BASUG, 2022-April-13, or was it April-31?

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

Ronald J. Fehd, SAS-L's Macro Maven,
senior maverick, theoretical programmer,
Fragile-Free Software Institute

Abstract

This paper reviews the three looping constructs: loop-repeat, do until, and do while; and offers examples of their useage. 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.

audience intermediate users and macro programmers.

information

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

In this paper

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

The loop-repeat algorithm

basic 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.

```
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 extensions : SAS provides extensions within its loop-repeat block.

Dorfman, "The Magnificent Do" 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.

```
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 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.

notes: see `Comparison Operators` in *SAS Language Reference, Concepts*

data step loops

loop repeat : We can build a loop in the data step using labels and goto statements. These simple examples illustrate each of the steps in the pseudo-code loop-repeat algorithm shown above.

```
1  *name: do-loops.sas;
2  data _null_;
3  retain i j k l 0;
4  *initial:; i = 1;
5  loop:      put i=;
6             if i eq 2 then goto done;
7  *iterate:; i+ +1;
8  *repeat :; goto loop;
9  done:      put 'loop-repeat: ' i=;
```

do-loops-data.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.

```
11 j = 1;
12 do while(j lt 3);
13    put j=;
14    j+ +1;
15    end;
16 put 'do j: ' j=;
```

do-loops-data.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.

```
18 k = 1;
19 do until(k ge 3);
20    put k=;
21    k+ +1;
22    end;
23 put 'do k: ' k=;
```

do-loops-data.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 26, and happens between lines 27 and 28.

```
25 l = 0;
26 do l = 1 to 2;
27    put l=;
28    end;
29 put 'do l: ' l=;
```

do-loops-data.log

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

note: Program `do-loops-data` is shown in the appendix, pg. 11.

macro loops

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

loop repeat : `loop: ... goto loop`

```
1  *name: do-loops-macro.sas;
2  %macro do_loops();
3  %local i;  %let i = 1;
4  %loop:     %put i=&i.;
5  %*test;    %if &i. eq 2 %then %goto done;
6  %*iterate; %let i = %eval(&i. +1);
7  %*repeat;  %goto loop;
8  %done:     %put loop-repeat: i=&i.;
```

do-loops-data.log

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

do while : `do while(j le 3)`

```
10 %local j; %let j = 1;
11 %do %while(&j lt 3);
12     %put j=&j.;
13     %let j = %eval(&j. +1);
14     %end;
15 %put do j: j=&j.;
```

do-loops-macro.log

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

do until : `do until(k ge 3)`

```
17 %local k; %let k = 1;
18 %do %until(&k ge 3);
19     %put k=&k.;
20     %let k = %eval(&k. +1);
21     %end;
22 %put do k: k=&k.;
```

do-loops-macro.log

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

do iterate : `do l = 1 to 2`

```
24 %local l;
25 %do l = 1 %to 2;
26     %put l=&l;
27     %end;
28 %put do l: l=&l.;
```

do-loops-macro.log

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

note: Program `do-loops-macro.sas` is shown in the appendix, pg. 11.

Testing during loop

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

continue : return to top of loop

```
1  *name: do-loop-skips.sas;
2  data _null_;
3  do i = 1 to 3;
4     put i= 'pre-test';
5     if  i le 2 then continue;
6     put i= 'post test';
7     end;
8  put 'done continue: ' i= ;
```

do-loop-skips.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
```

leave : conditional exit

```
10 do j = 1 to 3;
11    put j= 'pre-test';
12    if  j gt 2 then leave;
13    put j= 'post test';
14    end;
15 put 'done leave: ' j= ;
```

do-loop-skips.log

```
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
```

note: Program `do-loop-skips.sas` is shown in the appendix, pg. 12.

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`.

continue : return to top of loop

```
19 %macro do_tests(i=,j=);
20 %do i = 1 %to 3;
21     %put i=&i. pre-test;
22     %if  &i le 2 %then %goto continue;
23     %put i=&i. post test;
24     %continue:
25     %end;
26 %put done continue: i=&i. ;
```

do-loop-skips.log

```
76 i=1 pre-test
77 i=2 pre-test
78 i=3 pre-test
79 i=3 post test
80 done continue: i=4
```

leave : conditional exit

```
28 %do j = 1 %to 3;
29     %put j=&j. pre-test;
30     %if  &j. gt 2 %then %goto leave;
31     %put j=&j. post test;
32     %end;
33 %leave:
34 %put done leave: j=&j. ;
```

do-loop-skips.log

```
81 j=1 pre-test
82 j=1 post test
83 j=2 pre-test
84 j=2 post test
85 j=3 pre-test
86 done leave: J=3
```

note: Program `do-loop-skips.sas` is shown in the appendix, pg. 12.

Using logic in conditions

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

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.

do until : evaluate end_data at bottom of loop

```
1  *name: do-boolean.sas;
2  data do_until;
3  do until(end_data);
4     set sashelp.class
5         end = end_data;
6     output;
7     end;
8  stop;
9  run;
```

do-boolean.log

```
34 NOTE: There were 17 obs read
35       from the data set sashelp.class.
36 note: the data set
37       work.do_until has
38       17 obs and 5 variables.
```

do while not : evaluate end_data at top of loop

```
1  data do_while_not;
2  do while(not end_data);
3     set sashelp.class
4         end = end_data;
5     output;
6     end;
7  stop;
8  run;
```

do-boolean.log

```
51 NOTE: There were 17 obs read
52       from the data set sashelp.class.
53 NOTE: The data set
54       work.do_while_not has
55       17 obs 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.

notes: log line 38, `done: i=2`
shows that the evaluation of the `until` is done before the iteration.

```
1  *name: do-i-eq-until.sas;
2  data   do_i_eq_until;
3         *initialize:;
4         retain done 0;
5  do i = 1 to 3
6     until(done);
7     put i=;
8     output;
9     done = (i ge 2);
10         * returns boolean (0,1);
11    end;
12 put 'done: ' i=;
13 stop;
14 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, "Re: SAS novice question" posted a solution to SAS-L in Feb., 2000 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    do until ( last.id ) ;
40       set w ;
41       by id ;
42       do i = 1 to dim ( var ) ;
43          v ( method , i ) = var ( i ) ;
44       end ;
45    end ;
46 run ;
47
48 data q ( keep = id diff1 - diff3 ) ;
49    set t ;
```

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

```
1  *name: 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(last.sex);
8     set class;
9     by  sex;
10    end;
11 output;
12 put sex= name=;
13 run;
```

do-Whitlock.log

```
53 Sex=F Name=Mary
54 Sex=M Name=Thomas
55 NOTE: There were 17 obs read
56       from the data set WORK.CLASS.
57 NOTE: The data set
58       WORK.DO_WHITLOCK_LAST has
59       2 obs and 5 variables.
```

do until (first.by-var)

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

```
78 data do_whitlock_first;
79 do until(first.sex);
80    set class;
81    by  sex;
82    end;
83 output;
84 put sex= name=;
85 run;
```

do-Whitlock.log

```
78 Sex=F Name=Alice
79 Sex=M Name=Alfred
80 NOTE: There were 17 obs read
81       from the data set WORK.CLASS.
82 NOTE: The data set
83       WORK.DO_WHITLOCK_FIRST has
84       2 obs 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.sas

```
7  data do_until_twice;
8  do until(first.sex);
9     set class;
10    by  sex;
11    end;
12 put sex= name=;
13 output;
14
15 do until(last.sex);
16    set class;
17    by  sex;
18    end;
19 put sex= name=;
20 output;
21 run;
```

do-double-dow.log

```
42 Sex=F Name=Alice
43 Sex=F Name=Mary
44 Sex=M Name=Alfred
45 Sex=M Name=Thomas
46 NOTE: There were 11 obs read
47       from the data set WORK.CLASS.
48 NOTE: There were 17 obs read
49       from the data set WORK.CLASS.
50 NOTE: The data set
51       WORK.DO_DOUBLE_CLASS has
52       4 obs 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.

author contact info

Ronald J. Fehd mailto:Ron.Fehd.macro.maven@gmail.com

Suggested readings

- Cassell, “Double Your Pleasure, Double Your Words” shows a use of the DOW with the prx (Perl) functions.
- Chakravarthy, “The DOW (not that DOW!!!) and the LOCF in Clinical Trials” shows how to use the DOW for the LOCF algorithm.
- Dunn and Chung, “Retaining, Laggin, Leading, and Interleaving Data”, Examples 9–10 show how to calculate sum of variables using dougle-DOW.

References

Cassell, David L. (2004). “Double Your Pleasure, Double Your Words”. In: *Proceedings of the 29th Annual SAS® Users Group International Conference*. Coders Corner, 5 pp.; using the Whitlock DO-loop with the PRX* (Perl) functions to find doubled words in lines of text. URL: http://www2.sas.com/proceedings/sugi29/046-29.pdf (cit. on p. 9).

Chakravarthy, Venky (2003). “The DOW (not that DOW!!!) and the LOCF in Clinical Trials”. In: *Proceedings of the 28th Annual SAS® Users Group International Conference*. 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. URL: http://www2.sas.com/proceedings/sugi28/099-28.pdf (cit. on p. 9).

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

Dunn, Toby and Chang Y. Chung (2005). “Retaining, Laggin, Leading, and Interleaving Data”. In: *Proceedings of the Pharmaceutical SAS® Users Group Conference*. Tutorials, 8 pp.; topic: retain and lag functions; info: merge, reset of values to missing, DOW-loop: until(last.var). URL: http://pharmasug.org/2005/TU09.pdf (cit. on p. 9).

Fehd, Ronald J. (2007). “Do Which? Loop, Until or While? A Review Of Data Step And Macro Algorithms”. In: *Proceedings of the SAS® Global Forum Conference*. Coders Corner, 8 pp. URL: http://www2.sas.com/proceedings/forum2007/067-2007.pdf (cit. on p. 10).

Whitlock, Ian (2000). “Re: SAS novice question”. In: *Archives of the SAS-L listserve*. first use of do until(last.id). URL: http://www.listserv.uga.edu/cgi-bin/wa?A2=ind0002C&L=sas-l&P=R5155 (cit. on p. 8).

This paper was first published in

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

appendix: program listings

do-loops-data.sas

```
*name: do-loops-data.sas;
data _null_;
retain i j k l 0;
*initial:; i = 1;
loop:      put i=;
           if i eq 2 then goto done;
*iterate:; i+ +1;
*repeat :; goto loop;
done:      put 'loop-repeat: ' i=;

j = 1;
do while(j lt 3);
   put j=;
   j+ +1;
   end;
put 'do j: ' j=;

k = 1;
do until(k ge 3);
   put k=;
   k+ +1;
   end;
put 'do k: ' k=;

l = 0;
do l = 1 to 2;
   put l=;
   end;
put 'do l: ' l=;
stop;
run;
```

do-loops-macro.sas

```
*name: do-loops-macro.sas;
%macro do_loops();
%local i;  %let i = 1;
%loop:     %put i=&i.;
%*test;    %if &i. eq 2 %then %goto done;
%*iterate; %let i = %eval(&i. +1);
%*repeat;  %goto loop;
%done:     %put loop-repeat: i=&i.;

%local j; %let j = 1;
%do %while(&j lt 3);
    %put j=&j.;
    %let j = %eval(&j. +1);
    %end;
%put do j: j=&j.;

%local k; %let k = 1;
%do %until(&k ge 3);
```

```
    %put k=&k.;
    %let k = %eval(&k. +1);
    %end;
%put do k: k=&k.;

%local l;
%do l = 1 %to 2;
    %put l=&l;
    %end;
%put do l: l=&l.;
%mend;
%do_loops
```

do-loop-skips.sas

```
*name: do-loop-skips.sas;
data _null_;
do i = 1 to 3;
   put i= 'pre-test';
   if  i le 2 then continue;
   put i= 'post test';
   end;
put 'done continue: ' i= ;

do j = 1 to 3;
   put j= 'pre-test';
   if  j gt 2 then leave;
   put j= 'post test';
   end;
put 'done leave: ' j= ;
stop;
run;

%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. ;
%mend;
%do_tests
```

do-boolean.sas

```
*name: do-boolean.sas;
data do_until_endofile;
do until(endofile);
   set sashelp.class
       end = endofile;
   output;
   end;
stop;

data do_while_not_endofile;
do while(not endofile);
   set sashelp.class
       end = endofile;
   output;
   end;
stop;
run;
```

do-I-eq-until.sas

```
*name: do-i-eq-until.sas;
data   do_i_eq_until;
       retain done 0;
do i = 1 to 3   until(done);
   put i=;
   output;
   done = (i ge 2);* returns boolean (0,1);
   end;
put 'done: ' i=;
stop;
run;
```

sas-l-post-052734.txt

```
http://www.listserv.uga.edu/cgi-bin/wa?A2=ind0002C&L=sas-l&P=R5155

>>> Posting number 52734, dated 16 Feb 2000
Date:         Wed, 16 Feb 2000 13:41:18 -0500
Sender:       "SAS(r) Discussion" <SAS-L@LISTSERV.UGA.EDU>
From:         WHITLOI1 <retired>
Subject:      Re: SAS novice question

Jeremy,

SAS is a record oriented language so processing variables
on different records at the same time
usually means restructureing the data
to produce the required variables on one record.

The problem you posed has a simpler solution,
because the records of interest happen to be adjacent.
Take a look at the LAG and DIF functions.
```

```
On the other hand, learning how to manipulate data
in arrays and restructure data
will prove more valuable in the long run.

Here is code to illustrate getting all data on one record
where you have easy access

data w ;
  input id  method  var1  var2 var3  ;
cards ;
1       1          2         3          2
1       2          3         4          6
2       1          1         5          4
2       2          2         2          6
;

data t ( keep = id v1 - v6 ) ;
   array v (2,3) ;
   array var ( 3 ) ;
   do until ( last.id ) ;
      set w ;
      by id ;
      do i = 1 to dim ( var ) ;
         v ( method , i ) = var ( i ) ;
      end ;
   end ;
run ;

data q ( keep = id diff1 - diff3 ) ;
   set t ;
   array diff ( 3 ) ;
   array v (2,3) ;

   do i = 1 to hbound ( v , 2 ) ;
      diff ( i ) = v ( 2 , i ) - v ( 1 , i ) ;
   end ;
run ;

Of course, steps two and three could be combined in one step.
I left them separate so that it would be easier to study them.

Ian Whitlock <whitloi1@westat.com

____________________Reply Separator____________________
Subject:    SAS novice question
Author: H Jeremy Bockholt <jeremy-bockholt@UIOWA.EDU>
Date:       2/15/2000 12:06 PM

In SAS, is there a way to directly access a value
of a variable when there is more then one value?

example record
--------------------------------
id  method  var1  var 2 ... var10
1       1          2         3          2
1       2          3         4          6
```

```
2       1          1         5          4
2       2          2         2          6
------------------------------

I have a method variable with 2 levels{1,2},
so each of my var1-var10 has 2 values for each id.

I want to be able to refer directly to var1
at a specific method level.
Let's say I want to calculate

diff=var1[method==1] - var[method==2]

Is there an easy way to do this?

thanks in advance for any advice,
jeremy
```

---

do-Whitlock.sas

```
*name: do-Whitlock.sas;
proc sort data = sashelp.class
          out  =         class;
          by     sex name;

data do_whitlock_last;
do until(last.sex);
   set class;
   by  sex;
   end;
output;
put sex= name=;
run;

data do_whitlock_first;
do until(first.sex);
   set class;
   by  sex;
   end;
output;
put sex= name=;
run;

options fullstimer;
data do_dorfman_double_class;
do until(first.sex);
   set class;
   by  sex;
   end;
put sex= name=;
output;

do until(last.sex);
   set class;
   by  sex;
   end;
```

```
output;
put sex= name=;
run;

data do_if_first_or_last_class;
set class;
by  sex;
if first.sex or last.sex then do;
   put sex= name=;
   output;
   end;
run;
```

do-double-dow.sas

```
*name: do-double-dow.sas;
options fullstimer;
proc sort data = sashelp.class
          out  =         class;
          by     sex name;

data do_until_twice;
do until(first.sex);
   set class;
   by  sex;
   end;
put sex= name=;
output;

do until(last.sex);
   set class;
   by  sex;
   end;
put sex= name=;
output;
run;

data do_if_first_or_last;
set class;
by  sex;
if first.sex or last.sex then do;
   put sex= name=;
   output;
   end;
run;
```