**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 concentrate in hashtable.h where I declare hashtable std::string table[TABLE\_SIZE]; with static value of ten element static const int TABLE\_SIZE = 10;.

* *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*
* *Space Complexity*
* *Source Code*