

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.

```
1 #ifndef HASH_TABLE_H
2 #define HASH_TABLE_H
3
4 /* HashTable via open addressing */
5 template<class K, class V>
6 class HashTable {
7     private:
8         /* Class to begin filling out...*/
9     public:
10         /* Initialize the Hash Table with size size. */
11         HashTable(const int size, const float loadFactor);
12
13         /* Deconstructor shall free up memory */
14         ~HashTable();
15
16         /* Map key -> val.
17          * Return true if sucessful (it is unique.)
18          * Otherwise return false.
19          */
20         bool insert(const K &key, const V &val);
21
22         /* Print out the HashTable */
23         void print() const;
24
25         /* Remove the val associated with key.
26          * Return true if found and removed.
27          * Otherwise return false.
28          */
29         bool remove(const K &key);
30
31         /* Retrieves the V val that key maps to. */
32         V& operator[](const K &key);
33
34         /* Returns the current loadfactor for the Hash table (not the one
35          * passed in.)
36          * NOTE: When you hit the load factor, double the size of your array.
37          * WARNING: This function must work properly for you to pass ANY tests.
```

```
38         */
39         float loadFactor();
40     };
41
42     int hashCode(int key);
43     int hashCode(const std::string &key);
44
45     #include "hashtable.cpp"
46
47     #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 14, 2018 2359

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