https://github.com/hognogicristina/FLCD/tree/main/Lab9

Yacc file

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

#define YYDEBUG 1

int yyerror(const char *s);
%}

%token PROG;
%token INT;
%token REAL;
%token STR;
%token CHAR;
%token BOOL;
%token READ;
%token IF;
%token ELSE;
%token WRITE;
%token WHILE;
%token ARR;
%token SYS;
%token AND;
%token OR;
%token RAD;

%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 END_BLOCK;
%token BEGIN_BLOCK;
%token ENDL;

%token IDENTIFIER;
%token INTCONSTANT;
```

```
%token STRINGCONSTANT;


%start program

%%
program : PROG BRACKETOPEN stmtlist BRACKETCLOSE   { printf("program ->
prog { stmtlist }\n"); }
        ;

stmtlist : stmt              { printf("stmtlist -> stmt\n"); }
         | stmt stmtlist     { printf("stmtlist -> stmt stmtlist\n"); }
         ;

stmt : simplstmt            { printf("stmt -> simplstmt\n"); }
     | structstmt           { printf("stmt -> structstmt\n"); }
     ;

simplstmt : declaration           { printf("stmt -> declaration\n"); }
          | assignstmt            { printf("stmt -> assignstmt\n"); }
          | iostmt                { printf("stmt -> iostmt\n"); }
          | radstmt               { printf("stmt -> radstmt\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"); }
      | REAL               { printf("type1 -> real\n"); }
      | STR                { printf("type1 -> str\n"); }
      | CHAR               { printf("type1 -> char\n"); }
      | BOOL               { printf("type1 -> bool\n"); }
      ;

arraydecl : ARR OPEN type1 CLOSE SQBRACKETOPEN INTCONSTANT SQBRACKETCLOSE {
printf("arraydecl -> arr ( type1 ) [ INTCONSTANT ]\n"); }
          ;

assignstmt : IDENTIFIER EQ expression { printf("assignstmt -> IDENTIFIER =
expression\n"); }
           ;

operator : PLUS              { printf("operator -> +\n"); }
         | MINUS             { printf("operator -> -\n"); }
         | TIMES             { printf("operator -> *\n"); }
         | DIV               { printf("operator -> /\n"); }
         | MOD               { printf("operator -> %%\n"); }
         ;

expression : term                        { printf("expression -> term\n");
}
           | term operator expression    { printf("expression -> term
operator expression\n"); }
           ;

term : IDENTIFIER            { printf("term -> IDENTIFIER\n"); }
```

```
     | INTCONSTANT           { printf("term -> INTCONSTANT\n"); }
     | factor               { printf("term -> factor\n"); }
     ;

factor : MINUS IDENTIFIER           { printf("factor -> - IDENTIFIER\n");
}
       | radstmt                    { printf("factor -> radstmt\n"); }
       | IDENTIFIER SQBRACKETOPEN IDENTIFIER SQBRACKETCLOSE {
printf("factor -> IDENTIFIER [ IDENTIFIER ]\n"); }
       | IDENTIFIER SQBRACKETOPEN INTCONSTANT SQBRACKETCLOSE {
printf("factor -> IDENTIFIER [ INTCONSTANT ]\n"); }
       | OPEN expression CLOSE      { printf("factor -> ( expression
)\n"); }
       ;

iostmt : SYS DOT READ OPEN IDENTIFIER CLOSE       { printf("iostmt -> sys .
read ( IDENTIFIER )\n"); }
       | SYS DOT WRITE OPEN IDENTIFIER CLOSE      { printf("iostmt -> sys .
write ( IDENTIFIER )\n"); }
       | SYS DOT WRITE OPEN INTCONSTANT CLOSE     { printf("iostmt -> sys .
write ( INTCONSTANT )\n"); }
       | SYS DOT WRITE OPEN STRINGCONSTANT CLOSE  { printf("iostmt -> sys .
write ( STRINGCONSTANT )\n"); }
       | SYS DOT WRITE OPEN ENDL CLOSE            { printf("iostmt -> sys .
write ( endl )\n"); }
       ;

radstmt : RAD OPEN IDENTIFIER CLOSE               { printf("radstmt -> rad
( IDENTIFIER )\n"); }
        ;

structstmt : ifstmt           { printf("structstmt -> ifstmt\n"); }
           | whilestmt        { printf("structstmt -> whilestmt\n"); }
           ;

ifstmt : IF condition BEGIN_BLOCK COLON stmtlist END_BLOCK SEMICOLON
{ printf("ifstmt -> if condition begin : stmtlist end ;\n"); }
       | IF condition BEGIN_BLOCK COLON stmtlist ELSE BEGIN_BLOCK COLON
stmtlist END_BLOCK SEMICOLON { printf("ifstmt -> if condition begin :
stmtlist else begin : stmtlist end ;\n"); }
       ;

condition : expression RELATION expression                    {
printf("condition -> expression RELATION expression\n"); }
          | expression RELATION expression AND condition      {
printf("condition -> expression RELATION expression and condition\n"); }
          | expression RELATION expression OR condition       {
printf("condition -> expression RELATION expression or condition\n"); }
          ;

RELATION : BIGGEREQ           { printf("RELATION -> >=\n"); }
         | LESSEQ             { printf("RELATION -> <=\n"); }
         | BIGGER             { printf("RELATION -> >\n"); }
         | LESS               { printf("RELATION -> <\n"); }
         | EQQ                { printf("RELATION -> ==\n"); }
         | EQ                 { printf("RELATION -> =\n"); }
         | NEQ                { printf("RELATION -> !=\n"); }
         ;

whilestmt : WHILE condition BEGIN_BLOCK COLON stmtlist END_BLOCK SEMICOLON
{ printf("whilestmt -> while condition begin : stmtlist end ;\n"); }
```

```
          ;


%%

int yyerror(const char *s) {
    printf("%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");
}
```

Flex file for tokens

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lang.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_]
SIGNS [ !#%^*+-/<=>_.,:;]
STRING_CONSTANT (\"({LETTER}|{DIGIT}|_|{SIGNS})*\")
IDENTIFIER (#|{LETTER})({LETTER}|{DIGIT})*
BAD_IDENTIFIER ({DIGIT})+(#|{LETTER})+({LETTER}|{DIGIT})*

%%

"prog" { printf("reserved word: %s\n", yytext); return PROG; }
"int" { printf("reserved word: %s\n", yytext); return INT; }
"real" { printf("reserved word: %s\n", yytext); return REAL; }
"str" { printf("reserved word: %s\n", yytext); return STR; }
"char" { printf("reserved word: %s\n", yytext); return CHAR; }
"bool" { printf("reserved word: %s\n", yytext); return BOOL; }
"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; }
"write" { printf("reserved word: %s\n", yytext); return WRITE; }
"while" { printf("reserved word: %s\n", yytext); return WHILE; }
"arr" { printf("reserved word: %s\n", yytext); return ARR; }
"sys" { printf("reserved word: %s\n", yytext); return SYS; }
"and" { printf("reserved word: %s\n", yytext); return AND; }
"or" { printf("reserved word: %s\n", yytext); return OR; }
"rad" { printf("reserved word: %s\n", yytext); return RAD; }
```

```
"+" { 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 DOT; }
"," { printf("separator: %s\n", yytext); return COMMA; }
":" { printf("separator: %s\n", yytext); return COLON; }
";" { printf("separator: %s\n", yytext); return SEMICOLON; }
"end" { printf("separator: %s\n", yytext); return END_BLOCK; }
"begin" { printf("separator: %s\n", yytext); return BEGIN_BLOCK; }
"endl" { printf("separator: %s\n", yytext); return ENDL; }

{IDENTIFIER} { printf("identifier: %s\n", yytext); return IDENTIFIER; }

{BAD_IDENTIFIER} { printf("Error at token %s at line %d\n", yytext, lines);
return -1; }

{INT_CONSTANT} { printf("integer constant: %s\n", yytext); return
INTCONSTANT; }

{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);}

%%
```

Demo

1. Install bison on MacOS:

```
hognogicristina@Cristinas-MacBook-Air Laboratory 9 % brew install bison
```

2. Compile the bison file:

```
hognogicristina@Cristinas-MacBook-Air Laboratory 9 % bison -d lang.y
```

3. Generate the Lexer Code:

```
hognogicristina@Cristinas-MacBook-Air Laboratory 9 % flex -o lexer.c scanner.lxi
```

4. Compile the Generated C Code:

```
hognogicristina@Cristinas-MacBook-Air Laboratory 9 % gcc -o lang lang.tab.c lexer.c -L/opt/homebrew/opt/flex/lib -lfl
```

5. Run the Bison:

```
hognogicristina@Cristinas-MacBook-Air Laboratory 9 % ./lang p1.txt
```

Output:
https://github.com/hognogicristina/FLCD/blob/main/Lab9/output.txt

This yacc implementation is based on my syntax.in from L1b (https://github.com/hognogicristina/FLCD/blob/main/Lab1b/Syntax.in) and is also based on error handling and the lex file returns tokens and yacc uses it to return string of productions.