Documentation

Github:
https://github.com/Aurelian-Iancu/UBB-Computer-Science/tree/main/Semester5/Formal%20Languages%20and%20Compiler%20Design/Lab2

The symbol table is composed from 2 separate hash tables, one for identifiers and one for the constants. The hash table is solutioning the collisions problem using the separate chaining method which is done by having a list for each hash value of the table. Each element fromt he symbol table has as position a pair of indices, the first one being the index of the list in which the element is stored and the socnd one being its position in the list which exists at the position where its hash suggests. The hash function is value modulo the size of the list, for integer values, and the sum of the ASCII codes of the characters modulo the size of the list, for string constants/identifiers. The implementation of the hash table is generic.

Operations:
1. Hash table
- hash(int key) => the hash function for integer numbers.

Complexity analysis:
Best: O(1)
Worst: O(1)
Average: O(1)
- hash(String key) => the hash function for strings

Complexity analysis:
Best: O(1)
Worst: O(1)
Average: O(1)
- getCapacity() => the getter for the capacity of the hash table

Complexity analysis:
Best: O(1)
Worst: O(1)
Average: O(1)
- contains(T key) => returns true if the given key is in the hashtable or not

Complexity analysis:
Best: O(1)
Worst: O(n)
Average: O(n)
- getHashValue(T key) => return the corresponding position in the symbol table according to the type of the parameter ‘key’ (string or integer)

Complexity analysis:
Best: O(1)

Worst: O(1)
Average: O(1)

- toString()=> overrides the toString function and pretty prints the hashTable

Complexity analysis:
Best: O(1)
Worst: O(1)
Average: O(1)

- getPosition(T key) => returns a Pair consisting of the value of the hash and the position of our element in the list present at the hashed value.

Complexity analysis:
Best: O(1)
Worst: O(n)
Average: O(n)

- add(T key) => add the key to the hash table and return its position if the operations is successful; otherwise, throw an exception

Complexity analysis:
Best: O(1)
Worst: O(n)
Average: O(n)

2. Symbol table:

- addIdentifier(String name) => adds an Identifier to the identifiers hash table

Best: O(1)
Worst: O(n)
Average: O(n)

- addConstant(String name) => adds a constant to the constants hash table

Best: O(1)
Worst: O(n)
Average: O(n)

-hasIdentifier(String name) => returns true if the identifier is in the identifiers hash table
Best: O(1)
Worst: O(n)
Average: O(n)

-hasConstant(String name) => returns true if the constant is in the constants hash table
Best: O(1)
Worst: O(n)
Average: O(n)

-getPositionIdentifier(String name) => returns the pair position of an identifier in the identifiers hash table
Best: O(1)
Worst: O(n)
Average: O(n)

-getPositionConstant(String name) => returns the pair position of a constant in the constants hash table
Best: O(1)

Worst: O(n)
Average: O(n)
- toString()=> overrides the toString function and pretty prints the symbol table
Complexity analysis:
Best: O(1)
Worst: O(1)
Average: O(1)