Compiler Design

Lab Assignment-1

Syntax Analyser

Name: Arjun N R

SRN: PES2UG22CS910

Lex.l

```
%{
#include "y.tab.h"
#include <stdio.h>
#include <string.h>
extern int line_number;
%}

DIGIT   [0-9]
LETTER  [a-zA-Z]
ID      {LETTER}({LETTER}|{DIGIT})*
NUM     {DIGIT}+(\.{DIGIT}+)?([eE][+-]?{DIGIT}+)?
%%

"int"           { return INT; }
"float"         { return FLOAT; }
"double"        { return DOUBLE; }
"char"          { return CHAR; }
"void"          { return VOID; }
"if"            { return IF; }
```

```
"else"        { return ELSE; }
"for"         { return FOR; }
"while"       { return WHILE; }
"do"          { return DO; }
"switch"      { return SWITCH; }
"case"        { return CASE; }
"default"     { return DEFAULT; }
"break"       { return BREAK; }
"continue"    { return CONTINUE; }
"return"      { return RETURN; }
"main"        { return MAIN; } // Recognize main function
"+"           { return PLUS; }
"-"           { return MINUS; }
"*"           { return TIMES; }
"/"           { return DIVIDE; }
"!"           { return NOT; }
"++"          { return INC; }
"--"          { return DEC; }

"("           { return LPAREN; }
")"           { return RPAREN; }
"{"           { return LBRACE; }
"}"           { return RBRACE; }
"["           { return LBRACKET; }
"]"           { return RBRACKET; }
";"           { return SEMICOLON; }
```

```
","             { return COMMA; }
":"             { return COLON; }
"="             { return ASSIGN; }
"=="            { return EQ; }
"!="            { return NEQ; }
"<"             { return LT; }
">"             { return GT; }
"<="            { return LTE; }
">="            { return GTE; }
"&&"            { return AND; }
"||"            { return OR; }
{ID}            { yylval.str = strdup(yytext); return ID; }
{NUM}           { yylval.str = strdup(yytext); return NUM; }
"\'"(\\.|[^'\\])"\'"   { yylval.str = strdup(yytext); return NUM; }
"//".*          { /* Ignore single-line comments */ }
"/*"[^*]*"*/" { /* Ignore multi-line comments */ }
[ \t]           { /* Ignore spaces and tabs */ }
\n              { line_number++; }
.               { fprintf(stderr, "Error: Illegal character '%s' at line %d\n", yytext,
line_number); }


%%


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

Parser.y

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

// Function prototypes
extern int yylex();
void yyerror(const char *s);

extern FILE *yyin;
extern char *yytext;
int line_number = 1; // Track line numbers for error reporting
int error_count = 0; // Track the number of errors

%}

// Define the union for token values
%union {
   int num;
   float fnum;
   char *str;
}

// Define tokens
%token <str> ID
%token <num> NUM
```

```
%token <fnum> FNUM

%token <str> STRING_LITERAL

%token INT FLOAT DOUBLE CHAR VOID

%token IF ELSE FOR WHILE DO SWITCH CASE BREAK CONTINUE RETURN MAIN DEFAULT

%token ASSIGN EQ NEQ LT GT LTE GTE AND OR PLUS MINUS TIMES DIVIDE NOT INC DEC

%token LPAREN RPAREN LBRACE RBRACE LBRACKET RBRACKET SEMICOLON COMMA COLON

%token UMINUS


// Precedence and associativity

%nonassoc LT GT LTE GTE EQ NEQ

%right ASSIGN

%left PLUS MINUS

%left TIMES DIVIDE

%precedence UMINUS


// Grammar starts here

%start program


%%


program:

   translation_unit

   ;
```

```
translation_unit:
    external_declaration translation_unit
    | external_declaration
    ;

external_declaration:
    function_definition
    | declaration
    ;

function_definition:
    type ID LPAREN RPAREN LBRACE statement_list opt_return RBRACE
    | main_function
    ;

main_function:
    INT MAIN LPAREN RPAREN LBRACE statement_list opt_return RBRACE
    | VOID MAIN LPAREN RPAREN LBRACE statement_list opt_return RBRACE
    ;

opt_return:
    RETURN expression SEMICOLON
    | /* empty */
    ;

statement_list:
```

```
    statement statement_list
    | error SEMICOLON { yyerrok; } // Generic error recovery: consume the semicolon
    | /* empty */
    ;

statement:
    declaration
    | assignment SEMICOLON
    | assignment error { yyerror("Missing semicolon after assignment, expected ';'"); yyerrok; } // Specific error for missing semicolon - placed early
    | if_stmt
    | for_loop
    | while_loop
    | switch_stmt
    | break_stmt SEMICOLON
    | continue_stmt SEMICOLON
    | RETURN expression SEMICOLON
    | SEMICOLON
    | LBRACE statement_list RBRACE
    | function_call SEMICOLON
    | /* empty */
    ;

declaration:
    type variable_list SEMICOLON
    ;
```

```
type:
    INT | FLOAT | DOUBLE | CHAR | VOID
    ;

variable_list:
    variable
    | variable COMMA variable_list
    ;

variable:
    ID
    | ID ASSIGN expression
    | ID array_declaration
    ;

array_declaration:
    LBRACKET NUM RBRACKET
    | LBRACKET NUM RBRACKET array_declaration
    ;

assignment:
    ID ASSIGN expression
    | ID array_indices ASSIGN expression
    ;
```

```
assignment_statement:
    assignment SEMICOLON
    ;

array_indices:
    LBRACKET expression RBRACKET
    | array_indices LBRACKET expression RBRACKET
    ;

expression:
      expression relop e_expression
    | e_expression
    | STRING_LITERAL  // Allow string literals in expressions
    ;

relop:
      LT
    | GT
    | LTE
    | GTE
    | EQ
    | NEQ
    ;

e_expression:
      e_expression PLUS t
```

```
    | e_expression MINUS t
    | t
    ;

t:
     t TIMES f
    | t DIVIDE f
    | f
    ;

f:
     LPAREN expression RPAREN
    | ID
    | NUM
    ;

if_stmt:
    IF LPAREN condition RPAREN compound_stmt optional_else
    ;

optional_else:
    ELSE compound_stmt
    | ELSE LPAREN error RPAREN { yyerror("Condition not allowed with 'else' for 'else' block"); yyerrok; } // Error for condition after else (LPAREN detected)
    | ELSE error { yyerror("Syntax error in 'else' block"); yyerrok; } // General else error
    | /* empty */
```

;

for_loop:

FOR LPAREN assignment SEMICOLON condition SEMICOLON assignment RPAREN statement

;

compound_stmt:

LBRACE statement_list RBRACE

| statement

;

while_loop:

WHILE LPAREN condition RPAREN compound_stmt

| WHILE LPAREN condition RPAREN COLON error { yyerror("Colon not allowed after 'while' condition, expected '{'"); yyerrok; } // Error for colon after while condition

| WHILE LPAREN condition RPAREN error { yyerror("Syntax error in 'while' loop"); yyerrok; } // Generic while error

;

switch_stmt:

SWITCH LPAREN ID RPAREN LBRACE case_list RBRACE

;

case_list:

CASE NUM COLON statement case_list

```
    | DEFAULT COLON statement
    | /* empty */
    ;
condition:
    expression
    ;

break_stmt:
    BREAK
    ;

continue_stmt:
    CONTINUE
    ;

// Function Call related rules
function_call:
    ID LPAREN argument_list RPAREN
    ;

argument_list:
    /* empty */
    | expression
    | argument_list COMMA expression
    ;
%%
```

```
void yyerror(const char *s) {
    fprintf(stderr, "Error: %s, line number: %d, token: %s\n", s, line_number, yytext);
    error_count++;
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <input_file>\n", argv[0]);
        return 1;
    }

    FILE *fp = fopen(argv[1], "r");
    if (!fp) {
        perror("Error opening input file");
        return 1;
    }

    yyin = fp; // Set input stream to the file
    int result = yyparse();

    fclose(fp); // Close the file

    if (result == 0 && error_count == 0) {
        printf("Valid syntax\n");
    } else {
```

```
        printf("Syntax error(s): %d\n", error_count);
    }

    return 0;
}
```

Output screenshot: