Introduction to SAS (old)

Statistical Software

Preliminaries

In order to use SAS, you must first obtain an account. Contact the ITS Computer Consulting Center in ITTC 36 (phone 319-273-5555) for an application.

Some useful DCL command strings

DCL stands for Digital Command language. The DCL prompt is the dollar sign. From it you can enter DCL command strings, and run SAS. DCL command strings are composed of commands, parameters, and qualifiers. A command is an instruction to the operating system. A parameter defines what the command is to operate on. A qualifier defines how that action will occur.

Here are some useful DCL command strings:

DIRECTORY -- list the files in your directory
PRINT <filespec>-- print a file to the printer
TYPE <filespec>--display a file on the screen
LOOK <filespec>and TYPE/PAGE <filespec>-- display a file on the screen, a page at a time
DELETE <filespec>-- delete a file
PURGE --delete all but most recent version of your files
HELP -- invoke the help screens
COPY <filespec> <filespec>--copy a file
ED <filespec>--enter the EVE text editor
LOGOUT --log off the system

All DCL commands may be abbreviated. All that is needed are enough letters to make it unique. Four characters are always sufficient. Complete file specifications <filespec>have the following form:

```
node::device:[directory]filename.type;version
```

The node is for networked computers and is "ACAD" for the DEC alpha at UNI. The device is a logical name for a disk pack. For faculty, this is FAC. For students, this is STU1 or STU2. The directory is your Username, or a subdirectory you have created. Note that the brackets are required.

Both filenames and types can be up to 39 letters. They may consist of letters, numbers, the underscore, the hyphen, and the dollar sign. File names may start with either a letter or a number, but not the underscore, hyphen, or dollar sign. Version numbers are not required in most instances. Usually, all that is needed is a filename and a filetype.

Running a SAS job

Typically, two files need to be created prior to running SAS. First, there needs to be a file that contains the data. The results

of a survey, for example, need to be taken from the survey forms and entered into a file.

The second file needed is a file that contains SAS commands. This is a program written in the SAS language. The program contains instructions as to how to read the data. It also contains procedures for producing results from the data.

These files are usually created using the editor. Once you have created these two files, you can then run SAS. To run the program, type:

```
$SAS programfile
```

.*SAS*; is the default file type for a SAS program.

A message will appear indicating the status of your job, like this:

```
Programfile (queue SAS$BATCH, entry 533) started on SAS$BATCH
```

Batch queue SAS$BATCH, on LARRY::

```
Jobname        Username       Entry          Status
-------        --------       -----          -------
RECYCLE        Howard         533            Executing

Programfile (queue SAS$BATCH, entry 999) started on SAS$BATCH

SAS job has been submitted to the proper batch queue.
You will be notified when it finishes.

Results are being written to: FAC:[HOWARD]RECYCLE.LOG and FAC:[HOWARD]RECYCLE.LIS
```

When the job is done, a message like this will appear:

```
Job programfile (queue SAS$BATCH, entry 999) completed
```

Note that the queue is on node "LARRY". This is because the SAS package is actually located on the VAX 3100 workstation, whose logical name is "LARRY". The DEC Alpa takes care of submitting and receiving the job from the 3100, so it should not be of a concern. However, if the VAX 3100 workstation is down, you cannot run SAS.

To find out where your job is in the queue, type

```
$ show queue sas$batch/all
```

To delete your job from the queue, type

```
$ delete/entry=nnn
where nnn is your entry number n the queue.
```

Two files are created when the job runs successfully. The first will have the name of "programfile.LOG" and will contain all of the messages from SAS as it processed your program. This file is always created. SAS will give you messages as to whether errors occurred in this file. The second file contains only the output requested. It will have the name of "programfile.LIS". If SAS stopped because of errors and did not produce any results, this file will not be created.

The listing file will contain carriage control characters, and is 132 columns wide unless you include an option statement in your SAS program, described below.

Use the LOOK utility to see the file. The commands "L" and "R" in the LOOK utility will move you left and right. A SAS program can be created in an 80 column wide width by including this line as the first line in your SAS program.

```
options linesize=80;
```

To print your program and results, type this:

```
$ PRINT programfile.SAS {prints the program}
$ PRINT programfile.LIS {prints the output}
$ PRINT programfile.LOG {prints the log}
```

Running a SAS job from a subdirectory

To run a SAS job from a subdirectory, normally both the SAS program and the data file accessed by that program must reside in that subdirectory. To run the program from that directory, type:

```
$ SAS programfile
```

For example, to run a program "RECYCLE.SAS" in a subdirectory called "research", you would type:

```
$ SAS RECYCLE
```

The files created by SAS, " programfile.LIS", and "programfile.LOG", would then be created in that subdirectory.

An example survey

We will use the following survey to illustrate how to write a SAS program.

Suppose that the city of Cedar Falls is interested in recycling. They have hired you as a consultant. You have developed the following survey to study the problem. The survey will be distributed to residents who have garbage collection.

1. If the city provided a recycling center at the landfill for newspapers and cans, would you use it?

a) Yes, for both newspapers and cans
b) Yes, only for newspapers
c) Yes, only for cans
d) No

2. If the city provided a recycling center at the College SquareMall for newspapers and cans, would you use it?

a) Yes, for both newspapers and cans
b) Yes, only for newspapers
c) Yes, only for cans
d) No

3. Would you be willing to pay an extra $20 per year in garbage fees to have the city pick up newspapers and cans at your home in order to recycle them?

a) Yes
b) No
c) Unsure

4. What is your sex?

a) Male
b) Female

5. What is your family income?

a) $0 to $10,000
b) $10,001 to $20,000
c) $20,001 to $30,000
d) over $30,000

Coding the data

To enter the survey on the computer so that it can be analyzed by SAS, we first need to code the data. We should decide on codes for each answer. Also we should consider that some questions may not be answered. This must be coded as well. Use numbers rather than letters, as computers are faster at computing with numbers. Here is a coding scheme for the questionaire above:

Question 1:

0 = No Response
1 = a) Yes, for both newspapers and cans
2 = b) Yes, only for newspapers
3 = c) Yes, only for cans
4 = d) No

Question 2:

0 = No Response
1 = a) Yes, for both newspapers and cans
2 = b) Yes, only for newspapers
3 = c) Yes, only for cans
4 = d) No

Question 3:

0 = No Response
1 = a) Yes
2 = b) No
3 = c) Unsure

Question 4:

0 = No Response
1 = a) Male
2 = b) Female

Question 5:

0 = No Response

1 = a) $0 to $10,000
2 = b) $10,001 to $20,000
3 = c) $20,001 to $30,000
4 = d) over $30,000

We then code each survey in preparation for putting the data into a file. Writing out the coded data on graph paper or computer coding sheets often helps new users to make fewer mistakes. Then we can enter the coded data into a file in the computer.

You should number each survey. Use a sequential number such as 1, 2, 3, etc. rather than some other key like a social security number. Write this number on the original survey. This allows checking between the coded data and the surveys.

Use one survey per line, unless it does not fit. In that case take several lines to code a survey. When your survey is small, you may find it useful to put blank spaces between each data item. For large surveys, it is usually preferable not to have blank spaces since more data items can fit on a line. You must be consistant about which column contains the answer to a question. In a SAS program with fixed column input, the data is described by using the column number.

For our example, we would enter the survey ID, followed by the five coded answers, like this:

001 1 2 1 1 4
002 1 1 1 2 3
003 4 4 2 1 2
004 2 3 1 1 4
005 1 1 1 1 1
006 1 1 1 1 4
007 2 2 2 2 1

Survey #3 in this coding has responded d) to question 1, d) to question 2, b) to question 3, a) to question 4, and d) to question 5.

Entering data using the EVE text editor

To enter the EVE Text Editor on the DEC Alpha, type ED followed by the file name. If we wanted to call our data file "RECYCLE.DAT", then we would type this:

```
$ EDIT RECYCLE.DAT
```

To start entering data, just start typing. The [End of file] marker will move down to the next line. To access the command line, press the <DO> key. At the bottom of your screen you will see this:

```
Command:
```

```
Here you can type in any of the EVE commands. There are also keys that
are useful in editing. Here are some of the most useful commands and
keys (keys shown in brackets):
```

```
-access the command line <Do>
-access the HELP screens <Help> or HELP
-turn insert on/off <CRTL/A> or <F14>
-delete a character <comma on numeric pad>
-delete a word <minus on numeric pad>
-delete a line <PF4 on numeric pad>
-save the file and exit <Do> EXIT or <CTRL/Z>
-quit without saving <Do> QUIT
```

Note that since each survey can be coded using one line on the file, there would be as many lines in this file as there are questionaires. When you enter the data into a file, do not enter any descriptions or labels in this file. If we did enter descriptions, SAS would try to read these descriptions as data, and an error would occur. There should be no blank lines in this file.

When you are done entering the data, make sure that the [End of file] marker is directly below the last line of data, then press the <Do> key and type EXIT. You will then return to the DCL prompt($).

Creating a SAS program

A SAS program contains the commands to interpret the data and produce the statistics desired. A SAS program to read the data and print it would be this:

```
/* RECYCLE.SAS SAS Program written by Mary Howard.
This program does the analysis of a survey on recycling.
Written to demonstrate the use of SAS.   */

options linesize=80;

data dataset1;
infile 'recycle.dat';
INPUT id 1-3 q1 5 q2 7 q3 9 q4 11 q5 13;

proc print;
var id q1-q5;
```

Statements enclosed in /* and */ are comments. Comments in SAS may extend over more than one line. Comments are useful in documenting the name of the file containing the program, the date written, and the purpose of the program. When you have many programs on your disk, these comments are useful in determing where the program is and what it does.

The code starting with "DATA" is the data step. A data step in SAS defines a SAS data set that SAS procedures can use to analyze the data. A temporary work file is created during the data step, that will exist for the life of the program.

A data step often includes more than just defining the data in SAS. Assigning of labels to variables and values of variables, and computations of new variables are also included in the data step. When SAS processes the data step, it executes the entire data step for each observation from the input source(s), until there are no more observations to be read.

It is possible to reference more than one SAS data set in a program, which is why we give each set a name. It is also possible to save SAS data sets on disk so that they can be referred to by another SAS program.

If you are familiar with SPSS, but new to SAS, a SAS data set is very similar to a SPSS system file.

In the example above, we name the data set "DATASET1". The INFILE statement identifies an external, non-SAS file that

is to be read using the INPUT statement. The simplest form of this statement is to put the name of the file to be read between quotes. In the example above, our data file has the name 'RECYCLE.DAT', so our INFILE statement is this:

```
infile 'recycle.dat';
```

The INPUT statement describes the values in an input record and assigns input values to corresponding SAS variables. In the above example column input is used, where the input record is described by column number. In the above example, the value for the variable "Q1" is located in column 5 of the input record. SAS names, including variable names and data set names, can be up to 8 characters long. The first character must be a letter or underscore. Later characters can be letters, numbers, or underscores.

PROC PRINT is a procedure that prints the data. The variable names will be displayed at the top of the columns. It would produce output like this:

```
OBS ID Q1 Q2 Q3 Q4 Q5
1  1  1  2  1  1  4
2  2  1  1  1  2  3
3  3  4  4  2  1  2
4  4  2  3  1  1  4
5  5  1  1  1  1  1
6  6  1  1  1  1  4
7  7  2  2  2  2  1
```

Indentation and syntax in SAS

In SAS, each statement in a DATA or PROC step ends in a semicolon. Statements need not start in any column, and more than one statement may appear on a line. One statement may also be extended over several lines, where a semicolon would appear only at the end of the statement.

A convention we will use here is to put the procedure name on a line by itself, except for procedure options, followed by each procedure statement indented on lines following it, like this:

```
proc procedurename options;
statement;
statement;
statement;

proc procedurename options;
statement;
statement;
statement;
statement;

proc procedurename options;
statement;
```

Note that sometimes statements are required in a procedure or are related to each other. Consult the SAS User's Guide if you are unsure. The convention that is used here is to put one statement on a line (unless it does not fit). Also, we will write out the entire statement, even if it is possible to abbreviate parts of it. It is felt that some of the possible abbreviations make the program hard to read and understand for new users. Use blank lines between procedures. This also makes code easier to read.

Frequency tables

A frequency table shows the number of persons that responded in a certain way on a question. It also shows percentages. To include frequency tables in our program, enter the editor with the program and add these lines to the program:

```
proc freq:
tables q1 q2 q3 q4 q5;
```

On the output listing we would find five frequency tables. Each would look somewhat like this:

```
Q1

CUMULATIVE   CUMULATIVE
Q1    FREQUENCY   PERCENT   FREQUENCY    PERCENT
-------------------------------------------------
1           4     57.1            4       57.1
2           2     28.6            6       85.7
4           1     14.3            7      100.0
```

Here we see that the value 1 occurs 4 times, the value 2 occurs 2 times, and the value 4 occurs 1 time for question 1.

Labels for values

Adding labels for values can improve the readability of output. In frequency tables in SAS, value labels provide a description rather than the value on the frequency table. To add labels to the SAS program, change the SAS program to this:

```
/* RECYCLE.SAS SAS Program written by Mary Howard on 3/22/90.
This program does the analysis of a survey on recycling.
Written to demonstrate the use of SAS on the DEC Alpha.   */

proc format;
value f1fmt 0 = 'No Response'
1 = 'a) Yes, for both newspapers and cans'
2 = 'b) Yes, only for newspapers'
3 = 'c) Yes, only for cans'
4 = 'd) No';
value f2fmt 0 = 'No Response'
1 = 'a) Yes'
2 = 'b) No'
3 = 'c) Unsure';
value f3fmt 0 = 'No Response'
1 = 'a) Male'
2 = 'b) Female';
value f5fmt 0 = 'No Response'
1 = 'a) $0 to $10,000'
2 = 'b) $10,001 to $20,000'
3 = 'c) $20,001 to $30,000'
4 = 'd) over $30,000';

data dataset1;
infile 'recycle.dat';
input id 1-3 q1 5-5 q2 7-7 q3 9-9 q4 11-11 q5 13-13;
```

```
format q1-q2 f1fmt. q3 f2fmt. q4 f3fmt. q5 f4fmt.;

proc print;
var id q1-q5;
format id q1-q5 3.0;

proc freq;
tables q1 q2 q3 q4 q5;
```

In the above example, PROC FORMAT defines formats for values, but these formats are not actually assigned to variables until the data step. This allows us to use the same format for more than one variable, as in the case of Q1 and Q2 in our example. Note in the data step that when variables are assigned to formats, a period follows the name of each format. Format names must be 8 characters or less. They may include letters and numbers, but must both start and end with a letter.

Although up to 40 characters are allowed for value labels, some procedures use fewer. PROC FREQ uses only 16. You may wish to modify labels accordingly in your own work.

PROC PRINT has also changed. A specification has been added instructing SAS to print the values in 3 spaces each, with no decimals. If we had not added this, SAS would have printed the labels defined in PROC FORMAT rather than the actual values.

Labels for variables

Variable labels help improve the readability of variable names, since variable names must be no more than 8 characters. Variable labels can be up to 40 characters, though some procedures use fewer. To code variable labels in our SAS program, we would change the data step code to be this:

```
data dataset1;
infile 'recycle.dat';
input id 1-3 q1 5-5 q2 7-7 q3 9-9 q4 11-11 q5 13-13;
format q1-q2 f1fmt. q3 f2fmt. q4 f3fmt. q5 f4fmt.;
label
q1 = '1. Center at Landfill'
q2 = '2. Center at Mall'
q3 = '3. Willing to pay extra'
q4 = '4. Sex'
q5 = '5. Family Income';
```

Note that the statement keyword is "LABEL", not "LABELS".

The frequency tables with value and variable labels added would look like this:

```
1. Center at Landfill

CUMULATIVE   CUMULATIVE
Q1          FREQUENCY   PERCENT   FREQUENCY    PERCENT
-----------------------------------------------------------------
a) Yes, for both          4     57.1            4       57.1
b) Yes, only for          2     28.6            6       85.7
d) No                     1     14.3            7      100.0
```

2. Center at Mall

Q2	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
a) Yes, for both	3	42.9	3	42.9
b) Yes, only for	2	28.6	5	71.4
c) Yes, only for	1	14.3	6	85.7
d) No	1	14.3	7	100.0

3. Willing to pay extra

Q3	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
a) Yes	5	71.4	5	71.4
b) No	2	28.6	7	100.0

4. Sex

Q4	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
a) Male	5	71.4	5	71.4
b) Female	2	28.6	7	100.0

5. Family Income

Q5	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
a) $0 to $10,000	2	28.6	2	28.6
b) $10,001 to $2	1	14.3	3	42.9
c) $20,001 to $3	1	14.3	4	57.1
d) over $30,000	3	42.9	7	100.0

Crosstabulation

A crosstabulation presents the results of one variable by another variable. PROC FREQ in SAS handles crosstabulations as well as one-way frequencies. An asterisk indicates a crosstabulation. Perhaps we want to find out how question 1 was answered by sex. The program statements to do so would be:

```
proc freq;
tables q1*q4;
```

The crosstabulation produced by PROC FREQ would look like this:

```
TABLE OF Q1 BY Q4

Q1(1. Center at Landfill)     Q4(4. Sex)
```

```
FREQUENCY         |
PERCENT          |
ROW PCT          |
COL PCT          |a) Male |b) Femal|
|        |e       |  TOTAL
-----------------+--------+--------+
a) Yes, for both |      3 |      1 |      4
|  42.86 |  14.29 |  57.14
|  75.00 |  25.00 |
|  60.00 |  50.00 |
-----------------+--------+--------+
b) Yes, only for |      1 |      1 |      2
|  14.29 |  14.29 |  28.57
|  50.00 |  50.00 |
|  20.00 |  50.00 |
-----------------+--------+--------+
d) No            |      1 |      0 |      1
|  14.29 |   0.00 |  14.29
| 100.00 |   0.00 |
|  20.00 |   0.00 |
-----------------+--------+--------+
TOTAL                    5        2        7
71.43    28.57   100.00
```

Tables

PROC TABULATE in SAS produces high quality tables. Its features include complex crosstabulations, frequency counts, breakdowns, and statistical summaries of continuous as well as catagorical data.

PROC TABULATE can produce higher quality crosstabulation results than PROC FREQ. To produce a crosstabulation of question 1 with question 4 using PROC TABULATE, we would add this code to our program:

```
proc tabulate noseps formchar(1 6 7 8)='    ';
class q1 q4;
table q1, q4 * (n*f=7.0  pctn<:q1>*f=7.2);
keylabel n='count' pctn='percent';
title 'Results of question 1 by question 4 sex';
```

"NOSEPS" is an option in PROC TABULATE that requests no horizontal lines across the body of the table. The FORMCHAR option changes vertical lines to blanks in the table. These two options are used together in order to achieve dissertation style boxing, with only lines under the headings of the table.

The CLASS statement specifies which variables are categorical. Here both variables Q1 and Q4 are made up of categories, as opposed to continuous.

The TABLE statement specifies the variables and statistics to be used in the table. The comma separates those variables down the table (the stub) from those variables across the table (the banner). To the left of the comma we have "Q1". This means that the variable Q1 will be in the stub dimension. To the right of the comma we have this:

Q4 *(N*F=7.0 PCTN<Q1>*F=7.2)

Let us take this from left to right. We first encounter Q4. This means that Q4 is to be a variable in the banner. The * has two uses, depending on the context. When it is between variables or statistics, a nesting occurs of the variables or statistics to the right of the asterisk under the variable to the left of the asterisk. The second use is for formatting, which we will discuss below. The asterisk next to Q4 requests two statistics, N and PCTN, to be nested within the answers to Q4.

The set of parentheses is for the nesting of two statistics under one variable. N is a statistics that prints counts. Then we have "*F=7.0". This is the second use of the asterisk. This is a formatting specification. It prints the statistic N in 7 spaces with no decimals.

Then we have PCTN<Q1>*F=7.2. Again we see a formatting specification following the asterisk. This requests that the statistic PCTN, which prints percents, be printed in 7 spaces with 2 decimals. In brackets is the variable Q1, which is our variable down the stub. Percentages can be calculated either across rows or down columns. By including the stub variable in brackets we request that SAS calculate values that add to 100% down the column.

The KEYLABEL statement assigns labels to the statistics. The TITLE statement prints a title at the top of the page.

The resulting table is shown below:

```
Results of question 1 by question 4 sex
-------------------------------------------------------------------
4. Sex
--------------------------------
a) Male        b) Female
--------------- ---------------
COUNT  PERCENT  COUNT  PERCENT
-------------------------------- ------- ------- ------- -------
1. Center at Landfill
a) Yes, for both newspapers and
cans                                  3   60.00       1   50.00
b) Yes, only for newspapers           1   20.00       1   50.00
d) No                                 1   20.00
-------------------------------------------------------------------
```

A more complex table would present the results of questions 1, 2, and 3 by question 4 in one table. The code would be this:

```
proc tabulate noseps formchar(1 6 7 8)='    ';
class q1-q4;
table q1 q2 q3,(q4 all) * (n*f=7.0 pctn<q1 q2 q3>*f=7.2)/rts=20;
keylabel n='count' pctn = 'percent' all= 'total';
title 'Results of questions 1, 2, and 3 by sex';
proc tabulate noseps formchar(1 6 7 8)=' ';
```

Here we have changed the CLASS statement and the TABLE statement. The CLASS statement has been changed to include 4 variables as catagorical variables.

In the TABLE statement, remember from the previous example that the comma separates the stub from the banner. To the left of the comma we have "Q1 Q2 Q3". The space is the concatenation symbol. This asks for the results of Q1 to be followed by the results of Q2, then the results of Q3.

To the right of the comma we see this code: (Q4 ALL) * (N*F=7.0 PCTN<Q1 Q2 Q3)*F=7.2). The code: (Q4 ALL) concatenates the results of question 4 with a special summary class variable ALL. ALL in PROC TABULATE produces a

total. Then we see an asterisk, which indicates there is to be nesting of statistics within the two variables Q4 and ALL. The statistics requested are again N and PCTN, with formatting. Now inside the brackets we have <Q1 Q2 Q3>. This is because we have 3 variables in the stub, and we want percentages calculated for each.

Following this code is "/RTS=20". RTS stands for row title space. This is an option to make the column containing labels 20 columns wide.

The KEYLABEL statement has been changed to provide a label for the variable ALL. The title statement has been changed to a new title.

Here is what the resulting table would look like:

```
Results of questions 1, 2, and 3 by sex
----------------------------------------------------------------------
4. Sex
--------------------------------
a) Male        b) Female       TOTAL
--------------- --------------- ---------------
COUNT  PERCENT  COUNT  PERCENT  COUNT  PERCENT
------------------ ------- ------- ------- ------- ------- -------
1. Center at
Landfill
a) Yes, for both
newspapers and
cans                    3   60.00       1   50.00       4   57.14
b) Yes, only for
newspapers              1   20.00       1   50.00       2   28.57
d) No                   1   20.00                       1   14.29
2. Center at Mall
a) Yes, for both
newspapers and
cans                    2   40.00       1   50.00       3   42.86
b) Yes, only for
newspapers              1   20.00       1   50.00       2   28.57
c) Yes, only for
cans                    1   20.00                       1   14.29
d) No                   1   20.00                       1   14.29
3. Willing to pay
extra
a) Yes                  4   80.00       1   50.00       5   71.43
b) No                   1   20.00       1   50.00       2   28.57
----------------------------------------------------------------------
```

Walkthrough

Let us walk through what has been discussed above. First, sign on to the Alpha. Then create a data file. Let us call it "recycle.dat". From the DCL prompt, type this:

```
$ ed recycle.dat
```

Once in the editor, type in the following lines:

```
001 1 2 1 1 4
002 1 1 1 2 3
003 4 4 2 1 2
004 2 3 1 1 4
005 1 1 1 1 1
006 1 1 1 1 4
007 2 2 2 2 1
```

Save the file by pressing the <Do> key and typing EXIT.

Then create the SAS program. Let us call it "recycle.sas". From the DCL prompt, type this:

```
$ ed recycle.sas
```

Once in the editor, type in the following lines:

```
* recycle.sas SAS Program written by Mary Howard on 3/22/90.
This program does the analysis of a survey on recycling.
Written to demonstrate the use of SAS on the DEC Alpha.   */

options linesize=80;

proc format;
value f1fmt 0 = 'No Response'
1 = 'a) Yes, for both newspapers and cans'
2 = 'b) Yes, only for newspapers'
3 = 'c) Yes, only for cans'
4 = 'd) No';
value f2fmt 0 = 'No Response'
1 = 'a) Yes'
2 = 'b) No'
3 = 'c) Unsure';
value f4fmt 0 = 'No Response'
1 = 'a) Male'
2 = 'b) Female';

value f5fmt 0 = 'No Response'
1 = 'a) $0 to $10,000'
2 = 'b) $10,001 to $20,000'
3 = 'c) $20,001 to $30,000'
4 = 'd) over $30,000';

data dataset1;
infile 'sp90recycle.dat';
input id 1-3 q1 5-5 q2 7-7 q3 9-9 q4 11-11 q5 13-13;
format q1-q2 f1fmt. q3 f2fmt. q4 f3fmt. q5 f4fmt.;
label
q1 = '1. Center at Landfill'
q2 = '2. Center at Mall'
q3 = '3. Willing to pay extra'
q4 = '4. Sex'
```

```
q5 = '5. Family Income';

proc print;
var id q1-q5;
format id q1-q5 3.0;

proc freq;
tables q1 q2 q3 q4 q5;

proc freq;
tables q1 * q4;

proc tabulate noseps formchar(1 6 7 8)='    ';
class q1 q4;
table q1, q4 * (n*f=7.0  pctn<q1>*f=7.2);
keylabel n='count' pctn='percent';
title 'Results of question 1 by question 4 sex';

proc tabulate noseps formchar(1 6 7 8)='    ';
class q1-q4;
table q1 q2 q3,(q4 all) * (n*f=7.0 pctn<q1 q2 q3>*f=7.2)/rts=20;
keylabel n='count' pctn = 'percent' all = 'total';
title 'Results of questions 1, 2, and 3 by sex';
```

Save the file by pressing the <Do> key and typing EXIT.

Now run the program. From the DCL prompt, type this:

```
$ sas recycle
```

When the job is done, the following should appear:

```
Job RECYCLE (queue SAS$BATCH, entry 715) completed
```

Now look at the output.

```
$ look recycle.lis
```

To exit look, type "q".

Then print the output,type:

```
$ print recycle.lis
$ print recycle.log
```

Once a file is printed you will get a message on your screen, like this:

```
Job RECYCLE (queue SYS$PRINT,entry 297) completed
```

You can then logout and go to the I/O window room in 19 Business to pick up your output.

---

sas

Source URL: http://www.uni.edu/its/support/article/610