CHAPTER 1

Introduction

Imagine you have an assistant to help you write your SAS programs. Your assistant willingly and unfailingly follows your instructions allowing you to move on to other tasks. Repetitive programming assignments like multiple PROC TABULATE tables where the only difference between one table and the next is the classification variable are delegated to your assistant. Jobs that require you to run a few steps, review the output, and then run additional steps based on the output are not difficult; they are, however, time-consuming. With instructions on selection of subsequent steps, your assistant easily handles the work. Even having your assistant do simple tasks like editing information in TITLE statements makes your job easier.

Actually, you already have a SAS programming assistant: the SAS macro facility. The SAS macro facility can do all the tasks above and more. To have the macro facility work for you, you first need to know how to communicate with the macro facility. That's the purpose of this book: to show you how to communicate with the SAS macro facility so that your SAS programming can become more effective and efficient.

An infinite variety of applications of the SAS macro facility exist. An understanding of the SAS macro facility gives you confidence to appropriately use it to help you build your SAS programs. The more you use the macro facility, the more adept you become at using it. As your skills increase, you discover more situations where the macro facility can be applied. The macro programming skills you learn from this book can be applied throughout the SAS System.

You do not have to use any of the macro facility features to write good SAS programs, but if you do you might find it easier to complete your SAS programming assignments. The SAS programming language can get you from one floor to the next, one step after another. Using the

macro facility wisely is like taking an elevator to get to a higher floor: you follow the same path, but you'll likely arrive at your destination sooner.

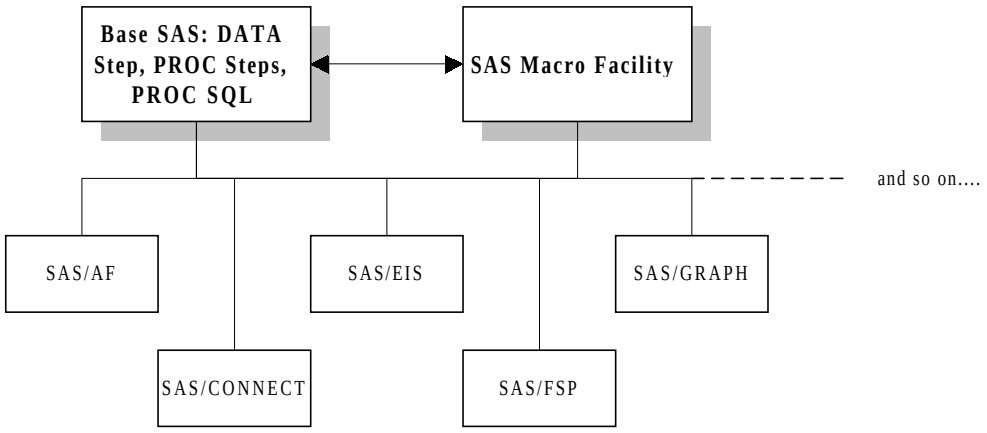
What Is the SAS Macro Facility?

Fundamentally, the SAS macro facility is a tool for text substitution. You associate a macro reference with text. When the macro processor encounters that reference, it replaces the reference with the associated text. This text can be as simple as text strings or as complex as SAS language statements. The macro processor becomes your SAS programming assistant in helping you construct your SAS programs.

The SAS macro facility is a component of base SAS. The base SAS product is integral to the SAS System and must be installed at your computing location if you want to write SAS programs or run SAS procedures in any of the SAS products. Therefore, if you have access to the SAS System, you have access to the macro facility and you can include macro facility features in your programs. Indeed, many of the SAS products that you license contain programs that use the macro facility.

The SAS macro facility works side-by-side with base SAS to build and execute your programs (Figure 1.1). The macro facility has its own language distinct from the SAS language, but the language and conventions of the macro facility are similar to the style and syntax of the SAS language. If you already write DATA steps, you have a head start on understanding the language and conventions of the macro facility.

Figure 1.1 How the SAS Macro Facility fits into the SAS System



The two main tools of the SAS macro facility are SAS macro variables and SAS macro programs. With SAS macro variables, you create references to larger pieces of text. A macro variable is

typically used to repeatedly insert a piece of text throughout a SAS program. SAS macro programs use macro variables and macro programming statements to build SAS programs. Macro programs can direct conditional execution of DATA steps and PROC steps. Macro programs can do repetitive tasks like creating or analyzing a series of data sets.

The following program uses a macro variable to select a subset of a data set. Information about the subset is included in the title. Macro language and references are highlighted.

```
%let mosold=4;
proc print data=books.ytdsales
                  where=(month(datesold)=&mosold));
  title "Books Sold for Month &mosold";
  var title salepric;
  sum salepric;
run;
```

Next is a macro program that when executed runs a PROC step on three different data sets. The macro language and references that generate the three steps are highlighted.

```
%macro sales;
  %do year=1997 %to 1999;
    proc means data=books.sold&year;
      title "Sales Information for &year";
      class section;
      var listpric salepric;
    run;
  %end;
%mend sales;
```

The macro facility was first released in SAS version 82.3 in 1982. There are few statements in the macro language, but these are very powerful. Compared to other software features, the macro language has changed little since its introduction and relatively little has been added to the language.

In a world of rapidly changing software tools and techniques, the macro facility remains one of the most widely used components of the SAS System. What you learn now about the macro facility will serve you for many years of SAS programming.

What Are the Advantages of the SAS Macro Facility?

Your SAS programming productivity can improve when you know how and when to use the SAS macro facility. The programs you write can become reusable, shorter, and easier to follow.

In addition, by incorporating macro facility features in your programs you can:

- Accomplish repetitive tasks quickly and efficiently. A macro program can be reused many times. Parameters passed to the macro program customize the results without having to change the code within the macro program.

- Provide a more modular structure to your programs. SAS language that is repetitive can be generated by macro language statements in a macro program and that macro program can be referenced in your SAS program. The reference to the macro program is similar to calling a subroutine. The main program becomes easier to read - especially if you give the macro program a meaningful name for the function that it performs.

Think about automated bill paying as a real world example of the concepts of macro programming. When you enroll in an automated bill paying plan, you no longer write checks each month to pay recurring bills like the mortgage, the utilities, the telephone, and so on. Without automated bill paying, it takes a certain amount of time each month for you to write checks to pay those recurring bills. The time that it takes to initiate the automated bill paying plan is likely longer in the month that you set it up than if you just wrote the checks for the monthly bills. But, once you have the automated bill paying plan established (and perhaps allowing the bank a little debugging time!), the amount of time you spend each month dealing with those recurring bills is reduced. You instruct your bank how to handle those recurring bills. In turn they, in effect, write all those checks for you.

That's what macro programming can do for you. Instead of editing the program each time parameters change (for example, same analysis program, different data set), you write a SAS program that contains macro language statements. These macro language statements instruct the macro processor how to make those code changes for you. Then, when you run the program again, the only changes you make are to the values that the macro language uses to edit your program. Like directing the bank to add the water department to your automatic payment plan.

Example: Macro Programming Using Macro Variables

Consider another illustration of macro programming, this time including a sample program. The data set that is analyzed here is used throughout this book. The data represent computer book sales at a fictitious bookstore.

The following program is a monthly sales report for the computer section of the bookstore. If you were not using macro facility features, you would have to change the program every month at every location where the month value was referenced. You would also have to change the program once a year at every location where the year value was referenced.

Rather than doing these multiple edits, you can create macro variables at the beginning of the program that are set to the month and the year of interest, and place references to these macro variables throughout the program where they are needed. When you get ready to submit the

program, the only changes you make are to the values of the macro variables. After you submit the program, the macro processor looks up the values of month and year that you set and substitutes those values as specified by your macro variable references.

In summary, you don't edit the DATA step and the PROC steps; you only change the values of the macro variables at the beginning of the program. The report layout stays the same, but the results are based on a different subset of the data set.

Don't worry about understanding the macro language coding at this point. Just realize that you can reuse the same program with different parameters to analyze a different subset of the data set. Macro language statements start with a percent sign (%). Macro variable references start with an ampersand (&). Both features are highlighted in the following code.

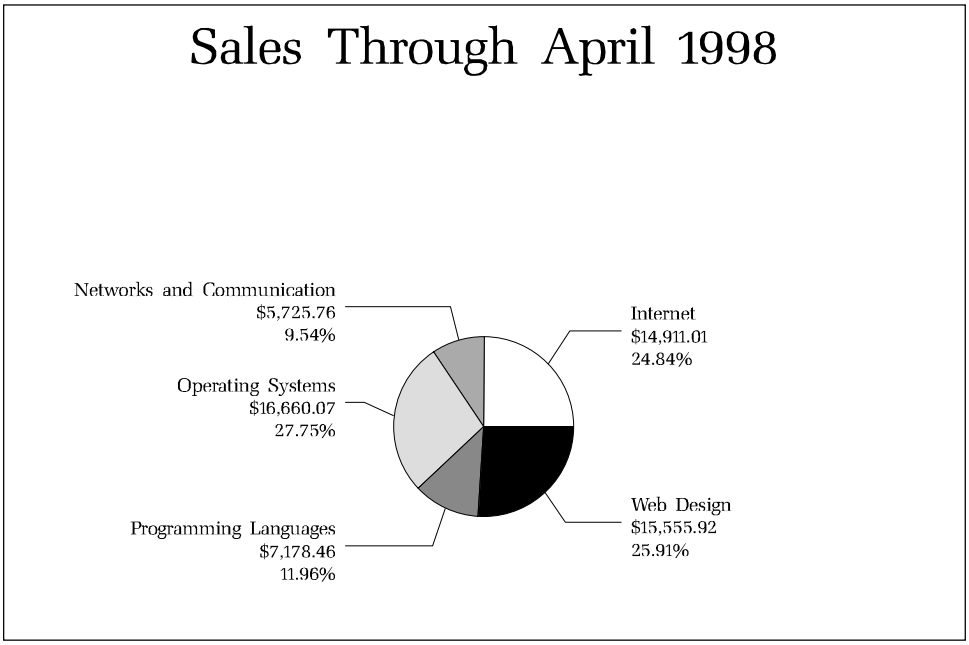
```
%let repmonth=4;
%let repyear=1998;
%let repmword=%sysfunc(mdy(&repmonth,1,&repyear),monname9.);

data month&repmonth;
  set books.ytdsales;
  mosale=month(datesold);
  label mosale='Month of Sale';
run;
proc tabulate data=month&repmonth;
  title "Sales During &repmword &repyear";
  where mosale=&repmonth and year(datesold)=&repyear;
  class section;
  var salepric listpric cost;
  tables section all='**TOTAL**',
         (salepric listpric cost)*(n*f=4. sum*f=dollar9.2);
run;

proc gchart data=month&repmonth
           (where=(mosale < %eval(&repmonth+1) and
                  year(datesold)=&repyear));
  title "Sales Through &repmword &repyear";
  pie section / coutline=black percent=arrow
                sumvar=salepric noheading ;
run;

quit;
```

The output for this program is in Figure 1.2.

Figure 1.2 Output for sample program illustrating SAS macro programming

Sales During April 1998 1

	Sale Price		List Price		Wholesale Cost	
	N	SUM	N	SUM	N	SUM
Section						
Internet	145	$4579.71	145	$4680.75	145	$3318.77
Networks and Communication	55	$1633.01	55	$1665.25	55	$1177.46
Operating Systems	132	$4016.45	132	$4108.40	132	$2916.03
Programming Languages	60	$1835.07	60	$1878.00	60	$1330.98
Web Design	131	$4015.50	131	$4114.45	131	$2910.87
TOTAL	523	$16079.74	523	$16446.85	523	$11654.09



Changing just the first line of the program from

```
%let repmonth=4;
```

to

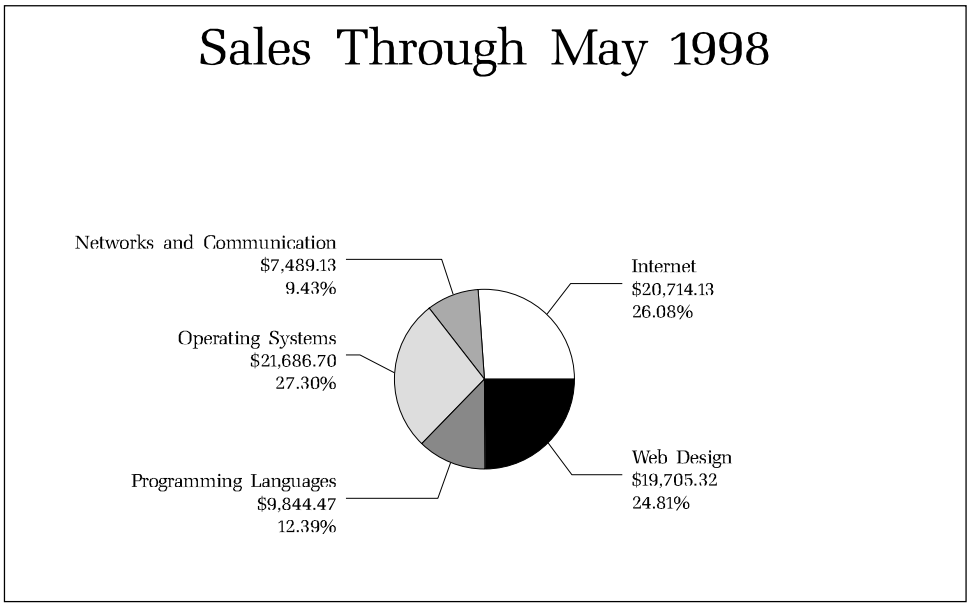
```
%let repmonth=5;
```

runs the same program, but now processes the data collected for May. No other editing of the program is required to process this subset. The output for May is shown in Figure 1.3.

Figure 1.3 Output for revised sample program

Sales During May 1998 1

	Sale Price		List Price		Wholesale Cost	
Section	N	SUM	N	SUM	N	SUM
Internet	190	$5803.11	190	$5949.50	190	$4213.08
Networks and Communication	60	$1763.37	60	$1805.00	60	$1280.48
Operating Systems	165	$5026.63	165	$5132.75	165	$3633.16
Programming Languages	90	$2666.00	90	$2727.50	90	$1931.61
Web Design	135	$4149.40	135	$4241.25	135	$3000.62
TOTAL	640	$19408.52	640	$19856.00	640	$14058.95



Where Can the SAS Macro Facility Be Used?

The macro facility can be used with all SAS products. We've seen in the monthly sales report an example of macro programming in base SAS.

Table 1.1 lists some SAS products and possible macro facility applications you can create. It also lists existing macro applications that come with the SAS product.

Table 1.1 SAS Macro Facility applications

SAS Product	Typical Applications of the Macro Facility
Base SAS	Customizes data set processing Customizes PROC steps Customizes reports Passes data between steps in a program Conditionally executes DATA steps and PROC steps Iteratively processes DATA steps and PROC steps Contains libraries of macro routines for working with macro variables
SAS Component Language*	Communicates between SAS program steps and SCL programs Communicates between SCL programs
SAS/CONNECT	Passes information between local and remote SAS sessions
SAS/GRAPH	Contains libraries of macro routines for annotating SAS/GRAPH output
SAS/IntrNet	Contains libraries of macro programs for formatting SAS output for web pages
SAS/TOOLKIT	Creates functions that can be used with the macro facility

*Note: For Version 7 of the SAS System, Screen Control Language is referred to as SAS Component Language, and it is abbreviated throughout this book as SCL.

Examples of the SAS Macro Facility

The following examples of the SAS macro facility illustrate some of the tasks that the macro processor can perform for you. There's no need to understand the coding of these programs at this point (although the code is included and may be useful to you later). What you should gain from this section is an idea of the kinds of SAS programming tasks that can be delegated to the macro processor.

Example: Displaying SAS System Information with Macro Variables Automatically Defined by the SAS System

The SAS System comes with a set of automatic macro variables that you can reference in your SAS programs. Most of these macro variables deal with system-related items like date, time, operating system, and version of SAS. Using these automatically defined macro variables is the simplest application of the macro facility. The following example incorporates some of these

automatic macro variables. Note that the automatic macro variable names are preceded by ampersands. Assume the report was run on May 31, 1998. The report shows sales for 1998 through May 31.

```
title "Sales Report";
title2 "As of &systime &sysday &sysdate";
title3 "Using SAS Version: &sysver";
proc means data=books.ytdsales n sum;
  var salepric;
run;
```

The output for this program is in Figure 1.4.

Figure 1.4 Output for program using automatically defined SAS macro variables

```
                        Sales Report
                   As of 15:48 Sunday 31MAY98
                     Using SAS Version: 6.12

                      Analysis Variable : SALEPRIC Sale Price


                         N           Sum
                      ------------------
                      2618      79439.74
                      ------------------
```

Example: Substituting Information Multiple Times

The example that started on page 5 demonstrated reuse of the same program by simply changing the values of the macro variables at the beginning of the program. A new subset of data is analyzed each time the values of the macro variables are changed.

It is relatively simple to create and reference these macro variables. Besides being able to reuse your program code, the advantages in using macro variables include reducing coding time and reducing programming errors by not having to manage so many lines of code.

Example: Conditional Processing of DATA Steps and PROC Steps

Macro programs can use macro variables and macro programming statements to select the steps and the SAS language statements executed in a SAS program. These conditional processing macro language statements are similar in syntax and structure to SAS language statements.

There are two PROC steps in the macro program below. The first PROC MEANS step runs daily. The second PROC MEANS step runs only on Fridays. The conditional macro language statements direct the macro processor to run the second PROC step only on Fridays. Assume the program was run on Friday, August 21, 1998.

Macro language statements start with percent signs and macro variable references start with ampersands.

```
%macro daily;
  proc means data=books.ytdsales(where=(datesold=today()))
                  maxdec=2 sum;
    title "Daily Sales Report for &sysdate";
    class section;
    var salepric;
  run;
  %if &sysday=Friday %then %do;
     proc means data=books.ytdsales
          (where=(today()-6 le datesold le today()))
          sum maxdec=2;
       title "Weekly Sales Report Week Ending &sysdate";
       class section;
       var salepric;
     run;
  %end;
%mend daily;

%daily
```

The output for this program is in Figure 1.5.

Figure 1.5 Output for sample program that uses conditional macro language statements

Daily Sales Report for 21AUG98

Analysis Variable : SALEPRIC Sale Price

SECTION	N Obs	Sum
Internet	6	176.70
Networks and Communication	2	65.90
Operating Systems	5	134.75
Programming Languages	1	24.95
Web Design	3	85.85

Weekly Sales Report Week Ending 21AUG98

Analysis Variable : SALEPRIC Sale Price

SECTION	N Obs	Sum
Internet	38	1112.33
Networks and Communication	7	236.76
Operating Systems	33	1055.58
Programming Languages	11	326.45
Web Design	41	1248.99

Example: Iterative Processing

Coding each iteration of a programming process that contains multiple iterations is a lengthy task. The %DO loops in the macro language can take over some of that iterative coding for you. A macro program can build the code for each iteration of a repetitive programming process based on the specifications of the %DO loop.

As an illustration of iterative processing, the following macro program creates 12 data sets, one for each month of the year. Without macro programming, you would have to enter the 12 data set names in the DATA statement and enter all the ELSE statements that direct observations to the right data set. A macro language %DO loop can build those statements for you.

```
%macro makesets;
  data
    %do i=1 %to 12;
      month&i
    %end;
    ;
    set books.ytdsales;
    mosale=month(datesold);
    if mosale=1 then output month1;
    %do i=2 %to 12;
      else if mosale=&i then output month&i;
    %end;
  run;
%mend makesets;

%makesets
```

After interpretation by the macro processor, the program becomes:

```
 data month1 month2 month3 month4 month5 month6
      month7 month8 month9 month10 month11 month12
      ;
   set books.ytdsales;
   mosale=month(datesold);
   if mosale=1 then output month1;
   else if mosale=2 then output month2;
   else if mosale=3 then output month3;
   else if mosale=4 then output month4;
   else if mosale=5 then output month5;
   else if mosale=6 then output month6;
   else if mosale=7 then output month7;
   else if mosale=8 then output month8;
   else if mosale=9 then output month9;
   else if mosale=10 then output month10;
   else if mosale=11 then output month11;
   else if mosale=12 then output month12;
 run;
```

Macro language statements built the SAS language DATA statement and all of the DATA step ELSE statements for you.

A few macro programming statements direct the macro processor to build the complete DATA step for you. By doing this, you avoid the tedious task of entering all the data set names and all those ELSE statements. Repetitive coding tasks are a breeding ground for bugs in your programs. Thus, turning these tasks over to the macro processor may reduce the number of errors in your SAS programs.

Example: Passing Information between Program Steps

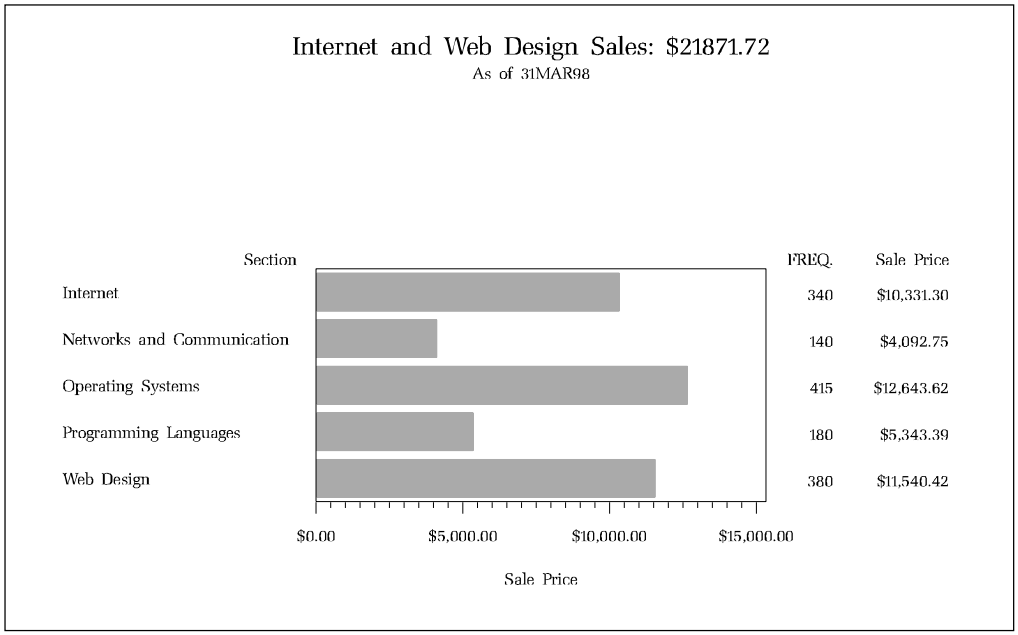
The macro facility can act as a bridge between steps in your SAS programs. The SAS language functions that interact with the macro facility can transfer information between steps in your SAS programs.

The program below calculates total sales for two sections in the computer department of the bookstore. That value is then inserted in the TITLE statement of the PROC GCHART output. The SYMPUT SAS language routine instructs the macro processor to hold on to the total sales value after the DATA step finishes. The total sales value is then available to subsequent steps in the program.

The output for the following program is in Figure 1.6.

```
data temp;
  set books.ytdsales end=lastobs;
  retain sumintwb 0;
  if section in (‘Internet','Web Design') then
    sumintwb=sumintwb + salepric;
  if lastobs then
    call symput(‘INTWEBSL',put(sumintwb,dollar10.2));
run;
proc gchart data=temp;
  title "Internet and Web Design Sales: &intwebsl";
  title2 "As of &enddate";
  hbar section / sumvar=salepric;
  format salepric dollar10.2;
run;

quit;
```

Figure 1.6 Output for program that passes data from a DATA step to a PROC step



Without the SYMPUT routine, you have to submit two programs. The first SAS program calculates the total sales for the two sections. After the first program ends, you find the total sales value in the output. Then you have to edit the second program and update the TITLE statement with the total sales value that you copied from the output of the first program.

Example: Interface to SAS Language Functions

The SAS language has libraries of functions that can be used in your macro language programs. Some uses of these functions include incorporating information about a data set in a title, checking the existence of a data set, and finding the number of observations in a data set.

In the following macro language program, a report is produced for a specified data set. The data set name is passed as a parameter to the macro program. The macro program determines the number of observations in the data set and the date that the data set was created. This information is inserted in the title. PROC MEANS then produces descriptive statistics.

```
%macro dsreport(dsname);
  title "Report on Data Set &dsname";
```

```
  %let dsid=%sysfunc(open(&dsname));

  %*----How many obs are in the data set?;
  %let nobs=%sysfunc(attrn(&dsid,nobs));

  %*----When was the data set created?;
  %let when = %sysfunc(putn(
                %sysfunc(attrn(&dsid,crdte)),datetime9.));

  title2 "Num Obs: &nobs   Date Created: &when";

  proc means data=&dsname sum maxdec=2;
    class section;
    var salepric;
  run;
%mend dsreport;

%dsreport(books.ytdsales)
```

The output for the above program is in Figure 1.7.

Figure 1.7 Output for macro program using SAS language functions

```
            Report on Data Set books.ytdsales
          Num Obs: 3489   Date Created: 15JUL98

        Analysis Variable : SALEPRIC Sale Price


        SECTION                      N Obs           Sum
        ------------------------------------------------
        Internet                       904      27619.62

        Networks and Communication     332       9869.86

        Operating Systems              950      28833.04

        Programming Languages          452      13445.51

        Web Design                     851      25973.70
        ------------------------------------------------
```

Example: SAS Component Language Programs and the Macro Facility

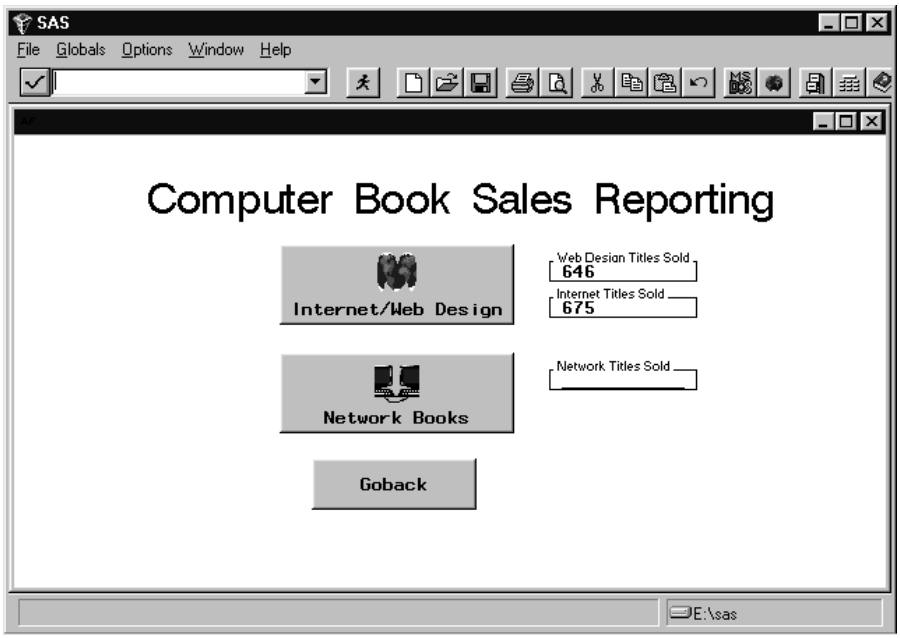
The macro facility can be used in SAS Component Language (SCL) programs to update SAS/AF screens, pass data between SCL programs, and pass data between SCL programs and SAS language programs.

In the following example, a SUBMIT block encloses a PROC FREQ step. The results of the PROC FREQ step are moved to macro variables using SAS language functions in a DATA step. The macro variables are also accessible to the SCL program. The SCL program obtains the values from the macro variables and then displays them on the SAS/AF screen. NWEB and NNET are SCL fields on the SAS/AF screen.

```
runrep:
  submit continue;
    proc freq data=books.ytdsales;
      tables section / out=freqs noprint;
    run;
    data _null_;
      set freqs;
      if section='Web Design' then
              call symput("WEBDESGN",count);
      else if section='Internet' then
              call symput("INTERNET",count);
    run;
  endsubmit;

  *----Update screen;
  nweb=symget("WEBDESGN");
  nnet=symget("INTERNET");
return;
```

Figure 1.8 shows an example of the above program. When the "Internet/Web Design" button is selected, the report runs, the macro variables are created, and the screen fields are updated.

Figure 1.8 Sample SAS/AF screen using a program that creates macro variables to update SAS/AF screen fields



How Can This Book Help You Understand the SAS Macro Facility?

This book is for beginning through experienced users of the SAS macro facility. It shows you how to delegate some of your SAS programming tasks to the macro facility.

This book assumes you have beginner to intermediate experience writing SAS language programs. SAS language and SAS programming concepts are not reviewed.

This book is less inclusive and spends less time on reference details than *SAS Macro Language: Reference*. Rather, the focus is on making the macro facility a tool you can use.

The technical aspects of macro processing are described in this book. While understanding the technical aspects is not necessary to begin to reap the benefits of the SAS macro facility, this knowledge may help you more wisely apply macro programming techniques.

Don't worry if the technical aspects are difficult to grasp at first. Instead, jump in and start using the simpler features of the macro facility. Try macro variables first. You're bound to make some

errors, but those errors help you understand macro processing. Eventually, as your macro programming skills improve, a more thorough understanding of macro processing can reduce the number of macro programming errors you make and make it easier to debug your programs.

This book starts with the easier features of the SAS macro facility. These features are building blocks for the later topics. The features of the macro facility are interrelated, and so occasionally you may see some features used before they are formally discussed.

Because macro facility features are interrelated, this book does not have to be read in a linear fashion. Work through sections as appropriate for your needs. Return to earlier sections when that information becomes pertinent. However, it is best to start with the technical information in Chapter 2 and move on to the macro variable chapter, Chapter 3. You might then work with macro variables extensively and try some of the features like macro functions and macro expressions that are described in Chapter 6. After gaining confidence in how macro variables work, you might try writing macro programs. You can learn how to do this in Chapter 4 and then try using the macro programming statements in Chapter 6.

You may find it useful to learn about macro facility interfaces before you cover macro programs. Chapter 7 includes information useful for SCL programmers, PROC SQL programmers, and DATA step programmers.

About the Data and Programs in this Book

The examples in this book are illustrated with sales data from a fictitious bookstore. The DATA step to create this data set is in Appendix C. A PROC CONTENTS listing for this data set is also in Appendix C.

The examples and screens in this book were produced using Release 6.12 of SAS under Windows 95. Where appropriate, information about Version 7 features is included.

The Typographical Styles in This Book

The following styles have special meaning when describing the syntax of macro language statements and SAS language statements in this book.

- Values in *`italics`* identify arguments and values that you supply.
- Arguments enclosed in angle brackets (`< >`) are optional.
- Arguments separated with a vertical bar (`|`) indicate mutually exclusive choices.

For example, the syntax of the %SYSEVALF function is written as follows:

```
%SYSEVALF(arithmetic expression|logical expression
    <,conversion-type>)
```

When specifying the %SYSEVALF function, you must specify either an arithmetic expression or a logical expression. Specifying a conversion type is optional.