Whenever the parser reduces a rule, it executes user C (or C++) code associated with the rule, known as the rule's *action*. The action appears in the braces after the end of the rule, before the semicolon or vertical bar. The action code can refer to the values of the right-hand side symbols as **$1**, **$2**, …, and can set the value of the left-hand side by setting **$$**.

Example:

```
%token NAME NUMBER
%%
statement:  NAME '=' expression
     |      expression              {cout << ($1) << endl;}
     ;
expression: expression '+' NUMBER {$$=$1+$3;}
     |      expression '-' NUMBER  {$$=$1-$3;}
     |      NUMBER                 {$$=$1;}
     ;
```

```
1.  int TokenMgr(int t)
2.  {   int tc=t;
3.      if (t==ID) {
4.          char s[1024];
5.          for (int a=0;a<strlen(yytext)&&a<1024;a++) s[a]=tolower(yytext[a]);
6.          s[strlen(yytext)]=0;
7.          yylval.token=new string(s);
8.          tc=RW[s];
9.          if (tc==0) tc=t;
10.     } else {
11.         yylval.token=new string(yytext);
12.     }
13.     PrintToken(tfs,tc,line,col);
14.     col+=yyleng;
15. return tc;
16. }
```

Figure 1. Function *TokenMgr*

Notes:

1. Function *TokenMgr* is a function located in file **paslex.l**. Function *TokenMgr* is called by the scanner for every token (except comments, and white space).
2. Note lines 7 and 11. Member token of **struct** *yylval* is assigned a pointer to all tokens recognized by the scanner.

```
%{
//-------------------------------------------------------------------
//C++ inlcude files
//-------------------------------------------------------------------
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
using namespace std;
//-------------------------------------------------------------------
//Supporting utilities
//-------------------------------------------------------------------
#include "SList.h"
//-------------------------------------------------------------------
//Application inlcude files
//-------------------------------------------------------------------
#include "paslex.h"
#include "paspar.h"
//-------------------------------------------------------------------
//Functions
//-------------------------------------------------------------------
void yyerror(char* m);
//-------------------------------------------------------------------
//Externals
//-------------------------------------------------------------------
extern ofstream tfs;
extern int line;
extern int col;
//-------------------------------------------------------------------
%}
%union {
  string* token;
  SList* slist;
}
```

Figure 2. File **paspar.y**

Code fragment	Discussion
#include "List.h"	Defines a list template that can be implemented as a list of strings that are used to store the strings of an identifier_list.
extern int line; **extern int col;**	The line and column are defined, stored, and computed in file **paslex.l**. These declarations make line and col available to all functions in file **paspar.y**.
%union { **string* token;** **List<string>* strlist;** **}**	This definition, given in the syntax of yacc, defines semantic variable types token and slist. By associating these types with terminal and non-terminal grammar symbols, the compiler designer can extract and create semantic information.

```
%token <token> TOKEN_BEGIN
%token <token> PLUS
…
%token <token> REALIT
%token <token> CHRLIT
%token <token> TOKEN_END

%type  <strlist> identifier_list
```

Figure 2. File **paspar.y** (continued)

Code fragment	Discussion
%token <token> PLUS	Associates type “token” a pointer to a string with the terminal symbol PLUS, +.
%type <strlist> identifier_list	Associates type “strlist”, a pointer to a list of strings with the nonterminal symbol identifier_list.

```
%%
program:
  PROGRAM ID program_parameters SEMICOLON
  declarations
  subprogram_declarations
  compound_statement
  PERIOD
  {tfs << endl << "program -> "
            << "PROGRAM ID(" << (*$2) << ") program_parameters ; "
            << "declarations subprogram_declarations compound_statement .";
  }
program_parameters:
  {tfs << endl << "program_parameters -> empty";}
program_parameters:
  LPAREN program_parameter_list RPAREN
  {tfs << endl << "program_parameters -> (program_parameter_list)";}
program_parameter_list:
  identifier_list
  {tfs << endl << "program_parameter_list -> identifier_list" << (*$1) ;}
identifier_list:
  ID
  {tfs << endl << "identifier_list -> ID(" << (*$1) << ")";
   $$=new SList;
   $$->Insert(*$1);
  }
identifier_list:
  identifier_list COMMA ID
  {tfs << endl << "identifier_list -> " << (*$1) << " , ID(" << (*$3) << ")";
   $1->Insert(*$3);
   $$=$1;
  }
…
%%
```

Figure 2. File **paspar.y** (continued)

Code fragment	**Discussion**

program_parameter_list:
  identifier_list
  {tfs << endl << "program_parameter_list ->
identifier_list" << (*$1) ;}
	Print the identifiers in the program_parameter_list. $1 is a yacc pseudo variable having type “slist”, pointer to a list of strings. The (*$1) invokes the operator overloaded function that print the entire list in **class** *SList*.

Code fragment	Discussion
identifier_list: ID {tfs << endl << "identifier_list -> ID(" << (*$1) << ")"; $$=new SList; $$->Insert(*$1); }	Print the ID since it is a pointer to a string. It is the new string created in file paslex.l Create an empty list of strings. Associate the empty list of strings with nonterminal identifier_list. Insert the first string in the list.
identifier_list: identifier_list COMMA ID {tfs << endl << "identifier_list -> " << (*$1) << " , ID(" << (*$3) << ")"; $1->Insert(*$3); $$=$1; }	Print the ID since it is a pointer to a string. It is the new string created in file paslex.l Insert the second and subsequent strings in the list. Copy the pointer from the identifier list on the right hand side to the left hand side.

```
//---------------------------------------------------------------------
//User function section
//---------------------------------------------------------------------
void yyerror(char* m)
{   cout << endl
        << "line(" << line << ") col(" << col << ") " << m;
   cout << endl;
}
```

Figure 2. File **paspar.y** (continued)

Code fragment	Discussion
void yyerror(char* m) { cout << endl << "line(" << line << ") col(" << col << ") " << m; cout << endl; }	Function *yyerror* is called by the parser, yyparse, when the input is not a sentence in the grammar, a syntax error. Note that global variables *line* and *col* are employed to print the last known location.