CS515

Assignment 10

The purpose of this assignment is to give you practice writing your own hash table in the context of the std::unordered_map container. This container implements the Map ADT as in our previous assignments, but the underlying data structure used now is a hash table, which doesn't maintain the ordering of the keys.

Download all files from `~cs515/public/10P`

Implement the **`UnorderedMap`** generic container using an underlying external chaining hash table with linked lists. **`UnorderedMap`** supports the same public interface as your previously implemented tree-based Map. **`UnorderedMap`** has much better performance because it uses a hash table as the underlying concrete data structure, but that also means it doesn't support ordered iteration of its elements. Indeed, our iterator for the hash-based Map is a very primitive forward iterator that goes through all the elements in an arbitrary order. (See comments in the header file for more details.)

You may use additional helper methods and modify the header file accordingly. You need to complete all methods declared in the header file `unorderedmap.h`, while meeting following requirements.

- The hash table uses external chaining technique for collision resolution. Each chain can be viewed as a bucket that holds entries that hash to the same value. Each chain is implemented as an unsorted linked list. For example, a newly inserted entry will be simply added to the front of the linked list.
- The library generic hash function `std::hash<T>` is used to calculate the hash value for our KEY type keys. The template header:

  ```
  template <typename KEY, typename T, typename H = std::hash<KEY>>
  ```

  indicates there are three type parameters: the key type KEY, the value type T and the function object type H, which is used to hold the hash function and is assigned a default value, the `std::hash<T>` function. To calculate the hash value, simply use

  ```
  std::size_t key = H()(newkey);
  ```

  and then mod the value `key` by the table size.
- The hash table will be rehashed dynamically when the load factor reaches 2. This means that the underlying hash table doubles the size (approximately since we want to keep the table size prime), and all entries be re-inserted into the table. The hash table does not shrink dynamically when the entry size reduces.
- A public method named `tsize()` is required so that we can verify your rehash implementation.

Submission:

- Submit **`unorderedmap.h`** and **`unorderedmap.cpp`.**
- You should also fill out the README file and submit it as well. Submit the following files to the appropriate assignment on Mimir:

  `unorderedmap.h unorderedmap.cpp README`
- Do not turn in executables or object code. Programs that produce compile time errors or warnings will receive a zero mark (even if it might work perfectly on your home computer).

- Be sure to provide comments in your program. You must include the information as the section of comments below:

```
/**     CS515 Assignment 10
File: XXX.cpp
Name: XXX
Section: X
Date: XXX
Collaboration Declaration: assistance received from TA, PAC etc.
*/
```

Some notes on grading:

- Programs are graded for correctness (output results and code details), following directions and using specified features, documentation and style.
- To successfully pass the provided sample tests is not an indication of a potential good grade; to fail one or more of these tests is an indication of a potential bad grade.
- You must test thoroughly your program with your own test data/cases to ensure all the requirements are fulfilled. We will use additional test data/cases other than the sample tests to grade your program.
- Here is a tentative grading scheme.

directions	100
Unordered Map run 1 (Insert)	10
Unordered Map run 1 valgrind	5
Unordered Map run 2 (operator[])	10
Unordered Map run 2 valgrind	5
Unordered Map run 3 (find)	8
Unordered Map run 3 valgrind	5
Unordered Map run 4 (find)	8
Unordered Map run 5 (erase)	8
Unordered Map run 5 valgrind	5
Unordered Map run 6 (erase)	8
Unordered Map run 7 (iterator++)	10
Unordered Map run 7 valgrind	4
Unordered Map run 8 (rehash)	10
Unordered Map run 8 valgrind	4