PES University, Bengaluru

Department of Computer Science and Engineering

UE22CS341B: COMPILER DESIGN
NAME: Rohit Yakkundi
SRN: PES2UG22CS819 (Sec – G)
LAB - 3

---

lexer.l:

```
%{
   #define YYSTYPE char*
   #include "y.tab.h"
   #include <stdio.h>
   extern void yyerror(const char*); // declare the error handling function
   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}

%%
\/\/(.*) ; // ignore comments
[\f\r\t ] ; // ignore whitespaces
\n                {++yylineno;}
"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;}
```

```
"for"              {return T_FOR;} /* added to support for */
"do"               {return T_DO;}
"#include"         {return T_INCLUDE;}
"main"  {return T_MAIN;}
\".*\"             {return T_STRLITERAL; }
"++"               {return T_INC;}   /* added to support unary increment op */
"--"               {return T_DEC;}   /* added to support unary decrement op */
"=="               {return T_EQCOMP;}
"!="               {return T_NOTEQUAL;}
">="               {return T_GREATEREQ;}
"<="               {return T_LESSEREQ;}
"||"               {return T_OROR;}
"&&"               {return T_ANDAND;}
"("                {return *yytext;}
")"                {return *yytext;}
"."                {return *yytext;}
","                {return *yytext;}
"{"                {return *yytext;}
"}"                {return *yytext;}
"["                {return *yytext;} /* added to support arrays */
"]"                {return *yytext;} /* added to support arrays */
"*"                {return *yytext;}
"+"                {return *yytext;}
";"                {return *yytext;}
"-"                {return *yytext;}
"/"                {return *yytext;}
"="                {return *yytext;}
">"                {return *yytext;}
"<"                {return *yytext;}
"!"                {return *yytext;}
{number}           {return T_NUM;}
{id}\.h   {return T_HEADER;}  // ending in .h => header file name
{id}               {return T_ID;}
.                  {yyerror("Unrecognized token");}
%%
```

parser.y

```
%{
   #include "sym_tab.c"
   #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 = 0;
%}

%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
%token T_LEFTSHIFT T_RIGHTSHIFT T_PRINTLN T_STRING T_FLOAT T_BOOLEAN
T_IF T_ELSE T_STRLITERAL T_DO T_INCLUDE
%token T_HEADER T_MAIN T_ID T_NUM

/* Define operator precedence and associativity */
%right '='
%left T_OROR
%left T_ANDAND
%left T_EQCOMP T_NOTEQUAL
%left '<' '>' T_LESSEREQ T_GREATEREQ
%left '+' '-'
%left '*' '/'
%right UMINUS
%nonassoc IFX
%nonassoc T_ELSE

%start START

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

PROG : MAIN PROG
     | DECLR ';' PROG
     | ASSGN ';' PROG
     |
     ;
```

```
DECLR : TYPE LISTVAR
      ;

LISTVAR : LISTVAR ',' VAR
        | VAR
        ;

VAR : T_ID '=' EXPR    {
                    if(check_symbol_table($1) == 0) {
                        insert_into_table($1, get_size(current_type), current_type, yylineno,
current_scope);
                        insert_value_to_name($1, $3, current_type);
                    }
                    else {
                        printf("Error: Variable %s already declared\n", $1);
                        exit(1);
                    }
                }
    | T_ID           {
                    if(check_symbol_table($1) == 0) {
                        insert_into_table($1, get_size(current_type), current_type, yylineno,
current_scope);
                    }
                    else {
                        printf("Error: Variable %s already declared\n", $1);
                        exit(1);
                    }
                }
    ;

TYPE : T_INT          { current_type = INT; }
     | T_FLOAT        { current_type = FLOAT; }
     | T_DOUBLE       { current_type = DOUBLE; }
     | T_CHAR         { current_type = CHAR; }
     ;

ASSGN : T_ID '=' EXPR {
                    if(check_symbol_table($1) == 1) {
                        insert_value_to_name($1, $3, current_type);
                    }
                    else {
                        printf("Error: Variable %s not declared\n", $1);
                        exit(1);
                    }
                }
      ;

EXPR : EXPR '+' EXPR      { $$ = $1; }
```

```
    | EXPR '-' EXPR      { $$ = $1; }
    | EXPR '*' EXPR      { $$ = $1; }
    | EXPR '/' EXPR      { $$ = $1; }
    | EXPR '<' EXPR      { $$ = $1; }
    | EXPR '>' EXPR      { $$ = $1; }
    | EXPR T_LESSEREQ EXPR    { $$ = $1; }
    | EXPR T_GREATEREQ EXPR   { $$ = $1; }
    | EXPR T_EQCOMP EXPR      { $$ = $1; }
    | EXPR T_NOTEQUAL EXPR    { $$ = $1; }
    | EXPR T_ANDAND EXPR      { $$ = $1; }
    | EXPR T_OROR EXPR        { $$ = $1; }
    | '(' EXPR ')'      { $$ = $2; }
    | '-' EXPR %prec UMINUS { $$ = $2; }
    | T_ID              { $$ = $1; }
    | T_NUM             { $$ = $1; }
    | T_STRLITERAL      { $$ = $1; }
    ;

STMT_LIST : STMT_LIST STMT_ITEM
          |
          ;

STMT_ITEM : DECLR ';'
          | ASSGN ';'
          | COND_STMT
          | WHILE_STMT
          | DO_WHILE_STMT ';'
          | BLOCK
          ;

COND_STMT : T_IF '(' EXPR ')' STMT_ITEM %prec IFX
          | T_IF '(' EXPR ')' STMT_ITEM T_ELSE STMT_ITEM
          ;

WHILE_STMT : T_WHILE '(' EXPR ')' STMT_ITEM
           ;

DO_WHILE_STMT : T_DO STMT_ITEM T_WHILE '(' EXPR ')'
              ;

BLOCK : '{' { current_scope++; } STMT_LIST '}' { current_scope--; }
      ;

MAIN : TYPE T_MAIN '(' EMPTY_LISTVAR ')' BLOCK
    ;

EMPTY_LISTVAR : LISTVAR
              |
```

```
            ;

%%

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


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

Sym_tab.c

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

// Global symbol table definition
table* t = NULL;

// Allocate space for symbol table
table* allocate_space_for_table() {
    table* t = (table*)malloc(sizeof(table));
    t->head = NULL;
    return t;
}

// Allocate space for symbol table entry
symbol* allocate_space_for_table_entry(char* name, int size, int type, int lineno, int scope) {
    symbol* s = (symbol*)malloc(sizeof(symbol));
    s->name = strdup(name);
    s->size = size;
    s->type = type;
    s->val = NULL;
    s->line = lineno;
    s->scope = scope;
    s->next = NULL;
    return s;
```

```
}

// Insert symbol into table
void insert_into_table(char* name, int size, int type, int lineno, int scope) {
    symbol* s = allocate_space_for_table_entry(name, size, type, lineno, scope);
    if(t->head == NULL) {
        t->head = s;
    } else {
        symbol* current = t->head;
        while(current->next != NULL) {
            current = current->next;
        }
        current->next = s;
    }
}

// Insert value for a symbol
void insert_value_to_name(char* name, char* value, int type) {
    symbol* current = t->head;
    while(current != NULL) {
        if(strcmp(current->name, name) == 0) {
            current->val = strdup(value);
            return;
        }
        current = current->next;
    }
}

// Check if symbol exists in table
int check_symbol_table(char* name) {
    symbol* current = t->head;
    while(current != NULL) {
        if(strcmp(current->name, name) == 0) {
            return 1;  // Found
        }
        current = current->next;
    }
    return 0;  // Not found
}

// Get size of data type
int get_size(int type) {
    switch(type) {
        case INT:
            return 4;
        case CHAR:
            return 1;
        case FLOAT:
```

```
            return 4;
        case DOUBLE:
            return 8;
        default:
            return 0;
    }
}

// Display symbol table
void display_symbol_table() {
    printf("\nSymbol Table:\n");
    printf("Name\tType\tSize\tValue\tLine\tScope\n");
    printf("-----------------------------------------\n");

    symbol* current = t->head;
    while(current != NULL) {
        char* type_str;
        switch(current->type) {
            case INT: type_str = "int"; break;
            case CHAR: type_str = "char"; break;
            case FLOAT: type_str = "float"; break;
            case DOUBLE: type_str = "double"; break;
            default: type_str = "unknown"; break;
        }

        printf("%s\t%s\t%d\t%s\t%d\t%d\n",
               current->name,
               type_str,
               current->size,
               current->val ? current->val : "NULL",
               current->line,
               current->scope);
        current = current->next;
    }
    printf("-----------------------------------------\n");
}
```

Sym_tab.h

```
#ifndef SYM_TAB_H
#define SYM_TAB_H

#define CHAR 1
#define INT 2
#define FLOAT 3
#define DOUBLE 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;

// Global symbol table declaration
extern table* t;

// Function declarations with proper types
table* allocate_space_for_table();
symbol* allocate_space_for_table_entry(char* name, int size, int type, int lineno, int scope);
void insert_into_table(char* name, int size, int type, int lineno, int scope);
void insert_value_to_name(char* name, char* value, int type);
int check_symbol_table(char* name);
void display_symbol_table();
int get_size(int type);  // Added get_size declaration

#endif
```

Compilation screen shot:
Sample_input.c

D: > SEM-6 NOTES > Compiler Design > CD-2024 > PES2UG22CS819 > LAB-3-symboltable > C sample_input1.c > main()

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

```

```
D:\SEM-6 NOTES\Compiler Design\CD-2024\PES2UG22CS819\LAB-3-symboltable>flex lexer.l

D:\SEM-6 NOTES\Compiler Design\CD-2024\PES2UG22CS819\LAB-3-symboltable>bison -dy parser.y

D:\SEM-6 NOTES\Compiler Design\CD-2024\PES2UG22CS819\LAB-3-symboltable>gcc lex.yy.c y.tab.c

D:\SEM-6 NOTES\Compiler Design\CD-2024\PES2UG22CS819\LAB-3-symboltable>a.exe < sample_input1.c
Valid syntax

Symbol Table:
Name    Type    Size    Value   Line    Scope
-----------------------------------------
a       int     4       NULL    3       1
b       float   4       NULL    4       1
c       double  8       NULL    5       1
d       char    1       NULL    6       1
-----------------------------------------

D:\SEM-6 NOTES\Compiler Design\CD-2024\PES2UG22CS819\LAB-3-symboltable>
```