CS 211 – Data Structures
Lab for Week #10

For this lab exercise, you will work either in 2-person teams or individually. You are HIGHLY ENCOURAGED to work in 2-person teams! The point here is to have teams discuss the lab and offer each other support in making sure the IDE of your choice is working.

Lab Exercise

This lab is to practice with a Hash Table that contains **int** values and uses probing. Your code will NOT make use of any pre-made hash table libraries or classes – you are to build the methods yourself, using a dynamically-allocated array to hold the table's values.

Define a class named **HashTable** that has the following methods. You may assume that only non-negative integer values will be placed into the **HashTable**.
(**T** is the data type of the values in the table – it's currently set to **unsigned int** using a **typedef** statement)

- Define a private method (not available to users of the class!) named **hashFunc** that performs the hashing function for the **HashTable**. It should expect the value to be inserted in the table and return a value from **0** to **tableSize – 1**, where **tableSize** is the size of the **HashTable**. My recommended coding (just for fun, and the way it will be tested) is to have it return return the square of the value modulo size, or **(value * value) % tableSize**, but you are free to try other hasing functions if you'd like.
  ```
  int HashTable::hashFunc(T value) const
  ```
- Two other private methods are pre-written for you:
  - **nextIndex** calculates the next index in the hash table in the probing sequence. As currently written, it uses Linear Probing. It's very useful to make this a separate function, as it will allow for different probing sequences! NOTE: This method uses Call By Reference to mutate the variable passed to it!
  - **bucketAvailable** returns **true** if the bucket is available during an insert operation (do NOT use for search or delete!).
- Constructors – a zero-argument constructor, and one that expects a **size** of the table, which is the number of items the **HashTable** can hold. The constructor should dynamically allocate the array (or use a vector) of type **T** of the appropriate size, and the **bool** array of the same size, initialized to all **false**, to keep track of whether a value has been deleted from the table (remember, in probing, you don't empty the bucket when you delete a value, you just mark it as "deleted"). Only positive integers should be inserted into the table in this lab, so you can initialize the table's values to **0**.
  ```
  HashTable::HashTable(int size)
  ```
- A method to insert a value into the **HashTable** that returns **true** if the value is successfully inserted and **false** if there is no room left in the **HashTable** for the value to be stored. the method should search the **HashTable** for an available spot (and use **nextIndex** as appropriate) until it either finds an empty bucket or until the entire array has been checked for an open bucket. Don't forget to check the **bool** array for deleted values, as a bucket containing a deleted value can be reclaimed for

inserting the new value (and set the **bool** back to **false** when you do insert!).

```
bool HashTable::insertValue(T value)
```

- A method to delete a value from the **HashTable** that returns **true** if the value is successfully found and deleted and **false** if the value is not found in the table. Remember, you don't overwrite the value in the table, as that may affect future probing searches for a value later. You should instead use the **bool** array to mark the bucket in the **HashTable** as having been deleted.

  ```
  bool HashTable::deleteValue(T value)
  ```

- A method to search for a value in the **HashTable** that returns **true** if the value is successfully found and **false** if the value is not found. It should perform a probe search using **nextIndex** as appropriate.

  ```
  bool HashTable::searchValue(T value) const
  ```

- A method to print the contents of the **HashTable**, putting all the values on the same line. Print an **X** for any position that has no value currently stored (either never used or marked as "deleted"). For example, your method could print to the screen: **Table size = 7, contents = `X 43 X X 30 X 6`**

  ```
  void HashTable::printTable() const
  ```

- At the top of **each** file, document these files as follows based on your 2-person team:

  ```
  # CS 211 Fall 2024 – Week 10 Lab
  # <Student_Name1> and <Student_Name2>
  ```

  Substitute your own names for the **`<Student_Name>`** placeholders above.

ONCE FINISHED, ALL STUDENTS SHOULD SUBMIT THE HashTable FILES VIA CANVAS.
SUBMIT THE HashTable.h FILE, EVEN IF NO CHANGES WERE MADE.
You should share copies of the same files to ALL team members for submission.
Use the Multiple File Submission tool – do not Zip them or make multiple submissions!