GitHub link: https://github.com/edwardinio18/LFTC/blob/main/Labs/Lab9

Yacc file

```
%token SPATIU;
%token SARILINIA;
%token GATALINIA;
%token PUNCTVIRGULA;
%token DOUAPUNCTE;
%token PUNCT;
%token VIRGULA;
%token PARANTEZADESCHISA;
%token PARANTEZAINCHISA;
%token ACOLADADESCHISA;
%token ACOLADAINCHISA;
%token PARANTEZAPATRATADESCHISA;
%token PARANTEZAPATRATAINCHISA;
%token LINIESUBTIRE;

%token IDENTIFIER;
%token INTCONSTANT;
%token STRINGCONSTANT;

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

#define YYDEBUG 1

int yyerror(const char *s);
%}

%token PROG;
%token CVNOU;
%token DACA;
%token ALTFEL;
%token CATTIMP;
%token FA;
%token PENTRU;
%token PENTRUFIECARE;
%token OPRESTETE;
%token CONTINUA;
%token SCRIE;
%token CITESTE;
%token IN;
%token DE;
%token ATUNCI;
%token INCEPE;
%token TERMINA;
%token INTREG;
%token REAL;
%token SFOARA;
%token SIR;
%token CONSTANT;
%token FUNCTIE;
%token INTOARCE;
%token ADVFALS;
%token CARACTER;
%token RADICAL;
%token SISI;
```

```
%token SAUSAU;
%token NU;

%token ADUNATE;
%token STERGETE;
%token ORIORI;
%token ORIORIINVERS;
%token ESTIEGAL;
%token VERIFICAEGAL;
%token VERIFICANUEGAL;
%token MAIMARE;
%token MAIMIC;
%token MAIMAREEGAL;
%token MAIMICEGAL;
%token LASUTA;

%start program

%%

program : PROG ACOLADADESCHISA stmtlist ACOLADAINCHISA { printf("program ->
prog { stmtlist }\n"); }
        ;

stmtlist : stmtlist stmt PUNCTVIRGULA  { printf("stmtlist -> stmtlist stmt
;\n"); }
         | stmtlist stmt               { printf("stmtlist -> stmtlist
stmt\n"); }
         | /* empty */                 { printf("stmtlist -> empty\n"); }
         ;

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

simplstmt : declaration     { printf("simplstmt -> declaration\n"); }
          | assignstmt      { printf("simplstmt -> assignstmt\n"); }
          | iostmt          { printf("simplstmt -> iostmt\n"); }
          | radstmt         { printf("simplstmt -> radstmt\n"); }
          ;

declaration : CVNOU IDENTIFIER DOUAPUNCTE type { printf("declaration ->
cvnou identifier : type\n"); }
            ;

type : type1               { printf("type -> type1\n"); }
     | arraydecl           { printf("type -> arraydecl\n"); }
     ;

type1 : INTREG             { printf("type1 -> intreg\n"); }
      | REAL               { printf("type1 -> real\n"); }
      | SFOARA             { printf("type1 -> sfoara\n"); }
      | CARACTER           { printf("type1 -> caracter\n"); }
      | ADVFALS            { printf("type1 -> advfals\n"); }
      ;

arraydecl : SIR PARANTEZADESCHISA INTCONSTANT PARANTEZAINCHISA DE type1 {
printf("arraydecl -> sir parantezadeschisa intconstant parantezainchisa de
type1\n"); }
          ;
```

```
assignstmt : IDENTIFIER ESTIEGAL expression { printf("assignstmt ->
identifier estiegal expression\n"); }
           ;

operator : ADUNATE           { printf("operator -> adunate\n"); }
         | STERGETE          { printf("operator -> stergete\n"); }
         | ORIORI            { printf("operator -> oriori\n"); }
         | ORIORIINVERS      { printf("operator -> orioriinvers\n"); }
         | LASUTA            { printf("operator -> lasuta\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 : STERGETE IDENTIFIER
{ printf("factor -> stergete identifier\n"); }
       | STERGETE INTCONSTANT
{ printf("factor -> stergete intconstant\n"); }
       | radstmt
{ printf("factor -> radstmt\n"); }
       | IDENTIFIER PARANTEZAPATRATADESCHISA IDENTIFIER
PARANTEZAPATRATAINCHISA     { printf("factor -> identifier [ identifier
]\n"); }
       | IDENTIFIER PARANTEZAPATRATADESCHISA INTCONSTANT
PARANTEZAPATRATAINCHISA    { printf("factor -> identifier [ intconstant
]\n"); }
       | PARANTEZADESCHISA expression PARANTEZAINCHISA
{ printf("factor -> ( expression )\n"); }
       ;

iostmt : CITESTE PARANTEZADESCHISA IDENTIFIER PARANTEZAINCHISA       {
printf("iostmt -> citeste ( identifier )\n"); }
       | SCRIE PARANTEZADESCHISA IDENTIFIER PARANTEZAINCHISA         {
printf("iostmt -> scrie ( identifier )\n"); }
       | SCRIE PARANTEZADESCHISA INTCONSTANT PARANTEZAINCHISA        {
printf("iostmt -> scrie ( intconstant )\n"); }
       | SCRIE PARANTEZADESCHISA STRINGCONSTANT PARANTEZAINCHISA     {
printf("iostmt -> scrie ( stringconstant )\n"); }
       ;

radstmt : RADICAL PARANTEZADESCHISA IDENTIFIER PARANTEZAINCHISA        {
printf("radstmt -> radical ( identifier )\n"); }
        ;

structstmt : ifstmt           { printf("structstmt -> daca\n"); }
           | whilestmt        { printf("structstmt -> cattimp\n"); }
           | foreachstmt      { printf("structstmt -> pentrufiecare\n"); }
           | forstmt          { printf("structstmt -> pentru\n"); }
           ;

ifstmt : DACA PARANTEZADESCHISA condition PARANTEZAINCHISA ATUNCI
ACOLADADESCHISA stmtlist ACOLADAINCHISA
```

```
{ printf("ifstmt -> daca parantezadeschisa condition parantezainchisa
atunci acoladadeschisa stmtlist acoladainchisa\n"); }
       | DACA PARANTEZADESCHISA condition PARANTEZAINCHISA ATUNCI
ACOLADADESCHISA stmtlist ACOLADAINCHISA ALTFEL ACOLADADESCHISA stmtlist
ACOLADAINCHISA    { printf("ifstmt -> daca parantezadeschisa condition
parantezainchisa atunci acoladadeschisa stmtlist acoladainchisa altfel
acoladadeschisa stmtlist acoladainchisa\n"); }
       ;

condition : expression relation expression                        {
printf("condition -> expression relation expression\n"); }
          | expression relation expression SISI condition         {
printf("condition -> expression relation expression sisi condition\n"); }
          | expression relation expression SAUSAU condition       {
printf("condition -> expression relation expression sausau condition\n"); }
          ;

relation : MAIMAREEGAL          { printf("relation -> maimareegal\n"); }
         | MAIMICEGAL           { printf("relation -> maimicegal\n"); }
         | MAIMARE              { printf("relation -> maimare\n"); }
         | MAIMIC               { printf("relation -> maimic\n"); }
         | VERIFICAEGAL         { printf("relation -> verificaegal\n"); }
         | ESTIEGAL             { printf("relation -> estiegal\n"); }
         | VERIFICANUEGAL       { printf("relation -> verificanuegal\n"); }
         ;

whilestmt : CATTIMP PARANTEZADESCHISA condition PARANTEZAINCHISA FA
ACOLADADESCHISA stmtlist ACOLADAINCHISA { printf("whilestmt -> cattimp
parantezadeschisa condition parantezainchisa fa acoladadeschisa stmtlist
acoladainchisa\n"); }
          ;

forstmt : PENTRU PARANTEZADESCHISA assignstmt PUNCTVIRGULA condition
PUNCTVIRGULA assignstmt PARANTEZAINCHISA FA ACOLADADESCHISA stmtlist
ACOLADAINCHISA { printf("forstmt -> pentru parantezadeschisa assignstmt ;
condition ; assignstmt parantezainchisa fa acoladadeschisa stmtlist
acoladainchisa\n"); }
        ;

foreachstmt : PENTRUFIECARE PARANTEZADESCHISA IDENTIFIER IN IDENTIFIER
PARANTEZAINCHISA FA ACOLADADESCHISA stmtlist ACOLADAINCHISA {
printf("foreachstmt -> pentrufiecare parantezadeschisa identifier in
identifier parantezainchisa fa acoladadeschisa stmtlist acoladainchisa\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");
}
```

Lex file

```
%{
#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 [stergete]?{NON_ZERO_DIGIT}{DIGIT}*|0
LETTER [a-zA-Z_]
SIGNS [ _.,:;]
STRING_CONSTANT \"[^\"]*\"
IDENTIFIER (€{LETTER})({LETTER}|{DIGIT}|_)*
BAD_IDENTIFIER (^€)({LETTER}|{DIGIT}|_)*

%%

"prog" { printf("reserved word: %s\n", yytext); return PROG; }
"cvnou" { printf("reserved word: %s\n", yytext); return CVNOU; }
"daca" { printf("reserved word: %s\n", yytext); return DACA; }
"altfel" { printf("reserved word: %s\n", yytext); return ALTFEL; }
"cattimp" { printf("reserved word: %s\n", yytext); return CATTIMP; }
"fa" { printf("reserved word: %s\n", yytext); return FA; }
"pentrufiecare" { printf("reserved word: %s\n", yytext); return
PENTRUFIECARE; }
"oprestete" { printf("reserved word: %s\n", yytext); return OPRESTETE; }
"continua" { printf("reserved word: %s\n", yytext); return CONTINUA; }
"scrie" { printf("reserved word: %s\n", yytext); return SCRIE; }
"citeste" { printf("reserved word: %s\n", yytext); return CITESTE; }
"in" { printf("reserved word: %s\n", yytext); return IN; }
"de" { printf("reserved word: %s\n", yytext); return DE; }
"atunci" { printf("reserved word: %s\n", yytext); return ATUNCI; }
"incepe" { printf("reserved word: %s\n", yytext); return INCEPE; }
"termina" { printf("reserved word: %s\n", yytext); return TERMINA; }
"intreg" { printf("reserved word: %s\n", yytext); return INTREG; }
"real" { printf("reserved word: %s\n", yytext); return REAL; }
"sfoara" { printf("reserved word: %s\n", yytext); return SFOARA; }
"sir" { printf("reserved word: %s\n", yytext); return SIR; }
"constanta" { printf("reserved word: %s\n", yytext); return CONSTANT; }
"functie" { printf("reserved word: %s\n", yytext); return FUNCTIE; }
"intoarce" { printf("reserved word: %s\n", yytext); return INTOARCE; }
"advfals" { printf("reserved word: %s\n", yytext); return ADVFALS; }
"caracter" { printf("reserved word: %s\n", yytext); return CARACTER; }
"radical" { printf("reserved word: %s\n", yytext); return RADICAL; }
"sisi" { printf("reserved word: %s\n", yytext); return SISI; }
"sausau" { printf("reserved word: %s\n", yytext); return SAUSAU; }

"adunate" { printf("operator: %s\n", yytext); return ADUNATE; }
"stergete" { printf("operator: %s\n", yytext); return STERGETE; }
"oriori" { printf("operator: %s\n", yytext); return ORIORI; }
"orioriinvers" { printf("operator: %s\n", yytext); return ORIORIINVERS; }
"estiegal" { printf("operator: %s\n", yytext); return ESTIEGAL; }
"verificaegal" { printf("operator: %s\n", yytext); return VERIFICAEGAL; }
"verificanuegal" { printf("operator: %s\n", yytext); return VERIFICANUEGAL;
}
```

```
"maimare" { printf("operator: %s\n", yytext); return MAIMARE; }
"maimic" { printf("operator: %s\n", yytext); return MAIMIC; }
"maimareegal" { printf("operator: %s\n", yytext); return MAIMAREEGAL; }
"maimicegal" { printf("operator: %s\n", yytext); return MAIMICEGAL; }
"lasuta" { printf("operator: %s\n", yytext); return LASUTA; }

"spatiu" { printf("symbol: %s\n", yytext); return SPATIU; }
"sarilinia" { printf("symbol: %s\n", yytext); return SARILINIA; }
"gatalinia" { printf("symbol: %s\n", yytext); return GATALINIA; }
";" { printf("symbol: %s\n", yytext); return PUNCTVIRGULA; }
":" { printf("symbol: %s\n", yytext); return DOUAPUNCTE; }
"." { printf("symbol: %s\n", yytext); return PUNCT; }
"," { printf("symbol: %s\n", yytext); return VIRGULA; }
"(" { printf("symbol: %s\n", yytext); return PARANTEZADESCHISA; }
")" { printf("symbol: %s\n", yytext); return PARANTEZAINCHISA; }
"{" { printf("symbol: %s\n", yytext); return ACOLADADESCHISA; }
"}" { printf("symbol: %s\n", yytext); return ACOLADAINCHISA; }
"[" { printf("symbol: %s\n", yytext); return PARANTEZAPATRATADESCHISA; }
"]" { printf("symbol: %s\n", yytext); return PARANTEZAPATRATAINCHISA; }
"_" { printf("symbol: %s\n", yytext); return LINIESUBTIRE; }

{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 (I'm on Mac so I will use brew):
   ```
   brew install bison
   ```
2. Compile the bison file:
   ```
   bison -d lang.y
   ```
3. Generate the lexer code:
   ```
   flex --header-file=lexer.h -o lexer.c lang.lxi
   ```
4. Compile the generated C code (I have an Intel chip and so flex is located at /usr/local/opt/flex for me. On an Apple Silicon chip, it is located at /opt/homebrew/opt/flex):
   ```
   gcc -o lang lang.tab.c lexer.c -L/usr/local/opt/flex/lib -lfl
   ```
5. Run the bison:
   ```
   ./lang p1.txt
   ```
6. Output of p1.txt:
   ```
   reserved word: prog
   symbol: {
   ```

```
stmtlist -> empty
reserved word: cvnou
identifier: €a
symbol: :
reserved word: intreg
type1 -> intreg
type -> type1
declaration -> cvnou identifier : type
simplstmt -> declaration
symbol: ;
stmt -> simplstmt ;
reserved word: cvnou
stmtlist -> stmtlist stmt
identifier: €b
symbol: :
reserved word: intreg
type1 -> intreg
type -> type1
declaration -> cvnou identifier : type
simplstmt -> declaration
symbol: ;
stmt -> simplstmt ;
reserved word: cvnou
stmtlist -> stmtlist stmt
identifier: €c
symbol: :
reserved word: intreg
type1 -> intreg
type -> type1
declaration -> cvnou identifier : type
simplstmt -> declaration
symbol: ;
stmt -> simplstmt ;
reserved word: citeste
stmtlist -> stmtlist stmt
symbol: (
identifier: €a
symbol: )
iostmt -> citeste ( identifier )
simplstmt -> iostmt
symbol: ;
stmt -> simplstmt ;
reserved word: citeste
stmtlist -> stmtlist stmt
symbol: (
identifier: €b
symbol: )
iostmt -> citeste ( identifier )
simplstmt -> iostmt
symbol: ;
stmt -> simplstmt ;
reserved word: citeste
stmtlist -> stmtlist stmt
symbol: (
identifier: €c
symbol: )
iostmt -> citeste ( identifier )
simplstmt -> iostmt
symbol: ;
stmt -> simplstmt ;
reserved word: cvnou
```

```
 stmtlist -> stmtlist stmt
 identifier: €smallest_int
 symbol: :
 reserved word: intreg
 type1 -> intreg
 type -> type1
 declaration -> cvnou identifier : type
 simplstmt -> declaration
 symbol: ;
 stmt -> simplstmt ;
 identifier: €smallest_int
 stmtlist -> stmtlist stmt
 operator: estiegal
 identifier: €a
 symbol: ;
 term -> identifier
 expression -> term
 assignstmt -> identifier estiegal expression
 simplstmt -> assignstmt
 stmt -> simplstmt ;
 reserved word: daca
 stmtlist -> stmtlist stmt
 symbol: (
 identifier: €smallest_int
 operator: maimare
 term -> identifier
 expression -> term
 relation -> maimare
 identifier: €b
 symbol: )
 term -> identifier
 expression -> term
 condition -> expression relation expression
 reserved word: atunci
 symbol: {
 stmtlist -> empty
 identifier: €smallest_int
 operator: estiegal
 identifier: €b
 symbol: ;
 term -> identifier
 expression -> term
 assignstmt -> identifier estiegal expression
 simplstmt -> assignstmt
 stmt -> simplstmt ;
 symbol: }
 stmtlist -> stmtlist stmt
 reserved word: daca
 ifstmt -> daca parantezadeschisa condition parantezainchisa atunci
acoladadeschisa stmtlist acoladainchisa
 structstmt -> daca
 stmt -> structstmt
 stmtlist -> stmtlist stmt
 symbol: (
 identifier: €smallest_int
 operator: maimare
 term -> identifier
 expression -> term
 relation -> maimare
 identifier: €c
 symbol: )
```

```
 term -> identifier
 expression -> term
 condition -> expression relation expression
 reserved word: atunci
 symbol: {
 stmtlist -> empty
 identifier: €smallest_int
 operator: estiegal
 identifier: €c
 symbol: ;
 term -> identifier
 expression -> term
 assignstmt -> identifier estiegal expression
 simplstmt -> assignstmt
 stmt -> simplstmt ;
 symbol: }
 stmtlist -> stmtlist stmt
 reserved word: scrie
 ifstmt -> daca parantezadeschisa condition parantezainchisa atunci
acoladadeschisa stmtlist acoladainchisa
 structstmt -> daca
 stmt -> structstmt
 stmtlist -> stmtlist stmt
 symbol: (
 identifier: €smallest_int
 symbol: )
 iostmt -> scrie ( identifier )
 simplstmt -> iostmt
 symbol: ;
 stmt -> simplstmt ;
 symbol: }
 stmtlist -> stmtlist stmt
 program -> prog { stmtlist }
         OK
```

In this implementation, I am also using lex to return tokens and yacc to return a string of productions.