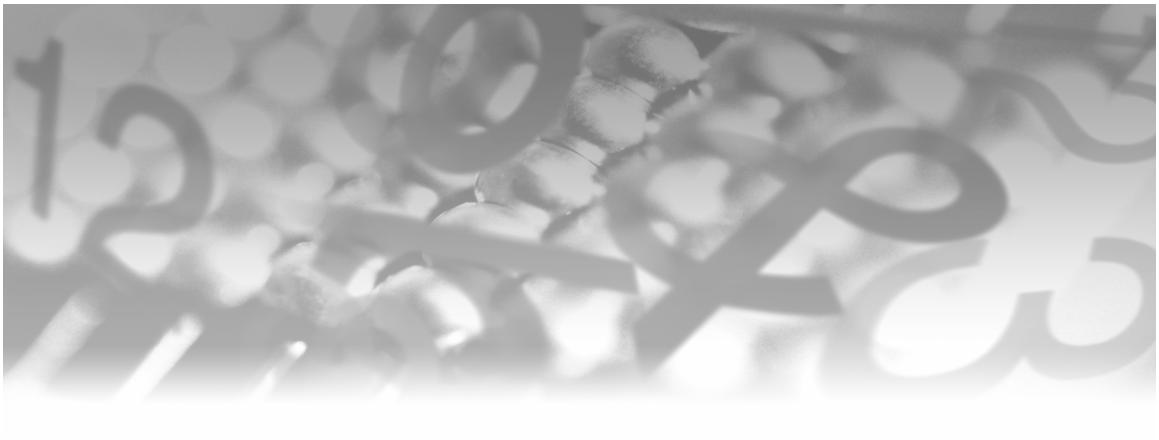


Part 1

Understanding the Concepts and Features of Macro Programming

Chapter 1 **Introduction 3**

Chapter 2 **Mechanics of Macro Processing 23**

Chapter 3 **Macro Variables 39**

Chapter 4 **Macro Programs 73**

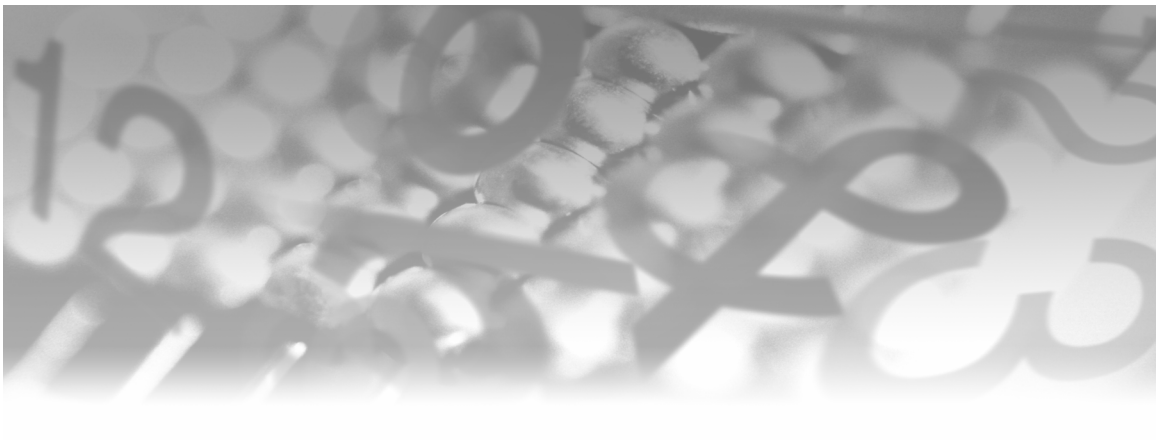
Chapter 5 **Understanding Macro Symbol Tables and the Processing of Macro Programs 101**

Chapter 6 **Macro Language Functions 133**

Chapter 7 **Macro Expressions and Macro Programming Statements 159**

Chapter 8 **Masking Special Characters and Mnemonic Operators 189**

Chapter 9 **Interfaces to the Macro Facility 217**



Chapter 1

Introduction

What Is the SAS Macro Facility? 4
What Are the Advantages of the SAS Macro Facility? 6
Where Can the SAS Macro Facility Be Used? 12
Examples of the SAS Macro Facility 13

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 SAS.

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.

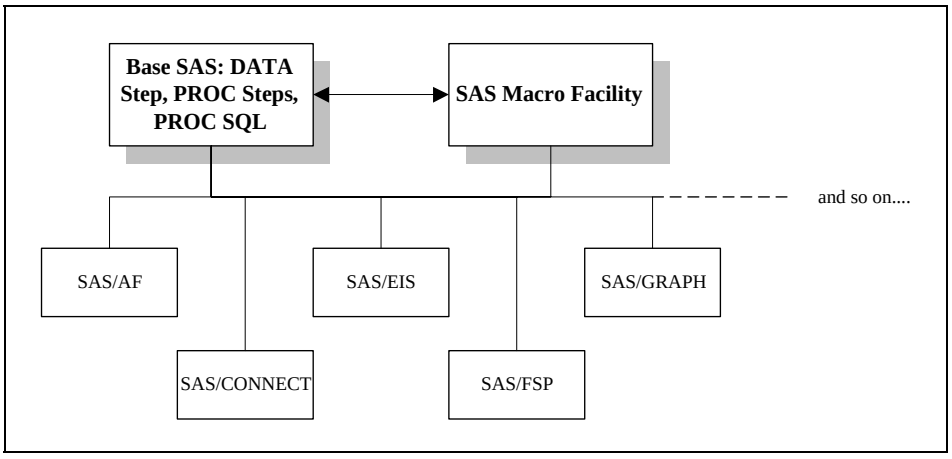
What Is the SAS Macro Facility?

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 SAS 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 SAS, 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.

As shown in Figure 1.1, the SAS macro facility works side-by-side with Base SAS to build and execute your programs. 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 SAS



The two main components 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 typical use of a macro variable is 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 such as creating or analyzing a series of data sets.

Example 1.1 shows how a macro variable can be used, and Example 1.2 shows how a macro program can be used.

Example 1.1: Using a Macro Variable to Select Observations to Process

The macro variable MONTH_SOLD defined in Program 1.1 is used to select a subset of a data set and place information in the report title. Macro language and macro variable references are in bold.

Program 1.1

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

Example 1.2: Using a Macro Program to Execute the Same PROC Step on Multiple Data Sets

When Program 1.2 executes, it submits a PROC MEANS step three times: once for each of the years 2007, 2008, and 2009. Each time, it processes a different data set. The macro language and references that generate the three steps are in bold.

Program 1.2

```
%macro sales;
  %do year=2007 %to 2009;
    proc means data=books.sold&year;
      title "Sales Information for &year";
      class section;
      var listprice saleprice;
    run;
  %end;
%mend sales;
%sales
```

The macro facility was first released in SAS 82.0 in 1982. There are relatively few statements in the macro language, and these statements are very powerful.

In a world of rapidly changing software tools and techniques, the macro facility remains one of the most widely used components of SAS. 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 initiate payments each month to pay recurring bills like the mortgage and the utilities. Without automated bill paying, it takes a certain amount of time each month for you to initiate payments 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 submitted a payment for each monthly bill. 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 initiate those monthly payments 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 1.3: Defining and 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.

Program 1.3 produces two reports for the computer section of the bookstore. The first is a monthly sales report. The second is a pie chart of sales from the beginning of the year through the month of interest.

If you were not using macro facility features, you would have to change the program every time you wanted the report for a different month and/or year. These changes would have to be made at every location where the month value and/or year value were 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.

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 be aware that you can reuse the same program to analyze a different subset of the data set by changing the values of the macro variables.

Note that macro language statements start with a percent sign (%) and macro variable references start with an ampersand (&). Both features are in bold in the following code.

Program 1.3

```
%let repmonth=4;
%let repyear=2007;
%let repmword=%sysfunc(mdy(&repmonth,1,&repyear),monname9.);

data temp;
  set books.ytdsales;
  mosale=month(datesold);
  label mosale='Month of Sale';
run;

proc tabulate data=temp;
  title "Sales During &repmword &repyear";
  where mosale=&repmonth and year(datesold)=&repyear;
  class section;
  var saleprice listprice cost;
  tables section all='**TOTAL**',
         (saleprice listprice cost)*(n*f=4. sum*f=dollar10.2);
run;

proc gchart data=temp
           (where=(mosale <= &repmonth and
                   year(datesold)=&repyear));
  title "Sales Through &repmword &repyear";
  pie section / coutline=black percent=outside
                sumvar=saleprice noheading ;
run;
quit;
```

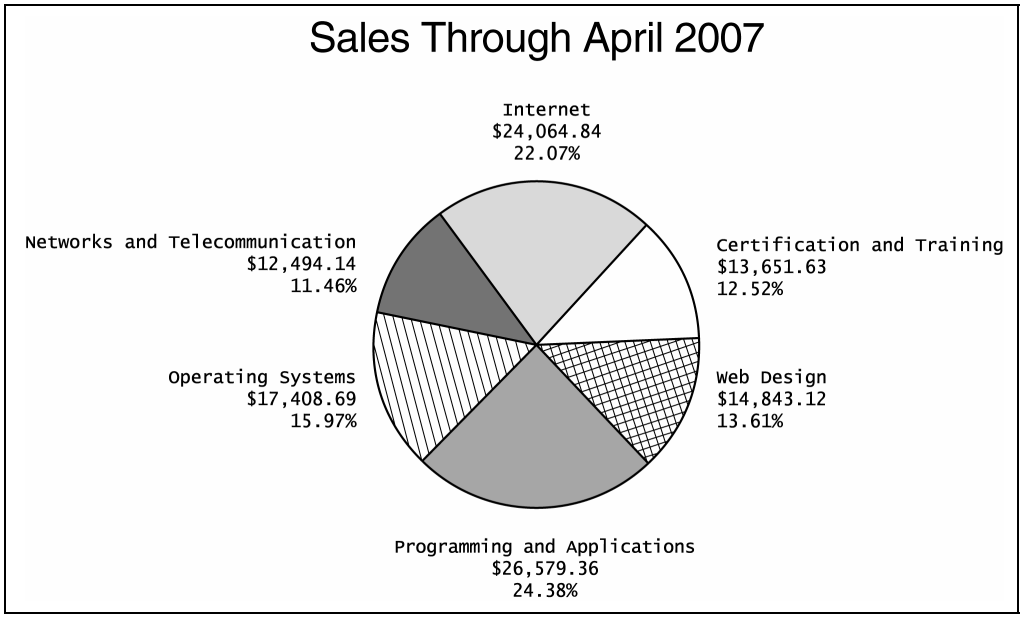
Output 1.3a presents the output from Program 1.3.

Output 1.3a Output from Program 1.3

Sales During April 2007

	Sale Price		List Price		Wholesale Cost	
	N	Sum	N	Sum	N	Sum
Section						
Certification and Training	62	$2,709.54	62	$2,745.90	62	$1,398.42
Internet	89	$3,896.43	89	$3,965.55	89	$2,018.03
Networks and Telecommunication	60	$2,627.57	60	$2,694.00	60	$1,376.47
Operating Systems	79	$3,467.53	79	$3,539.05	79	$1,780.11
Programming and Applications	130	$5,689.23	130	$5,806.50	130	$2,943.80
Web Design	57	$2,500.71	57	$2,559.15	57	$1,293.26
TOTAL	477	$20,891.01	477	$21,310.15	477	$10,810.10

(*continued*)



Changing just the first line of the program from

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

to

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

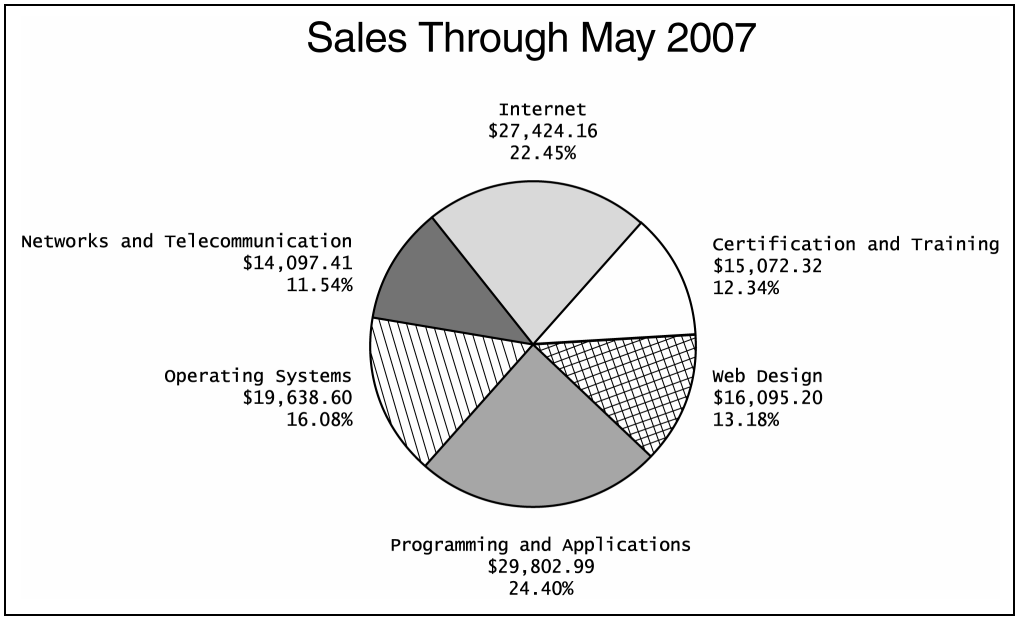
runs the same program, but now processes the data collected for May. No other editing of the program is required to process this subset. Output 1.3b presents the output for May.

Output 1.3b Output from revised Program 1.3

Sales During May 2007

	Sale Price		List Price		Wholesale Cost	
	N	Sum	N	Sum	N	Sum
Section						
Certification and Training	31	$1,420.69	31	$1,449.45	31	$741.81
Internet	79	$3,359.32	79	$3,425.05	79	$1,751.58
Networks and Telecommunication	38	$1,603.27	38	$1,660.10	38	$834.65
Operating Systems	51	$2,229.91	51	$2,284.45	51	$1,154.91
Programming and Applications	72	$3,223.63	72	$3,254.40	72	$1,639.38
Web Design	29	$1,252.09	29	$1,284.55	29	$650.97
TOTAL	300	$13,088.90	300	$13,358.00	300	$6,773.29

(*continued*)



Where Can the SAS Macro Facility Be Used?

The macro facility can be used with all SAS products. You'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 that you can create. It also lists existing macro applications that come with SAS.

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 program routines
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/TOOLKIT	Creates functions that can be used with the macro facility

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 might 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.

In addition to the examples that follow, Program 1.3 demonstrates 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 easy 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 edit so many lines of code.

Example 1.4: Displaying System Information

SAS 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 one of the simplest applications of the macro facility.

Program 1.4 incorporates some of these automatic macro variables, and these macro variables are in bold in the code. Note that the automatic macro variable names are preceded by ampersands. Assume the report was run on February 20, 2008.

Program 1.4

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

Output 1.4 presents the output from Program 1.4.

Output 1.4 Output for program using automatically defined SAS macro variables

```
                        Sales Report
                  As of 15:46 Wednesday 20FEB08
                    Using SAS Version: 9.1

                     The MEANS Procedure

            Analysis Variable : saleprice Sale Price

                        N             Sum
                    ---------------------
                    6096       263678.15
                    ---------------------
```

Example 1.5: Conditional Processing of SAS Steps

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

Macro program DAILY in Program 1.5 contains two PROC steps. 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 17, 2007.

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

Program 1.5

```
%macro daily;
  proc means data=books.ytdsales(where=(datesold=today()))
                  maxdec=2 sum;
    title "Daily Sales Report for &sysdate";
    class section;
    var saleprice;
  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 saleprice;
     run;
  %end;
%mend daily;

%daily
```

Output 1.5 presents the output from Program 1.5.

Output 1.5 Output from Program 1.5 that uses conditional macro language statements

```
              Daily Sales Report for 17AUG07                        1

                    The MEANS Procedure

          Analysis Variable : saleprice Sale Price
                                       N
Section                              Obs              Sum
-----------------------------------------------------------
Certification and Training             5           229.16
Internet                               7           312.96
Networks and Telecommunication         3           122.76
Operating Systems                      3           132.85
Programming and Applications           2           100.41
Web Design                             4           173.80
-----------------------------------------------------------

          Weekly Sales Report Week Ending 17AUG07                   2

                    The MEANS Procedure

          Analysis Variable : saleprice Sale Price

                                       N
Section                              Obs              Sum
-----------------------------------------------------------
Certification and Training            13           562.78
Internet                              28          1210.22
Networks and Telecommunication        15           645.16
Operating Systems                     27          1206.67
Programming and Applications          23          1007.57
Web Design                            14           610.61
-----------------------------------------------------------
```

Example 1.6: Iterative Processing of SAS Steps

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.

Program 1.6 illustrates iterative processing. It 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.

Program 1.6

```
%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 ELSE statements in the DATA step 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 the ELSE statements. Repetitive coding tasks are a breeding ground for bugs in your programs. Thus, turning these tasks over to the macro processor can reduce the number of errors in your SAS programs.

Example 1.7: 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.

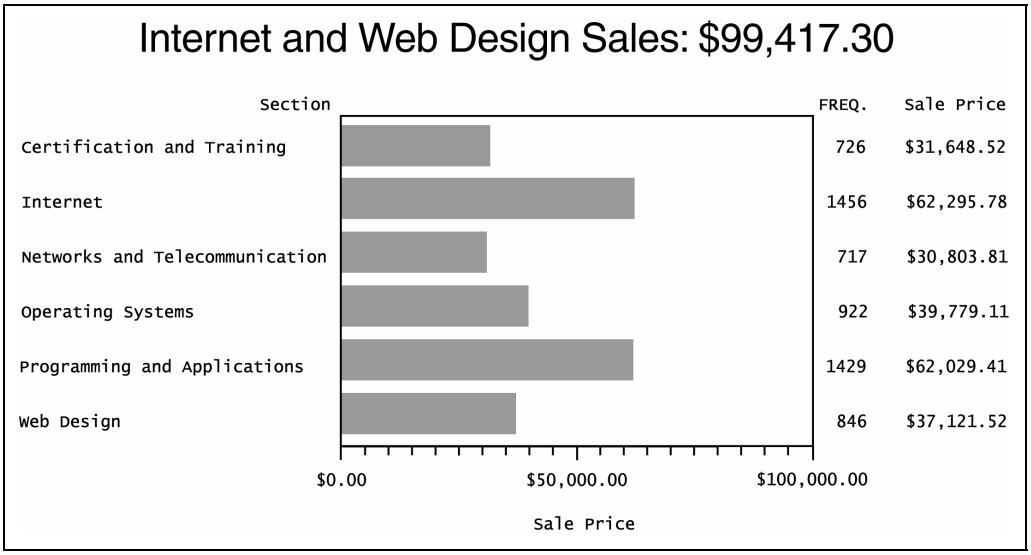
Program 1.7 calculates total sales for two sections in the computer department of the bookstore. That value is then inserted in the TITLE statement of the PROC GCHART output. The SYMPUTX SAS language routine instructs the macro processor to retain the total sales value in macro variable INTWEBSL after the DATA step finishes. The total sales value is then available to subsequent steps in the program.

Program 1.7

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

Output 1.7 presents the output from Program 1.7.

Output 1.7 Output from Program 1.7 that passes data from a DATA step to a TITLE statement



Without the SYMPUTX routine, you would have to submit two programs. The first SAS program would calculate the total sales for the two sections. After the first program ends, you find the total sales value in the output. Then, before submitting the second program, you would have to edit the second program and update the TITLE statement with the total sales value that you found in the output from the first program.

Example 1.8: Interfacing Macro Language and SAS Language Functions

The SAS language has libraries of functions that also 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.

Macro program DSREPORT in Program 1.8 produces a PROC MEANS report for a data set whose name is passed as a parameter to DSREPORT. The number of observations and creation date of the data set are obtained with the ATTRN SAS language function, and these attributes are inserted in the report title. The date value is formatted by the PUTN SAS language function. Macro program DSREPORT opens and closes the data set with the OPEN and CLOSE SAS language functions. The SAS language functions are underlined in Program 1.8.

Program 1.8

```
%macro dsreport(dsname);
  %*----Open 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.));

  %*----Close data set dsname identified by dsid;
  %let rc=%sysfunc(close(&dsid));

  title "Report on Data Set &dsname";
  title2 "Num Obs: &nobs   Date Created: &when";

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

%dsreport(books.ytdsales)
```

Output 1.8 presents the output from Program 1.8.

Output 1.8 Output from Program 1.8 that uses SAS language functions

Report on Data Set books.ytdsales
Num Obs: 6096 Date Created: 01JAN2007

The MEANS Procedure

Analysis Variable : saleprice Sale Price

Section	N Obs	Sum
Certification and Training	726	31648.52
Internet	1456	62295.78
Networks and Telecommunication	717	30803.81
Operating Systems	922	39779.11
Programming and Applications	1429	62029.41
Web Design	846	37121.52

Example 1.9: Building and Saving a Library of Utility Routines

In your work, you might frequently need to program the same process in different applications. Rather than rewriting the code every time, you might be able to write the code once and save it in a macro program. Later when you want to execute that code again, you just reference the macro program. With the macro facility, there are ways to save the code in special libraries and to even save the compiled code in permanent locations. For example, perhaps certain reports all require the same SAS options, titles, and footnotes. You can save these standardizations in a macro program and call the macro program rather than write the statements every time.

You can store the macro program in a special location called an autocall library. Then when you want to submit the macro program for compilation and execution during a later SAS session, you just need to tell SAS to look for macro programs in the autocall library, and you do not have to explicitly submit the code in the SAS session.

The macro program STANDARDOPTS in Program 1.9 submits an OPTIONS statement to ensure that three SAS options are in effect: NODATE, NUMBER, and BYLINE. It also specifies TITLE1 and FOOTNOTE1 statements. Assume STANDARDOPTS is stored in a file named STANDARDOPTS.SAS in Windows directory c:\mymacroprograms. (Note that if you were using UNIX, the filename must be in lowercase.)

Program 1.9

```
%macro standardopts;
  options nodate number byline;
  title "Bookstore Report";
  footnote1 "Prepared &sysday &sysdate9 at &systime using SAS
&sysver";
%mend standardopts;
```

In a later SAS session, you do not need to submit the previous code. Instead, you can submit the following OPTIONS statement and call the macro program STANDARDOPTS. The SASAUTOS option specifies that SAS have access to the autocall library shipped with SAS ("sasautos") and to the autocall library in the mymacroprograms folder.

```
options mautosource sasautos=(sasautos,'c:\mymacroprograms');
%standardopts
```

After submitting the macro program STANDARDOPTS, the title text on subsequent reports is

```
Bookstore Report
```

If STANDARDOPTS was submitted on February 22, 2008, from a SAS session that started at 8:36 using SAS 9.1, the footnote text on subsequent reports would be:

```
Prepared Friday 22FEB2008 08:36 using SAS 9.1
```