Documentation

link git:

https://github.com/cs-ubbcluj-ro/lab-work-computer-science-2024-carla-mirea/tree/main/4-Lex-Yacc

---

lang.lxi:

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "parser.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 ][\']

%%

"be"              { printf("Reserved word: %s\n", yytext); return BE; }
"number"          { printf("Reserved word: %s\n", yytext); return NUMBER; }
"integer"         { printf("Reserved word: %s\n", yytext); return INTEGER; }
"bool"            { printf("Reserved word: %s\n", yytext); return BOOL; }
"string"          { printf("Reserved word: %s\n", yytext); return STRING; }
"char"            { printf("Reserved word: %s\n", yytext); return CHAR; }
"const"           { printf("Reserved word: %s\n", yytext); return CONST; }
"check"           { printf("Reserved word: %s\n", yytext); return CHECK; }
"else"            { printf("Reserved word: %s\n", yytext); return ELSE; }
"readFromConsole" { printf("Reserved word: %s\n", yytext); return READFROMCONSOLE; }
"showInConsole"   { printf("Reserved word: %s\n", yytext); return SHOWINCONSOLE; }
"stopWhen"        { printf("Reserved word: %s\n", yytext); return STOPWHEN; }
"function"        { printf("Reserved word: %s\n", yytext); return FUNCTION; }
"for"             { printf("Reserved word: %s\n", yytext); return FOR; }
"start"     { printf("Reserved word: %s\n", yytext); return START; }

"+"               { printf("Operator: %s\n", yytext); return PLUS; }
"-"               { printf("Operator: %s\n", yytext); return MINUS; }
"*"               { printf("Operator: %s\n", yytext); return MULTIPLY; }
"/"               { printf("Operator: %s\n", yytext); return DIVIDE; }
"\\\\"            { printf("Operator: %s\n", yytext); return BACKSLASH; }
"%"               { printf("Operator: %s\n", yytext); return MODULO; }
"<"               { printf("Operator: %s\n", yytext); return LESS; }
"<="              { printf("Operator: %s\n", yytext); return LESSEQUAL; }
">"               { printf("Operator: %s\n", yytext); return GREATER; }
">="              { printf("Operator: %s\n", yytext); return GREATEREQUAL; }
"=="              { printf("Operator: %s\n", yytext); return EQUAL; }
"!="              { printf("Operator: %s\n", yytext); return NOTEQUAL; }
"&&"              { printf("Operator: %s\n", yytext); return AND; }
"||"              { printf("Operator: %s\n", yytext); return OR; }
"="               { printf("Operator: %s\n", yytext); return ASSIGN; }

"("               { printf("Separator: %s\n", yytext); return LEFTROUND; }
")"               { printf("Separator: %s\n", yytext); return RIGHTROUND; }
"{"               { printf("Separator: %s\n", yytext); return LEFTCURLY; }
"}"               { printf("Separator: %s\n", yytext); return RIGHTCURLY; }
"["               { 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 constant: %s\n", yytext); return NUMBER_CONST; }
{STRING_CONST}    { printf("String constant: %s\n", yytext); return STRING_CONST; }
{CHAR_CONST}      { printf("Character constant: %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);}

%%
```

lang.y:

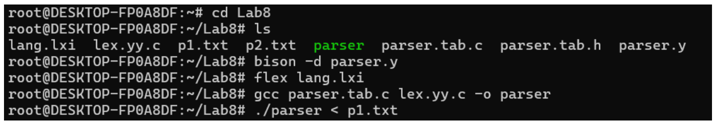
```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define YYDEBUG 1

int yylex();
void yyerror(char *);
%}

%token BE
%token NUMBER
%token INTEGER
%token BOOL
%token STRING
%token CHAR
%token CONST
%token CHECK
%token ELSE
%token READFROMCONSOLE
%token SHOWINCONSOLE
%token STOPWHEN
%token FUNCTION
%token FOR
%token START

%token PLUS
%token MINUS
%token MULTIPLY
%token DIVIDE
%token BACKSLASH
%token MODULO
%token LESS
%token LESSEQUAL
%token GREATER
%token GREATEREQUAL
%token EQUAL
%token NOTEQUAL
%token AND
%token OR
%token ASSIGN

%token LEFTROUND
%token RIGHTROUND
%token LEFTCURLY
%token RIGHTCURLY
%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 program

%%

program : START compound_statement

statement : declaration SEMICOLON
          | assignment_statement
          | return_statement SEMICOLON
          | iostmt SEMICOLON
          | if_statement
          | while_statement
          | for_statement

statement_list : statement
               | statement statement_list

compound_statement : LEFTCURLY statement_list RIGHTCURLY

expression : expression PLUS term
           | expression MINUS term
           | term

term : term MULTIPLY factor
     | term DIVIDE factor
     | term MODULO factor
     | factor

factor : LEFTROUND expression RIGHTROUND
       | IDENTIFIER
       | constant

constant : NUMBER_CONST
         | STRING_CONST
         | CHAR_CONST

iostmt : READFROMCONSOLE LEFTROUND IDENTIFIER RIGHTROUND
       | SHOWINCONSOLE LEFTROUND IDENTIFIER RIGHTROUND
       | SHOWINCONSOLE LEFTROUND constant RIGHTROUND

simple_type : NUMBER
            | INTEGER
            | BOOL
            | STRING
            | CHAR

array_declaration : simple_type IDENTIFIER LEFTBRACKET RIGHTBRACKET

declaration : BE IDENTIFIER simple_type
            | array_declaration

assignment_statement : IDENTIFIER ASSIGN expression SEMICOLON

if_statement : CHECK LEFTROUND condition RIGHTROUND compound_statement
             | CHECK LEFTROUND condition RIGHTROUND compound_statement ELSE compound_statement

while_statement : STOPWHEN LEFTROUND condition RIGHTROUND compound_statement

return_statement : FUNCTION expression

for_statement : FOR for_header compound_statement

for_header : LEFTROUND INTEGER assignment_statement condition assignment_statement RIGHTROUND

condition : expression relation expression
```

```
relation : LESS
         | LESSEQUAL
         | EQUAL
         | NOTEQUAL
         | GREATEREQUAL
         | GREATER

%%

void yyerror(char *s)
{
    fprintf(stderr, "Error: %s\n", s);
}

extern FILE *yyin;

int 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;
}
```

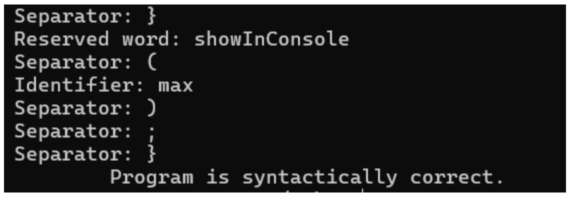
Demo:

We run the commands (in Ubuntu):



```
root@DESKTOP-FP0A8DF:~# cd Lab8
root@DESKTOP-FP0A8DF:~/Lab8# ls
lang.lxi  lex.yy.c  p1.txt  p2.txt  parser  parser.tab.c  parser.tab.h  parser.y
root@DESKTOP-FP0A8DF:~/Lab8# bison -d parser.y
root@DESKTOP-FP0A8DF:~/Lab8# flex lang.lxi
root@DESKTOP-FP0A8DF:~/Lab8# gcc parser.tab.c lex.yy.c -o parser
root@DESKTOP-FP0A8DF:~/Lab8# ./parser < p1.txt
```

Then we have the outputs corresponding to the program in the .txt file, including the message if it syntactically correct or not:



```
Separator: }
Reserved word: showInConsole
Separator: (
Identifier: max
Separator: )
Separator: ;
Separator: }
        Program is syntactically correct.
```