Question 1: Develop a template-based HashMap class that stores a <key, value> pair. In this hash map, the hash map itself will not have any hash function. It will always assume that each key passed to it has a member function named "hashCode()" that returns the hash code of that key. The hash map will use quadratic probing to resolve hash collisions. The load factor will never be allowed to exceed 0.75. If load factor becomes equal to 0.75, then double the capacity of array and rehash the elements from previous array into the new array. The outlook of your implementation will be something like:

```
template<typename k, typename v>
class HashItem
{
		k key;
		v value;
		int state;
};

template<typename k, typename v>
class HashMap
{
		HashItem<k, v>* arr;
		int capacity;
		int totalElements;
		//add any required private function

public:
		void insert(k key, v value)
		{
				int hashCode = key.hashCode();
				//now apply a compression function and then insert the key, value pair.
		}
		//add the remaining required public functions
};
```

Now, let's say we want to store keys as int and values as strings in this hash map. Our hash map object will be of type: **HashMap<int, string> hashObj;.** The issue with this type of hash map object is that our hash map requires the key to have a function hashCode(), but int is a primitive datatype and does not contain any function named hashCode(). So we create a wrapper struct/class for our key as follows:

```
struct Integer
{
		int integerNum;
		int hashCode()
		{
				int hashCode = key << 1 | key >> 5;  //it's the bitwise OR between the key shifted 1 time
left and the same key shifted 5 times right.
		}
};
```

Now, the HashMap object for storing int keys will become: **HashMap<Integer, string> hashObj;**, i.e. the datatype of keys in hashmap will become Integer.

You are required to develop at least the following functions for the HashMap class:

1. A default constructor
2. An Insert function to insert a <key, value> pair.
3. A remove function to remove a <key,value> pair. The user will pass the key to this function.
4. A get function that returns a pointer to the corresponding value of the key passed as parameter. If the key does not exist, return nullptr.
5. An overloaded [] operator which takes a key as parameter, and works like get function
6. A hashKey function which returns true if the given key exists, else it returns false.
7. A resize function that will double the capacity of the array and rehash the old keys from old array to the new array.
8. A destructor.

Question 2:

Now develop another HashMap class that works just like the above implementation. However it uses separate chaining to resolve hash collisions. In this case, your hash item class implementation will become:

```
template<typename k, typename v>
class HashItem
{
        list<k, v> items;
};
```

Implement the same member functions as you implemented in question 1. For list class, you can use any linked list that you have implemented in the lab or the implementation available on google classroom.

Question 3:

Now use the hash map class created in the first question to store student roll numbers and their cgpas. Roll number will be the key and in the form of a string, i.e., 19L-1111. Cgpa will be the value and in the form of a float. You will need to define the following wrapper struct/class for storing roll numbers and using them as keys in the hash map class.

```
struct WrapperString
{
        string stringVal;
        int hashCode()
        { //use a polynomial hash function to find the hash code of the string in stringVal;
        }
}
```

So the hash map object will be: HashMap<WrapperString, float>.