객체지향프로그래밍과 자료구조 (실습)

Lab 11 (보충설명) Thesaurus Dictionary

Prof. Young-Tak Kim

Advanced Networking Technology Lab. (YU-ANTL)

Dept. of Information & Comm. Eng, Yeungnam University, KOREA

(Tel : +82-53-810-2497; E-mail : ytkim@yu.ac.kr)

Outline

- Map Data Structure
- String Type Key
- Hash, Hash Table
- Cyclic Shift Hash Code
- Class Hash Map
- Class Hash Dictionary
  - class Range

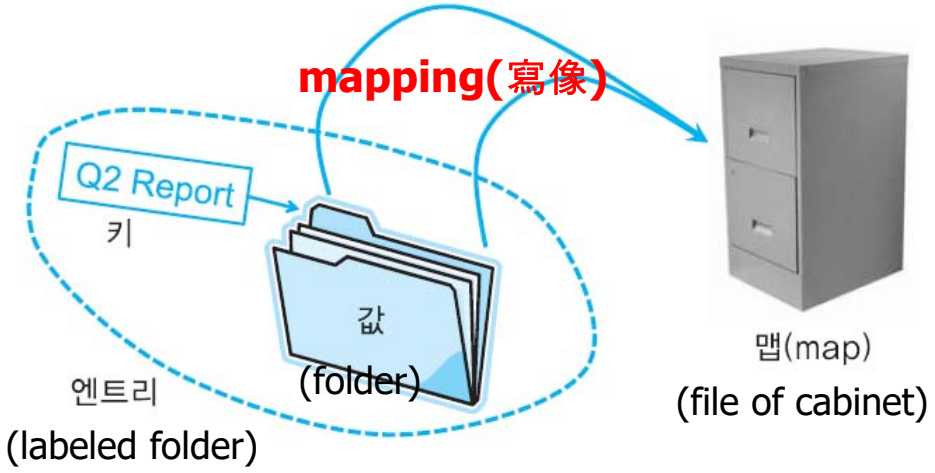
Maps

◆ Map

- models a searchable collection of key-value entries
- the main operations of a map are for **searching**, **inserting**, and **deleting** items
- **multiple entries** with the **same key** are ***not allowed***

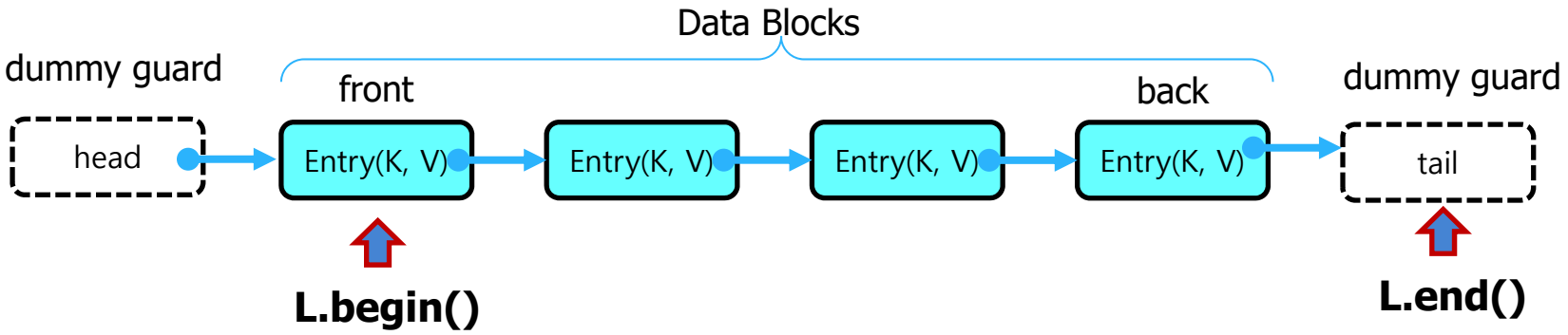
◆ Applications of Map:

- address book
- student-record database



A Simple List-Based Map

- **We can efficiently implement a map using an unsorted list**
  - We store the items of the map in a list **S** (based on a doubly-linked list), in arbitrary order



Hash Tables

- A hash function *h()* maps keys of a given type to integers in a fixed interval [0, *N* - 1]
- Example:

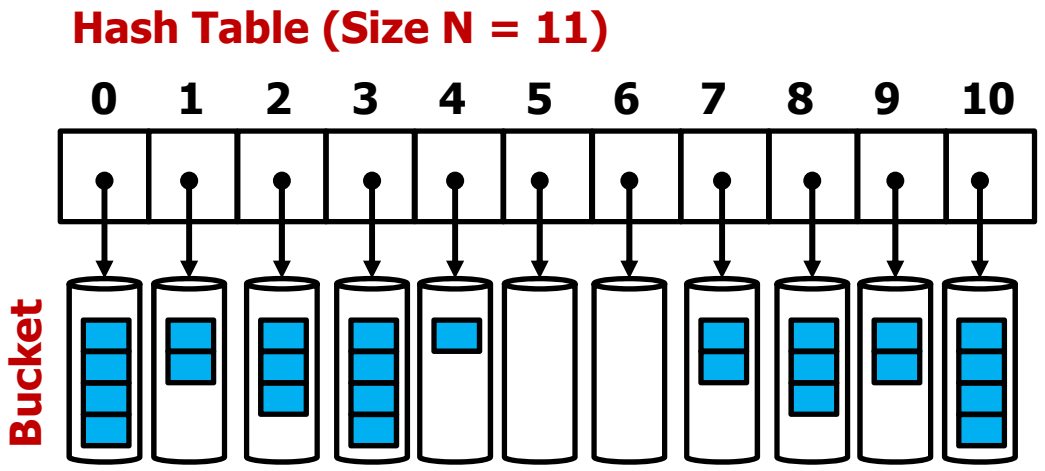
  $h(x) = x \bmod N$

  is a simple hash function for integer keys
- The integer $h(x)$ is called the hash value of key *x*
- A hash table for a given key type consists of
  - Hash function *h()*
  - Hash table (bucket array) of size *N*
- When implementing a map with a hash table, the goal is to store entry (*k*, v) at index $i = h(k)$

Bucket Arrays

◆ Bucket Array

- a *bucket array* for a hash table is an array A of size N, where each cell of A is thought of as a bucket (collection of key-value pairs), and the integer N defines the capacity of the array



Hash Functions

- A hash function is usually specified as the composition of two functions:

  (i) Hash code:

  h_1: keys $\rightarrow$ integers

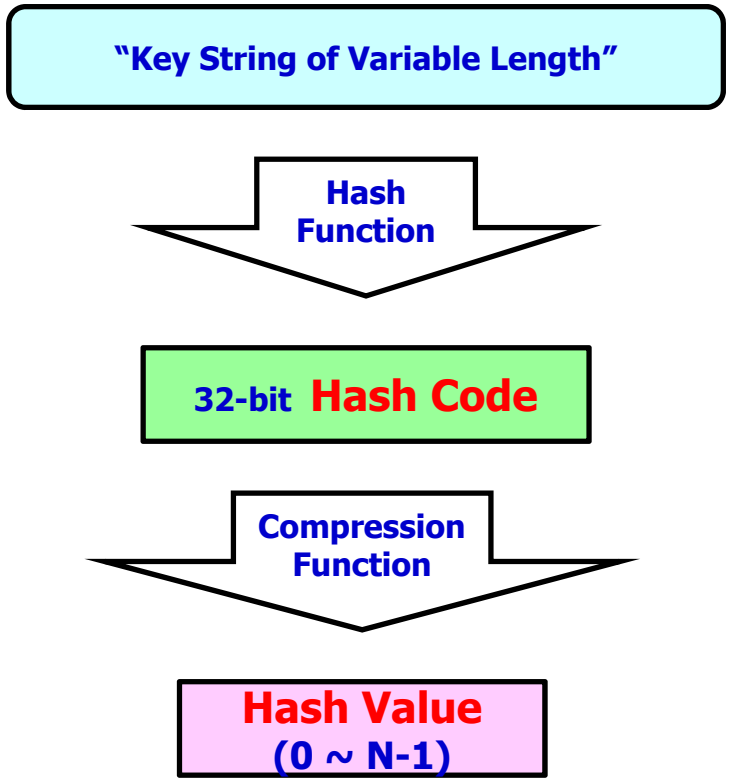
  (ii) Compression function:

  h_2: integers $\rightarrow$ [0, N - 1]

- The hash code is applied first, and the compression function is applied next on the result, i.e.,

  $h(x) = h_2(h_1(x))$

- The goal of the hash function is to "disperse" the keys in an apparently random way



Compression Functions

◆ Divide (modulo) Operation for Compression

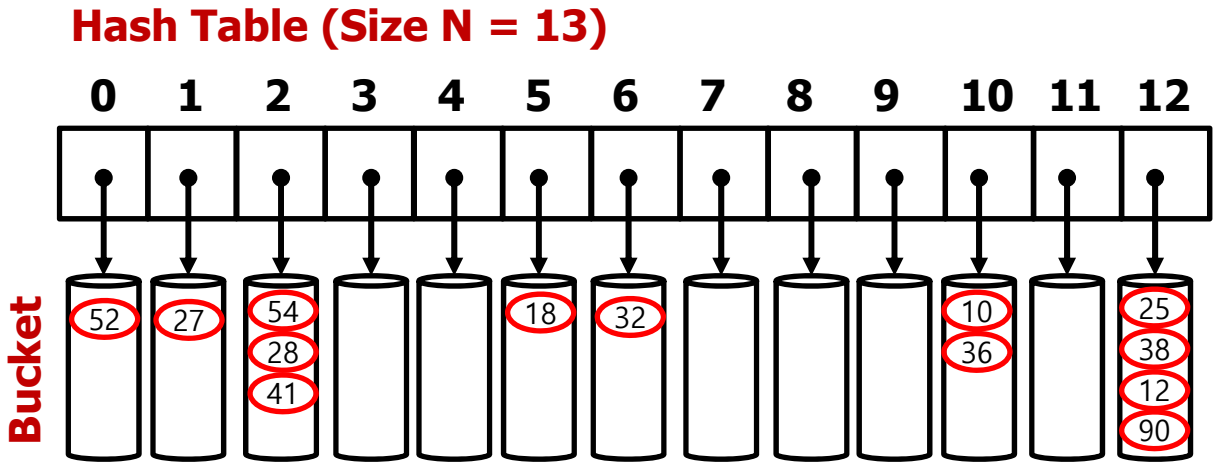
- $h(k) = |k| \mod N$
- The size N of the hash table is usually chosen to be a prime
- The reason has to do with number theory and is beyond the scope of this course

◆ Multiply, Add and Divide (MAD)

- $h(k) = |ak + b| \mod N$
- a and b are nonnegative integers such that
  $a \mod N \neq 0$
- Otherwise, every integer would map to the same value b

Collision Handling

- Collisions occur when different elements are mapped to the same cell
- **Separate Chaining:** let each cell in the table point to a linked list of entries that map there
- Separate chaining is simple, but requires additional memory outside the table
- Example of hash table (N = 13)



Map with Separate Chaining

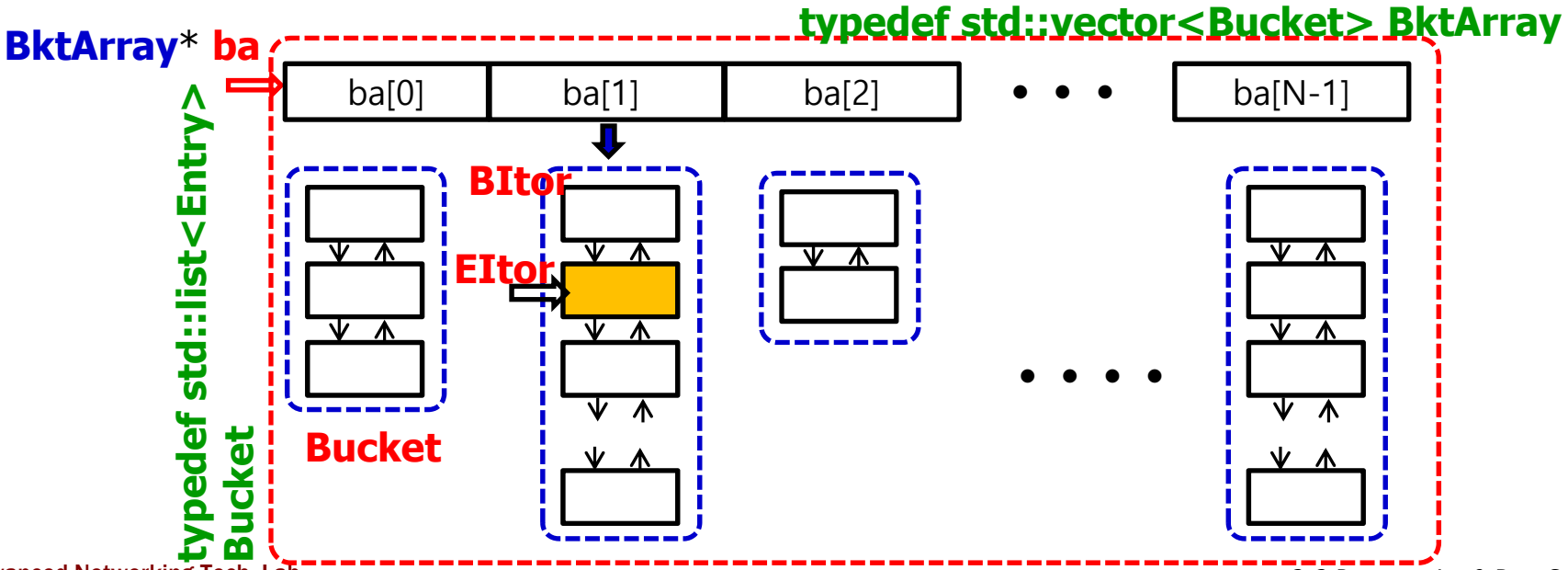
Delegate operations to a list-based bucket at each cell:

```
Algorithm find(k):
    Output: the position of the matching entry of the map,
                or  end if there is no key k in the map
    return A[h(k)].search(k)
        // delegate the find(k) to the list-based bucket at A[h(k)]

Algorithm insert(k, v):
    p = A[h(k)].insert(k, v)
        // delegate the insert(k, v) to the list-based bucket at A[h(k)]
    n = n + 1
    return p

Algorithm erase(k):
    Output: none
    A[h(k)].erase(k)
        // delegate the erase(k) to the list-based bucket at A[h(k)]
    n = n - 1
```

Hash Map 구현

◆ Bucket, Bucket Array, and Iterator

- std::list<Entry> **Bucket**
- std::vector<Bucket> **BktArray**
- HashMap::**Iterator** contains
  - const BktArray* ba; // which bucket array in the hash table
  - BItor bkt; // which bucket in the bucket array (hash table)
  - EItor ent; // which entry in the bucket



C++ Hash Map Implementation

◆ HashMap (1)

```
 /**    HashMap.h   (1)  */
#ifndef HASHMAP_H
#define HASHMAP_H
#include <list>
#include <vector>
#include "Entry.h"
#include "Exceptions.h"
#include "CyclicShiftHashCode.h"
#define  DEFAULT_HASH_TABLE_SIZE  101

template <typename K, typename V> // key, value, hash
class HashMap {
private:
    int num_ent;          // number of entries
    BktArray B;     // bucket array (Hash Table)
public:                   // public types
    typedef Entry<const K,V> Entry;     // a (key, value) pair
     typedef Entry<const K, V> Entry;// a (key,value) pair
     typedef std::list<Entry> Bucket;// a bucket of entries
     typedef std::vector<Bucket> BktArray;// a bucket array
     typedef typename BktArray::iterator BItor;// bucket iterator
     typedef typename Bucket::iterator EItor;// entry iterator
     class Iterator;
```

◆ HashMap (2)

```
/**    HashMap.h   (2)  */

public:
    HashMap(int capacity = DEFAULT_HASH_TABLE_SIZE);     // constructor
    int size() const  {return num_entry; }   // number of entries
    bool empty() const { return (num_entry == 0); } // is the map empty?
    Iterator find(const K& k);          // find entry with key k
    Iterator insert(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 of HashMap
    Iterator end();               // iterator to end entry of HashMap
    void fprintBucketSizes(ostream& fout); // printout bucket sizes
    void fprintBucket(ostream& fout, BItor bkt);

protected:              // protected types
     // HashMap utilities here
    Iterator _find(const K& k);          // find utility
    Iterator _insert(const Iterator& p, const Entry& e);   // insert utility
    void _erase(const Iterator& p);      // remove utility
    static bool _endOfBkt(const Iterator& p)     // end of bucket?
      { return p.ent == p.bkt->end(); }
```

◆ HashMap (3)

```
 /**    HashMap.h   (3)  */

public:          // public types
    // Iterator class declaration
    class Iterator {     // an HashMap::Iterator (& position)
    private:
            const BktArray* ba;   // which bucket array in the hash table
            BItor bkt;  // iterator of bucket in the bucket array (hash table)
            EItor ent;  // iterator of the bucket (iterator of list)
    public:
            Iterator(const BktArray& a, const BItor& b, const EItor& q = EItor())
                : ba(&a), bkt(b), ent(q) { }
            Iterator() {} // default constructor
            Entry& operator*() const;     // get entry
            bool operator==(const Iterator& p) const;   // are iterators equal?
            bool operator!=(const Iterator& p) const;   // are iterators different ?
            Iterator& operator++();     // advance to next entry
            friend class HashMap;     // give HashMap access
    }; // end class Iterator
}; // end of class HashMap

#endif
```

◆ HashMap (4)

```
/**   HashMap.cpp  (1)  */

#include <iostream>
#include "HashMap.h"
#include "Entry.h"

using namespace std;

template <typename K, typename V>          // constructor
HashMap<K,V>::HashMap(int capacity) : num_entry(0), B(capacity) { }

template <typename K, typename V>     // iterator to front
typename HashMap<K,V>::Iterator HashMap<K,V>::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
}

template <typename K, typename V>     // iterator to end
typename HashMap<K,V>::Iterator HashMap<K,V>::end()
{
     return Iterator(B, B.end());
}
```

◆ HashMap (5)

```
/**   HashMap.cpp  (2)  */

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

template <typename K, typename V>  // are iterators equal?
bool HashMap<K,V>::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
}

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

◆ HashMap (6)

```
/**   HashMap.cpp  (3)  */

template <typename K, typename V>     // advance to next entry
typename HashMap<K,V>::Iterator& HashMap<K,V>::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 (7)

```
/**   HashMap.cpp  (4)  */

template <typename K, typename V>     // find utility
typename HashMap<K,V>::Iterator HashMap<K,V>::_find(const K& k)
{
   CyclicShiftHashCode  hash;
   int i = hash(k) % B.size();        // calculate hash value i, using CyclicShiftHashCode()
   BItor bkt = B.begin() + i;          // the i-th bucket
   Iterator p(B, bkt, bkt->begin());              // start of i-th bucket
   while (!_endOfBkt(p) && (*p).key() != k)    // linear search for k in the bucket
        ++p.ent;
   return p;                          // return final position
}

template <typename K, typename V>     // find key
typename HashMap<K,V>::Iterator HashMap<K,V>::find(const K& k)
{
   Iterator p = _find(k);         // look for k
   if (_endOfBkt(p))                 // if could not find the given key?
     return end();                  // return end iterator
   else
     return p;                      // return its position
}
```

◆ HashMap (8)

```
/**   HashMap.cpp  (5)  */

template <typename K, typename V>      // insert utility
typename HashMap<K,V>::Iterator
HashMap<K,V>::_insert(const Iterator& p, const Entry& e) {
    EItor ins = p.bkt->insert(p.ent, e);     // insert before p using insert() of list<Entry>
     num_entry ++;                    // one more entry
    return Iterator(B, p.bkt, ins);     // return this position
}

template <typename K, typename V>     // insert/replace (v,k)
typename HashMap<K,V>::Iterator
HashMap<K,V>::insert(const K& k, const V& v) {
    Iterator p = _find(k);                    // search for k
    if (_endOfBkt(p)) {                       // k not found?
          return _insert(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 (9)

```
/**   HashMap.cpp  (6)  */

template <typename K, typename V>  // remove utility
void HashMap<K,V>::_erase(const Iterator& p) {
    p.bkt->erase(p.ent);              // remove entry from bucket
     num_entry --;                      // one fewer entry
}

template <typename K, typename V>  // remove entry at p
void HashMap<K,V>::erase(const Iterator& p)
   { _erase(p); }

template <typename K, typename V>  // remove entry with key k
void HashMap<K,V>::erase(const K& k) {
    Iterator p = _find(k);              // find k
    if (_endOfBkt(p))                   // not found?
          throw NonexistentElement("Erase of nonexistent");  // ...error
    _erase(p);                    // remove it
}
```

◆ HashMap (10)

```
template <typename K, typename V>
void HashMap<K, V>::fprintBucket(ostream& fout, BItor bkt)
{
    Iterator p(B, bkt, bkt->begin());
    MyVoca* pVoca;
    while (p.ent != bkt->end())
    {
        pVoca = p.getValue();
        fout << *pVoca << endl;
        ++p.ent;
    }
}
```

```
template <typename K, typename V>
void HashMap<K, V>::fprintBucketSizes(ostream& fout)
{
    int bkt_size;
    int max_ent, min_ent, total;
    int num_bkts, max_bkt = 0;
    double avg = 0.0;
    max_ent = min_ent = B[0].size();
    total = 0;
```

◆ HashMap (11)

```
    num_bkts = B.size();
    for (int bkt = 0; bkt < num_bkts; bkt++)
    {
        bkt_size = B[bkt].size();
        fout << "Bucket[" << setw(3) << bkt << "] : " << bkt_size << " entries"
             << endl;
        if (bkt_size > max_ent )
        {
            max_ent = bkt_size;
            max_bkt = bkt;
        }
        if (bkt_size < min_ent)
        min_ent = bkt_size;
        total += bkt_size;
    }
    avg = total / num_bkts;

    fout.precision(2);
    fout << "\nMax_ent (" << setw(2) << max_ent << "), min_ent (" << setw(
    << min_ent << "), avg (" << setw(5) << avg << ")" << endl;

    fout << "Bucket with maximum (" << max_ent << ") entries : " << endl;
    BItor bkt = B.begin() + max_bkt;// the ith bucket
    fprintBucket(fout, bkt);
}
```

```
Max_ent ( 8), min_ent ( 0), avg (2.4)
Bucket with maximum (8) entries :
semiconductor(n):
  - thesaurus( , )
  - example usage( )
emmission(n):
  - thesaurus( , )
  - example usage( )
material(n):
  - thesaurus( , )
  - example usage( )
configurate(n):
  - thesaurus( , )
  - example usage( )
interrupt(n):
  - thesaurus( , )
  - example usage( )
traversal(n):
  - thesaurus( , )
  - example usage( )
description(n):
  - thesaurus( , )
  - example usage( )
kinematics(n):
  - thesaurus( , )
  - example usage( )
```

◆ CyclicShiftHashCode

```
/** CyclicShiftHashCode.h */

#ifndef CYCLICSHIFTHASHCODE_H
#define CYCLICSHIFTHASHCODE_H
#include <string>

using namespace std;
#define BIT_SHIFTS 5
#define BITS_INT 32

class CyclicShiftHashCode
{
public:
    int operator() (const string key)
    {
        int len = key.length();
        unsigned int h = 0;
        for (int i = 0; i<len; i++)
        {
            h = (h << BIT_SHIFTS) | (h >> (BITS_INT - BIT_SHIFTS));
            h += (unsigned int)key.at(i);
        }

        return h;
    }
};

#endif
```

```
/**  main.cpp  (1) */
#include <iostream>
#include <fstream>
#include <iomanip>
#include "MyVoca.h"
#include "MyVocaList.h"
#include "HashMap.h"
#include "CyclicShiftHashCode.h"
#include "Entry.h"
#include <string>
using namespace std;

#define HASH_TABLE_SIZE 101

void main()
{
    ofstream fout;
    HashMap<string, MyVoca*> thesaurusHashMap("My Thesaurus Hash Map",
        HASH_TABLE_SIZE);
    HashMap<string, MyVoca*> *pHM = &thesaurusHashMap;
    HashMap<string, MyVoca*>::Iterator vocaHM_Iter;
    string keyWord;
    MyVoca* pVoca, voca;
    Entry<string, MyVoca*> vocaEntry;
```

```
/**  main.cpp  (2) */

    fout.open("output.txt");
    if (fout.fail())
    {
        cout << "Fail to open output.txt !!" << endl;
        exit;
    }

    fout << "Testing Thesaurus Hash Map " << endl;
    fout << "Hash Map Name (" << pHM->getName() << "), size ("
          << pHM->size() << ")" << endl;

    fout << "Inserting MyVoca into Hash Map ..." << endl;
    for (int i = 0; i < NUM_MY_TOEIC_VOCA; i++)
    {
        keyWord = myToeicVocaList[i].getKeyWord();
        pVoca = &myToeicVocaList[i];
        pHM->insert(keyWord, pVoca);
    }
```

```
/**  main.cpp  (3) */

    fout << "MyVOCA HashMap after insertions of MyVoca :" << endl;
    vocaHM_Iter = pHM->begin();
    while (vocaHM_Iter != pHM->end())
    {
        vocaEntry = *vocaHM_Iter;
        keyWord = vocaEntry.getKey();
        pVoca = vocaEntry.getValue();
        fout << keyWord << " : " << *pVoca << endl;
        ++vocaHM_Iter;
    }

    // printout the bucket size of HashMap
    pHM->fprintBucketSizes(fout);

    keyWord = myToeicVocaList[NUM_MY_TOEIC_VOCA - 1].getKeyWord();
    fout << "Testing search from MyVOCA HashMap  for keyWord (" << keyWord
        << ") : " << endl;
    vocaHM_Iter = pHM->find(keyWord);
    pVoca = (*vocaHM_Iter).getValue();
    fout << *pVoca << endl;

    fout.close();
}
```

◆ Execution Result

```
Testing Thesaurus Hash Map
Hash Map Name (My Thesaurus Hash Map), size (0)
Inserting MyVoca into Hash Map ...
MyVOCA HashMap after insertions of MyVoca :
offer : offer(v):
  - thesaurus(to propose, )
  - example usage(She must offer her banker new statistics in order to satisfy the bank's requirement for the loan. )

compromise : compromise(v):
  - thesaurus(settle, conciliate, find a middle ground, )
  - example usage(He does not like  sweet dishes so I compromised by adding just a small amount of sugar. )

mean : mean(v):
  - thesaurus(require, denote, intend, )
  - example usage(What do you mean by "perfect" ? )

imperative : imperative(n):
  - thesaurus(necessity, essential, requirement, )
  - example usage( )

delegate : delegate(v):
  - thesaurus(authorize, appoint, designate, )
  - example usage( )

foster : foster(adj):
  - thesaurus(substitute, adoptive, stand-in, )
  - example usage( )

Testing search from MyVOCA HashMap  for keyWord (imperative) :
imperative(n):
  - thesaurus(necessity, essential, requirement, )
  - example usage( )
```

Bucket Sizes of Hash Map

```
Bucket[  0] : 0 entries
Bucket[  1] : 0 entries      .....
Bucket[  2] : 0 entries
Bucket[  3] : 0 entries   Bucket[ 90] : 0 entries
Bucket[  4] : 0 entries   Bucket[ 91] : 0 entries
Bucket[  5] : 0 entries   Bucket[ 92] : 0 entries
Bucket[  6] : 0 entries   Bucket[ 93] : 0 entries
Bucket[  7] : 0 entries   Bucket[ 94] : 0 entries
Bucket[  8] : 0 entries   Bucket[ 95] : 0 entries
Bucket[  9] : 0 entries   Bucket[ 96] : 0 entries
Bucket[ 10] : 0 entries   Bucket[ 97] : 0 entries
Bucket[ 11] : 1 entries   Bucket[ 98] : 0 entries
Bucket[ 12] : 0 entries   Bucket[ 99] : 0 entries
Bucket[ 13] : 0 entries   Bucket[100] : 0 entries
Bucket[ 14] : 0 entries   Max_ent ( 1), min_ent ( 0), avg (    0)
```

Map with Multiple Entries of same Key

◆ Example of multiple entries of same key

```
MyVoca("mean", NOUN, { "average", "norm", "median", "middle", "midpoint", "(ant)
    extremity" }, { "the mean error", "the golden mean", "the arithmetical mean", "the geometric
    mean" }),
MyVoca("mean", ADJ, { "nasty", "poor", "middle", "miserly", "paltry" }, { "a man of mean
    intelligence", "a mean appearance" }),
MyVoca("mean", VERB, { "require", "denote", "intend" }, { "What do you mean by perfect ?" }),
MyVoca("compromise", NOUN, { "give-and-take", "bargaining", "accommodation" }, { "The
    couple made a compromise and ordered food to take out." }),
MyVoca("compromise", VERB, { "settle", "conciliate", "find a middle ground" }, { "He does not
    like  sweet dishes so I compromised by adding just a small amount of sugar." }),
```

◆ STL Multi-map

- STL Multimap provides mapping function for multiple entries of same key

Dictionary

◆ Dictionary ADT

- models a searchable collection of key-element entries
- the main operations of a dictionary are searching, inserting, and deleting items
- ***Multiple items*** with the same key ***are allowed***

◆ Applications of Dictionary Data Structure:

- word-definition pairs
- credit card authorizations
- DNS mapping of host names (e.g., datastructures.net) to internet IP addresses (e.g., 128.148.34.101)

Dictionary ADT

- **Dictionary ADT methods:**
  - size(): return the number of entries in D
  - empty(): return true if D is empty and false otherwise
  - find(k): if there is an entry with key k, returns an iterator p referring any such entry, else return the special iterator end
  - findAll(k): returns a pair of iterators (b, e), such that all entries with key value k lie in the range from b up to, but not including, e: [b, e) // starting at b and ending just prior to e
  - insert(k, v): insert an entry with key k and value v into D, returing an iterator referring to the newly created entry
  - erase(k): remove from D an arbitrary entry with key equal to k; an error condition occurs if D has no such entry
  - erase(p): remove from D the entry referenced by iterator p; an error condition occurs if D has no such entry
  - begin(): return an iterator to the first entry of the dictionary
  - end(): return an iterator to a position just beyond the end of the dictionary

Application of Dictionary

◆ Thesaurus Dictionary

- (https://en.wikipedia.org/wiki/Thesaurus) a **thesaurus** is a reference work that lists words grouped together according to similarity of meaning (containing synonyms and sometimes antonyms), in contrast to a dictionary, which provides definitions for words, and generally lists them in alphabetical order.
- Example)
  - mean [adjective] : nasty, poor, middle, miserly, paltry
    - a man of mean intelligence, a mean appearance
  - mean [noun] : average, norm, median, middle, midpoint, (antonym) extremity
    - the mean error, the golden mean, the arithmetical mean, the geometric mean
  - mean [verb] : require, denote, intend
    - What do you mean by "perfect" ?

(http://www.thesaurus.com/)

C++ Implementation of HashDict

◆ class HashDict (1)

```
/** HashDictionary.h (1) */

#ifndef HASH_DICTIONARY_H
#define HASH_DICTIONARY_H
#include "HashMap.h"
#define HASH_TABLE_SIZE 101

template <typename K, typename V>
class HashDict : public HashMap<K, V> {
public:                                  // public functions
    typedef typename HashMap<K, V>::Iterator Iterator;
    typedef typename HashMap<K, V>::Entry Entry;
    // Range class declaration
    class Range {          // an iterator range
    private:
        Iterator _begin;          // front of range
        Iterator _end;          // end of range
    public:
        Range() {} // default constructor
        Range(const Iterator& b, const Iterator& e)     // constructor
        : _begin(b), _end(e) { }
        Iterator& begin() { return _begin; }     // get beginning
        Iterator& end() { return _end; }     // get end
    }; // end class Range
```

◆ class HashDict (2)

```
/** HashDictionary.h (2) */

public:                              // public functions
    HashDict(int capacity = HASH_TABLE_SIZE);     // constructor
    Range findAll(const K& k);                    // find all entries with k
    Iterator insert(const K& k, const V& v);      // insert pair (k,v)
};  // end class HashDict

template <typename K, typename V>     // constructor
HashDict<K, V>::HashDict(int capacity) : HashMap<K, V>(capacity) { }

template <typename K, typename V>     // insert pair (k,v)
typename HashDict<K, V>::Iterator
HashDict<K, V>::insert(const K& k, const V& v) {
    Iterator p = _find(k);                    // find key
    Iterator q = _insert(p, Entry(k, v));     // insert it here
    return q;                    // return its position
}
```

◆ class HashDict (3)

```
/** HashDictionary.h (3) */

template <typename K, typename V>  // find all entries with k
typename HashDict<K, V>::Range
HashDict<K, V>::findAll(const K& k)
{
    Iterator b = _find(k);              // look up k
    Iterator p = b;
    while (p != end() && (*p).key() == k)
    {    // find next entry with different key or end of bucket array
         ++p;
    }
    return Range(b, p);                 // return range of positions
}

#endif
```

◆ main()

```
/** main.cpp (1) */

#include <iostream>
#include <fstream>
#include <string>
#include "HashMap.h"
#include "HashMap.cpp"
#include "CyclicShiftHashCode.h"
#include "Entry.h"
#include "HashDictionary.h"
#include "MyVoca.h"
#include "MyVocaList.h"

void main()
{
        ofstream fout;
        MyVoca* pVoca, voca;
        List_Str  thesaurus, usages;   // typedef list<string
        int word_count;
        MyVoca mv;
        string keyWord;
        HashDict<string, MyVoca*> myVocaDict;
        HashDict<string, MyVoca*>::Iterator itr;
        HashDict<string, MyVoca*>::Range range;
        Entry<string, MyVoca*> vocaEntry;
```

```
/** main.cpp (2) */


        fout.open("output.txt");
        if (fout.fail())
        {
            cout << "Fail to open output.txt !!" << endl;
            exit;
        }

        fout << "Inserting My Vocabularies to myVocaDict . . . " << endl;
        word_count = 0;
        for (int i = 0; i < NUM_MY_TOEIC_VOCA; i++)
        {
            pVoca = &myToeicVocaList[i];
            keyWord = myToeicVocaList[i].getKeyWord();
            myVocaDict.insert(keyWord, pVoca);
        }

        fout << "Total " << myVocaDict.size() << " words in my Voca_Dictionary .."
             << endl;

        // check all vocabularies in the hash_dictionary
        for (itr = myVocaDict.begin(); itr != myVocaDict.end(); ++itr)
        {
            pVoca = itr.getValue();
            fout << *pVoca << endl;
        }
        fout << endl;
```

```
/** main.cpp (3) */

        // printout bucket sizes of Hash map
        myVocaDict.fprintBucketSizes(fout);
        fout << endl;

        //string testWord = "mean";
        string testWord = "offer";
        range = myVocaDict.findAll(testWord);
        fout << "Thesaurus of [" << testWord << "]: \n";
        for (itr = range.begin(); itr != range.end(); ++itr)
        {
             pVoca = itr.getValue();;
             fout << *pVoca << endl;
        }
        fout << endl;
        fout.close();
}  // end main()
```

◆ Execution Result (1)

```
Inserting My Vocabularies to myVocaDict . . .
Total 13 words in my Voca_Dictionary ..
offer(v):
  - thesaurus(to propose, )
  - example usage(She must offer her banker new statistics in order to satisfy the bank's requirement for the loan. )

offer(n):
  - thesaurus(proposal, )
  - example usage(He accepted out offer to write the business plan. )

compromise(v):
  - thesaurus(settle, conciliate, find a middle ground, )
  - example usage(He does not like  sweet dishes so I compromised by adding just a small amount of sugar. )

compromise(n):
  - thesaurus(give-and-take, bargaining, accomodation, )
  - example usage(The couple made a compromise and ordered food to take out. )

mean(v):
  - thesaurus(require, denote, intend, )
  - example usage(What do you mean by "perfect" ? )

mean(adj):
  - thesaurus(nasty, poor, middle, miserly, paltry, )
  - example usage(a man of mean intelligence a mean appearance )

mean(n):
  - thesaurus(average, norm, median, middle, midpoint, (ant) extremity, )
  - example usage(the mean error the golden mean the arithmetical mean the geometric mean )
```

.....

```
delegate(v):
  - thesaurus(authorize, appoint, designate, )
  - example usage( )

delegate(n):
  - thesaurus(representative, agent, substitute, )
  - example usage( )

foster(adj):
  - thesaurus(substitute, adoptive, stand-in, )
  - example usage( )

foster(v):
  - thesaurus(nurture, raise, promote, advance, )
  - example usage( )
```

Bucket Sizes of HashDict/HashMap

```
Bucket[  0] : 0 entries
Bucket[  1] : 0 entries
Bucket[  2] : 0 entries
Bucket[  3] : 0 entries
Bucket[  4] : 0 entries
Bucket[  5] : 0 entries
Bucket[  6] : 0 entries
Bucket[  7] : 0 entries
Bucket[  8] : 0 entries
Bucket[  9] : 0 entries
Bucket[ 10] : 0 entries
Bucket[ 11] : 2 entries
Bucket[ 12] : 0 entries
Bucket[ 13] : 0 entries
Bucket[ 14] : 0 entries
Bucket[ 15] : 0 entries
```

.....

```
Bucket[ 90] : 0 entries
Bucket[ 91] : 0 entries
Bucket[ 92] : 0 entries
Bucket[ 93] : 0 entries
Bucket[ 94] : 0 entries
Bucket[ 95] : 0 entries
Bucket[ 96] : 0 entries
Bucket[ 97] : 0 entries
Bucket[ 98] : 0 entries
Bucket[ 99] : 0 entries
Bucket[100] : 0 entries
Max_ent ( 3), min_ent ( 0), avg (    0)
```

```
Thesaurus of [offer]:
offer(v):
  - thesaurus(to propose, )
  - example usage(She must offer her banker new statistics in order to satisfy the bank's requirement for the loan. )

offer(n):
  - thesaurus(proposal, )
  - example usage(He accepted out offer to write the business plan. )
```

Oral Test 11

11.1 문자열 (string) 자료형의 키워드에 대한 Hash code 계산에서 주로 많이 사용되는 Cyclic Shift Hash Code 대하여 상세하게 설명하라.

<Key Points>

(1) 키워드가 "Yeungnam"일 때 hash code 값이 각 단계별로 어떻게 계산되는가를 파악하도록 중간 값을 출력하고, 이 계산 과정을 설명할 것

단계 (i)	h (for-loop 시작 단계의 초기값)	h << BIT_SHIFTS	h >> (sizeof(int) - BIT_SHIFTS)	p[i]	h (for-loop 마지막 단계의 결과값)
0					
1					
. . . .					
len-1					

11.2 Hash Map을 STL vector와 STL list로 구현하는 경우, 내부 구조를 그림으로 표현하고, 구현하는 방법에 대하여 상세하게 설명하라 .

<Key Points>

(1) Bucket의 구성
(2) Bucket Array의 구성
(3) 키워드에 대한 Hash Value의 계산
(4) 키워드에 대한 해당 Bucket 탐색
(5) 키워드에 대한 Bucket 내부 Entry 탐색

Oral Test 11

11.3 Hash Map에서 사용하는 class Iterator 구조와 제공 연산자 오버로딩에 대하여 상세하게 설명하라.

<Key Points>

(1) class HashMap<K, V, H>의 class Iterator 구조 설명

(2) class Iterator 의 생성자

(3) class Iterator 의 operator*() 연산자 오버로딩

(4) class Iterator 의 operator++() 연산자 오버로딩

11.4 HashDict에서 사용하는 class Range의 구조와 데이터 멤버 및 멤버함수에 대하여 상세하게 설명하라.

<Key Points>

(1) class HashDict의 class Range 구조 설명

(2) class HashDict의 findAll() 멤버함수의 실행에서 range의 _begin과 _end가 결정되는 과정 설명

(3) range.begin()과 range.end() 를 사용하여 해당 구간의 vocabulary들을 출력하는 방법에 대한 설명