**Lab 2**

**GitHub:**

<https://github.com/923-Iliesi-Catrinel/Formal-Languages-and-Compiler-Design/tree/main/Lab%202>

I chose to implement the Hash Table data structure in C++. The Hash Table uses linear probing and double hashing to handle collisions. It also resizes the table when the load factor exceeds 0.75. I’m using size\_t instead of int as all of the keys are unsigned, and also for better compatibility on different systems.

The Hash Table contains the following:

* Members:
  + std::vector<HashNode<K, V>> table: A vector of HashNode entries, which store the key, value and a state (EMPTY initially, FILLED if populated and DELETED if deleted).
  + size\_t capacity: The current capacity of the hash table.
  + size\_t size: The number of FILLED entries in the hash table.
  + size\_t non\_deleted\_size: Number of entries that are either FILLED or DELETED. This member is used for shrinking the table when there are too many DELETED elements.
* Operations:
  + size\_t hash1(const K& key): Primary hash function that calculates the hash value for a given key. It uses the std::hash function to compute a hash code and then takes the modulo of the capacity to determine the position in the table. I chose std::hash because it handles the type of the key (int, string etc.) correctly.
  + size\_t hash2(const K& key): The secondary hash function is used for double hashing to solve collisions. I’m using it as a step for linear probing to uniformly distribute the keys.
  + size\_t probe(size\_t current\_position, size\_t step): Performs linear probing to resolve collisions. It calculates the next available position after a collision using double hashing.
  + void resize(): Doubles the capacity of the hash table and rehashes the existing elements when the load factor exceeds 0.75.
  + HashTable(): Constructor that initializes the hash table with the default capacity (16).
  + void add(const K& key, const V& value): Adds a key-value pair to the hash table. It handles collisions using linear probing and resizes the table if necessary. Complexity: BC = AC = O(1), WC = O(n).
  + void remove(const K& key): Removes a key-value pair from the hash table by marking the slot as deleted without deleting the slot, as this is needed for linear probing. Complexity: BC = AC = O(1), WC = O(n).
  + std::optional<V> get(const K& key): Fetches the value associated with the provided key. If the key is not existent it returns an empty optional ({}). Complexity: BC = AC = O(1), WC = O(n).
  + void clear(): Clears all entries of the hash table.
  + size\_t getSize(): Returns the number of elements in the hash table.
  + size\_t getCapacity(): Returns the capacity of the hash table.
  + bool isEmpty(): Returns true if HashTable is empty.
  + friend std::ostream& operator<<(std::ostream& os, const HashTable<K, V>& hashTable): Overloaded stream insertion operator used to output of the Hash Table in the console.

The Symbol Table is implemented generically, as a single instance, unique for identifiers and constants.

The Symbol Table contains the following:

* Members:
  + HashTable<K, size\_t> table: Hash Table for constants and identifiers.
  + size\_t table\_index: Auto-increment index for table entries.
* Operations:
  + SymbolTable(): Initializes the SymbolTable object with an empty HashTable and sets the table\_index to 0.
  + void add(const K& key): Adds a new key-value pair to the SymbolTable.
  + void remove(const K& key): Removes a key from the SymbolTable.
  + std::optional<size\_t> get(const K& key): Retrieves the value associated with a key from the SymbolTable.
  + size\_t getSize(): Gets the size of the SymbolTable, the number of key-value pairs in the table.
  + bool isEmpty(): Returns true if SymbolTable is empty.
  + void clear(): Clears all entries of the SymbolTable, making it empty.
  + friend std::ostream& operator<<(std::ostream& os, const SymbolTable<K>& table): Overloaded stream insertion operator used to output of the Symbol Table in the console.