객체지향프로그래밍과 자료구조

Ch 11. Hash Tables, Maps, Multi-map (Dictionary), Skip List

정보통신공학과

교수 김 영 탁

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

Outline

- **Map, STL map**
  - stores duplicated entries (key, value) with same key
- **Hash Tables**
  - bucket array
  - hash function (cyclic shift hash)
  - hash-based array/vector indexing
- **class HashMap**
- **STL multimap**
- **class HashDict**
  - Storing duplicated entries
  - Thesaurus dictionary
  - TOEIC vocabulary dictionary based on HashMap

Map

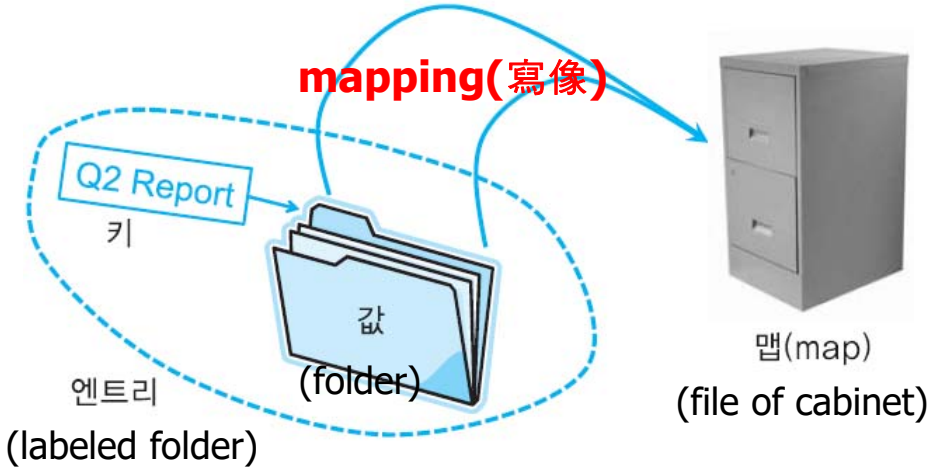
Maps

◆ Map

- models a searchable collection of key-value entries
- the main operations of a map are for **searching**, **inserting**, and **deleting** items
- 1:1 mapping between key and value(entry)
- **multiple entries** with the **same key** are ***not allowed***

◆ Applications of Map:

- address book
- student-record database



class Entry (class Pair)

- **An entry stores a key-value pair (k,v)**
- **Methods:**
  - key(): return the associated key
  - value(): return the associated value
  - setKey(k): set the key to k
  - setValue(v): set the value to v

```
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
};
```

Standard Template Library (STL) map

STL map class

- size() : return the number of elements in the map
- empty() : return true if the map is empty, and false otherwise
- find(k) : find the entry with ***key*** k and return an iterator to it;
  if no such key exists, return end
- operator[k] : produce a reference to the value of ***key*** k;
  if no such key exists, create a new entry for ***key*** k
- insert(pair(k, v)) : insert pair (k, v), returning an iterator to its position
- erase(k) : remove the element with ***key*** k
- erase(p) : remove the element referenced by ***iterator*** p
- begin() : return an iterator to the beginning of the map
- end() : return an iterator just past the end of the map
- equal_range(k) : returns the range of entries of the give key in multimap

STL map<K, V>::iterator itr

- returns a pointer to (key, value) pair
- itr->first // key
- itr->second // value

◆ Example Usage of STL map for class MyVoca

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

#ifndef MY_VOCA_H
#define MY_VOCA_H

#include <iostream>
#include <string>
#include <list>
using namespace std;

enum Word_Type {NOUN, VERB, ADJ, ADV, PREPOS}; // noun, verb, adjective, adverbs, preposition

typedef list<string> List_Str;
typedef list<string>::iterator Lst_Str_Itr;

class MyVoca
{
    friend ostream& operator<<(ostream& fout, MyVoca& mv)
    {
        const string wd_ty[] = { "n", "v", "adj", "adv", "prepos" };
        list<string>::iterator itr;
        fout << mv.keyWord << "(" << wd_ty[mv.type] << "): ₩n";
```

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

            fout << "  - thesaurus(";
            for (itr = mv.thesaurus.begin(); itr != mv.thesaurus.end(); ++itr)
            {
                  fout << *itr << ", ";
            }
            fout << ")" << endl;
            fout << "  - example usage(";
            for (itr = mv.usages.begin(); itr != mv.usages.end(); ++itr)
            {
                  fout << *itr << " ";
            }
            fout << ")";
            return fout;
      }
public:
      MyVoca(string kw, Word_Type wt, List_Str thes, List_Str ex_usg) :keyWord(kw), type(wt),
          thesaurus(thes), usages(ex_usg) {}
      MyVoca() {} // default constructor
      string getKeyWord() { return keyWord; }

private:
      string keyWord; // entry word (also key)
      Word_Type type;
      List_Str thesaurus; // thesarus of the entry word in the type
      List_Str usages;
};
#endif
```

```
/* MyVocaList.h */
#ifndef MY_VOCA_LIST_H
#define MY_VOCA_LIST_H

int NUM_MY_TOEIC_VOCA = 13;
MyVoca myToeicVocaList[]; // defined in MyVocaList.cpp

#endif
```

```
/* MyVocaList.cpp */
#include "MyVoca.h"
MyVoca  myToeicVocaList[] =
{
        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("offer", NOUN, { "proposal" }, { "He accepted out offer to write the business plan." }),
        MyVoca("offer", VERB, { "to propose" }, { "She must offer her banker new statistics in order to satisfy the bank's
                requirement for the loan." }),
        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." }),
        MyVoca("delegate", NOUN, { "representative", "agent", "substitute" }, { "" }),
        MyVoca("delegate", VERB, { "authorize", "appoint", "designate" }, { "" }),
        MyVoca("foster", VERB, { "nurture", "raise", "promote", "advance" }, { "" }),
        MyVoca("foster", ADJ, { "substitute", "adoptive", "stand-in" }, { "" }),
        MyVoca("imperative", ADJ, { "authoritative", "vital" }, { "" }),
        MyVoca("imperative", NOUN, { "necessity", "essential", "requirement" }, { "" }),
        //{ "-1", NOUN, { "" }, { "" } }, // end sentinel
};
```

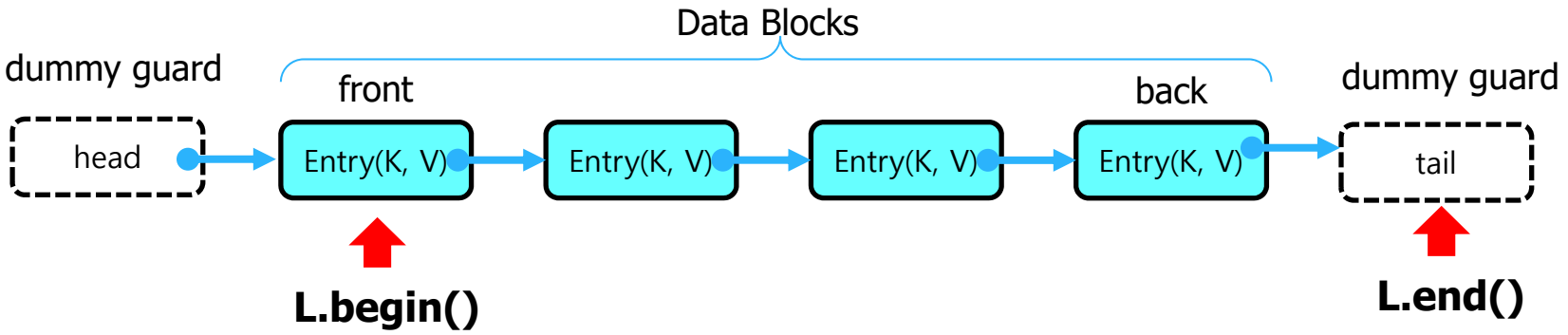
```
/* main_multimap  (1) */
#include <iostream>
#include <fstream>
#include <map>
#include <string>
#include "MyVoca.h"
#include "MyVocaList.h"
using namespace std;

template <typename K, typename V>
void fprint_myVocaMap(ostream& fout, const map<K, V>& mapVoca)
{
        // fprint for STL map
        for (auto itr = mp.begin(); itr != mp.end(); itr++) {
             fout << " [" << itr->first << " : " << *(itr->second) << "\n ]" << endl;
        }
}

int main()
{
    ofstream fout;
    string keyWord;
    string testWord = "mean";
    MyVoca* pVoca, voca;

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

```
/* main_multimap  (2) */

    map<string, MyVoca *> mapVoca;
    fout << "Inserting My Vocabularies to STL map ..... " << endl;
    for (int i = 0; i < NUM_MY_TOEIC_VOCA; i++)
    {
        keyWord = myToeicVocaList[i].getKeyWord();
        pVoca = &myToeicVocaList[i];
        mapVoca.insert(make_pair(keyWord, pVoca));  // make_fair() in std lib
    }
    fout << "Initialized status of myVocaMap : " << endl;
    fprint_myVocaMap(fout, mapVoca);

    fout << "--------------------" << endl;

    //keyWord = myToeicVocaList[0].getKeyWord();
    keyWord = string("compromise");
    auto range = mapVoca.equal_range(keyWord);
    for (auto itr = range.first; itr != range.second; itr++)
    {
        fout << itr->first << " : " << *(itr->second) << " " << endl;
    }

    fout.close();
}
```

```
Inserting My Vocabularies to multimap .....
Initialized status of myVocaMultimap :
 [compromise : compromise(n):
  - thesaurus(give-and-take, bargaining, accomodation, )
  - example usage(The couple made a compromise and ordered food to take out. )
 ]
 [delegate : delegate(n):
  - thesaurus(representative, agent, substitute, )
  - example usage( )
 ]
 [foster : foster(v):
  - thesaurus(nurture, raise, promote, advance, )
  - example usage( )
 ]
 [imperative : imperative(adj):
  - thesaurus(authoritative, vital, )
  - example usage( )
 ]
 [mean : mean(n):
  - thesaurus(average, norm, median, middle, midpoint, (ant) extremity, )
  - example usage(the mean error the golden mean the arithmetical mean the geometric mean )
 ]
 [offer : offer(n):
  - thesaurus(proposal, )
  - example usage(He accepted out offer to write the business plan. )
 ]
--------------------
compromise : compromise(n):
  - thesaurus(give-and-take, bargaining, accomodation, )
  - example usage(The couple made a compromise and ordered food to take out. )
```

A Simple List-Based Map

- **We can easily 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



find() of Map

Algorithm find(k):

input: a key k

output: the position of the matching entry of L,
or end if there is no key k in L

for each position p ∈ [L.begin(), L.end()) do
if (p.key() == k) then
return p

return end // there is no entry with key equal to k

insert() of Map

Algorithm insert(k,v):

input: a key-value pair (k, v)

output: the position of the inserted/modified entry

for each position p ∈ [L.begin(), L.end()) do

if (p.key() == k) then

*p ← (k, v)

return p

p ← L.insertBack((k,v)) **// there is no entry with key k**

n ← n + 1 **// increment number of entries**

return p

erase() of Map

Algorithm erase(k):

input: a key k

output: none

for each position p ∈ [L.begin(), L.end()) do

if (p.key() == k) then

L.erase(p)

n ← n – 1 **// decrement number of entries**

Performance of a List-Based Map

Performance:

- insert() takes $O(n)$ time since we need to determine whether it is already in the sequence
- find() and erase() take $O(n)$ time since in the worst case (the item is not found) we traverse the entire sequence to look for an item with the given key

Implementation of map with unsorted list

- effective only for maps of small size or for maps in which puts are the most common operations,
  while searches and removals are rarely performed (e.g., historical record of logins to a workstation)

Hash, Hash Table

Hash Functions

- A hash value calculation is usually composed of two functions:

  (i) Hash function:

  h_1: keys $\rightarrow$ integers

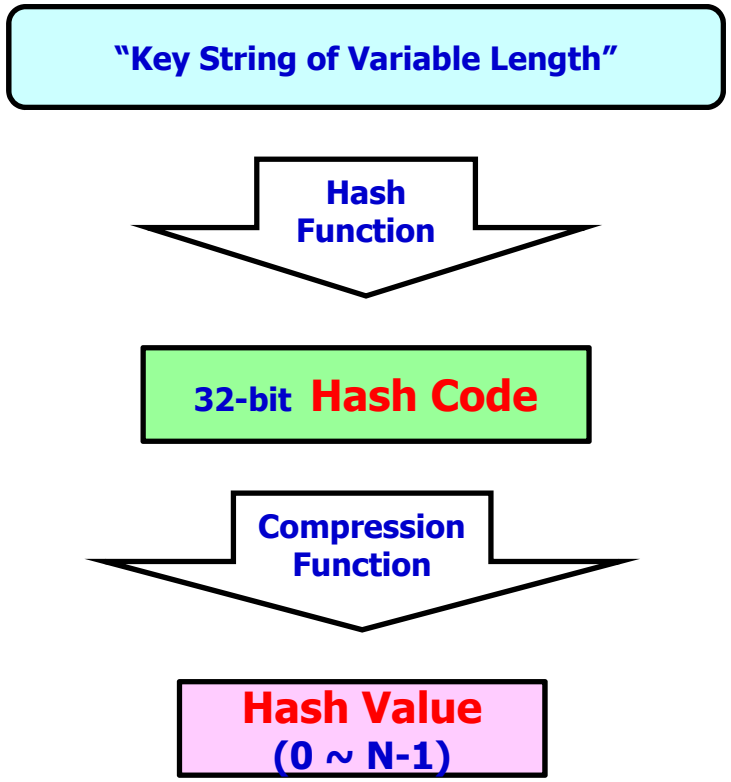
  (ii) Compression function:

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

- The hash function 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 value calculation is to "disperse" the keys in an apparently random way



Hash Calculation, Hash Table

- A hash calculation function $h() = h_2(h_1(x))$ maps keys of a given type (e.g., string of variable length) to integers in a fixed interval [0, N - 1]
- Example:

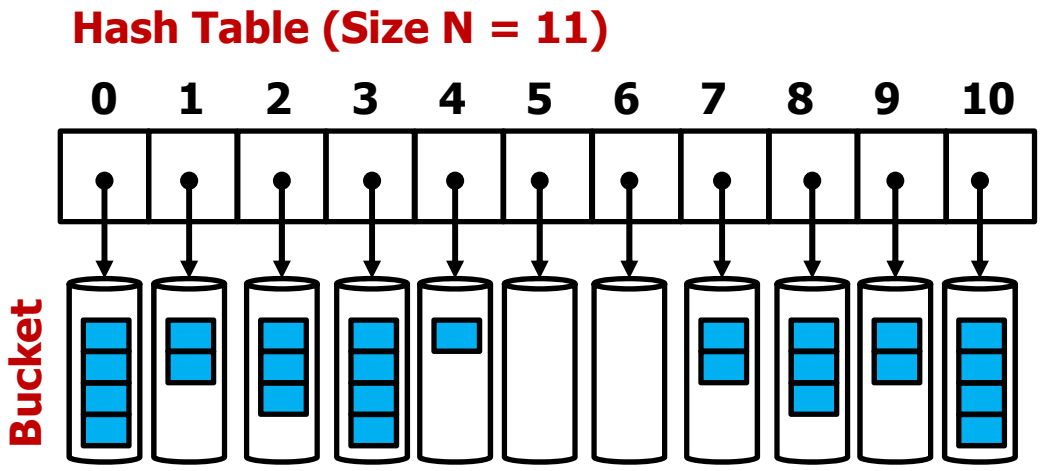
  $h(x) = x \bmod N$

  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



Polynomial Hash Codes

◆ Polynomial accumulation:

- We partition the bits of the key into a sequence of components of fixed length (e.g., 8, 16 or 32 bits)

  $$x_0 a^{k-1} + x_1 a^{k-2} + \ldots + x_{k-2} a + x_{k-1}$$

- We evaluate the polynomial

  $$p(a) = x_{k-1} + a(x_{k-2} + \ldots + a(x_2 + a(x_1 + ax_0)) \ldots)$$

  at a fixed value a, ignoring overflows

- Especially suitable for strings (e.g., the choice $a = 33$ gives at most 6 collisions on a set of 50,000 English words)

Cyclic Shift Hash Codes

◆ Cyclic Shift Hash Code

```
#define BIT_SHIFTS 5
#define BITS_INT 32

int CyclicShiftHash (const char* p, int len)
{
    unsigned int h = 0;
    for (int i=0; i< len; i++)
    {
        h = (h << BIT_SHIFTS) |
            (h >> (BITS_INT - BIT_SHIFTS));
        h += (unsigned int) p[i];
    }
    return h;
}
```

◆ Experimental results

- comparisons of the number of collisions for various shift amounts for 25,000 English words

shift	collisions		shift	collisions	
	total	max		total	max
0	23739	86	9	18	2
1	10517	21	10	277	3
2	2254	6	11	453	4
3	448	3	12	43	2
4	89	2	13	13	2
5	4	2	14	135	3
6	6	2	15	1082	6
7	14	2	16	8760	9
8	105	2			

Compression Functions

◆ Divide (modulo) Operation for Compression

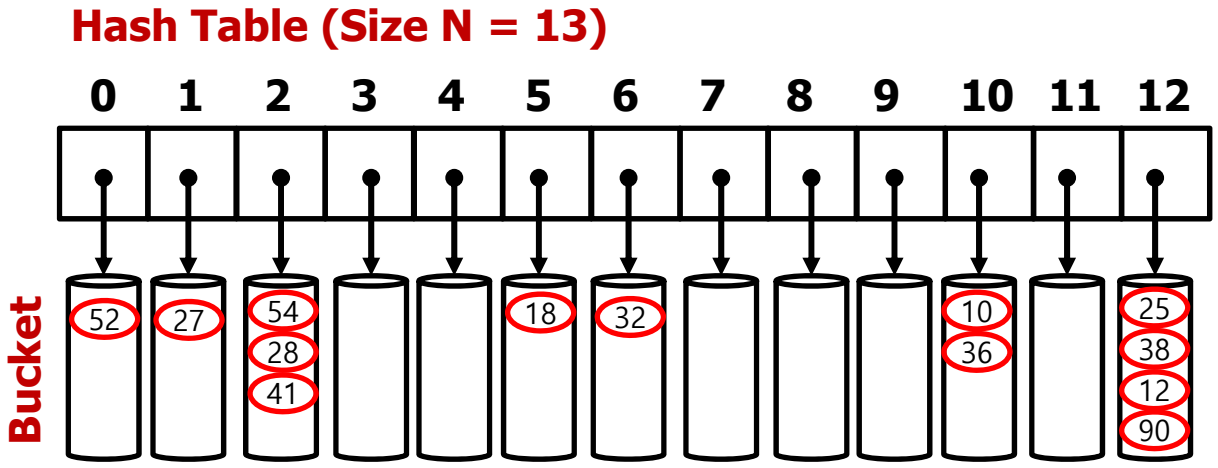
- $h(k) = |k| \mod N$
- The size N of the hash table is usually chosen to be a prime
- The reason has to do with number theory and is beyond the scope of this course

◆ 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:

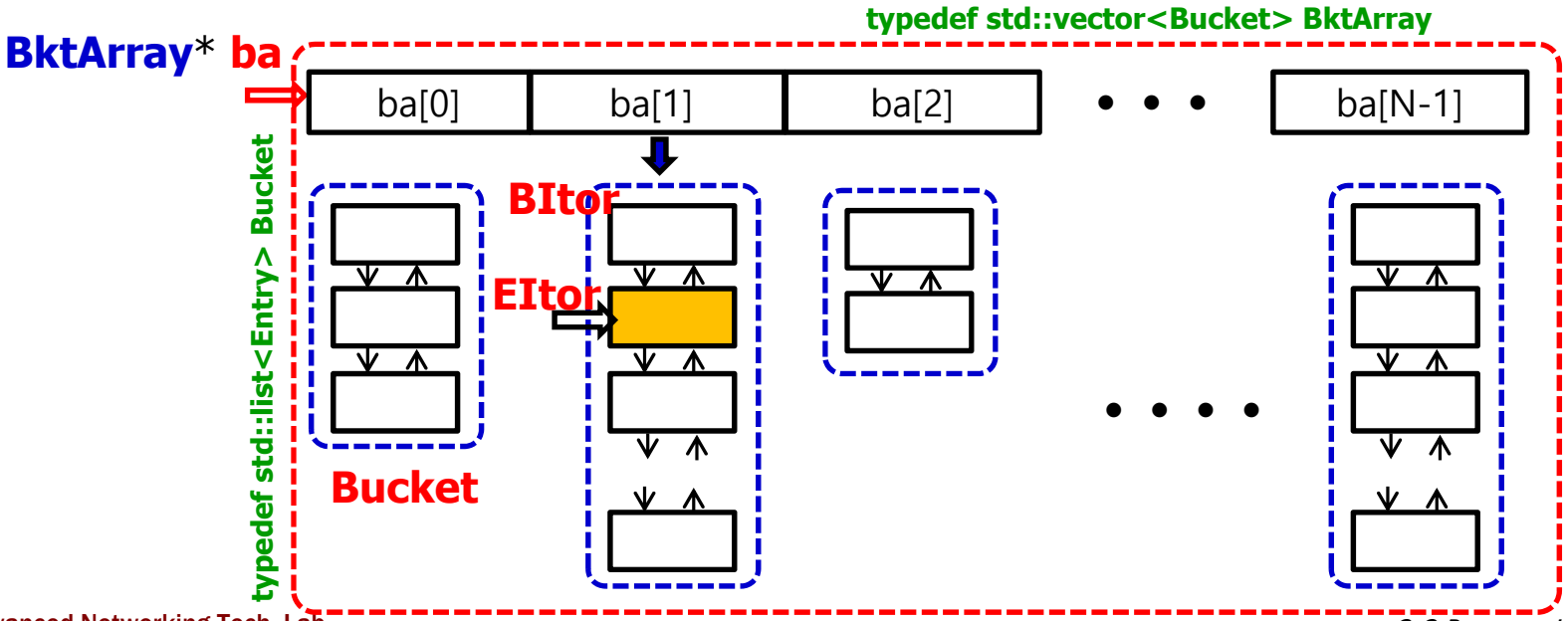
```
Algorithm find(k):
    Output: the position of the matching entry of the map,
                or  end if there is no key k in the map
    return A[h(k)].search(k)
        // delegate the find(k) to the list-based bucket at A[h(k)]

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 Map의 구현 및 활용

Hash Map 구현

◆ Bucket, Bucket Array, and Iterator

- std::list<Entry> **Bucket**
- std::vector<Bucket> **BktArray** // hash table
- 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



◆ class 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) // operator overloading of ()
    {
        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
```

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 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( )
```

```
/**  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.value();
        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).value();
    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)
```

Multi-Map, Dictionary

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

Example Usage of STL multimap

```
/** Sample usage of STL multimap (1) */
#include <iostream>
#include <fstream>
#include <map>
#include <string>
#include "MyVoca.h"
#include "MyVocaList.h"
using namespace std;

template <typename K, typename V>
void fprint_myVocaMultimap(multimap<K, V>& mm)
{
    // printout all entries in multimap
    for (auto itr = mm.begin(); itr != mm.end(); itr++)
    {
        cout << " [" << itr->first << " : " << *(itr->second) << "₩n ]" << endl;
    }
}
```

```
/** Sample usage of STL multimap (2) */

int main()
{
    ofstream fout;
    string keyWord;
    string testWord = "mean";
    MyVoca* pVoca, voca;

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

    multimap<string, MyVoca *> mm;
    fout << "Inserting My Vocabularies to multimap . . . " << endl;
    for (int i = 0; i < NUM_MY_TOEIC_VOCA; i++)
    {
        keyWord = myToeicVocaList[i].getKeyWord();
        pVoca = &myToeicVocaList[i];
        mm.insert(make_pair(keyWord, pVoca));
    }
    fprint_myVocaMultimap(mm);
```

```
/** Sample usage of STL multimap (3) */

    cout << "--------------------" << endl;

    //keyWord = myToeicVocaList[0].getKeyWord();
    keyWord = string("compromise");
    auto range = mm.equal_range(keyWord);
    for (auto itr = range.first; itr != range.second; itr++) {
        cout << itr->first << " : " << *(itr->second) << " " << endl;
    }

    fout.close();
} // end main()
```

```
Inserting My Vocabularies to multimap .....
Initialized status of myVocaMultimap :
 [compromise : compromise(n):
  - thesaurus(give-and-take, bargaining, accomodation, )
  - example usage(The couple made a compromise and ordered food to take out. )
 ]
 [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. )
 ]
 [delegate : delegate(n):
  - thesaurus(representative, agent, substitute, )
  - example usage( )
 ]
 [delegate : delegate(v):
  - thesaurus(authorize, appoint, designate, )
  - example usage( )
 ]
 [foster : foster(v):
  - thesaurus(nurture, raise, promote, advance, )
  - example usage( )
 ]
 [foster : foster(adj):
  - thesaurus(substitute, adoptive, stand-in, )
  - example usage( )
 ]
 [imperative : imperative(adj):
  - thesaurus(authoritative, vital, )
  - example usage( )
 ]
 [imperative : imperative(n):
  - thesaurus(necessity, essential, requirement, )
  - example usage( )
 ]
 [mean : mean(n):
  - thesaurus(average, norm, median, middle, midpoint, (
  - example usage(the mean error the golden mean the ari
 ]
 [mean : mean(adj):
  - thesaurus(nasty, poor, middle, miserly, paltry, )
  - example usage(a man of mean intelligence a mean app
 ]
 [mean : mean(v):
  - thesaurus(require, denote, intend, )
  - example usage(What do you mean by "perfect" ? )
 ]
                                                        [mean : mean(n):
                                                         - thesaurus(average, norm, median, middle, midpoint, (ant) extremity, )
                                                         - example usage(the mean error the golden mean the arithmetical mean the geometric mean )
                                                        ]
                                                        [mean : mean(adj):
                                                         - thesaurus(nasty, poor, middle, miserly, paltry, )
                                                         - example usage(a man of mean intelligence a mean appearance )
                                                        ]
                                                        [mean : mean(v):
                                                         - thesaurus(require, denote, intend, )
                                                         - example usage(What do you mean by "perfect" ? )
                                                        ]
                                                        [offer : offer(n):
                                                         - thesaurus(proposal, )
                                                         - example usage(He accepted out offer to write the business plan. )
                                                        ]
                                                        [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(n):
                                                         - thesaurus(give-and-take, bargaining, accomodation, )
                                                         - example usage(The couple made a compromise and ordered food to take out. )
                                                        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. )
```

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).value();
            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).value();
             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. )
```

Skip List 개요

Skip List 개요

◆ Linked List와 Binary Search Tree의 장단점

자료구조	장점	단점
Linked List	▪ 배열과 비교하여 정렬된 상태의 삽입과 삭제가 빠름	▪ 정렬된 상태로 유지하기 위하여 탐색 (search)이 느림 (O(N))
Binary Search Tree	▪ 항상 정렬된 상태를 유지 ▪ 균형화가 잘 되어 있는 경우 탐색(search)이 빠름	▪ 신속한 탐색을 위하여 항상 재균형화 (rebalancing)을 실행하여야 함

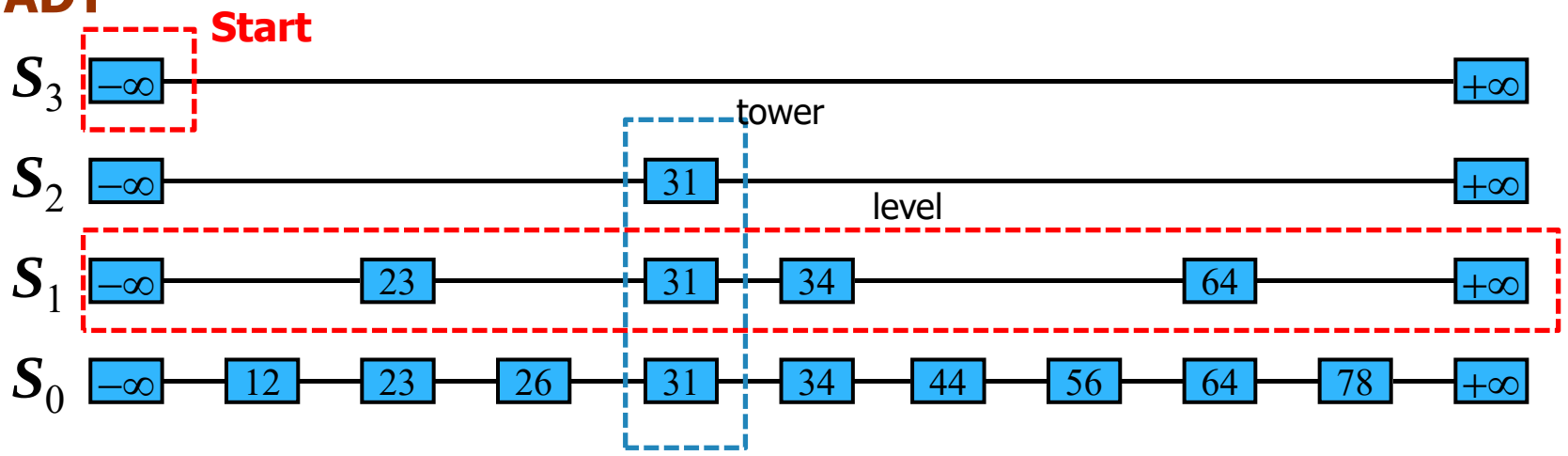
◆ Skip List

- Binary Search Tree의 재균형화 (rebalancing)를 실행하지 않고, 2차원 Linked list의 구조를 구성하여 균형잡힌 Binary Search Tree의 우수한 성능 (삽입, 삭제, 탐색)을 가지도록 함

Skip List

- **A skip list for a set S of distinct (key, element) items is a series of lists $S_0, S_1, \ldots, S_h$ such that**
  - Each list S_i contains the special keys $+\infty$ (PLUS_INF) and $-\infty$ (MINUS_INF)
  - List S_0 contains all keys of S in non-decreasing order
  - Each list is a subsequence of the previous one, i.e.,
    $$S_0 \supseteq S_1 \supseteq \ldots \supseteq S_h$$
  - List S_h contains only the two special keys ($+\infty$ and $-\infty$)

- **We show how to use a skip list to implement the dictionary ADT**



Search and Update Operations in a Skip List

◆ Operations for traversal in Skip List

- after(p) : return the position following *p* on the same level
- before(p) : return the position preceding *p* on the same level
- below(p) : return the position below *p* in the same tower
- above(p) : return the position above *p* in the same tower

$p_{h,j}$: j-th (tower) position at level h
after($p_{h,j}$) : $p_{h,j+1}$
before($p_{h,j}$) : $p_{h,j-1}$
below($p_{h,j}$) : $p_{h-1,j}$
above($p_{h,j}$) : $p_{h+1,j}$

◆ Search for a key *k* in a skip list:

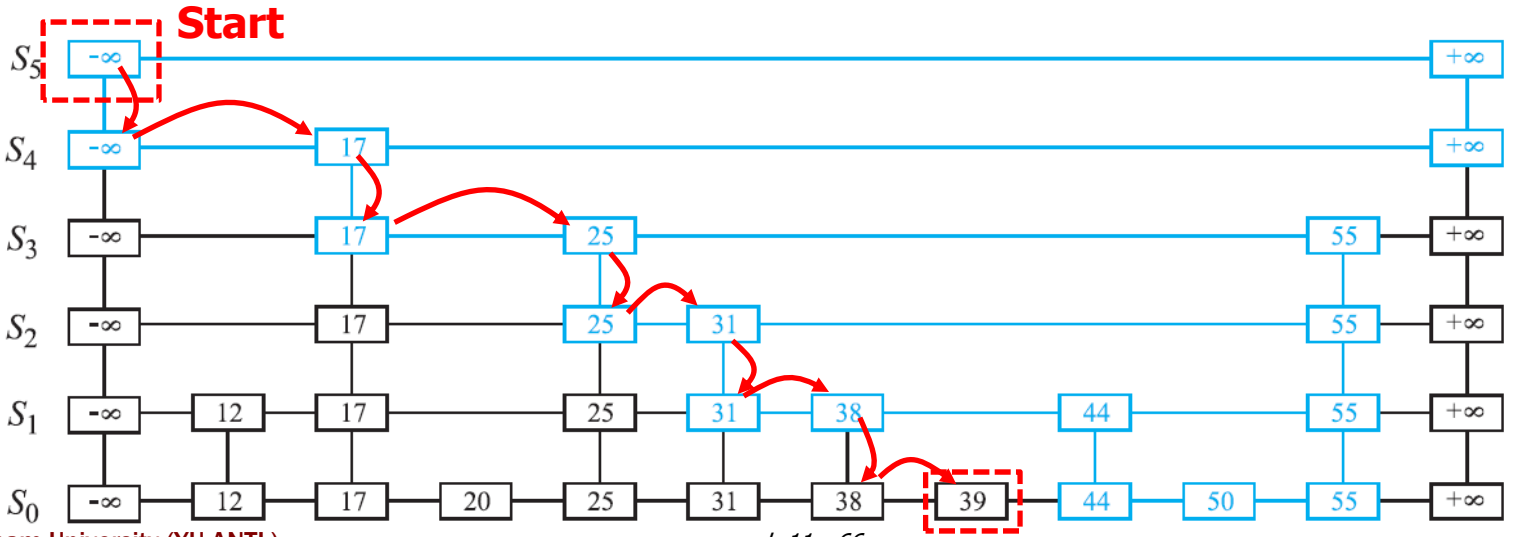
- Start at the first position of the top list (S_h)
- Continuously traverse the skip list, using skipping in higher level to find the proper range of the given key as fast as possible
- If it tries to drop down past the bottom list, we return ***null***

Search a SkipList

Algorithm **SkipSearch(k)**
Input: a search key k
Output: position p in the bottom list S_0 such that the entry at p has the largest key less than or equal to k

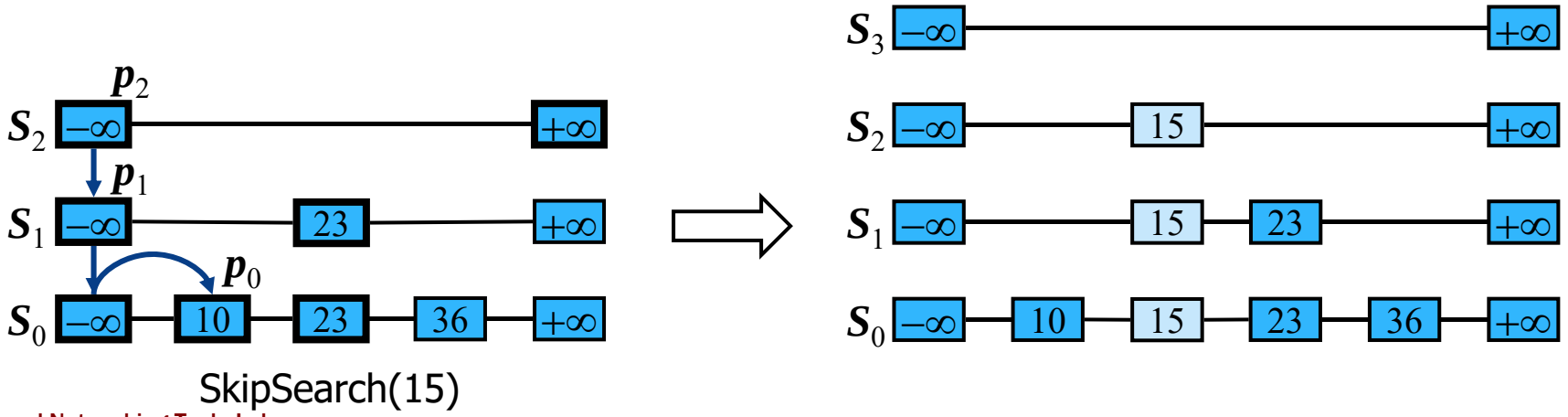
```
p = start   // start position
while (below(p) != NULL) do
      p = below(p)  // drop-down
      while ( k ≥ key (after(p)) do
            p = after(p)
return p
```

◆ example of searching 39 from the skip list



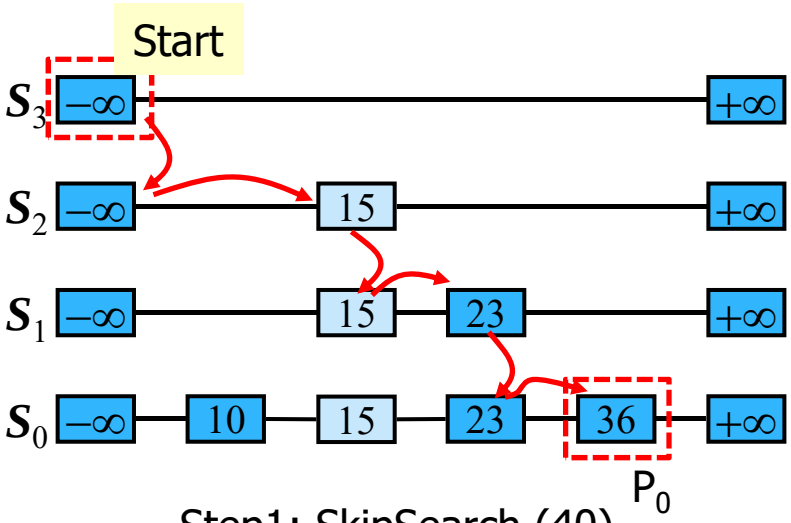
Insertion

- **To insert an entry (k, e) into a skip list, we use a randomized algorithm:**
  - We repeatedly toss a coin until we get tails, and we denote with i the number of times the coin came up heads
  - If $i \geq h$, we add to the skip list new lists $S_{h+1}, \ldots, S_{i+1}$, each containing only the two special keys
  - We search for k in the skip list and find the positions $p_0, p_1, \ldots, p_i$ of the items with largest key less than k in each list $S_0, S_1, \ldots, S_i$
  - For $j \leftarrow 0, \ldots, i$, we insert item (k, e) into list S_j after position p_j
- **Example: insert key 15, with $i = 2$**

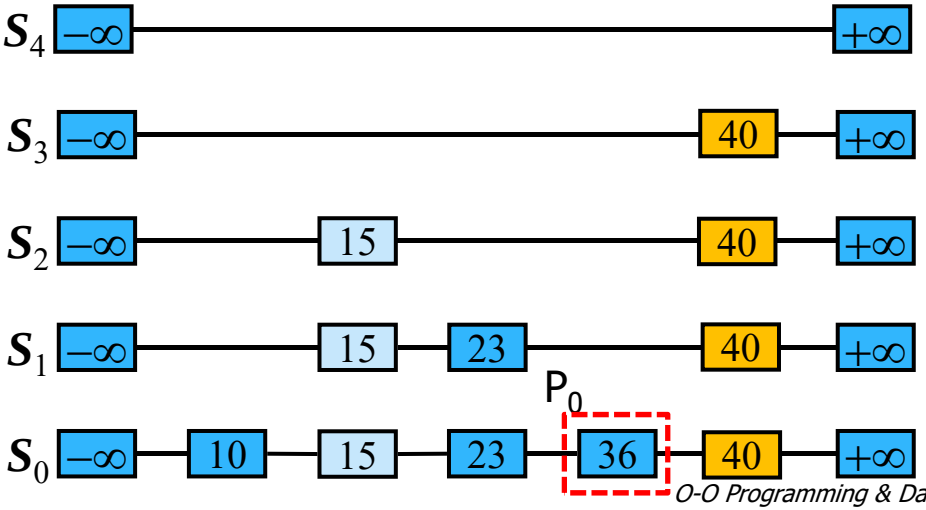


Insertion

◆ Example: insert key 40 with $i = 3$

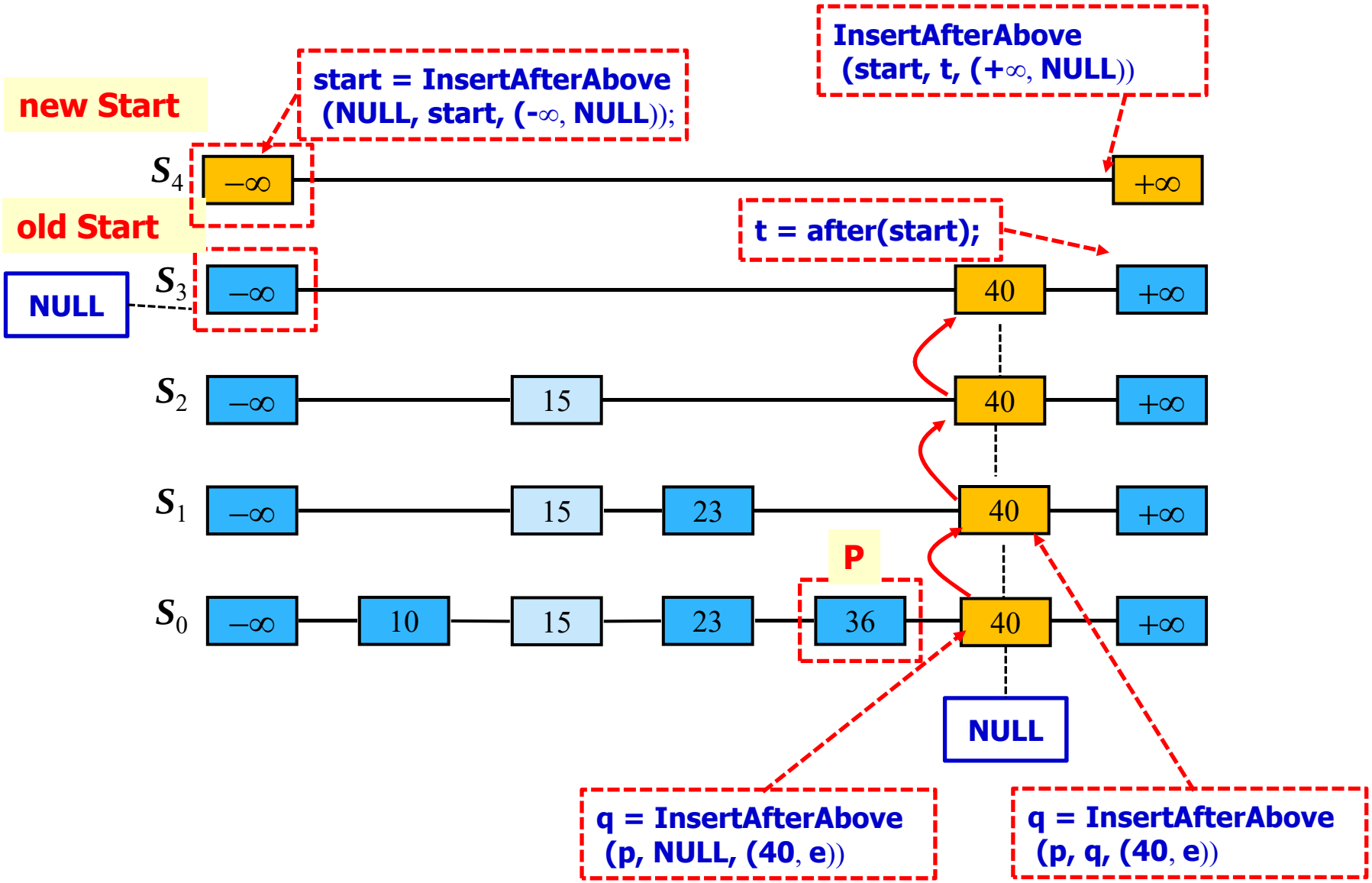


Step1: SkipSearch (40)



Step2: Insert (40) after P_0 with $i = 3$

◆ InsertAfterAbove(p, q, e)



◆ SkipInsert

```
Algorithm SkipInsert(k, v)
    Input: key k and value v
    Output: topmost position of the entry inserted in the skip list

     p = SkipSearch(k)
     q = NULL
     e = (k, v)
     i = -1
     repeat {
        i = i + 1
        if ( i ≥ h) then
        {
            h = h + 1
            t = after(start)
            start = insertAfterAbove(null, start, (-∞, NULL)) // level–h left node
            insertAfterAbove(start, t, (+∞, NULL)) // level-h right node
        }
        q = insertAfterAbove(p, q, e)  // after p, above q, insert e
        while (above(p) == NULL) do
              p = before(p)  // scan backward
         p = above(p)
     } until (coinFlip() == tails)
      n = n + 1
      return q
```

Randomized Algorithms

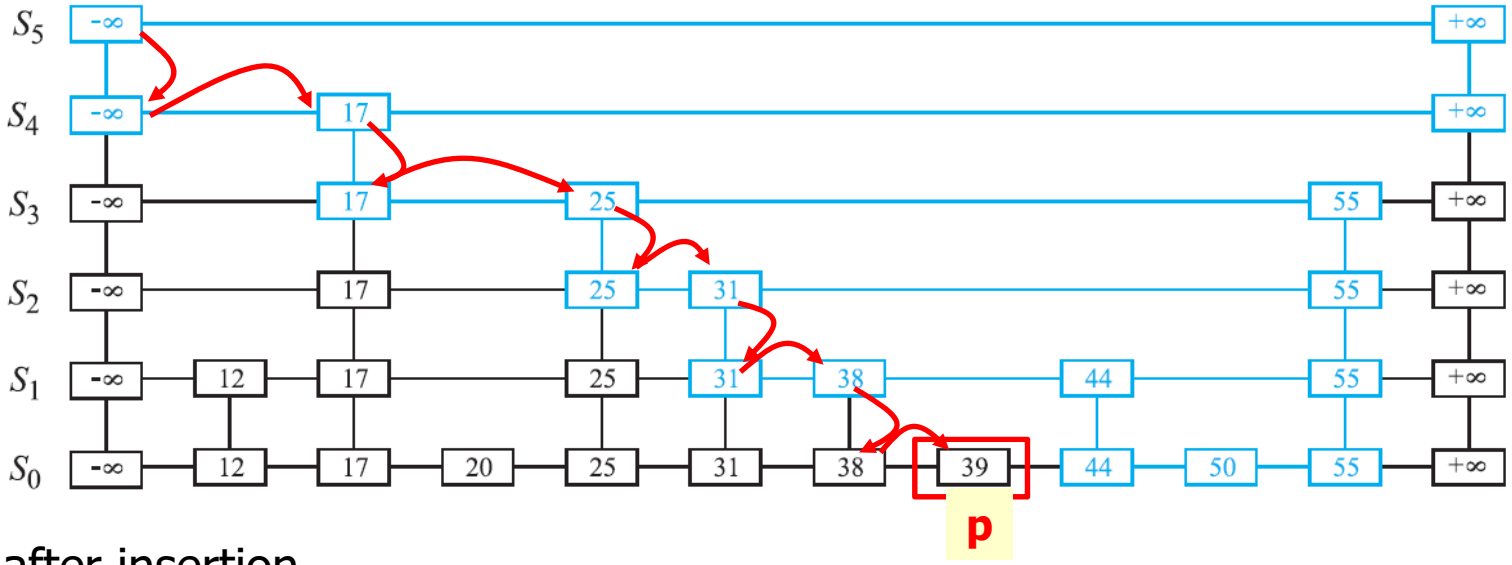
- A randomized algorithm performs coin tosses (i.e., uses random bits) to control its execution
- It contains statements of the type

```
b = random()
if b == 0
    do A …
else // b == 1
    do B …
```

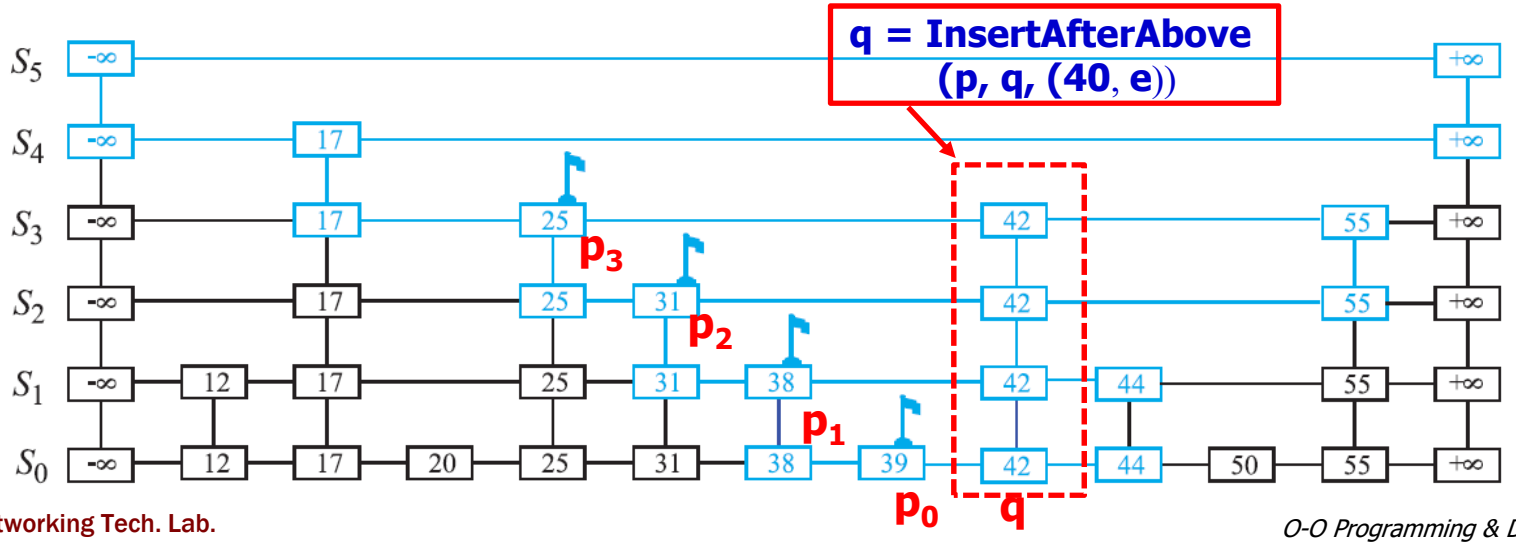
- Its running time depends on the outcomes of the coin tosses
- We analyze the expected running time of a randomized algorithm under the following assumptions
  - the coins are unbiased, and
  - the coin tosses are independent
- The worst-case running time of a randomized algorithm is often large but has very low probability (e.g., it occurs when all the coin tosses give "heads")
- We use a randomized algorithm to insert items into a skip list

◆ insertion of an entry (key 42)

- before insertion

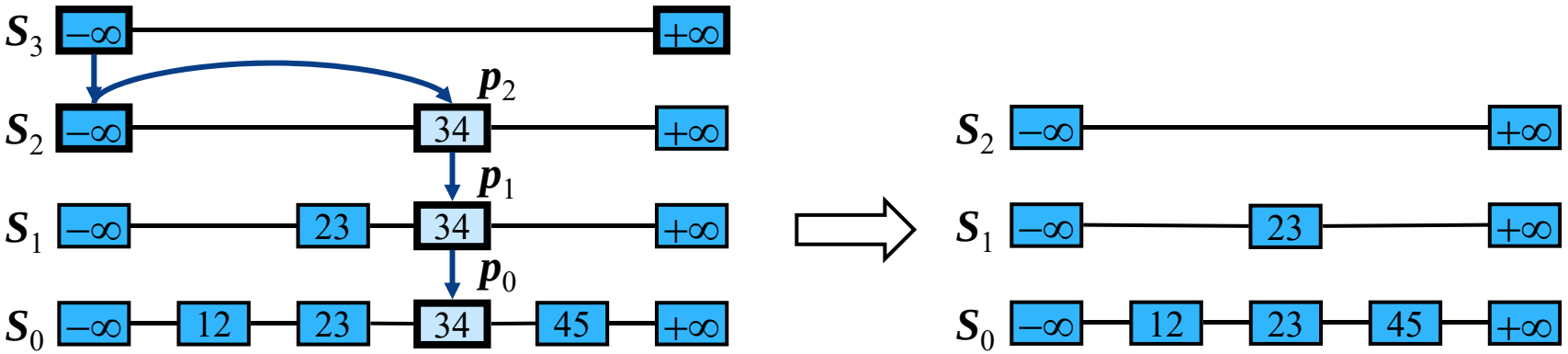


- after insertion

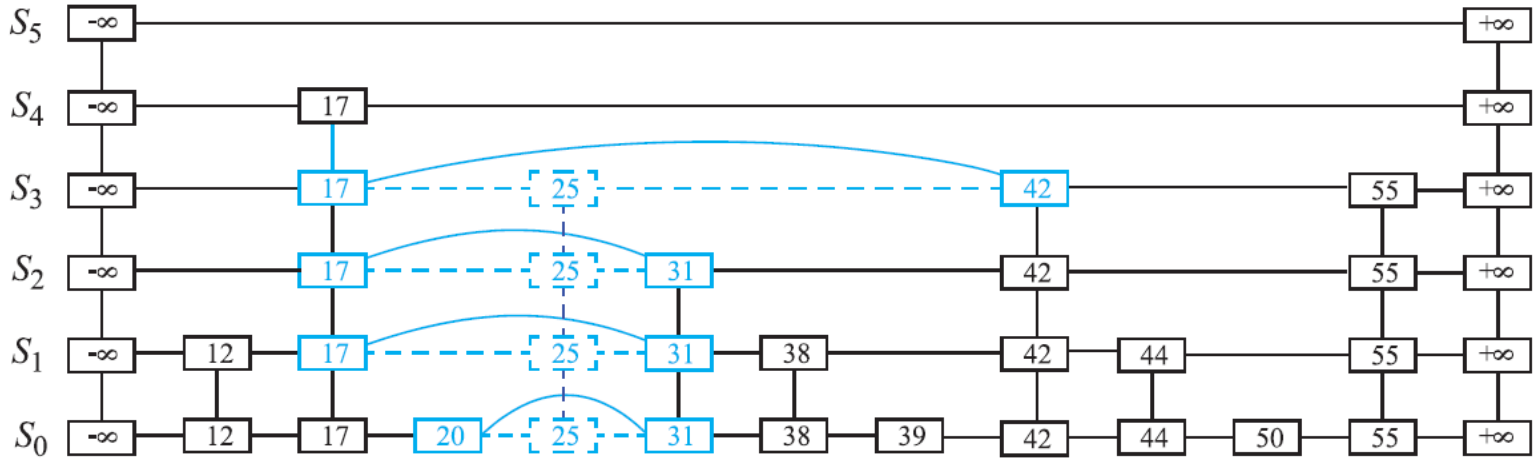


Deletion

- **To remove an entry with key x from a skip list, we proceed as follows:**
  - We search for x in the skip list and find the positions $p_0, p_1, \ldots, p_i$ of the items with key x, where position p_j is in list S_j
  - We remove positions $p_0, p_1, \ldots, p_i$ from the lists $S_0, S_1, \ldots, S_i$
  - We remove all but one list containing only the two special keys
- **Example: remove key 34**



◆ Removal in a Skip List (key = 25)

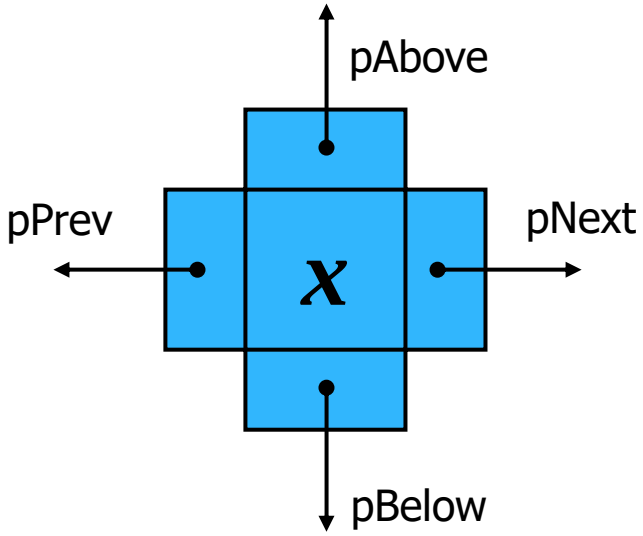


Skip List 구현

Implementation

- We can implement a skip list with quad-nodes
- A quad-node stores:
  - entry
  - link to the node prev
  - link to the node next
  - link to the node below
  - link to the node above
- Also, we define special keys PLUS_INF and MINUS_INF, and we modify the key comparator to handle them

- quad-node



```
/** Planet.h */

#include <iostream>
#include <string>
using namespace std;

class Planet {
    friend ostream& operator<<(ostream& fout, Planet& p)
    {
        fout << "Planet ( ID: " << p.getID() << ")";
        fout << ", name: " << p.getName();
        fout << ", relative mass: " << p.getRelativeMass();
        fout << ", distance: " << p.getDistance() << ")";
        return fout;
    }
public:
    Planet(int id, string n, double rM, double dist)
        :ID(id), name(n), relativeMass(rM), distance(dist)
    {} // constructor
    Planet() {} // default constructor
    int getID() {return ID;}
    string getName() { return name;}
    double getRelativeMass() { return relativeMass;}
    double getDistance() { return distance; }

private:
    int ID;
    string name;
    double relativeMass;
    double distance;
};
```

```
/** Generic Skip List.h (1) */
#include <iostream>
#include <stdlib.h>
#include <iomanip>
#include <time.h>
#include <limits>
#include <string>
#define PROBABILITY_OF_ABOVE_LEVEL 2
#define PLUS_INF INT_MAX
#define MINUS_INF INT_MIN

using namespace std;

template<typename Key, typename Element>
class SkipList {
protected:
    class QuadNode
    {
    private:
        Key _key;
        Element _elem;
    protected:
        QuadNode *up;
        QuadNode *down;
        QuadNode *prev;
        QuadNode *next;
        int level;
```

```
/** Generic Skip List.h (2) */
public:
        QuadNode(Key k=Key(),
            Element e = Element()) // constructor
             :_key(k), _elem(e)
        {
             up = down = prev = next = NULL;
             level = -1;
        }
        Key& key()  {return _key;}
        Element& element() {return _elem;}
        void setKey(Key& k) {_key = k;}
        void setElement(Element& e) {_elem = e;}
        friend class Position;
    }; // end of QuadNode
```

```
/** Generic Skip List.h (3) */
public:
    class Position{
    protected:
        QuadNode* pQN;
    public:
        Position(QuadNode* pSLN) {pQN = pSLN;}
        Position() {}
        Key& key() {return pQN->key();}
        Element& element() {return pQN->element();}
        Position below() { return Position(pQN->down);}
        Position above() { return Position(pQN->up);}
        Position after() { return Position(pQN->next);}
        Position before() { return Position(pQN->prev);}
        void setBelow(Position& p) { pQN->down = p.pQN;}
        void setAbove(Position& p) { pQN->up = p.pQN;}
        void setAfter(Position& p) { pQN->next = p.pQN;}
        void setBefore(Position& p)
          { pQN->prev = p.pQN;}
        bool operator==(Position& q)
          {return pQN == q.pQN;}
        bool operator!=(Position& q)
          {return pQN != q.pQN;}
        Element& operator*() { return pQN->element();}

        friend class SkipList;
    }; // end of class Position
```

```
/** Generic Skip List.h (4) */
private:
    Position start; // skip list top
    Position end; // skip list bottom
    int height;
    int num_elements;
public:
    SkipList()
    {
        height = 0;
        QuadNode* pQNode =
            new QuadNode(Key(MINUS_INF),
                Element());
        start = Position(pQNode);
        pQNode =
            new QuadNode(Key(PLUS_INF),
                Element());
        end = Position(pQNode);
        start.setAfter(end);
        start.setBefore(Position(NULL));
        start.setAbove(Position(NULL));
        start.setBelow(Position(NULL));
        end.setBefore(start);
        end.setAfter(Position(NULL));
        end.setBelow(Position(NULL));
        end.setAbove(Position(NULL));
        //bottom = Position(NULL);
        srand(time(NULL));
    }
```

```
/** Generic Skip List.h (5) */
    Position SkipListSearch(Key& k)
    {
        Position pos = start;
        while (pos.below() != Position(NULL))
        {
            pos = pos.below();

            while (k >= ((pos.after()).key()))
            {
                pos = pos.after();
                if (pos.after() == (Position)NULL)
                    break;
            } // inner while
        }  // outer while
        return pos;
    }

    Position SkipListInsert(ostream& fout,
        Key k, const Element e)
    {
        Position p, q, t;

        p = SkipListSearch(k);
        q = Position(NULL);
        int i = -1;
```

```
/** Generic Skip List.h (5) */

        do {
            i = i+1;
            if (i >= height)
            {
                height = height + 1;
                t = start.after();
                start = insertAfterAbove(fout,
                    Position(NULL), start, MINUS_INF,
                    Element());
                insertAfterAbove(fout, start,
                    t, PLUS_INF, Element());
            }
            q = insertAfterAbove(fout, p, q, k, e);
            while (p.above() == (Position)NULL)
            {
                p = p.before(); // scan backward
            }
            p = p.above(); // jump up to higher level
            //q = insertAfterAbove(p, q, k, e);
        } while (rand() % 2 == 0);
          // flip coin and continue if head occurs
        ++num_elements;
        return q;

    }
```

```
/** Generic Skip List.h (5) */

    void SkipListNodeDelete(Position p)
    {
        Position pprev, pnext;

        if (p != Position(NULL))
        {
            pprev = p.before();
            pnext = p.after();
            if (pprev != Position(NULL))
                pprev.setAfter(pnext);
            if (pnext != Position(NULL))
                pnext.setBefore(pprev);
            delete p.pQN;
        }
    }
```

```
/** Generic Skip List.h (5) */

    void SkipListRemove(ostream& fout, Key k)
    {
        Position p, pprev, pnext, p00, px0;
        Position p0y, pxy, old_p;
        int h_max, h;

        p = SkipListSearch(k);
        if ((p.key() != k) || (p.key() == MINUS_INF))
        {
            fout << "Key (" << k << ") is not found";
            fout << "in SkipList !!" << endl;
            return;
        }
```

```
/** Generic Skip List.h (6) */

        while (p != Position(NULL))
        {
            old_p = p;
            p = p.above();
            SkipListNodeDelete(old_p);
        }

        fout << "deleted skip list node (Key: ";
        fout  << k << ")" << endl;
        // compare the height of the skip list
        // and the highest tower level
        p00 = start;
        for (int i=height; i>0; i--)
        {
            p00 = p00.below();
        }

        px0 = p00.after();
        pxy = start;
        h_max = 0;
```

```
/** Generic Skip List.h (7) */

        while (px0.key() != PLUS_INF)
        {
            pxy = px0;
            h = 0;
            while(pxy != Position(NULL))
            {
                ++h;
                pxy = pxy.above();
            }
            if (h_max < h)
                h_max = h;
            px0 = px0.after();
        }
        if (h_max < height)  {
            fout << "Current height is less than";
            fout << " the new h_max (" << h_max;
            fout << ") ==> need adjustment !!";
            fout << endl;
        }
        for (int i=0; i<(height - h_max); i++)
        {
            p0y = start;
            pxy = start.after();
            start = start.below();
            SkipListNodeDelete(pxy);
            SkipListNodeDelete(p0y);
        }
        height = h_max;
    }
```

/** Generic Skip List.h (8) */

```
void PrintSkipList(ostream& fout)
{
    Position p, p00, px0, p0y, pxy, q, q0, qx;
    int level;

    fout << "Print Skip List Inside :"
    fout << current height (" << height << ")";
    fout << endl;
    p00 = start;
    level = height;
    for (int i=height; i>0; i--)
    {
        p00 = p00.below();
    }

    p0y = start;
    pxy = start;
    for (level = height; level >= 0; level--)
    {
        fout << "level" << setw(2);
        fout << level << "  ";
        if ( pxy.key() == MINUS_INF)
            fout << "-oo";
        else
            fout << setw(5) << pxy.key();
```

/** Generic Skip List.h (9) */

```
        px0 = p00.after();
        for (pxy = p0y.after();; pxy = pxy.after())
        {
            while((px0.key() != pxy.key()))
            {
                fout << " ------";
                px0 = px0.after();
            }
            if ( pxy.key() == PLUS_INF) {
                break;
            } else {
                fout << " -" << setw(5);
                fout << pxy.key();
            }
            px0 = px0.after();
        }
        if ( pxy.key() == PLUS_INF)
            fout << "₩ +oo";
        else
            fout << setw(5) << pxy.key();

        fout << endl;
        p0y = p0y.below();
        pxy = p0y;

    }

    fout << endl;

}
```

```
/** Generic Skip List.h (9) */

    Position insertAfterAbove (ostream& fout,
          Position p, Position q, const Key k,
          const Element e)
    {
        QuadNode* pNewNode =
            new QuadNode(k, e);
        Position n(pNewNode);
        n.setAbove(Position(NULL));
        n.setBelow(Position(NULL));
        n.setBefore(Position(NULL));
        n.setAfter(Position(NULL));

        if (p == Position(NULL))
        {
            if (q == Position(NULL))
            {
                fout << "Trying to insert at";
                fout << (NULL, NULL) position ";
                fout << endl;
                return Position(NULL);
            } else      {
                q.setAbove(n);
                n.setBelow(q);
                n.setAbove(Position(NULL));
                n.setAfter(Position(NULL));

                return n;
            }
        } else
```

```
/** Generic Skip List.h (9) */

        {
            if (q == Position(NULL)) {
                Position nx, ny;
                fout << "insert a node (" << n.key();
                fout << ") at level 0 " << endl;
                nx = p.after();
                p.setAfter(n);
                n.setBefore(p);
                n.setAfter(nx);
                nx.setBefore(n);
                n.setBelow(Position(NULL));
                n.setAbove(Position(NULL));
                return n;
            } else {
                Position nx, ny;
                nx = p.after();
                n.setAfter(nx);
                if (nx != Position(NULL))
                    nx.setBefore(n);
                p.setAfter(n);
                n.setBefore(p);
                ny = q.above();
                q.setAbove(n);
                n.setBelow(q);
                n.setAbove(ny);
                if (ny != Position(NULL))
                    ny.setBelow(n);
            }
        }
        return n;
    }
};
#endif
```

Skip List 응용 프로그램

```
/** main.cpp (1) */
#include <iostream>
#include <fstream>
#include "GenericSkipList.h"
#include "Planet.h"
#include <string>

using namespace std;
#define NUM_PLANETS 9

void main()
{
    ofstream fout;
    Planet *pSolarPlanets;
    int *planetIDs;
    pSolarPlanets = new Planet[NUM_PLANETS];
    Planet suffledSolarPlanets[] = {
        Planet(7, "Uranus", 14.5, 2870),
        Planet(5, "Jupiter", 318, 778),
        Planet(3, "Earth", 1.0, 150),
        Planet(4, "Mars", 0.107, 228),
        Planet(6, "Saturn", 95.1, 1430),
        Planet(2, "Venus", 0.815, 108),
        Planet(8, "Neptune", 17.2, 4500),
        Planet(1, "Mercury", 0.0558, 57.9),
        Planet(9, "Pluto", 0.11, 5900)
    };
```

```
/** main.cpp (2) */
    fout.open("output.txt");
    if (fout.fail())
    {    cout << "output.txt file opening failed !!\n";
         exit(1);
    }
    //testTaskSkipList();
    fout << "\n==================" << endl;
    fout << "Testing SkipList<int, Planet> ..." << endl;
    SkipList<int, Planet> planetSkipList;
    typedef SkipList<int, Planet>::Position Pos_planet;
    Pos_planet pPlanet;
    int id;
    int deleteList[NUM_PLANETS] = {2, 7, 6, 4, 8, 5, 3, 9, 1};
    fout <<" Inserting planets to";
    fout << planet SkipList ..." << endl;
    for (int i=0; i<NUM_PLANETS; i++)
    {
         fout << "Inserting SolarPlanet[";
         fout << i << "]: " << suffledSolarPlanets[i] << endl;
         id = suffledSolarPlanets[i].getID();
         planetSkipList.SkipListInsert(fout, id,  suffledSolarPlanets[i]);
         planetSkipList.PrintSkipList(fout);
         fout << endl;
    }
    fout << "\n********************* " << endl;
```

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

    for (int i=0; i<NUM_PLANETS; i++)
    {
        fout <<"Search planetSkipList and delete";
        fout << " the planet (";
        fout << deleteList[i] << ") : ";
        pPlanet = planetSkipList.SkipListSearch(deleteList[i]);
        if (pPlanet != Pos_planet(NULL))
        {
            fout << *pPlanet << endl;
            planetSkipList.SkipListRemove(fout, deleteList[i]);
        }
        planetSkipList.PrintSkipList(fout);
        fout << endl;
    }
}
```

```
=================
Testing SkipList<int, Planet> ...
 Inserting planets toplanet SkipList ...
Inserting SolarPlanet[0]: Planet ( ID: 7), name: Uranus, relative mass: 14.5, distance: 2870)
insert a node (7) at level 0
Print Skip List Inside : current height(1)
level 1  -oo ------ +oo
level 0  -oo -    7 +oo


Inserting SolarPlanet[1]: Planet ( ID: 5), name: Jupiter, relative mass: 318, distance: 778)
insert a node (5) at level 0
Print Skip List Inside : current height(2)
level 2  -oo ------ ------ +oo
level 1  -oo -    5 ------ +oo
level 0  -oo -    5 -    7 +oo


Inserting SolarPlanet[2]: Planet ( ID: 3), name: Earth, relative mass: 1, distance: 150)
insert a node (3) at level 0
Print Skip List Inside : current height(4)
level 4  -oo ------ ------ ------ +oo
level 3  -oo -    3 ------ ------ +oo
level 2  -oo -    3 ------ ------ +oo
level 1  -oo -    3 -    5 ------ +oo
level 0  -oo -    3 -    5 -    7 +oo


Inserting SolarPlanet[3]: Planet ( ID: 4), name: Mars, relative mass: 0.107, distance: 228)
insert a node (4) at level 0
Print Skip List Inside : current height(4)
level 4  -oo ------ ------ ------ ------ +oo
level 3  -oo -    3 ------ ------ ------ +oo
level 2  -oo -    3 ------ ------ ------ +oo
level 1  -oo -    3 ------ -    5 ------ +oo
level 0  -oo -    3 -    4 -    5 -    7 +oo


Inserting SolarPlanet[4]: Planet ( ID: 6), name: Saturn, relative mass: 95.1, distance: 1430)
insert a node (6) at level 0
Print Skip List Inside : current height(5)
level 5  -oo ------ ------ ------ ------ ------ +oo
level 4  -oo ------ ------ ------ -    6 ------ +oo
level 3  -oo -    3 ------ ------ -    6 ------ +oo
level 2  -oo -    3 ------ ------ -    6 ------ +oo
level 1  -oo -    3 ------ -    5 -    6 ------ +oo
level 0  -oo -    3 -    4 -    5 -    6 -    7 +oo


Inserting SolarPlanet[5]: Planet ( ID: 2), name: Venus, relative mass: 0.815, distance: 108)
insert a node (2) at level 0
Print Skip List Inside : current height(5)
level 5  -oo ------ ------ ------ ------ ------ ------ +oo
level 4  -oo ------ ------ ------ ------ -    6 ------ +oo
level 3  -oo ------ -    3 ------ ------ -    6 ------ +oo
level 2  -oo ------ -    3 ------ ------ -    6 ------ +oo
level 1  -oo ------ -    3 ------ -    5 -    6 ------ +oo
level 0  -oo -    2 -    3 -    4 -    5 -    6 -    7 +oo


Inserting SolarPlanet[6]: Planet ( ID: 8), name: Neptune, relative mass: 17.2, distance: 4500)
insert a node (8) at level 0
Print Skip List Inside : current height(5)
level 5  -oo ------ ------ ------ ------ ------ ------ ------ +oo
level 4  -oo ------ ------ ------ ------ -    6 ------ ------ +oo
level 3  -oo ------ -    3 ------ ------ -    6 ------ ------ +oo
level 2  -oo ------ -    3 ------ ------ -    6 ------ ------ +oo
level 1  -oo ------ -    3 ------ -    5 -    6 ------ ------ +oo
level 0  -oo -    2 -    3 -    4 -    5 -    6 -    7 -    8 +oo


Inserting SolarPlanet[7]: Planet ( ID: 1), name: Mercury, relative mass: 0.0558, distance: 57.9)
insert a node (1) at level 0
Print Skip List Inside : current height(5)
level 5  -oo ------ ------ ------ ------ ------ ------ ------ ------ +oo
level 4  -oo ------ ------ ------ ------ ------ -    6 ------ ------ +oo
level 3  -oo ------ ------ ------ -    3 ------ -    6 ------ ------ +oo
level 2  -oo -    1 ------ -    3 ------ ------ -    6 ------ ------ +oo
level 1  -oo -    1 ------ -    3 ------ -    5 -    6 ------ ------ +oo
level 0  -oo -    1 -    2 -    3 -    4 -    5 -    6 -    7 -    8 +oo


Inserting SolarPlanet[8]: Planet ( ID: 9), name: Pluto, relative mass: 0.11, distance: 5900)
insert a node (9) at level 0
Print Skip List Inside : current height(5)
level 5  -oo ------ ------ ------ ------ ------ ------ ------ ------ ------ +oo
level 4  -oo ------ ------ ------ ------ ------ -    6 ------ ------ ------ +oo
level 3  -oo ------ ------ ------ -    3 ------ -    6 ------ ------ ------ +oo
level 2  -oo -    1 ------ -    3 ------ ------ -    6 ------ ------ ------ +oo
level 1  -oo -    1 ------ -    3 ------ -    5 -    6 ------ ------ ------ +oo
level 0  -oo -    1 -    2 -    3 -    4 -    5 -    6 -    7 -    8 -    9 +oo
```

```
**********
Search planetSkipList and delete the planet(2) : Planet ( ID: 2), name: Venus, relative mass: 0.815, distance: 108)
deleted skip list node (Key: 2)
Print Skip List Inside : current height(5)
level 5  -oo ------ ------ ------ ------ ------ ------ ------ ------ +oo
level 4  -oo ------ ------ ------ ------ -    6 ------ ------ ------ +oo
level 3  -oo ------ -    3 ------ ------ -    6 ------ ------ ------ +oo
level 2  -oo -    1 -    3 ------ ------ -    6 ------ ------ ------ +oo
level 1  -oo -    1 -    3 ------ -    5 -    6 ------ ------ ------ +oo
level 0  -oo -    1 -    3 -    4 -    5 -    6 -    7 -    8 -    9 +oo


Search planetSkipList and delete the planet(7) : Planet ( ID: 7), name: Uranus, relative mass: 14.5, distance: 2870)
deleted skip list node (Key: 7)
Print Skip List Inside : current height(5)
level 5  -oo ------ ------ ------ ------ ------ ------ ------ +oo
level 4  -oo ------ ------ ------ ------ -    6 ------ ------ +oo
level 3  -oo ------ -    3 ------ ------ -    6 ------ ------ +oo
level 2  -oo -    1 -    3 ------ ------ -    6 ------ ------ +oo
level 1  -oo -    1 -    3 ------ -    5 -    6 ------ ------ +oo
level 0  -oo -    1 -    3 -    4 -    5 -    6 -    8 -    9 +oo


Search planetSkipList and delete the planet(6) : Planet ( ID: 6), name: Saturn
deleted skip list node (Key: 6)
Current height is less than the new h_max (4) ==> need adjustment !!
Print Skip List Inside : current height(4)
level 4  -oo ------ ------ ------ ------ ------ ------ +oo
level 3  -oo ------ -    3 ------ ------ ------ ------ +oo
level 2  -oo -    1 -    3 ------ ------ ------ ------ +oo
level 1  -oo -    1 -    3 ------ -    5 ------ ------ +oo
level 0  -oo -    1 -    3 -    4 -    5 -    8 -    9 +oo


Search planetSkipList and delete the planet(4) : Planet ( ID: 4), name: Mars,
deleted skip list node (Key: 4)
Print Skip List Inside : current height(4)
level 4  -oo ------ ------ ------ ------ ------ ------ +oo
level 3  -oo ------ -    3 ------ ------ ------ ------ +oo
level 2  -oo -    1 -    3 ------ ------ ------ ------ +oo
level 1  -oo -    1 -    3 -    5 ------ ------ ------ +oo
level 0  -oo -    1 -    3 -    5 -    8 -    9 +oo


Search planetSkipList and delete the planet(8) : Planet ( ID: 8), name: Neptune, relative mass: 17.2, distance: 4500)
deleted skip list node (Key: 8)
Print Skip List Inside : current height(4)
level 4  -oo ------ ------ ------ ------ +oo
level 3  -oo ------ -    3 ------ ------ +oo
level 2  -oo -    1 -    3 ------ ------ +oo
level 1  -oo -    1 -    3 -    5 ------ +oo
level 0  -oo -    1 -    3 -    5 -    9 +oo


Search planetSkipList and delete the planet(5) : Planet ( ID: 5), name: Jupiter, relative mass: 318, distance: 778)
deleted skip list node (Key: 5)
Print Skip List Inside : current height(4)
level 4  -oo ------ ------ ------ ------ +oo
level 3  -oo ------ -    3 ------ ------ +oo
level 2  -oo -    1 -    3 ------ ------ +oo
level 1  -oo -    1 -    3 ------ ------ +oo
level 0  -oo -    1 -    3 -    9 +oo


Search planetSkipList and delete the planet(3) : Planet ( ID: 3), name: Earth, relative mass: 1, distance: 150)
deleted skip list node (Key: 3)
Current height is less than the new h_max (3) ==> need adjustment !!
Print Skip List Inside : current height(3)
level 3  -oo ------ ------ +oo
level 2  -oo -    1 ------ +oo
level 1  -oo -    1 ------ +oo
level 0  -oo -    1 -    9 +oo


Search planetSkipList and delete the planet(9) : Planet ( ID: 9), name: Pluto, relative mass: 0.11, distance: 5900)
deleted skip list node (Key: 9)
Print Skip List Inside : current height(3)
level 3  -oo ------ +oo
level 2  -oo -    1 +oo
level 1  -oo -    1 +oo
level 0  -oo -    1 +oo


Search planetSkipList and delete the planet(1) : Planet ( ID: 1), name: Mercury, relative mass: 0.0558, distance: 57.9)
deleted skip list node (Key: 1)
Current height is less than the new h_max (0) ==> need adjustment !!
Print Skip List Inside : current height(0)
level 0  -oo +oo
```

Homework 11

Homework 11

11.1 Cyclic Shift Hash Code

- Cyclic Shift Hash Code 함수를 사용하여 주어진 영어 key word 들에 대한 32-bit hash code를 생성한 후, 모듈로 (modulo) 연산의 압축함수 (compression function)을 사용하여 6비트 크기의 hash value를 계산하라. 계산 결과 충돌이 발생하는 경우가 있는지 확인하고, 그 이유에 대하여 설명하라.
- hash code 및 hash value 계산 대상 key words는 TOEIC essential vocabulary 중에서 단어 길이가 5 ~ 10 character인 단어를 20개 이상 사용할 것.

11.2 myVocaDictionary

- TOEIC 핵심 어휘 공부를 위한 myVocaDictionary를 class HashDict 를 기반으로 구현하라.
- 구현된 myVocaDictionary에는 TOEIC 핵심 어휘를 **100** 개 이상 포함하도록 하고, 각 핵심 어휘의 품사별 유의어 (thesaurus)와 응용 예문을 포함하도록 하라.

11.3 Skip List

- Skip list를 구현하고, 위 11.1에서 사용하였던 TOEIC 핵심 어휘 20개를 차례로 삽입하면서 항상 정렬 상태가 유지되는지 확인하고, Skip list에 삽입하였던 순서대로 단어를 찾고, 이를 삭제를 하면서 남아있는 단어들의 정렬 상태가 유지되는가를 확인하라.