Github Link: <https://github.com/916RasnitaRadu/SEM5-FLCD/tree/main/Lab8>

**Lang.y specification file:**

%{

#include <stdio.h>

#include <stdlib.h>

int yyerror(char \*s);

#define YYDEBUG 1

%}

%token START;

%token INT;

%token STR;

%token CHAR;

%token READ;

%token IF;

%token ELSE;

%token PRINT;

%token WHILE;

%token ARR;

%token PLUS;

%token MINUS;

%token TIMES;

%token DIV;

%token LESS;

%token LESS\_EQ;

%token EQ;

%token NEQ;

%token BIGGER\_EQ;

%token EQQ;

%token BIGGER;

%token SQRT;

%token SQ\_BRACKET\_OPEN;

%token SQ\_BRACKET\_CLOSE;

%token SEMICOLON;

%token OPEN;

%token CLOSE;

%token BRACKET\_OPEN;

%token BRACKET\_CLOSE;

%token COMMA;

%token ID;

%token INT\_CONSTANT;

%token STRING\_CONSTANT;

%start Program

%%

Program : START BRACKET\_OPEN CompoundStatement BRACKET\_CLOSE { printf("PARSER: Program -> start { CompoundStatement }\n"); }

;

CompoundStatement : Statement SEMICOLON CompoundStatement { printf("PARSER: CompoundStatement -> Statement ; CompoundStatement\n"); }

| Statement SEMICOLON { printf("PARSER: CompoundStatement -> Statement ;\n"); }

;

Statement : DeclarationStatement { printf("PARSER: Statement -> DeclarationStatement\n"); }

| AssignmentStatement { printf("PARSER: Statement -> AssignmentStatement\n"); }

| IfStatement { printf("PARSER: Statement -> IfStatement\n"); }

| WhileStatement { printf("PARSER: Statement -> WhileStatement\n"); }

| PrintStatement { printf("PARSER: Statement -> PrintStatement\n"); }

| ReadStatement { printf("PARSER: Statement -> ReadStatement\n"); }

;

DeclarationStatement : ID Type COMMA DeclarationStatement { printf("PARSER: DeclarationStatement -> ID ( Type ) , DeclarationStatement\n"); }

| ID Type { printf("PARSER: DeclarationStatement -> ID ( Type )\n"); }

;

Type : INT { printf("PARSER: Type -> int\n"); }

| STR { printf("PARSER: Type -> str\n"); }

| CHAR { printf("PARSER: Type -> char\n"); }

| ARR { printf("PARSER: Type -> arr\n"); }

;

AssignmentStatement : ID EQ Expression { printf("PARSER: AssignmentStatement -> ID = Expression\n"); }

| ID EQ ArrayStatement { printf("PARSER: AssignmentStatement -> ID = ArrayStatement\n"); }

;

Expression : Expression PLUS Term { printf("PARSER: Expression -> Expression + Term\n"); }

| Expression MINUS Term { printf("PARSER: Expression -> Expression - Term\n"); }

| Term { printf("PARSER: Expression -> Term\n"); }

;

Term : Term TIMES Factor { printf("PARSER: Term -> Term \* Factor\n"); }

| Term DIV Factor { printf("PARSER: Term -> Term / Factor\n"); }

| Factor { printf("Term -> Factor\n"); }

;

Factor : OPEN Expression CLOSE { printf("PARSER: Factor -> ( Expression )\n"); }

| ID { printf("PARSER: Factor -> ID\n"); }

| INT\_CONSTANT { printf("PARSER: Factor -> INT\_CONSTANT\n"); }

| MINUS ID { printf("PARSER: Factor -> - ID\n"); }

| SQRT OPEN Expression CLOSE { printf("PARSER: Factor -> sqrt ( Expression )\n"); }

;

ArrayStatement : SQ\_BRACKET\_OPEN SQ\_BRACKET\_CLOSE { printf("PARSER: ArrayStatement -> []\n"); }

| SQ\_BRACKET\_OPEN ExpressionList SQ\_BRACKET\_CLOSE { printf("PARSER: ArrayStatement -> [ ExpressionList ]\n"); }

;

ExpressionList : Expression COMMA ExpressionList { printf("PARSER: ExpressionList -> Expression , ExpressionList\n"); }

| Expression { printf("PARSER: ExpressionList -> Expression\n"); }

;

IfStatement : IF Condition BRACKET\_OPEN CompoundStatement BRACKET\_CLOSE { printf("PARSER: IfStatement -> if Expression { CompoundStatement }\n"); }

| IF Condition BRACKET\_OPEN CompoundStatement BRACKET\_CLOSE ELSE BRACKET\_OPEN CompoundStatement BRACKET\_CLOSE { printf("PARSER: IfStatement -> if Expression { CompoundStatement } else { CompoundStatement }\n"); }

;

WhileStatement : WHILE Condition BRACKET\_OPEN CompoundStatement BRACKET\_CLOSE { printf("PARSER: WhileStatement -> while Expression { CompoundStatement }\n"); }

;

PrintStatement : PRINT OPEN Expression CLOSE { printf("PARSER: PrintStatement -> print ( Expression )\n"); }

| PRINT OPEN STRING\_CONSTANT CLOSE { printf("PARSER: PrintStatement -> print ( STRING\_CONSTANT )\n"); }

;

ReadStatement : READ OPEN ID CLOSE { printf("PARSER: ReadStatement -> read ( ID )\n"); }

;

Condition : Expression Relation Expression { printf("PARSER: Condition -> Expression Relation Expression\n"); }

;

Relation : LESS { printf("PARSER: Relation -> <\n"); }

| LESS\_EQ { printf("PARSER: Relation -> <=\n"); }

| EQQ { printf("PARSER: Relation -> ==\n"); }

| NEQ { printf("PARSER: Relation -> <>\n"); }

| BIGGER\_EQ { printf("PARSER: Relation -> >=\n"); }

| BIGGER { printf("PARSER: Relation -> >\n"); }

;

%%

int yyerror(char \*s) {

printf("PARSER: Error: %s", s);

}

extern FILE \*yyin;

int main(int argc, char\*\* argv) {

if (argc > 1)

yyin = fopen(argv[1], "r");

if (!yyparse())

fprintf(stderr, "\tOK\n");

}

Commands run:

bison -d lang.y

flex lang.lxi

gcc –o parser lex.yy.c lang.tab.c -lfl

./parser p.vtm