Paper 067-2007

Do Which? Loop, Until or While? A Review Of Data Step And Macro Algorithms

Ronald J. Fehd, Centers for Disease Control, and Prevention, Atlanta, GA, USA

ABSTRACT

This paper reviews the three looping constructs: loop-repeat, do until and do while and offers examples of their use. The purpose of this paper is to provide both pseudo-code and examples so that programmers may understand the difference in logic and make an appropriate choice for their algorithm. The two data step loop processing verbs: continue (return to loop-top), and leave (exit) are illustrated. Macro examples using %goto are shown for continue and leave. The Whitlock subsetting loop — also known as the Do-Whitlock (DOW) loop — and double-DOW are illustrated.

Topics covered:

loop:	loop . . . repeat using *goto*
	do *until*(. . .)
	do *while*(. . .)
test in loop:	if condition then . . . *continue*
	if condition then . . . *leave*
	macro %*goto*
do Whitlock:	do until(last.var)

Audience: intermediate users and macro programmers.

Keywords: do until, do while, Do-Whitlock, double DOW, DOW, until(last.by-var), loop, repeat, subsetting, until, while, Whitlock do-loop,

Contents

INTRODUCTION

SAS® software provides two loop control verbs: `until` and `while`. The difference between the two keywords is that `while` tests its condition at the top of the loop and `until` tests its condition at the bottom on the loop. This is not obvious because the syntax requires both verbs to come after the keyword `do`:

```
do while(...)
do until(...).
```

Many questions to the SAS-L listserve are from beginning programmers who do not understand the difference between these two loop control constructs nor the difference in logic needed to implement the same algorithm.

This paper provide a basic pseudo-code algorithm with code examples illustrating the loop-repeat, do until, and do while implementations.

LOOP-REPEAT ALGORITHM

This is the basic pseudo-code of a loop-repeat block. All algorithms implement these eight steps. As shown in the next pseudo-code example SAS provides some elaborate extensions.

loop algorithm

```
1  initial: assignment(s)
2  loop   :
3           pre-test assignment(s)
4  test   : if condition then goto done
5           post-test assignment(s)
6  iterate: assignment
7  repeat : goto loop
8  done   :
```

SAS provides extensions within its loop-repeat block.

Dorfman [3] discusses the details of the interation process where the loop control variable `Index` is initialized with the `from` value and incremented using the `by` value.

loop algorithm: SAS enhancements

```
1  initial: assignment(s): e.g.: allocate array(s)
2  assign : Index = value-1
3  loop   : *do;
4  test-1 : if    Index gt value-last then goto done
5  test-2 : if    while-condition     then goto done
6           pre-test  assignment(s)
7  test-3 : if continue-condition     then goto iterate
8  test-4 : if    leave-condition     then goto done
9           post-test assignment(s)
10 test-5 : if    until-condition     then goto done
11 iterate: Index = next value
12 repeat : *end; goto loop
13 done   :
```

DO WHICH?

The difference between `while` and `until` is obfuscated by their placement at the top of the loop construct. As shown in loop algorithm: SAS enhancements, above, the while-condition is evaluated at the top of the loop, test-2, line 5, whereas the until-condition is evaluated at the bottom of the loop, test-5, line 10.

The following examples show that care must be taken in understanding the logical operators (see `Comparison Operators` in *SAS Language Reference, Concepts*) used in the `while(...)` and `until(...)` tests. Compare the sets of values in each and note that they are exclusive: `lt`: less than, `ge`: greater than or equal.

LOOP REPEAT

We can build a loop in the data step using labels and goto statements. In this simple example I illustrate each of the steps in the pseudo-code loop-repeat algorithm shown above.

do-loops.sas

```
2  *initial:; I = 1;
3  loop:      put I=;
4             if I eq 2 then goto done;
5  *iterate:; I+ +1;
6  *repeat :; goto loop;
7  done: put 'loop-repeat: ' I=;
```

do-loops.log

```
51 I=1
52 I=2
53 loop-repeat: I=2
```

DO WHILE

The `while` test is evaluated at the top of the loop.

do-loops.sas

```
8  J = 1;
9  do while(J lt 3);
10    put J=;
11    J+ +1;
12    end;
13 put 'do J: ' J=;
```

do-loops.log

```
54 J=1
55 J=2
56 do J: J=3
```

DO UNTIL

The `until` test is evaluated at the bottom of the loop.

do-loops.sas

```
14 K = 1;
15 do until(K ge 3);
16    put K=;
17    K+ +1;
18    end;
19 put 'do K: ' K=;
```

do-loops.log

```
57 K=1
58 K=2
59 do K: K=3
```

DO ITERATE

Compare the iteration with the `do until` and `do while` examples above. It is hidden in the `do L` statement, line 21, and happens between lines 22 and 23.

do-loops.sas

```
20 L = 0;
21 do L = 1 to 2;
22    put L=;
23    end;
24 put 'do L: ' L=;
```

do-loops.log

```
60 L=1
61 L=2
62 do L: L=3
```

MACRO LOOPS

Macro loops follow the same logic as the data step loops.

do-loops.sas

```
28 %local I;  %let I = 1;
29 %loop:     %put I=&I.;
30 %*test;    %if &I. eq 2 %then %goto done;
31 %*iterate; %let I = %eval(&I. +1);
32 %*repeat;  %goto loop;
33 %done:     %Put loop-repeat: I=&I.;
34
35 %local J; %let J = 1;
36 %do %while(&J lt 3);
37     %put J=&J.;
38     %let J = %eval(&J. +1);
39     %end;
40 %put do J: J=&J.;
41
42 %local K; %let K = 1;
43 %do %until(&K ge 3);
44     %put K=&K.;
45     %let K = %eval(&K. +1);
46     %end;
47 %put do K: K=&K.;
48
49 %local L;
50 %do L = 1 %to 2;
51     %put L=&L;
52     %end;
53 %put do L: L=&L.;
```

do-loops.log

```
98  I=1
99  I=2
100 loop-repeat: I=2
```

do-loops.log

```
101 J=1
102 J=2
103 do J: J=3
```

do-loops.log

```
104 K=1
105 K=2
106 do K: K=3
```

do-loops.log

```
107 L=1
108 L=2
109 do L: L=3
```

TESTING DURING LOOP

CONTINUE OR LEAVE

Some loop processing algorithms require either a skip pattern — return to top of loop: `continue` — or a conditional exit: `leave`.

do-loop-tests.sas

```
1  DATA _Null_;
2  do I = 1 to 3;
3     put I= 'pre-test';
4     if  I le 2 then continue;
5     put I= 'post test';
6     end;
7  put 'done continue: ' I= ;
8
9  do J = 1 to 3;
10    put J= 'pre-test';
11    if  J gt 2 then leave;
12    put J= 'post test';
13    end;
14 put 'done leave: ' J= ;
```

do-loop-tests.log

```
41 I=1 pre-test
42 I=2 pre-test
43 I=3 pre-test
44 I=3 post test
45 done continue: I=4
46 J=1 pre-test
47 J=1 post test
48 J=2 pre-test
49 J=2 post test
50 J=3 pre-test
51 done leave: J=3
```

MACRO %GOTO

There are no comparable `%continue` nor `%leave` statements in the macro language. However, as shown in the next examples they can be implemented using labels and `%goto`.

do-loop-tests.sas

```
%Macro Do_Tests(i=,j=);
%do I = 1 %to 3;
   %put I=&I. pre-test;
   %if  &I le 2 %then %goto continue;
   %put I=&I. post test;
   %continue:
   %end;
%put done continue: I=&I. ;

%do J = 1 %to 3;
   %put J=&J. pre-test;
   %if  &J. gt 2 %then %goto leave;
   %put J=&J. post test;
   %end;
%leave:
%put done leave: J=&J. ;
```

do-loop-tests.log

```
I=1 pre-test
I=2 pre-test
I=3 pre-test
I=3 post test
done continue: I=4
J=1 pre-test
J=1 post test
J=2 pre-test
J=2 post test
J=3 pre-test
done leave: J=3
```

USING LOGIC IN CONDITIONS

USING BOOLEAN LOGIC

The following are equivalent: `do while(not EndoFile)`
`do until(EndoFile)`

This is an important difference to understand: that the same algorithm can be implement using the two verbs, but the logic is different because of when the condition is evaluated.

Note that boolean values are in (False, not False). The preceeding statement means that only zero (0) is false; `until` and `while` evaluation treats negative as well as positive values as True.

do-boolean.sas

```
DATA do_until_endofile;
do   until(EndoFile);
     set SAShelp.Class
         end = EndoFile;
     output;
end; stop;
```

do-boolean.log

```
NOTE: There were 19 observations read from the data set
      SASHELP.CLASS.
NOTE: The data set WORK.DO_UNTIL_ENDOFILE has 19 observations
      and 5 variables.
```

do-boolean.sas

```
DATA do_while_not_endofile;
do   while(not EndoFile);
     set SAShelp.Class
         end = EndoFile;
     output;
end; stop;
```

do-boolean.log

```
NOTE: There were 19 observations read from the data set
      SASHELP.CLASS.
NOTE: The data set WORK.DO_WHILE_NOT_ENDOFILE has 19
      observations and 5 variables.
```

COMBINING ITERATION WITH LOOP CONTROL

An interation loop may be combined with an `until` condition. As noted above care should be taken to ensure that the variable tested in the `until` has boolean values, — in (0,1) — only.
Note: line 38, `done: I=2` shows that the evaluation of the `until` is done before the iteration.

do-I-eq-until.sas

```
1  DATA do_I_eq_until;
2  *initial; retain Done 0;
3  do I = 1 to 3
4     until(Done);
5     put I=;
6     output;
7     Done = (I ge 2);%*boolean;
8  end;
9  put 'done: ' I=;
10 stop; run;
```

do-I-eq-until.log

```
36 I=1
37 I=2
38 done: I=2
```

WHITLOCK SUBSETTING: DO UNTIL(LAST.BY-VAR)

Ian Whitlock [sasl.52734 5, in Feb., 2000] posted a solution to SAS-L with a `do until(last.id)` which has come to be known as the Do-Whitlock (DOW) loop. Take the time to squint at this code and figure out what is happening.

What is missing? The subsetting `if last.id then output;` statement. (Inserted after line 44.)

sas-l-post-052734.txt

```
39 data t ( keep = id v1 - v6 ) ;
40    array v (2,3) ;
41    array var ( 3 ) ;
42    do until ( last.id ) ;
43       set w ;
44       by id ;
45       do i = 1 to dim ( var ) ;
46          v ( method , i ) = var ( i ) ;
47       end ;
48    end ;
49 run ;
```

Whitlock's subsettting loop can be more easily understood with an explicit `output;` statement.

do-Whitlock.sas

```
2  Proc Sort data = SAShelp.Class
3            out  =         Class;
4            by     Sex Name;
5
6  DATA do_Whitlock_last;
7  do until(EndoFile);
8     do until(last.Sex);
9        set Class end = EndoFile;
10       by  Sex;
11       end;
12    output;
13    put Sex= Name=;
14    end;
15 stop; run;
```

do-Whitlock.log

```
53 Sex=F Name=Mary
54 Sex=M Name=William
55 NOTE: There were 19 observations read from the data set
56       WORK.CLASS.
57 NOTE: The data set WORK.DO_WHITLOCK_LAST has 2 observations and
58       5 variables.
```

The DOW can be used with `first.by-var` as well.

do-Whitlock.log

```
17         DATA do_Whitlock_first;
18         do until(EndoFile);
19            do until(first.Sex);
20               set Class end = EndoFile;
21               by  Sex;
22               end;
23            output;
24            put Sex= Name=;
25            end;
26         stop; run;

Sex=F Name=Alice
Sex=M Name=Alfred
NOTE: There were 19 observations read from the data set
      WORK.CLASS.
NOTE: The data set WORK.DO_WHITLOCK_FIRST has 2 observations
      and 5 variables.
```

DOUBLE-DOW

Paul Dorfman and Howard Schrier have posted several examples to SAS-L using the DOW algorithm and showing expanded usages. Here is an example.

do-double-dow.log

```
6          DATA do_double_Class;
7          do until(EndoFile);
8             do until(first.Sex);
9                set Class end = EndoFile;
10               by  Sex;
11               end;
12            put Sex= Name=;
13            output;
14            do until(last.Sex);
15               set Class end = EndoFile;
16               by  Sex;
17               end;
18            put Sex= Name=;
19            output;
20            end;
21         stop; run;

Sex=F Name=Alice
Sex=F Name=Mary
Sex=M Name=Alfred
Sex=M Name=William
NOTE: There were 11 observations read from the data set
      WORK.CLASS.
NOTE: There were 19 observations read from the data set
      WORK.CLASS.
NOTE: The data set WORK.DO_DOUBLE_CLASS has 4 observations and
      5 variables.
```

CONCLUSION

The two do-loop verbs `until` and `while` are distinguished by the execution of their loop-exit tests. To implement the same algorithm requires using different test conditions.

The DOW and double-DOW are an interesting use of the `do until` loops.

Suggested Readings

- Cassell [1] shows a use of the DOW with the prx (Perl) functions.
- Chakravarthy [2] shows how to use the DOW for the LOCF algorithm.
- Dunn and Chung [4, Examples 9–10] show how to calculate sum of variables using dougle-DOW.

BIBLIOGRAPHY

[1] David L. Cassell. Double your pleasure, double your words. In *Proceedings of the 29th Annual SAS® Users Group International Conference*, 2004. URL `http://www2.sas.com/proceedings/sugi29/046-29.pdf`. Coders' Corner, 5 pp.; using the Whitlock DO-loop with the PRX* (Perl) functions to find doubled words in lines of text.

[2] Venky Chakravarthy. The DOW (not that DOW!!!) and the LOCF in clinical trials. In *Proceedings of the 28th Annual SAS® Users Group International Conference*, 2003. URL `http://www2.sas.com/proceedings/sugi28/099-28.pdf`. Coders' Corner, 4 pp.; explanation of default reset of non-retained vars to missing when using first. and last. processing; do until(last.pt) replaces retain and if first.pt processing.

[3] Paul M. Dorfman. The magnificent Do. In *Proceedings of the 9th Annual Southeast SAS® Users Group Conference*, 2002. URL `http://www.devenezia.com/papers/other-authors/sesug-2002/TheMagnificentDO.pdf`. Topic: sequential processing, do-loop; info: do-loop terminology, DOW-loop: until(last.var), optimization, simplification.

[4] Toby Dunn and Chang Y. Chung. Retaining, laggin, leading, and interleaving data. In *Proceedings of the Pharmaceutical SAS® Users Group Conference*, 2005. URL `http://pharmasug.org/2005/TU09.pdf`. Tutorials, 8 pp.; topic: retain and lag functions; info: merge, reset of values to missing, DOW-loop: until(last.var).

[5] Ian Whitlock. Re: SAS novice question. In *Archives of the SAS-L listserve*, 16 Feb. 2000. URL `http://www.listserv.uga.edu/cgi-bin/wa?A2=ind0002C&L=sas-l&P=R5155`. First use of do until(last.id).

Author: Ronald Fehd **mailto:RJF2@cdc.gov**
Centers for Disease Control
4770 Buford Hwy NE
Atlanta GA 30341-3724

	about the author:	
education:	B.S. Computer Science, U/Hawaii,	1986
	SUGI attendee	since 1989
	SAS-L reader	since 1994
experience:	programmer: 20+ years	
	data manager at CDC, using SAS: 18+ years	
	author: 10+ SUG papers	
SAS-L:	author: 3,000+ messages to SAS-L since1997	
	Most Valuable SAS-L contributor:	2001, 2003

Document Production: This paper was typeset in LaTeX. For further information about using LaTeX to write your SUG paper, consult the SAS-L archives:

```
http://www.listserv.uga.edu/cgi-bin/wa?S1=sas-l
Search for                  :
The subject is or contains: LaTeX
The author's address        : RJF2
Since                       : 01 June 2003
```