LINK TO GIT:

Lex.y file

```
%{
#include <stdio.h>
#include <stdlib.h>

#define YYDEBUG 1
%}

%token INT
%token BOOL
%token STRING
%token CHARACTER
%token IF
%token ELSE
%token WHILE
%token FUNCTION
%token READ
%token WRITE
%token TRUE
%token FALSE
%token FOR
%token LENGTH
%token IDENTIFIER
%token CONSTANT
%token COLON
%token SEMI_COLON
%token COMA
%token DOT
%token PLUS
%token MINUS
%token MULTIPLY
%token DIVISION
%token FLOOR_DIVISION
%token MODULO
%token LEFT_ROUND_BRACKETS
%token RIGHT_ROUND_BRACKETS
%token LEFT_SQUARE_BRACKETS
%token RIGHT_SQUARE_BRACKETS
%token LEFT_CURLY_BRACKETS
%token RIGHT_CURLY_BRACKETS
%token QUESTION_MARK
%token LESS_THAN
%token GREATER_THAN
%token LESS_OR_EQUAL_THAN
%token GREATER_OR_EQUAL_THAN
%token DIFFERENT
%token EQUAL
%token ASSIGNMENT
```

```
%token AND_OPERATOR
%token OR_OPERATOR

%start program

%%

program : FUNCTION LEFT_CURLY_BRACKETS declarationList compoundStatement
RIGHT_CURLY_BRACKETS ;
declarationList : declaration SEMI_COLON declarationList | declaration ;
declaration : type IDENTIFIER ;
type : primitiveType | primitiveType arrayDeclaration ;
primitiveType : BOOL | CHARACTER | INT | STRING ;
arrayDeclaration : LEFT_SQUARE_BRACKETS CONSTANT RIGHT_SQUARE_BRACKETS ;
compoundStatement : LEFT_CURLY_BRACKETS statementList RIGHT_CURLY_BRACKETS ;
statementList : statement SEMI_COLON statementList | statement ;
statement : simpleStatement | structStatement ;
simpleStatement : assignmentStatement | ioStatement ;
assignmentStatement : IDENTIFIER ASSIGNMENT expression ;
expression : term | term PLUS expression | term MINUS expression | term MULTIPLY expression |
term DIVISION expression | term FLOOR_DIVISION expression | term MODULO expression |
LEFT_ROUND_BRACKETS expression RIGHT_SQUARE_BRACKETS ;
term : IDENTIFIER | CONSTANT ;
ioStatement : READ type IDENTIFIER | WRITE IDENTIFIER | WRITE CONSTANT ;
structStatement : ifStatement | whileStatement | forStatement ;
ifStatement : IF LEFT_ROUND_BRACKETS conditionList RIGHT_ROUND_BRACKETS
LEFT_CURLY_BRACKETS statementList RIGHT_CURLY_BRACKETS | IF
LEFT_ROUND_BRACKETS conditionList RIGHT_ROUND_BRACKETS
LEFT_CURLY_BRACKETS statementList RIGHT_CURLY_BRACKETS ELSE
LEFT_CURLY_BRACKETS statementList RIGHT_CURLY_BRACKETS ;
whileStatement : WHILE LEFT_ROUND_BRACKETS conditionList
RIGHT_ROUND_BRACKETS LEFT_CURLY_BRACKETS statementList
RIGHT_CURLY_BRACKETS;
forStatement : FOR LEFT_ROUND_BRACKETS declaration SEMI_COLON  expression
SEMI_COLON IDENTIFIER RIGHT_ROUND_BRACKETS LEFT_CURLY_BRACKETS
statementList RIGHT_CURLY_BRACKETS ;
conditionList: condition | condition AND_OPERATOR condition | condition OR_OPERATOR
condition ;
condition : expression relation expression ;
relation : LESS_THAN | GREATER_THAN | LESS_OR_EQUAL_THAN |
GREATER_OR_EQUAL_THAN | DIFFERENT | EQUAL ;
length : LENGTH LEFT_ROUND_BRACKETS IDENTIFIER RIGHT_ROUND_BRACKETS

%%

yyerror(char *s)
{
  printf("%s\n", s);
}

extern FILE *yyin;
```

```
main(int argc, char **argv)
{
  if (argc > 1)
    yyin = fopen(argv[1], "r");
  if ( (argc > 2) && ( !strcmp(argv[2], "-d") ) )
    yydebug = 1;
  if ( !yyparse() )
    fprintf(stderr,"\t It seems that you do not have any errors: good job :) \n");
}
```

lex.lxi file

```
%{
#include <math.h>
#include <stdio.h>
#include "y.tab.h"
int lines = 0;
%}
%option noyywrap

DIGIT           [0-9]
NUMBER          [1-9][0-9]*|0
STRING          ["]([a-zA-Z])*["]
CONSTANT    {STRING}|{NUMBER}
ID            [a-zA-Z][a-zA-Z0-9]{0,10}

%%

"int"      {printf( "Reserved word: %s\n", yytext );  return INT;}
"bool"      {printf( "Reserved word: %s\n", yytext );  return BOOL;}
"string"        {printf( "Reserved word: %s\n", yytext );  return STRING;}
"character"     {printf( "Reserved word: %s\n", yytext );  return CHARACTER;}
"if"        {printf( "Reserved word: %s\n", yytext );  return IF;}
"while"             {printf( "Reserved word: %s\n", yytext );  return WHILE;}
"else"      {printf( "Reserved word: %s\n", yytext );  return ELSE;}
"function"      {printf( "Reserved word: %s\n", yytext );  return FUNCTION;}
"read"      {printf( "Reserved word: %s\n", yytext );  return READ;}
"write"     {printf( "Reserved word: %s\n", yytext );  return WRITE;}
"true"      {printf( "Reserved word: %s\n", yytext );  return TRUE;}
"false"     {printf( "Reserved word: %s\n", yytext );  return FALSE;}
"for"       {printf( "Reserved word: %s\n", yytext );  return FOR;}
"length"        {printf( "Reserved word: %s\n", yytext );  return LENGTH;}
"&&"       {printf( "Operator: %s\n", yytext ); return AND_OPERATOR;}
"||"      {printf( "Operator: %s\n", yytext ); return OR_OPERATOR;}

{ID}       {printf( "Identifier: %s\n", yytext ); return IDENTIFIER;}
{CONSTANT}          {printf( "Constant: %s\n", yytext ); return CONSTANT;}

":"         {printf( "Separator: %s\n", yytext ); return COLON;}
";"         {printf( "Separator: %s\n", yytext ); return SEMI_COLON;}
","         {printf( "Separator: %s\n", yytext ); return COMA;}
```

```
"."         {printf( "Separator: %s\n", yytext ); return DOT;}
"+"         {printf( "Operator: %s\n", yytext ); return PLUS;}
"-"         {printf( "Operator: %s\n", yytext ); return MINUS;}
"*"         {printf( "Operator: %s\n", yytext ); return MULTIPLY;}
"/"         {printf( "Operator: %s\n", yytext ); return DIVISION;}
"//"   {printf( "Operator: %s\n", yytext ); return FLOOR_DIVISION;}
"%"    {printf( "Operator: %s\n", yytext ); return MODULO;}
"("         {printf( "Separator: %s\n", yytext ); return LEFT_ROUND_BRACKETS;}
")"         {printf( "Separator: %s\n", yytext ); return RIGHT_ROUND_BRACKETS;}
"["         {printf( "Separator: %s\n", yytext ); return LEFT_SQUARE_BRACKETS;}
"]"         {printf( "Separator: %s\n", yytext ); return RIGHT_SQUARE_BRACKETS;}
"{"    {printf( "Separator: %s\n", yytext ); return LEFT_CURLY_BRACKETS;}
"}"    {printf( "Separator: %s\n", yytext ); return RIGHT_CURLY_BRACKETS;}
"?"    {printf( "Separator: %s\n", yytext ); return QUESTION_MARK;}
"<"         {printf( "Operator: %s\n", yytext ); return LESS_THAN;}
">"         {printf( "Operator: %s\n", yytext ); return GREATER_THAN;}
"<="   {printf( "Operator: %s\n", yytext ); return LESS_OR_EQUAL_THAN;}
">="   {printf( "Operator: %s\n", yytext ); return GREATER_OR_EQUAL_THAN;}
"!="   {printf( "Operator: %s\n", yytext ); return DIFFERENT;}
"=="   {printf( "Operator: %s\n", yytext ); return EQUAL;}
"="         {printf( "Operator: %s\n", yytext ); return ASSIGNMENT;}

[ \t]+   /* remove spaces */   {}

[\n]+   {++lines;}

[a-zA-Z][a-zA-Z0-9]{11,}     {printf("Illegal size of the identifier at line %d\n", lines); return -1;}

[0-9][a-zA-Z0-9]{0,10}       {printf("Illegal identifier at line %d\n", lines); return -1;}

.         {printf("Illegal symbol at line\n"); return -1;}
%%
```