COMPILER DESIGN – ASSIGNMENT – 1

NAME : ROHIT R YAKKUNDI

SRN : PES2UG23CS819

SECTION : G

Lexer.l:

```
%{
#include "parser.tab.h"
#include "symbol_table.h"
extern int yylineno;
extern SymbolTable *symbol_table;
int column = 1;
%}

%option yylineno
%option nounput
%option noinput

DIGIT       [0-9]
ID          [a-zA-Z_][a-zA-Z0-9_]*
NUMBER      {DIGIT}+
REAL        {DIGIT}+"."{DIGIT}+([Ee][+-]?{DIGIT}+)?
WHITESPACE  [ \t\r\f\v]+

%%

"#include"              { column += yyleng; return T_INCLUDE; }
"<"[^>]*">"           { column += yyleng; return T_HEADER; }

"int"                   { column += yyleng; return T_INT; }
"char"                  { column += yyleng; return T_CHAR; }
"float"                 { column += yyleng; return T_FLOAT; }
"double"                { column += yyleng; return T_DOUBLE; }
"void"                  { column += yyleng; return T_VOID; }

"if"                    { column += yyleng; return T_IF; }
"else"                  { column += yyleng; return T_ELSE; }
"while"                 { column += yyleng; return T_WHILE; }
"for"                   { column += yyleng; return T_FOR; }
"do"                    { column += yyleng; return T_DO; }
"break"                 { column += yyleng; return T_BREAK; }
"continue"              { column += yyleng; return T_CONTINUE; }
"return"                { column += yyleng; return T_RETURN; }

"++"                    { column += yyleng; return T_INC; }
"--"                    { column += yyleng; return T_DEC; }
"+"                     { column += yyleng; return '+'; }
"-"                     { column += yyleng; return '-'; }
```

```
43  "*"                     { column += yyleng; return '*'; }
44  "/"                     { column += yyleng; return '/'; }
45  "%"                     { column += yyleng; return '%'; }
46
47  "=="                    { column += yyleng; return T_EQ; }
48  "!="                    { column += yyleng; return T_NE; }
49  "<="                    { column += yyleng; return T_LE; }
50  ">="                    { column += yyleng; return T_GE; }
51  "<"                     { column += yyleng; return '<'; }
52  ">"                     { column += yyleng; return '>'; }
53  "="                     { column += yyleng; return '='; }
54
55  "&&"                    { column += yyleng; return T_AND; }
56  "||"                    { column += yyleng; return T_OR; }
57  "!"                     { column += yyleng; return '!'; }
58
59  "{"                     { column += yyleng; return '{'; }
60  "}"                     { column += yyleng; return '}'; }
61  "("                     { column += yyleng; return '('; }
62  ")"                     { column += yyleng; return ')'; }
63  "["                     { column += yyleng; return '['; }
64  "]"                     { column += yyleng; return ']'; }
65  ";"                     { column += yyleng; return ';'; }
66  ","                     { column += yyleng; return ','; }
67
68  {ID}        {
69      if (yyleng > MAX_NAME) {
70          printf("Warning: Identifier '%s' at line %d truncated to %d characters\n",
71                 yytext, yylineno, MAX_NAME);
72          yytext[MAX_NAME] = '\0';
73      }
74      yylval.string = strdup(yytext);
75      column += yyleng;
76      return T_IDENTIFIER;
77  }
78
79  {NUMBER}    {
80      yylval.number = atoi(yytext);
81      column += yyleng;
82      return T_NUMBER;
83  }
```

```
{REAL}      {
    yylval.real = atof(yytext);
    column += yyleng;
    return T_REAL;
}

{WHITESPACE}    { column += yyleng; }
\n              { column = 1; }

"//".*          { /* Skip single-line comments */ }
"/*"([^*]|\*+[^*/])*\*+"/"    {
    /* Skip multi-line comments */
    for(int i = 0; i < yyleng; i++) {
        if(yytext[i] == '\n') column = 1;
        else column++;
    }
}

.               {
    printf("Lexical Error: Unexpected character '%s' at line %d, column %d\n",
           yytext, yylineno, column);
    column += yyleng;
}

%%

int yywrap() {
    return 1;
}
```

Parser.y:

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "symbol_table.h"

extern int yylex();
extern int yylineno;
extern char* yytext;
void yyerror(const char* s);

SymbolTable *symbol_table;
%}

%union {
    int number;
    double real;
    char* string;
}

%token T_INCLUDE T_HEADER
%token T_INT T_CHAR T_FLOAT T_DOUBLE T_VOID
%token T_IF T_ELSE T_WHILE T_FOR T_DO
%token T_BREAK T_CONTINUE T_RETURN
%token T_INC T_DEC T_EQ T_NE T_LE T_GE T_AND T_OR
%token <string> T_IDENTIFIER
%token <number> T_NUMBER
%token <real> T_REAL

%start program

%%

program
    : includes declarations
    | declarations
    ;

includes
    : T_INCLUDE T_HEADER
    | includes T_INCLUDE T_HEADER
    ;
```

```
declarations
    : declaration
    | declarations declaration
    ;

declaration
    : function_declaration
    | var_declaration
    ;

function_declaration
    : type_specifier T_IDENTIFIER '(' parameter_list ')' compound_statement
    | type_specifier T_IDENTIFIER '(' ')' compound_statement
    ;

parameter_list
    : parameter_declaration
    | parameter_list ',' parameter_declaration
    ;

parameter_declaration
    : type_specifier T_IDENTIFIER
    | type_specifier T_IDENTIFIER '[' ']'
    | type_specifier T_IDENTIFIER '[' T_NUMBER ']'
    ;

var_declaration
    : type_specifier init_declarator_list ';'
    ;

init_declarator_list
    : init_declarator
    | init_declarator_list ',' init_declarator
    ;

init_declarator
    : T_IDENTIFIER                          { insert_symbol(symbol_table, $1, "var", yylineno); }
    | T_IDENTIFIER '=' expression          { insert_symbol(symbol_table, $1, "var", yylineno); }
    | array_declarator
    | array_declarator '=' array_initializer
    ;
```

```
array_declarator
    : T_IDENTIFIER '[' T_NUMBER ']'         {
        Symbol sym;
        sym.is_array = 1;
        sym.num_dimensions = 1;
        sym.dimensions[0] = $3;
        insert_symbol(symbol_table, $1, "array", yylineno);
    }
    | T_IDENTIFIER '[' T_NUMBER ']' '[' T_NUMBER ']' {
        Symbol sym;
        sym.is_array = 1;
        sym.num_dimensions = 2;
        sym.dimensions[0] = $3;
        sym.dimensions[1] = $6;
        insert_symbol(symbol_table, $1, "array", yylineno);
    }
    | T_IDENTIFIER '[' T_NUMBER ']' '[' T_NUMBER ']' '[' T_NUMBER ']' {
        Symbol sym;
        sym.is_array = 1;
        sym.num_dimensions = 3;
        sym.dimensions[0] = $3;
        sym.dimensions[1] = $6;
        sym.dimensions[2] = $9;
        insert_symbol(symbol_table, $1, "array", yylineno);
    }
    ;

array_initializer
    : '{' expression_list '}'
    ;

expression_list
    : expression
    | expression_list ',' expression
    ;
```

```
type_specifier
    : T_INT
    | T_CHAR
    | T_FLOAT
    | T_DOUBLE
    | T_VOID
    ;

statement
    : compound_statement
    | expression_statement
    | selection_statement
    | iteration_statement
    | jump_statement
    ;

compound_statement
    : '{' '}'                           { enter_scope(symbol_table); exit_scope(symbol_table); }
    | '{' statement_list '}'            { enter_scope(symbol_table); exit_scope(symbol_table); }
    | '{' declaration_list '}'          { enter_scope(symbol_table); exit_scope(symbol_table); }
    | '{' declaration_list statement_list '}' { enter_scope(symbol_table); exit_scope(symbol_table); }
    ;

declaration_list
    : var_declaration
    | declaration_list var_declaration
    ;

statement_list
    : statement
    | statement_list statement
    ;

expression_statement
    : ';'
    | expression ';'
    ;

selection_statement
    : T_IF '(' expression ')' statement
    | T_IF '(' expression ')' statement T_ELSE statement
    ;
```

```
iteration_statement
    : T_WHILE '(' expression ')' statement
    | T_DO statement T_WHILE '(' expression ')' ';'
    | T_FOR '(' for_init ';' for_cond ';' for_incr ')' statement
    ;

for_init
    : /* empty */
    | expression
    | type_specifier init_declarator_list
    ;

for_cond
    : /* empty */
    | expression
    ;

for_incr
    : /* empty */
    | expression
    ;

jump_statement
    : T_BREAK ';'
    | T_CONTINUE ';'
    | T_RETURN ';'
    | T_RETURN expression ';'
    ;

expression
    : assignment_expression
    ;

assignment_expression
    : conditional_expression
    | unary_expression '=' assignment_expression
    | T_IDENTIFIER '[' expression ']' '=' assignment_expression
    ;
```

```
conditional_expression
    : logical_or_expression
    ;

logical_or_expression
    : logical_and_expression
    | logical_or_expression T_OR logical_and_expression
    ;

logical_and_expression
    : equality_expression
    | logical_and_expression T_AND equality_expression
    ;

equality_expression
    : relational_expression
    | equality_expression T_EQ relational_expression
    | equality_expression T_NE relational_expression
    ;

relational_expression
    : additive_expression
    | relational_expression '<' additive_expression
    | relational_expression '>' additive_expression
    | relational_expression T_LE additive_expression
    | relational_expression T_GE additive_expression
    ;

additive_expression
    : multiplicative_expression
    | additive_expression '+' multiplicative_expression
    | additive_expression '-' multiplicative_expression
    ;

multiplicative_expression
    : unary_expression
    | multiplicative_expression '*' unary_expression
    | multiplicative_expression '/' unary_expression
    | multiplicative_expression '%' unary_expression
    ;
```

```
unary_expression
    : postfix_expression
    | T_INC unary_expression
    | T_DEC unary_expression
    | '+' unary_expression
    | '-' unary_expression
    | '!' unary_expression
    ;

postfix_expression
    : primary_expression
    | postfix_expression '[' expression ']'
    | postfix_expression T_INC
    | postfix_expression T_DEC
    ;

primary_expression
    : T_IDENTIFIER                          { add_line_use(symbol_table, $1, yylineno); }
    | T_NUMBER
    | T_REAL
    | '(' expression ')'
    ;

%%

void yyerror(const char* s) {
    fprintf(stderr, "Error at line %d: %s\n", yylineno, s);
}
```

OUTPUTS:

Test cases:

Assignment-1_nested_do_while_valid.c

```c
int main()
{
    int a,b=6; // initialization within declaration
    a = 5 + 3;
    do
    {
        a = 5;
        do
        {
            int k = 123 + 456 * 123;
            a = a + b;
        }
        while(a < b);
    }
    while (a<1); // do while loop

}
```

Output:

```
D:\SEM-6 NOTES\Compiler Design\Assignment\PES2UG23CS819 CD ASSIGNMENT -1\COMPLETED_A1\PES2UG23CS819-ASSIGNMENT1>parser.e
xe assignment-1_nested_do_while_valid.c
Starting parse...
Parsing completed successfully.

Symbol Table:
Name                Type        Scope     Line      Array     Uses
--------------------------------------------------------------------
a                   var         0         5         No        6 9 13 13 15 17
b                   var         0         5         No        13 15
k                   var         0         12        No
```

assignment-1_simple_for_valid.c:

```c
int main()
{
    int a, b = 6;
    char c[5]; // array declaration
    a = 5 + 3;
    b = ++a;                    // unary op
    for (i = 0; i < 5; ++i) // for loop
    {
        for (j = 10; j > 100; ++j)
        {
            int k = a + b;
            b = b + 100;
        }
    }
}
```

Output:

```
D:\SEM-6 NOTES\Compiler Design\Assignment\PES2UG23CS819 CD ASSIGNMENT -1\COMPLETED_A1\PES2UG23CS819-ASSIGNMENT1>parser.e
xe assignment-1_simple_for_valid.c
Starting parse...
Parsing completed successfully.

Symbol Table:
Name                Type        Scope     Line      Array     Uses
--------------------------------------------------------------------
a                   var         0         5         No        7 8
b                   var         0         5         No        8
c                   array       0         6         No
```

assignment-1_simple_do_while_valid.c:

```c
#include <stdio.h>

int main()
{
    int a,b=6; // initialization within declaration
    int x[2][3][4];
    int arr[5] = { 10, 20, 30, 40, 50 };
    a = 5 + 3;
    do {a = 5;} while (a<1); // do while loop

}
```

Output:

```
D:\SEM-6 NOTES\Compiler Design\Assignment\PES2UG23CS819 CD ASSIGNMENT -1\COMPLETED_A1\PES2UG23CS819-ASSIGNMENT1>parser.e
xe assignment-1_simple_do_while_valid.c
Starting parse...
Parsing completed successfully.

Symbol Table:
Name                Type        Scope    Line     Array    Uses
--------------------------------------------------------
a                   var         0        5        No       8 9 9
b                   var         0        5        No
x                   array       0        6        No
arr                 array       0        7        No
```

assignment-1_nested_for_invalid.c:

```c
int main()
{
    int a, b = 6;
    char c[5]; // array declaration
    a = 5 + 3;
    b = ++a;                    // unary op
    for (i = 0; i < 5; ++i) // for loop
    {
        for (j = 10; j > 100; ++j)
        {
            int k = a + b;
            b = b + 100;
        }
    }
}
```

Output:

```
D:\SEM-6 NOTES\Compiler Design\Assignment\PES2UG23CS819 CD ASSIGNMENT -1\COMPLETED_A1\PES2UG23CS819-ASSIGNMENT1>parser.e
xe assignment-1_nested_for_valid.c
Starting parse...
Error at line 10: syntax error
Parsing failed.
```

assign-1_test-1_invalid.c:

```c
int main()
{
    int a, b;
    a = 10;
    b = 11;
    a = 8997 / 5816 / 4258                                  //semi-colon missing

    if(a <= b)
    {
        int a4;
        a4 = 2416 + 4649;
    }
    else(a > b)                                             //no condition allowed for else block
    {
        int a3;
        a3 = 3916 + 3698 - 3684;
    }
    else
    {
        int a6;
        a6 = 5067 * 3179 / 3287;
    }
    while(a < b)
        a = 10;
    while(a <= b)
    {
        while(a <= b)
        {
            while(a >= b);                                  //no colon required for while
            {
                int a3;
                a3 = 3323 == 2665 + 297 > 5816;
                int a4;
                a4 = 6423 + 3661 * 1998;
                                                            //while block not closed
        }
    }
}
```

Output:

```
D:\SEM-6 NOTES\Compiler Design\Assignment\PES2UG23CS819 CD ASSIGNMENT -1\COMPLETED_A1\PES2UG23CS819-ASSIGNMENT1>parser.e
xe assign-1_test-1_invalid.c
Starting parse...
Error at line 9: syntax error
Parsing failed.
```

assign-1_test-2_invalid.c:

```c
/* This file contians C code with syntax errors.
A description of the error is given as a comment next to the error
*/

int main()
{
    int -abcs;                                  // variable name cannot start with -
    int 1abc;                                   // variable name cannot start with a digit
    int double;                                 // keywords cannot be used as variable names
    double = 100;
    int a, b, c;
    a = a:c;
    if(a < b):                                  // : not expected after if condition
        a = 10;
    if a >= a                                   // condition should be in parentheses
    {
        a = 4436 + 2045 - 5360 * 8997;
    }
    if(a > b)
    {
        if(a < b)
        {
            if(a == b)
            {
                int a5;
                a5 = 7876 * 1661 +* 146;        //+* is not a valid operator
            }
            else                                //else block missing
        }
    }
    while(a <= b)
        while(a <= b)
            while(a >= b)
            {
                int a3;
                a3 = 3323 == 2665 + 297 > 5816;
                int a4;
                a4 = 6423 + 3661 * 1998 * 9083 > 2841;
            }
}
```

Output:

```
D:\SEM-6 NOTES\Compiler Design\Assignment\PES2UG23CS819 CD ASSIGNMENT -1\COMPLETED_A1\PES2UG23CS819-ASSIGNMENT1>parser.e
xe assign-1_test-2_invalid.c
Starting parse...
Error at line 8: syntax error
Parsing failed.
```