Homework #3: Hash Tables

- Deadline: Jun. 17, 11:59 PM
- Recommended: Use Visual Studio 2022 or Visual Studio 2019

- Submission Requirements: 김철수_20251111.zip
  1. Report: PDF format (2~3 pages, plus appendices for code & comments only)
  2. Source Code: Files in .cpp and .h format
  3. Executable: Compiled .exe file

- Grading Criteria:
  - Code / Executable: 50%
  - Report: 50%

main.cpp

```cpp
#include "hashMap.h"

#include <iostream>

int main() {
    HashMap<std::string, int, stringHash> map;

    typedef HashMap<std::string, int, stringHash>::Iterator hashIter;

    map.put("apple", 100);
    map.put("banana", 150);
    map.put("cherry", 200);

    hashIter iter = map.begin();
    while (1) {
        if (iter == map.end()) break;
        std::cout << (*iter).key() << ": " << (*iter).value() << ", ";
        ++iter;
    }
    std::cout << std::endl;
```

```
apple: 100, cherry: 200, banana: 150,
cherry: 200, banana: 150,
cherry: 300, banana: 150,
banana: 150,
map is empty
```

```
map.erase("apple");

iter = map.begin();
while (1) {
    if (iter == map.end()) break;
    std::cout << (*iter).key() << ": " << (*iter).value() << ", ";
    ++iter;
}
std::cout << std::endl;

map.put("cherry", 300);

iter = map.begin();
while (1) {
    if (iter == map.end()) break;
    std::cout << (*iter).key() << ": " << (*iter).value() << ", ";
    ++iter;
}
std::cout << std::endl;
```

```
apple: 100, cherry: 200, banana: 150,
cherry: 200, banana: 150,
cherry: 300, banana: 150,
banana: 150,
map is empty
```

```
    iter = map.find("cherry");
    map.erase(iter);
    iter = map.begin();
    while (1) {
        if (iter == map.end()) break;
        std::cout << (*iter).key() << ": " << (*iter).value() << ", ";
        ++iter;
    }
    std::cout << std::endl;

    map.erase("banana");
    if (map.empty())
        std::cout << "map is empty" << std::endl;

    return 0;
}
```

```
apple: 100, cherry: 200, banana: 150,
cherry: 200, banana: 150,
cherry: 300, banana: 150,
banana: 150,
map is empty
```

hashMap.h

```cpp
#pragma once

#include <iostream>
#include <list>
#include <vector>

struct stringHash {
    std::size_t operator()(const std::string& key) const {
        std::size_t hash = 0;
        for (char c : key) {
            hash = (hash * 31) + c;
        }
        return hash;
    }
};

template <typename K, typename V>
class Entry {
...
};

template <typename K, typename V, typename H>
class HashMap {
...
};
```

```
template <typename K, typename V>
class Entry {                                          // a (key, value) pair
public:                                                // public functions
  Entry(const K& k = K(), const V& v = V())            // constructor
    : _key(k), _value(v) { }
  const K& key() const { return _key; }                // get key
  const V& value() const { return _value; }            // get value
  void setKey(const K& k) { _key = k; }                // set key
  void setValue(const V& v) { _value = v; }            // set value
private:                                               // private data
  K _key;                                              // key
  V _value;                                            // value
};
```

```
template <typename K, typename V, typename H>
class HashMap {
public:                                          // public types
  typedef Entry<const K,V> Entry;                // a (key,value) pair
  class Iterator;                                // a iterator/position
public:                                          // public functions
  HashMap(int capacity = 100);                   // constructor
  int size() const;                              // number of entries
  bool empty() const;                            // is the map empty?
  Iterator find(const K& k);                     // find entry with key k
  Iterator put(const K& k, const V& v);          // insert/replace (k,v)
  void erase(const K& k);                        // remove entry with key k
  void erase(const Iterator& p);                 // erase entry at p
  Iterator begin();                              // iterator to first entry
  Iterator end();                                // iterator to end entry
protected:                                       // protected types
  typedef std::list<Entry> Bucket;               // a bucket of entries
  typedef std::vector<Bucket> BktArray;          // a bucket array
  // ...insert HashMap utilities here
private:
  int n;                                         // number of entries
  H hash;                                        // the hash comparator
  BktArray B;                                    // bucket array
public:                                          // public types
  // ...insert Iterator class declaration here
};
```

```
Iterator finder(const K& k);                                  // find utility
Iterator inserter(const Iterator& p, const Entry& e);         // insert utility
void eraser(const Iterator& p);                               // remove utility
typedef typename BktArray::iterator BItor;                    // bucket iterator
typedef typename Bucket::iterator EItor;                      // entry iterator
static void nextEntry(Iterator& p)                            // bucket's next entry
  { ++p.ent; }
static bool endOfBkt(const Iterator& p)                       // end of bucket?
  { return p.ent == p.bkt->end(); }
```

```
class Iterator {                                              // an iterator (& position)
private:
  EItor ent;                                                  // which entry
  BItor bkt;                                                  // which bucket
  const BktArray* ba;                                         // which bucket array
public:
  Iterator(const BktArray& a, const BItor& b, const EItor& q = EItor())
    : ent(q), bkt(b), ba(&a) { }
  Entry& operator*() const;                                   // get entry
  bool operator==(const Iterator& p) const;                   // are iterators equal?
  Iterator& operator++();                                     // advance to next entry
  friend class HashMap;                                       // give HashMap access
};
```

hashMap.cpp

```cpp
#include "hashMap.h"

#include <iostream>
#include <list>
#include <vector>

template <typename K, typename V, typename H>
HashMap<K, V, H>::HashMap(int capacity) : n(0), B(capacity) {}

....

template class HashMap<std::string, int, stringHash>;
```

```
/* HashMap⟨K,V,H⟩ :: */                                    // constructor
  HashMap(int capacity) : n(0), B(capacity) { }

/* HashMap⟨K,V,H⟩ :: */                                    // number of entries
  int size() const { return n; }

/* HashMap⟨K,V,H⟩ :: */                                    // is the map empty?
  bool empty() const { return size() == 0; }
```

```
template <typename K, typename V, typename H> // get entry
typename HashMap<K,V,H>::Entry&
HashMap<K,V,H>::Iterator::operator*() const
  { return *ent; }
```

```
/* HashMap⟨K,V,H⟩ :: */                                    // are iterators equal?
  bool Iterator::operator==(const Iterator& p) const {
    if (ba != p.ba || bkt != p.bkt) return false;          // ba or bkt differ?
    else if (bkt == ba->end()) return true;                // both at the end?
    else return (ent == p.ent);                            // else use entry to decide
  }
```

```
/* HashMap⟨K,V,H⟩ :: */                                    // advance to next entry
 Iterator& Iterator::operator++() {
   ++ent;                                                  // next entry in bucket
   if (endOfBkt(*this)) {                                  // at end of bucket?
     ++bkt;                                                // go to next bucket
     while (bkt != ba->end() && bkt->empty())              // find nonempty bucket
       ++bkt;
     if (bkt == ba->end()) return *this;                   // end of bucket array?
     ent = bkt->begin();                                   // first nonempty entry
   }
   return *this;                                           // return self
 }
```

```
/* HashMap⟨K,V,H⟩ :: */                          // iterator to end
  Iterator end()
    { return Iterator(B, B.end()); }

/* HashMap⟨K,V,H⟩ :: */                          // iterator to front
  Iterator begin() {
    if (empty()) return end();                   // emtpty - return end
    BItor bkt = B.begin();                       // else search for an entry
    while (bkt->empty()) ++bkt;                  // find nonempty bucket
    return Iterator(B, bkt, bkt->begin());       // return first of bucket
  }
```

```
/* HashMap⟨K,V,H⟩ :: */                          // find utility
  Iterator finder(const K& k) {
    int i = hash(k) % B.size();                  // get hash index i
    BItor bkt = B.begin() + i;                   // the ith bucket
    Iterator p(B, bkt, bkt->begin());            // start of ith bucket
    while (!endOfBkt(p) && (*p).key() != k)      // search for k
      nextEntry(p);
    return p;                                    // return final position
  }
```

```
/* HashMap⟨K,V,H⟩ :: */                                    // find key
  Iterator find(const K& k) {
    Iterator p = finder(k);                                // look for k
    if (endOfBkt(p))                                       // didn't find it?
      return end();                                        // return end iterator
    else
      return p;                                            // return its position
  }
```

```
/* HashMap⟨K,V,H⟩ :: */                                    // insert utility
  Iterator inserter(const Iterator& p, const Entry& e) {
    EItor ins = p.bkt->insert(p.ent, e);                   // insert before p
    n++;                                                   // one more entry
    return Iterator(B, p.bkt, ins);                        // return this position
  }
```

```
/* HashMap⟨K,V,H⟩ :: */                                    // insert/replace (v,k)
  Iterator put(const K& k, const V& v) {
    Iterator p = finder(k);                                // search for k
    if (endOfBkt(p)) {                                     // k not found?
      return inserter(p, Entry(k, v));                     // insert at end of bucket
    }
    else {                                                 // found it?
      p.ent->setValue(v);                                  // replace value with v
      return p;                                            // return this position
    }
  }
```

```
/* HashMap⟨K,V,H⟩ :: */                                    // remove utility
  void eraser(const Iterator& p) {
    p.bkt->erase(p.ent);                                   // remove entry from bucket
    n--;                                                   // one fewer entry
  }

/* HashMap⟨K,V,H⟩ :: */                                    // remove entry at p
  void erase(const Iterator& p)
    { eraser(p); }

/* HashMap⟨K,V,H⟩ :: */                                    // remove entry with key k
  void erase(const K& k) {
    Iterator p = finder(k);                                // find k
    if (endOfBkt(p))                                       // not found?
      throw NonexistentElement("Erase of nonexistent"); // ...error
    eraser(p);                                             // remove it
  }
```