lab 19 Hash Tables with Open Addressing

Instructions: This lab continues our study of Hash Tables. In this lab implement a Hash Table with open addressing. You are free to choose which open addressing routine to use, but you must implement a programmable load factor. Implement a Hash Table whose constructor take an integer (the initial size of the hash table) and a load factor, insert, remove, and get. Hints: if the value is not found in the Hash Table return a value using the default constructor.

WARNING: The loadFactor function ***must*** work properly for you to pass ***any*** tests.

```
#ifndef HASH_TABLE_H
#define HASH_TABLE_H

/* HashTable via open addressing */
template<class K, class V>
class HashTable {
    struct Pair {
        K mKey;
        V mValue;
        Pair(const K key, const V value) {
            mKey = key;
            mValue = value;
        }
        bool operator==(const Pair &pair) {
            return mKey == pair.mKey;
        }
    };
    private:
        /* Class to begin filling out...*/
        int mLen;
        V mInvalid;
    public:
        /* Initialize the Hash Table with size size. */
        HashTable(const int size, const float loadFactor);

        /* Deconstructor shall free up memory */
        ~HashTable();

        /* Map key -> val.
         * Return true if sucessful (it is unique.)
         * Otheriwise return false.
         */
        bool insert(const K &key, const V &val);

        /* Print out the HashTable */
        void print() const;

```

```
        /* Remove the val associated with key.
         * Return true if found and removed.
         * Otherwise return false.
         */
        bool remove(const K &key);

        /* Retrieves the V val that key maps to. */
        V& operator[](const K &key);

        /* Returns the current loadfactor for the Hash table (not the one
         * passed in.)
         * WARNING: This function must work properly for you to pass ANY tests.
         */
        float loadFactor();
};

int hashcode(int key);
int hashcode(const std::string &key);

#include "hashtable.cpp"

#endif
```

Write some test cases:
Create some test cases, using cxxtestgen, that you believe would cover all aspects of your code.

Memory Management:
Now that are using new, we must ensure that there is a corresponding delete to free the memory. Ensure there are no memory leaks in your code! Please run Valgrind on your tests to ensure no memory leaks!

How to turn in:
Turn in via GitHub. Ensure the file(s) are in your directory and then:

- $ git add <files>
- $ git commit
- $ git push

Due Date: November 13, 2017 2359

Teamwork: No teamwork, your work must be your own.