**ALGORITHMS & DATA STRUCTURES**

**Lahcen Ouarbya**

**TOPIC 4: HASHING**

**LAB. SUBMISSION**

In this activity, you will work **individually** implementing a hash table with linear probing. Remember that you can discuss ideas with your classmates, but you cannot see other’s work neither others can see your individual work.

After finishing the implementation of your hash table, you must upload **a single file** with your code using the “Hash Tables Lab Submission” link available at learn.gold, Week 9. Your code will be graded immediately.

The official deadline for this lab submission is 3rd **January 2021, 11pm** .

|  |
| --- |
| **PART 1: YOUR TASK** |

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

h(k)=(a\*k+c) mod 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.

Because this lab submission will be automatically graded, you are provided with a skeleton code:

* a single file (HashTable.java) for Java programmers
* two files (HashTable.cpp and HashTable.hpp) for C++ programmers.

Your task is completing the code that makes the methods in that skeleton code operative. **You must not change the prototypes of the methods**. If you do that, your code will not pass the automatic tests and you will not get a good mark. You can add other methods if you want.

|  |
| --- |
| **PART 2a: JAVA PROGRAMMERS** |

Please, look at the content of the skeleton file you are provided for this lab submission (HashTable.java). You can see there are 6 methods you must implement:

* 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*.
* 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*. 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 & rehash. You choose the value of the extension factor.
* 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* in the hash table. If the number is found, it returns true. Otherwise, it must return false.
* remove: This function removes the number key 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.

**Notice:** You must keep the *buckets* member variable public and use values of the hash function *h* as indexes into that array. In a real implementation buckets would be declared as private, but it needs to be exposed for testing purposes. (You may of course add other private member variables).

|  |
| --- |
| **PART 2b: C++ PROGRAMMERS** |

Please, look at the content of the skeleton files you are provided for this lab submission (HashTable.cpp & HashTable.hpp). You can see there 7 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*. 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 & rehash. You choose the value of the extension factor.
* 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* in the hash table. If the number is found, it return true. Otherwise, it must return false.
* remove: This function removes the number key 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 float, the fraction of total hash buckets that are occupied.

|  |
| --- |
| **PART 3: Automatically graded system** |

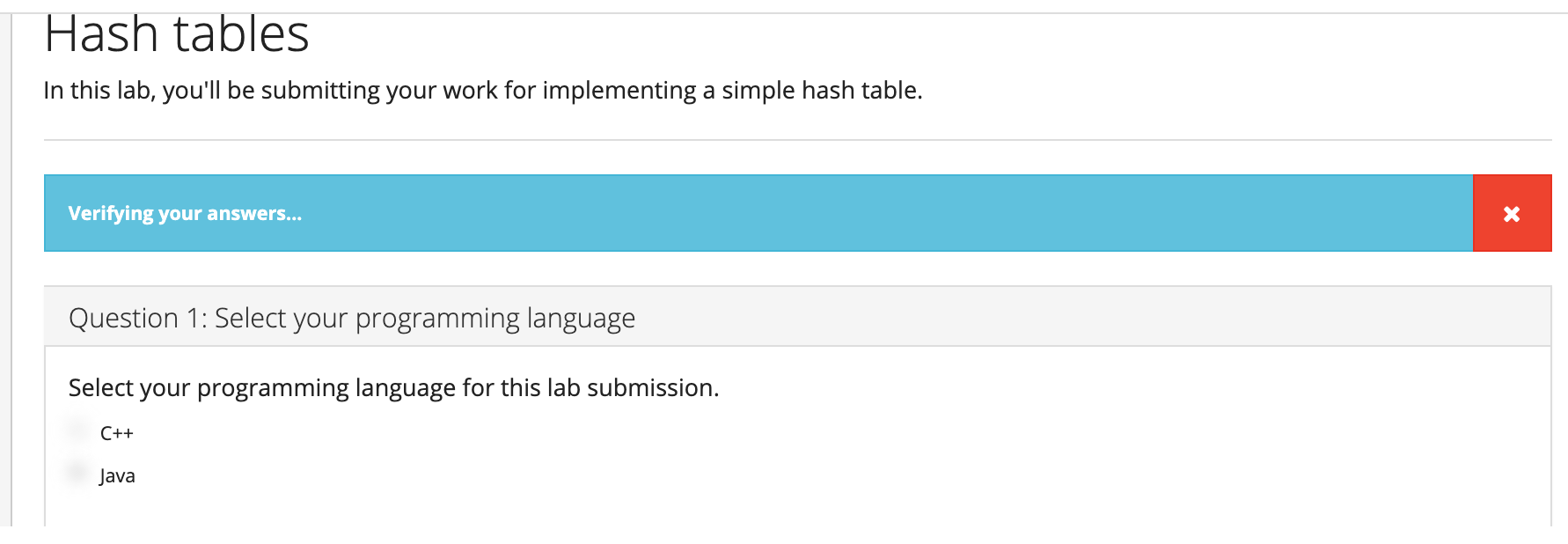
For this lab submission your code will automatically graded. As soon as you upload your submission, your code will be analysed and a report will be displayed on the screen. The report includes your mark in the assignment as well as any errors you still might need to correct.

**You can upload your submission as many times as you want**, without any block window. Your highest score will be considered.

You can get familiar with the system simply uploading the skeleton code provided to you. You must:

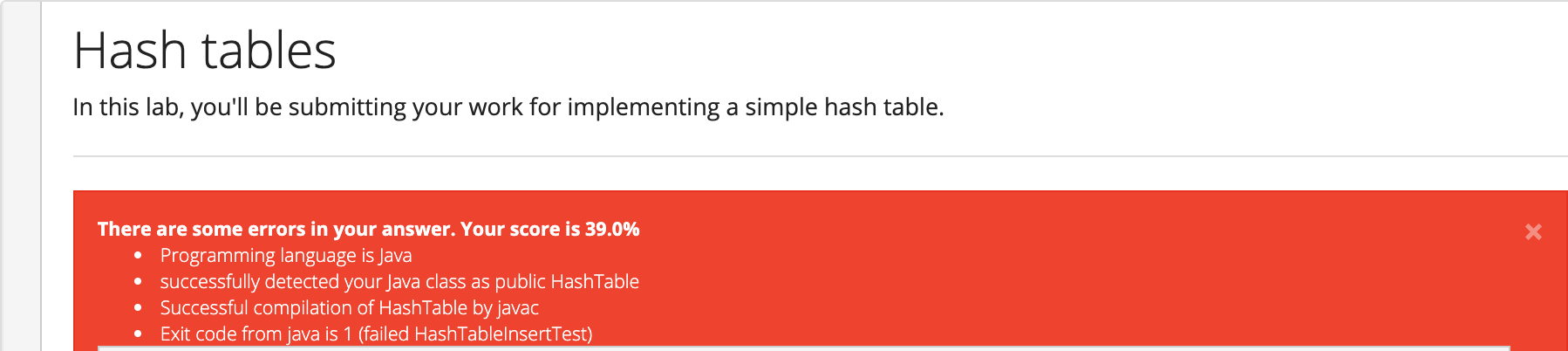
* select the language of your submission (Question 1)
* upload the file (Question 2). In the case of C++ programmers, you must compress the .cpp and .hpp files and upload the resulting .zip file
* answer the last personalised question (Question 3). You can omit Question 3 (you will not get full marks, but you can still check your code).

As soon as you press the button “Submit” you will see the message “Verifying your answers….” as shown in Figure 1.



**Figure 1. Message immediately after uploading your code**

Because none of the methods is implemented yet in the skeleton code, you will be next shown a red screen (signalising a mark under 40) with the following message: “There are some errors in your answer. Your score is 39%”.



**Figure 2. Report showing your mark**

Below the red screen, you will receive a report with the tests that your code failed. An example is shown in Figure 3.



**Figure 3. Error messages**

In Figure 3 the system is informing you that you failed the tests: *testInsertCollision*, *testInsertOne* and *testInsertSequence*. The names are self-explained.

Figure 4 shows you the screen when you upload a code that is almost ready (function extend() has not been implemented):



**Figure 4. Report after uploading an almost ready code**

Immediately below, you can see a message regarding the error in the code (Figure 5). In this case, the test that has failed is testInsertLots that insert more elements in the table than buckets available. Since the extend() function has not been implemented, the system throws this error.



**Figure 5. Error report due to function extend() not being implemented**