Condrea Adrian
Group 932

Lab 2 - Documentation

Github link: https://github.com/adriancondrea/FLCD_Project

The symbol table is implemented using hash table as data structure. It has two fields: the size and an array of Strings.

```
public class SymbolTable {
    private final String[] hashTable;
    private final int size;

    public SymbolTable(int size) {
        this.size = size;
        this.hashTable = new String[size];
    }
```

Every key is added in the hash table based on the hash value obtained from the hash function. For the hash function, I have used the sum of ascii characters % size (the basic hash function from the course)

```
/**
 * compute the hash value for the given key as the sum of ascii characters, modulo size
 *
 * @param key - the string to hash
 * @return the hash value for given key
 */
private int hashFunction(String key) {
    int hash = 0;
    for (int i = 0; i < key.length(); i++) {
        char c = key.charAt(i);
        hash += c;
    }
    return hash % size;
}
```

The SymbolTable has the addElement function defined:

```java
/**
 * if the element already belongs to the symbol table, return its position. Otherwise, add it and then return its position
 * collisions (two keys having the same hash value) are handled by open addressing (all keys are stored in the same hash table)
 * with linear probing (search for the next available location) with probing interval = 1.
 *
 * @param key - the element to add to the hash table
 * @return the position on which the element is in hash table
 */
public int addElement(String key) {
    int hash = hashFunction(key);

    while (hashTable[hash] != null) {
        if (hashTable[hash].equals(key)) {
            System.out.println("Element already added on position " + hash);
            return hash;
        }
        hash = (hash + 1) % size;
    }
    hashTable[hash] = key;
    System.out.println("New element added on position " + hash);
    return hash;
}
```