https://github.com/911-Abrudan-Rebeca/FLCD/tree/main/Lab9

Laboratory9

I use the lang.lxi file from Laboratory8 but with small modifications. (it returns tokens)

Lang.lxi

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

%option noyywrap
%option caseless

DIGIT [0-9]
NON_ZERO_DIGIT [1-9]
INT_CONSTANT [+-]?{NON_ZERO_DIGIT}{DIGIT}*|0
LETTER [a-zA-Z_]
SPECIAL_CHAR [ _]
STRING_CONSTANT (\"({LETTER}|{DIGIT}|{SPECIAL_CHAR})*\")
IDENTIFIER (#|{LETTER})({LETTER}|{DIGIT})*

BAD_IDENTIFIER ({DIGIT})+({LETTER})+({LETTER}|{DIGIT})*
BAD_CONST [+-]0|0{NON_ZERO_DIGIT}{DIGIT}*

%%
```

```
"var"     { printf("reserved word: %s\n", yytext); return VAR; }
"arr"     { printf("reserved word: %s\n", yytext); return ARR; }
"int"     { printf("reserved word: %s\n", yytext); return INT; }
"str"     { printf("reserved word: %s\n", yytext); return STR; }
"read"    { printf("reserved word: %s\n", yytext); return READ; }
"if"      { printf("reserved word: %s\n", yytext); return IF; }
"else"    { printf("reserved word: %s\n", yytext); return ELSE; }
"do"      { printf("reserved word: %s\n", yytext); return DO; }
"while" { printf("reserved word: %s\n", yytext); return WHILE; }
"print" { printf("reserved word: %s\n", yytext); return PRINT; }

"+"       { printf("operator: %s\n", yytext); return PLUS; }
"-"       { printf("operator: %s\n", yytext); return MINUS; }
"*"       { printf("operator: %s\n", yytext); return TIMES; }
"/"       { printf("operator: %s\n", yytext); return DIV; }
"%"       { printf("operator: %s\n", yytext); return MOD; }
">="      { printf("operator: %s\n", yytext); return BIGGEREQ; }
"<="      { printf("operator: %s\n", yytext); return LESSEQ; }
">"       { printf("operator: %s\n", yytext); return BIGGER; }
"<"       { printf("operator: %s\n", yytext); return LESS; }
"=="      { printf("operator: %s\n", yytext); return EQQ; }
"="       { printf("operator: %s\n", yytext); return EQ; }
"!="      { printf("operator: %s\n", yytext); return NEQ; }

"["       { printf("separator: %s\n", yytext); return SQBRACKETOPEN; }
"]"       { printf("separator: %s\n", yytext); return SQBRACKETCLOSE; }
"("       { printf("separator: %s\n", yytext); return OPEN; }
")"       { printf("separator: %s\n", yytext); return CLOSE; }
```

```
"{"         { printf("separator: %s\n", yytext); return BRACKETOPEN; }
"}"         { printf("separator: %s\n", yytext); return BRACKETCLOSE; }
","         { printf("separator: %s\n", yytext); return COMMA; }
":"         { printf("separator: %s\n", yytext); return COLON; }
";"         { printf("separator: %s\n", yytext); return SEMICOLON; }


{IDENTIFIER}            { printf("identifier: %s\n", yytext); return IDENTIFIER; }
{BAD_IDENTIFIER}        { printf("Error (identifier) at token %s at line %d\n", yytext, lines); return -1; }
{INT_CONSTANT}          { printf("integer constant: %s\n", yytext); return INTCONSTANT; }
{BAD_CONST}             { printf("Error (constant) at token %s at line %d \n", yytext, lines); exit(1);}
{STRING_CONSTANT}       { printf("string constant: %s\n", yytext); return STRINGCONSTANT; }

[ \t]+ {}
"//"(.)*[\n]+ {++lines;}
[\n]+ {++lines;}
. {printf("Error at token %s at line %d\n", yytext, lines); exit(1);}
%%
```

The yacc file (lang.y) has the four main sections:

```
%{
        C declarations  (headers, libraries)
%}
        yacc declarations (the tokens from lang.lxi)
%%
        Grammar rules (the ones from Lab1b)
%%
         Additional C code (file opening + parsing)
```

The lang.y:

```
%{
extern int yylex(void);
#include "y.tab.h"
#include <stdio.h>
#include <stdlib.h>
#define YYDEBUG 1
int yyerror(const char *s);
%}
%token VAR;
%token ARR;
%token INT;
%token STR;
%token READ;
%token IF;
%token ELSE;
%token DO;
%token WHILE;
%token PRINT;

%token PLUS;
%token MINUS;
%token TIMES;
%token DIV;
%token MOD;
%token BIGGEREQ;
%token LESSEQ;
%token BIGGER;
%token LESS;
%token EQQ;
```

```
%token EQ;
%token NEQ;

%token SQBRACKETOPEN;
%token SQBRACKETCLOSE;
%token OPEN;
%token CLOSE;
%token BRACKETOPEN;
%token BRACKETCLOSE;
%token DOT;
%token COMMA;
%token COLON;
%token SEMICOLON;

%token IDENTIFIER;
%token INTCONSTANT;
%token STRINGCONSTANT;
%start program
%%
program : VAR decllist SEMICOLON cmpdstmt    { printf("program -> var decllist ; cmpdstmt\n"); }
          ;
decllist : declaration                       { printf("decllist -> declaration\n"); }
         | decllist SEMICOLON declaration    { printf("decllist -> decllist ; declaration\n"); }
         ;
declaration : IDENTIFIER COLON type          { printf("declaration -> IDENTIFIER : type\n"); }
              ;
type : type1                                 { printf("type -> type1\n"); }
     | arraydecl                             { printf("type -> arraydecl\n"); }
     ;
```

```
type1 : INT                                          { printf("type1 -> int\n"); }
      | STR                                          { printf("type1 -> str\n"); }
      ;
arraydecl : ARR OPEN type1 SQBRACKETOPEN INTCONSTANT SQBRACKETCLOSE CLOSE
          {printf("arraydecl -> arr ( type1 [ INTCONSTANT ] )\n"); }
          ;

cmpdstmt : BRACKETOPEN stmtlist BRACKETCLOSE         { printf("cmpdstmt -> {stmtlist}\n"); }

stmtlist : stmt SEMICOLON stmtlist                   { printf("stmtlist -> stmt ; stmtlist\n"); }
         | stmt SEMICOLON                            { printf("stmtlist -> stmt ; \n"); }
         ;
stmt : simplstmt                                     { printf("stmt -> simplstmt\n"); }
     | structstmt                                    { printf("stmt -> structstmt\n"); }
     ;
simplstmt : assignstmt                               { printf("simplstmt -> assignstmt\n"); }
          | iostmt                                   { printf("simplstmt -> iostmt\n"); }
          ;
assignstmt : IDENTIFIER EQ expression    { printf("assignstmt -> IDENTIFIER = expression\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"); }
      ;

iostmt : READ OPEN IDENTIFIER CLOSE             { printf("iostmt -> read ( IDENTIFIER )\n"); }
      | PRINT OPEN IDENTIFIER CLOSE             { printf("iostmt -> print ( IDENTIFIER )\n"); }
      | PRINT OPEN STRINGCONSTANT CLOSE         { printf("iostmt -> print ( STRINGCONSTANT )\n"); }
      | PRINT OPEN INTCONSTANT CLOSE            { printf("iostmt -> print ( INTCONSTANT )\n"); }
      ;
structstmt : ifstmt                 { printf("structstmt -> ifstmt\n"); }
         | whilestmt                { printf("structstmt -> whilestmt\n"); }
         ;

ifstmt : IF OPEN condition CLOSE cmpdstmt        { printf("ifstmt -> if ( condition ) cmpd \n"); }
      | IF OPEN condition CLOSE cmpdstmt SQBRACKETOPEN ELSE cmpdstmt SQBRACKETCLOSE
{printf("ifstmt -> if ( condition ) cmpd [else cmpd]\n"); }
      ;
whilestmt : WHILE OPEN condition CLOSE cmpdstmt   {printf("whilestmt-> while ( condition ) {stmt}\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"); }
         | BIGGER         { printf("RELATION -> >\n"); }
         | BIGGEREQ       { printf("RELATION -> >=\n"); }
         ;
```

```
%%
int yyerror(const char *s) {
   printf("error: %s\n",s);
   return 0;
}
extern FILE *yyin;
int main(int argc, char** argv) {
   if (argc > 1)
      yyin = fopen(argv[1], "r");
   if (!yyparse())
      fprintf(stderr, "\tOK\n");
}
```

How to run:

1. flex lang.lxi
2. bison -dy lang.y
3. gcc lex.yy.c y.tab.c
4. a.exe p1.txt

The yacc returns the productions