NON-STATISTICAL USES OF SAS

Steven Mays
University of Texas Health Science Center at Dallas

Introduction

The everyday needs of a computer shop bring about new application of SAS. This has been particularly true for us at the UT Health Science Center at Dallas. I am in the general applications section of the computer center and am constantly in contact with people who need results. It is obviously true that SAS can help get these results, if they are statistical in nature. But what about ordinary applications, that are not necessarily statistical ? It is the intention here to present some of these uses. Perhaps, something may be conveyed that will aid the reader.

The following are several techniques and ideas, that have come about in the course of two years of experience.

Using SAS to update non-SAS data sets

There is often a need to update observations in a data set. If you are maintaining the data in the form of a SAS data set, then it is easy to use the MERGE statement. However, there may be a situation when the data is maintained in a standard O.S. data set. This happened to us in a situation where the data had to be preprocessed with several programs before being handled by SAS and SPSS. Our client decided he wanted to correct some 200 or so observations. We decided to try SAS to update the data. What this meant was that even though we needed to form a SAS data set, ultimately we wanted to go back to the original pre-SAS format. We realized that this may not have been the most efficient use of computer resources, but it was nice to have an existing mechanism to update, bypassing a program written from scratch. I believed it was an efficient use of computer software in the form of SAS to save valuable programmer time in a one-shot general applications environment. Here is a brief description of how we did it and comments on its effectiveness.

```
//STEP1 EXEC SAS
//SAS.DD1 DD DSN=JER.STAT1,UNIT=TAPE,DISP=OLD
//SAS.NEW DD DSN=&&PASS,DISP=(NEW,PASS),
// UNIT=SYSDA,SPACE=(CYL,(5,5))
//SAS.SYSIN DD *
DATA OLD;
INPUT DDNAME=DD1
PAT_NUM 2-6 CARD_NUM 79-80 CARD $ 1-80;
PROC SORT;BY PAT_NUM CARD_NUM;
DATA UPD;
INPUT
PAT_NUM 2-6 CARD_NUM 79-80 CARD $ 1-80;
CARDS;

 .... ...... ........
PROC SORT;BY PAT_NUM CARD_NUM;
DATA NEW;
MERGE OLD UPD;
BY PAT_NUM CARD_NUM;
DROP PAT_NUM CARD_NUM;
/*
```

```
//STEP2 EXEC FORNCX
//C.SYSIN DD *
        INTEGER COUNT
        INTEGER*2 ID
        LOGICAL*1 CARD(80)
        COUNT=0
1       READ(8) ID
        IF(ID.NE.99)GO TO 1
2       READ(8,END=9)CARD
        COUNT=COUNT+1
        WRITE(9,3)CARD
3       FORMAT (80A1)
        IF(COUNT.GE.9)GO TO 2
        WRITE(6,4)CARD
4       FORMAT(1X,80A1)
        GO TO 2
9       CALL EXIT
        END
/*
//X.FT06F001 DD SYSOUT=A
//X.FT08F001 DD DSN=&&PASS,DISP=(OLD,DELETE)
//X.FT09F001 DD DSN=JER.NEW.STAT1,DISP=(NEW,CATLG),UNIT=TAPE,
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=1600)
```

Our first experience with this worked, except it was noticed that NEW had more observations than OLD. Now, if there were inserts, as opposed to only replacements, then this would be conceivable. However, we knew before that there were no adds. How could we detect these bad updates ? Fortunately, SAS has a very simple tool to detect these errors:

```
DATA NEW ;
MERGE OLD UPD ; BY PAT_NUM CARD_NUM ;
IF MERGE = 2 THEN PUT PAT_NUM CARD ;
DROP PAT_NUM CARD_NUM ;
```

If you will remember MERGE accompanied by a BY statement provides two variables for assisting the programmer: LASTBY and MERGE. When MERGE = 2 then there is no observation in OLD with the current BY value. So the IF statement above would cause PAT_NUM and CARD to be printed when a bad update is made. With these we were able to trace down the errors, correct them and resubmit the update job. Even if there were to be inserts, the MERGE variable is helpful:

```
IF MERGE = 3 THEN PUT '*REPLACEMENT* ' PAT_NUM CARD ;
IF MERGE = 2 THEN PUT '*INSERTION  * ' PAT_NUM CARD ;
```

The second step puts the data set back into the regular format. If you have the SAS Programmer's Guide, there is a description of how one reads a SAS data set in FORTRAN and writes it out back in character representation. A better alternative is the use of PRTPCH, a member of the SAS supplementary procedure library. Try:

```
PROC PRTPCH; VAR CARD ;
```

```
PARMCARDS;
(10A8)
and in STEP1  override the FT02F001 DD statement
with
//SAS.FT02FT001 DD DSN=JER.NEW.STAT1,DISP=(NEW,CATLG),
//  UNIT=TAPE,
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=1600)
```

STEP2 would not be necessary then.

No doubt, there is more overhead than necessary. Work is being expended to create the SAS data sets OLD and UPD. Then after the merge, work is required to undo NEW. For the 9660 80-byte record data set about 42 c.p.u. secs. were used to do STEP1 and another 55 secs. to do STEP2. This is worth it, however, when you consider programming is reduced to about 40 lines of code, including JCL, SAS, and FORTRAN.

The use of SAS program statements in Reports

SAS program statements provide surprising program flexibility. If one is concerned in getting out a quick report, SET, DROP, IF, GO TO, and RETURN along with the PRINT procedure can give a respectable summary. To show SAS's capability a sales analysis example is presented.

Store X has seven salesmen. A record is kept for each sale. Each sales record contains the salesman's number, the part number, the part name, the quantity sold, and the price per unit. The gross amount is simply the product of quantity and price. One added factor is the determination of each man's commission, which depends on the man's rate, based on seniority and the total amount of his sales.

The report consists of two tables. The first one is the summary of the salesmen including each individual's total sales and the calculated commission. The second one is a detail of all the sales by salesman. To get the first table it is necessary to create the data set SUMMARY. Here is where the program statements come in handy. SUMMARY is a subset of SALES containing exactly one observation for each salesman. The program statements are shown on the next page.

```
    COMMENT      SALES ANALYSIS BY SALESMAN
    ;
    DATA SALES ;
    INPUT   S_NO  1-2       PART_NO $ 5-11
            PARTNAME $ 12-30        QUANTITY 31-34
            PRICE  35-42    SALESMAN $ 51-75 ;
    IF S_NO = 1 THEN SALESMAN = 'JACK ADAMS' ;
    IF S_NO = 2 THEN SALESMAN = 'BILL BELTON' ;
    IF S_NO = 3 THEN SALESMAN = 'WAYNE CAMPBELL' ;
    IF S_NO = 4 THEN SALESMAN = 'DICK HOLMES' ;
    IF S_NO = 5 THEN SALESMAN = 'SID MCDOWELL' ;
    IF S_NO = 6 THEN SALESMAN = 'NED TEMPLE' ;
    IF S_NO = 7 THEN SALESMAN = 'PETER YOUNG' ;
    IF S_NO = 1 THEN RATE = .15 ;
    IF S_NO = 2 THEN RATE = .10 ;
    IF S_NO = 3 THEN RATE = .15 ;
    IF S_NO = 4 THEN RATE = .10 ;
    IF S_NO = 5 THEN RATE = .10;
    IF S_NO = 6 THEN RATE = .20 ;
    IF S_NO = 7 THEN RATE = .15 ;
    GROSS = QUANTITY * PRICE ;
    CARDS ;
    3   2305   PENCIL SHARPENER   5   2.95
    6   3502   BOOK CASE          1   25.70
     .. ...   ... ...     .....
    PROC SORT ; BY S_NO ;

    DATA SUMMARY ; SET SALES ;
            IF NO KEEP_NO THEN GO TO SET2 ;
            IF KEEP_NO NE S_NO THEN GO TO SET1 ;
    ADD : TOTAL = TOTAL + GROSS ;
            ORATE = RATE ;
            MAN = SALESMAN ;
           RETURN ;
    SET1 : COMM = ORATE * TOTAL ;  COMM = FLOOR (COMM * 100 + .5) / 100 ;
            OUTPUT ;
    SET2 : KEEP_NO = S_NO ;
            TOTAL = 0 ;
           GO TO ADD ;
            DROP KEEP_NO ;

            PROC PRINT ; TITLE 'SALES SUMMARY ' ; ID MAN ;
            VAR TOTAL COMM ;
```

The total of sales is zeroed at the beginning of each salesman. (Note: it is important that SALES be sorted by salesman number prior to the creation of SUMMARY.) A record is output only when a change is encountered in the salesman number. The records contain the total and the commission for that particular salesman. The total is set to zero and the next salesman is processed. One thing should be kept in mind. When the last sales record of the last salesman is considered, there is no direct means within the program statements that allow for an observation for the last salesman. The way to get around this is inserting a dummy record, assigning 99 to the salesman's number. This means the dummy record will be sorted to the last, causing all the salesmen to be put on SUMMARY. Therefore, using 18 statements including those associated with PROC PRINT, one is able to produce the first table.

SALES SUMMARY

MAN	TOTAL	COMM
JACK ADAMS	3230.95	484.64
BILL BELTON	816.65	81.66
WAYNE CAMPBELL	93.31	14.00
DICK HOLMES	647.99	64.80
SID MCDOWELL	430.75	43.07
NED TEMPLE	225.70	45.14
PETER YOUNG	332.65	49.90
N=7		

The second table is much easier, for it is just a listing of SALES.

INPUT:

```
     PROC PRINT DATA = SALES ; BY SALESMAN ;
VAR PARTNO PARTNAME QUANTITY PRICE GROSS ;
```

OUTPUT:

SALES SUMMARY

SALESMAN=JACK ADAMS

OBS	PARTNO	PARTNAME	QUANTITY	PRICE	GROSS
1	3104	DESK -TYPE 1	2	85.00	170.00
2	2345	SUIT CASE	1	45.95	45.95
3	2708	DESK - TYPE 5	7	245.00	1715.00
4	2708	DESK -TYPE 5	4	325.00	1300.00

SALESMAN=BILL BELTON

OBS	PARTNO	PARTNAME	QUANTITY	PRICE	GROSS
1	3299	DESK - TYPE 4	3	125.00	375.00
2	1205	# 2 PENCIL	100	0.04	4.00
3	3103	FILING CABINET	3	85.15	255.45
4	3325	DESK LAMP	2	25.85	51.70
5	1704	DESK -TYPE 1	1	130.50	130.50

SALESMAN=WAYNE CAMPBELL

OBS	PARTNO	PARTNAME	QUANTITY	PRICE	GROSS
1	2305	PENCIL SHARPENER	5	2.950	14.75
2	2850	FORM 1200	2000	0.025	50.00
3	1864	BALL POINT PEN	4	0.890	3.56
4	1845	SLIDE RULE	2	12.500	25.00

SALESMAN=DICK HOLMES

OBS	PARTNO	PARTNAME	QUANTITY	PRICE	GROSS
1	1804	GLOBE	1	22.00	22.00
2	2100	DESK - TYPE 2	4	136.75	547.00
3	2484	ELECT. CALCULATOR	1	78.99	78.99

SALESMAN=SID MCDOWELL

OBS	PARTNO	PARTNAME	QUANTITY	PRICE	GROSS
1	2402	CARD TABLE	1	27.50	27.50
2	2401	DESK RADIO	1	35.00	35.00
3	3325	DESK LAMP	1	25.85	25.85
4	1105	CUSHIONED CHAIR	4	85.60	342.40
.....	....	....			

Since the dummy card is in SALES, it will show up also in the PRINT. It will be the last record printed since it was sorted to the end. If the PAGE option is envoked then each salesman will be printed on separate pages. The last page will contain the dummy record, so it can be easily discarded. If one prefers, a subset of SALES, made so that it excludes the dummy record, can be printed. Or even something like this:

```
DATA S ; INPUT - - - - ;
CARDS ;
- - - - - -
DATA DUMMY ; INPUT - - - - ;
CARDS ;
99
DATA SALES ;
SET S ; SET DUMMY ;
```

Then create SUMMARY from SALES, and print the second table from S. The alternative you choose depends on the quantity of data.

From this start you can build and add more program statements for specialized situations. Here are some new factors:

(1.) A salesman's commission is determined by the formula:

COMMISSION = (TOTAL SALES - QUOTA) * RATE OF COMMISSION

This means that besides the rate of commission, the salesman's quota must be known.

(2.) The salesmen work at two stores 1 and 2.

(3.) The first digit of the part number has a special significance. It describes the location where the item is stocked.

Therefore, two new tables will be produced, one by store and another by stock location within salesman will be produced. The sales summary by salesman with the new way to calculate the commission will again be done. To code for each salesman's quota and the store he works in requires an additional 14 IF statements. There is an alternative to all those IF's and that is through the use of the MERGE statement. Without further comment the SAS statements involved are begun on the next page:

```
COMMENT          USE MERGE TO SUPPLY
                 SALESMEN CODES
;
DATA M_CODES ;
INPUT  S_NO 1-2          RATE 4-5 2
       STORE  7          QUOTA 9-12
       SALESMAN $ 16-40 ;
CARDS;
 1 15 2  175   JACK ADAMS
 2 10 1  150   BILL BELTON
 3 15 2  125   WAYNE CAMPBELL
 4 10 2  125   DICK HOLMES
 5 10 1  135   SID MCDOWELL
 6 20 1  125   NED TEMPLE
 7 15 1  125   PETER YOUNG
99 99 9
DATA SALE ;
INPUT   S_NO 1-2          PART_NO $ 5-11          PARTNAME $ 12-30
        QUANTITY 31-34    PRICE 35-42             LOC 5
        LOCATION $ 51-65 ;
GROSS = QUANTITY * PRICE ;
IF LOC = 1 THEN LOCATION = 'LOCAL' ;
IF LOC = 2 THEN LOCATION = 'WAREHOUSE' ;
IF LOC = 3 THEN LOCATION = 'OUT OF TOWN' ;
CARDS ;
3   2305   PENCIL SHARPENER   5   2.95
6   3502   BOOK CASE          1   25.70
2   3299   DESK - TYPE 4      3   125.00
 .... .... .... ...

PROC SORT ; BY S_NO ;
DATA SALES ;
MERGE M_CODES SALE ;
BY S_NO ;
COMMENT          IF THERE IS NOT A RECORD FOR A SALESMAN THEN DELETE ;
IF NO RATE THEN DELETE ;
OUTPUT ;
```

```
COMMENT          SALES SUMMARY BY SALESMAN - WITH NEW SCHEME
                 FOR COMMISSION
;

DATA SUMMARY ; SET SALES ;
        IF NO KEEP_NO THEN GO TO SET2 ;
        IF KEEP_NO NE S_NO THEN GO TO SET1 ;
ADD : TOTAL = TOTAL + GROSS ;
        ORATE = RATE ;
        OQUOTA = QUOTA ;
        MAN = SALESMAN ;
       RETURN ;
SET1 : COMM = ORATE * (TOTAL - OQUOTA) ;
        COMM = MAX(0,COMM) ; COMM = FLOOR (COMM * 100 + .5) / 100 ;
        OUTPUT ;
SET2 : KEEP_NO = S_NO ;
        TOTAL = 0 ;
       GO TO ADD ;
        KEEP MAN TOTAL COMM ;

        PROC PRINT ; TITLE 'SALES SUMMARY ' ; ID MAN ;
        VAR TOTAL COMM ;
```

SALES SUMMARY

MAN	TOTAL	COMM
JACK ADAMS	3230.95	458.39
BILL BELTON	816.65	66.66
WAYNE CAMPBELL	93.31	0.00
DICK HOLMES	647.99	52.30
SID MCDOWELL	430.75	29.57
NED TEMPLE	225.70	20.14
PETER YOUNG	332.65	31.15
N=7		

```
COMMENT           SUMMARY BY STORE
;
PROC SORT DATA = SALES ; BY STORE ;
DATA SUMMARY ; SET SALES ;
        IF NO KEEP THEN GO TO SET2 ;
        IF KEEP NE STORE THEN GO TO SET1 ;
ADD : TOTAL = TOTAL + GROSS ;
        STORES=STORE ;
       RETURN ;
SET1 :  OUTPUT ;
SET2 : KEEP = STORE ;
        TOTAL = 0 ;
       GO TO ADD ;
        KEEP STORES TOTAL ;

PROC PRINT ; TITLE 'SALES BY STORE';
ID STORES ; VAR TOTAL ;
```

SALES BY STORE

STORES	TOTAL
1	1805.75
2	3972.25

N=2

```
COMMENT          SUMMARY BY STOCK LOCATION WITHIN SALESMAN ;
PROC SORT DATA=SALES ; BY S_NO LOC ;
DATA SUMMARY ; SET SALES ;
        IF NO KEEP_1 THEN GO TO SET2 ;
COMMENT          EITHER A CHANGE IN S_NO OR LOC CAUSES
                 A BREAK
;
         IF KEEP_1 NE S_NO THEN GO TO SET1 ;
        IF KEEP_2 NE LOC THEN GO TO SET1 ;
ADD : TOTAL = TOTAL + GROSS ;
        STOCKED=LOCATION ;
        MAN = SALESMAN ;
       RETURN ;
SET1:   OUTPUT ;
SET2 : KEEP_1 = S_NO ;
        KEEP_2 = LOC ;
        TOTAL = 0 ;
       GO TO ADD ;
        KEEP MAN STOCKED TOTAL ;

PROC PRINT ;
TITLE 'SUMMARY BY STOCK LOCATION WITHIN SALESMAN';
ID MAN ; VAR STOCKED TOTAL ;
```

SUMMARY BY STOCK LOCATION WITHIN SALESMAN

MAN	STOCKED	TOTAL
JACK ADAMS	WAREHOUSE	3060.95
JACK ADAMS	OUT OF TOWN	170.00
BILL BELTON	LOCAL	134.50
BILL BELTON	OUT OF TOWN	682.15
WAYNE CAMPBELL	LOCAL	28.56
WAYNE CAMPBELL	WAREHOUSE	64.75
DICK HOLMES	LOCAL	22.00
DICK HOLMES	WAREHOUSE	625.99
SID MCDOWELL	LOCAL	342.40
SID MCDOWELL	WAREHOUSE	62.50
SID MCDOWELL	OUT OF TOWN	25.85
NED TEMPLE	LOCAL	49.25
NED TEMPLE	WAREHOUSE	150.75
NED TEMPLE	OUT OF TOWN	25.70
PETER YOUNG	WAREHOUSE	295.85
PETER YOUNG	OUT OF TOWN	36.80

N=16

NON-STATISTICAL USES OF SAS

A Job Request Information System

The previous topics about report writing used an example that was made up. It dwelt mostly on the report capabilities of SAS and did not discuss how SAS assists in the management of the data. This section brings up an actual application that demonstrates SAS in these two areas. Whenever a client comes in with a request, we will fill out a job request form. Associated with this request is (1) a request number, (2) the investigator, (3) the project name, (4) the department, (5) a 30-character description of the job, (6) the date of the request, (7) the date the job is to be completed, (8) the status of the job, (9) the programmer(s) assigned to the job, (10) the computer used (DECsystem-10 or IBM 370), (11) the language(s), and (12) an internal account number. Once the job is initiated and entered into the JRIS (Job Request Information System), a status is attached to job. When the job is finished, it is assigned the status of "completed". It is not taken out, though, for theoretically , it could be restarted.

An actual run has been included to show the programming involved in adding and updating the JRIS data set. In addition, the run shows the statements necessary to prepare two listings - (1) by programmer, and (2) by investigator.

There are other things that are of interest besides these two listings. Frequencies of requests handled by programmer, by department, by machine, and by language are quite useful for evaluation and scheduling. Also, an alphabetical listing by investigator is most helpful in determining whether past jobs have been done for a given person.

Using SAS for JRIS has been a pleasant experience over the period of a year and a half that it is evolved. When changes were necessary to the system it was easy to live with. This is true when you notice only about 30 SAS statements are required in the basic program. Also, new and inventive ways to display the data have happened naturally. It is hard to imagine an alternative to SAS that could have done a simpler and more satisfactory job.

NON-STATISTICAL USES OF SAS

```
// EXEC SAS
//SAS.FILE DD DSN=JRIS,UNIT=DISKA,DISP=OLD,
//   DCB=LRECL=2222
//SAS.SYSIN DD *
DATA UPDATE;  INPUT
REQ_NO     #1 1 -4        INVEST   $ #1 5 -16       PROJECT  $ #1
17-28
DEPT     $ #1 29-36       PROG_NAM $ #1 37-66       DATE_IN  $ #1
67-74
DATE_DUE $ #2 1 -8        STAT       #2 9 -9        AUTHOR_A $ #2
10-21
AUTHOR_B $ #2 22-33       MACHINE  $ #2 34-36       LANGUAGE $ #2
37-40
MCRC_ACC $ #2 41-47       STATUS   $ #2 48-62;
IF STAT = 1 THEN STATUS = 'WAITING';
IF STAT = 2 THEN STATUS = 'NOT BEGUN';
IF STAT = 3 THEN STATUS = 'PRE-CODING';
IF STAT = 4 THEN STATUS = 'CODING';
IF STAT = 5 THEN STATUS = 'DATA ENTRY';
IF STAT = 6 THEN STATUS = 'DEBUGGING';
IF STAT = 7 THEN STATUS = 'USER CHECKOUT';
IF STAT = 8 THEN STATUS = 'PRODUCTION';
IF STAT = 9 THEN STATUS = 'CONTINUING';
IF STAT = 0 THEN STATUS = 'COMPLETED';
IF NO STAT THEN STATUS = '    ';
CARDS;
2152
        1
2177
        2
2179
        0
2180DIAL, J.    REHAB STRAT.RHAB.SCISAS CORRELATIONS
01/15/76
01/18/767HOUSTON B.              IBMSAS 470-080
  ..... ...... .....

PROC SORT; BY REQ_NO;
DATA NEWF;  MERGE FILE UPDATE;  BY REQ_NO;
TITLE 'APPLICATIONS SECTION:  JOBS AS OF JANUARY 13, 1976    ';
DATA FILE; SET NEWF;
PROC PRINT;
DATA SUB;  SET NEWF;
IF STAT > 1 OR NO STAT;
PROC SORT; BY AUTHOR_A;
PROC PRINT DOUBLE;  BY AUTHOR_A;  ID REQ_NO;
VAR DATE_DUE MCRC_ACC INVEST PROG_NAM STATUS LANGUAGE;
TITLE 'BREAKDOWN OF ACTIVE JOBS BY PROGRAMMER - JANUARY 13, 1976    ';
PROC SORT; BY INVEST;
PROC PRINT DOUBLE;  ID REQ_NO;
VAR MCRC_ACC INVEST DEPT PROG_NAM AUTHOR_A STATUS;
TITLE 'LISTING OF ACTIVE JOBS BY INVESTIGATOR - JANUARY 13, 1976    ';
/*
```

NON-STATISTICAL USES OF SAS

A Report Procedure

The sequence of topics has lead to this last one, the design of a report procedure. Here is a list of capabilities, lacking in existing procedures that would be nice for PROC REPORT to have:

(1) There would be greater control on column headings - As it now stands the variable name must be the column heading. It would be nice to have greater control on this: (a) The column heading is supplied by the user and is not necessarily the variable name. (b) It is no longer limited to 8 alpha-numeric characters. (c) The column heading can be more than one word. (d)Two lines can be employed to write the column heading.

(2) The user would have control over the format of the variables as they printed on the report.

(3) A specified subset of the numeric variables could be totaled. These variables would be arranged from major to minor, and a total line would be printed when a "break" was encountered among any one of them. Grand totals could also be kept.

(4) Special features could be included: (a) page numbers, (b) current date, (c) user specified page size, and (d) variables that caused breaks could be printed only when there was a change.

There are many other attributes that a REPORT procedure could have. For this reason it would probably be better if such a procedure be "home grown", meeting the needs of the local shop. But for the sake of discussion, how would one approach writing such a procedure ? Here are a few thoughts on how I would approach it:

(1) How would the procedure be invoked ?

It would look like a PROC PRINT except for the addition of a CLASSES procedure information statement.

```
PROC REPORT FORMAT=[C OR U] DOUBLE ;
VARIABLES I A B C D E F ;
CLASSES A B ;
BY I A B ;
```

The FORMAT option allows one to specify either Composed or User format for the printing of column headings and the body of the report. Composed format would be the default. The parameter DOUBLE has the same effect as it does in PROC PRINT, double spacing. Also, like PRINT, VARIABLES would specify the order that data would be printed across the page. CLASSES should list the numeric variables totals will be kept on. The BY statement identifies the variables that cause control breaks, from major to minor. It is assumed that the SAS data set is sorted by the BY variables.

(2) How is such a procedure written so that it can be incorporated into SAS?

The SAS programmer's guide, not to be confused with the white User's Guide, describes how one may go about programming the procedure. The programmer can read how he can retrieve from the PROC statement information supplied through the options and parameters. The procedure itself can be written in any one of the languages FORTRAN, PL/I, OR COBOL. There are several commonly used routines, giving easy access to the SAS data set and information about the variables.

(3) How does one handle the format of the report?

First, if it is user-supplied, one can take advantage of the PARMCARDS statement. This would allow one to read in the format to correspond to the ordering specified by the VARIABLES statement. The composed format would be a lot harder to take care of in programming. It would not be necessary to incorporate into REPORT, but it would be a nice addition.

(4) What about the column headings?

They, too, could be entered through the PARMCARDS statement. If the user does not supply them, then there are ways of getting the VARIABLE name to be the column heading.