<https://github.com/taniasasaran/UBB-Computer-Science/tree/main/Semester-5/FLCD>

For my Symbol Table, I chose to implement one Hash Table which can be used in both cases: 1 for all identifiers and constants altogether; and 2 for each one of the identifiers and constants.

The collision technique used for my hash table is separate chaining. The Hash Table uses two types of lists: one which stores all elements which have the same hash value and one which stores all such lists.

I have also defined the Pair as a data structure, composed of 2 integer numbers, representing the pair of positions an element is on (hash value first, position in the hash list second).

getSize() method returns the size of the Hash Table

getNumberOfElements() method returns the number of elements in the table, regardless of its size.

getPosition(String elem) method returns the position of the element, which is computed the following way: we compute the hash for that element, we check whether the list from that is not empty, if it is empty we return null, otherwise we take the list from that position, parse it and when we reach our terminal, we create a Pair with the hash and terminal’s index from the list.

get(Pair pos) method will return the terminal from the Symbol Table based on its position.

hash(String key) – represents the hash function and the approach used is to compute the sum of the characters modulo size.

contains (String elem) – returns true or false, depending if the element belongs to the Symbol Table or not

add(String elem) – adds an element to the Symbol Table and returns its position(when adding an existing element it doesn’t add it again and it returns the position of the existing element).