SAS Statements: All SAS statements end with a semicolon, ";". A statement may occur on one line, or run across several lines. Several statements can also be placed on the same line, as long as each statement ends with a semicolon. Statements need not start in the first column, so some lines can be intended for easier reading. But, all statements must end in a semicolon.

The Comment statement: Any SAS statement that starts with an asterisk will be a comment statement. Comment statements do not result in any output. They are useful to label your SAS LOG in order to determine which program was run and on what data, and to provide explanations of programming steps for yourself and others. Comments may be placed anywhere in your SAS program. The following lines are statements that you might include at the start of your program.

```
**EXST7015 EXAMPLE 1****************************************;
*** Simple linear regression example run in SAS (1/1/1960) ***;
*** James P. Geaghan ***;
************************************************************;
```

A second type of comment is not a SAS statement. This type of comment employs special brackets (/* and */), and is created by surrounding any section of the program with **/*** in the front and ***/** after the end. Anything included within these brackets will not be executed. As a result, the brackets can not only be used to add comments anywhere in the program, but also to deactivate a part of a program. For example, the statements

```
PROC SORT DATA=ONE; BY A B C; RUN;
PROC PRINT DATA=ONE; VAR A B C D E F; RUN;
```

can be "turned off" by adding the brackets.

```
/*
PROC SORT DATA=ONE; BY A B C; RUN;
PROC PRINT DATA=ONE; VAR A B C D E F; RUN;
*/
```

This can be useful to temporarily deactivate programming segments without actually deleting the statements.

Note that these two types of comments can be used anywhere in the program sections of SAS, but not within in the data listing if the data is included in the program.

Options: The options statement in SAS can modify the size of your output. For example, when SAS is printed directly to the printer a page size of 51 lines (PS=51) should fit on one page. A page size of 60 might fill a page and then print a few lines on a second page (if the printer is set for fewer than 60 lines). Do not request a page size larger than the printer will accommodate unless you plan to save SAS output to a file then reformat the output in a word processor. Other options include line size (LS=78 will fit nicely on most printed pages), nocenter, nodate and nonumber (to suppress page numbering) or PAGENO=1 to insure that the page numbering starts at 1 when the program is rerun.

```
OPTIONS PS=51 LS=78 NOCENTER NODATE NONUMBER;
```

Options statements can be included anywhere in the program after a RUN statement. If placed before the RUN, it will cancel previous OPTIONS statements. For example, all options in the following programming segment would be executed.

```
OPTIONS PS=51 LS=78 NOCENTER NODATE NONUMBER;
Proc print data=one; run;
OPTIONS PS=31 LS=80;
Proc plot data=one; plot x*y; run;
OPTIONS PS=52;
```

The program above would first print with a page size of 51 and a line size of 78, then execute a graph with a page size of 31 and line size of 80. If, however, the first RUN statement was not included, SAS would read past the PROC PRINT and read the second options statement before printing and the page size for the PROC PRINT would be 31. Note that after the PLOT statement and its RUN, the options are reset so that any additional output will have a page size of 52 and the line size remains at 80.

SAS Statements consist of keywords (words that instruct SAS on the operations and options that you want to execute) and user provided words (not to exceed 8 characters). For example, a SAS DATA statement might look like "DATA ONE;". The word DATA is a SAS keyword and tells SAS that the program is starting a DATA section. The user specifies the word ONE. Any 8 characters could be used.

The DATA statement: The "DATA" statement is necessary to develop a SAS data set. All of your programs will include a DATA statement. The DATA section of the program will usually include the following statements: **DATA** followed by an **INFILE** or **SET** statement followed by an **INPUT** statement. The INPUT statement actually reads the data so that you can process it. The DATA statement tells SAS what you wish to call this data set and the INFILE or SET statement tells SAS where the data is coming from and its characteristics. For example, the statements

```
DATA ONE; INFILE CARDS MISSOVER;
INPUT A $ 1-5 B C D $;
CARDS; RUN;
data records would go here
;
```

The DATA statement in this case tells SAS that you are creating a DATA set that will be called "ONE". **The INFILE statement** informs SAS that the data is to be found on "cards", meaning that it is included as part of the program following a "CARDS;" statement. An alternative input type is to have the data in an external file, apart from the program. In this case the INFILE statement would have a reference name for the file and the program would have to have an additional statement called a FILENAME statement to tell where the data could be found. For example, if the data were to be in an external file on the hard drive (C:\mydatafiles) and was called "newdata.dat", the filename and INFILE statements would be

```
FILENAME fileneme 'C:\MYDATAFILES\NEWDATA.DAT';
DATA ONE; INFILE filename MISSOVER;
```

The file name can be any name you wish to assign up to 8 letters long.

The INPUT statement informs SAS of the variables to be read and some of their characteristics. For example, the INPUT statement **`INPUT A $ 1-5 B C D $;`** from the program above instructs SAS to read a variable "A" which is a character variable (as indicated by the $) in columns 1 through 5 (indicated by the 1-5). Variables B and C follow variable A and are numeric (since they are not followed by a $) and the last variable, D, is another character variable. The values of the variables A, B, C and D should occur in the data records and be separate by blanks so SAS can find each one. Variable names must not exceed 8 characters. Some examples of acceptable names would are: `X Y DATE SIZE N32 M11 V3M4_6`.

The MISSOVER option is not always necessary, but it prevents SAS from going to the next line in the event that all variables are not found. For example, if the data set was as follows;

```
Fresh 7 3 Early
Dried 8 9 Early
Fresh 6 3
Dried 8 7 Late
Fresh 5 6 Late
```

then when SAS reached the third line and only found values for A, B and C it would go to the next line in order to find a value of D. The value of D would then be `Dried`, the first value in the 4th line. The MISSOVER option instructs SAS to place a value of *missing* for any values it cannot find on a line.

Examine the program below and answer the sample questions;

```
1          *** EXAMPLE 1 ********************
2          *** Data input, sort and print ***
3          ********************************;
4          OPTIONS PS=55 LS=77 NOCENTER NODATE NONUMBER;
5          DATA paintdry; INFILE CARDS MISSOVER;
6          INPUT status $ luster hardness timeofday $;
7          CARDS; RUN;
8          ;
9          PROC SORT; BY status luster hardness; RUN;
10         PROC PRINT; RUN;
```

Sample questions.

1. Where would the data be placed? (i.e. between what two lines).
2. How many lines will be printed on an output page?
3. What is the name of the SAS data set to be created?
4. How many variables are read from the data set?

5. Will SAS continue to the next line if it does not find values for all of the variables?
6. There is an error in the program. Can you find it?
7. Which line(s) are comments? Are all of these lines separate statements?

Procedures: While the DATA step is needed for data input and modifications, the processing of data is done in "procedures". Everything from listing the data to statistical analysis and graphics are done in procedures once the data is entered with the DATA step. A few basic procedures are discussed below.

PROC SORT: The data is frequently sorted in SAS either to enhance readability of printed output or to sort the data in ordered blocks so that SAS can process one block at a time. The SORT statement is always accompanied with a BY statement to instruct SAS which variables to use in ordering the data. The data will then be sorted in alphabetic (starting with A) or numeric order (starting with 0).

Suppose we had the following data set with variables called A (fresh or dried), B, C and D (early or late).

```
Fresh 7 3 Early
Dried 8 9 Early
Fresh 6 3
Dried 8 7 Late
Fresh 5 6 Late
```

In the program below the data is input to a SAS data set called ONE, sorted by A, B and C, and then printed.

```
DATA paintdry; INFILE CARDS MISSOVER;
INPUT status $ luster hardness timeoday $;
CARDS; RUN;
Fresh 7 3 Early
Dried 8 9 Early
Fresh 6 3
Dried 8 7 Late
Fresh 5 6 Late
;
PROC SORT; BY status luster hardness; RUN;
PROC PRINT; RUN;
```

The data listing would then read as follows.

```
Dried 8 7 Late
Dried 8 9 Early
Fresh 5 6 Late
Fresh 6 3
Fresh 7 3 Early
```

Since the data is sorted by status, luster and hardness, the listing has all values of **status** = Dried before all the values **status** = Fresh (alphabetically D comes before F). Then within the Dried and Fresh groups we have sorting by **luster**. In the case of **status** = Dried both values of **luster**

were 8, so this does not effect the output. However, values of **luster** were 5, 6 and 7 for **status** = Fresh and have been numerically ordered. Finally, we had **hardness** in the BY statement. Since both Dried and a value of 8 for B, these two observations were then sorted by **hardness**, so **hardness** = 7 comes before **hardness** = 9. Each of the Fresh lines had a different value of **luster**, so they were sorted by **luster** (because this came first, before **hardness**) and were not sorted by **hardness**.

Questions concerning the sample program above.

1. If there was a 6th line reading " `Fresh 6 2 Early` ", were would it fall in the sort order?
2. If there was s 6th line reading " `Cracked 7 3 Early` ", were would it fall in the sort order?

PROC PRINT: The print statement is used to obtain a listing of a data set. The procedure can be run by simply entering "PROC PRINT; RUN;". However, there are also a number of useful and sometimes necessary embellishments that control the operation of the PRINT statement. For example, if the input variables are "A B C D E F G", but only a few selected variables are to be printed and in a different order from the input statement, say "C E F A", then a VARIABLE statement (or VAR) would be needed. This statement would specify the variables and order to be printed (e.g. **`VAR C E F A;`**).

```
PROC PRINT; VAR C E F A; RUN;
```

There are also some important options that can be added to the SAS PRINT statement. Double spaced listing can be produced by the double option "PROC PRINT double;". In the event that we have created several data sets (e.g. ONE, TWO, THREE), we may need to tell SAS which one to print. If you do not specify, SAS will print the last data set created by the program. To specify a data set (e.g. data set "two") in PROC PRINT (and most other PROCs) enter "PROC PRINT DATA=two".

Enhancements to the DATA step and SAS program.

There are several things that can improve the readability and interpretability of your output. One is the TITLE statement. Title statements can be placed anywhere in the program and all output produced after the title statement will have the title at the top of the page. You can also have several title statements. For example, if we put the following title near the start of the program, the title will occur on all pages.

```
TITLE1 'James Geaghan - Assignment 1 : EXST7015':
```

All output produced by statements following this title would have the line,

```
James Geaghan - Assignment 1 : EXST7015
```

at the top of the page, unless we put in another TITLE1 statement. A second TITLE1 statement would replace the first.

To title individual procedures, in addition to the title line above, a TITLE2 statement would be used. For example, to add a second title lines "Data listing" to a PRINT procedure and "Scatter plot" to a plot procedure, place TITLE2 statements before the RUN statement for each procedure.

```
TITLE1 'James Geaghan - Assignment 1 : EXST7015';
TITLE2 'Data listing';
PROC PRINT; RUN;
TITLE2 'Scatter plot';
PROC PLOT; PLOT X*Y; RUN;
```

The data listing would then have the following titles at the top.

```
James Geaghan - Assignment 1 : EXST7015
Data listing
```

And the plot would have the following titles.

```
James Geaghan - Assignment 1 : EXST7015
Scatter plot
```

Note that the second TITLE2 replaces the first one, but does not effect the TITLE1. We could also have a TITLE3, TITLE4, and so forth up to TITLE9.

Another statement that can improve the readability and interpretability of your output is the LABEL statement. This statement serves to document the meaning of the variables in a data set and SAS will insert labels in a number of places in the output, facilitating interpretation. For example, for the data set created in a previous example we had the input statement,

```
INPUT X Y YEAR SEX $ PROFESN $;
```

We could add the following labels to explain the variables in more detail.

```
LABEL X = 'Age of participant (years)';
LABEL Y = 'Income ($ *1000)';
LABEL SEX = 'Sex as M or F';
LABEL Y = 'Profession';
```

Important note: *SAS does not distinguish between UPPER and lower case for programming statements. However, it will distinguish between cases <u>for input data</u> and for segments included in quotes (these are literal character strings), as in the TITLE and LABEL statements above. It is also very important that opening quotes be matched with a closing quote, or SAS will interpret your whole program as a literal string***.**

Sample program.

```
1          *** EXAMPLE 3 ********************;
2          *** Data input, sort and print ***;
3          ********************************;
4          OPTIONS PS=53 LS=79 NOCENTER NODATE NONUMBER;
5          DATA NEW3; INFILE CARDS MISSOVER;
6          INPUT day number type $ model $;
7          CARDS; RUN;
8          17 9 TRUCKS SEMI
9          18 8 TRUCKS SEMI
10         19 2 TRUCKS PICKUP
11         22 4 TRUCKS SEMI
12         16 3 CARS COUPE
13         17 2 CARS COUPE
14         18 3 CARS SEDAN
15         19 1 CARS SEDAN
16         22 5 CARS SEDAN
17         17 1 VANS 5DOOR
18         17 4 VANS 4DOOR
19         19 2 VANS 5DOOR
20         ;
21         PROC SORT DATA=NEW3; BY type model day number; RUN;
22         TITLE1 'My raw data is listed below';
23         PROC PRINT DATA=NEW3 double; VAR type model day number; RUN;
```

Questions about the program above,

1. How many observations are in the data set?
2. How many variables are in the data set?
3. What is the maximum number of lines that this program could print on a page of output?
4. If the data set is as indicated in the program, which data line will be printed **first** in the listing produced by proc print?
5. If the data set is as indicated in the program, which data line will be printed **last** in the listing produced by proc print?
6. Which variables are numeric and which are categorical?

PROC MEANS: Simple means can be readily obtained with the PROC MEANS statement. To get means for the variables X and Y, write the following program.

```
PROC MEANS DATA=ONE; VAR X Y; RUN;
```

By default the "PROC MEANS" statement would output the sample size (n), the mean, standard deviation, maximum and minimum of the variables analyzed. Other summary statistics can also be obtained. Some of those statistics and the key words are: N (n), STDERR (standard error), MEAN, MIN, MAX, SUM, STD (standard deviation), VAR (variance), USS (uncorrected sum of squares), and CSS (corrected sum of squares). Placing these options in the PROC MEANS statement will cause SAS to calculate and output only the listed statistics. For example, "PROC MEANS N MEAN STDERR USS; VAR X; RUN;" would cause SAS to calculate the mean, sample size, standard error and uncorrected sum of squares for the variable X.

Using the BY statement on a PROC. Frequently it is necessary to run a series of PROCs on different sections of the data set. For example, a BY statement could be used together with PROC MEANS to calculate means and variances separately for the different classes of vehicles (car, truck and van) for the data set below from Example 3.

```
17 9 TRUCKS SEMI
18 8 TRUCKS SEMI
19 2 TRUCKS PICKUP
22 4 TRUCKS SEMI
16 3 CARS COUPE
17 2 CARS COUPE
18 3 CARS SEDAN
19 1 CARS SEDAN
22 5 CARS SEDAN
17 1 VANS 5DOOR
17 4 VANS 4DOOR
19 2 VANS 5DOOR
```

First it is necessary to SORT the data BY the variable "type" which contains the type of vehicle. Then the MEANS procedure would be run with a "BY type;" statement.

```
PROC SORT DATA=ONE; BY type; RUN;
PROC MEANS DATA=ONE; BY type; VAR number day; RUN;
```

This would produce separate means for each vehicle type (car, truck, vans) for the quantitative variables (number and day). Note that when reading categorical variables each value is treated as a literal string and upper and lower case would not be considered equal. The variable values "TRUCK", "truck", and "Truck" would all be treated as different vehicle types.

The OUTPUT statement: It is often necessary to process output produced by SAS procedures. Many SAS procedures can produce output by means of an OUTPUT statement. An output statement has the following structure.

```
OUTPUT OUT=newname keyword=username keyword2=username2;
```

For example, to obtain output from the MEANS statement above and list it in a PRINT format instead of the usual PROC MEANS format, do the following.

```
PROC SORT DATA=NEW3; BY type; RUN;
PROC MEANS DATA=NEW3 NOPRINT; BY type; VAR number day;
OUTPUT OUT=THREE N=NNo DNo MEAN=NMEAN DMEAN VAR=NVAR DVAR; RUN;
TITLE1 'Outputted means are listed below';
PROC PRINT DATA=THREE; VAR TYPE NNo DMEAN NVAR DNo NMEAN DVAR; RUN;
```

Simple frequencies: Frequency tables can be made with a PROC FREQ statement. The FREQ statement counts the number of records in each category and presents the frequency of each record in a table.

If each record represents several observations, then a WEIGHT statement must be added in order to instruct SAS which variable on the data set represents the number of observations. Weight statements are used in many other procedures to calculate weighted statistics, such as weighted means (PROC MEANS and PROC UNIVARIATE) and weighted least squares analyses (PROC REG and PROC GLM).

```
PROC FREQ DATA=ONE; TABLE D*C; RUN;
PROC FREQ DATA=ONE; TABLE D*C; WEIGHT B; RUN;
```

Each cell in the frequency table will contain the frequency as well as the percent of the total table represented by that frequency, and the row and column percentages. The percentages can be suppressed by adding the options "NOPERCENT NOROW NOCOL" to the TABLE statement.

```
PROC FREQ DATA=ONE; TABLE D*C / nopercent norow nocol; RUN;
```

PROC FREQ can also be used to do Chi square tests. The Chi square test is requested with the CHISQ option and the individual cell contributions to the Chi square value is requested with a CELLCHI2 option on the table statement.

```
PROC FREQ DATA=ONE; TABLE D*C / chisq cellchi2; RUN;
```

Simple plots and charts: Scatter plots, histograms and other types of plots are available in various plotting procedures in SAS. Plotting on X and Y coordinates is done in PROC PLOT and bar charts are done in PROC CHART.

```
PROC PLOT DATA=ONE; PLOT Y*X; RUN;
PROC PLOT DATA=RESIDS; PLOT RESIDUAL*X; RUN;
PROC CHART DATA=ONE; VBAR C / TYPE=FREQ SUMVAR=A;
```

SAS functions: There are many functions available in SAS, including functions to calculate logarithms, trigonometric functions and functions for doing simple arithmetic and statistical calculations such as rounding numbers and calculating means of a group of numbers. Only one function will be discussed here as an illustration, this is the "MDY" SAS date function. The MDY stands for "**M**onth-**D**ay-**Y**ear" and it is used to combine these variables into a single variable representing a SAS date. A SAS date can then be "formatted" to present this information in a simpler form. One common format is "DATE7." which shows the date as a two-digit day, three-letter month and two-digit year (e.g. 07JUL93).