SOURCE CODE: https://github.com/davidalexandru1370/LabLFTC/blob/lab3/Lab3
(available only on the Lab3 branch)

Documentation

- For the symbol table I've used a generic hash table with separate chaining as collision handler. The size parameter from the constructor means the size of the of hashtable (number of cells). The symbol table stores the identifiers, strings and hashtables.
- The hash functions depend on the data type, for integers the hash function simply hash that number, for strings it sums up the ascii codes and hashes the sum. The hash number increases the efficiency of the hash function(mathematically, not asymptotically) if the number is prime.
- The hashtable uses an arrayList of arrayLists and it inserts the object at the index computed by hash function

  Methods:
- Hashtable:

getSize() - returns the size of the hashtable

hash(key: Int) - hash the key by computing key % size

hash(key: String) - hash the sum of ascii codes of key computing sum % ascii

hash(key: Char) - hash the key by computing key % size

computeHashValue(key: T) - compute the hash of the key treating each known data type

insert(key: T) - insert a new value into the hashtable

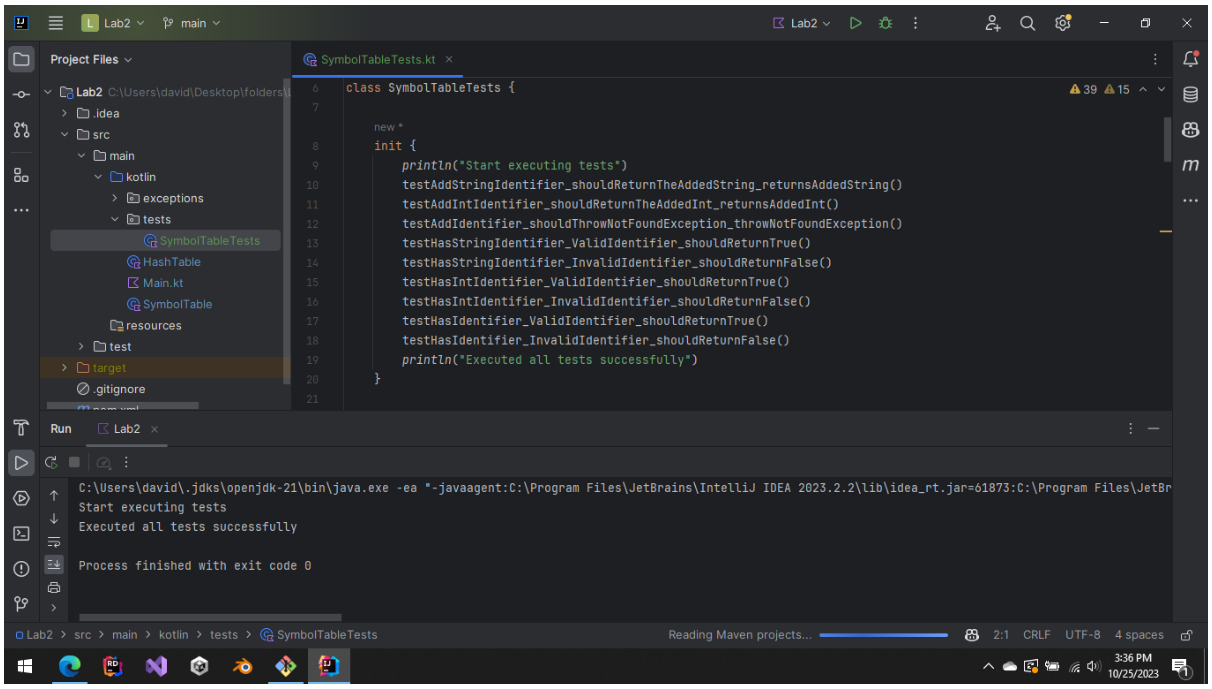
contains(key: T) - returns true if the hash table contains the key, otherwise false

getPosition(key: T) - returns the position of the hashed key in the arrayList representation

- Symbol table

addIntConstant(value: Int) - returns the a pair of integers first one representing the hash value and the second one the index in the array list
addIdentifier(value: String) - returns the a pair of integers first one representing the hash value and the second one the index in the array list
addStringConstant(value: String) - returns the a pair of integers first one representing the hash value and the second one the index in the array list

hasStringIdentifier(string: String) - returns true if the string exists in the string hashtable, otherwise false
hasIntIdentifier(int: Int) - returns true if the integer exists in the integers hashtable, otherwise false
hasIdentifier(identifier: String) - returns true if the integer exists in the identifiers hashtable, otherwise false
getIdentifierPosition(identifier: String) - returns a pair of integers, where the first integer is the hash and the second one the index in the array list
getIntIdentifierPosition(int: Int) - returns a pair of integers, where the first integer is the hash and the second one the index in the array list
getStringIdentifierPosition(string: String) - returns a pair of integers, where the first integer is the hash and the second one the index in the array list
getIntByPosition(position: Pair<Int, Int>) - returns the int by given position
getStringByPosition(position: Pair<Int, Int>) - returns the string by given position
getIdentifierByPosition(position: Pair<Int, Int>) - returns the identifier by given position



```kotlin
class SymbolTableTests {

    init {
        println("Start executing tests")
        testAddStringIdentifier_shouldReturnTheAddedString_returnsAddedString()
        testAddIntIdentifier_shouldReturnTheAddedInt_returnsAddedInt()
        testAddIdentifier_shouldThrowNotFoundException_throwNotFoundException()
        testHasStringIdentifier_ValidIdentifier_shouldReturnTrue()
        testHasStringIdentifier_InvalidIdentifier_shouldReturnFalse()
        testHasIntIdentifier_ValidIdentifier_shouldReturnTrue()
        testHasIntIdentifier_InvalidIdentifier_shouldReturnFalse()
        testHasIdentifier_ValidIdentifier_shouldReturnTrue()
        testHasIdentifier_InvalidIdentifier_shouldReturnFalse()
        println("Executed all tests successfully")
    }

```

```
C:\Users\david\.jdks\openjdk-21\bin\java.exe -ea "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\lib\idea_rt.jar=61873:C:\Program Files\JetBr
Start executing tests
Executed all tests successfully

Process finished with exit code 0
```

Lab3:

Scanner class:

- nextToken method skips all the irelevant info in the code like new lines, comments
- parseNewIdentifier : this method will check if the code match with a new identifier and add that identifier in the symbol table
- parseExistingIdentifier - this method will check if the code match with an existing identifier and add the existing location in the pif. This

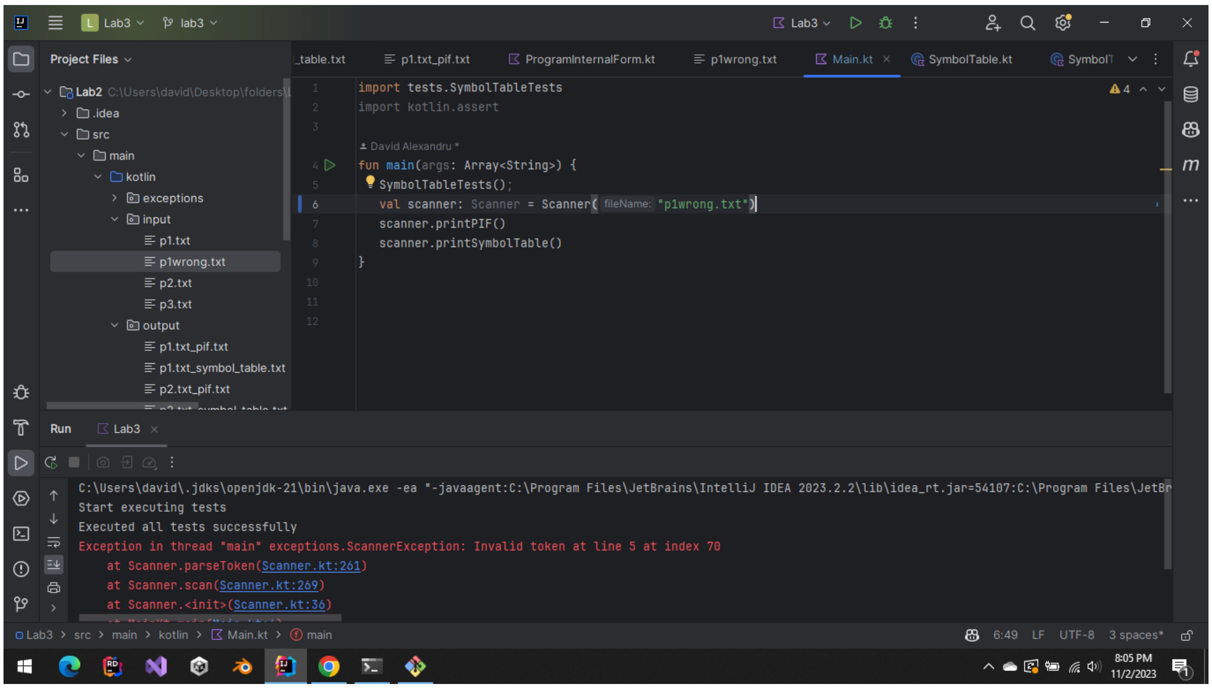
method will throw a Scanner Exception if the name of the variable is found in the separators, reserved words, operators

- parseSymbol - this method will check if the code match with an existing symbol or separator and will add it into pif
- parseReserveWord - this method will check if the code match with an existing reserved word
- parseStringConstant, parseIntConstant, parseCharConstant - Parses a string/int/car constant from the source code and adds it to the program's internal form.

  If a valid string/int/char constant is found, it is added to the program's internal form along with its location in the symbol table.

  Return value: true if a valid string/int/char constant is successfully parsed and added to the program's internal form; false otherwise.

- parseToken - this method will call all the above methods and stop at the first method which find a match
- scan - this method will scan until the last character in the input code
- printPIF - return a string with the PIF
- printSymbolTable - return a string with the Symbol Table



```
import tests.SymbolTableTests
import kotlin.assert

fun main(args: Array<String>) {
   SymbolTableTests();
   val scanner: Scanner = Scanner( fileName: "p1wrong.txt")
   scanner.printPIF()
   scanner.printSymbolTable()
}
```

```
C:\Users\david\.jdks\openjdk-21\bin\java.exe -ea "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\lib\idea_rt.jar=54107:C:\Program Files\JetBr
Start executing tests
Executed all tests successfully
Exception in thread "main" exceptions.ScannerException: Invalid token at line 5 at index 70
    at Scanner.parseToken(Scanner.kt:261)
    at Scanner.scan(Scanner.kt:269)
    at Scanner.<init>(Scanner.kt:36)
```

The error printed in case of error found in p1wrong.txt:

```
//minimum of three numbers, illegal tokens
program{
```

```
begin

   int 23a; //a variable can not start with a number
```