Compiler Design

Assignment -1

Name:Arjun N R

SRN: PES2UG22CS910

Lex file:

```
%{
#include <stdio.h>
#include "y.tab.h"
extern int yylineno;
void yyerror(char *s); // Declaration of yyerror (no definition here)
#include <stdlib.h>
#include <string.h>

// Function to truncate identifier and report error
void truncate_identifier(char *identifier);

// Track column number
int column = 1;

%}

DIGIT   [0-9]
LETTER  [a-zA-Z_]
ID      {LETTER}({LETTER}|{DIGIT})*
INT_LITERAL     {DIGIT}+
FLOAT_LITERAL   {DIGIT}+\.{DIGIT}+([eE][+-]?{DIGIT}+)?
```

```
COMMENT_SINGLE  "//".*
COMMENT_MULTI_START "/*"
COMMENT_MULTI_END   "*/"
INCLUDE_DIRECTIVE   "#include"[ \t]*[<"][^>]+[>"]|[#include][ \t]*["]([^"])+["]

%%

"int"          { return T_INT; }
"float"        { return T_FLOAT; }
"char"         { return T_CHAR; }
"double"       { return T_DOUBLE; }
"if"           { return T_IF; }
"while"        { return T_WHILE; }
"for"          { return T_FOR; }
"do"           { return T_DO; }
"else"         { return T_ELSE; }
"switch"       { return T_SWITCH; }
"case"         { return T_CASE; }
"break"        { return T_BREAK; }
"default"      { return T_DEFAULT; }
"main"         { return T_MAIN; }
"+"            { return '+'; }
"-"            { return '-'; }
"*"            { return '*'; }
"/"            { return '/'; }
"("            { return '('; }
```

```
")"           { return ')'; }

"{"           { return '{'; }

"}"           { return '}'; }

"["           { return '['; }

"]"           { return ']'; }

";"           { return ';'; }

","           { return ','; }

"="           { return '='; }

"<="          { return T_LE; }

">="          { return T_GE; }

"!="          { return T_NE; }

"=="          { return T_EQ; }

"&&"          { return T_AND; }

"||"          { return T_OR; }

"!"           { return '!'; }

"++"          { return T_INC; }

"--"          { return T_DEC; }

"%"           { return '%'; }

"<"           { return '<'; }

">"           { return '>'; }

"\'"          { return T_CHAR_QUOTE; }

{ID}          { if (strlen(yytext) > 31) { truncate_identifier(yytext); } yylval.sval =
strdup(yytext); return T_ID; }

{INT_LITERAL}   { yylval.sval = strdup(yytext); return T_NUM; }

{FLOAT_LITERAL} { yylval.sval = strdup(yytext); return T_NUM; }

{COMMENT_SINGLE} { column += strlen(yytext); /* Ignore single-line
comments */ }
```

```
{COMMENT_MULTI_START} {
    int c;
    column += 2; // Account for "/*"
    while (1) {
        c = input();
        if (c == EOF) {
            yyerror("Unterminated comment at end of file");
            return 0; // Or handle EOF appropriately
        }
        column++;
        if (c == '*') {
            if (input() == '/') {
                column += 2; // Account for "*/"
                break; // Comment ends
            } else {
                yyless(1); // Put back the character after '*' if it's not '/'
            }
        }
        if (c == '\n') {
            yylineno++;
            column = 1;
        }
    }
}
{INCLUDE_DIRECTIVE} { column += strlen(yytext); /* Ignore include directives */
}
[ \t]+        { column += strlen(yytext); /* ignore whitespace */ }
```

```
\n              { yylineno++; column = 1; }
.               { yyerror("Invalid character"); column++; }

%%

int yywrap(void) {
   return 1;
}

void truncate_identifier(char *identifier) {
   printf("Warning: Identifier '%s' is longer than 31 characters. Truncating to
'%.31s' at line %d, column %d.\n",
          identifier, identifier, yylineno, column);
   identifier[31] = '\0'; // Truncate the identifier
}

// Error function is only declared in lexer, defined in parser.y
// void yyerror(char *s) {
//     fprintf(stderr, "Error: %s at line %d, column %d\n", s, yylineno, column);
// }
```

Parser file

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
extern int yylineno;
```

```
extern FILE *yyin;
extern int yyparse();
void yyerror(char *s);
int yylex();
extern int column;
extern char *yytext;
int skip_to_sync(void);
int has_errors = 0;
%}

%union {
   char *sval;
}

%token <sval> T_ID T_NUM
%token T_INT T_FLOAT T_CHAR T_DOUBLE T_IF T_WHILE T_FOR T_DO T_ELSE T_SWITCH T_CASE T_BREAK T_DEFAULT T_MAIN
%token T_LE T_GE T_NE T_EQ T_AND T_OR T_NOT T_INC T_DEC T_CHAR_QUOTE

%%

P : T_INT T_MAIN '(' ')' '{' S '}'
  | error { has_errors = 1; skip_to_sync(); yyerrok; if (yylex() == 0) YYABORT; }
  ;

S : VarDeclr
```

```
  | AssignExpr ';' S

  | AssignExpr error { has_errors = 1; printf("Error: missing semicolon, line number: %d, token: %s\n", yylineno, yytext); skip_to_sync(); yyerrok; if (yylex() == 0) YYABORT; } S

  | T_IF '(' COND ')' '{' S '}' el S

  | T_IF '(' COND ')' S

  | T_IF '(' COND ')' '{' S '}' T_ELSE T_IF '(' COND ')' '{' S '}' el S

  | T_WHILE '(' COND ')' '{' S '}' S

  | T_WHILE '(' COND ')' S

  | T_DO '{' S '}' T_WHILE '(' COND ')' ';' S

  | T_FOR '(' ForInit ';' COND ';' Update ')' '{' S '}' S

  | Type T_ID '[' T_NUM ']' B ';' S

  | Type T_ID '[' T_NUM ']' '=' '{' Arrayelements '}' ';' S

  | T_SWITCH '(' D ')' '{' swt '}' S

  | /* empty */

  | error { has_errors = 1; printf("Error: syntax error, line number: %d, token: %s\n", yylineno, yytext); skip_to_sync(); yyerrok; if (yylex() == 0) YYABORT; } S

  ;


swt : T_CASE D ':' S T_BREAK ';' swt

    | T_DEFAULT ':' S

    | /* empty */

    ;


D : T_ID

  | T_NUM

  ;
```

```
el : T_ELSE '{' S '}'
   | /* empty */
   | T_ELSE '(' error { has_errors = 1; printf("Error: syntax error, line number: %d, token: else\n", yylineno); skip_to_sync(); yyerrok; }
   | T_ELSE error { has_errors = 1; printf("Error: dangling else, line number: %d, token: else\n", yylineno); skip_to_sync(); yyerrok; }
   ;

Arrayelements : T_NUM ',' Arrayelements
              | T_NUM
              ;

B : '[' T_NUM ']' B
  | /* empty */
  ;

Update : AssignExpr
       | /* empty */
       ;

M : T_INC
  | T_DEC
  ;

COND : E
```

```
    | error { has_errors = 1; printf("Error: syntax error, line number: %d, token: %s\n", yylineno, yytext); skip_to_sync(); yyerrok; if (yylex() == 0) YYABORT; }
    ;

ForInit : Type InitDeclarator
        | AssignExpr
        | /* empty */
        ;

AssignExpr : T_ID '=' E
           | T_ID '[' E ']' '=' E
           | T_ID M
           | M T_ID
           | Type T_ID '=' T_CHAR_QUOTE T_ID T_CHAR_QUOTE
           | T_ID '=' T_CHAR_QUOTE T_ID T_CHAR_QUOTE
           ;

E : E '+' T
  | E '-' T
  | E REL T
  | E LOG T
  | T
  ;

REL : '>'
    | '<'
    | T_LE
```

```
    | T_GE
    | T_NE
    | T_EQ
    ;

LOG : T_AND
    | T_OR
    | '!'
    ;

T : T '*' F
  | T '/' F
  | T '%' F
  | F
  ;

F : '(' E ')'
  | T_ID
  | T_NUM
  | M F
  ;

VarDeclr : Type ListVar ';' S
         ;

ListVar  : InitDeclarator
```

```
        | ListVar ',' InitDeclarator
        ;

InitDeclarator : T_ID
            | T_ID '=' E
            ;

Type : T_INT
    | T_FLOAT
    | T_CHAR
    | T_DOUBLE
    ;

%%

void yyerror(char *s) {
   if (strcmp(s, "Unrecognized token") == 0) {
      printf("Error: Unrecognized token, line number: %d, token: %s\n",
yylineno, yytext);
   }
}

int skip_to_sync(void) {
   int c;
   int brace_count = 0;
   while ((c = yylex()) != 0) {
      if (c == '{') {
```

```
            brace_count++;
        } else if (c == '}') {
            brace_count--;
            if (brace_count < 0) {
                printf("Error: unmatched closing brace, line number: %d, token: %s\n", yylineno, yytext);
                has_errors = 1;
                return c;
            }
        } else if (c == ';' && brace_count == 0) {
            return c;
        } else if (c == ':' && brace_count == 0) {
            printf("Error: Unrecognized token, line number: %d, token: %s\n", yylineno, yytext);
            has_errors = 1;
            return c;
        } else if (c == T_ELSE && brace_count == 0) {
            return c;
        } else if (c == T_IF && brace_count == 0) {
            return c;
        } else if (c == T_WHILE && brace_count == 0) {
            return c;
        }
    }
    if (brace_count > 0) {
        printf("Error: %d unmatched opening brace(s), line number: %d\n", brace_count, yylineno);
```

```
        has_errors = 1;
    }
    return 0;
}

int main(int argc, char *argv[]) {
    FILE *fp;
    has_errors = 0;
    if (argc > 1) {
        fp = fopen(argv[1], "r");
        if (!fp) {
            perror("Error opening input file");
            exit(1);
        }
        yyin = fp;
    }
    yylineno = 1;
    column = 1;
    if (!yyparse() && !has_errors)
        printf("Valid syntax\n");
    else
        printf("Parsing failed.\n");
    if (argc > 1)
        fclose(fp);
    return 0;
}
```

Valid testcases output:

```
PS C:\Users\arjun\Documents\SEM-6\CD\CompilerDesign\Assignment\Codebase\PES2UG22CS910> ./a.exe .\assignment-1_simple_for_valid.c
Valid syntax
PS C:\Users\arjun\Documents\SEM-6\CD\CompilerDesign\Assignment\Codebase\PES2UG22CS910> ./a.exe .\assignment-1_simple_do_while_valid.c
Valid syntax
PS C:\Users\arjun\Documents\SEM-6\CD\CompilerDesign\Assignment\Codebase\PES2UG22CS910> ./a.exe .\assignment-1_nested_for_valid.c
Valid syntax
PS C:\Users\arjun\Documents\SEM-6\CD\CompilerDesign\Assignment\Codebase\PES2UG22CS910> ./a.exe .\assignment-1_nested_do_while_valid.c
Valid syntax
PS C:\Users\arjun\Documents\SEM-6\CD\CompilerDesign\Assignment\Codebase\PES2UG22CS910>
```

Invalid testcases output:

```
PS C:\Users\arjun\Documents\SEM-6\CD\CompilerDesign\Assignment\Codebase\PES2UG22CS910> ./a.exe .\assign-1_test-1_invalid.c
Error: missing semicolon, line number: 10, token: if
Error: syntax error, line number: 15, token: a
Error: syntax error, line number: 22, token: int
Error: syntax error, line number: 23, token: =
Error: syntax error, line number: 24, token: }
Error: syntax error, line number: 25, token: (
Error: syntax error, line number: 27, token: while
Parsing failed.
```

```
PS C:\Users\arjun\Documents\SEM-6\CD\CompilerDesign\Assignment\Codebase\PES2UG22CS910> ./a.exe .\assign-1_test-2_invalid.c
Error: syntax error, line number: 8, token: -
Error: syntax error, line number: 9, token: int
Error: syntax error, line number: 10, token: int
Error: syntax error, line number: 11, token: double
Error: syntax error, line number: 12, token: int
Error: syntax error, line number: 13, token: a
Error: syntax error, line number: 14, token: if
Error: syntax error, line number: 16, token: if
Error: syntax error, line number: 20, token: (
Error: syntax error, line number: 32, token: (
Error: syntax error, line number: 33, token: (
Error: syntax error, line number: 34, token: (
Error: unmatched closing brace, line number: 41, token: }
Error: syntax error, line number: 32, token: (
Error: syntax error, line number: 33, token: (
Error: syntax error, line number: 34, token: (
Error: unmatched closing brace, line number: 41, token: }
Parsing failed.
```