Learning outcomes:

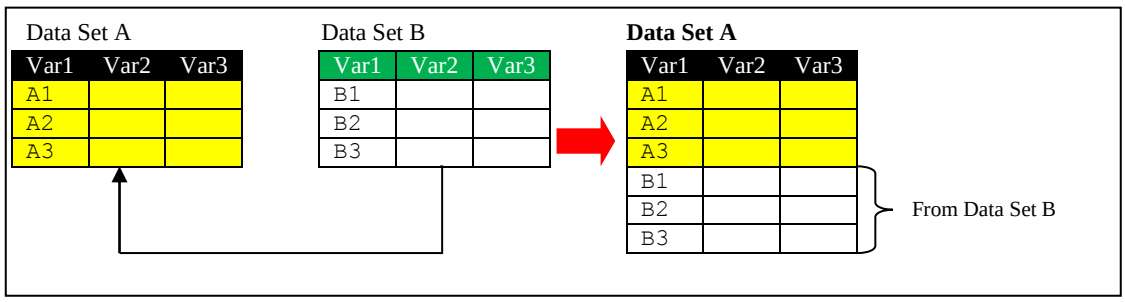
1. Appending a data set
2. Concatenating data sets
3. Interleaving data sets
4. Sorting a data set
5. Match merging
6. Introduction to SQL

SAS components learnt:

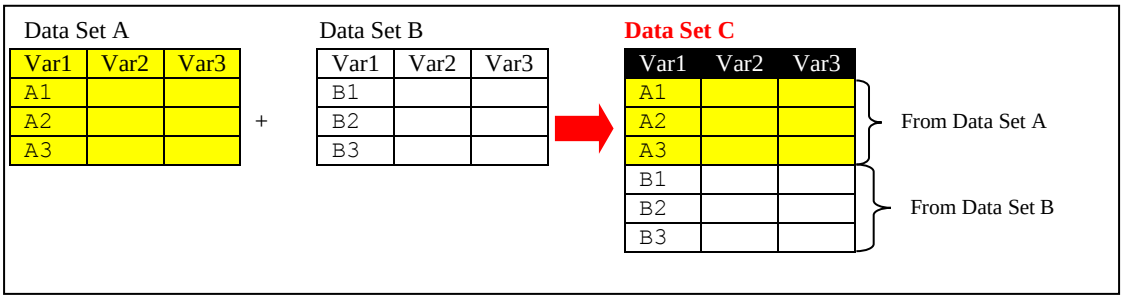
1. PROC APPEND
2. PROC SORT
3. RENAME= data set option for variables
4. SET statement with multiple data sets
5. MERGE statement together with BY statement for match merge
6. IN= data set option for variables

6.1 Techniques of combining data sets

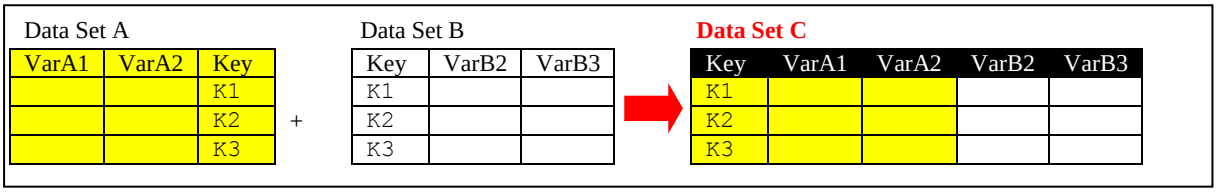
- Appending a data set



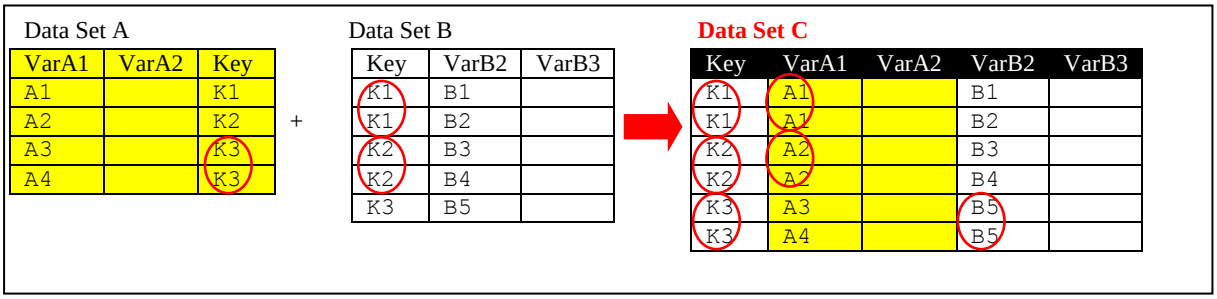
- Concatenating data sets



- Merging data sets using **match merging**
  - One to one match merging



  - One to many or many to one merging



- Nonmatches merging

Data Set A

VarA1	VarA2	Key
A1	…	K1
A2	…	K3
A3	…	K4
A4	…	K5

+

Data Set B

Key	VarB2	VarB3
K1	B1	…
K2	B2	…
K3	B3	…
K5	B4	…
K6	B5	…

Data Set C*

Key	VarA1	VarA2	VarB2	VarB3
K1	A1	…	B1	…
K2			B2	…
K3	A2	…	B3	…
K4	A3	…		
K5	A4	…	B4	…
K6			B5	…

* blank are missing.

- SAS issues a note to the log: "NOTE: MERGE statement has more than one data set with repeats of BY values."
- SAS does not produce Cartesian product for many-to-many merge.

6.2 Appending a Data Set

- Procedure APPEND
  - Syntax

```
PROC APPEND BASE=Base-SAS-data-set DATA=SAS-data-set <FORCE>;
RUN;
```

Append one "*SAS-data-set"* to another SAS data set, "*Base-SAS-data-set*". FORCE option is **<u>required</u>** for unlike-structure data sets appending.

  - Requirements:
    - Only two data sets can be used at a time in one step
    - The observations in the base data set are not read.
    - The variable information in the descriptor portion of the base data set cannot change. i.e. the descriptor portion of the base data set and the data set to be appended is the same. (How to list the descriptor portion information? Using Procedure Contents)
  - Like structure data set

AMA.Grad

Name	Gender	GradYr
John	M	10
Peter	M	10
Kitty	F	10

AMA.Grad11

Name	Gender	GradYr
Ralph	M	11
Karl	M	11
Ken	M	11

AMA.Grad

Name	Gender	GradYr
John	M	10
Peter	M	10
Kitty	F	10
Ralph	M	11
Karl	M	11
Ken	M	11

    - To append AMA.Grad11 to the AMA.Grad:

```
libname ama 'd:\SAS_datasets';
Proc append base=ama.grad data=ama.grad11;
run;
```

  - Unlike structure data set
    - The AMA.Grad11 data set is modified into AMA.Grad11m to include Phone number. Then the structure of AMA.Grad and AMA.Grad11m is unlike.

AMA.Grad11m

Name	Gender	Phone	GradYr
Ralph	M	…	11
Karl	M	…	11
Ken	M	…	11

    - We may propose the SAS program:

```
Proc append base=ama.grad data=ama.grad11m;
run;
```

(a) **Without FORCE** option, **no** observation is appended! A warning is written to the log window. At the same time an error statement is written:

```
ERROR: No appending done because of anomalies listed above. Use FORCE
option to append these files.
```

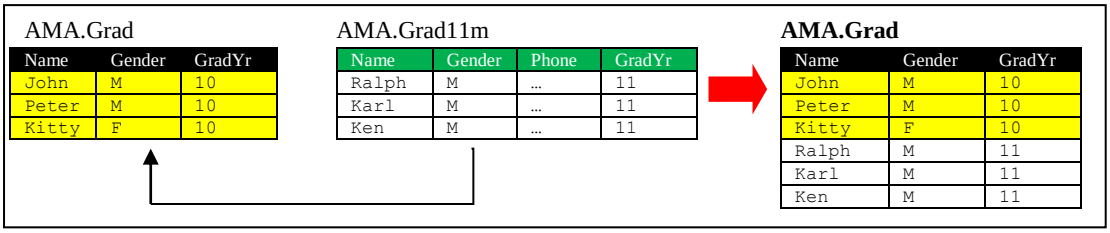
(b) FORCE option is added:

```
Proc append base=ama.grad data=ama.grad11m force;
run;
```

See the warning in the log window:

```
377  Proc append base=ama.grad data=ama.grad11m force;
378  run;

NOTE: Appending AMA.GRAD11M to AMA.GRAD.
WARNING: Variable phone was not found on BASE file. The variable will not
         be added to the BASE file.
NOTE: FORCE is specified, so dropping/truncating will occur.
NOTE: There were 6 observations read from the data set AMA.GRAD11M.
NOTE: 6 observations added.
NOTE: The data set AMA.GRAD has 9 observations and 3 variables.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

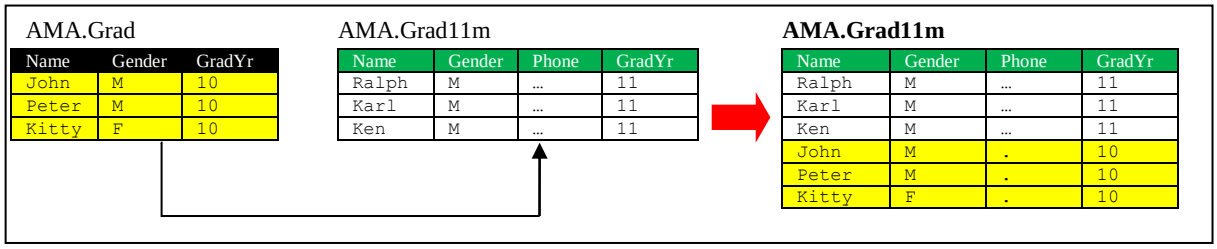
The following diagram summarises the SAS action of the above program. Note that a warning will be given:



(c) If we append AMA.Grad to AMA.Grad11m:

```
Proc append base=ama.grad11m data=ama.grad force;
run;
```

The following diagram summarises the SAS action of the above program. Periods (missing value) are put into the phone variable of records from AMA.GradNote that a warning will be given:



A warning is given in the log window:

```
343  Proc append base=ama.grad11m data=ama.grad force;
344  run;

NOTE: Appending AMA.GRAD to AMA.GRAD11M.
WARNING: Variable phone was not found on DATA file.
NOTE: There were 3 observations read from the data set AMA.GRAD.
NOTE: 3 observations added.
NOTE: The data set AMA.GRAD11M has 6 observations and 4 variables.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

- We summarise the appending of unlike-structured data sets into two cases:
  (a) BASE= data set contains a variable which is not in the DATA= data set: The observations are appended, but the observations from the DATA= data set have a missing value for the variable which was not present in the DATA= data set. The FORCE option is not necessary in this case.
  (b) DATA = data set contains a variable which is not in the BASE= data set: Use the FORCE option in the PROC APPEND statement to force the concatenation of the two data sets. The statement drops the extra variable and issues a warning message.

Ex. 6.1: (a) Use DATA step without INFILE /SET statement(s) to create AMA.Grad.

(b) Submit the following SAS statements.

```
Data AMA.grad11;
Length Name $10;
Name='Ralph';
Gender='M';
GradYr=11;
output;
Name='Karl';
Gender='M';
GradYr=11;
output;
Name='Ken';
Gender='M';
GradYr=11;
run;
```

Will it give the same output shown in the AMA.grad11 data set shown under "Like-structured data sets"? If no, what modification is required?

(c) (i) Submit the following program.

```
/*p6ex1.sas */
Proc append base=ama.grad data=ama.grad11;
run;
```

What is found in the output window?
Is there any message found in the log window?

(ii) Whether a LENGTH statement is required in the data set creation so that the one can be appended to another?

6.3 Concatenating Data Sets

- Syntax

```
DATA Concatenated-Data-Set;
  SET SAS-data-set-1 SAS-data-set-2… SAS-data-set-n;
  <additional SAS statements>
RUN;
```

  - The observations of the first data set appear first in the concatenated data set and then the observations of the second data set appear after the first data set. SAS put all observations of the third data set after the second one in the concatenated data set. The following diagram represents SAS action in concatenation:

    Concatenated-Data-Set

    Observations from SAS-data-set-1
    Observations from SAS-data-set-2
    .....
    Observations from SAS-data-set-n

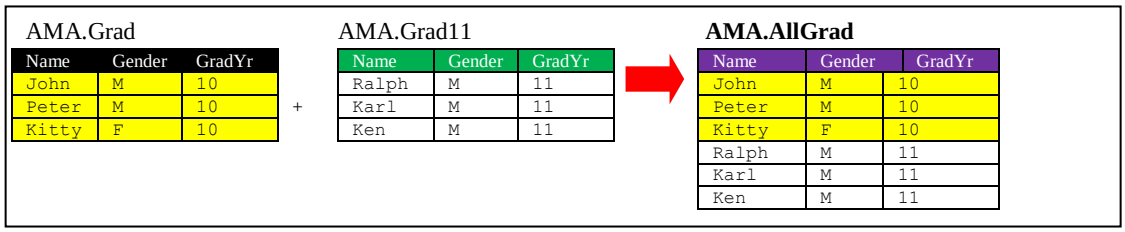
  - Compile-time error occurs if the same variable is not the same type in all SAS data sets in the SET statement.

- Graduate data revisit (Like-structured data sets)
  - We combine the two data sets, AMA.Grad and AMA.Grad11 into a new data set called AMA.AllGrad:

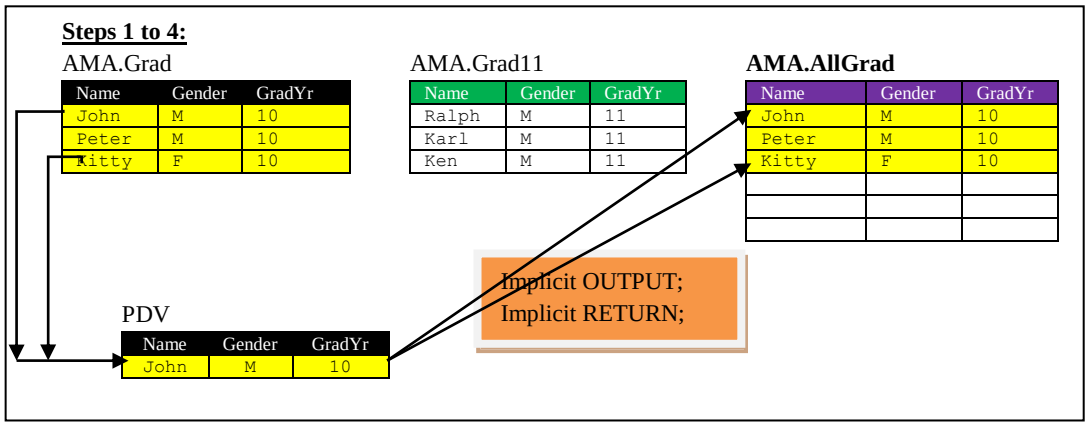
```
data ama.allgrad;
set ama.grad ama.grad11;
run;
```

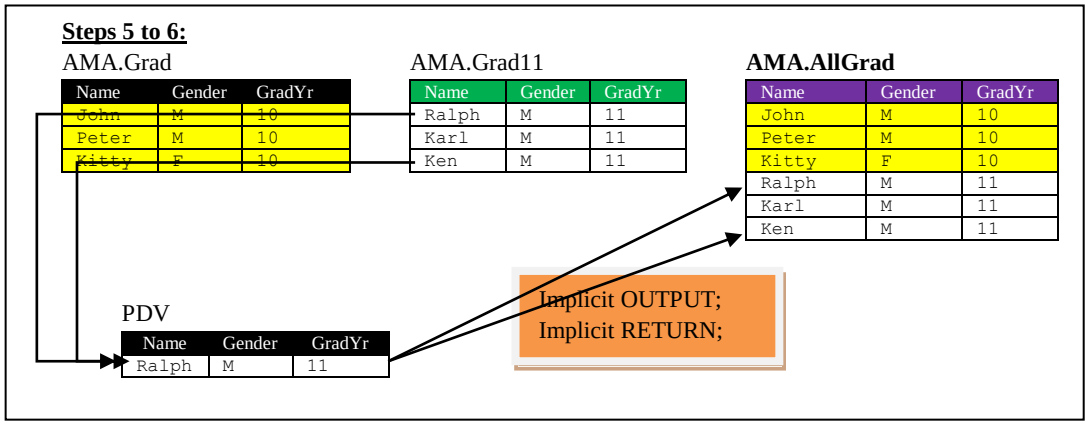
Implicit OUTPUT;
Implicit RETURN;

- The following diagram describes the final data set for concatenating data sets.



- How SAS process the data sets? We use PDV for the detail description:
  - Compilation stage: read the structure of the all data sets input (i.e . AMA.Grad, AMA.Grad11) in the PDV (i.e. AMA.AllGrad). For those variables repeated in the 2nd or other data sets, it will not be repeated in the PDV.
  - Execution stage:
    (a) Step 1: Initialise the PDV with structure same as the data set of AMA.Grad (i.e. SAS-data-set-1)
    (b) Step 2: Read the first observation of AMA.Grad (the 1st data set) into PDV.
    (c) Step 3: Process the additional SAS statement till OUTPUT statement or RUN statement. Output the PDV content into the output SAS data set when RUN statement is encountered.
    (d) Step 4: Check whether EOF (end of file) of AMA.Grad. If not, repeat the Step 2 and Step 3 to the next observation of AMA.Gard till EOF.
    (e) Step 5: Read the first observation of AMA.Grad11 (the 2nd data set) into PDV.
    (f) Step 6: Repeat Step 4 onto the data set AMA.Grad11 till EOF.





- Graduate data revisit (Unlike-structured data sets)
  - The data set, AMA.Grad11 is modified so that "GYr" represents "GradYr".
  - Combine the data sets, AMA.Grad and AMA.Grad11a into AMA.AllGrada:

AMA.Grad

Name	Gender	GradYr
John	M	10
Peter	M	10
Kitty	F	10

+

AMA.Grad11a

Name	Gender	GYr
Ralph	M	11
Karl	M	11
Ken	M	11

AMA.AllGrada

Name	Gender	GradYr	GYr
John	M	10	.
Peter	M	10	.
Kitty	F	10	.
Ralph	M		11
Karl	M		11
Ken	M		11

  - How SAS works on the data sets?
    - Compilation stage: read the structure of the all data sets input (i.e . AMA.Grad, AMA.Grad11a) in the PDV (i.e. AMA.AllGrad). For those variables repeated in the 2nd or other data sets, it will not be repeated in the PDV.
    - Execution stage: Same as process for the like-structured data sets.

Compilation:

AMA.Grad

Name	Gender	GradYr
John	M	10
Peter	M	10
Kitty	F	10

AMA.Grad11a

Name	Gender	GYr
Ralph	M	11
Karl	M	11
Ken	M	11

PDV

Name	Gender	GradYr	GYr

  - Use RENAME= option in a data set
    - The variable 'GYr' is in fact same as 'GradYr' and therefore we don't need 'GYr' in the concatenated data set.
    - Syntax of RENAME option

```
SAS-data-set(RENAME=(old-var-name-1 = new-var-name-1
                     old-var-name-2 = new-var-name-2
                     …
                     old-var-name-n = new-var-name-n))
```

      (a) RENAME option change the variable name of the *SAS-Data-Set* before the parentheses.
      (b) The *new-var-name* will be applied to the data set immediately in the compilation stage before making PDV.
    - The SAS program is modified to create same output as the AMA.AllGrad:

```
data ama.allgrad;
set ama.grad ama.grad11a(RENAME=(GYr=GradYr));
run;
```

Compilation:

AMA.Grad

Name	Gender	GradYr
John	M	10
Peter	M	10
Kitty	F	10

AMA.Grad11a

Name	Gender	GYr
Ralph	M	11
Karl	M	11
Ken	M	11

GradYr

PDV

Name	Gender	GradYr

- Interleaving
  - Interleaving intersperses observations from two or more data sets, based on the BY-variables.
  - The group of statements for interleaving is similar to that of concatenating data sets. The only difference is BY statement
  - Syntax

```
DATA Concatenated-Data-Set;
  SET SAS-data-set-1 SAS-data-set-2… SAS-data-set-n;
  BY <DESCENDING> by-variable(s);
  <additional SAS statements>
RUN;
```

  - Typically, it is more efficient to sort small SAS data sets and then interleave them as opposed to concatenating several SAS data sets and then sorting the resultant larger file.
  - AMA graduate data revisit with PDV process in details

```
data ama.allgrad;  *p6_concatenation.sas;
  set ama.grad ama.grad11a(RENAME=(GYr=GradYr));
  BY Name;
run;
```

Compilation:

AMA.Grad (Sorted)

Name	Gender	GradYr
John	M	10
Kitty	F	10
Peter	M	10

```
proc sort data=ama.grad;
by name;
run;
```

AMA.Grad11a (Sorted)

Name	Gender	GYr
Karl	M	11
Ken	M	11
Ralph	M	11

```
proc sort data=ama.grad11a;
by name;
run;
```

GradYr

PDV

Name	Gender	GradYr

Execution of first three steps: There is a pointer to point at the record being processed to each data set. Comparison will be made between the records from the two input files.When the record is chosen to copy to PDV, the pointer then moves to the next record. And the PDV is then copied to the output data set. This process will continue till end of file (EOF) of the two files.

Step1: Which Name comes first, John or Karl? John

AMA.Grad (Sorted)

Name	Gender	GradYr
John	M	10
Kitty	F	10
Peter	M	10

AMA.Grad11a (Sorted)

Name	Gender	GYr
Karl	M	11
Ken	M	11
Ralph	M	11

PDV

Name	Gender	GradYr
John	M	10

--→ Output record of John

Step2: Which Name comes first, Kitty or Karl? Karl

AMA.Grad (Sorted)

Name	Gender	GradYr
John	M	10
Kitty	F	10
Peter	M	10

AMA.Grad11a (Sorted)

Name	Gender	GYr
Karl	M	11
Ken	M	11
Ralph	M	11

PDV

Name	Gender	GradYr
Karl	M	11

--→ Output record of Karl

Step3: Which Name comes first, Kitty or Ken? Ken

AMA.Grad (Sorted)

Name	Gender	GradYr
John	M	10
Kitty	F	10
Peter	M	10

AMA.Grad11a (Sorted)

Name	Gender	GYr
Karl	M	11
Ken	M	11
Ralph	M	11

PDV

Name	Gender	GradYr
Ken	M	11

--→ Output record of Ken

The final output:

AMA.AllGrad

Name	Gender	GradYr
John	M	10
Karl	M	11
Ken	M	11
Kitty	F	10
Peter	M	10
Ralph	M	11

Ex. 6.2: Given a data set, AMA.Banquet_List containing the host name in BANQUET_HOST, no. of guest in NO_OF_GUEST, total spending for the banquet, TOTAL_SPENDING and gift quantity as QUANTITY, supplier name in SUPPLIER and an excel file, BonusGift.xls containing a list of suppliers on the sheet, 'Supplier' with SuppID = supplier ID, Gift = gift name and Quantity = minimum quantity for each purchase.

AMA.Banquet_List

Banquet_Host
No_of_Guest
Total_spending
Quantity
Supplier

BonusGift.xlsx

SuppID
Gift
Quantity

Write a SAS program to create a temporary SAS data set 'Banquet_list' as a copy of 'AMA.Banquet_List' and a temporary data set called 'supplier1' and rename the variables: 'SuppID' as 'Supplier' and 'Quantity' as 'MinQuantity'. Use 'Ggift' as the library reference for the BonusGift.xlsx.

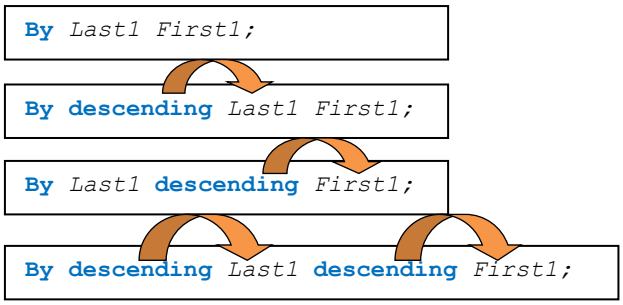
6.4 Sorting a Data Set

- Procedure SORT
  - Syntax

```
PROC SORT DATA=input-SAS-data-set <OUT=sorted-SAS-data-set NODUPKEY
EQUALS>;
  BY <DESCENDING> By-variable-1 … By-variable-n;
RUN;
```

- NODUPKEY option deletes observation with duplicate BY values. If duplicates exist, SAS keeps the first observation and delete the remaining having the same BY values.
- EQUALS option maintains the relative order of the observations within the input data set in the output data set for observations with identical BY values.
- Rearrange the observations by the values in ascending order, by default.

- The DESCENDING option reverses the sort order for the variable which immediately follows in the statement so that observations are sorted from the largest value to the smallest value. For example,



The orange arrows indicate the variable(s) which has\have descending ordering in the sorted data set.

- Sort the observations on *By-variable-1*, and then *By-variable-2*, … ,*By-variable-n* and output the observations in a sorted order.
- If no output data set is provided (i.e. OUT= option is not used), the sorted data replaces the input data by default. Otherwise, the input-SAS-data-set is kept unchanged and the sorted observations are output to a new file with name supplied by user.

- AMA Year 3 students data revisit
  - To sort the data set according to the order of university no. (Uni_no).
    (a) SAS program:

```
* p6example_sort.sas;
data visit;
  length Last_Name $20 First_Name $20 Gender $1 Resid_District
$20 Home_country $20 Project_Field $20 Visit_country $20;
   infile 'd:\SAS_datasets\AMAYr3visit.csv' dlm=',';
   Input Uni_no First_name $ Last_name $Gender $Telephone
Resid_District  $ Home_country  $ Project_Field $ Visit_country;
run;
*output to work.sorted_vist;
proc sort data=visit out=sorted_visit;
by Uni_no;
run;
*replaces observations in work. visit;
proc sort data=visit;
by Uni_no;
run;
* print out Uni_no First_Name Last_Name Gender Project_field of
Work.visit;
proc print data=visit;
var Uni_no First_Name Last_Name Gender Project_field;
run;
```

(b) Output:

Obs	Uni_no	First_ Name	Last_ Name	Gender	Project_Field
1	2012113456	Amy	Yip	F	Statistics
2	2012118349	Rosanna	Or	F	Operation Research
3	2012145690	Ricky	Lo	M	Statistics
4	2012198049	Benjamin	Tang	M	Operation Research
5	2012403456	Alice	Ng	F	Actuarial Science

(i) Only those variables specified in the VAR statement under procedure PRINT are printed in the output window.
(ii) The Uni_no is sorted in ascending order by default.

6.5 Merging Data Sets using Match merging

- Data processing of match merging(from SAS help)
  - Compilation phase: SAS reads the descriptor information of each data set that is named in the MERGE statement and then creates a program data vector that contains all the variables from all data sets as well as variables created by the DATA step. SAS creates the FIRST.variable* and LAST.variable* for each variable that is listed in the BY statement.

*For example, data set, Work.A has a variable, KEY.

Data Set A

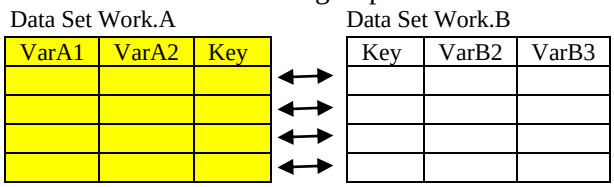
VarA1	VarA2	Key	FIRST.Key	LAST.Key
A1	…	K1	1	0
A2	…	K1	0	0
A3	…	K1	0	1
A4	…	K2	1	0
A5	…	K2	0	1

Then the first BY group has a value of ‘K1’ and the last BY group has a value of ‘K2’.

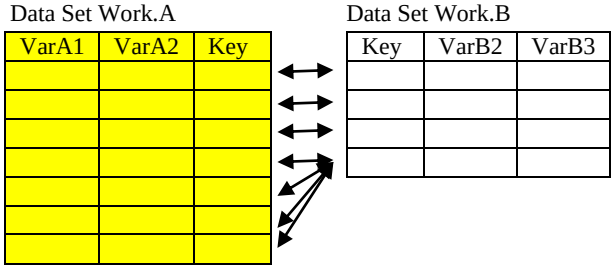
- Execution phase
  - *Step 1*: SAS looks at the first BY group in each data set which is named in the MERGE statement to determine which BY group should appear first in the new data set.
  - *Step 2*: The DATA step reads the first observation in that BY group from each data set into the program data vector (PDV). Note that the data sets are read in the order in which they appear in the MERGE statement. If a data set does not have observations in that BY group, SAS put missing values into the PDV for the variables unique to that data set.
  - *Step 3*: After processing the first observation from the last data set and executing other statements, SAS writes the contents of the PDV to the new data set. SAS retains the values of all variables in the PDV except those variables created by the DATA step. In that case, SAS puts missing values into PDV for those variables.
  - *Step 4*: SAS continues to merge observations (i.e. Steps 2 and 3) until it writes all observations from the first BY group to the new data set.

    Graphically for Steps 2 to 4:

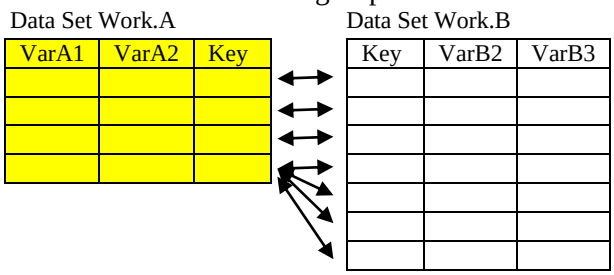
    (a) If size of the selected BY group for Work.A and Work.B are equal:



    (b) If size of the selected BY group for Work.A > that of Work.B:



    (c) If size of the selected BY group for Work.A < that of Work.B:



  - *Step 5*: When SAS has read all observations in a BY group from all data sets, it sets all variables in the PDV (except those created by SAS, e.g. _N_) to missing.
  - *Step 6*: SAS looks at the next BY group in each data set to determine which BY group should appear next in the new data set.
  - *Step 7*: SAS repeats these steps (Steps 2 to 6) until it reads all observations from all BY groups in all data sets.

- Precautions
  - The two data sets to be merged are required to be sorted by one or more common variables before the merging. Without the common variable, BY group cannot be formed. In the database languages the first two common variables are called the primary key and secondary key. Note that the keys are unique.
  - To check whether the data set is sorted or not, use procedure CONTENTS. An example:

```
proc contents data=visit;
run;
```

And the output:

```
                                The CONTENTS Procedure

      Data Set Name        WORK.VISIT                            Observations          5
      Member Type          DATA                                  Variables             9
      Engine               V9                                    Indexes               0
      … … … …
      Encoding             wlatin1  Western (Windows)


                               Engine/Host Dependent Information

Data Set Page Size          12288
Number of Data Set Pages    1
… … … …
Host Created                W32_VSHOME


                            Alphabetic List of Variables and Attributes

                                #    Variable          Type    Len

                                2    First_Name        Char     20
                                3    Gender            Char      1
                                5    Home_country      Char     20
                                1    Last_Name         Char     20
                                6    Project_Field     Char     20
                                4    Resid_District    Char     20
                                9    Telephone         Num       8
                                8    Uni_no            Num       8
                                7    Visit_country     Char     20


                                      Sort Information

                                   Sortedby       Uni_no
                                   Validated      YES
                                   Character Set  ANSI
```

  - Duplicates of the data sets are required to be removed. See procedure SORT for details.
  - Type of the merging
    - One-to-one
    - One –to-many or Many-to-one
    - Non-matches
    - Many-to-many

- Syntax

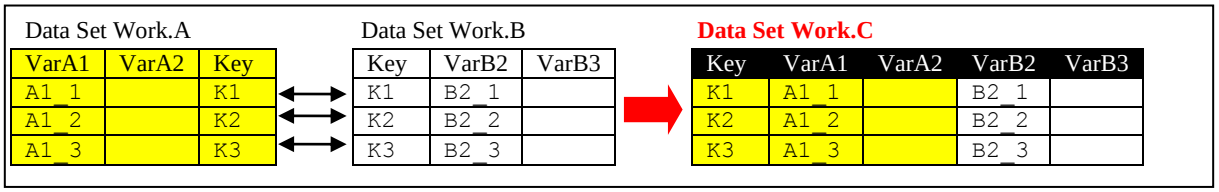
```
PROC SORT DATA= SAS-data-set-1 <NODUPKEY EQUALS>;
  BY <DESCENDING> By-variable-1 … By-variable-n;
RUN;
:
:
:
PROC SORT DATA= SAS-data-set-n <NODUPKEY EQUALS>;
  BY <DESCENDING> By-variable-1 … By-variable-n;
RUN;

DATA SAS-data-set;
  MERGE SAS-data-set-1 SAS-data-set-2 … SAS-data-set-n;
  BY <DESCENDING> By-variable-1 … By-variable-n;
  <additional SAS statements>
RUN;
```

  - The yellow highlighted portion might not be required provided that the data sets are sorted according to the BY variable(s).
  - Use of BY statement enables SAS to process data in groups. Data should arranged in either ascending or descending order of the values of the BY-variable(s). Each group has the same value(s) of BY-variable and we call the group as "BY-group" for easy referencing.
  - A BY statement after the MERGE statement performs a match-merge. The BY-variable(s) can be primary key and/or secondary key.
    - For example to merge data sets Work.A and Work.B into Work.C with the primary key variable "Key", we write the SAS program:

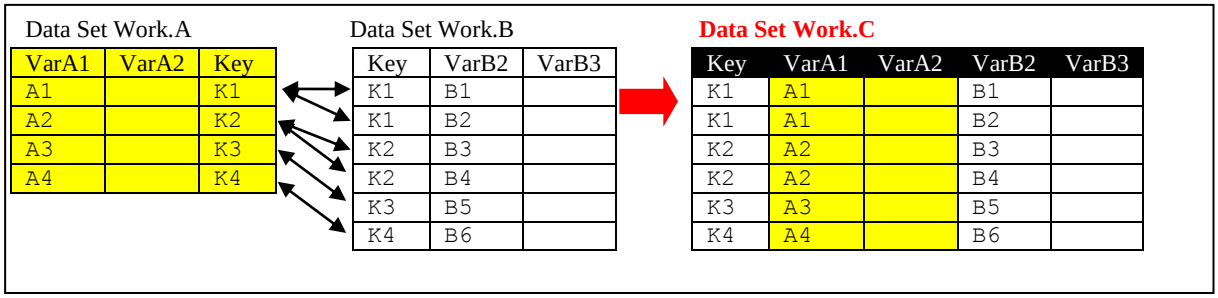
```
data work.c;
  merge work.a work.b;
  by key;
run;
```

    - The data set Work.A and Work.b should be sorted according to the BY-variable "KEY", before execution of the DATA step.

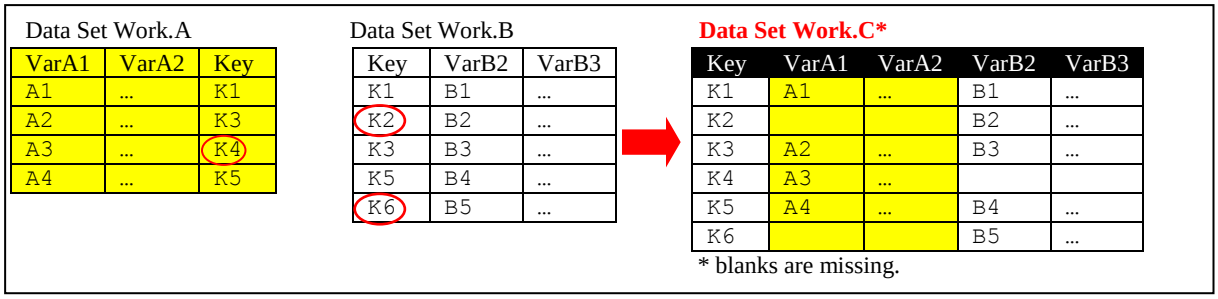
- Type of merge
  - One to one matching merge
    - A single observation in one set is related to one and only one observation from another data set based on the values of one or more selected variables.



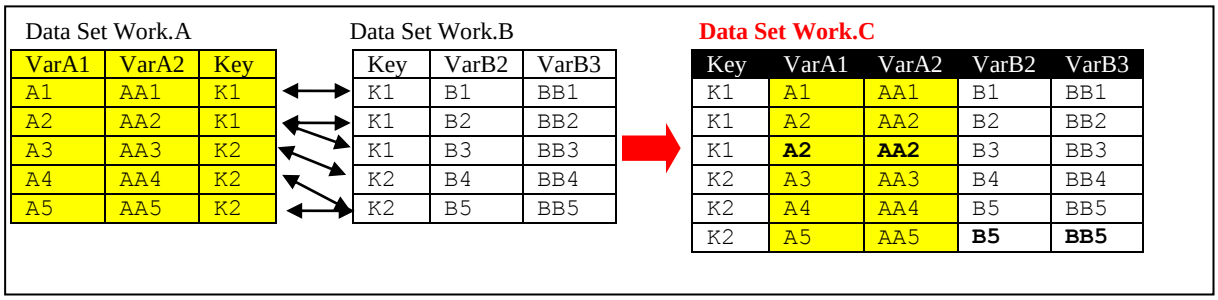
- One to many or many to one matching merge
  - A single observation in one data set is related to more than one observation from another data set based on the values of one or more selected variables and vice versa.



- Non-matching merge
  - At least one single observation in one data set is unrelated to any observation from another data set based on the values of one or more selected variables.



- Many-to-many matching merge



- Compilation and PDV processing (One-to-many merging)
  - During the compilation stage, all variables from the two data sets are used to set up the PDV in the memory. Note that the BY-variable is not repeated in the PDV.
  - During the execution
    - SAS reinitialises variables in the PDV at the start of every DATA step iteration. Variables created by an assignment statement are reset to missing, but variables which are read with a MERGE statement are **NOT** reset to **MISSING**.
    - After the PDV is initialised or reinitialized, SAS reads the values of the BY-variable from each matching data set. The values are compared. If they are the same, read the observation into PDV and output the PDV into the output data set.
    - Before reading additional observations during a match-merge, SAS first determines whether there are observations remaining for the current BY group.
      (a) If yes, the remaining observation(s) is/are read into the PDV, processed and written to the output data set.
      (b) If no, SAS reinitialises the remainder of the PDV, identifies the next BY-group and reads the corresponding observations.

- Compilation and PDV processing (Nonmatches)
  - The output data set results in the diagram can be obtained using the flowchart.
  - The IN option to data set
    - Syntax

```
SAS-data-set(IN=variable)
```

    - *variable* is a temporary numeric variable with value either 0 or 1:

Value	Indication
0	The *SAS-data-set* did NOT contribute to the current observation
1	The *SAS-data-set* did contribute to the current observation

    - We revisit the example given earlier and see how the observations of Data Set Work.C are formed.

(a) Program

```
data work.c;
  merge work.a(IN=IN_A) work.b (IN=IN_B);
  by key;
run;
```

(b) Input data sets:

Data Set Work.A

VarA1	VarA2	Key
A1	AA1	K1
A2	AA2	K3
A3	AA3	K4
A4	AA4	K5

Data Set Work.B

Key	VarB2	VarB3
K1	B1	BB1
K2	B2	BB2
K3	B3	BB3
K5	B4	BB4
K6	B5	BB5

(c) Compilation: PDV is formed based on VarA1, VarA2, Key, VarB2 andVarB3.

PDV

VarA1	VarA2	Key	IN_A	VarB2	VarB3	IN_B
			Drop			Drop

(d) Execution:

Note that the BY group variable is 'Key'. SAS finds the first BY group from Data Sets, Work.A and Work.B, i.e. 'K1'

Data Set Work.A

VarA1	VarA2	Key
A1	AA1	K1
A2	AA2	K3
A3	AA3	K4
A4	AA4	K5

Data Set Work.B

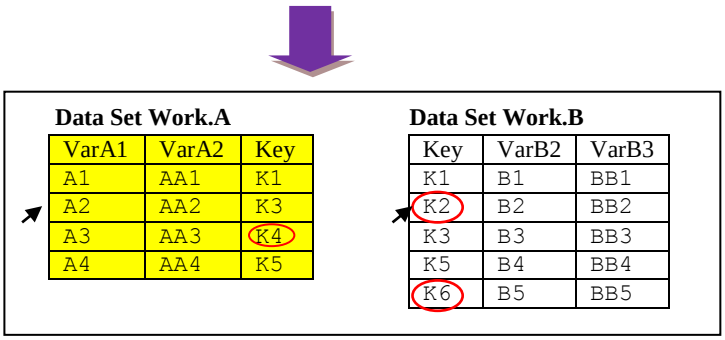
Key	VarB2	VarB3
K1	B1	BB1
K2	B2	BB2
K3	B3	BB3
K5	B4	BB4
K6	B5	BB5

PDV

VarA1	VarA2	Key	IN_A	VarB2	VarB3	IN_B
			Drop			Drop
A1	AA1	K1	1	B1	BB1	1

1 in IN_A indicates the observation is contributed into the PDV for output to new data set (Data Set Work.C). Similarly, for the case of having 1 in IN_B.

Note that all observations in the BY group, ‘K1’ are output to the new Data Set Work.C. Then SAS puts missing in the PDV for all variable unique to Work.A and Work.B. SAS continues to find the second BY group: ‘K2’.



Data Set Work.A

VarA1	VarA2	Key
A1	AA1	K1
A2	AA2	K3
A3	AA3	K4
A4	AA4	K5

Data Set Work.B

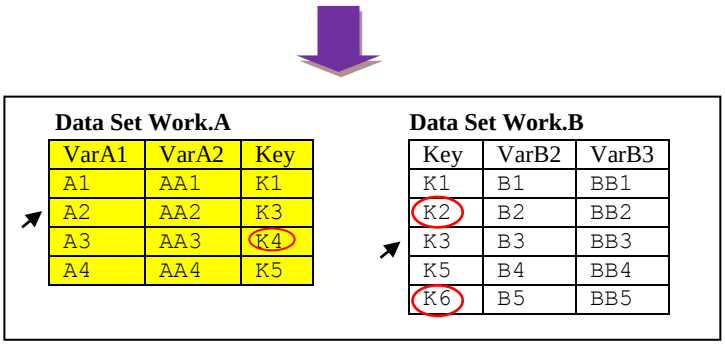
Key	VarB2	VarB3
K1	B1	BB1
K2	B2	BB2
K3	B3	BB3
K5	B4	BB4
K6	B5	BB5

PDV

VarA1	VarA2	Key	IN_A	VarB2	VarB3	IN_B
			Drop			Drop
		K2	0	B2	BB2	1

The two values of variable ‘KEY’ are different in the data sets. Since K2 is the first value after K1, therefore the observation in Work.B is a contribution to Work.C. 0 in IN_A indicates the observation in Work.A is NOT contributed into the PDV for output while 1 in IN_B indicates that the observation is a contribution to the PDV for output.

After outputting values from PDV onto Work.C, SAS looks at the NEXT BY group, which is ‘K3’. SAS puts missing values into PDV for those variables unique to the Data Sets, Work.A and Work.B.



Data Set Work.A

VarA1	VarA2	Key
A1	AA1	K1
A2	AA2	K3
A3	AA3	K4
A4	AA4	K5

Data Set Work.B

Key	VarB2	VarB3
K1	B1	BB1
K2	B2	BB2
K3	B3	BB3
K5	B4	BB4
K6	B5	BB5

PDV

VarA1	VarA2	Key	IN_A	VarB2	VarB3	IN_B
			Drop			Drop
A2	AA2	K3	1	B3	BB3	1

Both values of variable KEY match. Both observations are contributed into PDV for output. Therefore, 1 in IN_A and 1 in IN_B.

Note that all observations in the BY group, ‘K3’ are output to the new Data Set Work.C. Then SAS puts missing in the PDV for all variable unique to Work.A and Work.B. SAS continues to find the second BY group.

Data Set Work.A

VarA1	VarA2	Key
A1	AA1	K1
A2	AA2	K3
A3	AA3	K4
A4	AA4	K5

Data Set Work.B

Key	VarB2	VarB3
K1	B1	BB1
K2	B2	BB2
K3	B3	BB3
K5	B4	BB4
K6	B5	BB5

Ex. 6.3: Fill in values in the PDV for the above step.

PDV

VarA1	VarA2	Key	IN_A	VarB2	VarB3	IN_B
			Drop			Drop

The result data set:

Data Set Work.C*

Key	VarA1	VarA2	VarB2	VarB3
K1	A1	AA1	B1	BB1
K2			B2	BB2
K3	A2	AA2	B3	BB3
K4	A3	AA3		
K5	A4	AA4	B4	BB4
K6			B5	BB5

* Blanks are missing

- PDV results:

PDV (6-steps)

VarA1	VarA2	Key	IN_A*	VarB2	VarB3	IN_B*
			Drop			Drop
A1	AA1	K1	1	B1	BB1	1
		K2	0	B2	BB2	1
A2	AA2	K3	1	B3	BB3	1
A3	AA3	K4	1			0
A4	AA4	K5	1	B4	BB4	1
		K6	0	B5	BB5	1

* The variables, IN_A and IN_B are created with the IN= data option are only available during the execution and are NOT written to the SAS data set.

- Data set outputs

(a) Subsetting IF statement is used to output those matches records and non-matches records. We use the above example for demonstration:

```
data work.c <additional data sets for multiple outputs>;
  merge work.a(IN=IN_A) work.b (IN=IN_B);
  by key;

  <add subetting IF statement for single output or
   add IF-THEN-ELSE statements for multiple outputs>
run;
```

(i) Outputting the matches: "IF IN_A=1 and IN_B=1;" or "IF IN_A and IN_B;"
(ii) Outputting the nonmatches in Work.A : "IF IN_A=1 and IN_B=0;" or "IF IN_A and NOT IN_B";
(iii) Outputting the nonmatches in Work.B: "IF IN_A=0 and IN_B=1;" or "IF NOT IN_A and IN_B;"

(b) IF-THEN-ELSE statement is used for multiple outputs.
(i) Note that DATA statement is required to include all output data sets file name.
(ii) We dump the matches to Work.C, nonmatches in Work.A to Work.A_Only and nonmatches in Work.B toWork.B_Only. And the program becomes:

```
data work.c Work.A_Only Work.B_Only;
  merge work.a(IN=IN_A) work.b (IN=IN_B);
  by key;
   IF IN_A=1 and IN_B=1 THEN OUTPUT Work.c;
   ELSE IF IN_A=1 and IN_B=0 THEN OUTPUT Work.A_Only;
        ELSE IF IN_A=0 and IN_B=1 THEN OUTPUT Work.B_Only;
run;
```

(iii) The output data sets:

Data Set Work.C

Key	VarA1	VarA2	VarB2	VarB3
K1	A1	AA1	B1	BB1
K3	A2	AA2	B3	BB3
K5	A4	AA4	B4	BB4

Data Set Work.A_only*

Key	VarA1	VarA2	VarB2	VarB3
K4	A3	AA3		

Data Set Work.B_only*

Key	VarA1	VarA2	VarB2	VarB3
K2			B2	BB2
K6			B5	BB5

* Blanks are missing

Ex. 6.4: Given two data sets, dataX and dataY with common variable, CKEY:

Data Set Work.dataX

CKey	A	B
1	100	202
3	300	404

Data Set Work.dataY

CKey	C
1	C1
2	C2

Write an appropriate IF statement to be added into the following SAS program for creating the desired data sets:

```
data new;
   merge dataX (in=X) dataY (in=Y);
   by CKey;
run;
```

Desired Data Set

CKey	A	B	C
2			C2
3	300	404	

Ans:

- Other method for merging or manipulating data: SQL
  - The SQL procedure is the SAS implementation of Structured Query Language.
  - PROC SQL is part of base SAS software and is an alternative to DATA step or other SAS procedure.
  - SQL is a powerful data manipulation tool and is commonly used in banking and government systems.

- However, SQL tool is not covered in the SAS Base Programming Examination and therefore is not discussed in details in the course.
- SQL produces a Cartesian product of the many-to-many merge:

Data to be merged:

Data Set Work.A

VarA1	VarA2	Key
A1	AA1	K1
A2	AA2	K1
A3	AA3	K2
A4	AA4	K2
A5	AA5	K2

Data Set Work.B

Key	VarB2	VarB3
K1	B1	BB1
K1	B2	BB2
K1	B3	BB3
K2	B4	BB4
K2	B5	BB5

```
proc sql;
 create table Work.c as
 select VarA1, VarA2, VarB2,VarB3, B.Key from Work.A,Work.B
 where A.Key=B.Key;

 proc print data=c;
 run;
```

Output, which is different from the many-to-many merge:

Obs	Var A1	Var A2	Var B2	Var B3	Key
1	A1	AA1	B1	BB1	K1
2	A1	AA1	B3	BB3	K1
3	A1	AA1	B2	BB2	K1
4	A2	AA2	B1	BB1	K1
5	A2	AA2	B3	BB3	K1
6	A2	AA2	B2	BB2	K1
7	A3	AA3	B4	BB4	K2
8	A3	AA3	B5	BB5	K2
9	A4	AA4	B4	BB4	K2
10	A4	AA4	B5	BB5	K2
11	A5	AA5	B4	BB4	K2
12	A5	AA5	B5	BB5	K2

Ex. 6.5: Ex6.2 revisit. Given

AMA.Banquet_List

Banquet_Host
No_of_Guest
Total_spending
Quantity
Supplier

BonusGift.xls (Work.Supplier1)

SuppID
Gift
Quantity

Write a SAS program to list of the hosts who can find a supplier and the corresponding gift and minimum quantity to be ordered.