csc 326 lab 6

```
#include<iostream>
#include<list>
using namespace std;
struct Pair {
char key;
bool value;
};
class Hashmap {

private:
static const int buckets = 8;
list<Pair> table[buckets];

public:
bool isEmpty();
int hashFunc(int key);
void insert(const Pair& item);
void remove(int key);
Pair search(int key); // returns pair
void print();
};

bool Hashmap::isEmpty() {
for (int i = 0; i < buckets; i++) {
if (table[i].size() > 0)
return false;
}

return true;
}
int Hashmap::hashFunc(int key) {
return key % buckets;
}

void Hashmap::insert(const Pair& item) {
int hash = hashFunc(item.key);
auto& chain = table[hash]; // pointer to a linked list
// pointer to a head
bool exist = false;

for (auto it = chain.begin(); it != chain.end(); it++) {
if (it->key == item.key) {
```

```
exist = true;
it->value = item.value;
cout << "Key exist. Value was replaced" << endl;
break;
}
}

if (!exist) {
chain.emplace_back(item);
}
}
void Hashmap::remove(int key) {
int hash = hashFunc(key);
auto& chain = table[hash]; // pointer to a linked list
auto it = chain.begin(); // pointer to a data of the first node
bool exist = false;

for (; it != chain.end(); it++) {
if (it->key == key) {
exist = true;
it = chain.erase(it); // return a pointer to next value
cout << "Element was removed" << endl;
break;
}
}

if (!exist) {
cout << "Element was not found" << endl;
}
}
Pair Hashmap::search(int key) {
int hash = hashFunc(key);
auto& chain = table[hash]; // pointer to a linked list
auto it = chain.begin(); // pointer to a data of the first node

for (; it != chain.end(); it++) {
if (it->key == key) {
return it;
}
}

return NULL;
}
void Hashmap::print() {
```

```
for (int i = 0; i < buckets; i++) {
if (table[i].size() == 0) continue;
auto it = table[i].begin();

for (; it != table[i].end(); it++)
cout << "Key: " << it->key <<
" Value: " << it->value << endl;
}
}
int main() {
Hashmap map;
if (map.isEmpty()) {
cout << "Empty" << endl;
}
else {
cout << "Problem" << endl;
}
map.insert({ 'a', 0 });
map.insert({ 'd', 0 });
map.insert({ 'f', 0 });
map.insert({ 'e', 0 });
map.insert({ 'a', 1 });
map.insert({ 't', 0 });
map.insert({ 'd', 1 });
map.insert({ 'b', 0 });
map.print();
cout<<"
map.remove('a');
map.remove('d');
map.remove('f');
map.remove('f');
map.remove('e');
map.remove('b');
map.remove('t');
map.remove('d');
if (map.isEmpty()) {
cout << "Good job!" << endl;
}
else {
cout << "Problem!!!" << endl;
}
//given a string return first recurring character using implemented Hashtable
return 0;
}
```