

SAS Programming Fundamentals I

DCS/TASC/Advanced Support Team
Center for Information Technology
National Institutes of Health

Summer 2001

SAS Programming Fundamentals I

Many software applications are either totally menu driven, or totally command driven (“enter a command -see the result”). Base SAS software is neither totally menu driven or totally command driven. With Base SAS software, you use statements to write a series of instructions called a SAS program

This course introduces Base SAS software **programming** concepts. It provides those with expertise in their field but no programming experience, a programming foundation. The focus is on creating SAS data sets from raw data files (also referred to as text, ASCII, sequential, or flat files). Your raw data may be either internal to your SAS program, or in a separate file. Either way, you must tell SAS where to find and how to read your data.

MODULE 1: Getting started using SAS System Software

- SAS program objectives
- SAS programming process
- Two basic components of a SAS program
- SAS statements
- Naming conventions

MODULE 2: Where to find and how to read raw data

- Locate Data Lines—DATALINES, INFILE, FILENAME statements
- Identify variable attributes—numeric or character variable
- Read data lines—INPUT statement styles
- Read dates with SAS defined informats
- Write variables with SAS defined formats

MODULE 3: Temporary versus permanent SAS data sets

- DATA step logic
- PROC step logic
- Interpret messages in the SAS log
- Create temporary SAS data set
- Create permanent SAS data set
- Examine Proc Contents procedure

MODULE 1: Getting started using SAS System Software

- SAS program objectives
- SAS programming process
- Two basic components of a SAS program
- SAS statements
- Naming conventions

1. SAS program objectives

(1) What do you want to be able to draw conclusions about?

(2) What is the desired output?

- type of analysis desired
- type of summary desired
- type of report desired

(3) Know your data, that is, what data is needed to solve the program objectives?

If you are reading from a raw data file (also referred to as text, ASCII, sequential, or flat files)

- **Examine the record layout—know the arrangement of the data values**

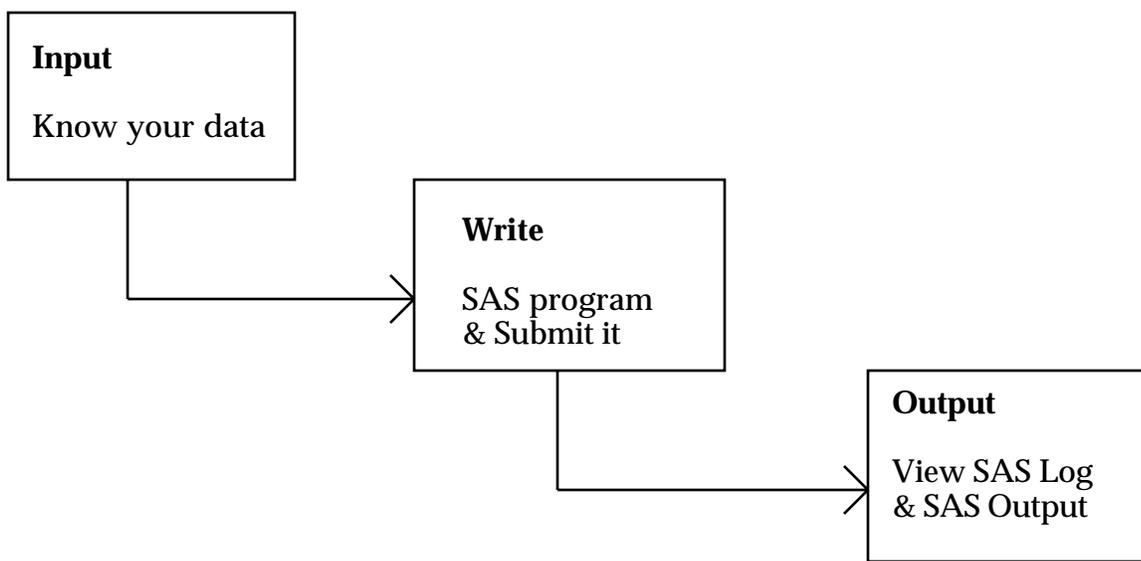
We must always be sure that we understand our data. If we do not, then it is essential that we ask for clarification. Even the best analytical tool in the world can only produce rubbish if that is what we give it. So, before we can start to read raw data, we should view it to confirm that it looks as we expect.

- **Prepare good test data.**
- **Edit the data lines**
- **Identify missing value(s)**

(4) Specify the variable name(s) and attribute and know what the variable values represent.

(5) Begin the programming process:

- Write the SAS program
- Test the SAS program with small and representative data



(For the next few pages don't worry about understanding all the details, definitions or syntax. Definitions and syntax rules will be explained as we move along)

Objective: We would like to obtain a SAS data set sorted by the name.

Given: data lines

Please note:

The words 'column ruler' and 'data line 1', 'data line 2', etc...are shown to illustrate concepts discussed during the class presentation. These words would NEVER be included in the actual data.

```

column          1          2
ruler          1234567890123456789012345

data line 1    PAUL      M 27 72 140
data line 2    JENNIFER  F 28 64 135
data line 3    RENEE     F 35 54 128
data line 4    MANUEL   M 35 60 125
data line 5    TONY     M 32 68 130
  
```

Data layout - a guide to identify the field locations for the raw data ('data layout' is sometimes called the 'field dictionary')

<u>Column name</u>	<u>Column location</u>	<u>Type or Attribute</u>	<u>Description</u>
FNAME	1-8	Character	Individuals' first name
SEX	11	Character	M=Male, F=Female
AGE	13-14	Numeric	Age range: 27 to 35
HT	16-17	Numeric	Height in inches
WT	20-22	Numeric	Weight in pounds

2. SAS Processing

SAS processing is the way the SAS language reads and transforms input data and generates the kind of output that you want.

A SAS program is a series of SAS statements executed in order. A SAS program is built on **two** basic components:

DATA step
PROC step

A SAS program may break-down into any number of **DATA** steps or **PROC** steps. It may be viewed as a string of steps in any order. Just as you can stack building blocks in any order, you can arrange **DATA** and **PROC** steps in any order.

DATA and **PROC** steps are made up of SAS statements. A step may have as few as one or as many as hundreds of statement. Each statement gives information or instructions to SAS.

Most statements work in only one type of step - in **DATA** steps but not **PROC** steps, or vice versa.

Generally, the **DATA** step manipulates data, and the **PROC** step analyzes, produces reports,

Data step

```
DATA Display_Sample;  
  INPUT  FNAME $ 1-8 SEX $ 11 AGE 13-14  
        HT 16-17  WT 20-22;  
DATALINES;  
PAUL    M 27 72  140  
JENNIFER F 28 64  135  
RENEE   F 35 54  128  
MANUEL  M 35 60  125  
TONY    M 32 68  130  
;
```

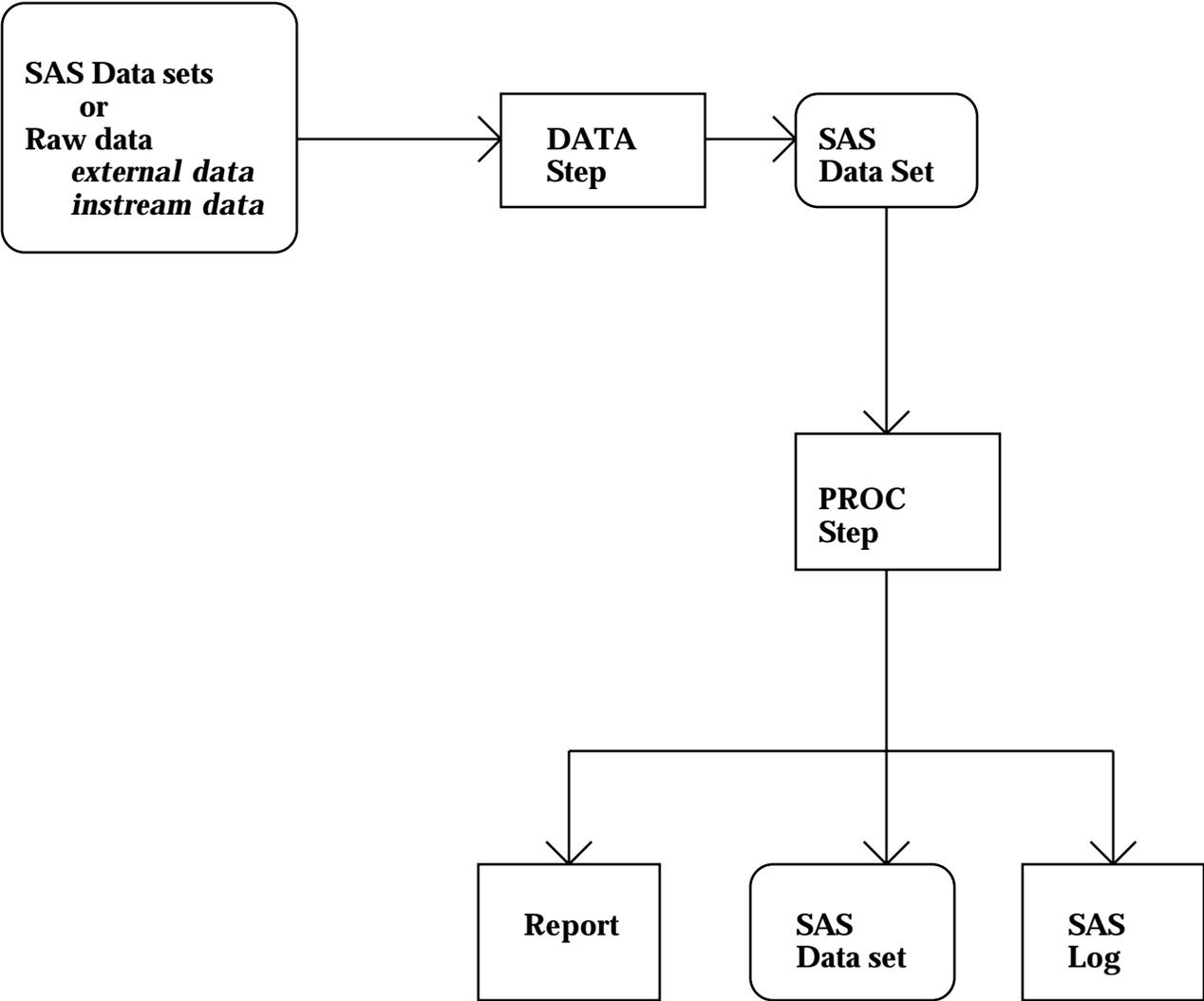
Proc step

```
PROC SORT;  
  BY FNAME;
```

Proc step

```
PROC PRINT;  
  TITLE 'Display of SAS data set - Display_Sample';  
RUN;
```

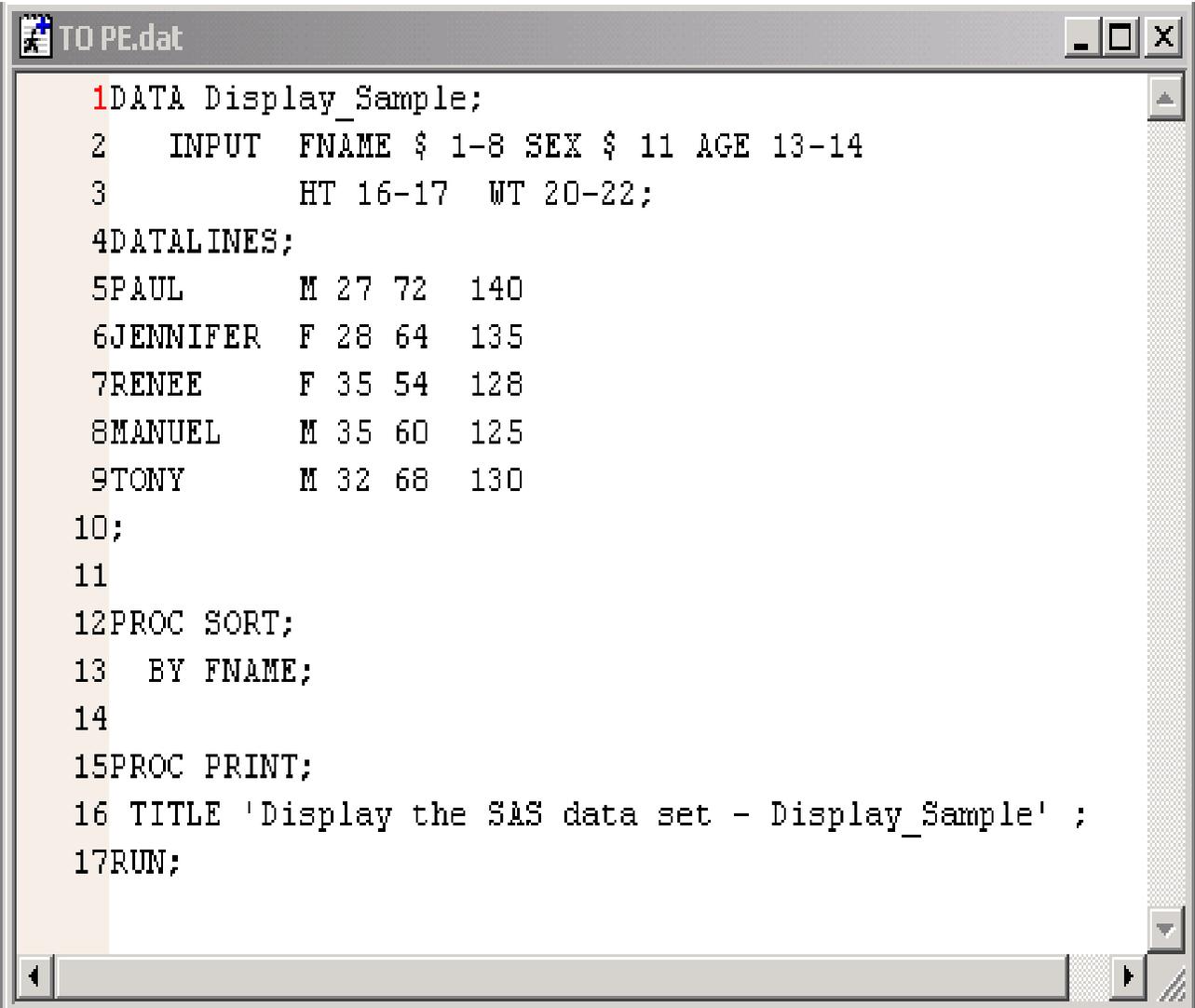
SAS Processing



2.1 SAS Enhanced Editor, SAS Log, and SAS Output windows

SAS Enhanced Editor window

In the Windows operating environment, an **Enhanced Editor** window opens by default. Like the Program Editor (e.g. SAS Version 6), you can use the **Enhanced Editor** window to enter, edit, and submit SAS programs. You can open multiple **Enhanced Editor** windows. To display an additional **Enhanced Editor** window, go to **View** and select **Enhanced Editor**.

A screenshot of the SAS Enhanced Editor window. The window title bar reads "TO PE.dat" and includes standard Windows window controls (minimize, maximize, close). The editor area contains the following SAS code:

```
1 DATA Display_Sample;
2   INPUT  FNAME $ 1-8 SEX $ 11 AGE 13-14
3         HT 16-17  WT 20-22;
4 DATALINES;
5 PAUL      M 27 72  140
6 JENNIFER  F 28 64  135
7 RENEE     F 35 54  128
8 MANUEL    M 35 60  125
9 TONY      M 32 68  130
10;
11
12 PROC SORT;
13   BY FNAME;
14
15 PROC PRINT;
16 TITLE 'Display the SAS data set - Display_Sample' ;
17 RUN;
```

The code defines a data set named 'Display_Sample' with variables for name (FNAME), sex (SEX), age (AGE), height (HT), and weight (WT). It lists five individuals: PAUL (M, 27, 72, 140), JENNIFER (F, 28, 64, 135), RENEE (F, 35, 54, 128), MANUEL (M, 35, 60, 125), and TONY (M, 32, 68, 130). The data is then sorted by name and printed.

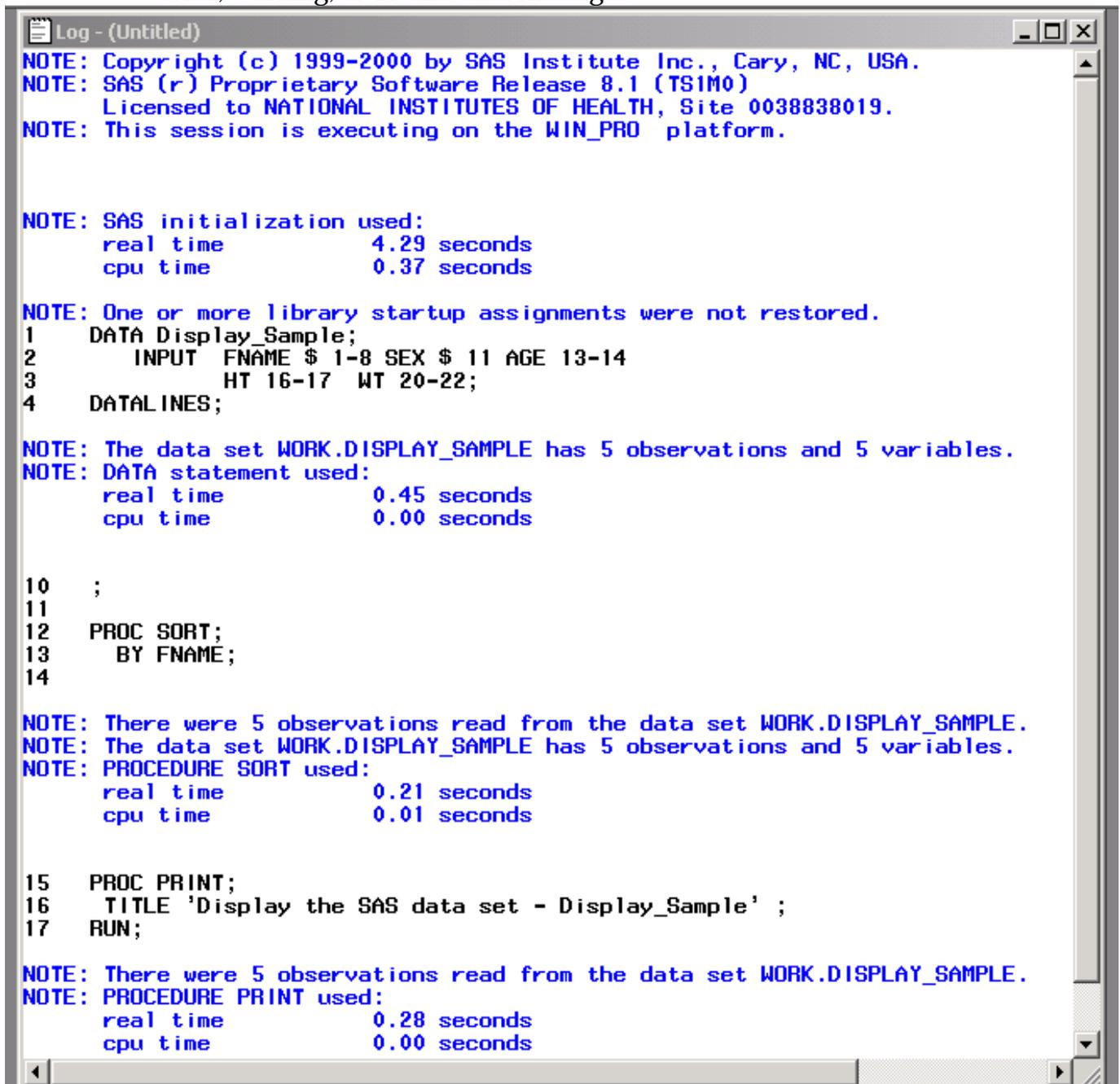
2.2 SAS Log window

The Log window displays messages about your SAS session and any SAS programs you submit.

After you submit a SAS program, any notes, errors, or warnings associated with your program as well as the program statements will appear in the Log window. These messages can help you debug a SAS program.

ALWAYS check the SAS Log window.

Submitting the SAS program does not mean your SAS program executed without error, warning, or invalid data messages.



```
Log - (Untitled)
NOTE: Copyright (c) 1999-2000 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) Proprietary Software Release 8.1 (TS1M0)
      Licensed to NATIONAL INSTITUTES OF HEALTH, Site 0038838019.
NOTE: This session is executing on the WIN_PRO platform.

NOTE: SAS initialization used:
      real time          4.29 seconds
      cpu time           0.37 seconds

NOTE: One or more library startup assignments were not restored.
1  DATA Display_Sample;
2      INPUT  FNAME $ 1-8 SEX $ 11 AGE 13-14
3          HT 16-17  WT 20-22;
4  DATALINES;

NOTE: The data set WORK.DISPLAY_SAMPLE has 5 observations and 5 variables.
NOTE: DATA statement used:
      real time          0.45 seconds
      cpu time           0.00 seconds

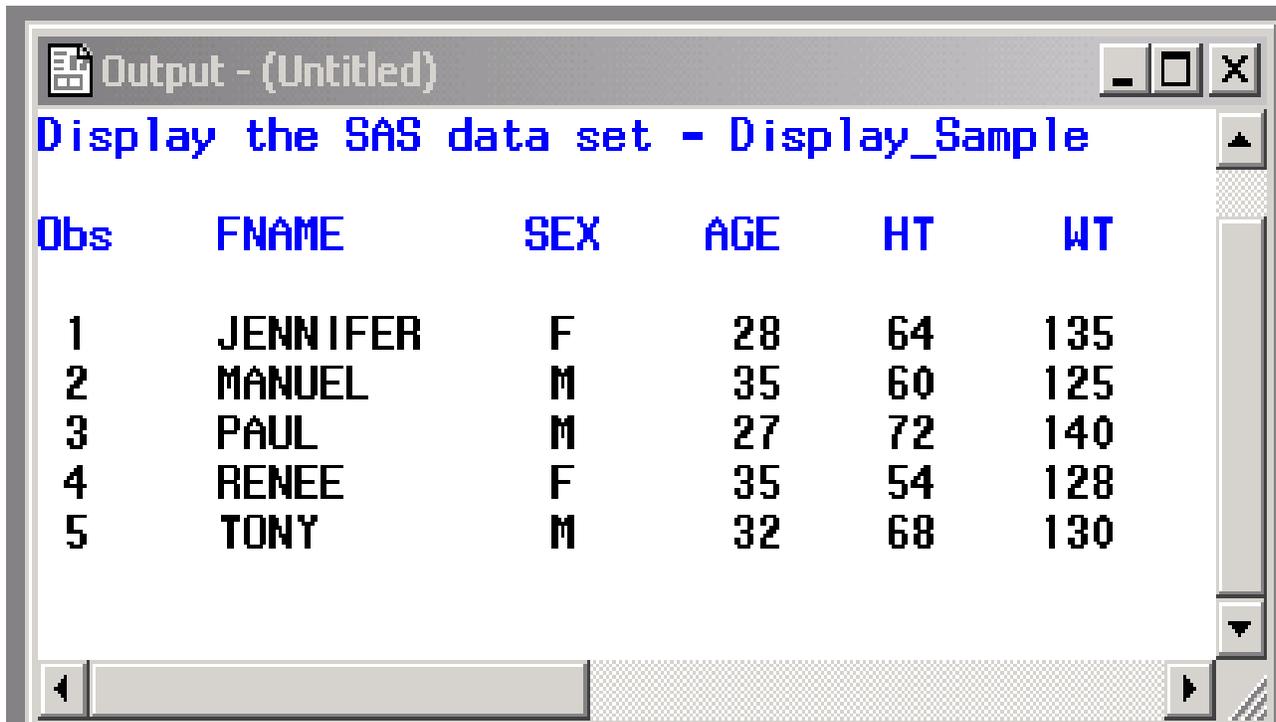
10 ;
11
12 PROC SORT;
13     BY FNAME;
14

NOTE: There were 5 observations read from the data set WORK.DISPLAY_SAMPLE.
NOTE: The data set WORK.DISPLAY_SAMPLE has 5 observations and 5 variables.
NOTE: PROCEDURE SORT used:
      real time          0.21 seconds
      cpu time           0.01 seconds

15 PROC PRINT;
16     TITLE 'Display the SAS data set - Display_Sample' ;
17 RUN;

NOTE: There were 5 observations read from the data set WORK.DISPLAY_SAMPLE.
NOTE: PROCEDURE PRINT used:
      real time          0.28 seconds
      cpu time           0.00 seconds
```

SAS Output window: SAS output generated by the PROC(s)



The screenshot shows a window titled "Output - (Untitled)" with a blue header "Display the SAS data set - Display_Sample". The data is presented in a table with columns labeled "Obs", "FNAME", "SEX", "AGE", "HT", and "WT". The rows contain the following data:

Obs	FNAME	SEX	AGE	HT	WT
1	JENNIFER	F	28	64	135
2	MANUEL	M	35	60	125
3	PAUL	M	27	72	140
4	RENEE	F	35	54	128
5	TONY	M	32	68	130

3. DATA step

A DATA step consists of a group of statements in the SAS language that reads raw data, or SAS data sets to create SAS data set(s). Once your data is accessible as a SAS data set, you can analyze the data and write reports by using any SAS **PROC**edure.

3.1 DATA step A DATA step in a program begins with a DATA statement and ends with a DATALINES statement, a RUN statement, another DATA statement, or another PROC statement.

A typical program starts with a DATA step to create a SAS data set and then passes the SAS data set to a **PROC** step for processing.

A sample DATA Step is entered in the Enhanced Editor window:

```
DATA Display_Sample;
  INPUT FNAME $ 1-8 SEX $ 11 AGE 13-14
        HT 16-17 WT 20-22;
DATALINES;
PAUL      M 27 72 140
JENNIFER  F 28 64 135
RENEE     F 35 54 128
MANUEL    M 35 60 125
TONY      M 32 68 130
;
```

3.1.1 SAS data set

When you work with SAS, you use files that are created and maintained by SAS, as well as files that are created and maintained by your operating environment, and that are not related to SAS. Files with formats or structures known to SAS are referred to as **SAS files**.

All SAS files reside in a **SAS data library**.

The most commonly used SAS file is a **SAS data set**.

A **SAS data set** is a group of data values that SAS creates and processes. A SAS data set contains

- **descriptor information** which describes the contents of the SAS data set
- a **table** with observations and variables
 - **observations** are organized in rows
An observation is a collection of data values that is associated with an entity such as a patient in a medical clinic
 - **variables** are organized in columns
Variables are characteristics of these entities, such as patient admission date, etc... When data values are incomplete, SAS uses missing values.

The following sample output is generated from the PROC PRINT procedure. It displays a table (e.g. a SAS data set).

OBS	FNAME	SEX	AGE	HT	WT
1	JENNIFER	F	28	64	135
2	MANUEL	M	35	60	125
3	PAUL	M	27	72	140
4	RENEE	F	35	54	128
5	TONY	M	32	68	130

Operating environment note:

OS/390 mainframe:

There is no limit to the number of variables that can be stored in a SAS data set.

Windows:

The maximum number of variables in a single SAS data set under Windows is 32,767. In addition, an observation under Windows cannot be longer than 5M. Therefore if you want your data set to contain 32,767 character variables, the longest each variable can be is 160 bytes.

However, a DATA step can reference more than 32,767 variables, if you write only 32,767 or fewer variables to the SAS data set. For example, you could drop some variables with a DROP= SAS data set option. The maximum number of variables a DATA step can reference is 2,147,483,647.

3.1.2 Descriptor portion of the SAS data set

The descriptor portion of a SAS data set is automatically created by SAS. The descriptor information for a SAS data set makes the data set self-documenting, that is, each data set can supply the attributes of the data and of its variables. Once the data is in the form of a SAS data set, you do not specify the attributes of the data or the variables in your program statements. SAS obtains the information directly from the data set.

Descriptor information includes...

- Data set name
- Date created
- Number of observations, observation length
- Number of variables

and for each variable the descriptor information includes...

- Type(numeric or character)
- Length (storage size in bytes)
- Name
- Number
- Label(if any)
- Informat for input(if any)
- Format for output(if any)

The PROC CONTENTS procedure lists the names of SAS data sets in a SAS library and other information from the directory of a SAS data set.

Sample program:

```
PROC CONTENTS DATA = Display_Sample ;  
  Title 'Displays descriptor portion of SAS data set - Display_Sample';  
RUN;
```

The output of the PROC CONTENTS procedure displays the descriptor portion of the SAS data set Display_Sample

...shown on next page...

3.1.3 Output from the PROC CONTENTS procedure

Displays **descriptor portion** of SAS data set Display_Sample

The CONTENTS Procedure

Data Set Name:	WORK.DISPLAY_SAMPLE	Observations:	5
Member Type:	DATA	Variables:	5
Engine:	V8	Indexes:	0
Created:	11:07 Thursday, May 11, 2000	Observation Length:	40
Last Modified:	11:07 Thursday, May 11, 2000	Deleted Observations:	0
Protection:		Compressed:	NO
Data Set Type:		Sorted:	NO
Label:			

-----Engine/Host Dependent Information-----

Data Set Page Size:	4096
Number of Data Set Pages:	1
First Data Page:	1
Max Obs per Page:	101
Obs in First Data Page:	5
Number of Data Set Repairs:	0
File Name:	C:\temp\SAS Temporary Files_TD193\display_sample.sas7bdat
Release Created:	8.0000M0
Host Created:	WIN_NT

-----Alphabetic List of Variables and Attributes-----

#	Variable	Type	Len	Pos
3	AGE	Num	8	0
1	FNAME	Char	8	24
4	HT	Num	8	8
2	SEX	Char	1	32
5	WT	Num	8	16

3.2 Proc step

Definition: A group of SAS statements beginning with a PROC statement that invoke a SAS procedure with a SAS data set as input.

The SAS Software System contains a collection of procedures (PROCs). They perform specific tasks on SAS data sets.

For instance...

(1) PROC SORT - to re-arrange the observations by any variable(s)

```
PROC SORT;  
  BY variable-list;
```

(2) PROC PRINT - to display the SAS data set

The PROC PRINT procedure is perhaps the most widely used SAS procedure. You will see it used frequently in these classnotes to display a SAS data set. In its simplest form, PROC PRINT(with no options) displays all variables for all observations.

```
PROC PRINT;
```

(3) PROC MEANS - to calculate descriptive statistics

One of the first things people usually want to do with their data, after reading it, is look at some simple statistics. Statistics such as the mean, standard deviation, and minimum and maximum values give you a feel for your data. In its simplest form, PROC MEANS(with no options) displays the number of non-missing values, the means, the standard deviation, the minimum and maximum values for each numeric variable.

```
PROC MEAN;
```

(4) PROC FREQ - to generate a simple list of counts

The most obvious reason for using PROC FREQ is to create tables showing the distribution of categorical data values, but PROC FREQ can also reveal irregularity in your data. For instance, data entry errors are often glaringly obvious in a frequency table.

In its simplest form... **PROC FREQ ;**
TABLES variable-list ;
...produces counts for each variable in the TABLE statement.

Beside the Base SAS procedures each of the SAS products or modules in the SAS Software System has an large number of PROC (s) or Procedures. Each PROC is designed to perform specific tasks.

Exercise 1

Check the appropriate selection

- (1) Before you begin to write a SAS program you must
 - a. know what you want to be able to conclude
 - b. know your data
 - c. prepare a good test data set
 - d. all of the above
- (2) A data value is
 - a. a line of data
 - b. a unit of information
 - c. a collection of data lines
- (3) An observation is
 - a. the name associated with a column
 - b. a row in a SAS data set that contains the specific data values for an individual entity
- (4) The function of a DATA step is to
 - a. describe the data to SAS
 - b. get the data in shape for processing
 - c. both a and b
- (5) How many types of SAS steps are there?
 - a. one
 - b. two
 - c. three
 - d. none of the above
- (6) The function of the PROC step is to
 - a. subset your data
 - b. run your program
 - c. call and execute a procedure
- (7) The function of the DATA statement is to
 - a. begin a DATA step
 - b. name the SAS data set being created
 - c. both a and b
- (8) To display observations and variables from a SAS data set you would invoke
 - a. PROC SORT
 - b. PROC MEANS
 - c. PROC PRINT

4.1 SAS Statements

A SAS statement is a series of items that may include keywords, SAS names, special characters, and operators. All SAS statements end with a semicolon. A SAS statement either requests SAS to perform an operation or gives information to the system.

Most SAS statements begin with a special word identified as a “**keyword**”. The keyword communicates the purpose of the statement to the SAS system.

Sample SAS statements

Keyword

Purpose

DATA sasdsname;

Tell SAS to begin a DATA step

DATALINES;

Indicate the raw data lines are entered in the program, that is, following the DATALINES statement. The term “instream” is used to define this situation. The DATALINES statement is more frequently used to **test** a SAS program with a small data sample.

INFILE fileref ;

Identifies an external file to read with an INPUT statement.

INPUT variable-list ;

Describes the arrangement of values in the input data record and assigns input values to the corresponding SAS variables

(...continued on next page)

Keyword

Purpose

IF condition **THEN** action;
 ELSE IF condition **THEN** action;
 ELSE IF condition **THEN** action;
 ELSE action ;

Allow SAS to conditionally execute statements.

IF expression ;
include in the data set

SUBSETTING IF statement...selects observation to

IF expression then **DELETE** ;

DELETE statement...selects observations to exclude

PROC procedure name ;

Tell SAS to begin a Proc step.

TITLEn 'text' ;

Place title at the top of every page of the SAS output.

LABEL variable='text' ;

Assign descriptive text to variable(s).

(Long label names (256) in SAS version 8)

FORMAT variable formatw. ;

Associate format with variable

RETAIN variables initial-value ;

To initial variable and retain its value from one iteration to another

BY variable-list ;

a **BY** statement required in the PROC SORT procedure step

WHERE condition ;

When used in a PROC step, observations that satisfy the condition are selected by the PROC

4.1.1 Syntax rules

Many software applications are either menu driven, or command driven (“enter a command-see the result”). SAS is neither. With SAS, you use statements to write a series of instructions called a SAS program.

The following SAS “grammar” rules control all SAS statements in SAS programming:

- (1) most SAS statements begin with a keyword
(in Base SAS exceptions are the SUM statement and the ASSIGNMENT statement)
- (2) signal the end of each SAS statement with a semicolon
- (3) upper- or lowercase, or a mixture of the two
- (4) free-format
 - Can begin SAS statements in any column of a line and write several statements on the same line

```
Proc sort ; BY fname ;
```
 - Can begin a statement on one line and continue it on another line, but you cannot split a word between two lines

```
INPUT FNAME $ 1-8 SEX $ 11  
      AGE 13-14 HT 16-17  
      WT 20-22;
```
 - SAS “words” can be separated with as many blanks as you want
Why?
A blank is not treated as a character in a SAS statement unless it is enclosed in quotation marks(e.g. a literal).

```
DATA Display_Sample  
  ; INPUT      FNAME $ 1-8      SEX $ 11  
      AGE 13-14 HT 16-17      WT 20-22;
```

4.2 Good programming style suggestions

It is recommended that you have a neat looking program with each statement on a line by itself and indentions to show the various parts of the program.

- (1) Begin DATA and PROC statements in column 1 and indent other statements.

```
DATA Display_Sample;  
  INFILE _____;  
  INPUT _____;
```

- (2) Use one line per SAS statement when possible.

- (3) Indent continuation of a SAS statement. This is a simple way to make your programs more readable, and it's a good habit to form. Programs should be formatted in a way which makes them easy to read and understand. Easy-to-read programs are time-savers for you, or the technical consultant.

```
INFILE _____;  
  INPUT FNAME $ 1-8 SEX $ 11  
        AGE 13-14 HT 16-17  
        WT 20-22;
```

(...continued on next page)

- (4) Use blank lines to separate major components of the SAS program.

PROC FORMAT;

VALUE \$SEXFMT 'M' = 'MALE' 'F' = 'FEMALE';

DATA NOT_AN_ACTUAL_SAS_PROGRAM ;

INPUT list variables and attributes ;

/* enter other SAS statements in the DATA step */

/* if you use 'instream' data, then a DATALINES statement is the very last statement in the DATA step */

DATALINES;

(enter data lines)

;

PROC PRINT;

run;

- (5) Comments are usually used to annotate the program, making it easy for someone to read your program and understand what you have done and why.
Two styles of comments you can use: (a) start with an asterisk (*) and ends with a semicolon, or (b) start with slash asterisk (/*) and ends with an asterisk slash (*/).

5. Naming conventions

SAS names that have increased to a maximum of 32 character are the following:

Columns or data set variables

SAS table(s) or SAS data set name(s)

Catalogs

Macro variables

Macros

Arrays

SAS names that have a maximum of 8 character are referred to as...

- libref

for instance... LIBNAME libref c:\pathname” ;

- fileref

for instance... FILENAME fileref c:\pathname” ;

- format-names for user-defined or user-written formats

(user-defined or user-written format used in the VALUE statement of the PROC FORMAT procedure)

SAS name that has a maximum of 7 character is referred to as...

- informat-name for user-defined or user-written informat

(user-defined or user-written informat used in the INVALUE statement of the PROC FORMAT procedure)

Technical speaking, each word in the SAS language belongs to one of four categories

- a SAS name ... for instance, data , _new , yearcutoff , descending
- a SAS literal ... for instance, ‘Chicago’ , ‘1234’ , ‘1998-00’
- numbers ... for instance, 5683 , 2.35 , -5 , ‘24may98’d
- special characters ... for instance, this is any single keyboard character other than letters, numbers, the underscore, and the blank

5.1 SAS table name(e.g. SAS data set name)

- are from 1 to 32 characters in length
- SAS data set name is not stored in mixed case. You can enter upper or lowercase letters. However, SAS processes names as uppercase regardless of how you type them.
- can be specified with 'pathname'

for instance....

```
PROC PRINT DATA = "c:\SAS Crs212demo\crash" ;  
    ...in the pathname you do not need to enter the .sas7bdat extension associated with  
    the two-level name of the SAS data set crash.sas7bdat
```

or

You may continue to use the LIBNAME statement as experienced SAS users may be accustomed to do.

```
LIBNAME Crs212 "c:\SAS Crs212demo" ;
```

```
PROC PRINT DATA = Crs212.crash ;  
RUN;
```

- must start with a letter (A through Z) or an underscore (_)
subsequent characters can be letters, numeric digits (0, 1, ...9), or underscores (_)
- **Must NOT** contain blanks or any other special character such as %\$!*&#@;,.
(Exception... In *filerefs* only, you can use the dollar sign(\$), the pound sign(#),
and the at sign(e.g. @)
- do not use the following names when you create SAS data sets:
NULL, _DATA_, _LAST_

5.1.2 Column name or variable name

- are from 1 to 32 characters in length
- must start with a letter (A through Z) or an underscore (_)
- can contain letters, numerals, or underscores (_)
- **Must NOT** contain blanks or any other special character such as %\$!*&#@;,.
- can be stored in mixed case

The mixed case is remembered and used for presentation purposes only. (SAS stores the case used in the first reference to a variable.) When SAS processes variable names, however, it internally uppercases them. You cannot, therefore, use the same letters with different combinations of lower- and- uppercase to represent **different** variables. For example, cat, Cat, and CAT all represent the **same** variable.

When a column name is created, the first usage of the column name determines its capitalization pattern for reports and displays. Any subsequent reference to the column name, whether in a procedure or a DATA step, is case-insensitive.

- SAS reserves a few names for automatic variables and variable lists. When creating variables, do not use the names of special SAS automatic variables (for example, `_N_` and `_ERROR_`).

5.1.3 Numbered range list

It is usually wise to choose variable names that help you remember which name goes with which variable.

There are times where you may have a very large number of variables to list. In such situations you may opt for the abbreviated list.

- **Numbered range lists** require you to have a series of variables with the same name, except for the last character or characters, which are consecutive numbers. The numbers can start and end anywhere as long as the number sequence between is complete.

For instance...

```
INPUT var3 var4 var5 var6 var7;      or      INPUT var3 - var7;
```

- **Name range lists** Name range lists depend on the internal order, or position, of the variables in the Program Data Vector(PDV).

For instance...

```
INPUT x a c h b;
```

In a KEEP statement the following abbreviated form is accepted

```
KEEP x -- b;
```

```
/* that is, all variables ordered from variable x through variable b */
```

Exercise 2

- (1) Place the word valid or invalid in the blank space corresponding to each SAS name.

SAS Name	VALID or INVALID?
(1) Y_1996	_____
(2) _STATE	_____
(3) STR - CODE	_____
(4) ZIP CODE	_____
(5) 1991	_____
(6) END_DATE	_____
(7) 2-Seats	_____
(8) DEPARTMENT	_____
(9) DEPT	_____
(10) ZIPCODE	_____

- (2) Circle syntax errors in SAS statements and explain your answer.

- (a) DATA SYN-TAX1;
 INPUT CITY \$ 1-6 DAYS 15
 DATALINES;
 PARIS 8
 LONDON 6
 ROME 3
 ;

 PROC PRINT
 RUN ;
- (b) DATA TEST;
 INPT NAME \$ 1-12 SCORES 14-16 ;
 DATALINES;
 JAMES 85
 JOHN 96
 MARY 100
 ;

 PROC PRINT;
 RUN;

(...continued on next page)

2(c) Names of the SAS data sets created in (a) and (b)?
Assumes you already made necessary corrections in (2a) and 2b)

2(d) How many SAS statements does the SAS System recognize in (b)?
Assumes you already made necessary corrections in (2b)

(3) The following SAS program represents poor format and programming style.

```
DATA PoorFormat;  
INPUT TEAM $ 1-3 SCORE 5-6 COACH $ 10-15  
      ;IF SCORE = 98;  
DATALINES;  
REG 91 COX  
GLM 98 SMITH  
;  
      PROC PRINT;  
VAR TEAM;
```

Rewrite the SAS program using the recommended suggestions for good programming style.

MODULE 2: Where to find and how to read raw data

- Locate Data Lines—`DATALINES`, `INFILE`, `FILENAME` statements
- Identify variable attributes—numeric or character variable
- Read data lines—`INPUT` statement styles
- Read dates with SAS defined informats
- Write variables with SAS defined formats

6. Methods: Getting data into the SAS System

SAS System software provides a number of ways to bring a variety of external data formats into SAS.

- **Reading raw data files, text, ASCII, sequential, or flat files through the DATA step**

*In terms of getting raw data into SAS this course will only focus on using the functions of the INPUT statement in the Data step to read **raw data**.*

*The raw data file is often referred to as a **text file**, an **ASCII file**, a **sequential file**, or a **flat file**.*

- **Point-and-Click data entry methods...**

- 1) **Microsoft Excel** and other standard data sources are accessed through the **SAS Import Wizard Facility**. The SAS Import Wizard is most frequently used.

'Fire-up' SAS >>> File >>> Import Data >>> follow instructions in the menu-driven Import Wizard

Standard data sources imported in SAS on windows environment:

- (a) Excel2000 data
- (b) Excel5 data
- (c) Excel4 data
- (d) ACCESS 2000 data
- (e) ACCESS 97 data
- (f) dBase (*.dbf) data
- (g) Lotus (.wk1), (.wk3), (.wk4) data
- (h) DLM(Delimited data)
- (i) Comma Separated Values(*.csv) data
- (j) Tab Delimited (.txt) data

Standard data sources imported in SAS 6.12 for the Mac:

- (a) DLM(Delimited file (*.*))
- (b) Comma Separated Values (*.csv)
- (c) Tab Delimited File (*.txt)

- 2) For **SAS/INSIGHT end-users** - may use the features within the SAS/INSIGHT product to enter new data for analysis

- **Converting other software data files into SAS data sets**

The increasing use of the personal computer in everyday life for work means that more and more data is already stored on computer media by a wide variety of different applications. It's important to be able to use this external data. SAS software provides a number of ways to bring a variety of external data formats into analysis

- . a) the Import/Export Wizard

- or

- b) SAS/ACCESS software for PC File Formats(ODBC)

- **Using SPSS , BMDP , XPORT engines**

6.1 DATALINES statement

Purpose: Use the DATALINES statement with an INPUT statement to read data that you enter directly in the program(in-stream data), rather than data stored in an external file.

DATALINES statement: DATALINES ;

Guidelines:

- (1) Must be the last statement in the DATA step (that is, place the DATALINES statement directly before the first data line.) When the compiler comes across the statement DATALINES; then it reads subsequent lines as data rather than source code.
- (2) Data lines cannot exceed 256 columns in the Windows environment.
Data lines cannot exceed 80 columns in the mainframe(OS/390) environment.
- (3) Terminate the data with a lone semicolon. This lone semicolon (that is, the NULL statement) is entered on one line.

SAS program entered in the Enhanced Editor window:

```
DATA Display_Sample ;  
  INPUT  FNAME $ 1-8 SEX $ 11 AGE 13-14 HT 16-17 WT 20-22 ;  
DATALINES ;  
PAUL      M 27 72  140  
JENNIFER  F 28 64  135  
RENEE     F 35 54  128  
MANUEL    M 35 60  125  
TONY      M 32 68  130  
;
```

6.2 INFILE Statement

Purpose: Identifies an external(raw data) file.

An INFILE statement is used to specify the source of data read by the INPUT statement. An INFILE statement is required in any data step which uses an INPUT statement. However, the INFILE statement is not needed when the data is entered 'in-stream' following the DATALINES statement.

An INFILE statement identifies the file to be read, it must execute before the INPUT statement and therefore must appear before the INPUT statement.

Sample INFILE statement in Windows

```
DATA EXTERNAL;  
  INFILE 'c:\sasclass\project1.dat' ;  
  INPUT FNAME $ 1-8 SEX $ 11 AGE 13-14 HT 16-17 WT 20-22 ;
```

Sample INFILE statement in OS/390 (mainframe)

```
DATA EXTERNAL ;  
  INFILE 'wxyzabc.rawdata1' ;  
  INPUT FNAME $ 1-8 SEX $ 11 AGE 13-14 HT 16-17 WT 20-22 ;
```


7. Identify variable attributes

Before we can begin to write an INPUT statement to read the data from each record, we need to assign to each variable...

- a variable name
- a character or numeric attribute

7.1 Numeric variable attributes

Standard numeric data contains...

- numbers with or without decimal point
- numbers written in scientific notation (example, 1.2E3)
- numbers with plus sign(+) or a minus sign (-)

Non-standard numeric data contains...

- dates
- numbers with embedded commas
- numbers with dollar sign
- numbers with percent sign

NOTE:

Variables should be defined as numeric when you expect to perform mathematical computations on the values.

7.2 Character variable attributes

(1) Character variables contain data values consisting of a combination of letters, numbers and special characters (like \$, %, #, *).

(2) Contains numerals, letters, or special characters (such as \$ or !).

(3) Character data values can range from 1 to 32,767 characters in length.
The default is 8.

Note:

LIST input places a restriction on character input values. Character input values cannot be longer than 8 bytes(e.g. by default)

(4) Character data can contain embedded blanks.

(5) Sometimes data that consist solely of numerals make more sense as character data than as numeric. For instance ZIP codes, social security numbers or telephone numbers make more sense as character data.

Also...fields that contain values for grouping variables (such as 1=single, 2=married) are usually declared character variable as they will not be used in mathematical computations.

Exercise 3

Identify the following data values as either numeric, character, or either.

- (1) PEREZ, JOSE _____
- (2) A1 _____
- (3) 012-40-7777 _____
- (4) -2.7 _____
- (5) B- _____
- (6) 0.6 _____
- (7) SOUTH ORANGE, NJ _____
- (8) M. K. SMITH _____
- (9) +25 _____
- (10) 98% _____

8. Read Data Lines

8.1 INPUT Statement

Purpose: The INPUT statement reads data lines and assigns names to the SAS variables that correspond to the data fields.

We can use the INPUT statement to describe column-oriented data, list data, and formatted data. These input styles can be mixed within a single INPUT statement.

8.2 Column Input: When data are arranged in columns or fixed fields, you can read them using column input. That is, read input values from specified columns and assigns them to the corresponding SAS variable.

General Form: INPUT variable [\$] start—end ;
 | | | |
 (A) (B) (C) (D)

(A) variable choose a variable name to represent the field

(B) \$ identify a character variable with a dollar sign. SAS assumes variable is numeric unless it is followed by a dollar sign.

(C) start the number of the first column containing the data values for the variable

(D) end the number of the last column containing the data values for the variable

Column INPUT is appropriate to read standard data.

- (1) Standard numeric data values are values that contain only numbers, scientific notation, decimal points and minus signs
- (2) Values arranged in fixed column positions.

Sample program to illustrate fixed column positions:

```
DATA STATE;  
  INPUT STNAME $ 1-10 STCODE 11-12 AREA 13-15;  
DATALINES;  
NEW YORK 15032  
MISSOURI 36175  
N CAROLINA47009  
;
```

8.2.1 How Character Values are Read with Column Input Style

- character variables can be up to 32K
- a blank field is read as missing and does not cause other fields to be read incorrectly
- character data can have embedded spaces
- spaces are not required between values
- you can skip unwanted variables

Sample: Column Input style statement

```
INPUT NAME $ 1-16 ;
```

/*Using the (\$) attribute...notice that values are left-justified in the SAS data set and leading blanks are trimmed*/

<u>Data Values</u>	<u>SAS Data Set</u>
Manuel Perez A.K. Smith	Manuel Perez A.K. Smith

8.2.2 How Numeric Values are Read with Column Input

- Numeric values can occur anywhere in the field
- The placement of a numeric value does not affect how it is stored
- Missing numeric values can be represented as either a blank or a period in the input data lines
- SAS displays missing numeric value as period in the output

Sample column input style statement

```
INPUT X 1-8;
```

<u>Data Value</u>	<u>SAS Data Set</u>
24.0	24
24	24
24.00	24
2.4E1	24
.	.
+24	24

8.2.3 Other Features of Column Input style

(1) Data fields can be read in any order.

```
INPUT THIRD 7-10 FIRST 1 SECOND 4-5 ;
```

```
column ruler      1234567890
                  0 35 3254
                  9 47 6456
```

(2) Skip unwanted variables.

```
INPUT THIRD 7-10 ;
```

(3) Data fields may be in adjacent columns.

```
INPUT FIRST 1 SECOND 2-3 THIRD 4-7 ;
```

```
column ruler      1234567
                  0353254
                  9476456
```

(4) Decimal points can be inserted in data values. The number following the column specification gives the number of digits to the right of the decimal if the input value does not contain an explicit decimal point.

```
INPUT FIRST 1 SECOND 2-3 THIRD 4-7 .2 ;
```

Observations, variables , and values displayed in the Output window

OBS	FIRST	SECOND	THIRD
1	0	35	32.54
2	9	47	64.56

Exercise 4 (computer assisted)

Given the following data lines:

```
column           1           2
ruler            12345678901234567890

                JOHN      M 18 1698
                MARY      F 21 1724
                HAROLD    M 17 1916
                JENNIFER  F 19 1877
```

- (1) Write a DATA step and create a SAS data set.

- (2) Use column input style and write an INPUT statement to read the data lines shown above. The four variables names are: NAME SEX AGE SCORE

- (3) Submit the program in SAS

- (4) Modify the INPUT statement to have one decimal place in SCORE?
(That is, you want SAS to read the value(s) 169.8, 172.4, 191.6, 187.7)

- (5) Submit the program in SAS

Exercise 5 (computer assisted)

Given the following data lines:

```
column          1
ruler           123456789012345

A001 12 201
A002 14 403
A003 17 611
B001 19 312
B002 11 599
C001 10 461
C002 15 519
```

- (1) Write a DATA step and create a SAS data set.
- (2) Use column input style and write an INPUT statement to read the data lines shown above.

The variable names and the (**start-end** columns) are:

```
ID 1-4   GROUP 1   X1 6 - 7   X2 9 -11
```

NOTE: the variable X2 should have 2 digits to the right of the decimal point.

Sample: List Input style

Reminder!

The process for list input is...the first field is read until a blank space is encountered. The blank space indicates the end of the field, and the data value is assigned for the first variable...and so forth until each record is read.

```
DATA FITNESS;  
  INPUT NAME $ AGE WEIGHT ;  
DATALINES;  
SMITH 40 178  
WHITE 38 150  
KANE . 160  
MORELLA 29 210.2  
;
```

Observations, variables, and values displayed in the Output window

OBS	NAME	AGE	WEIGHT
1	SMITH	40	178.0
2	WHITE	38	150.0
3	KANE	.	160.0
4	MORELLA	29	210.2

By default SAS displays a missing value for a numeric variable as a single period (.) and a missing character value as a blank space

Exercise 6

100 M 23 416
2 F 19 640
1726 F 33 917
26 M 40 596

Given the data lines shown above write an INPUT statement which uses list input style to read the data

The four variables are: ID SEX AGE VAR1

Exercise 7

Will the following INPUT statement

```
INPUT NAME $ AGE HT WT ;
```

read each of the data lines shown below?

- | | | | | | |
|-----|-------------|----|----|-----|-------|
| (1) | CHARLIE | 23 | 73 | 206 | _____ |
| (2) | MARY ANN | 43 | 61 | 131 | _____ |
| (3) | CHRISTOPHER | 29 | 67 | 183 | _____ |
| (4) | CRAWFORD | 19 | 62 | 166 | _____ |
| (5) | ROBERT | | 67 | 192 | _____ |
| (6) | BETSY | 29 | 60 | 119 | _____ |
| (7) | KEVIN | 31 | 65 | | _____ |
| (8) | CATHERINE | 22 | | 121 | _____ |

Exercise 8 (computer assisted)

Enter the following SAS program in the enhanced editor window and create a SAS data set

```
DATA Sample_List_Input ;  
    INPUT WEIGHT HEIGHT GENDER $ ;  
    DATALINES;  
    155 68 M  
    98 60 F  
    280 75 M  
    130 63 F  
    ;
```


8.4.1 SAS informats - Numeric

numeric informat

w. w is the total width or the number of columns to read

or

w.d w is the total width or the number of columns to read
d is the number of decimal places

The period in any informat (e.g. the informat w. or w.d) is very important.

3. Reads 3 columns of numeric data

4.2 Reads 4 columns of numeric data and inserts a decimal point 2 digits from the right

8.4.1.1 COMMAw.d Informat

Reads numeric values and removes commas, blanks, dollar signs, percent signs, dashes, and right parentheses from the input data.

For instance ...

Given the data line: \$1,000,000

Input x comma10. ;

Result: 1000000

SAS reads 10 columns and strips out non-numeric characters

8.4.2 Formatted Input Style with implied decimal

To insert a decimal point, use the **w.d** informat. If there is a decimal point in the input value, **it takes precedence over** that provided by the **w.d** informat.

8.4.3 SAS informats - Character

character informat

- \$w.** w is the total width or the number of columns to read
 This particular character informat (**\$w.**) trims leading blanks
- \$6.** Reads 6 columns of character data and trims leading and trailing blanks

8.4.4 Column input versus formatted input style

Column: INPUT FNAME \$ 4-18 AGE 23-24 ;
or
Formatted: INPUT @4 FNAME \$15. @23 AGE 2. ;

<u>SAS informat used</u>	<u>Raw data entered</u>	<u>What SAS does!</u>	<u>SAS data set value displayed</u>
3.	125	Reads 3 columns of numeric data	125
4.2	3.15	Reads 4 columns of numeric data containing a decimal point	3.15
4.2	315	Reads 4 columns of numeric data and inserts a decimal point 2 digits from the right	3.15
\$6.	MARTIN	Reads 6 columns of character data	MARTIN
\$6.	SUE	Reads 6 columns of character data and <u>strips off leading blanks</u>	SUE
COMMA9.	\$1,250.21	Reads 9 columns of numeric data and strips out <u>non numeric</u> characters	1250.21
\$CHAR15.	SAS Institute SAS Institute	Reads 15 columns of character data. Does not trim leading blanks.	SAS Institute SAS Institute
COMMA10.	\$1,000,001	Read 10 columns, strips commas, and \$	1000001
PERCENT5.	5% (20%)	Converts percentage to numbers	0.05 -0.2

8.4.5 Repeated and same width fields

A important point to remember before proceeding with this example is...

When you use column input or formatted input, the input pointer rests on the column after the last column read.

Given the following column input statement...

```
INPUT ID $ 1-9 WK1 11-12 WK2 14-15 WK3 17-18 WK4 20-21 WK5 23-24 WK6 26-27 ;
```

Note: In this example we want you to assume that columns 10, 13, 16, 19, 22, and 25 in the data lines are blank fields. Therefore after SAS reads columns 1-9, SAS's internal pointer automatically goes to column 10 and reads columns 10,11, 12, ...columns 13, 14, 15,... columns 16, 17, 18, etc.

Another point we want to illustrate is that one may use the following abbreviated formatted variable list (WK1-WK6) and read the same data line(s).

For instance...

```
INPUT ID $ 1-9 (WK1-WK6) (3.) ;
```

or

```
INPUT ID $ 1-9 @10(WK1-WK6) (3.) ;
```

8.4.6 +n column pointer

+n moves the pointer n spaces.

For this particular example we will assume that columns 13, 16, 19, 22, 25, and 28 in the data lines do not have a blank. That is, we will presume there is some number or character in columns 13, 16, 19, 22, 25, and 28 that we do not want SAS to read.

With the +1 column pointer designated, one space is skipped after reading each of the variables WK1-WK6.

```
INPUT ID $9. @11(WK1-WK6) (2. +1) ;
```

|
reads a field width of 2 and moves the
column pointer over 1 space to the right

Exercise 9 (computer assisted)

Given the following raw data...

Social Security Number	Annual Salary	Age	Race
123-87-4414	28,000	35	W
646-23-9182	29,500	37	B
012-43-7652	35,100	40	W
018-45-1357	26,500	31	W

Write a SAS program and create a SAS data set .

When you code the INPUT statement use the **comma11.** informat to read the Social Security Number field and the **comma6.** informat to read the Annual Salary field. Also use the column pointer control to read the Annual Salary field.

For instance..

```
INPUT Social_Security_no comma11. @14 Annual_Salary comma6. ;
```

Write a PROC PRINT step and use a FORMAT statement to display the SAS data set with SAS defined formats.

For instance...

```
PROC PRINT;  
  FORMAT Social_Security_no ssn11. Annual_Salary comma6. ;
```

Exercise 10

(1) Given this column INPUT statement

```
INPUT IDNUM $ 1-8 SEQ1 11-13 SEQ2 15-17 SEQ3 19-21 SEQ4 23-25 ;
```

Use formatted input style and rewrite this INPUT statement in two ways.
(Do not assume that columns 14, 18, 22, 26 are blank)

INPUT_____

INPUT_____

(3) <i>column</i>	1	2
<i>ruler</i>	1234567890123456789012345	
00	4890	6.13
01	5624	2330
02	24.1	.01

Field Dictionary

<u>Variable name</u>	<u>Field location</u>	<u>Field width</u>	<u>Variable type</u>
CODE	1-2	2	Character
X	12-15	4	Numeric
Y	20-23	4	Numeric

Write two different input statements to read this data using formatted INPUT style.

INPUT _____

INPUT _____

8.5 Read more than one data line per observation

To read data with more than one line per observation, use the **#n** line pointer control.

For example, this INPUT statement

```
INPUT FNAME $ 1-8 SEX $ 11 AGE 13-14 HT 16-17 WT 20-22 #2 PHONE $ 11-22 ;
```

reads the following data lines

```
PAUL      M 27 72 140
          205-542-6898
JENNIFER  F 28 64 135
          303-986-2425
MARY      F 35 54 128
          212-241-6304
```

Observations, variables, and values displayed in the SAS Output window

OBS	FNAME	SEX	AGE	HT	WT	PHONE
1	PAUL	M	27	72	140	205-542-6898
2	JENNIFER	F	28	64	135	303-986-2425
3	MARY	F	35	54	128	212-241-6304

Now suppose we have two data lines per observation, but we are only interested in data from the first line. We still use **#2** to indicate that there are a total of two data lines per observation. For instance, in the data lines above, if we want to read only FNAME, SEX, AGE, and HEIGHT, the following INPUT statement would read the data:

```
INPUT FNAME $ 1-8 SEX $ 11 AGE 13-14 HT 16-17 #2 ;
```

Observations, variables, and values displayed in the SAS output window

OBS	FNAME	SEX	AGE	HT
1	PAUL	M	27	72
2	JENNIFER	F	28	64
3	MARY	F	35	54

8.6 Create more than one observation from a single data line

The double trailing at sign (@@) tells SAS to hold the current data line through multiple executions of the DATA step. It works like a stop sign telling SAS, "STOP, hold that line of raw data."

Note:

By default the internal SAS pointer moves to the next record when there is no more data to be read on a line.

Suppose we want to read a sequence of 3 test scores per individual.

```
DATA TESTDATA ;
  INPUT TEST1 TEST2 TEST3 @@ ;
DATALINES;
56 72 46 84 96 89 95 90 75
80 70 79 100 98 87
;

PROC PRINT ;
RUN;
```

Observations, variables, and values displayed in the Output window

OBS	TEST1	TEST2	TEST3
1	56	72	46
2	84	96	89
3	95	90	75
4	80	70	79
5	100	98	87

9. Read dates with SAS defined informats

Dates can be tricky to work with. Some months have 30 days, some 31 days, some 28, and then leap years. SAS dates simplifies all of this.

A SAS date variable is the number of days since January 1, 1960.

Given dates are...

Jan 1, 1959

Jan 1, 1960

Jan 1, 1961

Jan 1, 2001

and the corresponding SAS date values are...

-365 days

0 days

+366 days

+14976 days

To let SAS know that a variable represents a date, use formatted input style and use a date informat following the variable name in the input statement.

SAS has a variety of date informats for reading dates in different forms. All of these informats convert your data to a number equal to the number of days since Jan. 1, 1960.

SAS defined date informats	Read date values in the form of...	Informats used...	Reads...
DATEw.	ddmmmyy ddmmmyyyy	DATE7. DATE9. DATE11.	15FEB92 15FEB1992 15-FEB-1992
DDMMYYw.	ddmmyy ddmmyyyy	DDMMYY6. DDMMYY8. DDMMYY8.	150292 15/02/92 15021992
MMDDYYw.	mmddy mmddyyyy	MMDDYY6. MMDDYY8. MMDDYY8.	021592 02 15 92 02151992
YYMMDDw.	yymmdd yyymmdd	YYMMDD6. YYMMDD8. YYMMDD8.	920215 92/02/15 19920215

Exercise 11 - Read dates in SAS

Associate each of the following with the appropriate SAS date informat:

Date value

SAS Date informat

05MAR1947

21/05/47

22 05 47

10/21/89

2-16-90

7FEB90

Exercise 12 (computer-assisted)

- (1) Correct the following INPUT statement to read the data lines corresponding to the variables GRP and SCORE when you have 4 subjects in a placebo group and 4 in a drug group.

```
DATA PLACEBO ;  
  INPUT GRP $ SCORE ;  
DATALINES;  
P 77 P 76 P 72 P 68  
D 81 D 82 D 84 D 82  
;
```

- (2) **Re-structure** the data lines (shown above) so that the following INPUT statement will read the data lines.

```
DATA PLACEBO ;  
  INPUT GRP $ SCORE ;  
DATALINES;
```

```
;  
;
```

10. FORMAT Statement

Purpose: Use a FORMAT statement to tell SAS to associate formats with variables.

General Form:

```
FORMAT variables format ;  
      |           |  
      (A)        (B)
```

- (A) variables names the variable(s) to assign a format.
- (B) format the format assigned to the variable.
Prefix a format with a \$ sign when a character variable is identified.

The FORMAT statement can go in either the DATA steps or the PROC steps.

If the FORMAT statement is used in the DATA step, format association is stored with the SAS data set for the duration of that SAS session.

If the FORMAT statement is used in the PROC step, formats are assigned to the variables — affecting only the results from that procedure. Subsequent procedures do not use the format. Formats used in the PROC step override any formats assigned to variables in the DATA step.

Sample format statements...

- (1) FORMAT TOTALS 7.2 ;
- (2) FORMAT AMTSALES DOLLAR9.2 CREDIT COMMA8.2 ;
- (3) FORMAT BIRTHDAY TESTDATE MMDDYY8. ;

11. Display variable values with SAS defined formats

SAS defined formats are used to display 'real' values in the SAS output.

- 1) **w.d** Writes numeric values in an output field(w positions wide), and (d positions to the right of the decimal point). The value of w accounts for the entire width(that is, the integer portion, the decimal point and the number of positions to the right of the decimal point).

If you **do not** specify a **d** value as part of the format SAS writes the value without a decimal point.

<u>Value</u>	<u>w.d</u>	<u>Result</u>
3702.51	7.	3703
3702.51	7.2	3702.51

An example of a Format statement is **Format NUM1 7.2 ;**

- 2) **DOLLARw.d** Writes numeric values with dollar sign, commas, and a decimal point. The range(2-32) of **w** represents the total width of the output field. Make **w** wide enough to write the numeric values, dollar sign, commas, and a period that separates the decimal portion. The range(0-31) of **d** optionally specifies the number of digits to the right of the decimal point. If **d** is 0, **DOLLARw.d** format does not write a decimal. If **d** is 2, **DOLLARw.d** format writes a decimal point and 2 decimal digits.

<u>Value</u>	<u>DOLLARw.d</u>	<u>Result</u>
3702.51	DOLLAR9.2	\$3,702.51
3702.51	DOLLAR6.0	\$3,703
3702.51	DOLLAR6.	\$3,703

An example of a Format statement: **Format AMT dollar9.2 ;**

- 3) **COMMAw.d** Does exactly the same as **DOLLARw.d** format except that no dollar sign is displayed. Make **w** wide enough to write the numeric values, commas, and the optional decimal point.

value	COMMAw.d	Result
3702.21	COMMA8.2	3,702.21

An example of a Format statement: **Format TOTAL comma8.2 ;**

- 4) **DATEw.** Display SAS date values in the form **ddmmmyy**, where **dd** is the day, **mmm** is the first three letters of the month name, and **yy** or **yyyy** is the year.

value	DATEw.	Result
10072	DATE7.	30JUL87
10072	DATE9.	30JUL1987

An example of a Format statement: **Format DOB date9. ;**

- 5) **MMDDYYw.** Display a SAS date value in the form **mmddy** or **mmddy**, where **mm**, **dd**, **yy** are integers representing the month, day, and (two- or four-digit)year.

value	MMDDYYw.	Result
10072	MMDDYY8.	07/30/87
	MMDDYY10.	07/30/1987

An example of a Format statement: **Format NEWDATE mmddy10. ;**

12. LABEL statement: To assigns descriptive labels to variable names

Long column labels: Maximum length for column labels is **256**.

for instance...

```
DATA long_label ;  
  INPUT  annual_salary  comma6. ;  
DATALINES;  
26,000  
45,000  
70,000  
;
```

```
PROC PRINT Data = long_label  Label ; /* Label option is necessary to display labels in Output*/  
  Var  annual_salary ;  
  Label  annual_salary = "Don't you wish this was your salary because airline  
  folks make so much more money and they get lots of benefits. But of course, you'd have  
  to be out of town a lot, so that is a real drawback to this job";  
  Format  annual_salary  dollar7. ;  
RUN;
```

Exercise 13 (computer-assisted)

1. Add a FORMAT statement to the DATA step shown below
 - (a) assign the date format `DATEw.` to the variable `DAY` and
 - (b) assign a format `w.d` to the variable `TEMP`.

```
DATA CITIES;  
INPUT  CITYNAME $  TEMP  @16 DAY MMDDYY8. ;
```

```
DATALINES;  
Austin  74.8    03/15/92  
Chicago 47.34   03/15/92  
Miami   80.3    03/16/92  
;
```

2. Rewrite the FORMAT statement in (1) so that the values of the variable `DAY` are displayed with the date format `mm/dd/yy`
3. In the program shown below, write a FORMAT statement in the DATA step so that the values of the variable `UNITPRICE` is displayed in each PROC output with dollar signs and commas, and the values of the variable `AMOUNT` is displayed with commas.

```
DATA  ITEMS;  
INPUT  ITEM $      UNITPRICE  AMOUNT  ;
```

```
DATALINES;  
UX123   999.29    5000  
UY500   1200.00   45000  
UZ550   1499.99   45500  
;
```

```
PROC PRINT;  
RUN;
```

4. Where would you place the FORMAT statement if it is to apply to the PROC PRINT only?

MODULE 3: Temporary and permanent SAS data sets

- DATA step logic
- PROC step logic
- Interpret messages in the SAS log
- Create temporary SAS data set
- Create permanent SAS data set
- Examine Proc Contents procedure

13. DATA step logic

An overview of the SAS DATA step may be a bit technical, but an understanding of how SAS works will help you to fully appreciate how the SAS System and the SAS DATA step in particular really works.

A DATA step is processed in two phases...*when you hit 'SUBMIT'*.

- compile phase
- execute phase

13.1 DATA Step - Compile Phase

What happens during the compilation phase?

(1) Checks syntax.

- missing or misspelled keywords
- invalid variable names
- missing or invalid punctuation
- invalid options

Most syntax errors prevent further processing of the DATA step.

Once a syntax error is detected, an ERROR message is written to the SAS log, and the SAS Step cannot be executed. However, the rest of the code is checked for other syntax errors.

(2) User errors are **not** detected. These types of errors are more dangerous because a program can compile and execute successfully, but generate incorrect results. Careful examination of the output is often the only way to detect user errors (logic errors).

(3) SAS software and operating system set up needed resources.

(a) Where is the data coming from?

(b) Opens the working areas.

- Input buffer is a temporary area that holds each data line as it is read in.
- **Program Data Vector** is a temporary area which defines a slot for each variable in the INPUT statement (*or when a SAS data set is read*) and all new variables created in the DATA step. The program data vector contains two automatic variables that can be used for processing but are NOT written to the data set as part of an observation.
 - **_N_** counts the number of times that the DATA step has begun to execute
 - **_ERROR_** signals the occurrence of an error caused by the data during execution.

(c) Prepares descriptor portion of the SAS data set.

(4) Each DATA step or PROC step must compile successfully before each is executed.

set up resources...

Data Line:

PAUL	M	27	72	140
JENNIFER	F	28	64	135
RENEE	F	35	54	128
MANUEL	M	35	60	125
TONY	M	32	68	130

SAS Program:

```
INPUT FNAME $ 1-8 SEX $ 11
      AGE 13-14 HT 16-17
      WT 20-22 ;
```

Input Buffer:

```
          1          2          3
123456789012345678901234567890123456
```

Program Data Vector:

N	_ERROR_	FNAME	SEX	AGE	HT	WT
-----	---------	-------	-----	-----	----	----

SAS Data Set:

descriptor portion.....>

- data set name
- date/time created
- variable names
- number of observations and variables
- informats and formats

data value portion.....>

PAUL	M	27	72	140
JENNIFER	F	28	64	135
RENEE	F	35	54	128
MANUEL	M	35	60	125
TONY	M	32	68	130

13.2 DATA Step - Execute phase

After the DATA step is compiled, it is ready for execution. During the execution phase, the data portion of the data is created. The data portion contains the data values.

What happens during execute phase?

1. At the beginning of the execution phase, the value of `_N_` is 1. Because there is no data errors, the value of the `_ERROR_` is 0.

Each variable in the Program Data Vector (PDV) is initialized to missing.

Missing numeric values are represented by a period and missing character values are represented by a blank.

Program Data Vector

<code>_N_</code>	<code>_ERROR_</code>	<code>FNAME</code>	<code>SEX</code>	<code>AGE</code>	<code>HT</code>	<code>WT</code>
1	0			.	.	.

2. INPUT statement reads the current data line and substitutes the actual data values for the missing values.

These data values are kept in the Program Data Vector until they are transferred to the SAS data set.

1	0	PAUL	M	27	72	140
---	---	------	---	----	----	-----

3. SAS then executes each statement in the DATA step for the current observation.
4. By default, the values in the Program Data Vector are written as an observation to the SAS data set.
5. Steps 1 to 4 are repeated for each observation(one observation at a time).

That means, SAS takes the first observation and “runs” all the way through the DATA step (e.g. line by line of code) before looping back to pick up the second data line, and so forth

6. Execution of the DATA step stops when there are no more data lines to process.
7. SAS proceeds to the next step in the SAS program.

14.PROC Step Logic

PROC step invokes a pre-written program called a SAS procedure (a PROC) and begins with a PROC statement.

PROC step automatically repeats the following functions for each observation in the data set.

- identifies the SAS data set
- reads an observation
- processes the observation
- stores the processed information such as the descriptive statistics, the produced listings, produced tables, produced analysis, produced reports
- many PROCs do create output SAS data sets

General form of the PROC statement:

PROC	procname	options	;
(A)	(B)	(C)	

- | | |
|--------------|--|
| (A) PROC | SAS keyword |
| (B) procname | names the procedure to operate on the SAS data set |
| (C) options | actions applicable to processing a procedure |

The PROC statement is most frequently followed by procedure information statements.

Sample PROC steps:

```
PROC PRINT DATA = sasdatasetname ;  
  VAR FNAME AGE ;  
  TITLE 'PROC Print output' ;  
RUN;
```

```
PROC SORT DATA = sasdatasetname ;  
  BY variables ; /* a required statement in the PROC SORT procedure */  
RUN;
```

```
PROC MEANS DATA = sasdatasetname ;  
  VAR variables ;  
RUN;
```

```
PROC FREQ DATA = sasdatasetname ;  
  TABLES variable(s) list ;  
  TITLE1 ' This ia a title ' ;  
  TITLE2 " Here is another title " ;  
  TITLE3 ' Here''s another title ' ;  
RUN;
```

The BY statement is optional in PROC(s) other than PROC SORT. If you do use a BY statement in other procedures, it tells SAS to perform a separate analysis for each combination of values of the BY variable(s) rather than treating all observations as one group. If your observations are not already sorted, then use PROC SORT to do so.

15. Interpret Messages in the SAS Log

When programming errors are made, such as misspelling SAS keywords, forgetting semicolons, or entering invalid data, the SAS log will display:

- the word NOTE, WARNING, or ERROR
- the location of the error
- a message explaining the error at the end of the SAS step.

1. Variable not found

Sample Log:

```
1          DATA ERR1 ;
2          INPUT IDCODE $   AGE ;
3          DATALINES;
```

NOTE: The data set WORK.ERR1 has 2 observations and 2 variables.
NOTE: The DATA statement used 0.04 CPU seconds.

```
6          ;
7          PROC PRINT ;
8          VAR ID AGE ;
```

ERROR: Variable ID not found.

NOTE: The SAS System stopped processing this step because of errors.

NOTE: The PROCEDURE PRINT used 0.01 CPU seconds.

ERROR: Errors printed on page 1.

NOTE: The SAS session used 0.22 CPU seconds.

NOTE: SAS Institute Inc., SAS Circle, PO Box 8000, Cary, NC
27512-8000

2. Invalid data error

If SAS detects a data error during execution phase, SAS will:

- Displays to the SAS log a note describing the error
- List the values stored in the input buffer
- List the values stored in the program data vector, including `_N_`, the number of the data line, and `_ERROR_`, is assigned 1 when an error is encountered
- Set the invalid data to missing
- Continue processing with the next data line

Sample Log:

```
1          DATA ERR2 ;
2          INPUT IDCODE $    AGE NUM1    ;
3          DATALINES;

NOTE: Invalid data for NUM1 in line 5 14-18.
RULE:      ----+-----1-----+-----2-----+-----3-----+-----4-----+-----5----
5          A02    27    6.5.9
IDCODE=A02 AGE=27 NUM1=. _ERROR_=1 _N_=2
NOTE: The data set WORK.ERR2 has 2 observations and 3 variables.
NOTE: The DATA statement used 0.06 CPU seconds.

6          ;
7
8          PROC PRINT ;
NOTE: The PROCEDURE PRINT printed page 1.

NOTE: The PROCEDURE PRINT used 0.02 CPU seconds.

NOTE: The SAS session used 0.24 CPU seconds.
NOTE: SAS Institute Inc., SAS Circle, PO Box 8000, Cary, NC
      27512-8000
```

Sample Output:

OBS	IDCODE	AGE	NUM1
1	A08	34	6.5
2	A02	27	.

16. Temporary SAS Library or Permanent SAS Library

Before we create a new SAS data set, we need to consider which library we will use to store it. Do we create a WORK library or a permanent SAS library?

16.1 Create a temporary SAS data set in the WORK library

Limitation: A temporary SAS data set exists only for the current SAS session.

Temporary data sets have an important limitation imposed on them. Temporary data sets are **deleted** at the end of the SAS batch job or interactive session. However, they can be used any number of times within the batch or interactive session.

When the first-level name is omitted, a temporary SAS data set is created and held in a special work space that is assigned the default libref **WORK**.

17. Create Permanent SAS Data Sets

Permanent SAS data sets **must** specify both level names.

Why create permanent SAS data sets?

- 1) Probably the most compelling reason to create and use permanent SAS data sets is speed. It is safe to say that typical SAS programs use most of the machine resources in the **DATA** step.
- 2) If you plan to be running many different analyses on a data set that will not be changing often, it is a good idea to make the SAS data set permanent for the duration of the analyses.
- 3) SAS data sets are also a good way to transfer data to other users.
- 4) Once we have created a permanent SAS data set, it is available for processing in subsequent SAS session.

Question: How is the libref or library reference name created for a permanent SAS Library?

Answer: You may use either the “point-and-click” method through the SAS system’s pop-up window, or you may actually code a LIBNAME statement in your SAS program.

Both methods illustrated during the class session.

17.2 Illustrate a couple of ways to create permanent SAS data set

Sample with the LIBNAME statement

```
LIBNAME IN1 "c:\class\samples" ; /*path-name may be enclosed in either single or double quotes*/
```

```
DATA IN1.Display_Sample ;  
  INPUT FNAME $ 1-8 SEX $ 11 AGE 13-14 HT 16-17 WT 20-22 ;  
DATALINES;  
PAUL      M 27 72 140  
JENNIFER  F 28 64 135  
RENEE     F 35 54 128  
MANUEL    M 35 60 125  
TONY      M 32 68 130  
;
```

Sample without the LIBNAME statement

```
DATA "c:\class\samples\display_sample" ; /*path-name may be enclosed in either single or double quotes*/  
  INPUT FNAME $ 1-8 SEX $ 11 AGE 13-14 HT 16-17 WT 20-22 ;  
DATALINES;  
PAUL      M 27 72 140  
JENNIFER  F 28 64 135  
RENEE     F 35 54 128  
MANUEL    M 35 60 125  
TONY      M 32 68 130  
;
```

Sample under OS/390 operating system (mainframe):

```
//ABC JOB (XYZ3,,A),NAME  
//  JCLLIB ORDER=ZABCRUN.PROCLIB  
//  EXEC SAS  
//SYSIN DD *  
  
LIBNAME OUT1 'XYZ3ABC.SAMPLES' DISP=(NEW,CATLG)  
          UNIT=FILE SPACE=(TRK,(1,1)) ;  
  
DATA OUT1.Display_Sample ;  
INPUT FNAME $ 1-8 SEX $ 11 AGE 13-14 HT 16-17 WT 20-22 ;  
DATALINES;  
PAUL      M 27 72 140  
JENNIFER  F 28 64 135  
RENEE     F 35 54 128  
MANUEL    M 35 60 125  
TONY      M 32 68 130  
;
```

17.3 Sample to access permanent SAS data set in a PROCedure

Sample under Windows:

```
LIBNAME IN1 "c:\class\samples" ; /* may use either double or single quotes*/  
PROC FREQ DATA = IN1.Display_Sample ;  
    TABLE SEX ;  
RUN;
```

or without the LIBNAME statement

```
PROC FREQ DATA = 'c:\class\samples\display_sample' ;  
    TABLE SEX ;  
RUN;
```

Sample under OS/390 (mainframe)

```
LIBNAME IN1 'XYZ3ABC.SAMPLES' ;  
PROC FREQ DATA = IN1.Display_Sample ;  
    TABLE SEX ;
```

18. Proc Contents procedure

Whenever one creates a permanent SAS data set, some form of documentation should be prepared. The documentation should provide sufficient means for someone to determine what information the SAS Library contains and how to get it. PROC CONTENTS has some nice ways to supplement hand-prepared documentation.

For instance...to display the contents of a permanent SAS data set called **Display_Sample**

```
LIBNAME IN1 "c:\class\samples" ;  
PROC CONTENTS DATA = IN1.Display_Sample ;  
RUN;
```

or without the LIBNAME statement

```
PROC CONTENTS DATA = 'c:\class\samples\display_sample' ;  
RUN;
```

Note for OS/390 (mainframe) users:

If you **do not** remember the second-level name of a permanent SAS data set on the OS/390 operating system(mainframe) you may use a special SAS name **_ALL_** to reference the SAS library. Although the special name **_ALL_** may be used in any operating environment; it is more commonly used on OS/390 (mainframe).

For instance...

```
LIBNAME IN1 'XYZ3ABC.SAMPLES' ;  
PROC CONTENTS DATA = IN1._ALL_ ;
```

The PROC CONTENTS procedure will display the names and content of all the SAS data sets identified in the SAS library.

Exercise 14 (computer-assisted)

Given the following data lines, create a permanent SAS data set and a temporary SAS data set. Run PROC CONTENTS on the permanent SAS data set and PROC PRINT on the temporary SAS data set.

```
Column          1          2
Ruler           12345678901234567890

TUESDAY 400 200 16
MONDAY 516 311 19
THURSDAY 319 419 101
FRIDAY 666 111 23
```

Exercise 15 (computer assisted)

Given the following set of data:

ID	RACE	SBP	DBP	HR
101	W	130	80	60
102	B	140	90	70
103	W	120	70	64
104	W	150	90	76
105	B	124	86	72

(Note: SBP is systolic blood pressure, DBP is diastolic blood pressure, and HR is heart rate.)

- 1) Use the data(above) and create a SAS data set.
- 2) Use the PROC SORT procedure to sort the data in increasing order of SBP
PROC SORT;
BY SBP ;
- 3) Use the PROC PRINT procedure to display the SAS data set
PROC PRINT ;
 - 3a) Use the VAR statement in the PROC PRINT procedure to exclude the variable HR
VAR ID RACE SBP DBP ;
 - 3b) Use a Title statement in the PROC PRINT procedure
Title 'Race and Hemodynamic Variables' ;

**Other Base SAS software
features that may be helpful
to you...**

SAS and year 2000

How is SAS Institute handling the date transition between the end of the 20th century and the beginning of the 21st century?

All versions of the SAS System represent dates from 1582 A.D. to 20,000 A.D. correctly. Leap years, century, and fourth-century adjustments are made automatically. Leap seconds are ignored, and the SAS System does not adjust for daylight savings time.

The SAS System is prepared to handle dates in the 21st century. The best situation is to have dates with four-digit years in both your external data sources and your SAS programs.

- If the dates in your external data sources or your SAS program contain four-digit years, then the SAS software system will accept those four-digit years without difficulty as long as you choose the appropriate SAS date informat
- If the dates in your external data sources or SAS programs contain two-digit years, the SAS System assumes by default that two-digit years represent dates in the 20th century, (that is, from 1900 to 1999). This 100-year span is controlled by the YEARCUTOFF = *system option*. The default values for the YEARCUTOFF= *system option* are show on the following page.

For instance...

datew. informat reads dates defined in the form **ddmmmyyyy** as well as **ddmmmyy**

yymmddw. informat reads dates defined in the form **yyymmdd** as well as **yymmdd**

ddmmyyw. informat reads dates defined in the form **ddmmyyyy** as well as **ddmmyy**

mmddyw. informat reads dates defined in the form **mmddyyyy** as well as **mmddy**

(...continued on next page)

The YEARCUTOFF=option is used by SAS software to assign a century prefix to two digit years used in SAS programs and input data.

How does the YEARCUTOFF=option work?

The YEARCUTOFF=option specifies the first year of a 100 year window in which all 2-digit years are assumed to be. For example, if the YEARCUTOFF= option is set to 1920, all 2-digit years are assumed to be in the period 1920 through 2019. This means that two-digit years from 20-99 will be assigned a century prefix of '19' and all 2 digit years from 00-19 will have a century prefix of '20'.

What types of date values are, and are not, affected by the YEARCUTOFF=option?

The YEARCUTOFF=option **affects** the interpretation of two digit years in the following cases:

- Reading date values from external files
- Specifying dates or year values in SAS functions
- Specifying SAS date literals

The YEARCUTOFF=option has **no affect** in the following cases:

- Processing dates with 4 digit years
- Processing dates already stored as SAS date values(the number of days since January 1, 1960)
- Displaying dates with SAS date formats

What are the current YEARCUTOFF = system option default values at NIH?

WIN95/98/NT

...Version 6 releases(6.10 and higher) default is 1900

...Version 8 default is 1920

Mac

...Version 6 releases(6.10 and higher) default is 1900

OS/390 operating system (mainframe)

...Version 6.09 default is 1920

...Version 8(8.1 and higher) default is 1920

SAS reads a file with hierarchical data

Raw data file can be hierarchical in structure, consisting of a header record and one or more detail records. Typically, each record contains a field that identifies the record type.

Using the single trailing @ sign to read specified field(s) from an hierarchical file

A single trailing @ sign, placed at the end of an INPUT statement means “hold the line” for the current execution of the DATA step. That is, the pointer will not be moved to the next record.

Sample raw data:

```
H IDNUM1      TONY          07JAN1991
T              21JAN1992
T              05FEB1991
H IDNUM2      RENEE          10JUN1994
T              24JUN1991
T              29JUL1998
T              13AUG1993
```

Notice that there are two INPUT statements. The first reads the character variable TYPE and then ends with a trailing @. The trailing @ holds each line of data while the IF statement tests it. If a record does not have a value 'H' for the variable TYPE, then the second INPUT statement never executes. Instead SAS returns to the beginning of the DATA step to process the next record, and does not add the unwanted observation to the TEMP5 data set.

Sample program:

```
DATA TEMP5 ;
  INFILE 'c:\sasclass\project2.dat' ;
  INPUT TYPE $1. @ ;
  IF TYPE = 'H' THEN INPUT @14 FNAME $6.
    @21 TESTDATE DATE9. ;
  ELSE DELETE; /* the DELETE statement is a topic discussed in SAS Programming Fundamentals II */

PROC PRINT;
  FORMAT TESTDATE MMDDYY9. ;
RUN;
```

Sample Output:

Observations, variables, and values displayed in the Output window

OBS	TYPE	FNAME	TESTDATE
1	H	TONY	01/07/91
2	H	RENEE	06/10/94

Modified list input style may be used to read values which contain embedded blanks and nonstandard values. Modified list input style uses two format modifiers,

- 1) :
- 2) &

1. The colon : format modifier

The colon : format modifier enables you to read nonstandard data values and character values longer than eight characters without embedded blanks

The colon : format modifier combines several SAS input styles. It is placed before a format (e.g., : 3. , : \$char20. , : date7.) and tells SAS to read the variable beginning in the next nonblank column and ending when either a blank column is read, the length of the variable (if character) is reached, or the end of the data line is reached. This feature frees the user from lining up data values in specific columns yet keeps the advantages of using special formats such as COMMAw.d and DATEw. informats.

Sample program:

```
DATA COLON ;  
    INPUT LNAME : $12. TAXDUE COMMA7. ;  
DATALINES ;  
BACKENHEIMER 2,350  
CUCCINELLI 1,230  
;  
  
PROC PRINT ;  
    FORMAT TAXDUE comma5. ;  
RUN ;
```

Sample output:

OBS	LNAME	TAXDUE
1	BACKENHEIMER	2,350
2	CUCCINELLI	1,230

2. The ampersand & modifier

The ampersand & modifier enables you to read character values that contain single embedded blanks

By default LIST input style cannot read variables with embedded blanks within a character string.

The ampersand & modifier instructs SAS to allow embedded blanks when reading character variables. It can either precede or follow the \$.

Pointer and pitfall:

Peculiar results when reading character data with LIST input style are often attributable to omission of the & format modifier. If your character data consists of several “words” per variable and you plan to use LIST input style, be sure you separate values for the variables by at least two blanks.

Sample program:

```
DATA AMPERSND;  
  INPUT  NAME & $ AGE ;  
DATALINES;  
MARY KIM  34  
JOHN DOE  28  
;  
  
PROC PRINT;  
RUN;
```

Sample output:

OBS	NAME	AGE
1	MARY KIM	34
2	JOHN DOE	28

Diagnostic tools:

The ? and ?? format modifiers can be used for numeric variables.

Pointer and pitfall:

SAS supplies these diagnostic tools as aids:

Use the ? and the ?? modifiers only when you are comfortable with the presence of bad data and SAS's default reaction to it. Consider using them only when you want to avoid long SAS Logs, when you are already aware that something may be wrong with the data, and when you are willing to let SAS set these values to missing.

The ? Format Modifier

The ? format modifier suppresses the message about invalid data. But it still prints the data line and the variable values.

Sample Log:

```
1          DATA ERR2 ;
2          INPUT IDCODE $ AGE NUM1 ? ;
3          DATALINES;

RULE:      ----+-----1-----+-----2-----+-----3-----+-----4-----+-----5----
5          A02 27 2.7.1
IDCODE=A02 AGE=27 NUM1=. _ERROR_=1 _N_=2
NOTE: The data set WORK.ERR2 has 2 observations and 3 variables.
NOTE: The DATA statement used 0.06 CPU seconds.

6          ;
7
8          PROC PRINT ;
NOTE: The PROCEDURE PRINT printed page 1.
NOTE: The PROCEDURE PRINT used 0.02 CPU seconds.

NOTE: The SAS session used 0.24 CPU seconds.
NOTE: SAS Institute Inc., SAS Circle, PO Box 8000, Cary, NC
      27512-8000
```

Sample Output:

OBS	IDCODE	AGE	NUM1
1	A08	34	6.5
2	A02	27	.

The ?? Format Modifier

The ?? format modifier suppresses the message about invalid data, the raw data listing, and the variable listing. Thus ?? makes even “dirty” data look “clean” to the reader of the SAS Log.

Sample Log:

```
1          DATA ERR2 ;
2              INPUT IDCODE $    AGE NUM1  ?? ;
3          DATALINES;
```

NOTE: The data set WORK.ERR2 has 2 observations and 3 variables.
NOTE: The DATA statement used 0.04 CPU seconds.

```
6          ;
7
8          PROC PRINT ;
```

NOTE: The PROCEDURE PRINT printed page 1.
NOTE: The PROCEDURE PRINT used 0.03 CPU seconds.

NOTE: The SAS session used 0.24 CPU seconds.
NOTE: SAS Institute Inc., SAS Circle, PO Box 8000, Cary, NC

Sample Output:

OBS	IDCODE	AGE	NUM1
1	A08	34	6.5
2	A02	27	.

