**ITE7107: Laboratory Exercise 003 Answers**

C++ Maps and Hash Tables

Note: Source codes used for this exercise, together with this answer sheet are uploaded in the GitHub repository: <https://github.com/rvillamangca/AMA-ITE7107-DataSturctureAlgorithms>. All program/s are written in C++.

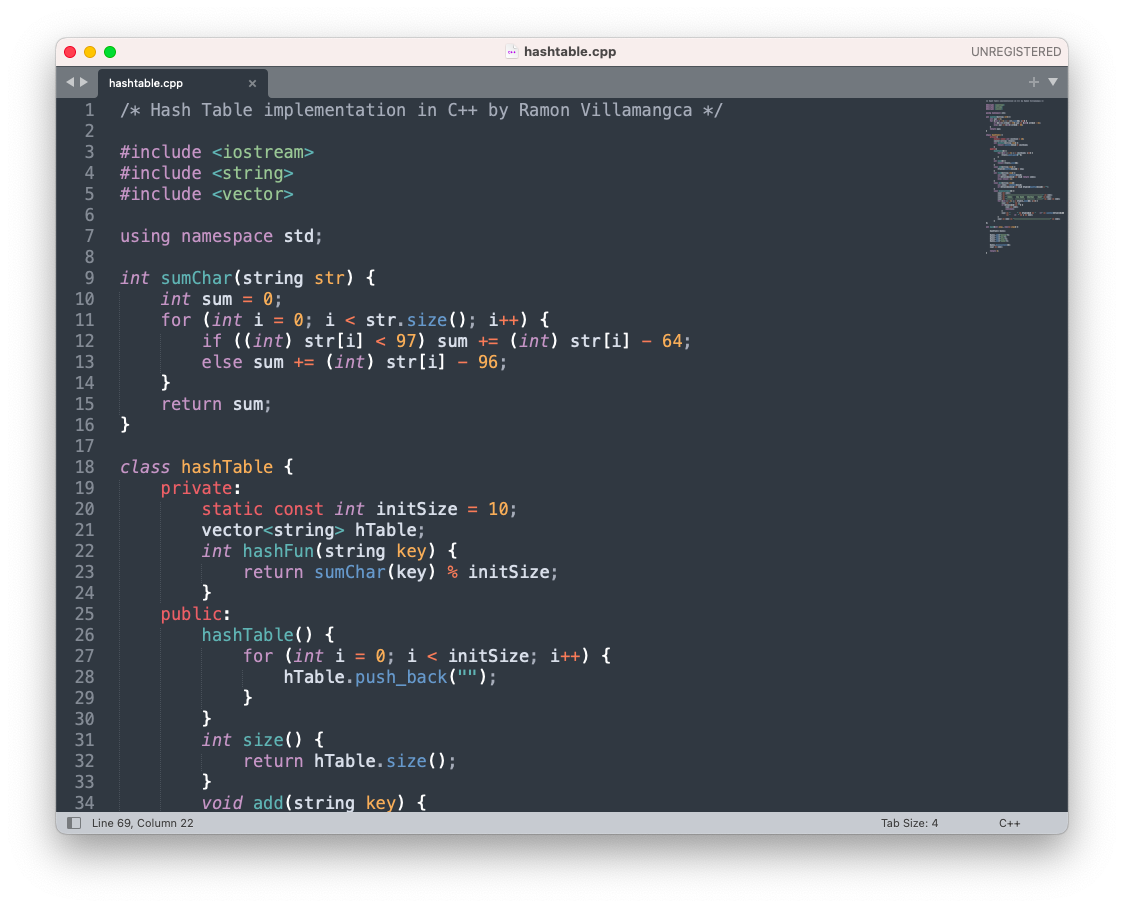
Question 1: **Consider storing the names: George, Amy, Alan and Sandy in a hashtable of size 10 using the hash function:**

**hash(name) = sum of characters mod 10**

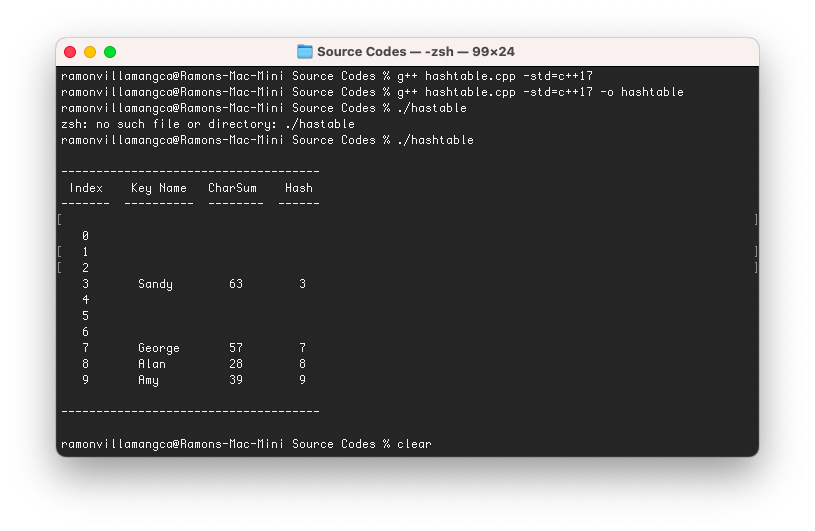
**where a=A=1, b=B=2, etc. Draw the hashtable that would be produced.**

Answer:

An implementation of a Hash Table written in C++, is shown below. The full code listing is in the attached “Source Code.zip”.



Using this implementation yields the following output on screen:



Question 2: **Consider hashing keys that are strings containing only lower-case letters. The hash function that will be used computes the product of the integer values of the characters in a key, using the scheme: a=0, b=1, c=2, etc. Why is this scheme not as good as using: a=1, b=2, etc.?**

Answer:

Let:

Scheme 1: Using “product of character values” as Hash Function with

a=0, b=1, c=2, etc.

Scheme 2: Using “product of character values” as Hash Function with

a=1, b=2, c=3, etc.

In reality, both hashing schemes are undesirable. A good hashing scheme would avoid collision as much as possible. For example in Scheme 1, the word “delivered” and “reviled” will both have the same hash! There would definitely many such collisions. Scheme 2, however, is the worse scheme of the two. Because of the assignment of “0” to the letter “a”, all words containing “a” will have a hash value of “0” (zero multiplied by any number is still zero). Remember, that “a” is the second most used letter in the English alphabet, next to “e” (see <https://en.wikipedia.org/wiki/Letter_frequency>). This means that the number of collision vying the index “0” of the Hash Table, will be greatly compounded.