Compiler design Lab-2 Task

Name:Arjun N R

SRN: PES2UG22CS910

Lexer.l file

```
%{
   #define YYSTYPE char*
   #include "y.tab.h"
   #include <stdio.h>
   #include <stdlib.h>
   #include <string.h>
   extern void yyerror(const char *);
   extern int yylineno;
%}

/* 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

%%
\/\/(.*) ;
[\t] ;
(\r\n|\n|\r) { yylineno;}
```

```
"int"           { yylval=strdup(yytext); return T_INT;}
"char"          { yylval=strdup(yytext); return T_CHAR;}
"double"        { yylval=strdup(yytext); return T_DOUBLE;}
"float"         { yylval=strdup(yytext); 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;}
\".*\"          {yylval=strdup(yytext); return T_STRLITERAL; }
"=="            {return T_EQCOMP;}
"!="            {return T_NOTEQUAL;}
">="            {return T_GREATEREQ;}
"<="            {return T_LESSEREQ;}
"("             {return *yytext;}
")"             {return *yytext;}
"."             {return *yytext;}
","             {return *yytext;}
"{"             {return *yytext;}
"}"             {return *yytext;}
"*"             {return *yytext;}
"+"             {return *yytext;}
";"             {return *yytext;}
"-"             {return *yytext;}
"/"             {return *yytext;}
"="             {return *yytext;}
">"             {return *yytext;}
"<"             {return *yytext;}
```

```
{number}        {
                        yylval=strdup(yytext);
                        return T_NUM;
                }
{id}\.h  {return T_HEADER;}
{id}            {
                        yylval=strdup(yytext);
                        return T_ID;
                }
.               {}
%%
int yywrap() {
   return 1;
}
```

Parser file

```
%{
        #include "sym_tab.h"
        #include "sym_tab.c" // Including .c for simplicity in this example. In real projects, compile and link.
        #include <stdio.h>
        #include <stdlib.h>
        #include <string.h>
        #define YYSTYPE char*

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

```
    table* sym_table; // Global symbol table
    int current_scope = 1; // Default scope is 1

    int current_type; // To store the current type being declared
    int get_type_code(char* type_str); // Function to get type code from string
    int get_size_from_type(int type_code); // Function to get size from type code

%}

%token T_INT T_CHAR T_DOUBLE T_WHILE  T_INC T_DEC   T_OROR T_ANDAND T_EQCOMP
T_NOTEQUAL T_GREATEREQ T_LESSEREQ T_LEFTSHIFT T_RIGHTSHIFT T_PRINTLN T_STRING
T_FLOAT 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"); display_symbol_table(sym_table); YYACCEPT; }
      ;

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


DECLR : TYPE { current_type = get_type_code($1); free($1); } LISTVAR
        ;
```

```
LISTVAR : LISTVAR ',' VAR
        | VAR
        ;

VAR: T_ID '=' EXPR      {
        if (check_symbol_table(sym_table, $1)) {
            fprintf(stderr, "Error: Redeclaration of variable '%s' at line %d\n", $1, yylineno);
        } else {
            int size = get_size_from_type(current_type);
            symbol* sym = init_symbol($1, size, current_type, yylineno, current_scope);
            insert_into_table(sym_table, sym);
        }
        free($1);
                        }
    | T_ID              {
        if (check_symbol_table(sym_table, $1)) {
            fprintf(stderr, "Error: Redeclaration of variable '%s' at line %d\n", $1, yylineno);
        } else {
            int size = get_size_from_type(current_type);
            symbol* sym = init_symbol($1, size, current_type, yylineno, current_scope);
            insert_into_table(sym_table, sym);
        }
        free($1);
                        }

//assign type here to be returned to the declaration grammar
TYPE : T_INT { $$ = strdup($1); }
     | T_FLOAT { $$ = strdup($1); }
```

```
      | T_DOUBLE { $$ = strdup($1); }
      | T_CHAR { $$ = strdup($1); }
      ;

/* Grammar for assignment */
ASSGN : T_ID '=' EXPR            {
            if (!check_symbol_table(sym_table, $1)) {
               fprintf(stderr, "Error: Undeclared variable '%s' at line %d\n", $1, yylineno);
            } else {
               insert_value_to_name(sym_table, $1, $3);
            }
            free($1);
                              }
          ;

EXPR : EXPR REL_OP E
      | E
      ;

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


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

```
F : '(' EXPR ')'
   | T_ID
   | T_NUM
   | T_STRLITERAL
   ;

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


/* Grammar for main function */
MAIN : TYPE T_MAIN '(' EMPTY_LISTVAR ')' '{' STMT '}';

EMPTY_LISTVAR : LISTVAR
                |
                ;

STMT : STMT_NO_BLOCK STMT
     | BLOCK STMT
     | /*epsilon*/
     ;
```

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

BLOCK : '{' STMT '}';

COND : EXPR
      | ASSGN
      ;



%%



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

int get_type_code(char* type_str) {
   if (strcmp(type_str, "int") == 0) return INT_TYPE;
   if (strcmp(type_str, "char") == 0) return CHAR_TYPE;
   if (strcmp(type_str, "float") == 0) return FLOAT_TYPE;
   if (strcmp(type_str, "double") == 0) return DOUBLE_TYPE;
   return 0; // Unknown type
}

int get_size_from_type(int type_code) {
```

```
    switch (type_code) {
        case CHAR_TYPE: return 1;
        case INT_TYPE: return 2;
        case FLOAT_TYPE: return 4;
        case DOUBLE_TYPE: return 8;
        default: return 0; // Unknown size
    }
}


int main(int argc, char *argv[]) {
    // If an input file is provided as a command line argument, open it.
    if (argc > 1) {
        FILE *fp = freopen(argv[1], "r", stdin);
        if (!fp) {
            perror("Error opening file");
            exit(EXIT_FAILURE);
        }
    }
    sym_table = init_table();
    yyparse();
    return 0;
}
```

Symbol table .c

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

```
table* init_table()
{
    table* t = (table*)malloc(sizeof(table));
    if (t == NULL) {
        fprintf(stderr, "Memory allocation failed for symbol table.\n");
        exit(EXIT_FAILURE);
    }
    t->head = NULL;
    return t;
}

symbol* init_symbol(char* name, int size, int type, int lineno, int scope)
{
    symbol* s = (symbol*)malloc(sizeof(symbol));
    if (s == NULL) {
        fprintf(stderr, "Memory allocation failed for symbol entry.\n");
        exit(EXIT_FAILURE);
    }
    s->name = strdup(name); // Allocate memory and copy name
    s->size = size;
    s->type = type;
    s->val = NULL; // Value will be updated later
    s->line = lineno;
    s->scope = scope;
    s->next = NULL;
    return s;
}
```

```
void insert_into_table(table* sym_table, symbol* sym_entry)
{
    if (sym_table == NULL || sym_entry == NULL) {
        fprintf(stderr, "Invalid arguments to insert_into_table.\n");
        return;
    }

    if (sym_table->head == NULL) {
        sym_table->head = sym_entry;
    } else {
        symbol* current = sym_table->head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = sym_entry;
    }
}

int check_symbol_table(table* sym_table, char* name)
{
    if (sym_table == NULL || name == NULL) {
        return 0; // Not found or invalid table
    }

    symbol* current = sym_table->head;
    while (current != NULL) {
        if (strcmp(current->name, name) == 0) {
            return 1; // Found
        }
```

```
        current = current->next;
    }
    return 0; // Not found
}

void insert_value_to_name(table* sym_table, char* name, char* value)
{
    if (sym_table == NULL || name == NULL) {
        return;
    }
    if (value == NULL) return; // if value is default value return back

    symbol* current = sym_table->head;
    while (current != NULL) {
        if (strcmp(current->name, name) == 0) {
            if (current->val != NULL) free(current->val); // Free old value if exists
            current->val = strdup(value);
            return;
        }
        current = current->next;
    }
     printf("Warning: Variable '%s' not found in symbol table to update value.\n", name);
}

void display_symbol_table(table* sym_table)
{
    if (sym_table == NULL) {
        printf("Symbol table is NULL.\n");
        return;
```

```
    }

    printf("Symbol Table:\n");
    printf("-----------------------------------------------------------------\n");
    printf("Name\tSize\tType\tLine No\tScope\tValue\n");
    printf("-----------------------------------------------------------------\n");

    symbol* current = sym_table->head;
    while (current != NULL) {
        char* type_str;
        switch (current->type) {
            case CHAR_TYPE: type_str = "char"; break;
            case INT_TYPE: type_str = "int"; break;
            case FLOAT_TYPE: type_str = "float"; break;
            case DOUBLE_TYPE: type_str = "double"; break;
            default: type_str = "unknown"; break;
        }
        printf("%s\t%d\t%s\t%d\t%d\t%s\n", current->name, current->size, type_str, current-
>line, current->scope, current->val != NULL ? current->val : "~");
        current = current->next;
    }
    printf("-----------------------------------------------------------------\n");
}
```

Symbol table.h

```
#ifndef SYM_TAB_H
#define SYM_TAB_H

#define CHAR_TYPE 1
```

```
#define INT_TYPE 2
#define FLOAT_TYPE 3
#define DOUBLE_TYPE 4

typedef struct symbol
{
   char* name;
   int size;
   int type;
   char* val;
   int line;
   int scope;
   struct symbol* next;
} symbol;

typedef struct table
{
   symbol* head;
} table;

extern table* sym_table; // Declare the global symbol table

table* init_table();

symbol* init_symbol(char* name, int size, int type, int lineno, int scope);

void insert_into_table(table* sym_table, symbol* sym_entry);

void insert_value_to_name(table* sym_table, char* name, char* value);
```

```
int check_symbol_table(table* sym_table, char* name);

void display_symbol_table(table* sym_table);

#endif // SYM_TAB_H
```

Sample input file

```
int main()
{
   int a;
   float b;
   double c;
   char d;
   a=10;
}
```

Output screenshot

```
PS C:\Users\arjun\Documents\SEM-6\CD\CompilerDesign\Lab2\codebase\PES2UG22CS910> ./a.exe .\sample_input1.c
Valid syntax
Symbol Table:
-------------------------------------------------------------------
Name    Size    Type    Line No Scope   Value
-------------------------------------------------------------------
a       2       int     3       1       10
b       4       float   4       1       ~
c       8       double  5       1       ~
d       1       char    6       1       ~
-------------------------------------------------------------------
```