SCSUG 2012

SAS® HASH Programming basics

Daniel Sakya, PPD Inc., Austin, TX

Abstract

HASH programming is a hot topic in the industry that started with SAS 9. This paper is intended to provide more exposure to novice or experienced SAS programmers that are looking for alternatives to data step programming. The concept of HASH programming is similar to the definition of an array in SAS. Several SAS users have benefitted from HASH programming by considerably reducing the processing time for compound data merging tasks. This paper will entail the introduction to HASH programming, its syntax, a few examples, and last but not least, the benefits on using HASH programming versus regular data step. The examples and syntax illustrated are in SAS 9.2.

Introduction

Data step programming is the cornerstone of SAS programming. However, when dealing with large volumes of data, it data step programming can be cumbersome and slow. This is where hash programming can become advantageous to provide safe, efficient, and straightforward tool for data manipulations including creating and accessing datasets.

Just like the concept of array, hash programming accesses the memory index tables for faster access to data. However, unlike arrays, using hash programming does not need sequential integers to access the data – it can be done via characters or numbers. Also, the data does not need to be sorted – one of the best-selling points.

Hash programming has increasingly become a hot topic that has piqued the interest of many SAS programmers, including yours truly, and continues to become a growing trend in the industry that can be a very useful alternative to data step programming. We will explore the syntax for hash programming in this paper with some examples to pique users' interests into using this object oriented approach in SAS.

Syntax

To declare a hash object, you use the DECLARE statement. The statement will let SAS know that the referenced object is a hash object.

```
declare hash x;
```

After a hash object has been declared, it needs to be initialized. This can be done via the _NEW_ operator:

```
x = _NEW_ hash( );
```

The above line of code creates and initializes the hash object x after it has been declared.

The above two statements can be condensed into one line with the following statement:

```
Declare hash X( );
```

SAS dataset options can also be used while declaring and defining hash objects but it is limited to the following operations:

- Rename
- Where
- Drop or Keep
- Output statement can be used in to specify an output dataset
- Subset observations based on observation number

The rename operation is used in the following example:

```
declare hash X(dataset: 'table (rename  =  (CPEVENT = VISIT) ) );
```

Examples

The following example declares and initializes the hash object. After the declaration, the key and data objects are defined, after which, values are added to the key and data objects.

```
data _null_;
    length patient $5 trtgrp $30;
    if _N_ = 1 then do;
        /* Define and initialize the hash object "eg" */
        declare hash eg;
        eg = _new_ hash();
        /*
            The above two statements can also be condensed into one line:
            declare hash eg();
        */

        /* Define key and data variables */
        x = eg.DefineKey('patient');
        x = eg.DefineData('trtgrp');
        x = eg.DefineDone();

        /* Variables may be uninitialized but the following statement can get rid of the warning */
        call missing(patient,trtgrp);
    end;

    /* Adding values to the key and data below */
    x = eg.add(key: 'X1', data: 'Placebo group');
    x = eg.add(key: 'Y1', data: 'Treatment conc. A');
    x = eg.add(key: 'X2', data: 'Treatment conc. B');
    x = eg.add(key: 'Y2', data: 'Control');

    /* Finding data associated with a key */
    x = eg.find(key: 'X2');
    if (x = 0) then put trtgrp=;
    else put 'Key not found';
run;
```

The above example defines has the hash object “eg” and associates the key object as “patient” and data object as “trtgrp”. The DefineDone statement lets SAS know that all definitions for the hash objects are now complete. The add statements add data to the key and data variables within the hash objects. Once the key and data variables have been defined, they can be accessed via the find statement.

The above code will output the following:

trtrgrp=Treatment conc. B

SAS log results:

```
300  data _null_;
301      length patient $5 trtgrp $30;
302      if _N_ = 1 then do;
303          /* Define and initialize the hash object "eg" */
304          declare hash eg;
305          eg = _new_ hash();
306          /*
307              The above two statements can also be condensed into one line:
308              declare hash eg();
309          */
310
311          /* Define key and data variables */
312          x = eg.DefineKey('patient');
313          x = eg.DefineData('trtgrp');
314          x = eg.DefineDone();
315
316          /* Variables may be uninitialized but the following statement can get rid of the
316! warning */
317          call missing(patient,trtgrp);
318      end;
319
320      /* Adding values to the key and data below */
321      x = eg.add(key: 'X1', data: 'Placebo group');
322      x = eg.add(key: 'Y1', data: 'Treatment conc. A');
323      x = eg.add(key: 'X2', data: 'Treatment conc. B');
324      x = eg.add(key: 'Y2', data: 'Control');
325
326      /* Finding data associated with a key */
327      x = eg.find(key: 'X2');
328      if (x = 0) then put trtgrp=;
329      else put 'Key not found';
330  run;

trtgrp=Treatment conc. B
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Multiple data or key objects can be added to the hash objects by separating the variables by a comma. For example, in the above illustration, multiple data variables can be defined with the following statement:

x = eg.DefineData('trtgrp','trtstdt','trtendt');

The above statement adds "trtgrp", "trtstdt", and "trtendt" to the "eg" hash object.

To add data to the data objects, the following statement can be used:

x = eg.add(key: 'X1', data: 'Control', data: '13FEB2012', data: '21MAR2012');

So why use hash programming?

The above syntax and examples may look a bit complex and scary enough to make you wonder why you should even bother with hash programming. However, there are various benefits of hash programming:

- Efficient and fast use of the virtual memory which can prove very useful when merging multiple datasets with large number of records.
- When merging multiple datasets, they don't need to be sorted as long as keys are properly defined.
- Hash objects can return multiple items unlike array objects that can only return one value.

- Typically, hash objects do the programming tasks faster than traditional data step or SQL programming (depending on the data, the available memory, your environment, etc.)
- Hash objects are also very good for data summarization and can typically execute the job up to twice as fast while utilizing a third of the memory when compared with data step programming.

Conclusion

SAS hash programming is a powerful and efficient object oriented approach for table lookups, merges, data summarization, and sorting purposes. I encourage users to perform and compare data step merges versus hash merges in terms of compilation and execution time. Users will notice that even though it may seem a bit complicated at first, they will write less lines of code and perform faster executions of their programs. There are numerous resources and papers out there that talk about efficiency in execution and provide more details to this wonderful memory based object oriented approach that every SAS user should be familiar with.

References

Hash objects – Why bother?
Barb Crowther
http://www.sas.com/offices/NA/canada/downloads/presentations/HUG09/Hash.pdf

An Introduction to SAS Hash Programming Techniques
Kirk Paul Lafler
http://www.scsug.org/SCSUGProceedings/2011/lafler1/An%20Introduction%20to%20SAS%20Hash%20Programming%20Techniques.pdf

I cut my processing time by 90% using hash tables - You can do it too!
Jennifer K. Warner-Freeman
http://www.nesug.info/Proceedings/nesug07/bb/bb16.pdf

Acknowledgements

I would like to thank SCSUG 2012 for accepting my abstract and giving me the opportunity to present this paper to the audience as well as Jeanina Worden and PPD for the encouragement to proceed with my first paper.

Contact information

Any comments or suggestions are greatly appreciated and can be sent to:

Daniel Sakya
Senior Programmer Analyst
PPD Inc.
7551 Metro Center Dr. Suite 300
Austin, TX 78744
(512) 747-5205
Daniel.sakya@ppdi.com