Data Structures and Algorithms Lab

Instructions

Work on this lab individually. You can use your books, notes, handouts etc. but you are not allowed to borrow anything from your peer student.

Marking Criteria

Show your work to the instructor before leaving the lab to get some or full credit.

What you have to do

Implement a class **HashTable** for storing students' records

```
struct Student
{
	string name;
	int marks;
};

class HashTable
{
private:
	Student *table;			//Dynamic array of Student to hold records
	int size;			//Total number of slots in the table
	int curSize;			//Current number of records present in the table

public:
	HashTable (int size);		//Constructor, store -1 (marks) and "EMPTY" (name) to indicate free
					//location in the HashTable
	~HashTable ();			//Destructor
	bool isEmpty ();		//Checks whether hash table is empty or not
	bool isFull ();			//Checks whether hash table is full or not
	double loadFactor ();		//Calculates & returns the load factor of the
					//hash table (curSize/size)
};
```

Note that the constructor of the **HashTable** class will allocate the array (table) such that it can contain up to **size** records.

In order to **insert** or **search** records in the **hash table**, you should use a **hash function** which takes the **MOD** of **marks** by **size** (which is the table size).

You are required to implement the following member functions of the **HashTable** class:

`bool insert (Student record)`

This function will use the above-mentioned **hash function** to determine the location at which **"record"** can be inserted in the hash table. If that location is already occupied (a collision) then this function should use **linear probing (*with increment of 1*)** to resolve that collision (i.e. it should look at the indices after that location, one by one, to search for an empty slot). This function should return **true**, if eventually an empty slot is found and **"record"** is stored there. If no empty slot is found, then this function should return **false**.

`bool search (Student record)`

This function will search for the given **"record (marks, name)"** in the hash table. It will accomplish this by using the **above-mentioned hash function** and **linear probing**. If the **"record"** is found then this function should return **true**. Otherwise, it should return **false**.

`bool remove (Student record)`

This function will try to **remove** the given **"record (marks, name)"** from the hash table. This function should return **true**, if the **"record"** is found and removed. And it should return **false** if the given **"record"** is not found in the table.

`void display ()`

This function will **display** the contents of the hash table on screen, **index by index**. For indices which are **empty**, this function should display the word **"EMPTY"**.

Also, write a **menu-based driver function** to illustrate the working of different functions of the **HashTable** class. The driver program should, first of all, ask the user to enter the **size** of the table. After that it should display the menu to the user.

```
Enter the size of the hash table: 11

1. Insert a record
2. Search for a record
3. Remove a record
4. Display the table
5. Display the load factor of the table
6. Exit

Enter your choice:
```

☺ ☺ ☺ **BEST OF LUCK** ☺ ☺ ☺