Jercan Ioana - https://github.com/jercanioana/FLCD

The assignment will be one of the following:

Implement the Symbol Table (ST) as the specified data structure, with the corresponding operations

1. Symbol Table:

a. unique for identifiers and constants (create one instance of ST)

2. Symbol Table (you need to implement the data structure and required operations) :

b. alphabetically sorted table

Implement a scanner (lexical analyzer): Implement the scanning algorithm and use ST from lab 2 for the symbol table.

For representing the symbol table, I used the BST. The order in which the elements are added is counted by positionInTable. In the SymbolTable I have two functions: adding a new node and searching for one. The add operation is a recursive one, as well as the search.

The classes I implemented are the following: Node (being a node in the tree) and having the following fields: right, left, info, position and positionInTable. I use this class in order to build the actual BST, which represents the Symbol Table. In the Symbol Table class I have as an attribute the root of the tree. Then, based on that root, I perform the add and search operations.

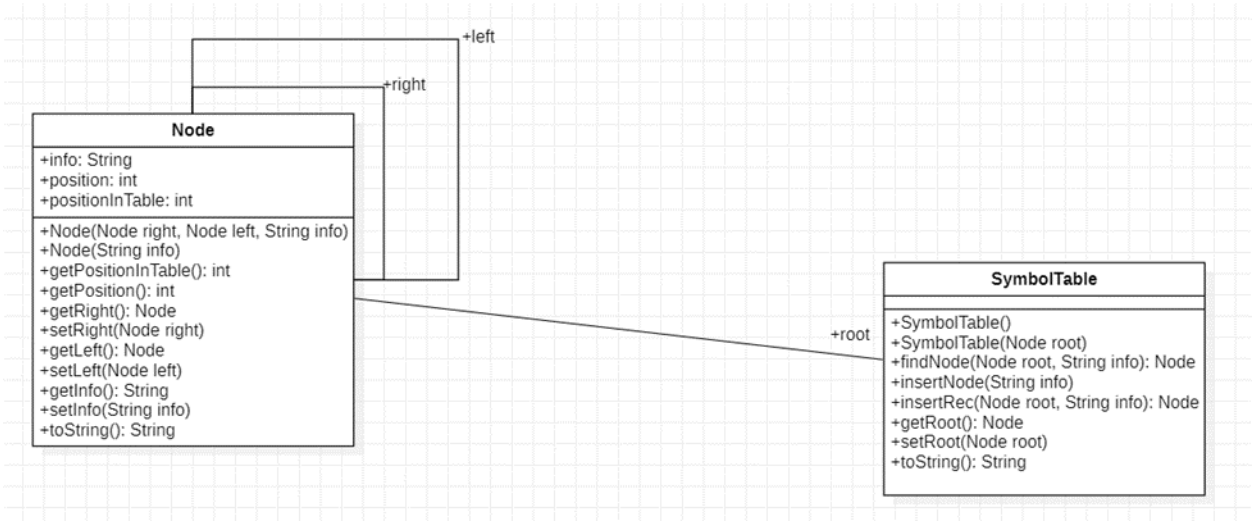
For this laboratory, I implemented three classes: Scanner, PIF and Pair.

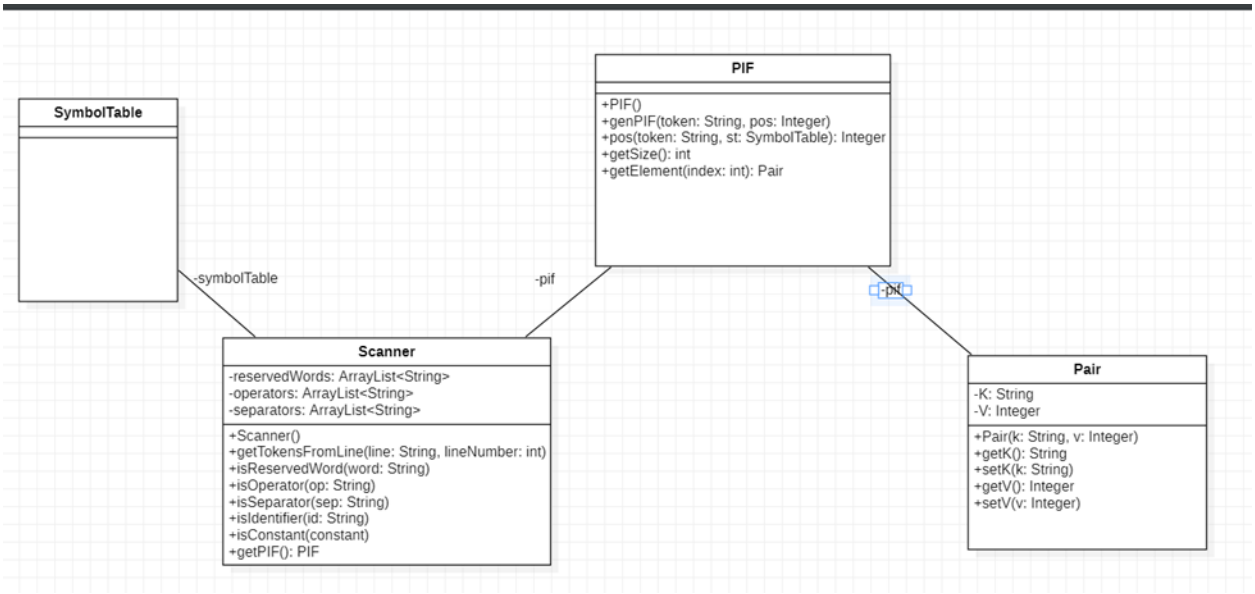
PIF represents the program's internal form, which has as an attribute an array of pairs. The class has 2 methods: genPIF, which creates a new pair and adds it to the pif and pos, which returns the position of the given token in the symbol table(if it is not already in the symbol table, it performs an add operation too).

Pair has 2 attributes:: a key (string) and a value(int). I use this class in order to represent the PIF.

Scanner contains as attributes the following: a list of reserved words, a list of separators, a list of operators, the pif and the symbol table. I have here a method called getTokensFromLine, which identifies the tokens by splitting them by separators, and then puts them into categories. The first case is if we have a reserved word or an operator, where we just put them in the PIF at position -1. The second case is if we have an identifier or constant. That is when we get the position from the symbol table (or add them if it's not already there) and add it at that position in the pif. The case for the string constants containing spaces needs to be treated differently. Finally, we go over the line again and add the separators too in the pif at position -1.

The splitting is done by the spaces, thus forcing spaces between tokens.


+left
+right
Node
+info: String
+position: int
+positionInTable: int
+Node(Node right, Node left, String info)
+Node(String info)
+getPositionInTable(): int
+getPosition(): int
+getRight(): Node
+setRight(Node right)
+getLeft(): Node
+setLeft(Node left)
+getInfo(): String
+setInfo(String info)
+toString(): String
+root
SymbolTable
+SymbolTable()
+SymbolTable(Node root)
+findNode(Node root, String info): Node
+insertNode(String info)
+insertRec(Node root, String info): Node
+getRoot(): Node
+setRoot(Node root)
+toString(): String


SymbolTable
-symbolTable
PIF
+PIF()
+genPIF(token: String, pos: Integer)
+pos(token: String, st: SymbolTable): Integer
+getSize(): int
+getElement(index: int): Pair
-pif
-pif
Scanner
-reservedWords: ArrayList<String>
-operators: ArrayList<String>
-separators: ArrayList<String>
+Scanner()
+getTokensFromLine(line: String, lineNumber: int)
+isReservedWord(word: String)
+isOperator(op: String)
+isSeparator(sep: String)
+isIdentifier(id: String)
+isConstant(constant)
+getPIF(): PIF
Pair
-K: String
-V: Integer
+Pair(k: String, v: Integer)
+getK(): String
+setK(k: String)
+getV(): Integer
+setV(v: Integer)