Formal-Languages-and-Compiler-Design/Lab9 at main · krisztinahorvath/Formal-Languages-and-Compiler-Design (github.com)

BONUS: Flex file returns tokens and bison returns strings of productions.

Parser.y

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

int yyerror(const char *s);

#define YYDEBUG 1
%}

%token MAIN;
%token INTEGER;
%token STRING;
%token ARRAY;
%token IF;
%token ELSE;
%token WHILE;
%token READ;
%token WRITE;

%token PLUS;
%token MINUS;
%token TIMES;
%token DIV;
%token LESS;
%token LESSEQ;
%token EQ;
%token NEQ;
%token BIGGEREQ;
%token EQQ;
%token BIGGER;

%token SQBRACKETOPEN;
%token SQBRACKETCLOSE;
%token SEMICOLON;
%token OPEN;
%token CLOSE;
%token BRACKETOPEN;
```

```
%token BRACKETCLOSE;
%token COMMA;

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

%start Program

%%
Program : MAIN BRACKETOPEN CmpdStmt BRACKETCLOSE { printf("Program -> main {
CmpdStmt }\n"); }
        ;

CmpdStmt : StmtList {printf("CmpdStmt -> StmtList\n");}
         ;

StmtList : Stmt StmtList     { printf("StmtList -> Stmt StmtList\n"); }
         | Stmt     { printf("StmtList -> Stmt\n"); }
         ;

Stmt : AssignStmt     { printf("Stmt -> AssignStmt\n"); }
     | IOStmt     { printf("Stmt -> IOStmt\n"); }
     | Declaration     { printf("Stmt -> Declaration\n"); }
     | IfStmt     { printf("Stmt -> IfStmt\n"); }
     | WhileStmt     { printf("Stmt -> WhileStmt\n"); }
     ;

AssignStmt : IDENTIFIER EQ Expression SEMICOLON   { 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"); }
       | STRINGCONSTANT     { printf("Factor -> STRINGCONSTANT\n"); }
       | INTCONSTANT     { printf("Factor -> INTCONSTANT\n"); }
       ;

IOStmt : READ OPEN IDENTIFIER CLOSE SEMICOLON  { printf("IOStmt -> read (
IDENTIFIER ) ;\n"); }
       | WRITE OPEN IDENTIFIER CLOSE SEMICOLON   { printf("IOStmt -> write (
IDENTIFIER ) ;\n"); }
       | WRITE OPEN STRINGCONSTANT CLOSE SEMICOLON   { printf("IOStmt -> write (
STRINGCONSTANT ) ;\n"); }
       ;

Declaration : Type IDENTIFIER SEMICOLON     { printf("Declaration -> Type
IDENTIFIER ;\n"); }
            ;

Type : INTEGER     { printf("Type -> INTEGER\n"); }
     | STRING     { printf("Type -> STRING\n"); }
     | ARRAY SQBRACKETOPEN INTCONSTANT SQBRACKETCLOSE    { printf("Type -> ARRAY
[INTCONSTANT]\n"); }
     ;

IfStmt : IF OPEN Condition CLOSE BRACKETOPEN StmtList BRACKETCLOSE     {
printf("IfStmt -> if (Condition) { StmtList }\n"); }
       | IF OPEN Condition CLOSE BRACKETOPEN StmtList BRACKETCLOSE ELSE
BRACKETOPEN StmtList BRACKETCLOSE     { printf("IfStmt -> if (Condition) {
StmtList } else { StmtList }\n"); }
       ;

WhileStmt : WHILE OPEN Condition CLOSE BRACKETOPEN StmtList BRACKETCLOSE     {
printf("WhileStmt -> while (Condition) { StmtList }\n"); }
          ;

Relation : LESS     { printf("Relation -> <\n"); }
         | LESSEQ     { printf("Relation -> <=\n"); }
         | EQQ     { printf("Relation -> ==\n"); }
         | NEQ     { printf("Relation -> !=\n"); }
         | BIGGEREQ     { printf("Relation -> >=\n"); }
         | BIGGER     { printf("Relation -> >\n"); }
         ;
Condition : Expression Relation Expression     { printf("Condition -> Expression
RELATION Expression\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");
    return 0;
}
```

Tokens from scanner.l

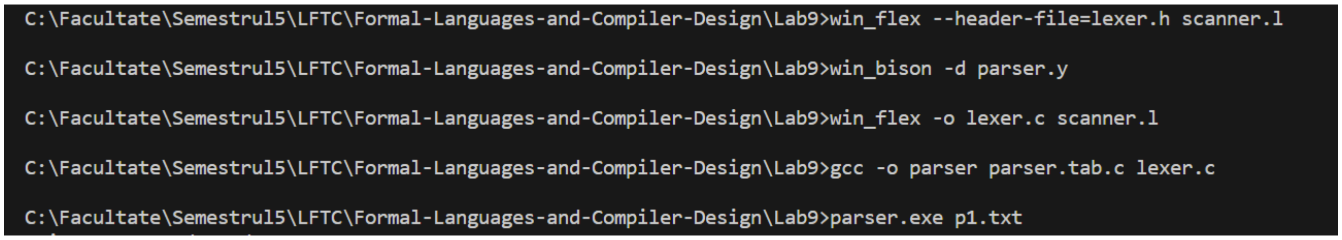
```
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include "parser.tab.h"
    int lines = 1;
%}

%option noyywrap
%option caseless

DIGIT [0-9]
NON_ZERO_DIGIT [1-9]
INTCONSTANT [+-]?{NON_ZERO_DIGIT}{DIGIT}*|0
LETTER [a-zA-Z_]
SPECIAL_CHAR [ ?:*\^+=.!]
STRINGCONSTANT (\"({LETTER}|{DIGIT}|{SPECIAL_CHAR})*\")
IDENTIFIER {LETTER}({LETTER}|{DIGIT})*
BAD_IDENTIFIER ({DIGIT})+({LETTER})+({LETTER}|{DIGIT})*

%%

"main" { printf("%s - reserved word\n", yytext); return MAIN; }
"integer" { printf("%s - reserved word\n", yytext); return INTEGER; }
"string" { printf("%s - reserved word\n", yytext); return STRING; }
"read" { printf("%s - reserved word\n", yytext); return READ; }
```

```
"if" { printf("%s - reserved word\n", yytext); return IF; }
"else" { printf("%s - reserved word\n", yytext); return ELSE; }
"write" { printf("%s - reserved word\n", yytext); return WRITE; }
"while" { printf("%s - reserved word\n", yytext); return WHILE; }
"array" { printf("%s - reserved word\n", yytext); return ARRAY; }

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

"[" { printf("%s - separator\n", yytext); return SQBRACKETOPEN; }
"]" { printf("%s - separator\n", yytext); return SQBRACKETCLOSE; }
";" { printf("%s - separator\n", yytext); return SEMICOLON; }
"(" { printf("%s - separator\n", yytext); return OPEN; }
")" { printf("%s - separator\n", yytext); return CLOSE; }
"{" { printf("%s - separator\n", yytext); return BRACKETOPEN; }
"}" { printf("%s - separator\n", yytext); return BRACKETCLOSE; }
"," { printf("%s - separator\n", yytext); return COMMA; }

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

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

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

{STRINGCONSTANT} { printf("%s - string constant\n", yytext); return
STRINGCONSTANT; }

[ \t]+ {}

[\n]+ {++lines;}

. {printf("ERROR at token %s at line %d\n", yytext, lines); exit(1);}

%%
```

Demo:

Prerequisites: have WinFlexBison installed on your Windows machine.

1. Create lexer header file
2. Compile bison file
3. Generate lexer code
4. Compile generated C files
5. Run the executable



```
C:\Facultate\Semestrul5\LFTC\Formal-Languages-and-Compiler-Design\Lab9>win_flex --header-file=lexer.h scanner.l

C:\Facultate\Semestrul5\LFTC\Formal-Languages-and-Compiler-Design\Lab9>win_bison -d parser.y

C:\Facultate\Semestrul5\LFTC\Formal-Languages-and-Compiler-Design\Lab9>win_flex -o lexer.c scanner.l

C:\Facultate\Semestrul5\LFTC\Formal-Languages-and-Compiler-Design\Lab9>gcc -o parser parser.tab.c lexer.c

C:\Facultate\Semestrul5\LFTC\Formal-Languages-and-Compiler-Design\Lab9>parser.exe p1.txt
```

After running all programs with the parser these are the outputs:

Formal-Languages-and-Compiler-Design/Lab9/output.txt at main · krisztinahorvath/Formal-Languages-and-Compiler-Design (github.com)