NOTE #1: DATA STEP: BASIC

Syntax Basic:

```
DATA yourdataname;

< statements >

RUN;
```

Note:

a. SAS is not case sensitive
b. A data name must be 32 characters or less in length and must start with a letter or _.
c. All statements must end with a semicolon (;).

Getting Data into SAS: There are basically three ways of getting data into SAS;

a. Typing data directly in the editor widow in DATA step (see Example1). You can also enter data using the Viewtable window (Tools/Table Editor).
b. Call in an external ASCII data file (text file) using *infile* statement in DATA step. Use PROC IMPORT for data format other than ASCII. (e.g. Excel, Lotus, etc.)
c. Using the Import Wizard under File/Import Data.

```
/*** Example1 (Typing data directly) ***/
/* put all the comments you want in between these brackets.
   Comments are shown in green in the Editor window */
* you can also put comments between star (*) and semicolon(;) ;

DATA mydata1;          *The name of data is mydata1;
INPUT city $ F;        *and has 2 variables city and F.;
                       *city is character and F is numeric variable;
 C=(F-32)*5/9;         *New variables can be defined after INPUT;
                       *statement;
 Format C F4.1;        *Format is used to specify the variable format;

 label C="Celsius" F="Fahrenheit";

DATALINES;               *Now, entering the data;
LA 85
SD 78
Orange 81
Anaheim 88
Riverside 94
LV 102
;                        *the end of data;

PROC PRINT label DATA=mydata1;   *print the data with labels;
RUN;
/*************************************************************/

/*** Example2 (INFILE) ***
Save the six lines of data in Example1 as an ASCII file (text file).
Assuming the file is named 'E:\testdata1.dat'.
*/
```

```
      DATA mydata2;
       INFILE 'E:\testdata1.dat';
       INPUT city $ F;
         C=(F-32)*5/9;
         Format C F4.1;
         label C="Celsius" F="Fahrenheit";
      RUN;

      PROC PRINT label DATA= mydata2;
      RUN;

/* You can also do */

      Filename test1 'E:\testdata1.dat';

      DATA mydata2;
       INFILE test1;
       INPUT city $ F;
         C=(F-32)*5/9;
         Format C F4.1;
         label C="Celsius" F="Fahrenheit";
      RUN;
      PROC PRINT label DATA= mydata2;
      RUN;
/*************************************************************/

  /*** Example3 (PROC IMPORT) ***
  Assuming the data above is saved as an excel file named
  E:\testdata1.xls */

  PROC IMPORT OUT=mydata3 REPLACE
      DATAFILE='E:\testdata1.xls';
      SHEET="sheet1";
      GETNAMES=NO;
  RUN;
/*************************************************************/
```

Making a DATA file from SAS data

You can generate an output data or report using FILE and PUT statement.

```
/*** Example4 ***/

      DATA mydata3; set mydata2;
      Today=Today( );
      FILE 'E:\outdata1.dat' PRINT;
      PUT @1 "Daily Maximum Temperature Report " Today MMDDYY8. //
          @2 "CITY" @15 "Celsius" @20 "Fahrenheit"/
          @2 city @20 F @25 C;
      RUN;


/* A better one. Header is used */

      DATA mydata2; set mydata1; today=today( );
```

```
FILE 'E:\outdata1.dat' PRINT notitle HEADER=head;
PUT  @2 city @20 C @32 F; return;
head:
PUT @1 "Daily Maximum Temperature Report " Today MMDDYY8. //
         @2 "CITY" @15 "Celsius" @30 "Fahrenheit"/
         @1 40*'=';
return;
RUN;
```

Working with Library

SAS Data Name consists of two levels: libref.dataname. For example, mylib.mydata1; mylib is the libref and mydata1 is the data name in the folder where you defined the library mylib. To have your data permanent you have to make a library in a specified folder and use the two-level data name in a DATA step. Note that libref name can not exceed 8 characters in length. You can name the data without the library reference (libref). Then SAS will use the default libref name WORK. This library is temporary means that once you close a SAS session the data will disappear. There are two ways of making a library.

1. Before a DATA step, type

```
LIBNAME mylib 'E:\SASCLASS';
```

   Now, the library mylib is generated. Use the two-level data name in the DATA step.

```
DATA mylib.mydata1;
  < statements>
RUN;
```

2. You can also define a library using the New Library Window in SAS Explorer (right click in the Active Library window then New). Once it is done, use the two-level name in the DATA step.

NOTE: Without making a library, you can also make a permanent SAS data by direct referencing in DATA step. For example,

```
DATA 'E:\SASCLASS\mydata1';
  INFILE 'E:\SASCLASS\testdata.dat';
  < INPUT statements>; RUN;
```

```
/* PRACTICE
Make the data in Example1 a permanent data using a library via both
methods.
*/
```

```
/************* In-class exercise ******************************/
/*
NAME DOB HT WT SEX WAGE EDU
Susie Frieman   07/11/81 42 41  F  34000  HS
Charlie Smith   10/26/54 65 160 M  125000 C
Calvin Lee      01/10/81 69 140 M  87525  C
Lucy Chen       11/23/79 58 97  F  48900
Bob Lead        07/30/65 72 182 M  38500  HS
Sue Halfcott    01/07/68 58 110 F  79000  HS
Hans Richman    03/06/48 60 125 M  154500 C
```

1. Copy and paste the data above in notepad and save as prac1.dat in the directory E:\SASCLASS (E is a CDR drive. Make sure your drive name. You should make the directory SASCLASS in your CD first.)
2. copy and paste the following code to a new SAS editor window (File-> new program). */

```
OPTIONS LINESIZE=75 PAGESIZE=54 NODATE PAGENO=1;
DM "output;clear;log;clear"; *this will clear output and log window;

LIBNAME SASPRAC 'E:\SASCLASS';

DATA SASPRAC.ONE;
 INFILE 'E:\SASCLASS\prac1.dat' FIRSTOBS=2 MISSOVER;
 LENGTH NAME $ 16;
 INPUT @1 NAME $16.  @17 DOB MMDDYY8. Height Weight @33 Gender $ @36 Salary @43 Edu $;
            *Make sure the column numbers and modify the numbers accordingly;
 LABEL DOB='DATE OF BIRTH';
 LABEL Edu='Education';
 RUN;
PROC PRINT;  *if DATA= is missing, SAS uses the most recent data;
 RUN;

PROC FORMAT;
 VALUE $edulevel 'HS'= 'High School or below'
                 'C' = 'College or above';
 VALUE $Sex      'F' = 'Female'
                 'M' = 'Male' ;
 VALUE incomelevel     0 -< 50000 = 'Low'
                   50000 -< 100000 = 'Mid'
                   100000 - HIGH = 'High' ;

PROC Print label data=SASPRAC.ONE;
  FORMAT DOB WORDDATE18. Gender $Sex. Salary DOLLAR11.2 Salary incomelevel. Edu
$edulevel.;
  TITLE 'SAS Data Step Class Practice';
run;

PROC Contents data=SASPRAC.ONE; run;

DATA _NULL_; SET SASPRAC.ONE;
 Newsalary = salary * 1.05;
 Newmonth = Newsalary/12;
 FILE 'C:\teaching\SAS\pract1_out.dat' PRINT NOTITLE;
 PUT @3 "Salary raise notice for " Name /
     @3 55*'='  /
     @5 "Your Annual Salary for next year will be" Newsalary Dollar11.2 /
     @5 "Your new monthly wage of  "  Newmonth Dollar11.2  /
     @5 "will appear on your March paycheck." /
     @3 55*'='  /
     // ;
 PUT _PAGE_;
 RUN;
```

/* 3. Run the code. Detail discussion will be given during class*/

Some options in INFILE statement

```
DATA yourdataname;
  INFILE 'E:\foldername\dataname.dat' <options>;
  INPUT var1 var2 … ;
```

MISSOVER : To tell SAS there are missing data at end of the data line (see the inclass-prac). Without this option SAS will go to the next line and keep reading the variable's value. That means SAS will not read the missing data as missing.

TRUNCOVER: Similar to MISSOVER but can be used with column or formatted input.

DLM = 'delimiter' : In case data values are separated by a delimiter other than spaces. For example, DLM = ',' tells that data values are separated by comma.

DSD : DSD can be used if character values are placed in quotes (example, "*Lastname,firstname*"). Many cases data come with this format (for example, CVS data from Excel). DSD option treats two consecutive commas as a missing value. (see example below)

FIRSTOBS = *n* : Some data come with header and/or column names. You can have SAS to start reading the data at *n*th line.

OBS = *n* : In case you want to read only a part of data. With this option, SAS will read *n* data lines. This option can be used with FIRSTOBS option.

```
/*** Example5 (Delimited Data) ***/
/* Assume this data saved as 'E:\SASCLASS\example5.dat'

MS PROGRAM CANDIDATE RECORD
DATE 12/23/08
STATISTICS PROGRAM
"Cathy, Wood", Senior, Comp SCI, 3.48, 1270
"Mike, Michaski", Junior, SCI, 2.96,
"Henry, Wood", Senior, COMP ENG, 3.04, 1380
"Kim, Andy", Senior, Math, 3.25, 1250
"Mueller, Kay", Junior, APP STAT, 3.65, 1320

*/
LIBNAME SASPRAC 'E:\SASCLASS';
DATA SASPRAC.student;
INFILE 'E:\SASCLASS\example5.dat' FIRSTOBS=4 DLM=',' DSD MISSOVER;
LENGTH name $ 16 Major $ 10;
INPUT Name $ Grade $ Major $ GPA GRE;
RUN;

PROC PRINT DATA= SASPRAC.student; RUN;

/* Run the code without DSD option and see what happens */

/*****************************************************/
```

```
/* Data step above can be written as below **************/

DATA SASPRAC.student;
INFILE 'E:\SASCLASS \example5.dat' FIRSTOBS=4 DLM=',' DSD MISSOVER;
INPUT Name : $16.  Grade $ Major $ GPA GRE; RUN;

PROC PRINT DATA= SASPRAC.student; RUN;

/*****************************************************************
Note the colon after the variable 'Name'. This tells SAS to read the
next variable when the delimiter (comma this case) is encountered.
Without this, SAS will read 16 characters past the delimiter. Try
the code without the colon and see.
*****************************************************************/
```

NOTE: INFILE options can be used with DATALINES statement. For example, you can still use DLM option with DATALINES. See example below.

```
/*** Example5 (revisit) ***/

DATA SASPRAC.student;
INFILE datalines FIRSTOBS=4 DLM=',' DSD MISSOVER;
LENGTH name $ 16 Major $10;
INPUT Name $ Grade $ Major $ GPA GRE;

DATALINES;
MS PROGRAM CANDIDATE RECORD
DATE 12/23/08
STATISTICS PROGRAM
"Cathy, Wood", Senior, Comp SCI, 3.48, 1270
"Mike, Michaski", Junior, SCI, 2.96,
"Henry, Wood", Senior, COMP ENG, 3.04, 1380
"Kim, Andy", Senior, Math, 3.25, 1250
"Mueller, Kay", Junior, APP STAT, 3.65, 1320
;
RUN;

PROC PRINT DATA= SASPRAC.student; RUN;
```

SAS FUNCTIONS (Excerpted from SAS Online Documentation)

Arithmetic Functions

ABS(argument)	returns absolute value
DIM<n>(array-name)	returns the number of elements in a one-dimensional array or the number of elements in a specified dimension of a multidimensional array. **n** specifies the dimension, in a multidimensional array, for which you want to know the the number of elements.
DIM(array-name,bound-n)	returns the number of elements in a one-dimensional array or the number of elements in the specified dimension of a multidimensional array **bound-n** specifies the dimension in a multidimensional array, for which you want to know the number of elements.
HBOUND<n>(array-name)	returns the upper bound of an array
HBOUND(array-name,bound-n)	returns the upper bound of an array
LBOUND<n>(array-name)	returns the lower bound of an array
LBOUND(array-name,bound-n)	returns the lower bound of an array
MAX(argument,argument, ...)	returns the largest value of the numeric arguments
MIN(argument,argument, ...)	returns the smallest value of the numeric arguments
MOD(argument-1, argument-2)	returns the remainder
SIGN(argument)	returns the sign of a value or 0
SQRT(argument)	returns the square root

Character Functions

BYTE(n)	returns one character in the ASCII or EBCDIC collating sequence where **n**is an integer representing a specific ASCII or EBCDIC character
COLLATE(start-position<,end-position>) | (start-position<,,length>)	returns an ASCII or EBCDIC collating sequence character string
COMPBL(source)	removes multiple blanks between words in a character string
COMPRESS(source<,characters-to-remove>)	removes specific characters from a character string
DEQUOTE(argument)	removes quotation marks from a character value
INDEX(source,excerpt)	searches the source for the character string specified by the excerpt
INDEXC(source,excerpt-1<, ... excerpt-n>)	searches the source for any character present in the excerpt
INDEXW(source,excerpt)	searches the source for a specified pattern as a word
LEFT(argument)	left-aligns a SAS character string
LENGTH(argument)	returns the length of an argument
LOWCASE(argument)	converts all letters in an argument to lowercase

```
QUOTE(argument)              adds double quotation marks to a character
                             value
RANK(x)                      returns the position of a character in the
                             ASCII or EBCDIC collating sequence
REPEAT(argument,n)           repeats a character expression
REVERSE(argument)            reverses a character expression
RIGHT(argument)              right-aligns a character expression
SCAN(argument,n<,delimiters>) returns a given word from a character
                             expression
SOUNDEX(argument)            encodes a string to facilitate searching
SUBSTR(argument,position<,n>)=characters-to-replace   replaces
                             character value contents
var=SUBSTR(argument,position<,n>) extracts a substring from an argument.
                             (var is any valid SAS variable name.)
TRANSLATE(source,to-1,from-1<,...to-n,from-n>)  replaces specific
                             characters in a character expression
TRANWRD(source,target,replacement) replaces or removes all occurrences
                             of a word in a character string
TRIM(argument)               removes trailing blanks from character
                             expression and returns one blank if the
                             expression is missing
TRIMN(argument)              removes trailing blanks from character
                             expressions and returns a null string if
                             the expression is missing
UPCASE(argument)             converts all letters in an argument to
                             uppercase
VERIFY(source,excerpt-1<,...excerpt-n) returns the position of the
                             first character unique to an expression
```

Date and Time Functions

```
DATDIF(sdate,edate,basis)  returns the number of days between two dates
DATE()                     returns the current date as a SAS date value
DATEJUL(julian-date)       converts a Julian date to a SAS date value
DATEPART(datetime)         extracts the date from a SAS datetime value
DATETIME()                 returns the current date and time of day
DAY(date)                  returns the day of the month from a SAS date
                           value
DHMS(date,hour,minute,second) returns a SAS datetime value from date,
                           hour, minute, and second
HMS(hour,minute,second)    returns a SAS time value from hour, minute,
                           and second
HOUR(<time | datetime>)    returns the hour from a SAS time or datetime
                           value
INTCK('interval',from,to)  returns the number of time intervals in a
                           given time span
INTNX('interval',start-from,increment<,'alignment'>)  advances a date,
                           time, or datetime value by a given interval,
                           and returns a date, time, or datetime value
JULDATE(date)              returns the Julian date from a SAS date
                           value
MDY(month,day,year)        returns a SAS date value from month, day,
                           and year values
MINUTE(time | datetime)    returns the minute from a SAS time or
                           datetime value
MONTH(date)                returns the month from a SAS date value
```

QTR(date)	returns the quarter of the year from a SAS date value
SECOND(time | datetime)	returns the second from a SAS time or datetime value
TIME()	returns the current time of day
TIMEPART(datetime)	extracts a time value from a SAS datetime value
TODAY()	returns the current date as a SAS date value
WEEKDAY(date)	returns the day of the week from a SAS date value
YEAR(date)	returns the year from a SAS date value
YRDIF(sdate,edate,basis)	returns the difference in years between two dates
YYQ(year,quarter)	returns a SAS date value from the year and quarter

Mathematical Functions

AIRY(x)	returns the value of the AIRY function
DAIRY(x)	returns the derivative of the AIRY function
DIGAMMA(argument)	returns the value of the DIGAMMA function
ERF(argument)	returns the value of the (normal) error function
ERFC(argument)	returns the value of the (normal) error function
EXP(argument)	returns the value of the exponential function
GAMMA(argument)	returns the value of the GAMMA function
IBESSEL(nu,x,kode)	returns the value of the modified bessel function
JBESSEL(nu,x)	returns the value of the bessel function
LGAMMA(argument)	returns the natural logarithm of the GAMMA function
LOG(argument)	returns the natural (base e) logarithm
LOG2(argument)	returns the logarithm to the base 2
LOG10(argument)	returns the logarithm to the base 10
TRIGAMMA(argument)	returns the value of the TRIGAMMA function

Noncentrality Functions

CNONCT(x,df,prob)	returns the noncentrality parameter from a chi-squared distribution
FNONCT(x,ndf,ddf,prob)	returns the value of the noncentrality parameter of an F distribution
TNONCT(x,df,prob)	returns the value of the noncentrality parameter from the student's t distribution

Probability and Density Functions

CDF('dist',quantile,parm-1,...,parm-k)	computes cumulative distribution functions
LOGPDF|LOGPMF('dist',quantile,parm-1,...,parm-k)	computes the logarithm of a probability density (mass) function. The two functions are identical.
LOGSDF('dist',quantile,parm-1,...,parm-k)	computes the logarithm of a survival function
PDF|PMF('dist',quantile,parm-1,...,parm-k)	computes probability density (mass) functions
POISSON(m,n)	returns the probability from a POISSON distribution
PROBBETA(x,a,b)	returns the probability from a beta distribution

```
PROBBNML(p,n,m)        returns the probability from a binomial
                       distribution
PROBCHI(x,df<,nc>)     returns the probability from a chi-squared
                       distribution
PROBF(x,ndf,ddf<,nc>)  returns the probability from an F distribution
PROBGAM(x,a)           returns the probability from a gamma distribution
PROBHYPR(N,K,n,x<,r>)  returns the probability from a hypergeometric
                       distribution
PROBMC                 probabilities and critical values (quantiles)
                       from various distributions for multiple
                       comparisons of the means of several groups.
PROBNEGB(p,n,m)        returns the probability from a negative binomial
                       distribution
PROBBNRM(x,y,r)        standardized bivariate normal distribution
PROBNORM(x)            returns the probability from the standard normal
                       distribution
PROBT(x,df<,nc>)       returns the probability from a Student's t
                       distribution
SDF('dist',quantile,parm-1,...,parm-k)    computes a survival function
```

Quantile Functions

```
BETAINV(p,a,b)       returns a quantile from the beta distribution
CINV(p,df<,nc>)      returns a quantile from the chi-squared
                     distribution
FINV(p,ndf,ddf<,nc>) returns a quantile from the F distribution
GAMINV(p,a)          returns a quantile from the gamma distribution
PROBIT(p)            returns a quantile from the standard normal
                     distribution
TINV(p,df<,nc>)      returns a quantile from the t distribution
```

Sample Statistics Functions

```
CSS(argument,argument,...)     returns the corrected sum of squares
CV(argument,argument,...)      returns the coefficient of variation
KURTOSIS(argument,argument,...) returns the kurtosis (or 4th moment)
MAX(argument,argument, ...)    returns the largest value
MIN(argument,argument, ...)    returns the smallest value
MEAN(argument,argument, ...)   returns the arithmetic mean (average)
MISSING(numeric-expression | character-expression)    returns a numeric
                               result that indicates whether the
                               argument contains a missing value
N(argument,argument, ....)     returns the number of nonmissing values
NMISS(argument,argument, ...)  returns the number of missing values
ORDINAL(count,argument,argument,...) returns the largest value of a
                               part of a list
RANGE(argument,argument,...)   returns the range of values
SKEWNESS(argument,argument,argument,...) returns the skewness
STD(argument,argument,...)     returns the standard deviation
STDERR(argument,argument,...)  returns the standard error of the mean
SUM(argument,argument,...)     returns the sum
USS(argument,argument,...)     returns the uncorrected sum of squares
VAR(argument,argument,...)     returns the variance
```

State and ZIP Code Functions

```
FIPNAME(expression)  converts FIPS codes to uppercase state names
FIPNAMEL(expression) converts FIPS codes to mixed case state names
```

```
FIPSTATE(expression) converts FIPS codes to two-character postal codes
STFIPS(postal-code)  converts state postal codes to FIPS state codes
STNAME(postal-code)  converts state postal codes to uppercase state
                     names
STNAMEL(postal-code) converts state postal codes to mixed case state
                     names
ZIPFIPS(zip-code)    converts ZIP codes to FIPS state codes
ZIPNAME(zip-code)    converts ZIP codes to uppercase state names
ZIPNAMEL(zip-code)   converts ZIP codes to mixed case state names
ZIPSTATE(zip-code)   converts ZIP codes to state postal codes
```

Trigonometric and Hyperbolic Functions

```
ARCOS(argument) returns the arccosine
ARSIN(argument) returns the arcsine
ATAN(argument)  returns the arctangent
COS(argument)   returns the cosine
COSH(argument)  returns the hyperbolic cosine
SIN(argument)   returns the sine
SINH(argument)  returns the hyperbolic sine
TAN(argument)   returns the tangent
TANH(argument)  returns the hyperbolic tangent
```

Truncation Functions

```
CEIL(argument)                 returns the smallest integer that is
                               greater than or equal to the argument
FLOOR(argument)                returns the largest integer that is less
                               than or equal to the argument
FUZZ(argument)                 returns the nearest integer if the
                               argument is within 1E-12
INT(argument)                  returns the integer value
ROUND(argument,round-off-unit) rounds to the nearest round-off unit
TRUNC(number, length)          truncates a numeric value to a specified
                               length
```

Function used with INPUT statement

**varname $** A dollar sign ($) following a variable name means that the variable has character values. If character values have length of over 8, you can specify with the LENGTH statement. If you don't specify the length and values exceed 8 characters, SAS will truncate the value at the 8th character.

varname $k. SAS will read the variable values up to *k* (greater than 8) characters. The length can also be specified using LENGTH statement before the INPUT statement.

@n Most basic data format is free format input that data is separated by at least one space between values. If the data is column formatted you should use the column controller @. For example, `INPUT @5 score;` tells SAS to move the column pointer to the 5th column and read *score*.

+m This moves the column point *m* spaces to the right.

@@ If there are multiple observations in a line you must add @@ (double trailing sign) at the end of the INPUT statement so that SAS continue to read the next observation. Without this, once SAS finishes reading an observation, the pointer will move to the next line. For example,

```
DATA prac;
INPUT Hts Wts @@;
DATALINES;
158 47 178 79 180 98 163 51
;
PROC PRINT; RUN;
```

varname : The colon (:) following the variable name tells SAS to read the next variable when the delimiter (comma this case) is encountered. Without this, SAS will read characters past the delimiter. Try the code without the colon and see. This function can be used with DLM option in INFILE statement.

varname n-m This tell SAS the variable values are in column *n* to *m*. For example,

```
INPUT name $ 1-16 ;
```

#n If an observation consists of more than one line, this row pointer can be used (see example6).

varname $ & This indicates the character variable with a blank. SAS will read the next variable with double or more spaces (see example7).

```
/* Example6 (Column arranged Data) */

   /* Assume this data saved as 'E:\SASCLASS\example6.dat'

   Cathy Heywood        12/14/87 F      125 7th St      Long Beach
   Senior Comp SCI      3.48  1270
   Mike Michaski        10/03/89 M      25 Harbor Dr    Anaheim
   Junior SCI           2.96
   Henry Wood           01/25/86 M      16 Euclid Dr    Fullerton
   Senior COMP ENG      3.04  1380
   Kim Andy             04/17/91 F      568 E st        Orange
   Senior Math          3.25  1250
   Mueller Kay          11/23/82 F      5847 E2nd St    Irvine
   Junior APP STAT      3.65  1320
   */

   DATA SASPRAC.example6 (DROP = Add City DOB_c Birthyear);
      INFILE 'E:\SASCLASS\example6.dat' MISSOVER TRUNCOVER;
      INPUT #1 Name $15. @17 DOB MMDDYY8. @17 DOB_c $ @26 Gender $1. @28
               Add $12. @41 City $11.
             #2 Grade $ @8 Major $8. @17 GPA GRE;
      Format DOB date9.;

      Age=INT(YRDIF(DOB,TODAY( ), 'Actual'));

      Address=TRIM(Add) || ', ' || City;

      Length Name_g $17.;
      Name_g=LEFT(Name) || '/' || Gender;

      Birthyear=SCAN(DOB_c,3);   *Using SCAN, Birthyear is a character var;
      YOB=INPUT(Birthyear,f2.)+1900; *INPUT coverts a character var to numeric;
                                     *YOB is now numeric var so that you can;
                                     *do numerical calculations;

        /* You can also do YOB=INPUT(SCAN(DOB_c,3))+1900
                        or YOB=INPUT(SUBSTR(DOB_c,7,2))+1900 */

      LABEL Name_g='Name/Gender' DOB='Date of Birth' YOB='Year of Birth';
   RUN;

   PROC PRINT LABEL DATA= SASPRAC.example6; RUN;

   DATA SASPRAC.example6_1; set SASPRAC.example6;
      if Grade = 'Senior' & Age > = 20; RUN;

   PROC PRINT DATA= SASPRAC.example6_1; RUN;


   /* Example7 (example $&)*/

   DATA SASPRAC.example6 ;
      INFILE 'C:\teaching\SAS\example6.dat' MISSOVER TRUNCOVER;
      Length Name $ 15 Add $ 12 City $ 11;
      INPUT #1 Name $ & @17 DOB MMDDYY8. @26 Gender $1. @28 Add $ &
            @41 City $ &
            #2 Grade $ @8 Major $8. @17 GPA GRE;
      Format DOB worddate12.; RUN;

   PROC PRINT DATA= SASPRAC.example6; RUN;
```