CSCE 221 Spring 2021

Programming Assignment 6

Hashing

Approved Includes

```
<functional>
<iostream>
<list>
<sstream>
<stdexcept>
<vector>
"hashtable_separate_chaining.h"
"hashtable_open_addressing.h"
```

Code Coverage

You must submit a test suite for each task that, when run, covers at least 90% of your code. You should, at a minimum, invoke every function at least once. Best practice is to also check the actual behavior against the expected behavior, e.g. verify that the result is correct. You should be able to do this automatically, i.e. write a program that checks the actual behavior against the expected behavior.

Your test suite should include ALL tests that you wrote and used, including tests you used for debugging. You should have MANY tests.

Starter Code

```
for X in {separate_chaining, open_addressing}
      hashtable_X.h
      hashtable_X_tests.cpp
      X_compile_test.cpp
Makefile
```

Files to Submit

```
hashtable_open_addressing.h
hashtable_open_addressing_tests.cpp
hashtable_separate_chaining.h
hashtable_separate_chaining_tests.cpp
```

Task 1: Separate Chaining Hash Table

Implement a separate chaining hash table.

Requirements

Files

`hashtable_separate_chaining.h` - contains the template definitions
`hashtable_separate_chaining_tests.cpp` - contains the test cases and test driver (main)

Class

```
template <class Key, class Hash=std::hash<Key>>
class HashTable;
```

Functions (public)

Constructors

`HashTable()` - makes an empty table with 11 buckets and max load factor 1
`explicit HashTable(size_t)` - makes an empty table with the specified number of buckets

Iterators

Optional

Capacity

`bool is_empty() const` - returns true if the table is empty
`size_t size() const` - returns the number of values in the table

Modifiers

`void make_empty()` - remove all values from the table. Do not change the number of buckets. Do not change the maximum load factor.
`bool insert(const Key&)` - insert the given lvalue reference into the table, rehashing **only** if the maximum load factor is exceeded, return true if insert was successful (false if item already exists)
`size_t remove(const Key&)` - remove the specified value from the table, return number of elements removed (0 or 1).

Lookup

`bool contains(const Key&) const` - returns Boolean true if the specified value is in the table

Bucket Interface

`size_t bucket_count() const` - return the number of buckets in the table

`size_t bucket_size(size_t) const` - return the number of values in the specified bucket (by index); throw `std::out_of_range` if the bucket index is out of bounds of the table.
`size_t bucket(const Key&) const` - return the index of the bucket that contains the specified value (or would contain it, if it existed)

Hash Policy

`float load_factor() const` - return the current load factor of the table
`float max_load_factor() const` - return the current maximum load factor of the table
`void max_load_factor(float)` - set the maximum load factor of the table, forces a rehash if the new maximum is less than the current load factor, throws `std::invalid_argument` if the input is invalid
`void rehash(size_t)` - set the number of buckets to the specified value and rehash the table if the total number of buckets has changed. If the new number of buckets would cause the load factor to exceed the maximum load factor, then the new number of buckets is at least size() / max_load_factor().

Visualization

`void print_table(std::ostream&=std::cout) const` - pretty print the table. Required to exist and produce reasonable output, the empty table should print "`<empty>\n`", but the format of the output is not graded

Optional

`HashTable(const HashTable&)` - constructs a copy of the given table
`HashTable(HashTable&&)` - move constructs a copy of the given (rvalue) table
`~HashTable()` - destructs this table
`HashTable& operator=(const HashTable&)` - assigns a copy of the given table
`HashTable& operator=(HashTable&&)` - move assigns a copy of the given (rvalue) table
`void insert(Key&&)` - insert the given rvalue reference into the table using move semantics

Example

```
std::cout << "make an empty hash table with 11 buckets for strings" <<
std::endl;
HashTable<std::string> table(11);

std::cout << "initial size is " << table.size() << std::endl;
std::cout << "initial bucket count is " << table.bucket_count() << std::endl;
std::cout << "initial load factor is " << table.load_factor() << std::endl;
std::cout << "initial max load factor is " << table.max_load_factor() <<
std::endl;

std::cout << "insert several strings" << std::endl;
table.insert("And them who hold High Places");
table.insert("Must be the ones to start");
table.insert("To mold a new Reality");
table.insert("Closer to the Heart");
table.insert("The Blacksmith and the Artist");
table.insert("Reflect it in their Art");
table.insert("Forge their Creativity");
table.insert("Closer to the Heart");
table.insert("Philosophers and Plowmen");
table.insert("Each must know their Part");
table.insert("To sow a new Mentality");
table.insert("Closer to the Heart");
table.insert("You can be the Captain");
table.insert("I will draw the Chart");
table.insert("Sailing into Destiny");
table.insert("Closer to the Heart");

std::cout << "size is " << table.size() << std::endl;
std::cout << "bucket count is " << table.bucket_count() << std::endl;
std::cout << "load factor is " << table.load_factor() << std::endl;
std::cout << "max load factor is " << table.max_load_factor() << std::endl;

{
      std::cout << "print the table" << std::endl;
      std::stringstream ss;
      table.print_table(ss);
      std::cout << ss.str() << std::endl;
}

std::cout << "remove \"Philosophers and Plowmen\"" << std::endl;
table.remove("Philosophers and Plowmen");
std::cout << "remove \"Each must know their Part\"" << std::endl;
```

```
table.remove("Each must know their Part");

std::cout << "size is " << table.size() << std::endl;
std::cout << "bucket count is " << table.bucket_count() << std::endl;
std::cout << "load factor is " << table.load_factor() << std::endl;
std::cout << "max load factor is " << table.max_load_factor() << std::endl;

{
      std::cout << "print the table" << std::endl;
      std::stringstream ss;
      table.print_table(ss);
      std::cout << ss.str() << std::endl;
}

std::cout << "set max load factor to 2" << std::endl;
table.max_load_factor(2);
std::cout << "rehash the table to size 7" << std::endl;
table.rehash(7);

std::cout << "size is " << table.size() << std::endl;
std::cout << "bucket count is " << table.bucket_count() << std::endl;
std::cout << "load factor is " << table.load_factor() << std::endl;
std::cout << "max load factor is " << table.max_load_factor() << std::endl;

{
      std::cout << "print the table" << std::endl;
      std::stringstream ss;
      table.print_table(ss);
      std::cout << ss.str() << std::endl;
}

std::cout << "find \"The Blacksmith and the Artist\"" << std::endl;
size_t index = table.bucket("The Blacksmith and the Artist");
std::cout << " ==> bucket " << index << std::endl;
std::cout << "     which has " << table.bucket_size(index) << " elements" <<
std::endl;

std::cout << "make the table empty" << std::endl;
table.make_empty();

std::cout << "size is " << table.size() << std::endl;
std::cout << "bucket count is " << table.bucket_count() << std::endl;
std::cout << "load factor is " << table.load_factor() << std::endl;
std::cout << "max load factor is " << table.max_load_factor() << std::endl;
```

```
{
	std::cout << "print the table" << std::endl;
	std::stringstream ss;
	table.print_table(ss);
	std::cout << ss.str() << std::endl;
}
```

Example Output

Note: your machine's `std::hash` may return different values than mine, leading to different output. That's OK. The behavior on Gradescope will not be affected.

```
make an empty hash table with 11 buckets for strings
initial size is 0
initial bucket count is 11
initial load factor is 0
initial max load factor is 1
insert several strings
size is 13
bucket count is 23
load factor is 0.565217
max load factor is 1
print the table
2: [Closer to the Heart]
4: [Each must know their Part]
7: [To mold a new Reality]
9: [To sow a new Mentality, Philosophers and Plowmen]
10: [I will draw the Chart]
11: [And them who hold High Places, The Blacksmith and the Artist]
12: [You can be the Captain]
13: [Must be the ones to start]
15: [Reflect it in their Art]
16: [Sailing into Destiny]
19: [Forge their Creativity]

remove "Philosophers and Plowmen"
remove "Each must know their Part"
size is 11
bucket count is 23
load factor is 0.478261
max load factor is 1
print the table
2: [Closer to the Heart]
7: [To mold a new Reality]
```

```
9: [To sow a new Mentality]
10: [I will draw the Chart]
11: [And them who hold High Places, The Blacksmith and the Artist]
12: [You can be the Captain]
13: [Must be the ones to start]
15: [Reflect it in their Art]
16: [Sailing into Destiny]
19: [Forge their Creativity]

set max load factor to 2
rehash the table to size 7
size is 11
bucket count is 7
load factor is 1.57143
max load factor is 2
print the table
0: [Sailing into Destiny, The Blacksmith and the Artist, To sow a new
Mentality]
1: [Must be the ones to start, Closer to the Heart]
2: [To mold a new Reality]
3: [Reflect it in their Art, You can be the Captain]
4: [Forge their Creativity, I will draw the Chart]
5: [And them who hold High Places]

find "The Blacksmith and the Artist"
 ==> bucket 0
     which has 3 elements
make the table empty
size is 0
bucket count is 7
load factor is 0
max load factor is 2
print the table
<empty>
```

Task 2: Open Addressing Hash Table

Implement an open addressing hash table.

You can use linear probing, quadratic probing, or cubic probing. No matter which you choose, the load factor should always be less than or equal to 0.5.

Requirements

Files

`hashtable_open_addressing.h` - contains the template definitions
`hashtable_open_addressing_tests.cpp` - contains the test cases and test driver (main)

Class

```
template <class Key, class Hash=std::hash<Key>>
class HashTable;
```

Functions (public)

Constructors

`HashTable()` - makes an empty table with 11 cells
`explicit HashTable(size_t)` - makes an empty table with the specified number of cells

Iterators

Optional

Capacity

`bool is_empty() const` - returns true if the table is empty
`size_t size() const` - returns the number of active values in the table
`size_t table_size() const` - return the number of cells in the table

Modifiers

`void make_empty()` - remove all values from the table. Do not change the number of cells.
`bool insert(const Key&)` - insert the given lvalue reference into the table, rehashing if the maximum load factor is exceeded, return true if insert was successful (false if item already exists). Don't overwrite deleted values unless they are equal to value being inserted (i.e. undelete).
`size_t remove(const Key&)` - remove the specified value from the table, return number of elements removed (0 or 1). Use lazy deletion.

Lookup

`bool contains(const Key&) const` - returns Boolean true if the specified value is in the table

Position

`size_t position(const Key&) const` - return the index of the cell that would contain the specified value. This method handles collision resolution.

Visualization

`void print_table(std::ostream&=std::cout) const` - pretty print the table. Required to exist and produce reasonable output, the empty table should print "`<empty>\n`", but the format of the output is not graded

Optional

`HashTable(const HashTable&)` - constructs a copy of the given table
`HashTable(HashTable&&)` - move constructs a copy of the given (rvalue) table
`~HashTable()` - destructs this table
`HashTable& operator=(const HashTable&)` - assigns a copy of the given table
`HashTable& operator=(HashTable&&)` - move assigns a copy of the given (rvalue) table
`void insert(Key&&)` - insert the given rvalue reference into the table using move semantics

Example

```
std::cout << "make an empty hash table with 11 buckets for strings" <<
std::endl;
HashTable<std::string> table(11);

std::cout << "initial size is " << table.size() << std::endl;
std::cout << "initial table size is " << table.table_size() << std::endl;

std::cout << "insert several strings" << std::endl;
table.insert("And them who hold High Places");
table.insert("Must be the ones to start");
table.insert("To mold a new Reality");
table.insert("Closer to the Heart");
table.insert("The Blacksmith and the Artist");
table.insert("Reflect it in their Art");
table.insert("Forge their Creativity");
table.insert("Closer to the Heart");
table.insert("Philosophers and Plowmen");
table.insert("Each must know their Part");
table.insert("To sow a new Mentality");
table.insert("Closer to the Heart");
table.insert("You can be the Captain");
table.insert("I will draw the Chart");
table.insert("Sailing into Destiny");
table.insert("Closer to the Heart");

std::cout << "size is " << table.size() << std::endl;
std::cout << "table size is " << table.table_size() << std::endl;

{
  std::cout << "print the table" << std::endl;
  std::stringstream ss;
  table.print_table(ss);
  std::cout << ss.str() << std::endl;
}

std::cout << "remove \"Philosophers and Plowmen\"" << std::endl;
table.remove("Philosophers and Plowmen");
std::cout << "remove \"Each must know their Part\"" << std::endl;
table.remove("Each must know their Part");

std::cout << "size is " << table.size() << std::endl;
std::cout << "table size is " << table.table_size() << std::endl;
```

```
{
  std::cout << "print the table" << std::endl;
  std::stringstream ss;
  table.print_table(ss);
  std::cout << ss.str() << std::endl;
}

std::cout << "find \"The Blacksmith and the Artist\"" << std::endl;
size_t index = table.position("The Blacksmith and the Artist");
std::cout << " ==> cell " << index << std::endl;

std::cout << "make the table empty" << std::endl;
table.make_empty();

std::cout << "size is " << table.size() << std::endl;
std::cout << "table size is " << table.table_size() << std::endl;

{
  std::cout << "print the table" << std::endl;
  std::stringstream ss;
  table.print_table(ss);
  std::cout << ss.str() << std::endl;
}
```

Example Output

Note: your machine's `std::hash` may return different values than mine, leading to different output. That's OK. The behavior on Gradescope will not be affected.

```
make an empty hash table with 11 buckets for strings
initial size is 0
initial table size is 11
insert several strings
size is 13
table size is 47
print the table
8: Forge their Creativity
11: Philosophers and Plowmen
16: I will draw the Chart
23: Each must know their Part
26: Closer to the Heart
27: Sailing into Destiny
30: The Blacksmith and the Artist
31: You can be the Captain
32: To mold a new Reality
```

```
33: Reflect it in their Art
36: To sow a new Mentality
37: Must be the ones to start
40: And them who hold High Places

remove "Philosophers and Plowmen"
remove "Each must know their Part"
size is 11
table size is 47
print the table
8: Forge their Creativity
16: I will draw the Chart
26: Closer to the Heart
27: Sailing into Destiny
30: The Blacksmith and the Artist
31: You can be the Captain
32: To mold a new Reality
33: Reflect it in their Art
36: To sow a new Mentality
37: Must be the ones to start
40: And them who hold High Places

find "The Blacksmith and the Artist"
 ==> cell 30
make the table empty
size is 0
table size is 47
print the table
<empty>
```

C++ Syntax Tips

How to use template Hash parameter

Your class declaration looks like this:

```
template <class Key, class Hash=std::hash<Key>>
class HashTable;
```

The second argument of the template declaration specifies the default value for the Hash type as `std::hash<Key>`. This is a function object that defines a hash function, i.e. it responds to `operator()`. Whatever type is used as the `Hash` parameter must respond to `operator()` as well as a few other requirements (see `documentation for std::hash<Key>`). The salient features are that the function takes a single parameter of type `Key`, returns a `size_t` value, and does not throw exceptions.

To invoke the hash function, you can do the following, which instantiates a temporary hash function object and immediately uses it:

```
size_t hash_value = Hash{}(value_to_hash);
```

How to Find Primes Quickly

As you know, your hash tables should have prime-valued sizes. This requires that you be able to find prime numbers. Here is my advice:

The primality test in C-family on Wikipedia is plenty fast. It uses the neat fact that all primes > 3 are of the form 6k ± 1 and so can take steps of size 6 rather than just 2 (as naive trial division would).
But, you should check out Fermat or Miller-Rabin, too (Pseudoprimes are good enough for hash tables).

If you want to test that your implementation is “fast-enough”, use it to find all the (probable-)primes up to 2 million, i.e. invoke it 1 million times (check 1M odd numbers). This should take less than 1 second on a modern computer with a fast-enough implementation of primality testing. There are 148933 primes less than 2 million.

About using code/algorithm resources

If you use code from Wikipedia or any other approved resource, e.g. for primality testing, you should put a comment citing the source in the same place that you use the code. The same goes for pseudocode and algorithms that you did not develop on your own.

BUT! Remember: other students and tutoring services are not approved resources.

The textbook, the TAs, and the instructor are all high-quality, approved resources.

Also remember: you should write your own code as much as possible in this course. The best way to understand is to do. Copy+paste is not “doing”. If you need help, you should come to office hours with the instructor and TAs.