COMPILER DESIGN LAB 1

NAME: JANVII RV

SRN: PES2UG22CS232

LEX.L CODE:

```
%{
   #include "y.tab.h"
   #include <stdio.h>

   int yylineno = 1;
   void yyerror(const char*);
%}
digit [0-9]
id [a-zA-Z_][a-zA-Z0-9_]*
number {digit}+|{digit}*\.{digit}+
char '.'|'\\n'|'\\t'|'\\r'|'\\0'

%%
"/*"([^*]|\*+[^/])*"*/"      { /* ignore multi-line comments */ }
"//".*                       { /* ignore single-line comments */ }
"=="                         { return TOKEN_EQUAL; }
"!="                         { return TOKEN_NOT_EQUAL; }
">="                         { return TOKEN_GREATER_EQUAL; }
"<="                         { return TOKEN_LESS_EQUAL; }
"||"                         { return TOKEN_LOGICAL_OR; }
"&&"                         { return TOKEN_LOGICAL_AND; }
">"                          { return TOKEN_GREATER_THAN; }
```

```
"<"                    { return TOKEN_LESS_THAN; }

"="                    { return TOKEN_ASSIGN; }

"!"                    { return TOKEN_LOGICAL_NOT; }

"+"                    { return TOKEN_PLUS; }

"-"                    { return TOKEN_MINUS; }

"*"                    { return TOKEN_MULTIPLY; }

"/"                    { return TOKEN_DIVIDE; }

"%"                    { return TOKEN_MODULO; }

"("                    { return TOKEN_LEFT_PAREN; }

")"                    { return TOKEN_RIGHT_PAREN; }

"{"                    { return TOKEN_LEFT_BRACE; }

"}"                    { return TOKEN_RIGHT_BRACE; }

"["                    { return TOKEN_LEFT_BRACKET; }

"]"                    { return TOKEN_RIGHT_BRACKET; }

","                    { return TOKEN_COMMA; }

";"                    { return TOKEN_SEMICOLON; }

":"                    { return TOKEN_COLON; }

"int"                  { return TOKEN_INT; }

"main"                 { return TOKEN_MAIN; }

"char"                 { return TOKEN_CHAR; }

"double"               { return TOKEN_DOUBLE; }

"float"                { return TOKEN_FLOAT; }

"return"               { return TOKEN_RETURN; }

"break"                { return TOKEN_BREAK; }

"continue"             { return TOKEN_CONTINUE; }

"switch"               { return TOKEN_SWITCH; }

"case"                 { return TOKEN_CASE; }

"while"                { return TOKEN_WHILE; }
```

```
"do"                    { return TOKEN_DO; }
"if"                  { return TOKEN_IF; }
"for"                  { return TOKEN_FOR; }
"else"                  { return TOKEN_ELSE; }
"default"                 { return TOKEN_DEFAULT; }
{char}                  { yylval.char_val = yytext[1]; return TOKEN_CHAR_LITERAL; }
{number}                 { yylval.float_val = atof(yytext); return TOKEN_NUMBER; }
{id}                  { yylval.int_val = 0; /* Or assign something meaningful */ return
TOKEN_IDENTIFIER; }
\n                    { yylineno++; }
[ \t]+                  { /* ignore whitespace */ }
.                    { printf("Unexpected character: %s\n", yytext); }
%%

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

PARSER.Y CODE:

```
%{
#include <stdio.h>
#include <stdlib.h>
extern int yylineno;
void yyerror(const char *s);
int yylex(void);
typedef union {
   float float_val;
   int int_val;
   char char_val;
} YYSTYPE;

#define YYSTYPE_IS_DECLARED 1

int error_count = 0;

extern char *yytext;

%}


%union {

   float float_val;

   int int_val;

   char char_val;

}


%token TOKEN_NUMBER

%token TOKEN_CHAR_LITERAL

%token TOKEN_IDENTIFIER

%token TOKEN_GREATER_THAN TOKEN_LESS_THAN TOKEN_EQUAL TOKEN_ASSIGN
TOKEN_LESS_EQUAL TOKEN_GREATER_EQUAL

%token TOKEN_LEFT_PAREN TOKEN_RIGHT_PAREN TOKEN_LEFT_BRACE
TOKEN_RIGHT_BRACE TOKEN_LEFT_BRACKET TOKEN_RIGHT_BRACKET

%token TOKEN_PLUS TOKEN_MINUS TOKEN_MULTIPLY TOKEN_DIVIDE TOKEN_MODULO

%token TOKEN_COMMA TOKEN_SEMICOLON TOKEN_COLON

%token TOKEN_INT TOKEN_CHAR TOKEN_DOUBLE TOKEN_FLOAT TOKEN_RETURN
TOKEN_BREAK TOKEN_CONTINUE

%token TOKEN_SWITCH TOKEN_CASE TOKEN_WHILE TOKEN_DO TOKEN_IF TOKEN_FOR
TOKEN_ELSE TOKEN_DEFAULT TOKEN_MAIN

%token TOKEN_LOGICAL_OR TOKEN_LOGICAL_AND TOKEN_NOT_EQUAL
TOKEN_LOGICAL_NOT


/* Operator precedence and associativity - reordered for clarity */
```

```
%right TOKEN_ASSIGN

%left TOKEN_LOGICAL_OR

%left TOKEN_LOGICAL_AND

%left TOKEN_EQUAL TOKEN_NOT_EQUAL

%left TOKEN_LESS_THAN TOKEN_LESS_EQUAL TOKEN_GREATER_THAN
TOKEN_GREATER_EQUAL

%left TOKEN_PLUS TOKEN_MINUS

%left TOKEN_MULTIPLY TOKEN_DIVIDE TOKEN_MODULO

%right TOKEN_LOGICAL_NOT

%nonassoc LOWER_THAN_ELSE

%nonassoc TOKEN_ELSE


%%

program: TOKEN_INT TOKEN_MAIN TOKEN_LEFT_PAREN TOKEN_RIGHT_PAREN
TOKEN_LEFT_BRACE body TOKEN_RIGHT_BRACE

;


body: statements

;


statements: /* empty */

        | statements statement

        | statements declaration

        | statements error TOKEN_SEMICOLON {

            fprintf(stderr, "Error in statements at line %d\n", yylineno);

            yyerrok;

         }

;
```

```
declaration: datatype var_list TOKEN_SEMICOLON

;


datatype: TOKEN_INT

      | TOKEN_CHAR

      | TOKEN_FLOAT

      | TOKEN_DOUBLE

;


var_list: var_declaration

       | var_list TOKEN_COMMA var_declaration

;


var_declaration: TOKEN_IDENTIFIER

             | TOKEN_IDENTIFIER TOKEN_ASSIGN expr

             | array_declaration

;


array_declaration: TOKEN_IDENTIFIER TOKEN_LEFT_BRACKET TOKEN_NUMBER
TOKEN_RIGHT_BRACKET

;


statement: simple_statement

        | compound_statement

;


simple_statement: assignment TOKEN_SEMICOLON

              | TOKEN_BREAK TOKEN_SEMICOLON
             | TOKEN_CONTINUE TOKEN_SEMICOLON
             | return_stmt
;

compound_statement: conditional
              | loop
              | switch_stmt
              | block
;

block: TOKEN_LEFT_BRACE statements TOKEN_RIGHT_BRACE
;

assignment: TOKEN_IDENTIFIER TOKEN_ASSIGN expr
        | TOKEN_IDENTIFIER TOKEN_LEFT_BRACKET expr TOKEN_RIGHT_BRACKET
TOKEN_ASSIGN expr
;

expr: or_expr
;

or_expr: and_expr
      | or_expr TOKEN_LOGICAL_OR and_expr
;

and_expr: rel_expr
       | and_expr TOKEN_LOGICAL_AND rel_expr
;
```

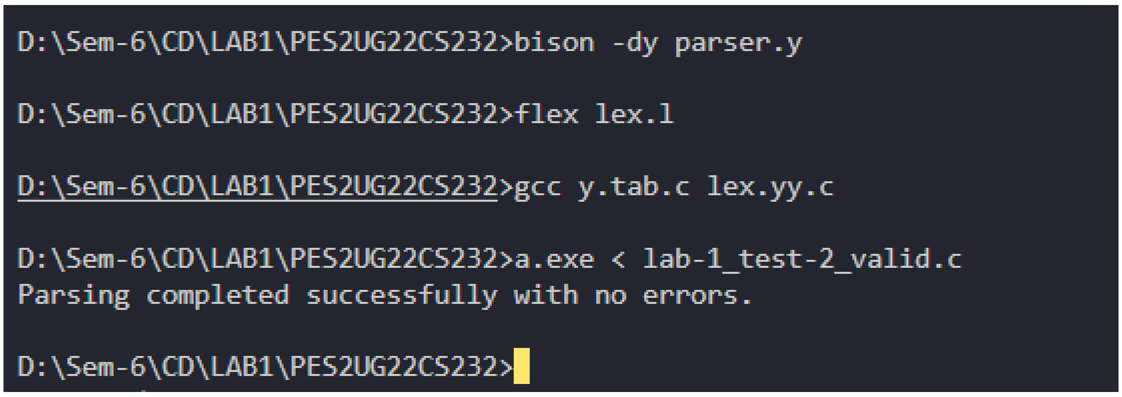
```
rel_expr: add_expr
        | rel_expr relational_op add_expr
;

add_expr: mult_expr
        | add_expr TOKEN_PLUS mult_expr
        | add_expr TOKEN_MINUS mult_expr
;

mult_expr: unary_expr
         | mult_expr TOKEN_MULTIPLY unary_expr
         | mult_expr TOKEN_DIVIDE unary_expr
         | mult_expr TOKEN_MODULO unary_expr
;

unary_expr: primary_expr
         | TOKEN_LOGICAL_NOT unary_expr
;

primary_expr: TOKEN_NUMBER
           | TOKEN_CHAR_LITERAL
           | TOKEN_IDENTIFIER
           | TOKEN_IDENTIFIER TOKEN_LEFT_BRACKET expr TOKEN_RIGHT_BRACKET
           | TOKEN_LEFT_PAREN expr TOKEN_RIGHT_PAREN
;

relational_op: TOKEN_EQUAL
```

```
            | TOKEN_NOT_EQUAL

            | TOKEN_LESS_THAN

            | TOKEN_LESS_EQUAL

            | TOKEN_GREATER_THAN

            | TOKEN_GREATER_EQUAL

;


conditional: if_stmt

        | if_else_stmt

;


if_stmt: TOKEN_IF TOKEN_LEFT_PAREN expr TOKEN_RIGHT_PAREN statement %prec
LOWER_THAN_ELSE

;


if_else_stmt: TOKEN_IF TOKEN_LEFT_PAREN expr TOKEN_RIGHT_PAREN statement
TOKEN_ELSE statement

;


loop: while_loop

   | for_loop

   | do_while_loop

;


while_loop: TOKEN_WHILE TOKEN_LEFT_PAREN expr TOKEN_RIGHT_PAREN statement

;


for_loop: TOKEN_FOR TOKEN_LEFT_PAREN for_init TOKEN_SEMICOLON expr
TOKEN_SEMICOLON assignment TOKEN_RIGHT_PAREN statement
```

```
;

for_init: /* empty */
        | assignment
        | declaration
;

do_while_loop: TOKEN_DO statement TOKEN_WHILE TOKEN_LEFT_PAREN expr
TOKEN_RIGHT_PAREN TOKEN_SEMICOLON
;

switch_stmt: TOKEN_SWITCH TOKEN_LEFT_PAREN expr TOKEN_RIGHT_PAREN
TOKEN_LEFT_BRACE case_statements TOKEN_RIGHT_BRACE
;

case_statements: /* empty */
               | case_statements TOKEN_CASE TOKEN_NUMBER TOKEN_COLON statements
               | case_statements TOKEN_DEFAULT TOKEN_COLON statements
;

return_stmt: TOKEN_RETURN expr TOKEN_SEMICOLON
;
%%

void yyerror(const char *s) {
    error_count++;
    fprintf(stderr, "Error at line %d: %s, unexpected '%s'\n", yylineno, s, yytext);
}
```

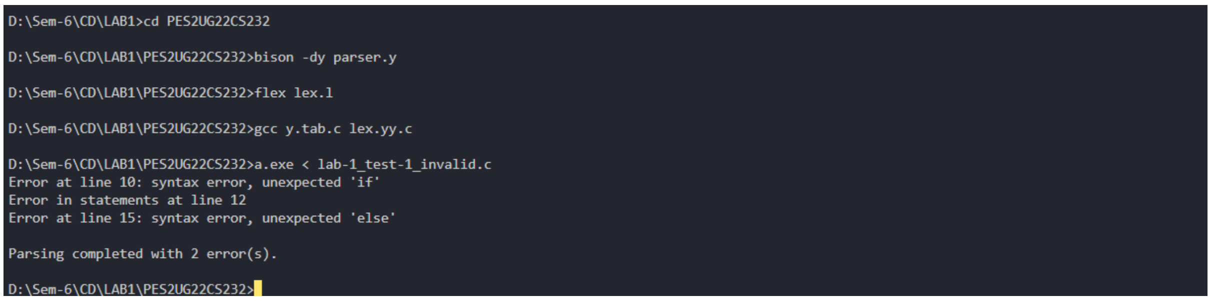
```
int main(void) {
    int result = yyparse();
    if (error_count > 0) {
        printf("\nParsing completed with %d error(s).\n", error_count);
        return 1;
    }
    printf("Parsing completed successfully with no errors.\n");
    return 0;
}
```

OUTPUT SCREENSHOT (VALID):



```
D:\Sem-6\CD\LAB1\PES2UG22CS232>bison -dy parser.y

D:\Sem-6\CD\LAB1\PES2UG22CS232>flex lex.l

D:\Sem-6\CD\LAB1\PES2UG22CS232>gcc y.tab.c lex.yy.c

D:\Sem-6\CD\LAB1\PES2UG22CS232>a.exe < lab-1_test-2_valid.c
Parsing completed successfully with no errors.

D:\Sem-6\CD\LAB1\PES2UG22CS232>
```

OUTPUT SCREENSHOT (INVALID):



```
D:\Sem-6\CD\LAB1>cd PES2UG22CS232

D:\Sem-6\CD\LAB1\PES2UG22CS232>bison -dy parser.y

D:\Sem-6\CD\LAB1\PES2UG22CS232>flex lex.l

D:\Sem-6\CD\LAB1\PES2UG22CS232>gcc y.tab.c lex.yy.c

D:\Sem-6\CD\LAB1\PES2UG22CS232>a.exe < lab-1_test-1_invalid.c
Error at line 10: syntax error, unexpected 'if'
Error in statements at line 12
Error at line 15: syntax error, unexpected 'else'

Parsing completed with 2 error(s).

D:\Sem-6\CD\LAB1\PES2UG22CS232>
```