Github link: https://github.com/AvramPop/flct/tree/master/lab3/src/com/company

Assignment:
Implement the scanning algorithm and use ST from lab 2 for the symbol table.
Input: Programs p1/p2/p3/p1err and token.in from Lab1a
Output: PIF.out, ST.out, message "lexically correct" or "lexical error + location"

Scanner algorithm:
The scanner algorithm takes each line of the input program and makes a list of valid tokens out of it. It then adds the tokens to their positions in the symbol table, if they are not operators, separators or keywords (those go to the unexisting index, (i.e. <-1, -1>)).
Scanner pseudocode:

```
Foreach token:
    If token is operator or token is separator or token is keyword:
        addOther(token)
        Else if token is identifier or token is constant:
            add(token)
        Else:
            addError(token)
```

For the tokenizer, the special cases are operators, separators and strings. In case they are not those, the tokenizer goes on reading char by char, until a "special case" mentioned before occurs.
Tokenizer pseudocode:

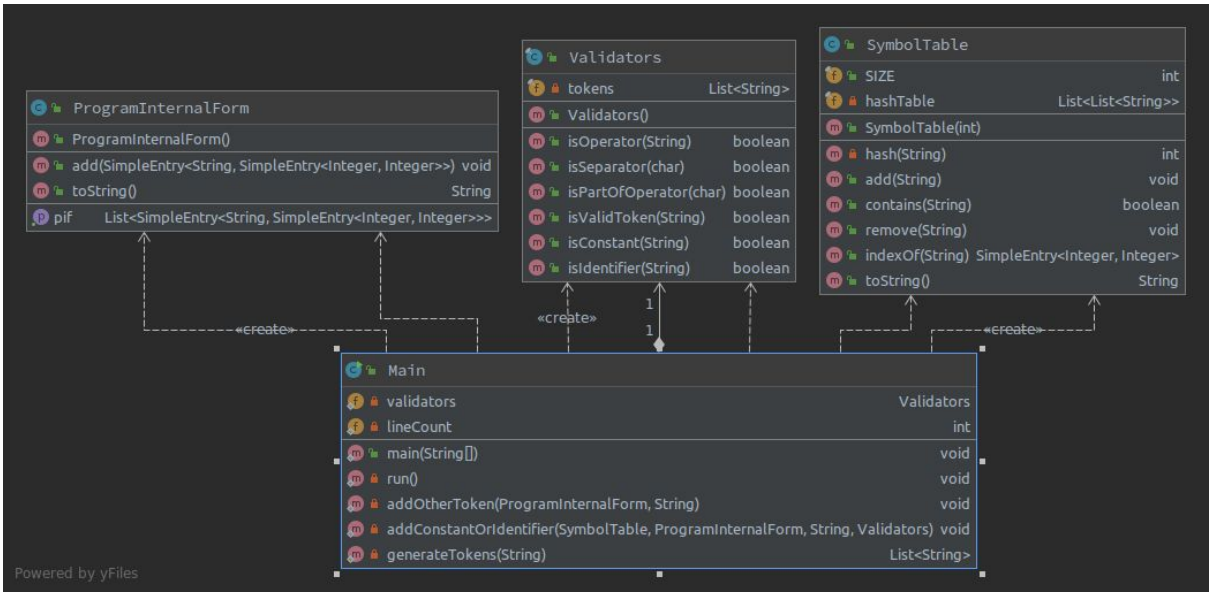
```
Foreach index in line:
    If line[index] is doublequote (") :
        String, index = parseString(line, index)
        Add String to result
        Keep analysing from index
    Else if line[index] is part of operator:
        If  line[index] + next  line[index + 1] is operator
            Add operator to result
            Keep analysing from currentIIndex + 2
        If (line[index] is "+" or line[index] is "-") and next line[index + 1] is digit:
            Token, index = parseDigit(line, index)
            Add number to result
            Keep analysing from currentIIndex + index
        Else:
            Add line[index] as operator to result
    Else if line[index] is separator:
        Add separator to result
        Keep analysing from index
    Else:
        Append line[index] to current token
Return result
```

If the token is not valid, append to the exception output buffer a proper message.

Program Internal Form:
My PIF is represented as a list of **pairs** defined as: <token, pair<slot, index in slot>>, where **slot** is the *index of the slot in the hash table* and **index in slot** is the *index of the token in the current slot*. Be *token* a constant or identifier, instead of the real value we will hold the special values “id” and “const”.

Symbol table:
My symbol table is represented using my custom implementation of a hash table. It uses a classical hash function on strings, i.e. *(s[0] * 31 ^ (n - 1) + ... + s[n - 1] * 31 * 0) % SIZE*, where SIZE is the fixed number of slots in my hash table. Behind the scenes I hold a list of lists of strings of variable sizes. Hence, any token will have an index consisting in a pair: <slot, index in slot> (for more information, please see the explanation for the PIF).

Scanner class structure:



Example of error message:

```
Lexical error: Unknown token '@' @ line 6
Lexical error: Unknown token '+0' @ line 9
Lexical error: Unknown token '"gcd is:  + a);' @ line 11
```

Constant regex check:

```
^(0|[\\+\\-]?[1-9][0-9]*)$|^[\\+\\-]?(0|([1-9][0-9]*))(\\.[0-9]*[1-9]+)$|^\\".*\\"$
```

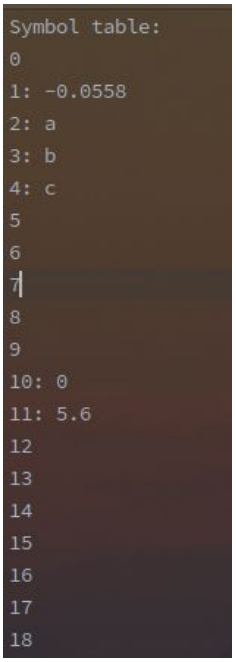
Identifier regex check:

```
^([a-zA-Z])+([a-zA-Z]|[0-9])*$
```

Example of PIF:

```
Program internal form:
token: "begin" @ slot: -1 index: -1
token: "{" @ slot: -1 index: -1
token: "double" @ slot: -1 index: -1
token: "id" @ slot: 2 index: 0
token: "," @ slot: -1 index: -1
token: "id" @ slot: 3 index: 0
token: "," @ slot: -1 index: -1
token: "id" @ slot: 4 index: 0
token: ";" @ slot: -1 index: -1
token: "id" @ slot: 4 index: 0
token: "=" @ slot: -1 index: -1
token: "const" @ slot: 11 index: 0
token: ";" @ slot: -1 index: -1
token: "read" @ slot: -1 index: -1
token: "(" @ slot: -1 index: -1
token: "id" @ slot: 2 index: 0
token: ")" @ slot: -1 index: -1
token: ";" @ slot: -1 index: -1
token: "read" @ slot: -1 index: -1
token: "(" @ slot: -1 index: -1
token: "id" @ slot: 3 index: 0
token: ")" @ slot: -1 index: -1
token: ";" @ slot: -1 index: -1
token: "while" @ slot: -1 index: -1
token: "(" @ slot: -1 index: -1
token: "id" @ slot: 4 index: 0
token: "const" @ slot: 10 index: 0
```

Example of ST:



```
Symbol table:
0
1: -0.0558
2: a
3: b
4: c
5
6
7
8
9
10: 0
11: 5.6
12
13
14
15
16
17
18
```

Syntax.in

program = "begin {" programStatement "}".
programStatement = {declarationList | compoundStatement}.
declarationList = declaration ";" | declaration ";" declarationList.
declaration = type " " identifierList.
identifierList = identifier ";" | identifier ", " identifierList.
basicType = "string" | "int" | "double".
sequenceType = basicType "[]".
type = basicType | sequenceType.
compoundStatement = "{" statementList "}".
statementList = statement | statement ";" statementList.
statement = simpleStatement | structuredStatement.
simpleStatement = assignmentStatement | ioStatement.
assignmentStatement = identifier "=" expression.
expression = term | expression arithmeticOrderOne term.
arithmeticOrderOne = "+" | "-".
arithmeticOrderTwo = "*" | "/" | "%".
term = term arithmeticOrderTwo factor | factor.
factor = "(" expression ")" | identifier.
ioStatement = "read" | "print" "(" identifier ")".
structuredStatement = compoundStatement | ifStatement | whileStatement.
ifStatement = "if" condition compoundStatement ["else" compoundStatement].
whileStatement = "WHILE" condition compoundStatement.
condition = expression relation expression.
relation = "<" | "<=" | "==" | "!=" | ">=" | ">".

Lexic.in

1. Alphabet letter = "A" | "B" | ... | "Z" | "a" | "b" | ... | "z".
nonZeroDigit = "1" | ... | "9".
digit = "0" | nonZeroDigit.

2. Lexic operator = "+" | "-" | "*" | "/" | "%" | "==" | "<" | "<=" | ">" | ">=" | "&&" | "||" | "!".
separator = "[" | "]" | "{" | "}" | ";" | " " | ".".
reservedWord = "int" | "double" | "string" | "if" | "else" | "for" | "while" | "read" | "print" | "begin".
identifier = letter | letter {letter | digit}.
All characters which are used in the definitions above (letters, operators etc.)
int = "0" | ["+" | "-"] nonZeroDigit{digit}.
string = “{letter | digit}”.
double = int | int "." {digit}