Learning outcomes:

1. Create a SAS data set from an Excel worksheet
2. Create an Excel spreadsheet from a SAS data set
3. Create a SAS data set from a delimited raw data file
4. Create a SAS data set using line pointer controls
5. Introduction to variables manipulation in DATA step
6. DATA step compilation and execution phases

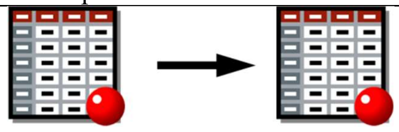
SAS components learnt:

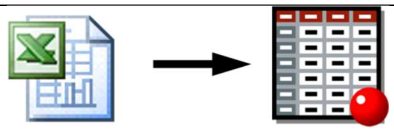
1. DATA step
2. PROC COPY
3. IMPORT Wizard
4. EXPORT Wizard
5. LENGTH statement
6. INPUT statement/multiple INPUT statements
7. Assignment statements
8. DROP statement
9. KEEP statement
10. Use of "@" and "@@" in INPUT statement
11. DATALINES statement

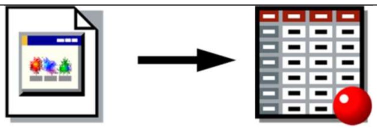
3.1 Source of SAS Data Sets

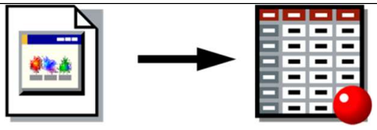
- The source of SAS data sets can be summarized as 5 categories: SAS data sets, Excel Worksheet, delimited raw data, other raw data text file and in-stream data. The source data from a SAS data set or in-stream data are discussed in Chapter 2.
- LIBNAME statement and DATA steps are used for recreating new data sets. Note that the syntaxes are slightly different.

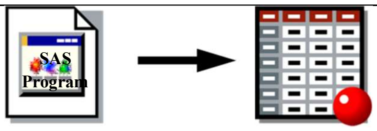
Data Source	SAS Operation	SAS Program Syntax

SAS Data Sets		libname ____________________;
		data ______________;
		set ___________;
		...
		run;

Excel Worksheet		libname ______________________;
		data ________________;
		set ______________;
		...
		run;

Delimited Raw Data Files, e.g. csv		data ______________________;
		infile ______________;
		input ______________;
		...
		run;

Other Raw Data Text Files		data ______________________;
		infile ______________;
		input ____@_________;
		...
		run;

In-stream Data		data ______________________;
		input ______________@@;
		datalines;
		...
		run;

3.2 Creating a SAS data sets from reading Excel Worksheet

- Program Syntax

```
LIBNAME libref 'physical-file-name';
DATA output-SAS-data-set;
       SET input-SAS-data-set;
       WHERE where-expression;
       KEEP variable-list;
       LABEL variable = 'label'
             variable = 'label'
             variable = 'label';
       FORMAT variable(s) format ;
RUN;
```

- The LIBNAME statement enables you to reference worksheets directly in a DATA step or to read from and write to a Microsoft Excel worksheet as if it were a SAS data set.
- Once SAS has a *libref* assigned to an Excel workbook, the workbook cannot be opened in Excel.
- In order to disconnect SAS from the data source and **closes any resources** that are associated with that libref's connection, disassociation with a *libref* is required. Use a LIBNAME statement by specifying the *libref* and the CLEAR option:

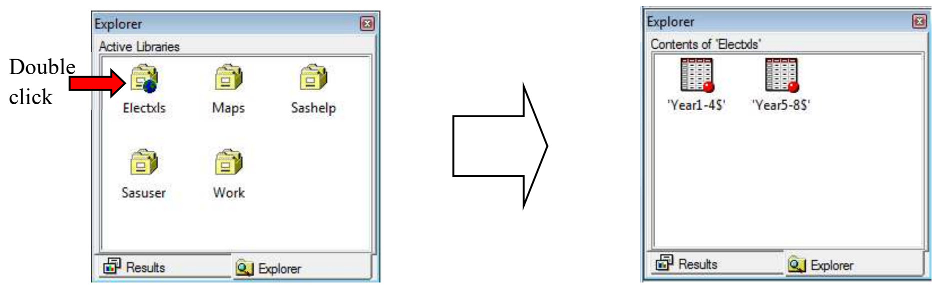
```
Libname libref clear;
```

- The SET statement is used for the reading of input data file. When excel file is used, the input data set can be written as `libref.'worksheet_name$'n`.
- The DATA, WHERE, KEEP, LABEL, FORMAT statements are defined as stated in Chapter 2.

- An example: Electricity data
  - The electricity data have been re-entered into spreadsheet format. Two more columns for the first date and last date of the period are added. Besides this, observations are split into observations of Years 1 to 4 and of Years 5 to 8. These observations are saved in worksheet "YEAR1-4" and "YEAR5-8" respectively.
  - Submit the following LIBNAME statement:

```
libname electxls 'd:\SAS_datasets\electricity.xls';
```

  - Folder "Electxls" appears inside the SAS libraries and double click Library "Electxls",'Year1-4$' and 'Year5-8$' tables are inside the library.
  - The SAS data file name for these tables are: 'ELECTXLS.YEAR1-4$'n and 'ELECTXLS.YEAR5-8$'n.



  - Procedure CONTENTS can be used to list all tables:

```
proc contents data=electxls._all_;
run;
```

The outputs are:

- Summary of the library:

```
                              The CONTENTS Procedure

                                    Directory
                    Libref          ELECTXLS
                    Engine          EXCEL
                    Physical Name   d:\SAS_datasets\electricity.xls
                    User            Admin


                                                DBMS
                                       Member   Member
                        #   Name       Type     Type
                        1   'Year1-4$'  DATA     TABLE
                        2   'Year5-8$'  DATA     TABLE
```

Found in reading xls ONLY!

- Information of first table

```
                              The CONTENTS Procedure

 Data Set Name        ELECTXLS."'Year1-4$'"n     Observations          .
 Member Type          DATA                       Variables             7
 Engine               EXCEL                      Indexes               0
 Created              .                          Observation Length    0
 Last Modified        .                          Deleted Observations  0
 Protection                                      Compressed            NO
 Data Set Type                                   Sorted                NO
 Label
 Data Representation  Default
 Encoding             Default


                     Alphabetic List of Variables and Attributes

   #    Variable       Type    Len    Format     Informat     Label

   5    AvTemp         Num       8                            AvTemp
   3    Cost           Num       8                            Cost
   4    Deg_Days       Num       8                            Deg Days
   6    FirstDate      Num       8    DATE9.     DATE9.       FirstDate
   7    LastDate       Num       8    DATE9.     DATE9.       LastDate
   1    Time_Period    Char     17    $17.       $17.         Time Period
   2    kWh            Num       8                            kWh
```

- Information of second table

```
                              The CONTENTS Procedure

 Data Set Name        ELECTXLS."'Year5-8$'"n     Observations          .
 Member Type          DATA                       Variables             7
 Engine               EXCEL                      Indexes               0
 Created              .                          Observation Length    0
 Last Modified        .                          Deleted Observations  0
 Protection                                      Compressed            NO
 Data Set Type                                   Sorted                NO
 Label
 Data Representation  Default
 Encoding             Default


                     Alphabetic List of Variables and Attributes

   #    Variable       Type    Len    Format     Informat     Label

   5    AvTemp         Char      1    $1.        $1.          AvTemp
   3    Cost           Num       8                            Cost
   4    Deg_Days       Num       8                            Deg Days
   6    FirstDate      Num       8    DATE9.     DATE9.       FirstDate
   7    LastDate       Num       8    DATE9.     DATE9.       LastDate
   1    Time_Period    Char     17    $17.       $17.         Time Period
   2    kWh            Num       8                            kWh
```

Average temperature is numeric but missing data in EXCEL are denoted by blanks!

- To print out the data of second table, use procedure PRINT:

```
proc print data=electxls.'YEAR5-8$'n;
run;
```

- To release the Excel file for opening, use LIBNAME statement:

```
libname electxls CLEAR;
```

- Use DATA steps
  Syntax

```
Libname Librefxls 'Input-Excel-file-in-full-path';
Data output-sas-data-set-name-1;
   Set Librefxls.input-sas-data-set-name-1;
Run;
Data output-sas-data-set-name-2;
   Set Librefxls.input-sas-data-set-name-2;
Run;
:
:
Data output-sas-data-set-name-n;
   Set Librefxls.input-sas-data-set-name-n;
Run;


PROC CONTENTS data=Librefxls._all_;
Run;
```

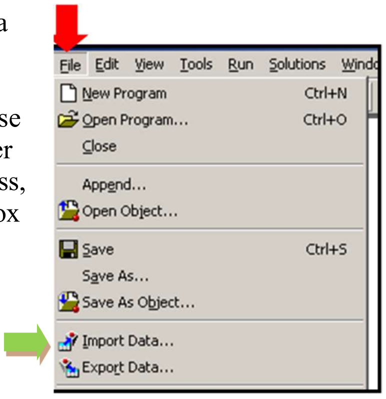
- The *i*-th yellow-highlighted name is the *i*-th worksheet name to be input from Excel.
- Create two SAS data sets for the two worksheets of electricity.xls:

```
*p3ex1.sas;
libname xls "D:\SAS_Datasets\electricity.xls";
data Years1;
  set xls.'Year1-4$'n;
run;

data Years2;
  set xls.'Year5-8$'n;
run;

proc contents data=work._all_ ;
run;
```

Ex. 3.1: (a) How many observations are there for Work.Years1 and Work.Years2?
(b) Is there any error when "Years1" in the above SAS program is replaced by "Years1-4"?

Ans:

- IMPORT Wizard and Procedure
  - Data can be read from external PC files to SAS data sets.
  - The Import Wizard and procedure is a part of Base SAS and enable access to delimited files. The license to SAS/ACCESS Interface to PC File Formats, offer an access to/from Microsoft Excel, Microsoft Access, dBASE, JMP, Lotus 1-2-3, SPSS, Stata, and Paradox files.
  - the wizard is a point-and-click interface and the procedure is code-based. The code can be automatically generated as SAS program.
  - To invoke the wizard from the SAS windowing environment, select **File** > **Import Data**.
  - Follow the instructions and type in the program name and location for saving the PROC IMPORT for the importing data process.



- Program generated from IMPORT Wizard

```
PROC IMPORT OUT= output-SAS-data-set
            DATAFILE= "Excel-file-in-full-name"
            DBMS=EXCEL REPLACE;
          RANGE="'Worksheet-Name$'";
       GETNAMES=YES;
          MIXED=NO;
       SCANTEXT=YES;
        USEDATE=YES;
       SCANTIME=YES;
RUN;
```

Ex. 3.2: Use IMPORT Wizard to import the Excel worksheet "Year5-8" in Electricity.xls. Set the name of the SAS data-set as "Work.Years_a". Fill in the blanks of PROC IMPORT:

```
PROC IMPORT OUT= Work.Years_a
            DATAFILE= "__________________________"
            DBMS=EXCEL REPLACE;
          RANGE="________________________";
       GETNAMES=YES;
          MIXED=NO;
       SCANTEXT=YES;
        USEDATE=YES;
       SCANTIME=YES;
RUN;
```

3.3 Create an Excel spreadsheet from a SAS data set

- Use DATA steps
  - Syntax

```
libname SAS_ref "location of SAS source data sets";
libname libxls "output Excel spreadsheet name and its path";

data libxls.worksheet-name;
  set SAS_ref.Source-SAS-data-set;
run;

libname libxls clear;
```

Close resources for assessing Excel

- Use LIBNAME statement to define the name and path of the Excel file for output
- The name of the worksheet is defined in the DATA statement.

- The SAS data sets: electricity consumption (ELECTRICITY) and alcohol and tobacco use in animated children's movies (CHMOVIE) are output to Excel file named as d:\SAS_datasets\new.xls:

```
*p3_outputxls.sas;
libname temp "d:\SAS_datasets";
libname newxls "d:\SAS_datasets\new.xls";

data newxls.elect;
  set temp.electricity;
run;

data newxls.chmovie;
set temp.chmovie;
run;

libname newxls clear;
```

Procedure COPY

- Syntax to create an excel file for SAS data sets:

```
Libname Librefxls 'Output-Excel-file-in-full-path';
PROC COPY IN=source-library OUT=Librefxls;
        SELECT SAS-data-set-1 SAS-data-set-2 … SAS-data-set-n;
RUN;
```

  - "*SAS-data-set-i*" represents the i-th data set in the source library.

When the LIBNAME statement is used to define a library for storing SAS data sets, PROC COPY is used to copy the SAS-data-sets into another library named in the OUT option.

Ex. 3.3: Define the directory of d:\SAS_datasets as library "INLIB" in the SAS library. Write the SAS codes using procedure Copy to copy the SAS data sets: electricity consumption (ELECTRICITY) and alcohol and tobacco use in animated children's movies (CHMOVIE) into an Excel file called output.xls:

EXPORT Wizard and Procedure

- Data can be written between SAS data sets and external PC files.
- The Export Wizard and procedure is a part of Base SAS and enable access to delimited files. The license to SAS/ACCESS Interface to PC File Formats, offer an access to/from Microsoft Excel, Microsoft Access, dBASE, JMP, Lotus 1-2-3, SPSS, Stata, and Paradox files.
- The wizard is a point-and-click interface and the procedure is code-based. The code can be automatically generated as SAS program.
- To invoke the wizard from the SAS windowing environment, select **File** > **Export Data** (See diagram of IMPORT Wizard).
- Follow the instructions and type in the program name and location for saving the PROC EXPORT for the exporting data process.

- Program generated from EXPORT Wizard
  - An example of exporting the SAS data set "Work.Years_a" into Excel file "abc.xls" with sheet name='Year5-8".

```
PROC EXPORT OUT= Work.Years_a  /*p3_exportSAS_EXCEL.sas */
            OUTFILE= "d:\SAS_datasets\abc.xls"
            DBMS=EXCEL REPLACE;
           SHEET="Year5-8";
RUN;
```

3.4 Create a SAS data set from a delimited raw data file

- Delimited raw data file
  - A delimiter is used to separate values of an observation.
  - Comma separated values (.CSV) format is commonly used. The delimiter is ",".
  - Data can be classified as standard and non-standard forms:
    - Standard: No special SAS instructions is required. An example:

```
58     -23      67.23      00.99      5.67E5
```

    - Non-standard: special instructions are required to be specified in INPUT statement. e.g. different format for different variables, see the example:

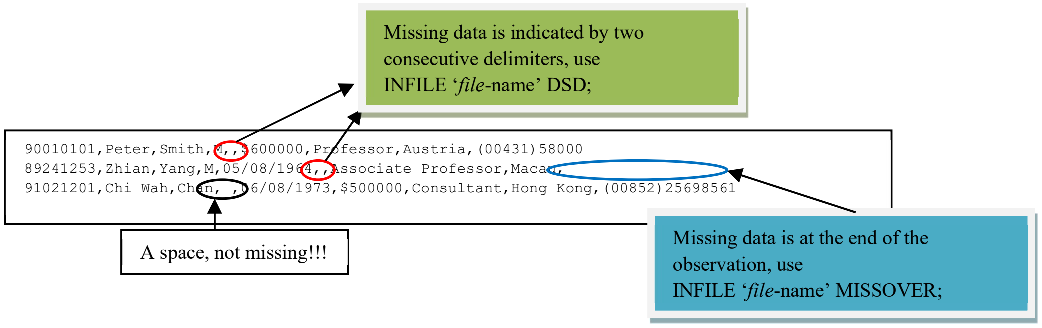
```
5,823      (23)      $67.23      01/12/1999      12MAY2006       1.2E-2
```

- Program Syntax

```
DATA output-SAS-data-set;
        LENGTH variable(s) <$> length;
        INFILE 'raw-data-file-name' <DLM='delimiter'><DSD>
                                    <MISSOVER>;
        INPUT variable <$> variable <:informat>;
        KEEP variable-list;
        LABEL variable = 'label'
              variable = 'label'
              variable = 'label';
        FORMAT variable(s) format;
RUN;
```

- LENGTH statement
  - specifies the storing size defined in the program data vector (PDV) in the memory for each variable listed in the INPUT statement.
  - The default length for character and numeric variables is 8 bytes.
- INFILE statement
  - specifies name of the raw data file (can be in full path or not). Note that different computers have different default paths.
  - DLM/DSD/MISSOVER options would be combined differently for different type of missing data.
  - Without DLM option, **space** delimited raw data file is assumed to be input. i.e. space is the delimiter in this case.
  - `DLM=','` for CSV files.

- When DSD (delimiter-sensitive data) option is used, SAS treats two consecutive delimiters as a missing values and removes quotation marks from character values.
- A MISSOVER option is used when there is a missing value at the end of a record. It prevents SAS from loading a new record when the end of the current record is reached. And the fields without any values are then set to missing.



```
90010101,Peter,Smith,M,,$600000,Professor,Austria,(00431)58000
89241253,Zhian,Yang,M,05/08/1964,,Associate Professor,Macau,
91021201,Chi Wah,Chan, ,06/08/1973,$500000,Consultant,Hong Kong,(00852)25698561
```

Missing data is indicated by two consecutive delimiters, use INFILE '*file*-name' DSD;
A space, not missing!!!
Missing data is at the end of the observation, use INFILE '*file*-name' MISSOVER;

- INPUT statement
  - specifies the variables and their order from left to right given in the raw data file.
  - "$" is added after a variable name to indicate a character value instead of a numeric value.
  - SAS informat is an instruction how SAS read the values into the variable. The general form of the SAS informat:

```
<$>informat<w>.<d>
```

(a) $ indicates a character informat
(b) Informat names the SAS informat or used-defined informat (See user-defined format (PROC FORMAT)).
(c) *w* specifies the number of columns to read in the input data
(d) *d* specifies an optional decimal scaling factor in the numeric informats
(e) *w* and *d* can be omitted and the informats become:

```
<$>informat.
```

(f) Provided that LENGTH statement is given, SAS use the length defined in the statement for the size of the numeric and character variables.

Numeric Data	Informat	Raw Data Value	SAS Data Value
Currency	`COMMA.`	$123,456	123456
	`DOLLAR.`		
	`COMMAX.`	$123.456	123456
	`DOLLARX.`		
	`EUROX.`	€123.456	123456
DATE	`MMDDYY.`	010160 01/01/60 01/01/1960	0
	`DDMMYY.`	311259 31/12/59 31/12/1959	-1
	`DATE.`	31DEC60 31DEC1960	365

Ex. 3.4: Write a SAS program to read the AMA contact information which is saved as AMA_Contacts.csv. Note that the variables are listed in this order: name, phone number and fax number. Assume the phone and fax numbers are character type.

```

```

Output:

```
Name                          Phone                Fax

Li-qun Qi                (852) 2766-4591     (852) 2362-0945
David G. Luenberger
Chi-kin Chan             (852) 2766-6928     (852) 2362-0945
Joseph H.W. Lee          (852) 2766-6951
Dorothy Chung            (852) 2766-6920     (852) 2362-0945
```

Remark: There are two versions of INFILE statement for creating the above output.

Output for `infile 'd:\SAS_datasets\AMA_contacts.csv' DLM=',';`

```
Name                          Phone                Fax

Li-qun Qi                (852) 2766-4591     (852) 2362-0945
David G. Luenberger      Chi-kin Chan        (852) 2766-6928
Joseph H.W. Lee          (852) 2766-6951     Dorothy Chung
```

Output for `infile 'd:\SAS_datasets\AMA_contacts.csv' MISSOVER;`

```
Name        Phone                Fax

Li-qun      Qi,(852)        2766-4591,(852)
David       G.              Luenberger,,
Chi-kin     Chan,(852)      2766-6928,(852)
Joseph      H.W.            Lee,(852)
Dorothy     Chung,(852)     2766-6920,(852)
```

Question: why the output is totally different from the one using option "DLM=',"'?

3.5 Create SAS data set from other forms of ASCII data file

- Multiple INPUT statements for multiple records
  - Syntax

```
DATA SAS-data-set;
INFILE 'raw-data-file-name';
INPUT specifications;
⁝
INPUT specifications;
<additional SAS statements>
RUN;
```

  - Each INPUT statement ends
  - Used to read a raw data file with multiple records per observations
  - For example, name and address:

```
Miss Dorothy CHUNG
FJ610,
The Hong Kong Polytechnic University
Hung Hom
Hong Kong, China
```

- Data split into Full_Name (1st line), Address1 (2nd line), Address2 (3rd line), District Code (4th line), Country (5th line).
- Each INPUT statement normally allows SAS to read one line of the input file. In fact, SAS counts the number of characters for each input variable. When the number of character of the current line is less than the number of character of the variable to be input, SAS will look for the next line.
- Skipping a line during reading an input data set: "INPUT;"

Ex. 3.5: Submit the following two programs. Compare the results of data sets Work.Address1 and Work.Address2.

```
*p3_address.sas;
data address1;
    infile 'd:\SAS_datasets\address.txt'  ;
    input Fullname $30.;
    input Address1 $48.;
    input Address2 $48.;
    input District $20.;
    input  Country $30.;
run;

proc print data=address1;
run;

data address2;
    infile 'd:\SAS_datasets\address.txt'  ;
    input Fullname $30.;
    input;
    input Address2 $48.;
    input District $20.;
    input  Country $30.;
run;

proc print data=address2;
run;
```

Obtain the output for Work.Address1 and Work.Address2.

FJ610 is not read. So INPUT Statement without a variable list means skipping a line during reading a data set.

- Line Pointer Controls
  - Relative line pointer control
    - Use of forward slash '/' in INPUT statement
    - SAS loads the next record when it encounters a forward slash.
    - The forward slash is known as a relative line pointer control which moves the pointer relative to the line on which it currently positioned.
    - Address example revisit:

```
*p3_forward_slash.sas;
data address3;
  infile 'd:\SAS_datasets\address.txt';
  input Fullname $30. /
         Address1 $48. /
         Address2 $48. /
         District $20. /
         Country $30.;
run;

proc print data=address3;
run;

data address4;
  infile 'd:\SAS_datasets\address.txt';
  input Fullname $30. /
         / /
        District $20. /
         Country $30.;
run;

proc print data=address4;
run;
```

Output:

Work.Address3

```
Obs        Fullname         Address1                Address2                 District    Country

 1    Miss Dorothy CHUNG    FJ610     The Hong Kong Polytechnic University   Hung Hom   Hong Kong, China
```

Work.Address4

```
                    Obs         Fullname          District      Country

                     1     Miss Dorothy CHUNG     Hung Hom     Hong Kong, China
```

  - Absolute line pointer control
    - Use a hash "#" to specify which line in a group in the INPUT statement:

```
INPUT specifications / specifications;
```

    - Address data revisit:

```
*p3_hash.sas;
data address5;
  infile 'd:\SAS_datasets\address.txt';
  input #1 Fullname $30.
        #4 District $20.
        #5 Country $30.;
run;

proc print data=address5;
run;
```

Output:

Work.Address5

```
        Obs         Fullname          District     Country

         1      Miss Dorothy CHUNG    Hung Hom    Hong Kong, China
```

- The single trailing @

```
INPUT specifications … @;
```

  - The pointer position does not change.
  - No new record is read into the input buffer.
  - The next INPUT statement for the same iteration of the DATA step continues to read the same record rather than a new one.
  - SAS releases a record held by a trailing @ when
    (a) A null INPUT statement executes(input;)
    (b) An INPUT statement without a trailing @ executes
    (c) The next iteration of the DATA step begins

Data are tabulated in columns, use different '@' in the INPUT statement for the following purposes:

- Holding a Record in the Input Buffer: one single "@" forces SAS stopping to read the next value.
  - An example with raw data: The first column contain 'C' and 'L'. A letter 'C' is followed by a course code and Lecturer name; A letter 'L' is followed by the lecturer surname and his/her telephone extension.

Course_lecturer_info.txt:

```
----+----1----+----2----+
C AMA151  Watson
L Williams 0324
L Flores   7420
C AMA202  Sen
L Lee      7085
```

  - In order to read the lecturers' information correctly, a stopping "@" is required after reading the first column. Only the lecturers' information is output to AMALecturer. So the SAS program is written as:

```
*p3_AMALectuerer.sas;
data AMALecturer(drop=type course lecturer);
   infile 'd:\SAS_Datasets\course_lecturer_info.txt';
   retain Course Lecturer;
   input type $1. @;
   if type='C' then
      input @3 course $  Lecturer $;
   else if type='L' then
      do;
        input @3  LectName $9. Phone_Ext $ ;
        output AMALecturer;
      end;
run;
```

- Reading fixed width columns
  - INPUT statement is changed to the following syntax:

```
INPUT var-1 <$> m1-n1 var-2 <$> m2-n2 <…>;
```

(a) *m1* is the start column position and *n1* is final column position of *var-1*.
(b) Similarly, for the case of *var-2*.
(c) $ is used for a character variable

- Move pointer position for reading

```
INPUT @m1 var-1 <$> @m2 var-2 <$> <…@m-n var-n <$>>;
```

(a) Move the pointer to column position *m1* before reading variable *var-1*.
(b) Move the pointer to column position *m2* before reading variable *var-2*.
(c) $ is added after the name of a character variable.

- Two TOEFL scores for students belonging to Club1.

```
data club1;
   input Id Name $18.
         Class $ 24-30 Toefl1 Toefl2;
   Datalines
/*--+----|----+----|----+----|----+----|*/;
023 David Shaw          red    213 240
049 Amelia Serrano      yellow 200 300
...
;
```

(a) Variable 'Name' is a character variable with length 18 ($18.)
(b) Variable 'Class' is using fixed column position
(c) Those variables without '$' are numeric.
(d) The column position of class can be changed to '@24' before reading 'Class" and use '@30' to move SAS prompt to position 30 before reading TOFEL Score 1. The above SAS program can be amended in the following way:

```
data club1;
   input Id Name $18.
         @24 Class $ @30 Toefl1 Toefl2;
   datalines;
023 David Shaw          red    213 240
049 Amelia Serrano      yellow 200 300
...
;
```

- Data are listed in the SAS program
  - Use of DATALINES statement (Refer to Chapter 2).
    - Observations are listed right after DATALINES statement.
    - All variables of one observations are listed in one line.
    - When multiple observations are written, use the double trailing @ to control the input pointer across iterations of the DATA step.
    - An example

```
*p3example_datalines.sas;
data test;
   input name $ age @@;
   datalines;
John 13 Monica 12 Sue 15 Stephen 10
Marc 22 Lily 17
;

data test1;
   input name $ age ;
   datalines;
John 13 Monica 12 Sue 15 Stephen 10
Marc 22 Lily 17
;
```

(a) Dataset Work.test has six observations

Name	**Age**
John	13
Monica	12
Sue	15
Stephen	10
Marc	22
Lily	17

(b) Dataset Work.test1 has two observations

Name	**Age**
John	13
Marc	22

3.6 Variables manipulation in DATA step

- Assignment statement

```
variable = expression;
```

- The *variable* can be the existing variable or a new one and will be automatically included in the output data set. If the variable is not required to be output, use DROP statement.
- The *expression* is a sequence of operands and operator which form a set of instructions for producing a value. It can be defined as arithmetic expression of constants and/or variables (e.g. `Year_Salary=12*Month_Salary`), or values derived from SAS implicitly defined function(s) of variable(s).
- Operands are constants (numeric, character or date) and variable (numeric or character).

Type of constants	Example of an assignment statement
Numeric	Monthly_Salary=3000;
Character	Name='Dorothy' ;
Date	Join_date='01JAN2010'd ;

Ex. 3.6: What is the maximum length of a variable?
What is the restriction on the first character of variables?

- SAS functions
  - For example, create a new variable for the record edit date. Edit_date=TODAY(). TODAY function give the SAS date of today.
  - Details of SAS functions will be discussed in Chapter 4.

- KEEP and DROP statements
  - Syntax

```
KEEP variable-list;
```

```
DROP variable-list;
```

- The KEEP statement specifies the variable for output. i.e. Only observations of those variables listed in the KEEP statement will be written to the output data set(s).
- The DROP statement specifies the variable to be omitted from the output data set(s).

Ex. 3.7: Create a new temporary data set with name "NEW" from SAS data file "electricity.sas7bdat" so that the average temperature is expressed in degree Celsius only (x). [Hint: x = (temperature in degree F-32)×5/9; Do we need the original average temperature in Fahrenheit?]

3.7 DATA step compilation and execution phases

Program Data Vector or Logical Program Data Vector (PDV)

- The second area in memory which provides slots to store temporarily the following:
  - Relative variable number (not displayed in the memory area)
  - Position in the dataset
  - Name
  - Data type (numeric (no $ sign), character ($) in the memory area)
  - Length in bytes
  - Informat
  - Format
  - Variable label
  - Flags to indicate dropping and retaining of variables

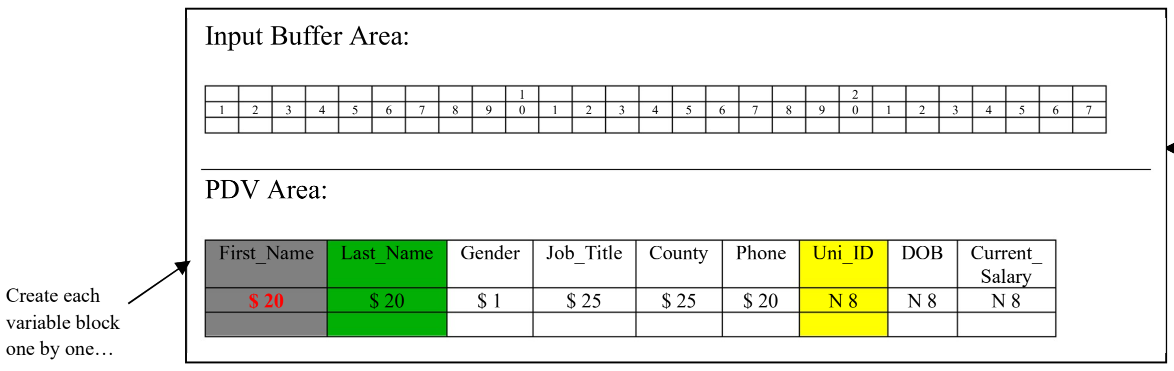
- Student records example:
  - Data in csv

```
90010101,Peter,Smith,M,01/07/1965,$600000,Professor,Austria,(00431)58000
89241253,Zhian,Yang,M,15/08/1964,$750000,Associate Professor,Macau,(00853)123456
91021201,Chi Wah,Chan,F,06/08/1973,$500000,Consultant,Hong Kong,(00852)25698561
```

  - SAS program for reading the data:

```
*p3_grad93.sas;
Data work.grad93;
  Infile 'AMAgrad93.csv' dlm=',';
  Length First_Name $ 20 Last_Name $20 Gender $1
         Job_Title $25 Country $25 Phone $ 20;
 Input Uni_ID First_Name $ Last_Name $ Gender $ DOB:DDMMYY.
      Current_Salary: Dollar. Job_Title $ Country $ Phone $;
Run;
```

  - During the compilation, an input buffer is allocated in the memory unit for temporary storage of the input data. The PDV is created block by block according to the order of the variable listed from left to right in INPUT statement.



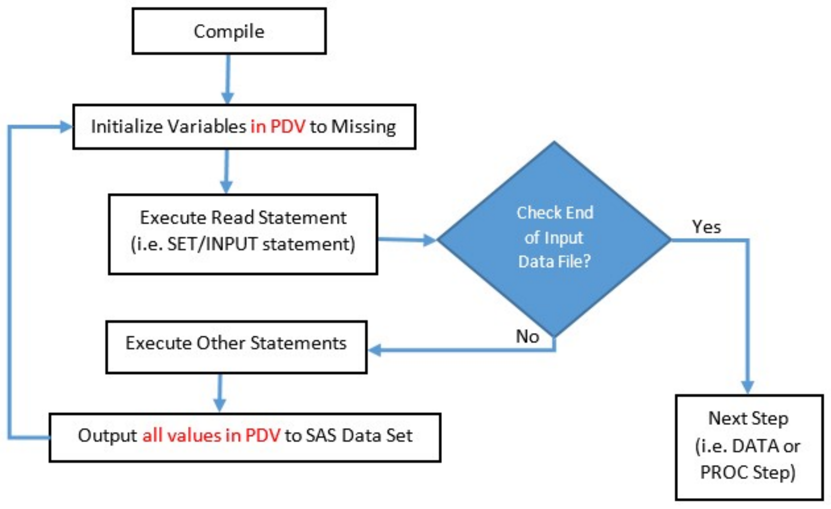
First_Name	Last_Name	Gender	Job_Title	County	Phone	Uni_ID	DOB	Current_ Salary
$ 20	$ 20	$ 1	$ 25	$ 25	$ 20	N 8	N 8	N 8

Note: The actual ordering of variable creation in PDV: First_Name, Last_Name, Gender, Job_Title, Country, Phone, (LENGTH statement) Uni_ID, DOB, Current_Salary. (INPUT statement).

- Data Processing
  - The DATA step is processed in two phases:
    - Compilation
      (a) Check the syntax of the DATA step statements
      (b) Create an input buffer to hold the current raw data file record
      (c) Create a program data vector to hold the current SAS observation
      (d) Create the descriptor portion of the output data set.

    - Execution
      (a) Read one observation into the input buffer
      (b) Put the input buffer line into PDV
      (c) Check whether the observation is the last observation. If yes, proceed next STEP (i.e. DATA STEP or a procedure step); otherwise continue the execution process.
      (d) Output the observation into SAS data set

    - A flow chart represents the whole data processing:

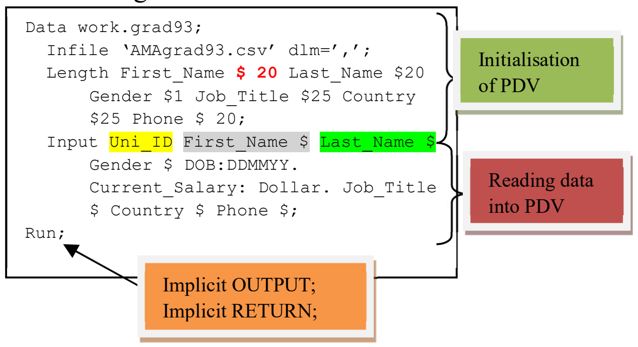

Compile
Initialize Variables in PDV to Missing
Execute Read Statement (i.e. SET/INPUT statement)
Check End of Input Data File?
Yes
No
Execute Other Statements
Output all values in PDV to SAS Data Set
Next Step (i.e. DATA or PROC Step)

- Students record example revisit:

Data

```
90010101,Peter,Smith,M,01/07/1965, …
89241253,Zhian,Yang,M,15/08/1964,$750000,…
91021201,Chi Wah,Chan,F,06/08/1973,$500000,…
…
```

SAS Program



```
Data work.grad93;
  Infile 'AMAgrad93.csv' dlm=',';
  Length First_Name $ 20 Last_Name $20
      Gender $1 Job_Title $25 Country
      $25 Phone $ 20;
  Input Uni_ID First_Name $ Last_Name $
      Gender $ DOB:DDMMYY.
      Current_Salary: Dollar. Job_Title
      $ Country $ Phone $;
Run;
```

- Compilation (given above)
- Execution

(a) Initialization step (First iteration)

Input Buffer Area:

1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7

PDV Area:

First_Name	Last_Name	Gender	Job_Title	Country	Phone	Uni_ID	DOB	Current_ Salary
$ 20	$ 20	$ 1	$ 25	$ 25	$ 20	N 8	N 8	N 8
						.	.	.

(b) Execution of INPUT statement (First iteration)

Input Buffer Area:

1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7
9	0	0	1	0	1	0	1	,	P	e	t	e	r	,	S	m	i	t	h	,	M	,	0	1	/	0

PDV Area:

First_Name	Last_Name	Gender	Job_Title	Country	Phone	Uni_ID	DOB	Current_ Salary
$ 20	$ 20	$ 1	$ 25	$ 25	$ 20	N 8	N 8	N 8
Peter	Smith	M				90010101	…	

(c) Output to SAS data set (First iteration)

SAS data set, Work.grad93:

First_Name	Last_Name	Gender	Job_Title	Country	Phone	Uni_ID	DOB	Current_ Salary
Peter	Smith	M				90010101	…	

(d) Initialization step (Second iteration), i.e. Reinitialization of PDV: Program starting from DATA statement up to the last statement before INPUT statement. PDV are cleared.

Input Buffer Area:

1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7
8	9	2	4	1	2	5	3	,	Z	h	i	a	n	,	Y	a	n	g	,	M	,	0	5	/	0	8

PDV Area:

First_Name	Last_Name	Gender	Job_Title	Country	Phone	Uni_ID	DOB	Current_ Salary
$ 20	$ 20	$ 1	$ 25	$ 25	$ 20	N 8	N 8	N 8
						.	.	.

(e) Execution of INPUT statement (Second iteration)

Input Buffer Area:

1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7
8	9	2	4	1	2	5	3	,	Z	h	i	a	n	,	Y	a	n	g	,	M	,	0	5	/	0	8

PDV Area:

First_Name	Last_Name	Gender	Job_Title	Country	Phone	Uni_ID	DOB	Current_ Salary
$ 20	$ 20	$ 1	$ 25	$ 25	$ 20	N 8	N 8	N 8
Zhian	Yang					89241253	.	.

(f) Output to SAS data set (Second iteration)

SAS data set: Work.grad93 (direct output from PDV)

First_Name	Last_Name	Gender	Job_Title	Country	Phone	Uni_ID	DOB	Current_ Salary
Peter	Smith	M				90010101	…	
Zhian	Yang	M				89241253	…	

(g) Continue the steps of (a) to (c) until End Of File (EOF)

- Processing of assignment statements:
  - Monthly salary is calculated via the formula: Current_Salary/12 and the graduate status is added.
  - Two assignment statements are added to the SAS program:

```
Data work.grad93;
  Infile ‘AMAgrad93.csv’ dlm=’,’;
  Length First_Name $ 20 Last_Name $20 Gender $1
         Job_Title $18 Country $25 Phone $ 20 Status $10;
  Input Uni_ID First_Name $ Last_Name $ Gender $ DOB:DDMMYY.
       Current_Salary: Dollar. Job_Title $ Country $ Phone $;
  Monthly_Salary=Current_Salary/12;
  Status=’Graduate’;
Run;
```

  - During compilation:
    Just before the line of Monthly Salary is compiled,
    PDV Area:

First_ Name	Last_ Name	Gender	Job_ Title	Country	Phone	Status	Uni_ID	DOB	Current_ Salary
$ 20	$ 20	$ 1	$ 18	$ 25	$ 20	$ 10	N 8	N 8	N 8

When the line of “Monthly_Salary” is compiled,
PDV Area:

First_ Name	Last_ Name	Gender	Job_ Title	Country	Phone	Status	Uni_ID	DOB	Current_ Salary	Monthly_ Salary
$ 20	$ 20	$ 1	$ 18	$ 25	$ 20	$ 10	N 8	N 8	N 8	N 8

When the line of “Status” is complied, no change is found in PDV, as “Status” variable is already found in PDV.

Missing values will be put to the PDV for variable defined in the assignment statement(s) during the re-initialization.

- The value will be calculated in SAS first and store in PDV before output to a SAS data set.

- Processing of KEEP and DROP statements:
  - Suppose we only need the university ID, full name, gender and the graduation year from the SAS data set ‘Grad93’. We output the required data into a temporary SAS data set, called, “part_grad93”.

```
Data work.part_grad93;
  Set grad93;
  DROP Current_Salary Monthly_Salary Job_Title DOB Country
        Phone Status;
  GraduateYr=1993;
Run;
```

- During compilation, ‘D’ is labeled to the variables to be dropped in the program.

First_ Name	Last_ Name	Gender	Job_ Title	Coun try	Phon e	Status	Uni_ID	DOB	Current_ Salary	Monthly_ Salary	Graudate Yr
			D	D	D	D		D	D	D	
$ 20	$ 20	$ 1	$ 18	$ 25	$ 20	$ 10	N 8	N 8	N 8	N 8	N 8

- During the execution, the reading and assignment processes are the same as discussed earlier.
- The only difference is the output to the SAS data set. See the first iteration.

PDV

First_ Name	Last_ Name	Gender	Job_ Title	Coun try	Phon e	Status	Uni_ID	DOB	Current_ Salary	Monthly_ Salary	Graudate Yr
			D	D	D	D		D	D	D	
$ 20	$ 20	$ 1	$ 18	$ 25	$ 20	$ 10	N 8	N 8	N 8	N 8	N 8
Peter	Smith	M				…	90010101	…			1993

Dropping variables

SAS data set, Work.part_grad93:

First_ Name	Last_ Name	Gender	Uni_ID	Graduate Yr
Peter	Smith	M	90010101	1993

- The blue highlighted line can be replaced by these two statements:

```
KEEP Uni_ID First_Name Last_Name Gender;
KEEP GraduateYr;
```

- The KEEP or DROP statements is applied from PDV to the output data set. Graphically,

The ‘keep/drop’ label line of the blue highlighted line is kept here for reference.

PDV

			D	D	D	D		D	D	D	
First_ Name	Last_ Name	Gender	Job_ Title	Country	Phone	Status	Uni_ID	DOB	Current_ Salary	Monthly _Salary	GraudateYr
K	K	K					K				K
$ 20	$ 20	$ 1	$ 18	$ 25	$ 20	$ 10	N 8	N 8	N 8	N 8	N 8
Peter	Smith	M				…	90010101	…			1993

Keeping or dropping variables

SAS data set, Work.part_grad93:

First_ Name	Last_ Name	Gender	Uni_ID	Graduate Yr
Peter	Smith	M	90010101	1993

- If the variable 'GraduateYr' is not listed in the KEEP statement, it will disappear in the output SAS data set.

DROP= and KEEP= options

- These options can be placed in the DATA statement and SET statement and the syntax is:

```
DATA output-SAS-data-set(DROP=variable-list);
```

```
DATA output-SAS-data-set(KEEP=variable-list);
```

```
SET input-SAS-data-set(DROP=variable-list);
```

```
SET input-SAS-data-set(KEEP=variable-list);
```

- The DROP= list specifies the variables to be dropped in the output-SAS-data-set in the DATA statement and in the input-SAS-data-set in the SET statement.
- The KEEP= list specifies the variables to be written to the output-SAS-data-set in the DATA statement and in the input-SAS-data-set in the SET statement.
- Example of DROP option in DATA statement:
  - SAS program

```
Data work.part_grad93(DROP=Current_Salary Monthly_Salary
Job_Title DOB Country Phone Status);
  Set grad93;
  GraduateYr=1993;
Run;
```

  - PDV compilation

(a) Compiling SET statement

First_ Name	Last_ Name	Gender	Job_ Title	Country	Phone	Status	Uni_ ID	DOB	Current_ Salary	Monthly_ Salary
$ 20	$ 20	$ 1	$ 18	$ 25	$ 20	$ 10	N 8	N 8	N 8	N 8

(b) Compiling GraduateYr=1993 statement

First_ Name	Last_ Name	Gender	Job_ Title	Country	Phone	Status	Uni_ ID	DOB	Current_ Salary	Monthly_ Salary	GraudateYr
$ 20	$ 20	$ 1	$ 18	$ 25	$ 20	$ 10	N 8	N 8	N 8	N 8	N 8

(c) Dropping the variable at the point of implicit output

First_ Name	Last_ Name	Gender	Job_ Title	Country	Phone	Status	Uni_ ID	DOB	Current_ Salary	Monthly_ Salary	GraudateYr
			D	D	D	D		D	D	D	
$ 20	$ 20	$ 1	$ 18	$ 25	$ 20	$ 10	N 8	N 8	N 8	N 8	N 8

dropping variables for output

SAS data set, Work.part_grad93:

First_ Name	Last_ Name	Gender	Uni_ID	Graduate Yr

- Example of DROP option in SET statement:
  - SAS program

```
Data work.part_grad93;
  Set grad93(DROP=Current_Salary Monthly_Salary Job_Title DOB
Country Phone Status);
  GraduateYr=1993;
Run;
```

  - PDV compilation diagram:
    (a) Compiling SET statement

First_ Name	Last_ Name	Gender	Uni_ID
$ 20	$ 20	$ 1	N 8

(b) Compiling GraduateYr=1993 statement

First_ Name	Last_ Name	Gender	Uni_ID	GraudateYr
$ 20	$ 20	$ 1	N 8	N 8

(c) Implicit output

SAS data set, Work.part_grad93:

First_ Name	Last_ Name	Gender	Uni_ID	Graduate Yr

3.8 Summary

Reading data into SAS	- Source: • SAS data sets • EXCEL files • csv files • other text files • in-stream data using DATALINES - Techniques: • INPUT statement/multiple INPUT statements • Use of '&' for embedded blanks • Use of line pointer controls: '@', '@@' • Use of position ranges
EXCEL data files	- PROC CONTENTS - Import/export wizard
Program compilation and execution	- Processing of Program Data Vector