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.

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,

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
This paper contains the following topics.

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>2</td>
</tr>
<tr>
<td>Do Which?</td>
<td>3</td>
</tr>
<tr>
<td>Data Step Loops</td>
<td>3</td>
</tr>
<tr>
<td>Macro Loops</td>
<td>4</td>
</tr>
<tr>
<td>Testing During Loop</td>
<td>5</td>
</tr>
<tr>
<td>Using Logic In Conditions</td>
<td>6</td>
</tr>
<tr>
<td>Conclusion</td>
<td>8</td>
</tr>
<tr>
<td>Bibliography</td>
<td>9</td>
</tr>
</tbody>
</table>
INTRODUCTION

Overview

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 listserv 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.

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

Dorfman [3] discusses the details of the iteration 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?

Discussion

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, line 5, whereas the until-condition is evaluated at the bottom of the loop, test, 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.

DATA STEP LOOPS

Examples

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

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>I = 1;</td>
<td>*initial:</td>
</tr>
<tr>
<td>3</td>
<td>put I=;</td>
<td>loop:</td>
</tr>
<tr>
<td>4</td>
<td>if I eq 2 then goto done;</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>I+ +1;</td>
<td>*iterate:</td>
</tr>
<tr>
<td>6</td>
<td>done: put 'loop-repeat: ' I=;</td>
<td></td>
</tr>
</tbody>
</table>

do-loops.log

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td>I=1</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>I=2</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>loop-repeat: I=2</td>
<td></td>
</tr>
</tbody>
</table>

Do While: The while test is evaluated at the top of the loop.

do-loops.sas

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>J = 1;</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>do while(J lt 3);</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>put J=;</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>J+ +1;</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>end;</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>put 'do J: ' J=;</td>
<td>do J: J=3</td>
</tr>
</tbody>
</table>

do-loops.log

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>J=1</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>J=2</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>do J: J=3</td>
<td></td>
</tr>
</tbody>
</table>

Do Until: The until test is evaluated at the bottom of the loop.

do-loops.sas

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>K = 1;</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>do until(K ge 3);</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>put K=;</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>K+ +1;</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>end;</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>put 'do K: ' K=;</td>
<td>do K: K=3</td>
</tr>
</tbody>
</table>

do-loops.log

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>57</td>
<td>K=1</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>K=2</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>do K: K=3</td>
<td></td>
</tr>
</tbody>
</table>

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.

L=0

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>L = 1 to 2;</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>put L=;</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>end;</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>put 'do L: ' L=;</td>
<td>do L: L=3</td>
</tr>
</tbody>
</table>

do-loops.log

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>L=1</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>L=2</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>do L: L=3</td>
<td></td>
</tr>
</tbody>
</table>
MACRO LOOPS

Examples

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

Loop Repeat: loop: ... goto loop

```sas
%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.;
```

Do While: do while(J le 3)

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

Do Until: do until(K ge 3)

```sas
%local K; %let K = 1;
%do %until(&K ge 3);
%put K=&K.;
%let K = %eval(&K. +1);
%end;
%put do K: K=&K.;
```

Do Iterate: do L = 1 to 2

```sas
%local L;
%do L = 1 %to 2;
%put L=&L;
%end;
%put do L: L=&L.;
```
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

```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= ;
```

**Leave:** conditional exit

```sas
do J = 1 to 3;
   put J= 'pre-test';
   if J gt 2 then leave;
   put J= 'post test';
end;
put 'done leave: ' J= ;
```

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

```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. ;
```

**Leave:** conditional exit

```sas
%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. ;
```
USING LOGIC IN CONDITIONS

Using Boolean Logic

The following are equivalent:

```sas
do while(not EndoFile)
do until(EndoFile)
```

This is an important difference to understand: that the same algorithm can be implemented 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 preceding statement means that only zero (0) is false; `until` and `while` evaluation treats negative as well as positive values as True.

**Do Until EndoFile**: evaluate EndoFile at bottom of loop

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

**Do While Not EndoFile**: evaluate EndoFile at top of loop

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

Combining Iteration With Loop Control

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

```sas
DATA do_I_eq_until;
*initial; retain Done 0;
do I = 1 to 3
  until(Done);
put I=;
output;
Done = (I ge 2);%*boolean;
end;
put 'done: ' I=;
stop; run;
```

```sas
NOTE: There were 17 observations read
from the data set SASHELP.CLASS.
NOTE: The data set WORK.DO_UNTIL_ENDOFILE has
17 observations and 5 variables.
```

```sas
NOTE: There were 17 observations read
from the data set SASHELP.CLASS.
NOTE: The data set WORK.DO_WHILE_NOT_ENDOFILE has
17 observations and 5 variables.
```

```sas
DATA do_I-eq-until.sas
DATA do_I-eq-until.log
36 I=1
37 I=2
38 done: I=2
```

```sas
36 I=1
37 I=2
38 done: I=2
```
Ian Whitlock [sasl.52734 6 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.)

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

The DOW can be used with first.by-var as well.
Paul Dorfman and Howard Schrier have posted several examples to SAS-L using the DOW algorithm and showing expanded usages. Here is an example.

```sas
DATA do_double_Class;
  do until(EndoFile);
    do until(first.Sex);
      set Class end = EndoFile;
      by Sex;
      end;
    put Sex= Name=;
    output;
    do until(last.Sex);
      set Class end = EndoFile;
      by Sex;
      end;
    put Sex= Name=;
    output;
  end;
  stop; run;
```

**CONCLUSION**

**Summary**

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


To get the code examples in this paper search www.sascommunity.org for Do-Which-Loop-Until-or-While.zip.

This paper was first published in Fehd [5] sgf2007.067.

About the author:

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

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. In the USA and other countries ® indicates USA registration.

To get the code examples in this paper search www.sascommunity.org for Do-Which-Loop-Until-or-While.zip.

This paper was first published in Fehd [5] sgf2007.067.

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