O-O Programming & Data Structure Lab. 11

11. Thesaurus Dictionary 구현을 위한 class HashDict 설계 및 구현

11.1 class MyVoca

```
/** MyVoca.h */

#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)
        { . . . . .  }
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
```

11.2 MyVocaList.h

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

int   NUM_MY_TOEIC_VOCA = 100;
MyVoca myToeicVocaList[]; // defined in MyVocaList.cpp

#endif
```

11.3 class Entry

```
template<typename K, typename V>
class Entry
{
        friend ostream& operator<<(ostream& fout, Entry<K, V>& entry)
        {   . . . . .       }
public:
        Entry(K key, V value) { _key = key; _value = value; }
        Entry() { } // default constructor
        ~Entry() {}
        void setKey(const K& key) { _key = key; }
        void setValue(const V& value) { _value = value; }
        K getKey() const { return _key; }
        V getValue() const { return _value; }
        bool operator>(const Entry<K, V>& right) const { return (_key > right.getKey()); }
        bool operator>=(const Entry<K, V>& right) const { return (_key >= right.getKey()); }
        bool operator<(const Entry<K, V>& right) const { return (_key < right.getKey()); }
        bool operator<=(const Entry<K, V>& right) const { return (_key <= right.getKey()); }
```

```
        bool operator==(const Entry<K, V>& right) const { return ((_key == right.getKey()) && (_value ==
right.getValue())); }
        Entry<K, V>& operator=(Entry<K, V>& right);
        void fprint(ostream fout);
private:
        K _key;
        V _value;
};
```

11.4 class CyclicShiftHashCode

```
/* CyclicShiftHashCode */
#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;
        }
 };
```

11.5 class HashMap

```
/** HashMap.h */
#ifndef HASHMAP_H
#define HASHMAP_H

#include <list>
#include <vector>
#include "Entry.h"
#include "Exceptions.h"

template <typename K, typename >
class HashMap {
public:                                 // public types
        typedef Entry<const K, V> Entry;    // a (key,value) pair
        class Iterator;
public:                                 // public functions
        HashMap(int capacity = 101);        // constructor
        int size() const;                   // number of entries
        bool empty() const;                 // is the map empty?
        Iterator find(const K& k);          // find entry with key k
        Iterator 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
        Iterator end();                     // iterator to end entry
protected:                              // protected types
        typedef std::list<Entry> Bucket;    // a bucket of entries
        typedef std::vector<Bucket> BktArray;       // a bucket array

        // 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
        typedef typename BktArray::iterator BItor;      // bucket iterator
        typedef typename Bucket::iterator EItor;        // entry iterator
        static void _next(Iterator& p)                  // bucket's next entry
        {
```

```
                ++p.ent;
        }
        static bool _endOfBkt(const Iterator& p)        // end of bucket?
        {
                return p.ent == p.bkt->end();
        }

private:
        int n;                                  // number of entries
        BktArray B;                             // bucket array

public:                                 // public types
        // Iterator class declaration
        class Iterator {                                // an iterator (& position)
        protected:
                EItor ent;                              // which entry
                BItor bkt;                              // which bucket
                const BktArray* ba;                     // which bucket array
        public:
                Iterator() {} // default constructor
                Iterator(const BktArray& a, const BItor& b, const EItor& q = EItor())
                        : ent(q), bkt(b), ba(&a) { }
                Entry& operator*();             // get entry
                V getValue() { Entry& e = *ent; return e.value(); }
                bool operator==(const Iterator& p) const;     // are iterators equal?
                bool operator!=(const Iterator& p) const;     // are iterators different ?
                Iterator& operator++();         // advance to next entry
                Iterator& advanceEItor()
                {
                        ++ent;
                        return *this;
                }
                friend class HashMap;           // give HashMap access
        };
};
#endif
```

11.6 class HashDict

```
template <typename K, typename V>
class HashDict : public HashMap<K, VH> {
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
        };
public:                                         // public functions
        HashDict(int capacity = DEFAULT_HASH_SIZE);                     // constructor
        Range findAll(const K& k);                              // find all entries with k
        Iterator insert(const K& k, const V& v);                // insert pair (k,v)
};
```

11.7 main() function

```
/** main.cpp */

#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;
	List_Str usages;
	int word_count;
	MyVoca mv;
	string keyWord;
	HashDict<string, MyVoca*, CyclicShiftHashCode> myVocaDict;
	HashDict<string, MyVoca*, CyclicShiftHashCode>::Iterator itr;
	HashDict<string, MyVoca*, CyclicShiftHashCode>::Range range;
	Entry<string, MyVoca*> vocaEntry;

	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);
	}

	//cout << endl;
	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;

	//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();
}
```

11.8 Example output

```
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, accommodation, )
  - 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 )
imperative(n):
  - thesaurus(necessity, essential, requirement, )
  - example usage( )
imperative(adj):
  - thesaurus(authoritative, vital, )
  - example usage( )
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( )

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>

(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					

(2) Hash Map 을 STL vector 와 STL list 로 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

(3) Hash Map 에서 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

(4) HashDict 에서 사용하는 class Range 의 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