Lab 2: Symbol Table

Name:Kushagra Agarwal
SRN:PES2UG22CS275

Lexer.l

```
%{
   #define YYSTYPE char*
   #include "y.tab.h"
   #include <stdio.h>
   #include <string.h>

   extern void yyerror(const char *); // Declare error handling function
%}

/* Regular Definitions */
digit   [0-9]
letter  [a-zA-Z]
id      {letter}({letter}|{digit})*
digits  {digit}+
opFraction  (\.{digits})?
opExponent  ([Ee][+-]?{digits})?
number      {digits}{opFraction}{opExponent}?

%option yylineno

%%
   // Ignore comments
   \/\/.*                  { /* Ignore single-line comments */ }

   // Ignore whitespace
   [\t\n ]                 { /* Ignore spaces and newlines */ }

   // Keywords
   "int"                   { return T_INT; }
   "char"                    { return T_CHAR; }
   "double"                    { return T_DOUBLE; }
   "float"                  { return T_FLOAT; }
   "while"                   { return T_WHILE; }
   "if"                    { return T_IF; }
   "else"                   { return T_ELSE; }
   "do"                     { return T_DO; }
   "#include"                  { return T_INCLUDE; }
   "main"                    { return T_MAIN; }

   // String literals
   \".*\"                  { yylval = strdup(yytext); return T_STRLITERAL; }

   // Operators & Comparisons
   "=="                     { return T_EQCOMP; }
   "!="                    { return T_NOTEQUAL; }
   ">="                     { return T_GREATEREQ; }
   "<="                     { return T_LESSEREQ; }
   "("                     { return '('; }
   ")"                        { return ')'; }
   "{"                        { return '{'; }
   "}"                        { return '}'; }
   "*"                        { return '*'; }
   "+"                        { return '+'; }
   "-"                        { return '-'; }
   "/"                        { return '/'; }
   "="                        { return '='; }
   ">"                        { return '>'; }
   "<"                        { return '<'; }
   ";"                        { return ';'; }

   // Number (integer, fraction, or exponent)
   {number}                   { yylval = strdup(yytext); return T_NUM; }

   // Header file names (e.g., stdio.h)
   {id}\.h                    { return T_HEADER; }

   // Identifiers
   {id}                       { yylval = strdup(yytext); return T_ID; }

   // Handle unrecognized characters
   .                          { return yytext[0]; }
%%

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

Parser.y

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

    #define YYSTYPE char*

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

    int current_type;
    int current_scope = 1;
    char temp[100];
%}

%token T_INT T_CHAR T_DOUBLE T_WHILE T_INC T_DEC T_OROR T_ANDAND T_EQCOMP
T_NOTEQUAL
%token T_GREATEREQ T_LESSEREQ T_LEFTSHIFT T_RIGHTSHIFT T_PRINTLN T_STRING
T_FLOAT
%token T_BOOLEAN T_IF T_ELSE T_STRLITERAL T_DO T_INCLUDE T_HEADER T_MAIN T_ID
T_NUM

%start START

%%
```

```
START :
    PROG { printf("Valid syntax\n"); YYACCEPT; }
;

PROG :
    MAIN PROG
    | DECLR ';' PROG
    | ASSGN ';' PROG
    | /* empty */
;

DECLR :
    TYPE LISTVAR
;

LISTVAR :
    LISTVAR ',' VAR
    | VAR
;

VAR :
    T_ID '=' EXPR {
        int idx = find_symbol($1, current_scope);
        if (idx != -1) {
            printf("Variable %s already declared\n", $1);
            printf("Error: %s at line %d\n", $1, yylineno);
        } else {
            insert_symbol($1, get_size(current_type), current_type, yylineno, current_scope, $3);
        }
    }
    | T_ID {
        int idx = find_symbol($1, current_scope);
        if (idx != -1) {
            printf("Variable %s already declared\n", $1);
            printf("Error: %s at line %d\n", $1, yylineno);
        } else {
            insert_symbol($1, get_size(current_type), current_type, yylineno, current_scope, NULL);
        }
    }
;

TYPE :
    T_INT    { current_type = 2; }
    | T_FLOAT  { current_type = 3; }
    | T_DOUBLE { current_type = 4; }
    | T_CHAR   { current_type = 1; }
;

ASSGN :
    T_ID '=' EXPR {
        int idx = find_symbol($1, current_scope);
        if (idx == -1) {
            printf("Variable %s not declared\n", $1);
            printf("Error: %s at line %d\n", $1, yylineno);
        } else {
            update_symbol_value(idx, $3);
        }
    }
;
```

```
EXPR :
    EXPR REL_OP E { $$ = $1; }
    | E { $$ = $1; }
;

E :
    E '+' T { sprintf(temp, "%f", atof($1) + atof($3)); $$ = strdup(temp); }
    | E '-' T { sprintf(temp, "%f", atof($1) - atof($3)); $$ = strdup(temp); }
    | T { $$ = $1; }
;

T :
    T '*' F { sprintf(temp, "%f", atof($1) * atof($3)); $$ = strdup(temp); }
    | T '/' F {
        if (atof($3) != 0) {
            sprintf(temp, "%f", atof($1) / atof($3));
            $$ = strdup(temp);
        } else {
            yyerror("Division by zero");
            $$ = strdup("0");
        }
    }
    | F { $$ = $1; }
;

F :
    '(' EXPR ')' { $$ = $2; }
    | T_ID {
        int idx = find_symbol($1, current_scope);
        if (idx == -1) {
            printf("Variable %s not declared\n", $1);
            printf("Error: %s at line %d\n", $1, yylineno);
            $$ = strdup("0");
        } else if (!symtab[idx].value) {
            printf("Variable %s not initialized\n", $1);
            printf("Error: %s at line %d\n", $1, yylineno);
            $$ = strdup("0");
        } else {
            $$ = strdup(symtab[idx].value);
        }
    }
    | T_NUM { $$ = $1; }
    | T_STRLITERAL { $$ = $1; }
;

REL_OP :
    T_LESSEREQ
    | T_GREATEREQ
    | '<'
    | '>'
    | T_EQCOMP
    | T_NOTEQUAL
;

MAIN :
    TYPE T_MAIN '(' EMPTY_LISTVAR ')' '{' STMT '}'
;

EMPTY_LISTVAR :
    LISTVAR
    | /* empty */
;

STMT :
    STMT_NO_BLOCK STMT
    | BLOCK STMT
    | /* empty */
;

STMT_NO_BLOCK :
    DECLR ';'
    | ASSGN ';'
;

BLOCK :
    '{' STMT '}'
;

%%

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

int main(int argc, char* argv[]) {
    init_table();
    yyparse();
    display_symbol_table();
    return 0;
}
```

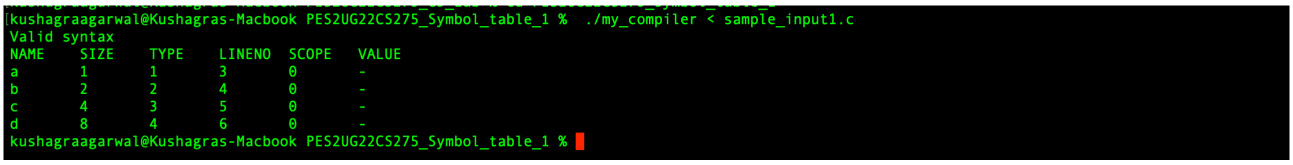
Sym_tab.c

```
#include "sym_tab.h"
// Define the global symbol table
struct symbol symtab[100];
int symtab_index = 0;
// Initialize symbol table
void init_table() {
symtab_index = 0;
}
// Find a symbol in the table
int find_symbol(char* name, int scope) {
for(int i = 0; i < symtab_index; i++) {
if(strcmp(symtab[i].name, name) == 0 && symtab[i].scope == scope) {
return i;
}
}
return -1;
}
// Insert a symbol into the table
void insert_symbol(char* name, int size, int type, int lineno, int scope, char* value) {
symtab[symtab_index].name = strdup(name);
symtab[symtab_index].size = size;
symtab[symtab_index].type = type;
symtab[symtab_index].lineno = lineno;
symtab[symtab_index].scope = scope;
symtab[symtab_index].value = value ? strdup(value) : NULL;
symtab_index++;
}
// Update symbol value
void update_symbol_value(int index, char* value) {
if(symtab[index].value) {
free(symtab[index].value);
}
symtab[index].value = strdup(value);
}
// Display symbol table
void display_symbol_table() {
printf("Name\tsize\ttype\tlineno\tscope\tvalue\n");
for(int i = 0; i < symtab_index; i++) {
printf("%s\t%d\t%d\t%d\t%d\t%s\n"
,
symtab[i].name,
symtab[i].size,
symtab[i].type,
symtab[i].lineno,
symtab[i].scope,
symtab[i].value ? symtab[i].value : "");
}
}// Get type size
int get_size(int type) {
    switch(type) {
    case 1: return 1; // char
    case 2: return 2; // int
    case 3: return 4; // float
    case 4: return 8; // double
    default: return 0;
    }
    }
```

sym_tab.h

```
#ifndef SYM_TAB_H
#define SYM_TAB_H
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// Symbol table entry structure
struct symbol {
char* name;
int size;
int type;
int lineno;
int scope;
char* value;
};
// Make symtab accessible to other files
extern struct symbol symtab[100];
extern int symtab_index;
// Function declarations
void init_table();
int find_symbol(char* name, int scope);
void insert_symbol(char* name, int size, int type, int lineno, int scope, char* value);
void update_symbol_value(int index, char* value);
void display_symbol_table();
int get_size(int type);
```

```
#endif
```



```
kushagraagarwal@Kushagras-Macbook PES2UG22CS275_Symbol_table_1 %  ./my_compiler < sample_input1.c
Valid syntax
NAME    SIZE    TYPE    LINENO  SCOPE   VALUE
a       1       1       3       0       -
b       2       2       4       0       -
c       4       3       5       0       -
d       8       4       6       0       -
kushagraagarwal@Kushagras-Macbook PES2UG22CS275_Symbol_table_1 % 
```