https://github.com/915-Nichifor-Dragos/FLCD/tree/master/Lab10

Lex Specification File

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lang.tab.h"
int currentLine = 1;
%}

%option noyywrap

IDENTIFIER          [a-zA-Z_][a-zA-Z0-9_]*
NUMBER_CONST        0|[+|-]?[1-9][0-9]*([.][0-9]*)?|[+|-]?0[.][0-9]*
STRING_CONST [\"][a-zA-Z0-9 ]+[\"]
CHAR_CONST          [\'][a-zA-Z0-9 ][\']

%%

"int"           {printf("Reserved word: %s\n", yytext); return INT;}
"string" {printf("Reserved word: %s\n", yytext); return STRING;}
"bool"          {printf("Reserved word: %s\n", yytext); return BOOL;}
"read"          {printf("Reserved word: %s\n", yytext); return READ;}
"write"         {printf("Reserved word: %s\n", yytext); return WRITE;}
"if"            {printf("Reserved word: %s\n", yytext); return IF;}
"else"          {printf("Reserved word: %s\n", yytext); return ELSE;}
"while"         {printf("Reserved word: %s\n", yytext); return WHILE;}
"return"        {printf("Reserved word: %s\n", yytext); return RETURN;}
"function"          {printf("Reserved word: %s\n", yytext); return FUNCTION;}
```

```
"list"          {printf("Reserved word: %s\n", yytext); return LIST;}
"true"          {printf("Reserved word: %s\n", yytext); return TRUE;}
"false"         {printf("Reserved word: %s\n", yytext); return FALSE;}
"and"           {printf("Reserved word: %s\n", yytext); return AND;}
"or"            {printf("Reserved word: %s\n", yytext); return OR;}

"+"             {printf("Operator %s\n", yytext); return plus;}
"-"             {printf("Operator %s\n", yytext); return  minus;}
"*"             {printf("Operator %s\n", yytext); return mul;}
"/"             {printf("Operator %s\n", yytext); return division;}
"%"             {printf("Operator %s\n", yytext); return mod;}
"<="            {printf("Operator %s\n", yytext); return lessOrEqual;}
">="            {printf("Operator %s\n", yytext); return moreOrEqual;}
"<"             {printf("Operator %s\n", yytext); return  less;}
">"             {printf("Operator %s\n", yytext); return more;}
"=="            {printf("Operator %s\n", yytext); return equal;}
"!="            {printf("Operator %s\n", yytext); return different;}
"="             {printf("Operator %s\n", yytext); return eq;}

"{"             {printf("Separator %s\n", yytext); return leftCurlyBracket;}
"}"             {printf("Separator %s\n", yytext); return rightCurlyBracket;}
"("             {printf("Separator %s\n", yytext); return leftRoundBracket;}
")"             {printf("Separator %s\n", yytext); return rightRoundBracket;}
"["             {printf("Separator %s\n", yytext); return leftBracket;}
"]"             {printf("Separator %s\n", yytext); return rightBracket;}
":"             {printf("Separator %s\n", yytext); return colon;}
";"             {printf("Separator %s\n", yytext); return semicolon;}
","             {printf("Separator %s\n", yytext); return comma;}
"'"             {printf("Separator %s\n", yytext); return apostrophe;}
```

```
"\""                {printf("Separator %s\n", yytext); return quote;}

{IDENTIFIER}                {printf("Identifier: %s\n", yytext); return IDENTIFIER;}
{NUMBER_CONST}                      {printf("Number: %s\n", yytext); return NUMBER_CONST;}
{STRING_CONST}                      {printf("String: %s\n", yytext); return STRING_CONST;}
{CHAR_CONST}            {printf("Character: %s\n", yytext); return CHAR_CONST;}

[ \t]+              {}
[\n]+     {currentLine++;}

[0-9][a-zA-Z0-9_]*                  {printf("Illegal identifier at line %d\n", currentLine);}
[+|-]0              {printf("Illegal numeric constant at line %d\n", currentLine);}
[+|-]?[0][0-9]*([.][0-9]*)?                     {printf("Illegal numeric constant at line %d\n", currentLine);}
[\'][a-zA-Z0-9 ]{2,}[\']|[\'][a-zA-Z0-9 ][a-zA-Z0-9 ][\']                   {printf("Illegal character constant at line
%d\n", currentLine);}
[\"][a-zA-Z0-9_]+|[a-zA-Z0-9_]+[\"]                 {printf("Illegal string constant at line %d\n",
currentLine);}

%%
```

YACC Specification File

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

#define YYDEBUG 1
%}

%token INT
%token STRING
%token BOOL
%token READ
%token WRITE
%token IF
%token ELSE
%token WHILE
%token RETURN
%token FUNCTION
%token LIST
%token TRUE
%token FALSE
%token AND
%token OR

%token plus
%token minus
%token mul
%token division
%token mod
```

```
%token lessOrEqual
%token moreOrEqual
%token less
%token more
%token equal
%token different
%token eq

%token leftCurlyBracket
%token rightCurlyBracket
%token leftRoundBracket
%token rightRoundBracket
%token leftBracket
%token rightBracket
%token colon
%token semicolon
%token comma
%token apostrophe
%token quote

%token IDENTIFIER
%token NUMBER_CONST
%token STRING_CONST
%token CHAR_CONST

%start function
```

%%

function : FUNCTION compound_statement

statement : declaration semicolon | assignment_statement | return_statement semicolon | iostmt semicolon | if_statement | while_statement

statement_list : statement | statement statement_list

compound_statement : leftCurlyBracket statement_list rightCurlyBracket

expression : expression plus term | expression minus term | term

term : term mul factor | term division factor | term mod factor | factor

factor : leftRoundBracket expression rightRoundBracket | IDENTIFIER | constant | bool_values | LIST | list_element

constant : NUMBER_CONST | STRING_CONST | CHAR_CONST

iostmt : READ leftRoundBracket IDENTIFIER rightRoundBracket | WRITE leftRoundBracket IDENTIFIER rightRoundBracket | WRITE leftRoundBracket constant rightRoundBracket

simple_type : INT | STRING | BOOL

array_declaration : LIST simple_type IDENTIFIER leftBracket rightBracket

declaration : simple_type IDENTIFIER | array_declaration

assignment_statement : simple_type IDENTIFIER eq expression semicolon | IDENTIFIER eq expression semicolon

if_statement : IF leftRoundBracket condition_list rightRoundBracket compound_statement | IF leftRoundBracket condition_list rightRoundBracket compound_statement ELSE compound_statement

while_statement : WHILE leftRoundBracket condition rightRoundBracket compound_statement

return_statement : RETURN expression

condition : expression relation expression

condition_list : condition | condition comparison_values condition_list

relation : less | lessOrEqual | equal | different | moreOrEqual | more

bool_values : TRUE | FALSE

comparison_values : AND | OR

list_element : IDENTIFIER leftBracket IDENTIFIER rightBracket

%%

```
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, "\tProgram is syntactically correct.\n");
	return 0;
}
```

Demo

Run the command in the directory:

```
PS C:\Users\Dragos\Desktop\FLCD\Lab10> flex lang.lxi
```

After the first command, run:

```
PS C:\Users\Dragos\Desktop\FLCD\Lab10> bison -d lang.y
```

After the second command, run:

```
PS C:\Users\Dragos\Desktop\FLCD\Lab10> gcc -o my_compiler lex.yy.c lang.tab.c
```

An executable (my_compiler.exe) was created after the third command, so we can now run the program.

We have 4 examples for which we can run the program (p1.txt, p2.txt, p3.txt and p1err.txt)

In this demo, I am going to run the program for p2.txt, using the following command:

```
PS C:\Users\Dragos\Desktop\FLCD\Lab10> ./my_compiler .\p2.txt
```

Output

```
Reserved word: function
Separator {
Reserved word: int
Identifier: x
Separator ;
Reserved word: bool
Identifier: isPrime
Operator =
Reserved word: true
Separator ;
Reserved word: read
Separator (
Identifier: x
Separator )
Separator ;
Reserved word: if
Separator (
Identifier: x
Operator <
Number: 2
Reserved word: or
Identifier: x
Operator >
Number: 2
Reserved word: and
Identifier: x
Operator %
Number: 2
Operator ==
Number: 0
Separator )
Separator {
Identifier: isPrime
Operator =
Reserved word: false
Separator ;
Separator }
Reserved word: int
Identifier: d
Operator =
Number: 3
Separator ;
Reserved word: while
Separator (
Identifier: d
Operator *
```

```
Identifier: d
Operator <=
Identifier: x
Separator )
Separator {
Reserved word: if
Separator (
Identifier: x
Operator %
Identifier: d
Operator ==
Number: 0
Separator )
Separator {
Identifier: isPrime
Operator =
Reserved word: false
Separator ;
Separator }
Identifier: d
Operator =
Identifier: d
Operator +
Number: 2
Separator ;
Separator }
Reserved word: write
Separator (
Identifier: isPrime
Separator )
Separator ;
Separator }
        Program is syntactically correct.
PS C:\Users\Dragos\Desktop\FLCD\Lab10>
```