krisztinahorvath/Formal-Languages-and-Compiler-Design (github.com)

Documentation

The Symbol Table is composed of one hash table with separate chaining. The hash table is represented as a list and every position is another list, to be able to store values that hash to the same position. An element of the symbol table has as position a pair of indices, the first one is the index of the bucket/inner list in which the element is stored, and the second one is the actual position inside the list. The hash function is value modulo the number of buckets in the list, the value is computed by a built-in function. The implementation of the hash table is generic.

Hash table:

- hashFunction(key: T): int – returns the position in the Symbol table of the list in which the value will be added
- add(key: T): (int, int) – adds the key to the hash table and returns its position on success, otherwise returns null
- contains(key: T): boolean – returns true/false if the key is in the hash table or not
- getPosition(key: T): (int, int) – returns the position in the hash table of the given key
- toString() (overridden method) – returns the string representation of the hash table
- for all the above functions the average-case complexity is O(1), since we can access directly the elements by knowing the position they hash to and we assume that there are no collisions and the hash function is well-distributed. The worst-case complexity for adding and searching is O(n), when all keys hash to the same bucket and there are n keys in it

Symbol table:

- has one hash tables
- addSymbol(identifier: String): (int, int) – adds a symbol and returns its position in the symbol table
- hasSymbol(identifier: String): boolean – returns true if the symbol is in the symbol table, false otherwise
- getSymbolPosition(identifier: String): (int, int) – returns the position of the symbol in the symbol table
- toString() (overridden method) – returns the string representation of the symbol table and its hash tables
- since all the above functions call and return hash table functions, their average-case complexity is O(1) and worst-case complexity for adding and searching a symbol is O(n) using the same reasoning