SAS HASH OBJECT

Tsunsian

09-MAR-2022

Agenda

- Understand SAS Hash Object
- Declaration & Instantiation of SAS Hash Object
- Methods and Attributes of Hash Object
- Hash Iterator Object
- Potential Applications in Clinical Trials Programming

Understand SAS Hash Object

What is a SAS hash object?

A classic description of SAS Hash Object

- The hash object is an in-memory lookup table accessible from the DATA step. A hash object is loaded with records and is only available from the DATA step that creates it. A hash record consists of two parts: a key part and a data part. The key part consists of one or more character and numeric values. The data part consists of zero or more character and numeric values. (Secosky and Bloom, 2007)

Secosky, Jason, and Bloom Janice. (2007). Getting Started with the DATA Step Hash Object.

A more comprehensive look

- a SAS table (a.k.a. hash table) with rows (hash items) and columns (hash variables), the columns can be key and/or data part.
- resides in memory, only created and accessible from the DATA step, lasts only within the scope of the DATA step in which it lives.
- dynamically grows/shrinks at run time.
- uses the dot notation syntax (e.g. `myHash.add()`), made accessible through the DATA Step Component Interface (DSCI).
- all variables in the hash table (key or data) must be defined in the PDV at compile time, but only data variables have ability to update their PDV counterpart.

A more comprehensive look (cont’d)

- a hash table is of type Object (neither num nor char), within the hash table, the key and data can be either numeric or character.
- The data variables can also be objects, e.g. hash of hashes.
- fast operations because of direct memory addressing. (Schacherer, 2015 have some performance tests.) In fact, the speed of search is irrespective of the number of items in the hash table. (Dorfman and Henderson, 2017)
- Drawbacks:
    - Limited by memory size
    - Syntax is different from Base SAS

Dorfman, Paul M., and Henderson, Don. (2017). Beyond Table Lookup: The Versatile SAS® Hash Object.
Schacherer, Chris. (2015). Introduction to SAS® Hash Objects.

Declaration & Instantiation of Hash Object

How to create/define a hash object?

Declaration & instantiation

Method 1:

```
data _null_;
  /* declare & instantiate in one step
*/
  DECLARE HASH myHash();
run;
```

Method 2:

```
data _null_;
  /* declare first */
  DECLARE HASH myHash;
  /* instantiate when used */
  myHash = _NEW_ HASH();
run;
```

Note: Words in ALL CAPS are reserved keywords.

Declaration & instantiation (cont'd)

Method 1:

```
data _null_;
  /* declare & instantiate in one step */
  DECLARE HASH myHash(<argument_tag-1: value-1, ...argument_tag-n: value-n>);
run;
```

Method 2:

```
data _null_;
  /* declare first */
  DECLARE HASH myHash;
  /* instantiate when used */
  myHash = _NEW_ HASH(<argument_tag-1: value-1, ...argument_tag-n: value-n>);
run;
```

Note: Words in ALL CAPS are reserved keywords.

Argument tags in the instantiation

Argument tag	Valid argument tag values	Description
DATASET	*'dataset_name <(datasetoption)>'*	Specifies the name of a SAS data set to load into the hash object. Allows renaming variables, WHERE subsetting, etc.
DUPLICATE	*'replace' | 'r': stores the last duplicate key record.* *'error' | 'e': reports an error to the log if a duplicate key is found.*	Determines how to handle duplicate keys if they are not allowed. The default is to store the first key and ignore subsequent duplicates.
MULTIDATA	*'YES' | 'Y': Multiple data items are allowed for each key.* *'NO' | 'N': Only one data item is allowed for each key.* *Default is NO.*	specifies whether multiple data items are allowed for each key.
ORDERED	*'ascending' | 'a': Data is returned in ascending key-value order.* *'descending' | 'd': Data is returned in descending key-value order.* *'YES' | 'Y': same as specifying 'ascending'.* *'NO' | 'N': Data is returned in some undefined order.* *Default is NO.*	Specifies whether or how the data is returned in key-value order if you use the hash object with a hash iterator object or if you use the hash object OUTPUT method.

Argument tags in the instantiation (cont'd)

Argument tag	Valid argument tag values	Description
HASHEXP	*n: Default is 8, which equates to a hash table size of 2^8 or 256. The maximum value for HASHEXP is 20.*	The hash object's internal table size, where the size of the hash table is 2^n. The hash table size is not equal to the number of items that can be stored.
SUMINC	*'variable_name'*	maintains a summary count of hash object keys (i.e. key summary). The SUMINC argument tag is given a DATA step variable, which holds the sum increment. The sum increment is how much to add to the key summary for each reference to the key.
KEYSUM	*'variable_name'*	specifies the name of a variable that tracks the key summary for all keys. A key summary is a count of how many times a key has been referenced on a FIND method call. The KEYSUM variable is part of the output data set and works when one or more data items exist for a key.

A taste of SAS Hash Object

Construct an empty hash table and fill it later:

```
data _null_;
  length k $1 d 8;
  DCL HASH h();
  h.DEFINEKEY('k');
  h.DEFINEDATA('d');
  h.DEFINEDONE();
  k='A'; d=100; h.ADD();
  h.ADD(KEY:'B',DATA:80);
  h.OUTPUT(DATASET:'work.simple');
run;
```

Construct and fill a hash table with a SAS data set:

```
data ae;
  if _n_=1 then do;
    if 1>2 then set sdtm.dm(keep=rfstdtc);
    DCL HASH h(DATASET: 'sdtm.dm');
    h.DEFINEKEY('usubjid');
    h.DEFINEDATA('rfstdtc');
    h.DEFINEDONE();
  end;
  set myAE; * assuming USUBJID is present in
myAE ;
  rc=h.FIND();
  if rc ne 0 then call missing(rfstdtc);
run;
```

Note: Words in ALL CAPS are reserved keywords.

Note: *rc* is a user-defined variable to store the return code of a method call, see more details later.

Methods and Attributes of Hash Object

What operations can we do on a hash object?

Methods of Hash Object

- **DEFINEKEY**: defines the key variables, required for the hash object constructor
- **DEFINEDATA**: defines the data variables, if omitted, are automatically assumed to the same as keys.
- **DEFINEDONE**: indicates key and data definitions are complete, required for the constructor. If the DATASET argument tag is used, this is when the dataset is loaded into the hash object.

Return Code: in general, a method call has two statuses: failure and success. A method call returns 0 upon success, and nonzero otherwise. Oftentimes, we want to supply a variable to receive the return code. If we don't supply a return code variable and the method fails, an appropriate error message will be printed to the log.

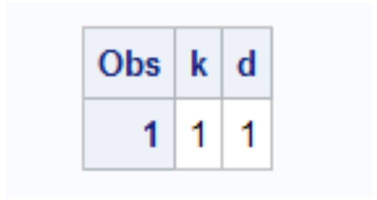
```
data _null_;
   length k $1 d 8;
   DCL HASH h();
   rc=h.DEFINEKEY('k');
   put rc=;
   rc=h.DEFINEDATA('d');
   put rc=;
   rc=h.DEFINEDONE();
   put rc=;
   call missing(k, d);
run;
```

```
rc=0
rc=0
rc=0
```

Methods of Hash Object (cont’d)

- **ADD**: adds data associated with the given key to the hash table. By default, however, if the given key already exists in the hash object, the data won’t be added.
- **REPLACE**: overwrites the data portion if the key already exists. If the key is not in the table yet, the REPLACE method inserts the key and data into the hash table.
- **OUTPUT**: outputs data portion of the hash table into a SAS dataset. Dataset options can be used. If the hash object has been pre-defined as an ordered one, the output dataset will be sorted accordingly.

```
data _null_;
  length k d 8;
  dcl hash h();
  h.definekey('k');
  h.definedata('k','d');
  h.definedone();
  k=1; d=1; rc=h.ADD();
  k=1; d=2; rc=h.ADD();
  h.OUTPUT(DATASET:'test');
run;
```

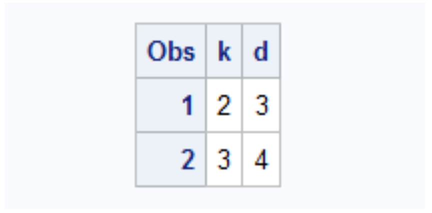


Obs	k	d
1	1	1

Methods of Hash Object (cont’d)

- **ADD**: adds data associated with the given key to the hash table. By default, however, if the given key already exists in the hash object, the data won’t be added.
- **REPLACE**: overwrites the data portion if the key already exists. If the key is not in the table yet, the REPLACE method inserts the key and data into the hash table.
- **OUTPUT**: outputs data portion of the hash table into a SAS dataset. Dataset options can be used. If the hash object has been pre-defined as an ordered one, the output dataset will be sorted accordingly.

```
data _null_;
  length k d 8;
  dcl hash h();
  h.definekey('k');
  h.definedata('k','d');
  h.definedone();
  k=2; d=2; rc=h.ADD();
  k=2; d=3; rc=h.REPLACE();
  k=3; d=4; rc=h.REPLACE();
  h.OUTPUT(DATASET:'test');
run;
```



Obs	k	d
1	2	3
2	3	4

Methods of Hash Object (cont'd)

- **CHECK**: checks existence of the specified key in the hash object. Do not overwrite PDV variables.
- **FIND**: determines whether the specified key is stored in the hash object and if found, updates the data variables in the PDV.
- **CLEAR**: removes all items from the hash table without deleting the instance of Hash Object
- **DELETE**: deletes the hash object

```
data _null_;
  dcl hash h();
  h.definekey('k');
  h.definedata('k','d');
  h.definedone();
  k=2; d=4; rc=h.add();
  call missing(k, d);
  k=2; rc=h.CHECK(); put (k d rc) (=);
  k=2; rc=h.FIND();  put (k d rc) (=);
  k=3; rc=h.CHECK(); put (k d rc) (=);
  k=3; rc=h.FIND();  put (k d rc) (=);
run;
```

```
k=2 d=.  rc=0
k=2 d=4 rc=0
k=3 d=4 rc=160038
k=3 d=4 rc=160038
```

Methods of Hash Object (cont’d)

- **EQUALS**: Determines whether two hash objects are equal.
- **REF**: Consolidates the CHECK and ADD methods into a single method call.
- **REMOVE**: Removes the data that is associated with the specified key from the hash object. The REMOVE method deletes both the key and the data from the hash object.
- **SUM**: Retrieves the summary value for a given key from the hash table and stores the value in a DATA step variable.

Non-unique key:data pairs?

- The examples we have seen so far involve only unique keys, i.e. one data value per key value. For example, ADD and REPLACE methods do not allow duplicate keys.
- What if we need a hash table with duplicate keys, i.e. multiple data items per unique key value?

Non-unique key:data pairs?

- The hash object constructor argument tag **MULTIDATA: “YES”,** first introduced with SAS v9.2, significantly expands the usage of SAS hash object, e.g. made one-to-many and many-to-many merges much easier. (Dorfman, 2016)

```
data adsl;
  if _n_=1 then do;
    if 0 then set sdtm.cm(keep=cmtrt cmstdtc cmdose);
    dcl hash h(dataset: "sdtm.cm(where=(cmoccur='Y'))", MULTIDATA: "y");
    h.definekey('usubjid');
    h.definedata('cmtrt','cmstdtc','cmdose');
    h.definedone();
  end;
  set dm_plus;
  <operations to perform certain calculations, e.g. cumulative dose of a rescue mediation during a certain time frame>
run;
```

Dorfman, Paul M.. (2016). Using the SAS® Hash Object with Duplicate Key Entries.

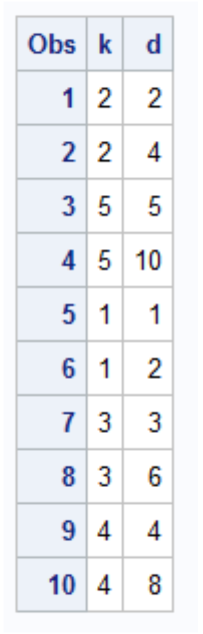
Methods of Hash Object (cont’d)

- **REPLACEDUP**: Replaces the data associated with the specified key’s current data with new data
- **REPLACE**: replaces all data values associated with the specified key (SAS v9.4)
- **REMOVEDUP**: removes the data associated with the specified key's current data item
- **REMOVE**: removes all data items for the specified key

Methods of Hash Object (cont’d)

- **FIND_NEXT**: determines if there’s a next item in the multiitem list of the current key, and if so, retrieves the data associated with it.
- **HAS_NEXT**: determines whether there is a next item in the current key's multiple data item list.
- **FIND_PREV**: determines if there’s a previous item for the current key, and if so, update the PDV with its data values.
- **HAS_PREV**: determines whether there is a previous item in the current key's multiple data item list.

```
data _null_;
  dcl hash h(MULTIDATA:'y’);
  h.definekey('k’);
  h.definedata('k','d’);
  h.definedone();
  do k=1 to 5;
    do i=1 to 2;
      d=i*k;
      h.add();
    end;
  end;
  h.output(dataset: 'test');
run;
```



Obs	k	d
1	2	2
2	2	4
3	5	5
4	5	10
5	1	1
6	1	2
7	3	3
8	3	6
9	4	4
10	4	8

Methods of Hash Object (cont’d)

- **FIND_NEXT**: determines if there’s a next item in the multiitem list of the current key, and if so, retrieves the data associated with it.
- **HAS_NEXT**: determines whether there is a next item in the current key's multiple data item list.
- **FIND_PREV**: determines if there’s a previous item for the current key, and if so, update the PDV with its data values.
- **HAS_PREV**: determines whether there is a previous item in the current key's multiple data item list.

```
data _null_;
  if 0 then set test(keep=k d);
  dcl hash h(dataset:'test', multidata:'y');
  h.definekey('k’);
  h.definedata('k','d’);
  h.definedone();
  call missing(k, d);
  /* print out records with k=1, 2, 3 */
  do k=1 to 3;
    do rc=h.find() by 0 while(rc=0);
      put k= d=;
      rc=h.FIND_NEXT();
    end;
  end;
run;
```

```
k=1 d=1
k=1 d=2
k=2 d=2
k=2 d=4
k=3 d=3
k=3 d=6
```

Methods of Hash Object (cont’d)

- **DO_OVER**: traverses a list of duplicate keys in the hash object, a more concise way. (SAS V9.4)
- **RESET_DUP**: resets the pointer to the beginning of a duplicate list of keys when you use the DO_OVER method. (SAS V9.4)
- **SUMDUP**: Retrieves the summary value for the current data item of the current key and stores the value in a DATA step variable.

```
data _null_;
  if 0 then set test(keep=k d);
  dcl hash h(dataset:'test', multidata:'y');
  h.definekey('k’);
  h.definedata('k','d’);
  h.definedone();
  call missing(k, d);
  /* print out records with k=1, 2, 3 */
  h.RESET_DUP();
  do k=1 to 3;
    do while(h.DO_OVER() eq 0);
      put k=  d=;
    end;
  end;
run;
```

Attributes of Hash Object

Attribute	Syntax	Description
ITEM_SIZE	*variable_name=object*.NUM_ITEMS;	Returns the size (in bytes) of an item in a hash object.
NUM_ITEMS	*variable_name=object.*ITEM_SIZE;	Returns the number of items in the hash object.

Note: The product of ITEM_SIZE and NUM_ITEMS provides a good approximation of memory usage by the hash object.

Hash Iterator Object

What is a hash iterator object? How to make use of it?

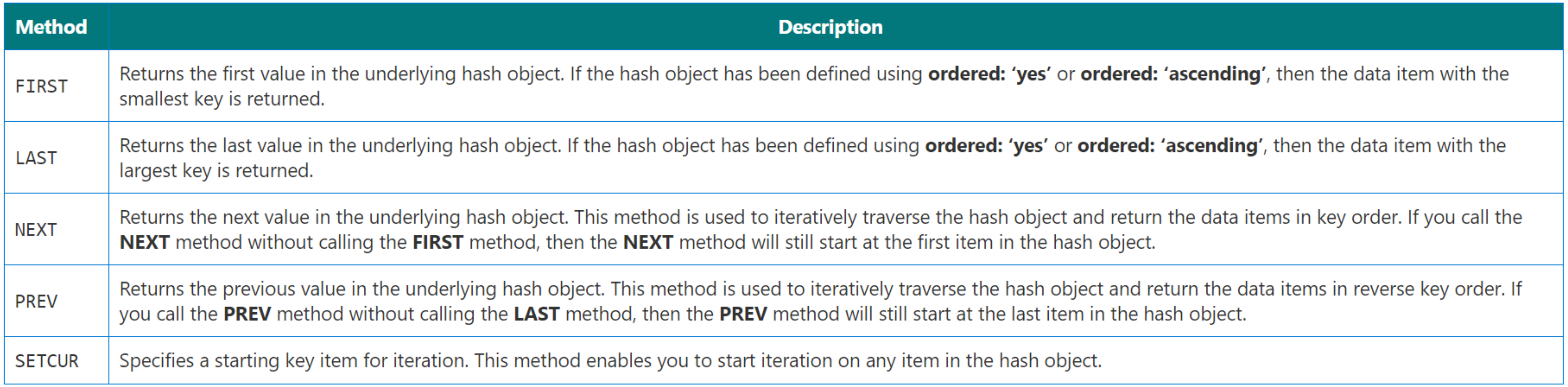
Hash Iterator Object

- Hash Iterator object is another component object made available through the DSCI. It provides a way to sequentially go through all (or part of) the items in a hash object in a **keyless** manner, in forward or reverse order.
- Declaration and instantiation of hash iterator object are similar to hash object (they are of the same type):
    - One-step approach:

      ```
      DECLARE HITER myHiter('myhash');
      ```
    - Two-step approach:

      ```
      DECLARE HITER myHiter;
      myhiter = _NEW_ HITER('myHash');
      ```

Methods of Hash Iterator Object



Method	Description
FIRST	Returns the first value in the underlying hash object. If the hash object has been defined using **ordered: 'yes'** or **ordered: 'ascending'**, then the data item with the smallest key is returned.
LAST	Returns the last value in the underlying hash object. If the hash object has been defined using **ordered: 'yes'** or **ordered: 'ascending'**, then the data item with the largest key is returned.
NEXT	Returns the next value in the underlying hash object. This method is used to iteratively traverse the hash object and return the data items in key order. If you call the **NEXT** method without calling the **FIRST** method, then the **NEXT** method will still start at the first item in the hash object.
PREV	Returns the previous value in the underlying hash object. This method is used to iteratively traverse the hash object and return the data items in reverse key order. If you call the **PREV** method without calling the **LAST** method, then the **PREV** method will still start at the last item in the hash object.
SETCUR	Specifies a starting key item for iteration. This method enables you to start iteration on any item in the hash object.

Methods of Hash Iterator Object (cont'd)

```
data _null_;
  dcl hash h(multidata: 'y', ORDERED: 'a');
  dcl hiter hi('h');
  h.definekey('k');
  h.definedata('k','d');
  h.definedone();

  do k=3,2,2,1,3;
    d=byte(k+64);
    h.add();
  end;

  put 'Data items in key order: ';
  do while (hi.NEXT() eq 0);
    put k= d=;
  end;

  put 'Data items in reverse key order: ';
  do rc=hi.LAST() by 0 while (rc=0);
    put k= d=;
    rc=hi.PREV();
  end;
run;
```

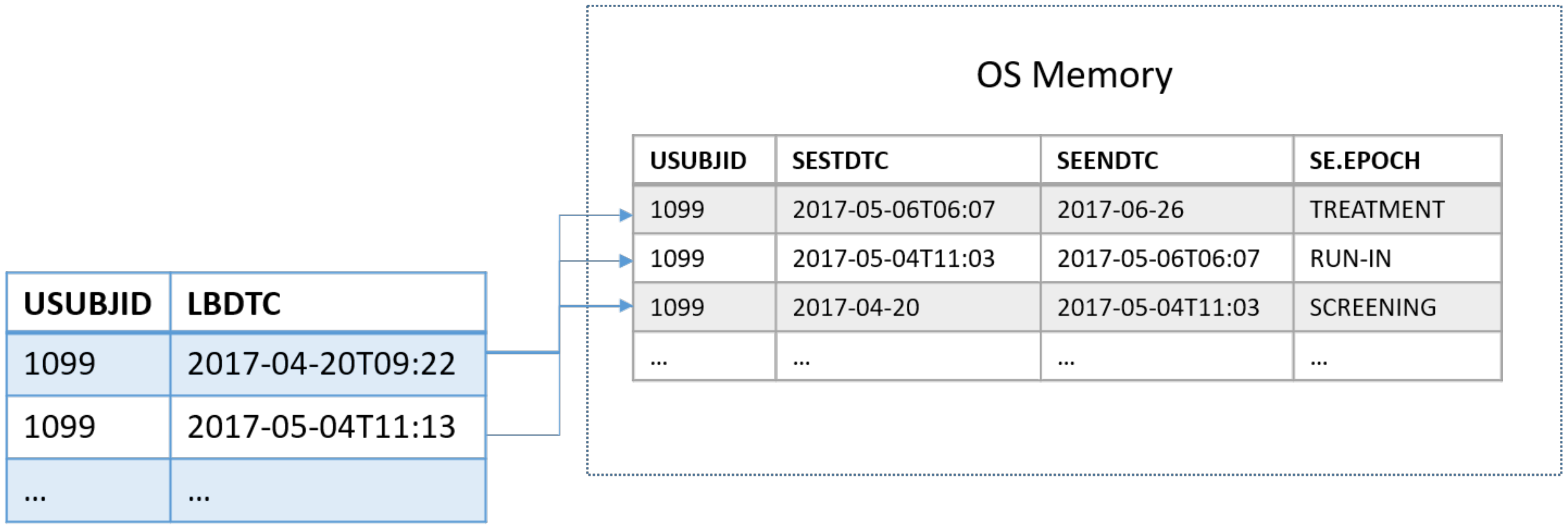
```
Data items in key order:
k=1 d=A
k=2 d=B
k=2 d=B
k=3 d=C
k=3 d=C
Data items in reverse key order:
k=3 d=C
k=3 d=C
k=2 d=B
k=2 d=B
k=1 d=A
```

Potential Applications in Clinical Trials Programming

How may we use hash objects in the clinical trials programming environment?

CASE 1: table lookup

- Problem:
  - ✓ Suppose we have the following derivation rule for SDTM.LB.EPOCH: Set to SDTM.SE.EPOCH on the subject element which satisfies SE.SESTDTC <= LB.LBDTC <= SE.SEENDTC.



USUBJID	LBDTC
1099	2017-04-20T09:22
1099	2017-05-04T11:13
...	...

OS Memory

USUBJID	SESTDTC	SEENDTC	SE.EPOCH
1099	2017-05-06T06:07	2017-06-26	TREATMENT
1099	2017-05-04T11:03	2017-05-06T06:07	RUN-IN
1099	2017-04-20	2017-05-04T11:03	SCREENING
...	...	...	...

CASE 1: Sample code

```
data lb2;
  if _n_=1 then do;
    if 0 then set sdtm.se(keep=sestdtc seendtc epoch rename=(epoch=_epoch));
    dcl hash h (dataset:"sdtm.se", multidata:'y');
    h.definekey("usubjid") ;
    h.definedata("sestdtc", "seendtc", "_epoch");
    h.definedone();
  end;
  length EPOCH $200;
  set lb;

  call missing(epoch, _epoch);
  if not missing(lbdtc) then do rc=h.find() by 0 while (rc=0);
    if sestdtc <= lbdtc <= seendtc then do;
      epoch=_epoch;
      leave;
    end;
    rc=h.find_next();
  end;
run;
```

- Load SDTM.SE into a hash table as a lookup table.
- For each record in the LB dataset, loop through all epochs associated with the subject, and break out of the loop when the epoch meeting the solicited criteria is found.

CASE 2: Hash of Hashes

- Problem:
  - ✓ Suppose we want to split COMMENT into COVAL1-10 at a max length of 200 by word, i.e. without breaking up individual words.

CASE 2: Sample code

```
data co2;
  if _n_=1 then do;
    /* hash table to hold all words of COVALn */
    dcl hash h;
    dcl hiter hi;
    h = _new_ hash(ordered: 'y');
    h.definekey('count');
    h.definedata('word');
    h.definedone();

    /* hash table to hold all COVALn, each of which is a hash object itself */
    dcl hash x(ordered: 'y');
    x.definekey('nvars’);
    x.definedata('h’);
    x.definedone();
  end;
  set co;
  length tempStr word $400;

  /* clear both objects to get ready for processing a new observation */
  x.clear();
  h.clear();
  count=1; * order of words;
  nvars=1; %* count of hash object h’s;
  call missing(tempStr);
```

CASE 2: Sample code (cont'd)

```
do while(scan(comment,count," ") ne ''); /* split long strings */
    word=scan(comment,count," ");
    tempStr=catx(" ",tempStr,word);
    if length(tempStr)<=200 then h.add();
    else do;
      x.add();
      /* create a new hash object h to if the previous one cannot acceptable 
any more word */
      h = _new_ hash(ordered: 'y');
      h.definekey('count');  h.definedata('word');  h.definedone();
      h.add();
      nvars+1; tempStr=word;
    end;
    count+1;
  end;
  if h.num_items gt 0 then x.add(); * any left-over;
```

```
  /* read hash of hashes and concatenate words within each hash object h */
  array acovals{10} $200 coval1-coval10;
  do i=1 to x.num_items;
    x.find(key:i);
    hi = _new_ hiter('h');
    do while (hi.next()=0);
      acovals[i]=catx(" ",acovals[i],word);
    end;
  end;
run;
```

Step 1: Split strings by word:

- each hash object H holds as many words as possible within 200-char limit;
- Hash object X holds as H's.

Step 2: Concatenate the words to form COVALx:

- Loop thru X and its H's, and return COVAL
- Each H corresponds to one COVAL
- Each COVAL is within 200-char limit.

CASE 3: data aggregation

- Problem:
  - ✓ Suppose we have the following definition for VSBLFL: set to “Y” for the last non-missing result prior to dosing

CASE 3: Sample code

```
data vs2;
  length base_vsdtc $19;
  if _n_=1 then do;
    dcl hash b();
    b.definekey('usubjid','vstestcd');
    b.definedata('usubjid','vstestcd','base_vsdtc');
    b.definedone();

    do until(eof);
      set vs end=eof;
      call missing(base_vsdtc);
      rc=b.find();
      if base_vsdtc < vsdtc < rfxstdtc and not missing(vsorres) then do;
        base_vsdtc=vsdtc;
        b.replace();
      end;
    end;
    b.output(dataset: 'baseline(rename=base_vsdtc=vsdtc)');

    dcl hash x(dataset:'baseline');
    x.definekey('usubjid','vstestcd','vsdtc');
    x.definedone();
  end;
  call missing(of _all_);
  set vs;
  if x.find()=0 then vsblfl='Y';
run;
```

- Create an empty hash table for holding baseline records
- Step thru input data set and accumulate baseline records, and save them to the previously created hash table.
- Load the baseline data set into a new hash table for merging back.

CASE 4: Array sorting

- Problem:
  - ✓ Suppose we are asked to create a race code variable with the following assignment, and concatenate them in an alphabetical order if multiple races:
    - White -> W
    - Black or African American -> B
    - Asian -> A
    - Native Hawaiian or Other Pacific Islander -> P
    - American Indian or Alaska Native -> N

CASE 4: Sample code

```
data adsl;
  if _n_=1 then dcl hash h;
  set dm_plus;
  length racecd $5 _race $1;
  array arace{*} race1-race3;
  if race ne 'MULTIPLE' then racecd=put(race,$racecd.);
  else do;
    h = _new_ hash(ordered: 'y');
    dcl hiter hi ('h');
    h.definekey('_race');
    h.definedone();

    do i=1 to dim(arace);
      _race=put(arace[i],$racecd.);
      if not missing(_race) then h.add();
    end;
    /* put back in alphabetical order */
    do while(hi.next() eq 0);
      racecd=catx('/', racecd, _race);
    end;
  end;
  drop i _race;
run;
```

- Create an ordered hash, with the key (and data) being the individual race code.
- Populate the hash table with race code(s)
- Write out – already sorted alphabetically

Thank You!