ALGORITHMS AND DATA STRUCTURES 2

TOPIC 5: HASHING
FORMATIVE EXERCISE

In this activity, you will work **individually** to implement a hash table with linear probing.

PART 1: YOUR TASK

In this formative exercise, your task is to implement a hash table with **linear probing**. The hash function to implement is as follows:

$$h(k)=(a*k+c) \bmod m$$

where coefficients a, c and m are positive integer numbers and m is the number of buckets of the hash table. The client of the hash table will specify these parameters.

This exercise will be automatically graded. Thus, you are provided with two files: HashTable.cpp and HashTable.hpp.

Your task is to complete the code that makes the methods in that skeleton code operative. **You must not change the prototypes of the methods**. You can add other methods and variables if you want.

PART 2: THE METHODS

Please, look at the content of the skeleton files you are provided with for this formative exercise (HashTable.cpp and HashTable.hpp). You can see there are seven methods you must implement in the .cpp file:

- HashTable: This is a constructor that initialises the values of *a*, *c* and *m* (*a* takes the value of _*a*, *c* the value of _*c* and *m* the value of _*m*) and allocates memory space (to store *m* integer values) to the hash table *buckets*.

- ~HashTable: This method deletes the hash table.

- insert: This function is in charge of inserting strictly positive numbers into the hash table, **using linear probing for collision resolution**. The number to insert is stored in the variable *key* (the input argument of the function). If the load factor of the hash table is already 1 when a new number is going to be inserted, you have to apply extend and rehash. You choose the value of the extension factor of the hash table.

- extend: This method increases the size of the hash table. To do so, you must create a new (bigger) array temporarily storing the content of the hash table. Then, you increase the size of *buckets* (using new) and rehash the contents of the temporary array into *buckets*.

- find: This function searches for the number *key* (the input argument of the function) in the hash table. If the number is found, it returns true. Otherwise, it must return false.

- remove: This function removes the number *key* (the input argument of the function) from the hash table. Remember that, because of linear probing, a number present in the hash table might not be in its natural position.

- loadFactor: This function returns, as a double, the fraction of total hash buckets that are occupied.

PART 3: SUBMISSION

When you want to submit your work, please compress both files (.cpp and .hpp) into a zip file. The compressed file can have any name.

In the My submission tab, press the blue ‘Create Submission’ button. Below the text ‘Upload Files and Submit’, press the blue ‘HashTables’ button, select the file and click ‘Submit’.

Upon submitting, you will see the date of the submission in grey. It might take a while for the system to grade your code. You can wait for the grade or log out and come back later.

Once the grade is ready, you can click on the submission tab and then on ‘Show grader output’ to check on eventual errors.

You can upload your submission as many times as you want. Your highest score will be considered.

You can get familiar with the system by simply uploading the skeleton code provided for you.