SESUG Paper 233-2018

Recursive Programming Applications in Base SAS®

Jinson J. Erinjeri and Pratap Kunwar, Emmes Corporation

ABSTRACT

Programmers employ recursive programming when faced with tasks which are structured hierarchically. Recursive programming involves the call of the same program or function within itself to solve such tasks. For the recursive program to execute effectively, it requires an entry condition, a recursive sequence and an exit condition. Recursive programming can be employed in base SAS® and this paper presents some of the cases where recursive programming can be applied. In addition, the cases presented in the paper can help instill a programmer of how and when to take the recursive approach while programming.

INTRODUCTION

Recursive programming is based on the concept of recursion which is a process in which the function calls itself directly or indirectly. The basic idea behind recursion is to reduce a complex problem to a smaller instance of itself. For the recursion to materialize, there are basically three requirements - an entry condition, a recursive sequence and an exit condition. The entry condition is the state the function begins with whereas the exit condition is the state at which the function will finally return with no more calls to itself any more. The recursive sequence is lines of code in the body of the function which calls to itself.

In this paper, we intend to introduce the recursive programming approach in Base SAS with some common applications. In Base SAS, recursive programs can be written by creating a SAS Macro which calls itself. Recursive approaches are specifically helpful with tasks which are structured hierarchically. It is also important to note that recursion approach may not be always efficient with regards to resources. In this paper, we have presented seven applications of employing recursion programming approach. It is important to be mindful of errors of infinite recursion and stack overflow when developing recursion based codes. Infinite recursion occurs when there is an incorrect implementation or lack of an exit condition. Stack overflow occurs when there is an improper exit condition or when there are too many calls for the available memory in the machine to execute.

APPLICATIONS

Application 1: Factorial of a Number

Factorial of a number is a classic example where recursive approach can be applied. The factorial of an integer is the product of all integers up to and including the integer. For example, the factorial of 4 denoted by 4! is 4x3x2x1=24. The recursive program and output(log) to generate the factorial of a number is shown in Display 1. The two code snippets presented will give the exact same results, the only difference being the placement of entry, exit and recursive sequence within the code.

```
%macro fact1(num);
  %if &num=1 %then 1;
  %else %if &num=0 %then 1;
  %else %eval(&num * %fact1(%eval(&num-1)));
%mend fact1;
%let k1=%fact1(4);
%put THE FACTORIAL BY USING MACRO FACT1 IS &k1.;

%macro fact2(num);
  %if &num>1 %then %eval(&num * %fact2(%eval(&num-1)));
  %else %if &num=1 %then 1;
  %else %if &num=0 %then 1;
%mend fact2;
%let k2=%fact2(4);
%put THE FACTORIAL BY USING MACRO FACT2 IS &k2;
```

```
1     %macro fact1(num);
2      %if &num=1 %then 1;
3      %else %if &num=0 %then 1;
4      %else %eval(&num * %fact1(%eval(&num-1)));
5     %mend fact1;
6     %let k1=%fact1(4);
7     %put THE FACTORIAL BY USING MACRO FACT1 IS &k1.;
THE FACTORIAL BY USING MACRO FACT1 IS 24
8
9     %macro fact2(num);
10     %if &num>1 %then %eval(&num * %fact2(%eval(&num-1)));
11     %else %if &num=1 %then 1;
12     %else %if &num=0 %then 1;
13    %mend fact2;
14    %let k2=%fact2(4);
15    %put THE FACTORIAL BY USING MACRO FACT2 IS &k2;
THE FACTORIAL BY USING MACRO FACT2 IS 24
```

Display 1. Code and Log Output for Factorial of a Number

Application 2: Fibonacci number

The Fibonacci Sequence is the series of numbers 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 and so forth. The next number in the sequence is found by adding the two numbers before it. For example, 13 is obtained by adding the previous two numbers, 5 and 8. In this example, 13 is the 7th term in the Fibonacci sequence and is obtained by adding the 6th and 5th term. The process of adding the previous two numbers can be obtained recursively as shown in Display 2.

```
/*Fibonacci Number*/
%macro fib(n);
  %if &n = 0 %then 0;
  %else %if &n = 1 %then 1;
  %else %if &n = 2 %then 1;
  %else %eval(%fib(%eval(&n-1))+ %fib(%eval(&n-2)));
%mend;

%let a1=%fib(7);
%put THE FIBONACCI NUMBER IS &a1.;


22    %macro fib(n);
23    %if &n = 0 %then 0;
24    %else %if &n = 1 %then 1;
25    %else %if &n = 2 %then 1;
26    %else %eval(%fib(%eval(&n-1))+ %fib(%eval(&n-2)));
27    %mend;
28    *%fib(5);
29
30    %let a1=%fib(7);
31    %put THE FIBONACCI NUMBER IS &a1.;
THE FIBONACCI NUMBER IS 13
```

Display 2. Code and Log Output for Fibonacci Number

Application 3: Cumulative Sum of Natural Numbers

Following the same logic of generating a Fibonacci number, the cumulative sum of natural numbers (including the number) can be recursively programmed as shown in Display 3. For example, the cumulative sum of natural number up to 5 is 5+4+3+2+1=15 and this output is also presented in Display 3.

```
/*Cumulative Sum of Natural Numbers*/
%macro sums(num);
  %if &num=0 %then 0;
  %else %eval(&num + %sums(%eval(&num-1)));
%mend;

%let s1=%sums(5);
%put THE CUMULATIVE SUM OF NATURAL NUMBER IS &s1.;
```

```
1     /*Cumulative Sum of Natural Numbers*/
2     %macro sums(num);
3     %if &num=0 %then 0;
4     %else %eval(&num + %sums(%eval(&num-1)));
5     %mend;
6     *%sums(5);
7
8     %let s1=%sums(5);
9     %put THE CUMULATIVE SUM OF NATURAL NUMBER IS &s1.;
THE CUMULATIVE SUM OF NATURAL NUMBER IS 15
```

Display 3. Code and Log Output for Cumulative Sum of Natural Numbers

Application 4: Greatest Common Denominator of Two Numbers

The Greatest Common Denominator (GCD) of two numbers is the largest integer that divides both these numbers without any remainder. It is also referred to as greatest common divisor, greatest common factor, highest common divisor or highest common factor. There are many algorithms to accomplish this such as Brute Force, Euclid and Dijkstra. In this example we have used Dijkstra's algorithm and is presented in Display 4. The idea behind this algorithm is that if m and n are two numbers and m>n, then GCD of m and n is the same as GCD of m-n and n. Mathematically, GCD(m,n) = GCD(m-n,n) due to fact that if m and n are divisible by a number with no remainder then m-n is also divisible by the same number with no remainder.

```
/*Greatest Common Denominator of Two Numbers*/
%macro gcd(num1,num2);
       %if &num1=0 or &num2=0 %then 0;
       %else %if &num1=&num2  %then &num1;
       %else %if &num1>&num2 %then %gcd(%eval(&num1-&num2),&num2);
       %else %if &num1<&num2 %then %gcd(&num1,%eval(&num2-&num1));
%mend;

%let gcd=%gcd(81,9);
%put THE GREATEST COMMON DENOMINATOR OF TWO NUMBERS IS &gcd.;
```

```
1     /*Greatest Common Denominator of Two Numbers*/
2     %macro gcd(num1,num2);
3         %if &num1=0 or &num2=0 %then 0;
4         %else %if &num1=&num2  %then &num1;
5         %else %if &num1>&num2 %then %gcd(%eval(&num1-&num2),&num2);
6         %else %if &num1<&num2 %then %gcd(&num1,%eval(&num2-&num1));
7     %mend;
8     %let gcd=%gcd(81,9);
9     %put THE GREATEST COMMON DENOMINATOR OF TWO NUMBERS IS &gcd.;
THE GREATEST COMMON DENOMINATOR OF TWO NUMBERS IS 9
```

Display 4. Code and Log Output for Greatest Common Denominator of Two Numbers

Application 5: Create a Sequence List

Creating a sequence list is very common in day to day programming and the recursive approach to perform this is shown in Display 5. The comma between the items of the list can be excluded and this is especially useful in creating sequential variables while using arrays. Display 5 shows the code as well as the log output for both these scenarios.

```
/*Creating a Sequence List with Comma*/
%macro varlist(n,pref);
%if &n=1 %then &pref.1;
%else %varlist(%eval(&n-1),&pref.), &pref&n;
%mend varlist;

%let seq1=%varlist(10,jinson);
%put THE SEQUENCE WITH COMMA IS &seq1.;

/*Creating a Sequence List without Comma*/
%macro varlist(n,pref);
%if &n=1 %then &pref.1;
%else %varlist(%eval(&n-1),&pref.) &pref&n;
%mend varlist;

%let seq2=%varlist(10,jinson);
%put THE SEQUENCE WITHOUT COMMA IS &seq2.;
```

```
124  /*Creating a Sequence List with Comma*/
125  %macro varlist(n,pref);
126  %if &n=1 %then &pref.1;
127  %else %varlist(%eval(&n-1),&pref.), &pref&n;
128  %mend varlist;
129
130  %let seq1=%varlist(10,jinson);
131  %put THE SEQUENCE WITH COMMA IS &seq1.;
THE SEQUENCE WITH COMMA IS jinson1, jinson2, jinson3, jinson4, jinson5, jinson6, jinson7,
jinson8, jinson9, jinson10
132
133  /*Creating a Sequence List without Comma*/
134  %macro varlist(n,pref);
135  %if &n=1 %then &pref.1;
136  %else %varlist(%eval(&n-1),&pref.) &pref&n;
137  %mend varlist;
138
139  %let seq2=%varlist(10,jinson);
140  %put THE SEQUENCE WITHOUT COMMA IS &seq2.;
THE SEQUENCE WITHOUT COMMA IS jinson1 jinson2 jinson3 jinson4 jinson5 jinson6 jinson7
jinson8 jinson9 jinson10
```

Display 5. Code and Log Output for Creating a Sequence List

Application 6: List Files under a Directory/Sub-Directory

Recursive approach is best suited for finding files under a directory since it has a hierarchical set up. This example can be found at http://support.sas.com/kb/45/805.html The code lists all the files that contain a particular extension within a directory and its sub-directories. The code is reproduced from the mentioned link and shown in Display 6 along with the snapshot of the log.

```
/*List Files under a Directory/Sub-Directory*/
%macro drive(dir,ext);
  %local filrf rc did memcnt name i;
  /* Assigns a fileref to the directory and opens the directory */
  %let rc=%sysfunc(filename(filrf,&dir));
  %let did=%sysfunc(dopen(&filrf));
```

```
  /* Make sure directory can be open */
  %if &did eq 0 %then %do;
   %put Directory &dir cannot be open or does not exist;
   %return;
  %end;

   /* Loops through entire directory */
   %do i = 1 %to %sysfunc(dnum(&did));
     /* Retrieve name of each file */
     %let name=%qsysfunc(dread(&did,&i));
     /* Checks to see if the extension matches the parameter value */
     /* If condition is true print the full name to the log        */
      %if %qupcase(%qscan(&name,-1,.)) = %upcase(&ext) %then %do;
        %put &dir\&name;
      %end;
     /* If directory name call macro again */
      %else %if %qscan(&name,2,.) = %then %do;
        %drive(&dir\%unquote(&name),&ext)
      %end;
   %end;

  /* Closes the directory and clear the fileref */
  %let rc=%sysfunc(dclose(&did));
  %let rc=%sysfunc(filename(filrf));

%mend drive;

/* First parameter is the directory of where your files are stored. */
/* Second parameter is the extension you are looking for.           */
%drive(C:\Jinson\checks,sas)
```

```
221  %drive(C:\Jinson\checks,sas)
C:\Jinson\checks\code1.sas
C:\Jinson\checks\code2.sas
C:\Jinson\checks\Male\sas_bday_reminder.sas
C:\Jinson\checks\sitemacro_code.sas
```

Display 6. Code and Log Output for Listing Files under a Directory/Sub-Directory

The macro call includes two parameters - the directory and the file extension. The output lists all the files with the specified extension under the directory as well as sub-directories. Note that the %drive macro is called recursively if it encounters a sub-directory.

Application 7: Standard MedDRA Queries (SMQ's)

Medical Dictionary for Regulatory Activities (MedDRA) is an adverse event coding system which is a means of gathering related adverse events (Preferred Terms) into hierarchical groups based on medical conditions or areas of interest. These groupings are called Standardized MedDRA Queries (SMQ's). This medical dictionary is updated twice a year and the most current version is 21.0. The recursive approach can be employed to process the SMQ's to obtain all the Preferred Terms (PT) under a specific SMQ. The snapshot of the data structure for MedDRA is shown in Display 7. In Display 7, the SMQ code(smqCode) of 20000005 represents Hepatic disorders. The termLevel variable with values of 4, 5 and 0 indicate PT, LLT or child SMQ respectively. The variable termStatus indicates whether a term is active or not. Child SMQ is indicated by the value "S" in the termCategory variable.

SMQ's can have multiple levels as shown for Hepatic disorder SMQ (20000005) in Display 7. Hepatic disorder SMQ includes five separate SMQ's – 20000006, 20000014, 20000016, 20000017 and 20000018. These five SMQ's can in turn have multiple SMQ's and so forth. With this type of data structure, direct linking to PT code is difficult and this is an ideal case for applying recursive programming. The code shown in Display 8 creates a SAS data set with PT's associated with a SMQ and its subordinates. The first macro will create macro variables for list of subordinate SMQ's and number of subordinate SMQ's. In addition, SMQ's with PT's will be appended to a base data set of PT's. The second macro involves the recursive processing where after the first pass, all the subordinate SMQ's are linked to

their associated PT codes and finally appended.

	termCategory	termStatus	smqCode	termCode	termLevel	termScope
1	S	A	20000005	20000006	0	0
2	S	A	20000005	20000014	0	0
3	S	A	20000005	20000016	0	0
4	S	A	20000005	20000017	0	0
5	S	A	20000005	20000018	0	0
6	S	A	20000006	20000007	0	0
7	S	A	20000006	20000008	0	0
8	S	A	20000006	20000009	0	0
9	S	A	20000006	20000015	0	0
10	S	A	20000007	20000010	0	0
11	S	A	20000007	20000011	0	0
12	S	A	20000007	20000012	0	0
13	S	A	20000007	20000013	0	0
14	S	A	20000011	20000208	0	0
15	S	A	20000011	20000209	0	0

Display 7. Snapshot of SMQ Data Structure

```
%macro indsmqpt;
proc sql;
/*Obtaining subordinate SMQ's (active)*/
      select termcode
      into :subsmq separated by " "
      from medra.smq_content
      where smqcode=&smqcode and termstatus="A" and termcategory="S";
/*Obtaining the count of subordinate SMQ's (active)*/
      select count(termcode)
      into :subcnt trimmed
      from medra.smq_content
      where smqcode=&smqcode and termstatus="A" and termcategory="S";
quit;
/*Obtaining PT's (active) that constitute a SMQ*/
data smqs;
      set medra.smq_content;
      if  smqcode=&smqcode and termlevel=4 and termstatus='A' ;
      keep smqcode termlevel termstatus termcode termscope;
run;
/*Appending the data set with all the PT's constituting the SMQ's*/
proc datasets library=work;
      append base=smq_final data=smqs;
      delete smqs;
quit;
%mend indsmqpt;

%macro ptssmq(smqcode);
      %local subsmq subcnt i;
      /*Calling the macro to execute the SMQ's*/
      %indsmqpt;
    /*Obtaining each subordinate SMQ and calling them recursively*/
      %do i=1 %to &subcnt.;
            %local smq&i;
            %let smq&i=%scan(&subsmq,&i);
            %ptssmq(&&smq&i);
      %end;
%mend;
%ptssmq(20000005);
```

Display 8. Code for Obtaining Preferred Terms Associated with SMQ's

Display 9 shows the snapshot of SAS data set output from the code presented in Display 8. This data set contains all the PT's associated with Hepatic disorder SMQ and its subordinates. The data set can be further processed to obtain the actual text of the PT terms as shown in the second data set in Display 9.

	termStatus	smqCode	termCode	termLevel	termScope
34	A	20000208	10073071	4	2
35	A	20000208	10073073	4	2
36	A	20000208	10073074	4	2
37	A	20000208	10074766	4	1
38	A	20000208	10077861	4	2
39	A	20000209	10019695	4	2
40	A	20000209	10061203	4	2
41	A	20000012	10004269	4	2
42	A	20000012	10018821	4	2
43	A	20000012	10019629	4	2
44	A	20000012	10019646	4	2
45	A	20000012	10052285	4	2
46	A	20000012	10053973	4	2
47	A	20000012	10054885	4	2
48	A	20000012	10067796	4	2
49	A	20000012	10077922	4	2
50	A	20000012	10079685	4	2
51	A	20000012	10079889	4	2
52	A	20000013	10000804	4	2
53	A	20000013	10003445	4	2
54	A	20000013	10003547	4	2
55	A	20000013	10004659	4	2
56	A	20000013	10004661	4	2
57	A	20000013	10004664	4	2

390	A	20000208	10073071	4	2	Hepatocellular carcinoma
391	A	20000208	10073073	4	2	Hepatobiliary cancer
392	A	20000208	10073074	4	2	Hepatobiliary cancer in situ
393	A	20000208	10074766	4	1	Liver ablation
394	A	20000208	10077861	4	2	Cholangiosarcoma
395	A	20000209	10019695	4	2	Hepatic neoplasm
396	A	20000209	10061203	4	2	Hepatobiliary neoplasm

Display 9. Snapshot of Output of SMQ's with Associated PT's

CONCLUSION

The applications presented in this paper shows how a recursive approach can be implemented to solve tasks. The use of recursive approach can enhance the efficiency and flexibility of a macro and is especially useful for investigative and testing purposes.

REFERENCES

SAS Online Documents. Accessed July 18, 2018. Available at http://support.sas.com/kb/45/805.html

Shen, David and Chen, Gary. 2013. "Practice of SMQs for Adverse Events in Analysis of Safety Data and Pharmacovigilance." *PharmaSUG 2013,* Chicago, IL
Available at https://www.pharmasug.org/proceedings/2013/HO/PharmaSUG-2013-HO02.pdf

CONTACT INFORMATION

Your comments/questions/criticisms are valued and encouraged. Please contact the authors at:

Jinson J. Erinjeri
The Emmes Corporation
401 N Washington St.
Rockville, MD 20850
E-mail: jerinjeri@emmes.com

Pratap Kunwar
The Emmes Corporation
401 N Washington St.
E-mail: pkunwar@emmes.com