Github Link: https://github.com/pauladam2001/Sem5_FormalLanguagesAndCompilerDesign/tree/master/Lab3

I used a Hash Table for the implementation of the Symbol Table (unique for identifiers and constants). I used an Arr ay of Pairs for solving collisions. After the hash function is called
for an element, if that positions already has an element, I add a new pair of the form Pair <String, Integer> (String be ing the element and Integer a unique counter incremented after each
added element).

For the Symbol Table I have 3 functions:
- getHash - hashes an element and returns the hash % size of the array
- add - adds an element to the symbol table. First we compute its hash, after we check if the element is already prese nt in the symbol table (using the find function). If it's not, we add
  the element in form of a Pair<Element, Counter> and we increase the counter
- find - checks if an element is present in the symbol table. First we compute its hash, then, on the hash position we g o through the pairs. If nothing is found the function returns true,
  or false otherwise

I used an Array of the form <Pair<String, Pair<Integer, Integer>> for the implementation of the Program Internal Fo rm. The key of the "big" Pair is the name of the element and the value is
formed from another Pair, which has the key = the code of the identifier/constant/operator/separator/reservedWord a nd the value = the position of the element in the Symbol Table.

The Scanner has the following functions:
- scan - goes through every line of the program and tokenize each line
- tokenize - called from scan, tokenizes each line
- buildSymbolTableAndProgramInternalForm - called from scan, populates the Symbol Table and the Program Inter nal Form based on the tokenized lines
- writeInFiles - writes the Symbol Table in ST.out and the Program Internal Form in PIF.out