancestors: &ancestors;
```

Which produces this in the log:

```
NOTE: Ancestors: /Mark Jones/David Jones
```

EXAMPLE 1: SEARCHING FOR FILE NAMES

You can use recursion to get a list of all files matching a pattern name in all sub-directories under a given directory. The idea is simple: loop through the names of the files in the current directory, then, for all sub-directories, call the macro on itself with the sub-directory as the new parent directory. A sample macro implementation follows:

```
data files;
  length file $200;
stop; run;
%macro findMatches (dir,pattern);
  %put NOTE: Searching &dir;
  /** DEFINE LOCAL MACRO VARIABLES **/
  %local files i dnum did matches rc
  filename;
  /* SET FILENAME TO CURRENT DIRECTORY */
  filename dir "&dir";
  %let did=%sysfunc(dopen(dir));
  %let dnum=%sysfunc(dnum(&did));
  %do i = 1 %to &dnum;
    %let files
      =&files/%sysfunc(dread(&did,&i));
    %if %index(
      %upcase(%scan(&files,&i,/)),
      %upcase(&pattern)) %then
      %let matches
        =&matches/%scan(&files,&i,/);
  %end;
  %let rc=%sysfunc(dclose(&did));
  /** WRITE MATCHES **/
  %if %length(&matches)>0 %then %do;
    data matches;
      length file $200;
      %let i = 1;
      %do %while(
        %length(%scan(&matches,&i,/))>0
      );
        file="&dir\%scan(&matches,&i,/)";
        output;
        %let i=%eval(&i+1);
      %end;
    run;
    proc append base=files new=matches;
    run;
  %end;
  /** CALL MACRO FOR EACH SUBDIRECTORY **/
  %let filename=file;
  %local i;
  %do i = 1 %to &dnum;
    %let rc=
      %sysfunc(filename(filename,
        &dir\%scan(&files,&i,/)));
    %let did=%sysfunc(dopen(file));
    /** IF CANT OPEN FILE ASSUME DIRECTORY **/
```

```

    %if &did^=0 %then %do;
        %let rc=%sysfunc(dclose(&did));
        %findMatches(&dir%scan(&files,&i,/),
                    &pattern);
    %end;
%end;
%mend;
options nomprint;
%findMatches(d:\dward,.exe);

```

EXAMPLE 2: CONVERTING AN SCL LIST INTO A STRING

The SCL list is a powerful tool available to SAS®/AF developers. Lists are filled with items that can be numeric, character, or another list, thus a hierarchy of data can be put into a list in the form of data tree. In order to traverse a tree structure you can use a recursive SCL method. Once the method has been created you could use it like this:

```

Dcl char string;
init:
    ** BUILD THE SAMPLE LIST **;
    l={ one=1, two={ a='LETTER A', b='LETTER B',
                    c={ I='RN I', II='RN II' } } };
    ** PUT THE LIST INTO A STRING **;
    string=method('cptools.trackuser.methods.scl',
                  'stufflist',l);
    ** DISPLAY THE STRING TO THE LOG **;
    put string=;
    ** PUT THE STRING INTO ANOTHER LIST **;
    m=method('cptools.trackuser.methods.scl',
              'expandlist',string);
    rc=putlist(m,'m',0);
return;

```

The above example uses two methods, stufflist and expandlist. The code for the methods (it doesn't format nicely in this single-column document):

```

** STUFFLIST 1/10/01 DWard
    Create a string that represents a list
    Restrictions: You cannot have a list item
    that contains braces {}, and equals sign, or a
    comma **;
length string name value $32767;
stuffList: method listid:i:list
optional=string:u:char return=char;
    if listlen(listid)<=0 then
        return('{}');
    do i = 1 to listlen(listid);
        if i=1 then string=trim(string)||'{';
        else string=trim(string)||',';
        if nameitem(listid,i)^='' then
            string=trim(string)||
            nameitem(listid,i)||'=';
        select itemtype(listid,i);
            when ('C') string=trim(string)||
            quote(getitemc(listid,i));
            when ('N') string=trim(string)||
            compress(getitemn(listid,i));
            otherwise rc=method('methods.scl',
            'stuffList',getiteml(listid,i),
            string);
        end;
        if i=listlen(listid) then
            string=trim(string)||'}';
    end;
    return(string);
endmethod;
** EXPANDLIST: Take a string and convert it into
a list
    String should be from stuffList
    1/10/01 DWard **;
Dcl list sublist;

```

```

expandList: method string:u:char
optional=listid:u:list i:u:num
return=list;
    if listlen(listid)>-1 then
        sublist=listid;
    else sublist=makelist();
    if length(string)<=2 then
        return(sublist);
    if i<=2 then i=2;
    do i = i to length(string);
        if substr(string,i,1)=' ' then i=i-1;
        if substr(string,i,1)='{' then do;
            i=i+1;
            return(sublist);
        end;
    else do;
        name='';
        value=substr(string,i);
        np=indexc(value,'{','');
        oi=i;
        i=i+np-1;
        if substr(string,i,1)='{' then
            i=i-1;
        value=substr(value,1,np-1);
        value=scan(value,1,',','');
        np=index(value,'=');
        if np then do;
            name=scan(value,1,'=');
            value=substr(value,np+1);
        end;
        if substr(value,1,1)='{ ' then do;
            sublist2=makelist();
            rc=insertl(sublist,sublist2,
                -1,name);
            i=oi+np+1;
            rc=method('methods.scl',
                'expandList',string,sublist2,i);
        end;
    else if substr(value,1,1)='"' then
        do;
            value=tranwrd(substr(value,2,
                length(value)-2),'"', '');
            rc=insertc(sublist,value,
                -1,name);
        end;
    else rc=insertn(sublist,value,
        -1,name);
    end;
end;
return(sublist);
endmethod;

```

The author used this technique when two SAS® sessions needed to communicate an SCL list over a TCP/IP socket connection that did not support sending the list, only ASCII data.

CONCLUSION

Recursion is a powerful technique that will allow you to perform advanced functions with your SAS® programs, whether they are written in SAS®/Macro or SCL.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

David L. Ward
InterNext, Inc.
254 Resnik Ct.
Somerset, NJ 08873
Work Phone: (732) 470-6783
Email: dward@SAS®help.com