Documentation

Link to GitHub : https://github.com/diana-dr/Formal-Languages-and-Compiler-Design/tree/master/Lab%208

spec.lxi

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

int currentLine = 1;
%}

%option noyywrap
%option caseless

DIGIT           [0-9]
NZ_DIGIT    [1-9]
ZERO        [0]
NUMBER      {NZ_DIGIT}{DIGIT}*
SIGN        [+]|[-]
INTEGER         {ZERO}|{NUMBER}|{SIGN}{NUMBER}
SIGNER_INTEGER {SIGN}{NUMBER}
SPECIAL_CHAR
"_"|"."|","|";"|":"|"?"|"!"|"@"|"/"|"("|")"|"-"|"+"|"="|"{"|"}"|"*"|"["|"]"|"$"|
"%"|"^"|" "
CHAR        {DIGIT}|{SPECIAL_CHAR}|[a-zA-Z]
CHARACTER       "'"{CHAR}"'"
STRING          [\"]{CHAR}*[\"]
CONSTANT                {STRING}|{INTEGER}|{CHARACTER}
IDENTIFIER              [a-zA-Z_][a-zA-Z0-9_]*

%%

and {return AND;}
do {return DO;}
or {return OR;}
not {return NOT;}
if {return IF;}
else {return ELSE;}
elif {return ELIF;}
while {return WHILE;}
for {return FOR;}
read {return READ;}
write {return WRITE;}
int {return INTEGER;}
string {return STRING;}
char {return CHAR;}
function {return FUNCTION;}
bool {return BOOL;}
return {return RETURN;}


{CONSTANT} {return IDENTIFIER;}
{IDENTIFIER} {return CONSTANT;}

; {return SEMI_COLON;}
"," {return COMMA;}
\t {return DOT;}
\{ {return OPEN_CURLY_BRACKET;}
\} {return CLOSED_CURLY_BRACKET;}
\[ {return OPEN_SQUARE_BRACKET;}
```

```
\] {return CLOSED_SQUARE_BRACKET;}
\( {return OPEN_ROUND_BRACKET;}
\) {return CLOSED_ROUND_BRACKET;}

\+ {return PLUS;}
\- {return MINUS;}
\* {return MUL;}
\/ {return DIV;}
\% { return PERCENT;}
\< { return LT;}
\> { return GT;}
\<= { return LE;}
\>= { return GE;}
"=" { return ATRIB;}
\== { return EQ;}
\!= { return NOT_EQ;}

[\n\r] {currentLine++;}
[ \t\n]+ {}

(\+0)|(\-0) printf("! Lexical error: %s\n", yytext);
{INTEGER}{IDENTIFIER} printf("! Lexical error: %s\n", yytext);
0{INTEGER} printf("! Lexical error: %s\n", yytext);
```

spec.y

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

#define YYDEBUG 1
%}

%token DO
%token AND
%token OR
%token NOT
%token IF
%token ELSE
%token ELIF
%token WHILE
%token FOR
%token READ
%token WRITE
%token INTEGER
%token STRING
%token CHAR
%token BOOL
%token RETURN
%token FUNCTION
%token IDENTIFIER
%token CONSTANT
%token SEMI_COLON
%token COMMA
%token DOT
%token OPEN_CURLY_BRACKET
%token CLOSED_CURLY_BRACKET
%token OPEN_SQUARE_BRACKET
%token CLOSED_SQUARE_BRACKET
%token OPEN_ROUND_BRACKET
%token CLOSED_ROUND_BRACKET
%token PLUS
%token MINUS
```

```
%token MUL
%token DIV
%token PERCENT
%token LT
%token GT
%token LE
%token GE
%token ATRIB
%token EQ
%token NOT_EQ

%left '+' '-' '*' '/'


%start program_stmt

%%

program_stmt : FUNCTION compound_stmt
             ;

compound_stmt : OPEN_CURLY_BRACKET stmt_list CLOSED_CURLY_BRACKET
              ;

stmt_list : stmt
          | stmt stmt_list
          ;

stmt : simple_stmt
     | complex_stmt
     ;

simple_stmt : decl_stmt
          | assign_stmt
          | return_stmt
          ;

complex_stmt : if_stmt
             | loop_stmt
             ;

decl_stmt : type IDENTIFIER NZidentifier
          | type IDENTIFIER ATRIB expression NZEidentifier
          ;

NZidentifier : COMMA IDENTIFIER NZidentifier
             | SEMI_COLON
             ;

NZEidentifier : COMMA IDENTIFIER ATRIB expression NZEidentifier
              | SEMI_COLON
              ;

type : primary_types
     ;

primary_types : INTEGER
              | CHAR
              | STRING
              | BOOL
              ;
assign_stmt : IDENTIFIER ATRIB expression
            ;

expression : term operator expression
           | term
```

```
          ;

operator : PLUS
         | MINUS
         ;

term : factor MUL term
     | factor DIV term
     | factor
     ;

factor : OPEN_ROUND_BRACKET expression CLOSED_ROUND_BRACKET
       | IDENTIFIER
       | IDENTIFIER OPEN_SQUARE_BRACKET expression CLOSED_SQUARE_BRACKET
       | CONSTANT
       ;

return_stmt : RETURN expression
            ;

if_stmt : IF condition DO compound_stmt
        | IF condition DO compound_stmt elif_stmt
        | IF condition AND condition DO compound_stmt
        ;

elif_stmt : ELIF condition DO compound_stmt elif_stmt
          | ELIF condition DO compound_stmt
          | ELSE DO compound_stmt
          ;

loop_stmt : for_stmt
          | while_stmt
          ;

for_stmt : FOR OPEN_ROUND_BRACKET for_first condition SEMI_COLON assign_stmt
CLOSED_ROUND_BRACKET compound_stmt
         | FOR OPEN_ROUND_BRACKET for_first condition CLOSED_ROUND_BRACKET
compound_stmt
         ;

for_first : decl_stmt
          | assign_stmt
          ;

while_stmt : WHILE OPEN_ROUND_BRACKET condition CLOSED_ROUND_BRACKET
compound_stmt
           ;

condition : expression relational_operator expression conditional_operator
condition
          | NOT expression relational_operator expression conditional_operator
condition
          | expression relational_operator expression
          | NOT expression relational_operator expression

relational_operator : GT
                    | LT
                    | GE
                    | LE
                    | EQ
                    | NOT_EQ
                    ;

conditional_operator : AND
                     | OR
                     ;
```

```
%%

yyerror(char *s)
{
  printf("%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,"\tThe input program is valid according to the given grammar
rules.\n");
  else
    printf( "The input program is incorrect.\n" );
  return 0;
}
```