**Software Development: Data Structures and Performance (SCQF Level 8)**

**HL9T 35**

**Assessment**

**Outcome 1 Part 1 of 2**

#### Creative Industries

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Student** | | Kamil Milej | | | | | | |
| **I.D.** | | 2264629 | | | | | | |
| **Date** | | 02/05/2025 | | | | | | |
|  | **Pass** | |  | **Fail** |  | **Remediation** |  |  |
| **Tutor** | |  | | | | | | |

#### Creative Industries

***Hash Table Implementation Documentation***

* *Algorithm Description*

In our task we used ADT (Abstract Data Types) which implement what method should be implemented by our program. The main logic is concentrated in hashtable.h , where I declare a hashtable with a std::string table[TABLE\_SIZE]; with a static value of ten elements, static const int TABLE\_SIZE = 10;. Also in this file hashtable.h there are main methods which I use like:

Init() to initialise a new array with 10 empty strings.

Hash() count sum of ASCII and divide by 10 and take modulo.

Insert() count index in hash(key) if the place is empty, save the string in this place, but if not empty, then move the next index by using linear probing.

Search() count hash for a key and search if for this hash it is this searching String, if not, check linear probing and stop only if find a correct key, find an empty space or check all the table.

printable() is a method which prints the indexes 0–9 and what is in each place. Shows empty if the field is empty.

Also, I used two functions outside hashtable.cpp in main.cpp:

insertDataSet() has a reference to a hashtable object and to the data which will be added. For each element count hash, count how many step needed, call table.insert(key) to put into array and at the end return how many steps count.

searchDataSet() take same parameters like in insertdataSet() for each String count hash(), try to find this String using table.search() and count how many steps and also return steps.

* *Data Structure used*

The data structure we used for this task is a hash table with linear probing, which helps to resolve a collision when putting an element in the array. We put in static fixed-size array, which can't change and we make ten empty elements. A hash table, also called a hash map, stores key-value pairs or just keys, like in our tasks and uses a hash function to fit the index using the sum of the ASCII value of the string modulo. Example. ASCII value / 10 and we take modulo value (%) from this and show a place in array. Also, linear probing is used when the place after counting modulo is full, and we need to search the next place to put the value, most of the time just the next index in the array.

* *Test complexity analysis*

To evaluate the performance of our hash table implementation, we made three types of tests:  
1. Optimal Case Dataset where data was chosen randomly like in normal life.  
2. Worst-case Dataset worstCaseSet = {"aaa", "cdf", "cee", "cff", "def"}; have 3 time same modulo value 1/1/1/3/3 and all need to be moved.  
Overfilled Dataset which has more then 10 elements to show what happen on bigger array operation and pushing system to past capacity. Also demonstrate how the algorithm handle insertion if full capacity, just try to put at all array index and if not space just take another element and do the same.

* *Time Complexity Analysis*

Hash table operations depend on the load factor (how full the table is) and the distribution of the hash function. What scenario we have :

*Best Case* (O(1)): No collision. Key is inserted/searched at the first computed index.  
*Average Case:* Linear probing might be needed a few times (depending on load factor).  
*Worst Case (O(n)):* Table is full or clustering causes a key to traverse almost the entire array.

* *Space Complexity*

It measures the amount of memory required by the algorithm, excluding the memory used by the input data. In this task the space complexity of this hash table implementation is O(n), where n is the size of the table (TABLE\_SIZE = 10). This is because we declare a static fixed-size of array with value 10. Memory is allocated for all 10 slots at the start, regardless of how many keys are actually insered into out array.