Paper 103-2010

Look Before You Leap – A Technique to Simulate a Look-Ahead Function

Leonard Landry, Statistics Canada, Ottawa, Ontario, Canada

ABSTRACT

In base SAS® programming there is a very useful function called LAG which allows a SAS program to reference previous values of a SAS program variable. Wouldn't it be wonderful if a similar function were available to look ahead at the value of a dataset field in the next observation. There is no such function available but with some simple programming logic a "look ahead" functionality can be achieved. This paper will describe a technique that can be used to achieve this.

INTRODUCTION

Most SAS programmers are familiar with the LAG function. It is a simple and easy to use function that can be used in a SAS data step to return the value of the argument for a previous observation. By specifying LAGn the function can return the value lagged by n observations. Unfortunately there is no corresponding function to look ahead at the value of a dataset field in the next or subsequent observations. There are many practical cases where this functionality would be a useful feature.

The purpose of this paper is to describe two variations of a programming technique that can be used to achieve a "look ahead" functionality. The paper will explain how and why the technique works along with a simple example of each technique. This will be followed by a more practical example where the author used this technique in a real programming situation.

THEORY

When SAS reads a dataset sequentially it maintains a pointer within the dataset which it moves ahead by one on each read. The same dataset may be referenced more than once in a SAS data step and for each reference SAS will maintain a separate pointer to the dataset. If we read the same dataset twice using two SET statements with the pointer for the second read maintained one observation ahead of the first read then we are able to simulate a "look ahead" function. This is achieved by using the FIRSTOBS option on the second SET statement and specifying a value of FIRSTOBS=2.

Other factors must be considered. On the "look ahead" read we must only read the fields that we want to look ahead at and we must rename these fields so that they are brought into the program data vector as unique fields. Also the "look ahead" read will trigger the EOF indicator when it reads the last record in the dataset so some special coding is required to read the last record.

Another alternative to using two SET statements is to perform a one to one merge of the dataset with itself and use the FIRSTOBS option on one of the dataset names to maintain that pointer one record ahead of the other. This technique has the advantage of not requiring any special processing to read the last record.

EXAMPLES

The objective of the first two examples is to read a SAS dataset and output a new dataset with one additional field added. The new field will be the value of one of the fields in the next observation. It follows logically that the value of this new field for the last observation will be missing since there is not another observation to look ahead to. The two examples use the same dataset as input and both produce the expected results.

EXAMPLE 1 – Using two SET Statements

In this example we are using two SET statements to read the same dataset twice with each iteration of the data step. On the second read we will maintain the pointer one record ahead of the first. We only want to look ahead at the value of the field VAR2 so we will specify this field in a KEEP option and we will rename the field to NEXTVAR2 with a RENAME option. Below is the sample data and program code.

DATA

OBS	VAR1	VAR2
1	1	A
2	2	B
3	3	C
4	4	D
5	5	E

Program 1

```
*************************************************************
* This is the first technique – using two SET statements
*************************************************************;

data test2;
   if eof=0 then set test(firstobs=2 keep=var2
                          rename=(var2=nextvar2))end=eof;
            else nextvar2=' ';
   set test;

run;
```

Results

OBS	VAR1	VAR2	NEXTVAR2
1	1	A	B
2	2	B	C
3	3	C	D
4	4	D	E
5	5	E	.

This program works but notice that we must be careful not to execute the SET statement that is pointing ahead once it has reached end of file. We must put this SET statement in a conditional IF and execute it only when EOF=0. Also we must manually set the NEXTVAR2 to missing for the last record.

EXAMPLE 2 – Merging to itself

This is perhaps a simpler solution as we do not have to worry about the end of file problem. With a merge the end of file condition is reached only when the last record is read from all files being merged. We also do not have to manually set the NEXTVAR2 to missing for the last record. The code for this option and the results are shown below.

Program 2

```
*************************************************************
* This is the second technique - merging a file to itself
*************************************************************;

Data test2;
  merge test test(firstobs=2 keep=var2 rename=(var2=nextvar2));
run;
```

Results

OBS	VAR1	VAR2	NEXTVAR2
1	1	A	B
2	2	B	C
3	3	C	D
4	4	D	E
5	5	E	

EXAMPLE 3 – A Practical Application

This example will describe a situation in which the author was able to use this "look ahead " technique to solve a routine request in his day to day work.

The author works with a group of research analysts who do research work in the Canadian Labour Market. One of the files that is often used by these researchers is a longitudinal file spanning 1983 to 2007 that contains information on workers and their earnings. The researcher wanted to know the number of workers in this file in each year who had income in that year and none in the next year. The file was consistent across all years for the included group of workers and maintained their consistent identifiers. Data would be in the file for a given worker in a given year if and only if he had earnings in that year. The file could contain multiple records for the same worker in a given year if the worker had multiple sources of income. Also some filtering of the data would be required to satisfy other constraints asked for by the researcher.

The solution was to sort the file using the NODUPKEY option by WORKER_ID and YEAR. This would eliminate duplicate records for a given worker in a given year. The required filtering was also performed in the SORT by including a WHERE clause. Next, the sorted file was read sequentially while using a "look ahead" technique to look at the values for the WORKER_ID and YEAR fields in the next observation. If the next observation was for the same worker and the year was not the next sequential year then this represented a wage gap. Also if the next observation was for a different worker and the current year was not 2007 then this also indicated a wage gap. The counts of wage gaps for each year were stored in an array and the appropriate array member was incremented each time a wage gap was encountered. After all records were read the counts stored in the array were written to a SAS dataset and then exported to an EXCEL spreadsheet.

Below are the program code and the results.

Program 3

```
*****************************************************************
* sort file by casenum, year. Also apply some filtering
*****************************************************************;

proc sort nodupkey data=lwf.lwf8307(keep=rectype casenum year age sex)
          out=wagegap;
   where rectype<3 & sex>0 & age>0;
   by casenum year;
run;

*****************************************************************
* find all workers with a wage gap from LWF file
*****************************************************************;

DATA wagegap2(keep=year count_of_wagegaps);
  array counts(1983:2006) 8. _temporary_;

  merge wagegap wagegap(firstobs=2 keep=casenum year
        rename=(casenum=nextcasenum year=nextyear)) end=end1;

  if year < 2007 & ((nextcasenum = casenum & nextyear ne year + 1)
                   | nextcasenum ne casenum)
     then counts(year)+1;

  if end1
     then do year=1983 to 2006;
               count_of_wagegaps = counts(year);
               output;
          end;
run;


***************************
* Export the tables to EXCEL
***************************;

PROC EXPORT DATA=wagegaps2
            OUTFILE= "h:\sastests\data\wagegaps_by_year"
            DBMS=EXCEL2000 REPLACE;
     SHEET="wagegaps";
run;
```

Results

year	count_of_wagegaps
1983	94735
1984	90315
1985	92683
1986	90393
1987	92207
1988	93155
1989	106048
1990	124425
1991	128399
1992	122040
1993	112479
1994	107832
1995	112360
1996	105056
1997	104910
1998	97789
1999	93840
2000	104369
2001	108626
2002	102389
2003	113653
2004	104981
2005	105808
2006	103755

CONCLUSIONS

The examples presented in this paper show that the techniques described for achieving a “look ahead” functionality are simple, easy to use, and work successfully in these types of applications where a file is read sequentially. Of the two techniques presented the author feels that the second one, performing a one to one merge of a file to itself, is the simpler and easier to use. Although the examples shown here only look ahead one observation it should be obvious that the “look ahead” can be any number of observations based on the value used in the FIRSTOBS option.

REFERENCES

- Erik W. Tilanus 2008. “SET, MERGE and Beyond” Proceedings of SAS Global Forum 2008, Paper 167-2008
- SAS Institute Inc. , Combining and Modifying SAS DataSets: Examples, Version 6, First Edition, Cary, NC

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please contact the author at:

Leonard Landry
Database Manager,
Social Analysis Division,
Statistics Canada
leonard.landry@statcan.gc.ca