Hashtable Practice
Create a project using the ChainedHashtable.h code given below, or use the following repl.it link. https://repl.it/@ringodarius/ChainedHashtable Fork the project into your own account. Use the code to answer the question below.

Basic understanding

1. Observe the declaration of the htable variable in the protected section of the ChainedHashtable class. This is a list of LinkedList header nodes. Note that there is the use of unique_ptr and shared_ptr. Is the LinkedList header node unique or shared? Is the list of Nodes unique or shared?

Htable is an unique pointer while node is a shared pointer

2. Observe the getHash method. What is the algorithm used to enter the value into the table?

Data is modulated by capacity or default 21

3. Take a look at the constructor. What is the default capacity of the hashtable?

21

4. Observe the implementation of the insert method. What would the value h be if the user were inserting the value 10? (assume the default capacity of the hashtable)

10

5. What gets created if that particular element does not exist?

It goes in the same point and points to the next one it makes a node pointer

6. What would be the positions of the following values? 18, 21, 25

0, 4, 18

7. Create a program to insert those values into the hashtable. The display the hashtable to check your answers.

I was right

8. Observe the insert method again. What happens when there is a collision when the LinkedList header already exists?

It makes node and points to the next one. The next value inserted is the first one

9. Does the new item get placed before or after the item in the LinkedList header?

before

10. Where would the value 31 be placed in the hashtable? How about 42, 52, 60?

0: 42, 10: 52 -> 31, 18: 60

11. Add those values to your hashtable and run the program to check your answers.

Yep

Contains/Delete

1. Observe the contains() method. What is used to find the LinkedListheader where the item is contained?

It uses getHash and then moves through the mode

2. What happens when the item is not in that particular list? Why would it not need to search in the rest of the array?

It only checks the node list and return false

3. Observe the remove() method. What pointers are updated if the item to be removed is the head of the list?

The node pointer in the list is updated

4. What pointers are updated when the item to be removed is in the middle of the list?

Both the node pointer and the list itself

Why Prime Number?

1. Create a hashtable with a non-prime number for the capacity by using the following code.

```
ChainedHashtable<int>ht(128);
for (int i = 0; i < 100; i++) {
      ht.insert(rand() % 1000);
}
cout << ht << endl;
```

2. Observe the following:
    a. About how many positions had collisions?

24

    b. What was the longest chain of collisions?

4

    c. How many positions had that same amount of collisions?

2

3. Change the capacity of the hashtable to the prime number (127) and answer the same questions…
    a. About how many positions had collisions?

22

    b. What was the longest chain of collisions?

5

    c. How many positions had that same amount of collisions?

0

4. Was there an advantage to having a capacity of a prime number? Why or why not?

I don't really know i might have done something wrong

5. Change the random number to be rand() % 10 instead of 1000. Run the program. Is this a very efficient use of a function? Why or why not?

No all the numbers are in the first 10 and out of every position it doesn't really work more efficient with more numbers and more spots

Chained Hashtable Code

```
#pragma once

#include <iostream>
#include <iomanip>
#include <memory>
#include <sstream>

using namespace std;

template<class Type>
struct Node {
  Node(Type data): data(data), next(nullptr){}

  Type data;
  shared_ptr <Node<Type>> next;
};

template<class Type>
class ChainedHashtable;

template<class Type>
ostream& operator<<(ostream& out, const ChainedHashtable<Type> &t);

template<class Type>
class ChainedHashtable {
protected:
  int sz;     //Number of elements in table
  int capacity;   //Table size
  unique_ptr<shared_ptr<Node<Type>>[]> htable;  //An Array of LinkedList
Node Headers

public:
  ChainedHashtable(int capacity = 21) :
    sz(0), capacity(capacity), htable(new
shared_ptr<Node<Type>>[capacity]) {}
  void insert(Type e);    //Inserts the element in the table
  bool remove(Type e);    //Removes an element from the table
  bool contains(Type e);    //Returns true if the value exists
  bool empty() { return sz == 0; }  //Returns true if the list is empty
  int size() { return sz; }     //Returns the number of elements in the
table
  int getHash(Type data) { return data % capacity; }  //Hash based upon
mod division
  friend ostream& operator<< <>(ostream& out, const ChainedHashtable<Type>
&t); //Displays the Hashtable
};

template<class Type>
void ChainedHashtable<Type>::ChainedHashtable::insert(Type data) {
  int h = getHash(data);
```

```
  if (htable[h]) {
    auto temp = make_shared<Node<Type>>(data);
    temp->next = htable[h];
    htable[h] = temp;
  }
  else {
    htable[h] = make_shared<Node<Type>>(data);
  }
  sz++;
}

template<class Type>
bool ChainedHashtable<Type>::remove(Type data) {
  if (empty()) {
    throw runtime_error("Table is empty");
  }
  int t = getHash(data);

  //If the node is the first item in the list
  if (htable[t] == nullptr) {
    return false;
  }
  if (htable[t]->data == data) {
    htable[t] = htable[t]->next;
    sz--;
    return true;
  }

  auto curr = htable[t]->next;
  auto prev = htable[t];
  while (curr) {
    if (curr->data == data) {
      prev->next = curr->next;
      sz--;
      return true;
    }
    curr = curr->next;
    prev = prev-> next;
  }

  return false;
}

template<class Type>
bool ChainedHashtable<Type>::contains(Type e) {
  int t = getHash(data);

  auto curr = htable[t];
  while (curr) {
    if (curr->data == data) {
      return true;
    }
    curr = curr->next;
```

```
  }
  return false;
}

template<class Type>
ostream& operator<<(ostream& out, const ChainedHashtable<Type> &t) {
  for (int i = 0; i < t.capacity; i++) {
    out << std::setw(8) << i << ": ";

    auto curr = t.htable[i];
    while (curr) {
      out << curr->data << " ";
      if (curr->next) out << "-> ";
      curr = curr->next;
    }
    out << endl;
  }
  return out;
}
```