# CHAPTER 4 Macro Processing

## Introduction

This chapter describes macro processing and shows the typical pattern that the SAS System follows to process a program containing macro elements. For most macro programming, you do not need this level of detail. It is provided to help you understand what is going on behind the scenes.

## Defining and Calling Macros

*Macros* are compiled programs that you can invoke (or *call*) in a submitted SAS program or from a SAS command prompt. Like macro variables, you generally use macros to generate text. However, macros provide additional capabilities:

- Macros can contain programming statements that enable you to control how and when text is generated.
- Macros can accept parameters. This allows you to write generic macros that can serve a number of uses.

To compile a macro, you must submit a macro definition. The general form of a macro definition is

%MACRO *macro-name*;
    <*macro_text*>

%MEND <*macro_name*>;

where *macro_name* is a unique SAS name that identifies the macro and *macro_text* is any combination of macro statements, macro calls, text expressions, or constant text.

When you submit a macro definition, the macro processor compiles the definition and produces a member in the session catalog. The member consists of compiled macro program statements and text. The distinction between compiled items and noncompiled (text) items is important for macro execution. Examples of text items include:

- macro variable references
- nested macro calls

- □ macro functions, except %STR and %NRSTR
- □ arithmetic and logical macro expressions
- □ text to be written by %PUT statements
- □ field definitions in %WINDOW statements
- □ model text for SAS statements and SAS Display Manager System commands.

When you want to call the macro, you use the form
%*macro_name*
A later section illustrates calling a macro. The following section illustrates how the macro processor compiles and stores a macro definition.

## How the Macro Processor Compiles a Macro Definition

When you submit a SAS program, the contents of the program goes to an area of memory called the input stack. The example program in Figure 4.1 on page 30 contains a macro definition, a macro call and a PROC PRINT step. This section illustrates how the macro definition in the example program is compiled and stored.

**Figure 4.1** Macro APP in the Input Stack

```
                    Input Stack
%macro app(goal);
   %if &sysday=Friday %then
      %do;
         data thisweek;
           set lastweek;
           if totsales > &goal
             then bonus = .03;
           else bonus = 0;
      %end;
%mend app;
%app(10000)
proc print;
run;
```

Using the same process described in Chapter 2, "SAS Programs and Macro Processing," the word scanner begins tokenizing the program. When the word scanner detects % followed by a non-blank character in the first token, it triggers the macro processor. The macro processor examines the token and recognizes the beginning of a macro definition. The macro processor pulls tokens from the input stack and compiles until the %MEND statement terminates the macro definition (Figure 4.2 on page 31).

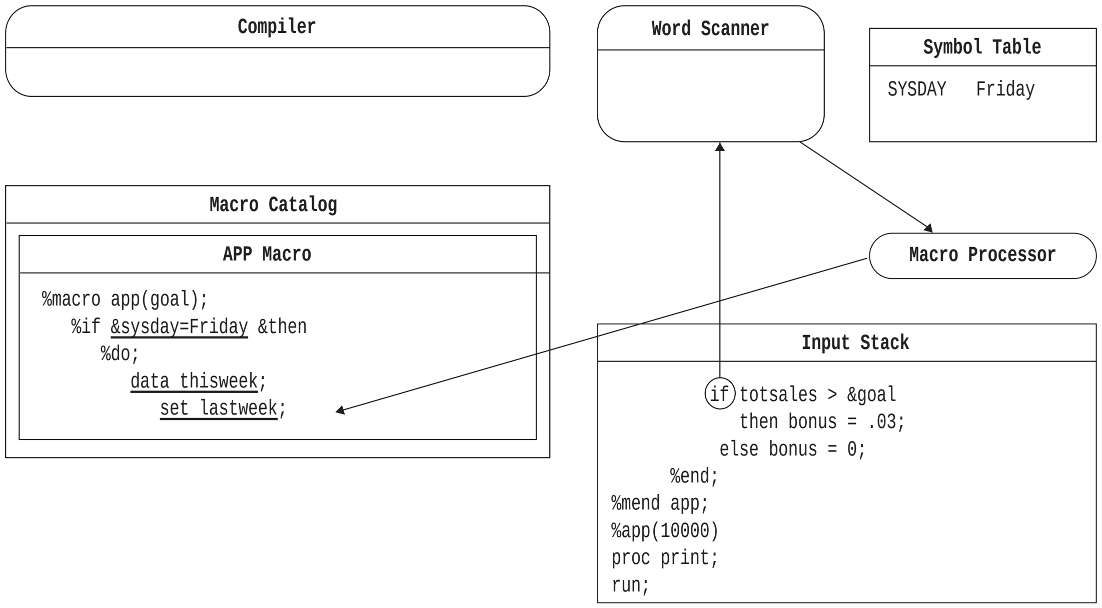
During macro compilation, the macro processor

- □ creates an entry in the session catalog.
- □ compiles and stores all macro program statements for that macro as macro instructions.
- □ stores all noncompiled items in the macro as text.

*Note:* Text items are underlined in the illustrations in this chapter. △

If the macro processor detects a syntax error while compiling the macro, the macro processor checks the syntax in the rest of the macro and issues messages for any additional errors it finds. However, the macro processor does not store the macro for execution. A macro that the macro processor compiles but does not store is called a *dummy macro*.

**Figure 4.2** Macro APP in the Input Stack

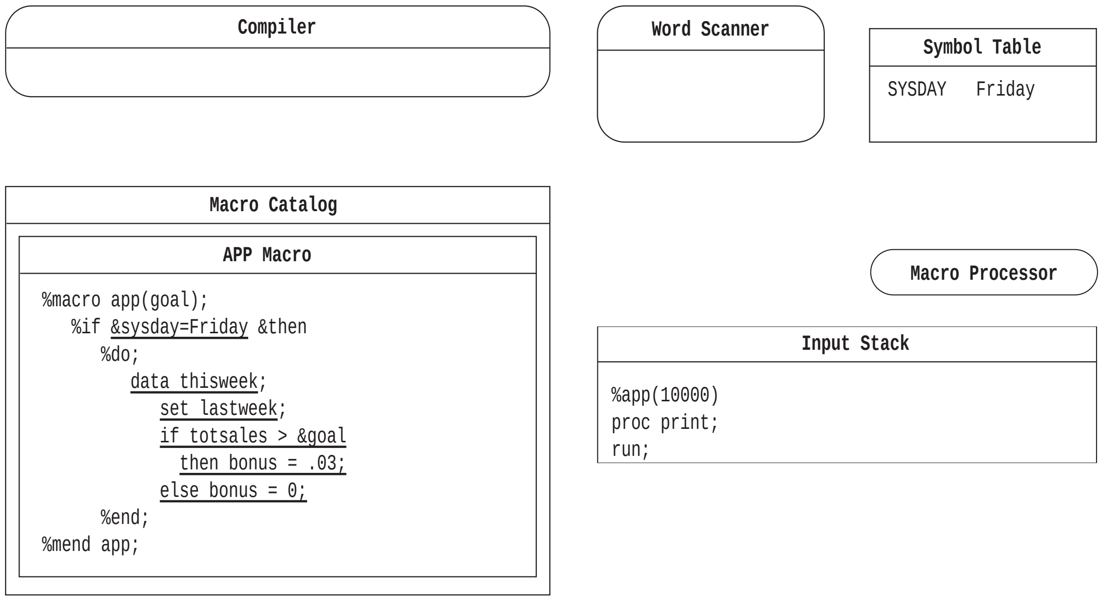


```
%macro app(goal);
   %if &sysday=Friday &then
      %do;
         data thisweek;
            set lastweek;
```

```
          if totsales > &goal
             then bonus = .03;
           else bonus = 0;
      %end;
%mend app;
%app(10000)
proc print;
run;
```

In this example, the macro definition is compiled and stored sucessfully (Figure 4.3). For the sake of illustration, the compiled APP macro looks like the original macro definition that was in the input stack. The entry would actually contain compiled macro instructions with constant text. The constant text in this example is underlined.

**Figure 4.3** The Compiled Macro APP



```
%macro app(goal);
   %if &sysday=Friday &then
      %do;
         data thisweek;
            set lastweek;
            if totsales > &goal
              then bonus = .03;
            else bonus = 0;
      %end;
%mend app;
```

```
%app(10000)
proc print;
run;
```

## How the Macro Processor Executes a Compiled Macro

Macro execution begins with the macro processor opening the SASMACR catalog to read the appropriate macro entry. As the macro processor executes the compiled instructions in the macro entry, it performs a series of simple repetitive actions. During macro execution, the macro processor

- executes compiled macro program instructions
- places noncompiled constant text on the input stack
- waits for the word scanner to process the generated text
- resumes executing compiled macro program instructions.

To continue the example from the previous section, Figure 4.4 on page 32 shows the lines remaining in the input stack after the macro processor compiles the macro definition APP.

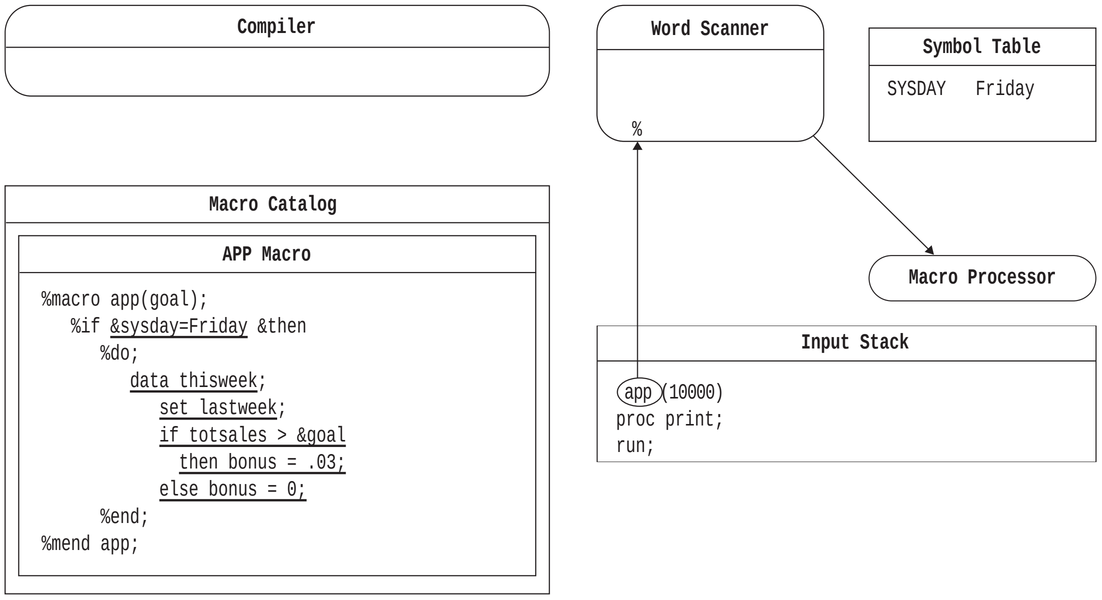
**Figure 4.4** The Macro Call in the Input Stack

**Input Stack**

```
%app(10000)
proc print;
run;
```

The word scanner examines the input stack and detects % followed by a nonblank character in the first token. It triggers the macro processor to examine the token (Figure 4.5 on page 32).

**Figure 4.5** Macro Call Entering Word Queue



```
%macro app(goal);
   %if &sysday=Friday &then
      %do;
         data thisweek;
            set lastweek;
            if totsales > &goal
              then bonus = .03;
            else bonus = 0;
      %end;
%mend app;
```

```
app(10000)
proc print;
run;
```

The macro processor recognizes a macro call and begins to execute macro APP, as follows:

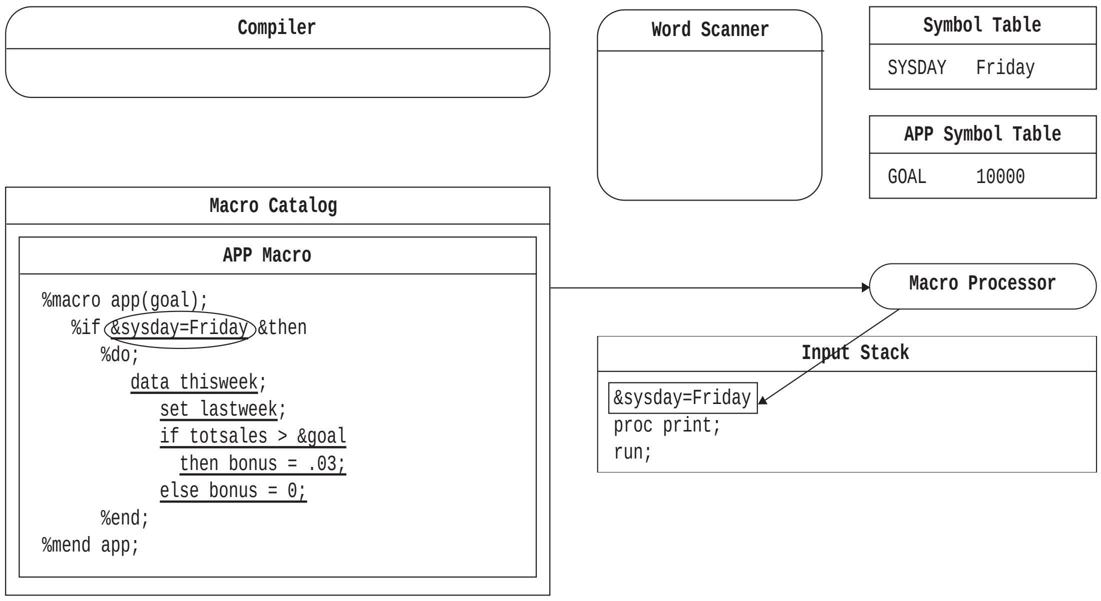
**1** The macro processor opens the session catalog and creates a local symbol table with an entry for the parameter GOAL.

**2** The macro processor removes the tokens for the macro call from the input stack and places the parameter value in GOAL entry in the APP symbol table.

**3** The macro processor encounters the compiled %IF instruction and recognizes that the next item will be text containing a condition.

**4** The macro processor places the text **`&sysday=Friday`** on the input stack ahead of the remaining text in the program (Figure 4.6 on page 33) and waits for the word scanner to tokenize the generated text.

**Figure 4.6** Text for %IF Condition on Input Stack



```
%macro app(goal);
   %if &sysday=Friday &then
      %do;
         data thisweek;
            set lastweek;
            if totsales > &goal
              then bonus = .03;
            else bonus = 0;
      %end;
%mend app;
```

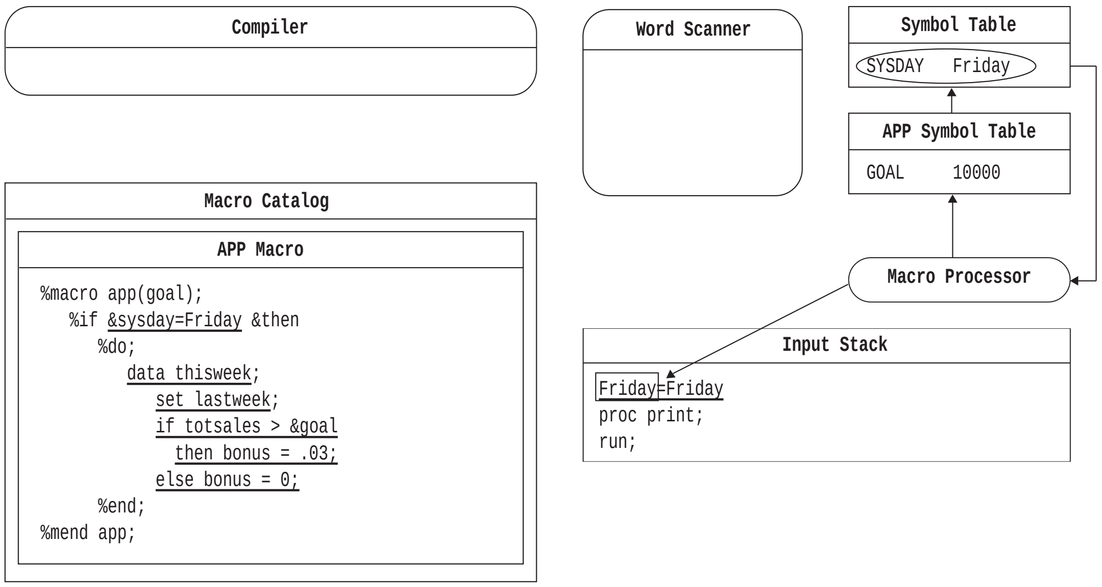
```
&sysday=Friday
proc print;
run;
```

**5** The word scanner starts tokenizing the generated text, recognizes an ampersand followed by nonblank character in the first token, and triggers the macro processor.

**6** The macro processor examines the token, finds a possible macro variable reference &SYSDAY. The macro processor first searches the local APP symbol table for a matching entry and then the global symbol table. when the macro processor finds the entry in the global symbol table, it replaces macro variable in the input stack with the value **`Friday`** (Figure 4.7).

**7** The macro processor stops and waits for the word scanner to tokenize the generated text.

**Figure 4.7** Input Stack after Macro Variable Reference Is Resolved

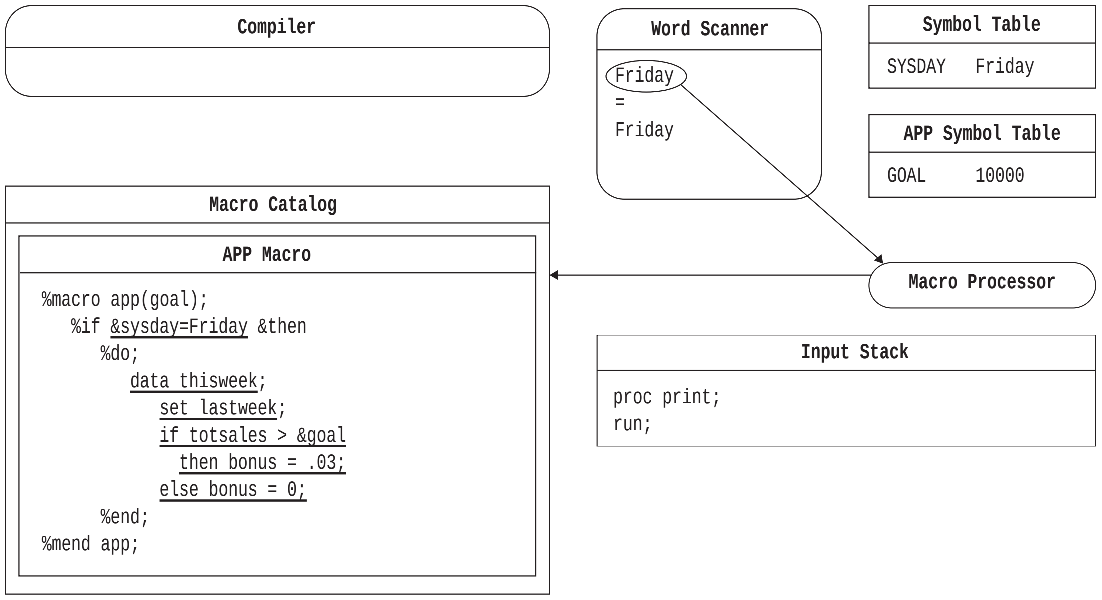


```
%macro app(goal);
   %if &sysday=Friday &then
      %do;
         data thisweek;
            set lastweek;
            if totsales > &goal
              then bonus = .03;
            else bonus = 0;
      %end;
%mend app;
```

8 The word scanner then read **`Friday=Friday`** from the input stack.

9 The macro processor evaluates the expression **`Friday=Friday`** and, because the expression is true, proceeds to the %THEN and %DO instructions (Figure 4.8).

**Figure 4.8** Macro Processor Receives the Condition



```
%macro app(goal);
   %if &sysday=Friday &then
      %do;
         data thisweek;
            set lastweek;
            if totsales > &goal
              then bonus = .03;
            else bonus = 0;
      %end;
%mend app;
```

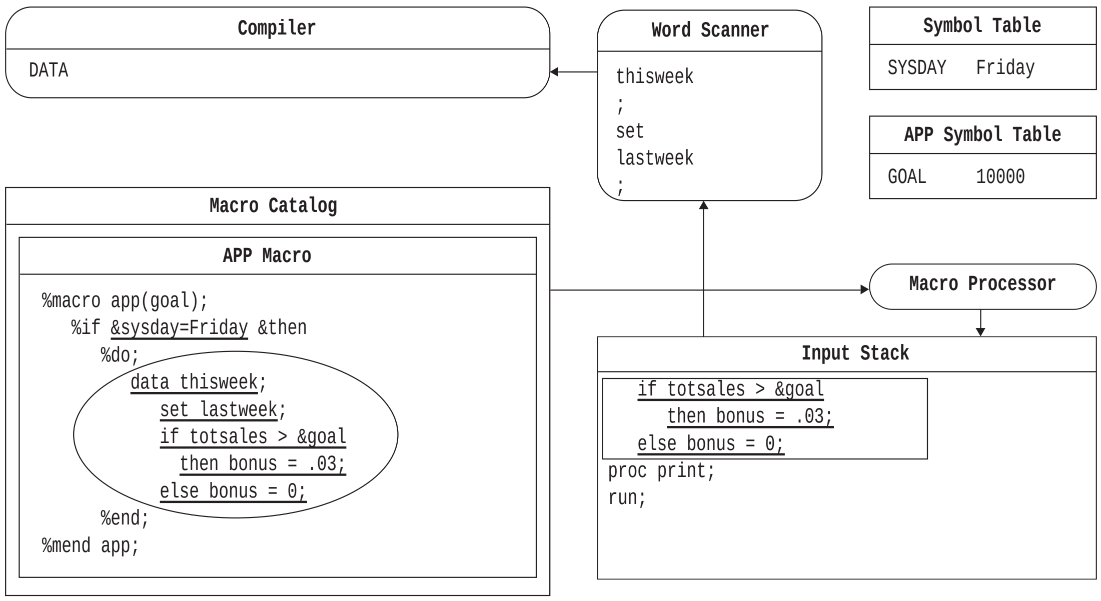
10 The macro processor executes the compiled %DO instructin and recognizes that the next item is text.

11 The macro processor places the text on top of the input stack and waits for the word scanner to begin tokenization.

12 The word scanner reads the generated text from the input stack, and tokenizes it.

**13** The word scanner recognizes the beginning of a DATA step, and triggers the compiler to begin accepting tokens. The word scanner transfers tokens to the compiler from the top of the stack (Figure 4.9).

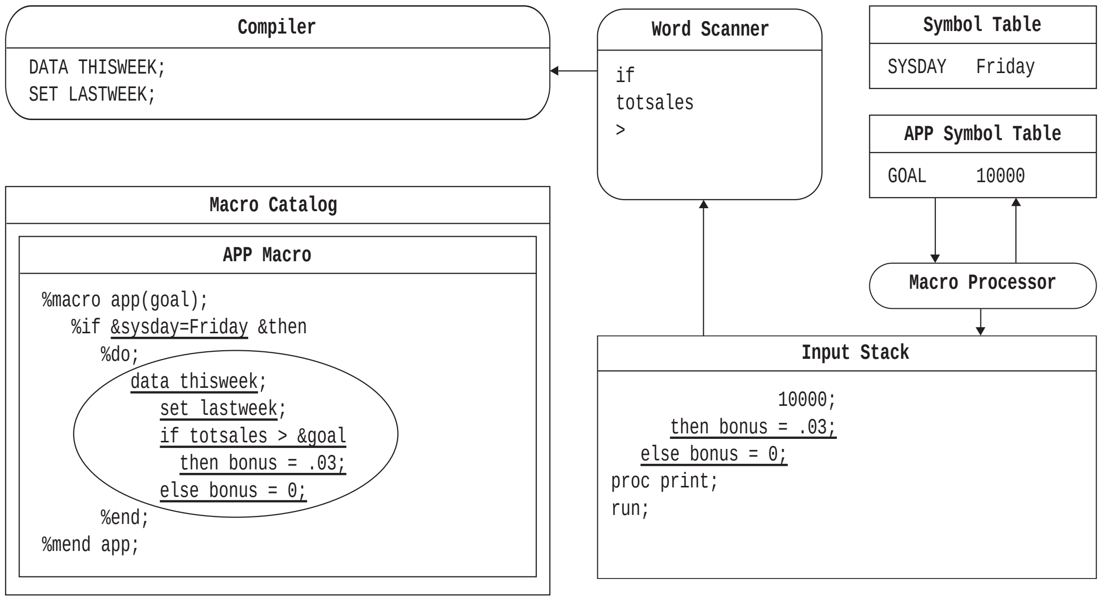
**Figure 4.9** Generated Text on Top of Input Stack



**14** When the word scanner detects & followed by a nonblank characater (the macro variable refernece &GOAL), it triggers the macro processor.

**15** The macro processor looks in the local APP symbol table and resolves the macro variable reference &GOAL to **10000**. the macro processor places the value on top of the input stack, ahead of the remaining text in the program (Figure 4.10).

**Figure 4.10** The Word Scanner Reads Generated Text



**16** The word scanner resumes tokenization. When it has completed tokenizing the generated text, it triggers the macro processor.

**17** The macro processor resumes processing the compiled macro instructions. It recognizes the end of the %DO group at the %END instructin and proceeds to %MEND.

**18** the macro processor executes the %MEND instruction, removes the local symbol table APP, and macro APP ceases execution.

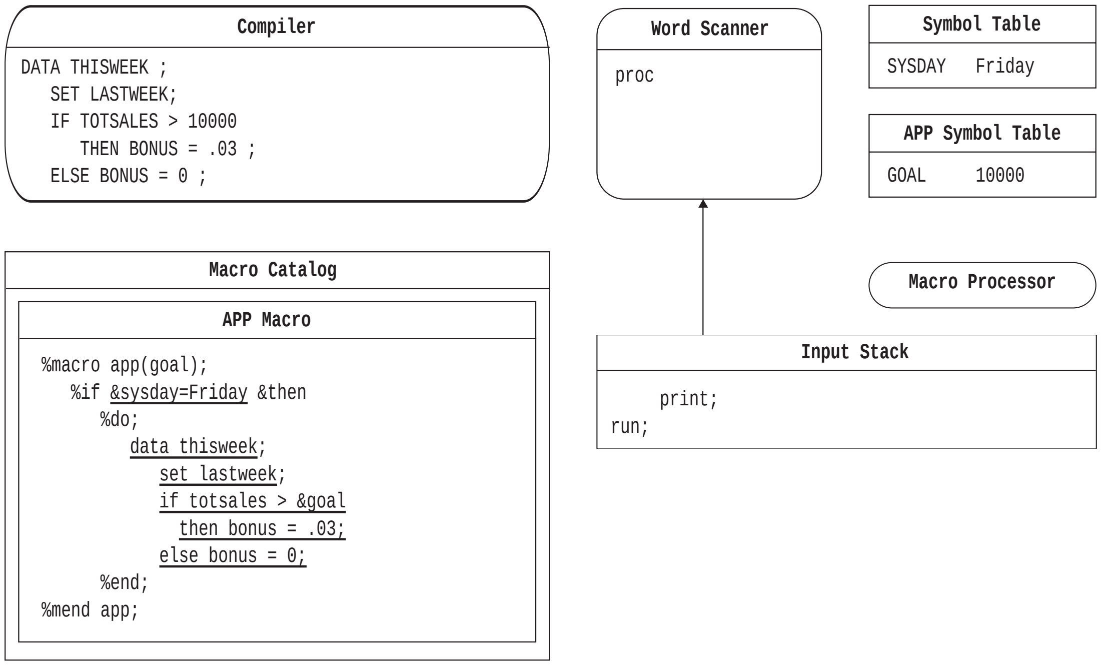
**19** The macro processor triggers the word scanner to resume tokenization.

**20** The word scanner reads the first token in input stack (PROC), recognizes the beginning of a step boundary, and triggers the DATA step compiler.

**21** The compiled DATA step is executed, and the DATA step compiler is cleared.

**22** The word scanner signals the PRINT procedure (a separate executable not illustrated), which pulls the remaining tokens.

**Figure 4.11** The Remaining Statements are Compiled and Executed



```
DATA THISWEEK ;
   SET LASTWEEK;
   IF TOTSALES > 10000
      THEN BONUS = .03 ;
   ELSE BONUS = 0 ;
```

```
%macro app(goal);
   %if &sysday=Friday &then
      %do;
         data thisweek;
            set lastweek;
            if totsales > &goal
              then bonus = .03;
            else bonus = 0;
      %end;
%mend app;
```

```
     print;
run;
```

## Summary

The previous sections illustrate the relationship between macro compilation and execution and DATA step compilation and execution. The relationship contains a pattern of simple repetitive actions. These actions begin when text is submitted to the input stack and the word scanner begins tokenization. At times the word scanner waits for the macro processor to perform an activity, such as searching the symbol tables or compiling a macro definition. If the macro processor generates text during its activity, then it pauses while the word scanner tokenizes the text and sends the tokens to the appropriate target. These tokens may trigger other actions in parts of the SAS system, such as the DATA step compiler, the command processor, or a SAS procedure. If this is the case, the macro processor waits for these actions to be completed before resuming its activity. When the macro processor stops, the word scanner resumes tokenization. This process continues until the entire program has been processed.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS Macro Language: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999. 310 pages.

**SAS Macro Language: Reference, Version 8**

1-58025-522-1

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, October 1999