Macros and Macro-like Applications for Minicomputers
Virginia M. Dineley, SAS Institute Inc., Cary, NC
Toby J. Trott, SAS Institute Inc., Cary, NC

ABSTRACT
Version 5 of the SAS® System for minicomputers has limited support of the macro facility. This paper demonstrates programming techniques that utilize the power of the DATA step, as well as the %INCLUDE statement, %LET statement and the macro functions SYMGET and CALL SYMPUT. In particular, we demonstrate how several examples from the SAS Guide to Macro Processing, Version 5 Edition, can be adapted for use in the minicomputer environments.

INTRODUCTION
The macro facility provides enormous capabilities to those running the SAS System in the mainframe and PC environments. Many people in the minicomputer environment feel that the lack of complete support of the macro facility imposes serious restrictions on the types of processing that can be done. The purpose of this paper is to demonstrate how the functionality provided by the macro facility can be accomplished using base SAS software only. The examples in this paper are from the SAS Guide to Macro Processing, Version 5 Edition. They have been rewritten to accomplish the same tasks using only the DATA step and the subset of the macro facility available to minicomputer users.

CURRENT FEATURES
Version 5 of the macro facility for minicomputers consists of
- %LET statement
- SYMPUT routine
- SYMGET function.
The %LET statement and the SYMPUT routine enable you to assign values to macro variables. The SYMGET function allows you to assign the value of an existing macro variable to a DATA step variable. As the examples show, the manipulation of macro variables provides great flexibility in SAS programming.

In addition to the features above, the examples demonstrate the use of the
- %INCLUDE statement
- TTOPEN function
- TTREAD function
- TWRITE function
- TTCLOSE function.
The %INCLUDE statement is used to include SAS statements from external files. The Institute-supplied sample functions, TTOPEN, TWRITE, TTREAD, and TTCLOSE, are used to write lines to a terminal and to read lines that have been entered at the terminal. These functions provide the capability of creating interactive SAS applications. These functions are available in Release 5.18 of the SAS System running under the PRIMOS® and AOS/VS operating systems.

MACRO VARIABLES
Macro variables differ from DATA step variables in that they can be defined anywhere in a SAS program except after a CARDS statement. The value of a macro variable remains constant until it is changed. A macro variable is defined by using a %LET statement or the SYMPUT routine. Once the macro variable is defined, you can use the value of the macro variable through a macro variable reference. The simplest type of macro variable reference consists of an ampersand (&) placed in front of the macro variable name. For example, &MACVAR refers to the macro variable, MACVAR. When this reference is encountered during compilation of the SAS program, the macro processor replaces it with the current value of the macro variable, MACVAR. For more information on macro variables see the SAS User's Guide: Basics, Version 5 Edition.

WRITING GENERIC PROGRAMS
You can write generic programs using macro variable substitution, for example,
DATA STATE;
SET PERM.STATES;
IF STATE="STATE";
PROC PRINT DATA=STATE;
TITLE "Listing of 'STATE Residents";
RUN;
In this program the macro variable STATE is referenced in four places. When the program is compiled, all references to STATE are replaced by the current value of STATE. Use the following %LET statement to assign a value of NC to this macro variable:
%LET STATE=NC;
Then execute the program. The macro variable reference is replaced by the text NC. The code executes as if you had specified
DATA NC;
SET PERM.STATES;
IF STATE="NC";
PROC PRINT DATA=NC;
TITLE "Listing of NC Residents";
RUN;
The data set NC is created and contains only those observations from PERM.STATES where STATE=NC. Execution of the PROC step prints the NC data set and places the text NC in the title.

Then you can create a data set containing observations from the PERM.STATES data set, where STATE=AL, simply by specifying
%LET STATE=AL;
and reexecuting the program.

For applications that are run regularly, with only changes in the data being processed, it is best to create a file containing the
application program and then use %LET statements to supply the variable information. In the next example a file named CHART.SAS, which contains the following statements, is created:

```sas
%LET OPTIONS = DISCRETE;
%LET CHARTVAR = GROSSPAY;
%INCLUDE CHART;
```

To execute this program you need to assign values for the data set name, the type of chart you want to create, the variables you want to process, and any options you want to use to create the chart. To create a horizontal bar chart using the variable DEPT set name, the type of chart you want to create, the variables you want to process, and any options you want to use to create the chart, you simply specify:

```sas
PROC CHART DATA=IN.PAYROLL;
  TYPE=HBAR;
  DSN=IN.PAYROLL;
  OPTIONS_DISCRETE;
  INCLUDE CHART;
```

The SYMPUT routine is used to assign values to macro variables during DATA step execution. The first argument to the SYMPUT routine identifies a macro variable. The second argument can be a DATA step variable, expression, or character literal. When this statement is executed, the value of the second argument is assigned to the macro variable specified in the first argument. When the DATA step executes, a value is assigned to the macro variable PROG. The value of the macro variable that is created by the SYMPUT routine cannot be accessed until the DATA step has run. The macro variable reference following the DATA step is resolved to its current value, and the appropriate SAS program is brought into the session and is executed.

**GENERATING PROGRAMS**

The macro language iterative %DO statement is often used to generate repetitive code. This is accomplished in the DATA step by iteratively writing SAS statements to a file and using the %INCLUDE statement to include that file.

In the next example, there are twelve external data files called IN1.DAT, IN2.DAT,...,IN12.DAT. Each file contains product inventory information, the date the product was purchased, the inventory control number, and the cost of the item. At different times during the year you are asked to produce reports for varying periods of the year.

For example, in March you are asked to print separate reports for January and February. The following example generates a SAS program to perform this task:

```sas
DATA _NULL_;
  FILE 'PROG.SAS';
  DO I=BEGIN to END;
    MONTH= 'MONTH' I LEFT I;
    PUT 'DATA "MONTH"';
    PUT 'DROP "MONTH"';
    PUT 'DROP "INPUT DATE DAT" PRODUCT & COST"';
    PUT 'FORMAT DATE DATE9.';
    PUT INPUT DATE DATE7. PRODUCT S COST;
  END;
  IF _I=1 THEN DO;
    PUT CALL SYMPUT('PROG','%INCLUDE MISS');
    PUT CALL SYMPUT('PROG','%INCLUDE OK');
    FLAG=O AND LAST THEN DO;
    PUT 'PROC PRINT DATA="MONTH"';
    PUT 'TITLE "PRODUCT INVOICES FOR MONTH"';
    PUT 'RUN';
    END;
  END;
RUN;
```

The macro variables BEGIN and END control the iterative DO loop that is used to write SAS statements to the file PROG.SAS. By changing the values of BEGIN and END with %LET statements, you control which data are processed, as well as the amount of repetitive code that is generated. To produce reports for January and February, specify:

```sas
%LET BEGIN=1;
%LET END=2;
```

As the loop executes, the SAS code is written to PROG.SAS. Notice that the SYMPUT routine is used here to assign the actual
month (January, February) to the macro variable MONTHVAL. The reference to MONTHVAL in the TITLE statement places the name of the month in the title.

The generated code follows:

```sas
PROC PRINT DATA=KONTH1;
    TITLE "PRODUCT INVOICES FOR 'KONTHVAL";
    DROP HONTH;
    IF ..1L=1 TREM DO;
        CALL SYMPUT('NKONTHVAL',KONTH);
        KONTH=SU8STR(KONTH, 1, INDEX(HONTH," ");
    END;
    PROC PRINT DATA=KONTH2;
        TITLE "PRODUCT INVOICES FOR 'KONTHVAL";
        IF ..1L=1 THEN DO;
            CALL SYMPUT('NKONTHVAL',KONTH);
            KONTH=SU8STR(KONTH, 1, INDEX(HONTH," ");
        END;
    RUN;
```

Execution of the %INCLUDE statement brings the generated program into the SAS session. Execution of this program produces Output 1.

```
Product invoices for January

<table>
<thead>
<tr>
<th>OBS</th>
<th>DATE</th>
<th>PRODUCT</th>
<th>COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>02MAR1987</td>
<td>3258328</td>
<td>351.21</td>
</tr>
<tr>
<td>2</td>
<td>13MAR1987</td>
<td>716164</td>
<td>87.21</td>
</tr>
<tr>
<td>3</td>
<td>14MAR1987</td>
<td>715993</td>
<td>432.87</td>
</tr>
</tbody>
</table>

Product invoices for February

<table>
<thead>
<tr>
<th>OBS</th>
<th>DATE</th>
<th>PRODUCT</th>
<th>COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>02FEB1987</td>
<td>256532</td>
<td>267.43</td>
</tr>
<tr>
<td>2</td>
<td>13FEB1987</td>
<td>943156</td>
<td>99.99</td>
</tr>
<tr>
<td>3</td>
<td>14FEB1987</td>
<td>99364</td>
<td>682.90</td>
</tr>
</tbody>
</table>
```

Output 1 Listing of Product Invoices for January and February

**GENERATING PROC STEPS USING DATA STEP VALUES**

It is often necessary to write PROC steps that are data dependent. This usually means that you have to process the data, find the desired values, and then hard code these values into SAS code. As the next example shows, it is possible to generate a PROC step with macro variables that contain the data dependent values you want inserted into SAS code.

In this example, the CALL SYMPUT statements are used to create macro variables containing the number of the current BY groups, the first and last observations in each BY group, and the mean salary in each BY group. Next, generate a program, PLOTIT.SAS, that contains a separate PROC PLOT step for each BY value in the data set. Once the program is created, you can use the %INCLUDE statement to create one plot for each BY group. Each plot has a reference line drawn to the mean salary. The mean salary differs for each BY group.

The complete program follows:

```sas
PROC PRINT DATA=KONTH1;
    DROP HONTH;
    INPUT DATE DATE 7. PRODUCT $ COST;
    FORMAT DATE DATE"9.;
    DROP HONTH;
    IF ..1L=1 TREM DO;
        CALL SYMPUT('NKONTHVAL',KONTH);
        KONTH=SU8STR(KONTH,1,INDEX(HONTH," ");
    END;
    PROC PRINT DATA=KONTH2;
        TITLE "PRODUCT INVOICES FOR 'KONTHVAL";
        IF ..1L=1 THEN DO;
            CALL SYMPUT('NKONTHVAL',KONTH);
            KONTH=SU8STR(KONTH,1,INDEX(HONTH," ");
        END;
    RUN;
```

In the first DATA step, each iteration creates four macro variables, FIRST, LAST, REF, and GROUP. For example, during the first execution by concatenating the value of _N_ to the root value of FIRST, FIRST1 are created.

When this program is executed, the macro variable references in the TITLE statement are surrounded by double quotes. This is necessary to force resolution of the macro variables inside of the quotation marks; if single quotes had been used, the title would have appeared as, GROUP=' 'GROUP'1 .

Note that the titles in the TITLE2 statements are surrounded by double quotes. This is necessary to force resolution of the macro variables inside of the quotation marks, if single quotes had been used, the title would have appeared as, GROUP=GROUP2 .

When this program is executed, the macro variable references are replaced by the values assigned during execution of the first DATA step. See Output 2.
Transferring Information with SYMPUT and SYMGET

Another function of the macro facility is the ability to transfer information from one data set to another. The next example demonstrates how this is performed using the SYMPUT routine and the SYMGET function.

The SYMGET function assigns the value of a macro variable to another function of the macro facility is the ability to transfer information for employees of Dusty Department Store. The final report displays each employee’s name, department, salary, and the store’s total salary expenditure. For a more detailed discussion of this example, see pages 121-122 in the SAS Guide to Macro Processing, Version 5 Edition.

Interactive Data Entry

The macro language statements %PUT and %INPUT are used to write information to a terminal and to retrieve information that has been entered at a terminal. These functions are accomplished in Release 5.16 of the SAS System running under VMS with the following Institute-supplied sample functions: TTOPEN, TTWRITE, TTREAD, and TTCLOSE. These functions are distributed with the production tape and are documented in SAS Technical Report P-169, User-Written Functions, Formats, and Informats for the Version 5 SAS System under VMS. The functions are defined as:

TTOPEN opens a channel to a device. Subsequent terminal I/O functions use the channel that is opened by this function.

TTWRITE sends characters from a variable to the opened channel.

TTREAD gets characters from the channel and stores them in a string.

TTCLOSE closes the channel.

These functions are used for writing interactive applications. The example shown below uses these functions to provide data entry function from the terminal:

The final report is shown in Output 3.

Output 3 Salaries Calculated as a Percentage of Total Department and Store Salary

For a more detailed discussion of this example, see pages 121-122 in the SAS Guide to Macro Processing, Version 5 Edition.
When the program executes the following lines are displayed to the terminal:

```
PLEASE ENTER INFORMATION REQUESTED
ENTER A # AT ANY TIME TO STOP
```

At this point data can be entered at the terminal. The data entered are assigned to the variable INBUF. The value is then assigned to the variable STUDY. The process continues until a # is entered at the terminal.

### THE SUMMARY SYSTEM

Chapter 11 of the SAS Guide to Macro Processing, Version 5 Edition demonstrates how to write a system of macros. The example in that chapter illustrates a macro system that executes the SUMMARY procedure, modifies the output data set, and produces two reports. The data set and variables to be processed and the statistics requested are supplied by the user. As a final example, the techniques discussed throughout this paper are combined to create a SUMMARY system without macro.

The SUMMARY system is an interactive system that prompts the user for the name of the data set he wants summarized, the name of the CLASS variable/variables, the variables to summarize, the statistics requested, and whether or not to use the NWAY option. Then the user is asked if he wants to display two reports. The requested output is displayed to the terminal.

This example uses a permanent data set, SASDATA.WEATHER. The observations contain the maximum and minimum temperatures from two cities reported for a two-year period. The data set has variables CITY, DATE, DAY, MONTH, YEAR, HIGHTEMP, and LOWTEMP.

The system is organized into the following SAS programs:

- **SUMMARY.SAS** prompts the user for information and brings in the programs necessary to produce the requested reports. This is the driver program.
- **WORDS.SAS** extracts words from a list.
- **SUMM.SAS** constructs the PROC SUMMARY step.
- **SHAPE.SAS** modifies the output data set from PROC SUMMARY.
- **PRNT1.SAS** produces the first report.
- **PRNT2.SAS** produces the second report.

An example execution of the SUMMARY system is shown below:

```
PLEASE ENTER INFORMATION REQUESTED
ENTER A # AT ANY TIME TO STOP
```

Enter the name of the data set to be summarized: SASDATA.WEATHER

Enter CLASS variables: CITY

Enter variables to be summarized: HIGHTEMP LOWTEMP

Enter statistics request from the following list:

- SUM
- MEAN
- VAR
- STD
- MIN
- MAX
- RANGE

Do you want to display FIRST report(Y/N): Y

Do you want to display SECOND report(Y/N): Y

Execution produces the FIRST and SECOND reports, which are shown in Output 4 and Output 5, respectively.

### The FIRST Report

#### CITY IS SEATTLE

<table>
<thead>
<tr>
<th>STATISTIC</th>
<th>HIGHTEMP</th>
<th>LOWTEMP</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN</td>
<td>13.0</td>
<td>63.0</td>
<td>50.0</td>
</tr>
<tr>
<td>MAX</td>
<td>112.0</td>
<td>76.0</td>
<td>44.0</td>
</tr>
</tbody>
</table>

#### CITY IS SPOKANE

<table>
<thead>
<tr>
<th>STATISTIC</th>
<th>HIGHTEMP</th>
<th>LOWTEMP</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN</td>
<td>11.0</td>
<td>65.0</td>
<td>54.0</td>
</tr>
<tr>
<td>MAX</td>
<td>100.0</td>
<td>76.0</td>
<td>31.0</td>
</tr>
</tbody>
</table>

### Output 4 The FIRST Report

### The SECOND Report

#### CITY IS SEATTLE

<table>
<thead>
<tr>
<th>STATISTIC</th>
<th>HIGHTEMP</th>
<th>LOWTEMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN</td>
<td>13.0</td>
<td></td>
</tr>
<tr>
<td>MAX</td>
<td>112.0</td>
<td></td>
</tr>
<tr>
<td>RANGE</td>
<td>50.0</td>
<td></td>
</tr>
</tbody>
</table>

#### CITY IS SPOKANE

<table>
<thead>
<tr>
<th>STATISTIC</th>
<th>HIGHTEMP</th>
<th>LOWTEMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN</td>
<td>11.0</td>
<td></td>
</tr>
<tr>
<td>MAX</td>
<td>100.0</td>
<td></td>
</tr>
<tr>
<td>RANGE</td>
<td>54.0</td>
<td></td>
</tr>
</tbody>
</table>

### Output 5 The SECOND Report

**SUMMARY.SAS**

SUMMARY.SAS, shown below, prompts the user for information at the terminal and then brings in the programs that are needed for processing.

```
DATA A;
LENGTH INBUF $ 50;
INBUF = ' '; CHANNEL = 0;
OPEN THE CHANNEL FOR INPUT
RC = TTOPEN('DEV=T90',CHANNEL);
CC = 'O010X I 0A10';
* THIS SECTION WILL ASSIGN THE VARIABLE INFORMATION TO * MACRO VARIABLES WHICH WILL BE USED FOR LATER PROCESSING;
RC = TTWRITE(CHANNEL,'PLEASE ENTER INFORMATION REQUESTED') ;
RC = TTWRITE(CHANNEL,'ENTER A # AT ANY TIME TO STOP');
RC = TTWRITE(CHANNEL,CC)
```

Enter the name of the data set to be summarized: SASDATA.WEATHER

Do you want the NWAY option (Y/N): Y

Enter CLASS variables: CITY

Enter variables to be summarized: HIGHTEMP LOWTEMP

Enter statistics request from the following list:

- SUM
- MEAN
- VAR
- STD
- MIN
- MAX
- RANGE

Do you want to display FIRST report(Y/N): Y

Do you want to display SECOND report(Y/N): Y
SUMM.SAS

SUMM.SAS, shown below, generates a SAS program, PROCSUM.SAS, that contains a PROC SUMMARY step from information supplied by the user and WORDS.SAS.

DATA _NULL_;
LENGTH VARS &NUMVARS;
LENGTH STATS &NUMSTATS;
LENGTH ROOT &NUMROOTS;
LENGTH VSTRING &NUMVARS;
LENGTH RETURN &NUMSTATS;
LENGTH SSTRING &NUMSTATS;
DATA _NULL_;
FILE 'PROCSUM.SAS';
PUT 'PROC SUMMARY DATA=README;';
PUT 'CLASS=README;';
PUT 'VARS=README;';
RUN;

SHAPE.SAS

SHAPE.SAS, shown below, constructs two DATA steps that modify the output data set generated by PROC SUMMARY.

DATA _NULL_;
FILE 'TEMP1.SAS';
PUT 'DATA TEMP1;';
PUT 'SET TEMP;'
PUT 'LENGTH CLASS &NUMCLASS;
PUT 'CLASS=OVERALL;';
PUT 'MACVAR=CLASS;';
PUT 'MACVAR=CLASS;';
RUN;

DATA _NULL_;
FILE 'TEMP2.SAS';
PUT 'DATA TEMP2;';
PUT "SET TEMP1;"
DO I = 1 TO NUMVARS;
  DO J = 1 TO NUMSTATS;
    PUT "CTJ. I. J. 1. J. 1. ;";
  ENDD;
PUT "OUTPUT;"
END;
PUT "RUN;"
RUN;

When SHAPE.SAS is executed, the files TEMP1.SAS and TEMP2.SAS are created. These files are included to create the reshaped data sets.

TEMP1.SAS contains

DATA TEMP1;
  SET TEMP;
  LENGTH CLASS $ 2M;
  IF _TYPE_ = 0 THEN CLASS = 'OVERALL';
  ELSE DO;
    CLASS = ';
    IF ICLASS = '1' THEN CLASS = TRIM(CLASS) II TRIM(LEFT(I));
    CLASS = SUBSTR(CLASS, II);
  ENDD;
RUN;

A partial listing of the TEMP1 data set is shown in Output 6.

<table>
<thead>
<tr>
<th>OBS</th>
<th>CITY</th>
<th><em>TYPE</em></th>
<th>FREQ</th>
<th>S11</th>
<th>S21</th>
<th>S31</th>
<th>S12</th>
<th>S22</th>
<th>S32</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SEATT</td>
<td>1</td>
<td>646</td>
<td>1</td>
<td>13</td>
<td>62</td>
<td>63</td>
<td>89</td>
<td>76</td>
<td>CITY IS SEATTLE</td>
</tr>
<tr>
<td>2</td>
<td>SPOK</td>
<td>1</td>
<td>646</td>
<td>-13</td>
<td>16</td>
<td>65</td>
<td>60</td>
<td>89</td>
<td>76</td>
<td>CITY IS SPOKANE</td>
</tr>
</tbody>
</table>

Output 6 Partial Listing of the TEMP1 Data Set

TEMP2.SAS contains

DATA TEMP2;
  SET TEMP1;
  VARLIST = S11 S21 S31 S12 S22 S32;
  LABEL S11 = "MIN*HIGHTEMP" S12 = "HIN*LOWTEMP" S21 = "MAX*HIGHTEMP" S22 = "MAX*LOWTEMP" S31 = "RANGE*HIGHTEMP" S32 = "RANGE*LOWTEMP";
  CLASS = "1;"
  FORMAT VARLIST;
RUN;

A partial listing of the data set TEMP2 is shown in Output 7.

<table>
<thead>
<tr>
<th>CITY</th>
<th><em>TYPE</em></th>
<th>FREQ</th>
<th>S11</th>
<th>S21</th>
<th>S31</th>
<th>S12</th>
<th>S22</th>
<th>S32</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEATT</td>
<td>1</td>
<td>646</td>
<td>1</td>
<td>13</td>
<td>62</td>
<td>63</td>
<td>89</td>
<td>76</td>
<td>CITY IS SEATTLE</td>
</tr>
<tr>
<td>SPOK</td>
<td>1</td>
<td>646</td>
<td>-13</td>
<td>16</td>
<td>65</td>
<td>60</td>
<td>89</td>
<td>76</td>
<td>CITY IS SPOKANE</td>
</tr>
</tbody>
</table>

Output 7 Partial Listing of the TEMP2 Data Set

PRNT1.SAS

PRNT1.SAS, shown below, generates a SAS program, PRNT1.SAS, that contains a PROC PRINT step that prints the first data set created by TEMP1.SAS.

DATA _NULL_;
LENGTH VARLIST $ 50;
VARLIST = ";
FILE 'PRINT1.SAS';
PUT 'PROC PRINT UNIFORM DATA=TEMP1 SPLIT="";';
PUT 'BY NOTSORTED CLASS;';
PUT 'ID CLASS;';
DO I = 1 TO NUMVARS;
  DO J = 1 TO NUMSTATS;
    VARLIST = TRIM(VARLIST) II 'I' II TRIM(LEFT(I));
    LABEL = '"' II TRIM(SYMGET('SYMPUT(I,1.))') II '"' II TRIM(LEFT(I));
    PUT "S" I J 1. '=' LABEL;
  ENDD;
END;
RUN;

The generated program, PRNT1.SAS contains the statements shown below:

PROC PRINT UNIFORM DATA=TEMP1 SPLIT="";
BY NOTSORTED CLASS;
ID CLASS;
VAR S11 S21 S31 S12 S22 S32;
LABEL S11 = "MIN*HIGHTEMP" S12 = "HIN*LOWTEMP" S21 = "MAX*HIGHTEMP" S22 = "MAX*LOWTEMP" S31 = "RANGE*HIGHTEMP" S32 = "RANGE*LOWTEMP";
CLASS = "1;"
FORMAT S11 S21 S31 S12 S22 S32 10.1;
RUN;
PRNT2.SAS, shown below, generates a SAS program, PRINT2.SAS, that prints the second data set, TEMP2, produced by TEMP2.SAS.

```
DATA _NULL_;
FILE 'PRINT2.SAS';
PUT 'PROC PRINT UNIFORM DATA=TEMP2 SPLIT="4"';
PUT 'BY MORTONITED CLASS;';
PUT 'ID CLASS;';
PUT 'VAR STAT LOWTEMP;'
PUT 'FORMAT STAT $8. LOWTEMP 10.1;'
PUT 'LABEL STAT=REQUESTED STATISTIC"';
DO I = 1 TO NUMVARS;
  PUT I TO STAT I , "="I, "="I , "="I;'
END;
PUT 1 TO 'CLASS="1"';
PUT 'RUN';
```

The generated program, PRINT2.SAS, contains the statements

```
PROC PRINT UNIFORM DATA=TEMP2 SPLIT="4";
BY MORTONITED CLASS;
VAR STAT LOWTEMP;
FORMAT STAT $8. LOWTEMP 10.1;
LABEL STAT=REQUESTED STATISTIC
"CLASS="";
RUN;
```

For more detailed information on the SUMMARY system refer to Chapter 11 in the SAS Guide to Macro Processing, Version 5 Edition. Examples are provided as a demonstration of how macro programs can be converted to execute in the minicomputer environment.

**CONCLUSION**

The examples in this paper demonstrate techniques that can be used to create programs that have the same functions as programs written with the SAS macro facility. Techniques that allow you to write generic programs that can be reexecuted with varying information are shown. The DATA step PUT statement allows you to generate programs that are dependent on data values. The %INCLUDE statement combined with the SYMPUT routine is used to conditionally execute PROC steps or DATA steps. The SYMGET function gives you the capability to access data across steps. The user-written functions TTOPEN, TTWRITE, TTREAD, and TTTCLOSE provide you with the capability to write interactive applications.

The techniques presented in this paper can be combined to solve complex programming problems that previously seemed unsolvable.

**REFERENCES**


SAS is a registered trademark of SAS Institute Inc., Cary, NC.

VMS is a trademark of Digital Equipment Corp.

PRIMOS is a registered trademark of Prime Computer, Inc.