932/2 David Catalin-Ioan

Lab2

The Symbol Table was implemented using a hash table. The hashMap field inside the class is an array of java template arraylists. The reason why we use arraylists is in order to place elements with the same hash function result inside the same container. This is a chaining conflict resolution technique, which is also one of the most popular and widely used techniques. We initialize the array using a given capacity inside the constructor. The threshhold represents the percentage up to which we want to fill the hash table (given by the size/capacity ratio) before we resize it by essentially doubling the capacity and re-hashing every element inside the hashMap.

The method “add” adds a new identifier to the hashMap, unless it had already been added before. It also checks whether we go above the threshhold, in which case we resize the hashMap.

The method “resize” creates a new hashMap, with double the capacity as the current one, and adds (re-hashes) every element from the current hashMap to the new one. This new hashMap will replace the current one.

The method “find” tries to find an identifier. If it is inside the Symbol table, it returns an integer array containing 2 values. The first value represents the hash value result and the second one represents the index inside the arraylist where we found the identifier. If the identifier is not found inside the Symbol table, the method returns an array containing 2 integers, both with values -1.

The method “hash” hashes a string into an integer that is between 0 (inclusive) and capacity (exclusive). It does that by adding the ascii codes of all the characters inside the identifier and computing the % of that sum with the capacity of the hashMap.

Link to Github repo for Lab2: https://github.com/Catalin-David/FLCD/tree/main

Link to Github repo for Lab1b: https://github.com/Catalin-David/FLCD/tree/lab1b