----------ALPHABET----------

a. upper and lowercase letters : A-Z, a-Z;
b. underline character : '_'
c. decimal digits: 0-9

----------LEXIC----------

a. Special Symbols, representing:
- operators:
'+', '-', '*', '/', '%', '=', '==', '<>', '!=', '<', '<=', '>', '>=', '!'. "||", "&&"

- separators:
space, newline, '["]' , '{"}' , ',', ';', ":", " (quotes), " ' " (single quotes)

- reserved words:
true, false, fun, const, int, string, bool, char, array, and, return, for, while, if, else, throw, catch, switch, case, struct

b. Identifiers:
identifier = letter{letter | digit}
letter = "A" | "B" | "C" | ... | "Z" | "a" | "b" | ... | "z"
digit = "0" | "1" | ... | "9"

c. Constants
1. integer - rule
int_const = ["+" | "-"] non_zero_digit {digit} | "0"
non_zero_digit = "1" | "2" |... | "9"

2. character
character_const = 'letter' | 'digit'

3. bool
bool_const = "true" | "false";

4. string
string_const = """{letter | digit | "_" | | "-" | " "}

—----------TOKEN—----------

+
-
*
/
%
==
!=
<>
<
<=
>
>=
!
=
&&
||
'
"
(
)
{
}
,
:
;
true
false
var
const
int
string
bool
char
array
and
return

for
while
if
else
throw
catch
switch
case
struct

----------Syntactical Rules----------

```
Program      ::= Statement{ Statement } ;

Statement    ::= DeclarationStatement
               | Assignment
               | Input
               | Output
               | Conditional
               | Loop
               | Comment ;

Type         ::= "int"
               |   "bool"
               |   "char"
               |   "string"
               |   "array<"Type">[" NonZeroDigit{Digit}"]" ;

DeclarationStatement   ::= Type Identifier ["=" Expression] ";" ;
Assignment      ::=  Identifier"=" Expression ";" ;
Input           ::= "read("Identifier{,Identifier}")";
Output          ::= "print" ( Expression{, Expression} ) ";" ;
Conditional     ::= "if" "(" Expression ")" "{" Program "}" [ "else" "{" Program
"}" ] ;
Loop           ::= "while" "(" Expression ")" "{" Program "}" ;
Comment        ::= "//" {string} | "##" {string};
```

```
Expression      ::= Term { Operator Term } ;
Term            ::= Identifier
                  | IntConstant
                  | StringLiteral
                  | BoolConstant
                  | "("Expression")";

IntConstant     ::= [ "+" | "-" ] ( NonZeroDigit { Digit } ) | "0" ;
StringLiteral   ::= """{character literal}"""
BoolConstant  ::= "true" | "false" ;

Operator        ::= "+" | "-" | "*" | "/" | "%" | "==" | "!=" | "<" | "<=" | ">" | ">=" |
"=" | "&&" | "||";
Letter          ::= "A" | "B" | "C" | ... | "Z" | "a" | ... "z" ;
NonZeroDigit::= "1" | "2" | ... | "9" ;
Digit           ::= "0" | "1" | "2" | ... | "9" ;
```

P1 - Compute the maximum of 3 numbers

```
int max(int firstNumber, int secondNumber, int thirdNumber){
    if(firstNumber <= thirdNumber && secondNumber <= thirdNumber){
        return thirdNumber;
    } else if (firstNumber <= secondNumber && thirdNumber <=
secondNumber) {
        return secondNumber;
    } else {
        return firstNumber;
    }
}
```

P2 - Compute the greatest common divisor of 2 numbers

```
int gcd(int firstNumber, int secondNumber) {
    while(firstNumber <> secondNumber){
        if(firstNumber < secondNumber) {
            secondNumber = secondNumber - firstNumber;
        } else {
            firstNumber = firstNumber - secondNumber;
        }
    }
}
```

P3 - Compute the sum of n numbers, computer the max/min of n numbers

```
// A comment
## Also a comment

int sum(array<int> numbers) {
    var sum = 0;
    for(int i < numbers.size(); i++){
        sum += numbers[i];
    }
    return sum;
}
```

P1Error - **SyntacticErrors**

```
int max(int firstNumber, int secondNumber, int thirdNumber) {
    if firstNuumber <= thirdNumber && secondNumber <= thirdNumber {
## error: no parentheses
        return thirdNumber
## error: no ;
    } else if (first number <= secondNumber && thirdNumber <=
secondNumber) {
        return second number;
    } else {
        return firstNumber;
    }
}
```