Lab 9 documentation - https://github.com/Lois-Beltechi/Formal-Languges-and-Compiler-Design/tree/main/lab9

Parser.y

```
%{

#include <stdio.h>

#include <stdlib.h>


int yyerror(char *s);


#define YYDEBUG 1

%}


%token PROG;

%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 LESSEQ;

%token EQ;

%token NEQ;

%token BIGGEREQ;

%token EQQ;

%token BIGGER;

%token SQRT;


%token SQBRACKETOPEN;

%token SQBRACKETCLOSE;

%token SEMICOLON;

%token OPEN;

%token CLOSE;

%token BRACKETOPEN;

%token BRACKETCLOSE;

%token COMMA;


%token IDENTIFIER;

%token INTCONSTANT;

%token STRINGCONSTANT;

%token CHARCONSTANT;


%start Program


%%

Program : PROG BRACKETOPEN CompoundStatement BRACKETCLOSE     { printf("Program -> prog {
CompoundStatement }\n"); }

        ;
```

```
CompoundStatement : Statement SEMICOLON CompoundStatement     { printf("CompoundStatement -> 
Statement ; CompoundStatement\n"); }
                | Statement SEMICOLON                    { printf("CompoundStatement -> Statement ;\n"); }
                ;
Statement : DeclarationStatement     { printf("Statement -> DeclarationStatement\n"); }
         | AssignmentStatement     { printf("Statement -> AssignmentStatement\n"); }
         | IfStatement     { printf("Statement -> IfStatement\n"); }
         | WhileStatement     { printf("Statement -> WhileStatement\n"); }
         | PrintStatement     { printf("Statement -> PrintStatement\n"); }
         | ReadStatement     { printf("Statement -> ReadStatement\n"); }
         ;
DeclarationStatement : IDENTIFIER OPEN Type CLOSE COMMA DeclarationStatement { 
printf("DeclarationStatement -> IDENTIFIER ( Type ) , DeclarationStatement\n"); }
                | IDENTIFIER OPEN Type CLOSE     { printf("DeclarationStatement -> IDENTIFIER ( Type )\n"); }
                | SEMICOLON {printf("Empty DeclarationStatement\n");}
                ;
Type : INT     { printf("Type -> int\n"); }
    | STR     { printf("Type -> str\n"); }
    | CHAR     { printf("Type -> char\n"); }
    | ARR     { printf("Type -> arr\n"); }
    ;
AssignmentStatement : IDENTIFIER EQ Expression     { printf("AssignmentStatement -> IDENTIFIER = 
Expression\n"); }
                 | IDENTIFIER EQ ArrayStatement     { printf("AssignmentStatement -> IDENTIFIER = 
ArrayStatement\n"); }
                 ;
Expression : Expression PLUS Term     { printf("Expression -> Expression + Term\n"); }
          | Expression MINUS Term     { printf("Expression -> Expression - Term\n"); }
          | Term     { printf("Expression -> Term\n"); }
          ;
```

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

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

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

    ;

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

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

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

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

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

       ;

ArrayStatement : SQBRACKETOPEN SQBRACKETCLOSE    { printf("ArrayStatement -> []\n"); }

              | SQBRACKETOPEN ExpressionList SQBRACKETCLOSE    { printf("ArrayStatement -> [
ExpressionList ]\n"); }

              ;

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

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

              ;

IfStatement : IF Condition BRACKETOPEN CompoundStatement BRACKETCLOSE  { printf("IfStatement -> if
Expression { CompoundStatement }\n"); }

            | IF Condition BRACKETOPEN CompoundStatement BRACKETCLOSE ELSE BRACKETOPEN
CompoundStatement BRACKETCLOSE  { printf("IfStatement -> if Expression { CompoundStatement } else
{ CompoundStatement }\n"); }

            ;

WhileStatement : WHILE Condition BRACKETOPEN CompoundStatement BRACKETCLOSE  {
printf("WhileStatement -> while Expression { CompoundStatement }\n"); }

              ;

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

              | PRINT OPEN STRINGCONSTANT CLOSE     { printf("PrintStatement -> print ( STRINGCONSTANT
)\n"); }

              | PRINT OPEN CHARCONSTANT CLOSE { printf ("PrintStatement" -> print ( STRINGCONSTANT
)\n");}
```

```
            ;
ReadStatement : READ OPEN IDENTIFIER CLOSE     { printf("ReadStatement -> read ( IDENTIFIER )\n"); }
            ;
Condition : Expression Relation Expression     { printf("Condition -> Expression Relation Expression\n"); }
        ;
Relation : LESS     { printf("Relation -> <\n"); }
        | LESSEQ     { printf("Relation -> <=\n"); }
        | EQQ     { printf("Relation -> ==\n"); }
        | NEQ     { printf("Relation -> <>\n"); }
        | BIGGEREQ     { printf("Relation -> >=\n"); }
        | BIGGER     { printf("Relation -> >\n"); }
        ;
%%
int yyerror(char *s) {
   printf("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");
}
```