Chaining Logic in One Data Step  
Libing Shi, Ginny Rego  
Blue Cross Blue Shield of Massachusetts, Boston, MA

ABSTRACT  
Event dates stored in multiple rows pose many challenges that have typically been resolved by lengthy SAS® code. This is particularly true if the earliest start date, the final end date and the total number of event days must all be determined. This paper examines how to use a RETAIN statement in conjunction with FIRST.var and LAST.var processing to find and keep the earliest start date and final end date for an event. At the same time, DO loops and IF/THEN statements are incorporated to track the number of days the event occurred as well as any “gap” days, i.e. days between the start and end date on which the event did not occur. All this information can be obtained through one easy to use and understand data step.

INTRODUCTION  
In the health care field new data is collected and added to existing data on a daily basis. As healthcare data analysts we often need to look through many rows of data to find the beginning, end and duration of an event. The challenge is compounded since rows may represent different time periods. In the case of enrollment data, each month of enrollment is represented by one row. Locating a row that shows a member’s enrollment began in January and ended in December is a good start, but what was the actual duration of enrollment 45 days or 365 days between the start and end dates? Hospital admissions represent a similar situation. Each new admission is represented by a separate row of data. Multiple rows must be evaluated if we want to know the total number of days spent as an inpatient in a specified time period. For pharmacy data we may be asked to calculated medication regime adherence with multiple rows of prescription information.

By collapsing overlapping event dates and “chaining” together abutting event dates that are stored in different rows we are able to pinpoint the earliest start and latest end dates and summarize the total number of enrollment months, inpatient days or days a patient had a medication. Although the examples outlined in this paper are health care related, we believe this data step is versatile enough to be used in many different applications.

THE DATA  
The sample data below replicates member enrollment data. Each row is one enrollment segment. Enrollment segments usually begin on the first day of the month and end on the last day of the month. However, irregular rows that may have dates that overlap the previous enrollment segment or have gaps between the end of the previous segment and the beginning of the following segment can occur when certain changes to a member’s information takes place.

In the sample, we have limited the data to four key variables.

- The variable mem_num is a unique combination of 16 numerals for each member.
- The variable model indicates the type of coverage a member has. Model may remain constant or change from month to month.
- The variable cov_eff_dt is the date coverage became effective for each enrollment period.
- The variable cov_term_dt is the date coverage terminated for each enrollment period.

<table>
<thead>
<tr>
<th>mem_num</th>
<th>model</th>
<th>cov_eff_dt</th>
<th>cov_term_dt</th>
</tr>
</thead>
<tbody>
<tr>
<td>4440888130000001</td>
<td>HMO</td>
<td>06/01/2007</td>
<td>06/10/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>HMO</td>
<td>06/11/2007</td>
<td>06/30/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>PPO</td>
<td>06/18/2007</td>
<td>06/30/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>PPO</td>
<td>07/01/2007</td>
<td>07/31/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>PPO</td>
<td>08/01/2007</td>
<td>08/31/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>HMO</td>
<td>09/01/2007</td>
<td>09/30/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>HMO</td>
<td>10/01/2007</td>
<td>10/19/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>HMO</td>
<td>10/01/2007</td>
<td>10/31/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>HMO</td>
<td>11/01/2007</td>
<td>11/30/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>HMO</td>
<td>12/01/2007</td>
<td>12/02/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>HMO</td>
<td>12/03/2007</td>
<td>12/31/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>HMO</td>
<td>01/01/2008</td>
<td>01/12/2008</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>HMO</td>
<td>01/15/2008</td>
<td>01/31/2008</td>
</tr>
</tbody>
</table>
PREPARE DATA BY SORTING

It is critical that the data be sorted by the grouping variables (the variables that will be used in the BY statement in the data step) and the event dates before the chaining data step is implemented. For all grouping variables the dates must be in chronological order or the output data set will be incorrect. Here are two examples of how different sort orders will affect the output data set:

**Example A:** Sorting by \textit{mem\_num} (as the grouping variable) and \textit{cov\_eff\_dt} and \textit{cov\_term\_dt}

```plaintext
PROC SORT DATA = NESUG_test_enroll
    OUT = temp (KEEP = mem\_num  cov\_eff\_dt  cov\_term\_dt);
    BY mem\_num  cov\_eff\_dt  cov\_term\_dt;
```

<table>
<thead>
<tr>
<th>mem_num</th>
<th>cov_eff_dt</th>
<th>cov_term_dt</th>
</tr>
</thead>
<tbody>
<tr>
<td>4440888130000001</td>
<td>06/01/2007</td>
<td>06/10/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>06/11/2007</td>
<td>06/30/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>06/18/2007</td>
<td>06/30/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>07/01/2007</td>
<td>07/31/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>08/01/2007</td>
<td>08/31/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>09/01/2007</td>
<td>09/30/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>10/01/2007</td>
<td>10/19/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>10/01/2007</td>
<td>10/31/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>11/01/2007</td>
<td>11/30/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>12/01/2007</td>
<td>12/02/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>12/03/2007</td>
<td>12/31/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>01/01/2008</td>
<td>01/12/2008</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>01/15/2008</td>
<td>01/31/2008</td>
</tr>
</tbody>
</table>

**Example B:** Sorting by \textit{mem\_num} and \textit{model} (as the grouping variables) and \textit{cov\_eff\_dt} and \textit{cov\_term\_dt}

```plaintext
PROC SORT DATA = NESUG_test_enroll
    OUT = temp (KEEP = mem\_num model  cov\_eff\_dt  cov\_term\_dt);
    BY mem\_num model  cov\_eff\_dt  cov\_term\_dt;
RUN;
```

<table>
<thead>
<tr>
<th>mem_num</th>
<th>model</th>
<th>cov_eff_dt</th>
<th>cov_term_dt</th>
</tr>
</thead>
<tbody>
<tr>
<td>4440888130000001</td>
<td>HMO</td>
<td>06/01/2007</td>
<td>06/10/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>HMO</td>
<td>06/11/2007</td>
<td>06/30/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>HMO</td>
<td>06/18/2007</td>
<td>06/30/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>HMO</td>
<td>07/01/2007</td>
<td>07/31/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>HMO</td>
<td>10/01/2007</td>
<td>10/19/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>HMO</td>
<td>10/01/2007</td>
<td>10/31/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>HMO</td>
<td>11/01/2007</td>
<td>11/30/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>HMO</td>
<td>12/01/2007</td>
<td>12/02/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>HMO</td>
<td>12/03/2007</td>
<td>12/31/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>HMO</td>
<td>01/01/2008</td>
<td>01/12/2008</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>HMO</td>
<td>01/15/2008</td>
<td>01/31/2008</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>PPO</td>
<td>07/01/2007</td>
<td>07/31/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>PPO</td>
<td>08/01/2007</td>
<td>08/31/2007</td>
</tr>
<tr>
<td>4440888130000001</td>
<td>PPO</td>
<td>09/01/2007</td>
<td>09/30/2007</td>
</tr>
</tbody>
</table>

For this example, if we sort by both \textit{mem\_num} and \textit{model} the \textit{cov\_eff\_dt} and \textit{cov\_term\_dt} dates will no longer be in chronological order. The final output will calculate the total enrolled days, gap days and breaks respectively for the rows where \textit{model} equals “HMO.” However, the rows where \textit{model} equals “PPO” will not be counted correctly. Make sure that only keep the variables needed for the chain in the output dataset after sorting. Every variable in the sorted output data set will affect the final results.

Depending on the project and the desired output, knowing how long someone was enrolled by some other variable in the data set such as \textit{model} may be important. In Appendix A we have provided and example of how to calculate enrollment by \textit{model} and \textit{mem\_num}. 
Once the data is correctly sorted, here are a few possible scenarios that may be encountered:
1. The enrollment segment in one row overlap the segment in another row.
2. The dates of one enrollment row abut the dates in the next enrollment row.
3. Enrollment dates in one row are within the dates of another row.
4. There are gap days between enrollment segments.
5. There is only one enrollment segment for a particular member.

CHAIN THE DATA
Now that the data are sorted we are ready to begin evaluating the relationship of the dates in each row and get down to answering the question of when coverage began and ended and how many days each member was enrolled between the earliest `cov_eff_dt` and latest `cov_term_dt`.

First we name the output data set that will contain all the chained enrollment information for each member. We also DROP variables from the output data set that will not contain summary level information.

```plaintext
DATA Mbr_Enroll (DROP = cov_eff_dt cov_term_dt newstart);
SET temp;
```

Next we use a BY statement to initialize FIRST.var and LAST.var processing. The order of variable(s) in this BY statement must be consistent with the by grouping variable(s) in the previous PROC SORT statement. Otherwise your final output dataset may contain misleading results.

```plaintext
BY mem_num;
```

A RETAIN statement is used to hold the value of variables over different iterations of the data step. We will also need to create new variables and retain their values in order to make comparisons over multiple rows. See the bullets below for an explanation of the function of the newly created variables.

```plaintext
RETAIN startElig newstart endElig gap break tot_mem_days;
```

- `startElig`: the earliest `cov_eff_dt` found for each member
- `newstart`: records the `cov_eff_dt` when a gap exists between enrollment segments
- `endElig`: the last `cov_term_dt` found for each member
- `gap`: contains the total number of days for which the member did not have coverage between the `cov_term_dt` and `cov_end_dt`.
- `break`: contains the number of times gap days appeared between enrollment segments
- `tot_mem_days`: cumulating the total days of enrollment for a group

FIRST `mem_num` identifies the first row for the member. Set the initial value of gap and break equal to zero to indicate no gaps or breaks have occurred yet. Set `startElig` equal to the `cov_eff_dt`; sorting the data in the prior to the data step assures us that the first row for a member contains the earliest enrollment start date. In this step `newstart` is also set to the `cov_eff_dt`, although it may be changed in later statements within the data step. For now `endElig` is set to the `cov_term_dt` as it is the last enrollment termination date we have up to this point.

```plaintext
IF FIRST. mem_num THEN DO;
   tot_mem_days=0;
   gap=0;
   break=0;
   startElig=cov_eff_dt;
   newstart=cov_eff_dt;
   endElig=cov_term_dt;
END;
```

This statement resolves scenarios where one enrollment segment overlaps the previous but has a greater `cov_term_dt` than the previous enrollment segment. We will keep all the start dates we set in the previous step but we will keep the new `cov_term_dt`.

```plaintext
ELSE IF cov_eff_dt=newstart AND cov_eff_dt< endElig AND cov_term_dt>EndElig
THEN
   endElig=cov_term_dt;
```
When the two enrollment segments abut each other then we keep all the start dates set-up in the first IF statement and reset the endElig to the cov_term_dt.

```
ELSE IF cov_eff_dt = endElig+1 THEN  endElig=cov_term_dt;
```

**NOTE:** If the cov_eff_dt and cov_term_dt in one row are inside the cov_eff_dt and cov_term_dt of another row, then we need to eliminate the row. From a programming standpoint we do not need to do anything to accomplish this task. This scenario will not meet any of the criteria set forth in the statements in this data step and is effectively deleted.

When there are gap days between enrollment segments, we need to:
1. Reset all of our newly created variables except startElig
2. Add days between cov_eff_dt and endElig to any previously calculated gap days
3. Add one to the count of break
4. Update newstart and endElig to current cov_eff_dt and cov_term_dt.

**NOTE:** Always calculate gap and breaks before resetting newstart and endElig.

```
ELSE IF cov_eff_dt>endElig+1 THEN DO;
gap = gap+ (cov_eff_dt-endElig-1);  break+1;
newstart=cov_eff_dt;
endElig=cov_term_dt;
END;
```

Each row will be compared to the retained enrollment information until LAST.mem_num is reached. LAST.mem_num is the last enrollment row for that member. Once the last record is reached, we will calculate the total number of enrollment days minus any gap days.

```
if last.mem_num then
tot_mem_days=endElig-startElig -gap +1;
```

Only the last row for each member will be outputted to the data set. We will also format our newly created date fields which would be SAS dates in our new data set if we did not format them.

```
if last.mem_num;
format startElig endElig mmdyy8.;
run;
```

Depending on what information we’ve been asked to report, we may want to keep enrollment segments for which there are gaps between in separate rows for each member or we may want to summarize all the enrollment information into one row for each member. The SAS codes in Appendix A will provide solutions for different real world requests.

**CONCLUSION**
The data set created as a result of this data step will contain the summarized enrollment information for each member. This data can be used in many different ways, including reporting the average length of enrollment of members in this population or for selecting members for inclusion in a study population. Usually the variables that need to be chained are date/time variables. However, it is possible to use the same logic to chain character variables together. For instance, medical procedures can be chained by the order of treatment for a certain disease. The logic behind this small piece of code is straightforward and can easily be applied to many different scenarios. It will run smoothly in Window and UNIX platform. Just be sure that you sort the data properly, before you start to chain things together.
ACKNOWLEDGEMENTS
Special thanks to Jean Murphy and Michele DiPaulo for their invaluable feedback on the paper content and John Dawson for his consultation and help developing this code.

CONTACT INFORMATION
We welcome any comments or questions. Please contact the authors at:

Libing Shi
e-mail: Libing.Shii@bcbsma.com

Virginia Rego
e-mail: Virginia.Rego@bcbsma.com

Blue Cross Blue Shield of Massachusetts
401 Park Drive
Boston, MA 02215

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.
APPENDIX
%macro chain (indata=, /*name of input SAS dataset*/
               start_dt=, /*start date of the event*/
               end_dt=,  /*end date of the event*/
               outdata=, /*name of output SAS dataset*/
               bygrpvar=, /*variable names for grouping the chain*/
               test=)   /*a test pointer to decide the output data contents*/
               ;
/*Sort the data before chaining*/
Proc sort data = &indata.
   out = temp (keep = &bygrpvar. &start_dt &end_dt);
   by &bygrpvar. &start_dt &end_dt;
run;

%let lastgrpvar=%scan(&bygrpvar.,-1); /*the last variable of the grouping*/
%IF &test= summary %THEN %DO;
   DATA &outdata._&test. (drop = &start_dt &end_dt newstart);
      set temp;
      by &bygrpvar. ;
      retain startElig newstart endElig gap break tot_mem_days;
      IF FIRST.&lastgrpvar. THEN DO;
         tot_mem_days=0;
         gap=0;
         break=0;
         startElig=&start_dt;
         newstart=&start_dt;
         endElig=&end_dt;
      END;
      ELSE IF &start_dt>=newstart and &start_dt<= endElig and &end_dt>EndElig THEN
         endElig=&end_dt;
      ELSE IF &start_dt = endElig+1 THEN endElig=&end_dt;
      ELSE IF &start_dt>endElig+1 THEN DO;
         gap = gap+ (&start_dt-endElig-1);
         break+1;
         newstart=&start_dt;
         endElig=&end_dt;
      END;
      if last.&lastgrpvar. then tot_mem_days=endElig-startElig -gap +1;
      if last.&lastgrpvar.;
      format startElig endElig mmddyy8.;
      %END;
%ELSE %IF &test=detail %THEN %DO;
   DATA &outdata._&test. (drop = &start_dt &end_dt);
      set temp;
      by &bygrpvar. ;
      retain startElig endElig gap break tot_mem_days;
      IF FIRST.&lastgrpvar. THEN DO;
         tot_mem_days=0;
      END;
%END;
gap=0;
break=0;
startElig=&start_dt;
endElig=&end_dt;
END;
ELSE IF &start_dt>=startElig and &start_dt<= endElig and &end_dt>endElig THEN DO;
    endElig=&end_dt;
    tot_mem_days=endElig-startElig +1;
    END;
ELSE IF &start_dt = endElig+1 THEN DO;
    endElig=&end_dt;
    tot_mem_days=endElig-startElig +1;
    END;
ELSE IF &start_dt>endElig+1 THEN DO;
    gap = &start_dt-endElig-1;
    break=1;
    startElig=&start_dt;
    endElig=&end_dt;
    tot_mem_days=endElig-startElig +1;
    END;
format startElig endElig mmddyy8.;
proc sort; by &bygrpvar. startElig descending endElig;
%END;
proc sort nodupkey; by &bygrpvar. startElig;
run;
%mend;
%chain (indata=NESUG_test_enroll, start_dt=cov_eff_dt, end_dt=cov_term_dt, outdata=enroll, bygrpvar=mem_num, test=summary);
%chain (indata=NESUG_test_enroll, start_dt=cov_eff_dt, end_dt=cov_term_dt, outdata=enroll2, bygrpvar=mem_num model, test=summary);
%chain (indata=NESUG_test_enroll, start_dt=cov_eff_dt, end_dt=cov_term_dt, outdata=enroll2, bygrpvar=mem_num model, test=detail );