# 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;

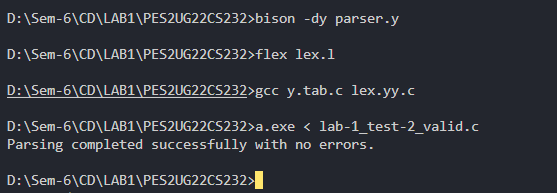
}

printf("Parsing completed successfully with no errors.\n");

return 0;

}

**OUTPUT SCREENSHOT (VALID):**



**OUTPUT SCREENSHOT (INVALID):**

