**Homework 11**

|  |
| --- |
| 그림입니다. 원본 그림의 이름: YU_UI_RGB-10.png 원본 그림의 크기: 가로 2256pixel, 세로 3047pixel 프로그램 이름 : Adobe ImageReady |

|  |  |
| --- | --- |
| 과목명 | 객체지향프로그래밍과자료구조 |
| 교수님 | 김영탁 교수님 |
| 이 름 | 김주환 |
| 학 번 | 21812158 |
| 일 자 | 2021.11.24.수 |

**11.1 ~ 2 Cyclic Shift Hash Code & myVocaDictionary**

|  |
| --- |
| /\* main.cpp \*/  /\* Description  \* Hash Dictionary 설계 및 구현  \* Programmed by J. H. Kim  \* Last updated : 2021-11-24 \*/  #include <iostream>  #include <fstream>  #include <string>  #include "HashMap.h"  #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\*> myVocaDict;  HashDict<string, MyVoca\*>::Iterator itr;  HashDict<string, MyVoca\*>::Range range;  Entry<string, MyVoca\*> vocaEntry;  CyclicShiftHashCode hash;  unsigned int h = 0;  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.operator\*().getValue();  h = hash(pVoca->getKeyWord());  fout << "[ Key Word : " << setw(15) << pVoca->getKeyWord();  fout << ", Hash Code : " << setw(10) << h;  fout << ", Hash Value : " << setw(5) << h % DEFAULT\_HASH\_TABLE\_SIZE << "]" << endl;  }  fout << endl;  fout << "Status of each Bucket" << endl;  myVocaDict.fprintBucketSizes(fout, myVocaDict.begin());  fout << 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.operator\*().getValue();  fout << \*pVoca << endl;  }  fout << endl;  fout << endl;  fout.close();  } |
| /\* HashDictionary.h \*/  #ifndef HASH\_DICTIONARY\_H  #define HASH\_DICTIONARY\_H  #include "HashMap.h"  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  public: // public functions  HashDict(int capacity = DEFAULT\_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 = this->\_find(k); // find key  Iterator q = this->\_insert(p, Entry(k, v)); // insert it here  return q; // return its position  }  template <typename K, typename V> // find all entries with k  typename HashDict<K, V>::Range  HashDict<K, V>::findAll(const K& k)  {  Iterator b = this->\_find(k); // look up k  Iterator p = b;  while (p != this->end() && (\*p).getKey() == k)  { // find next entry with different key or end of bucket array  ++p;  }  return Range(b, p); // return range of positions  }  #endif |
| /\* HashMap.h \*/  #ifndef HASHMAP\_H  #define HASHMAP\_H  #include <list>  #include <vector>  #include "Entry.h"  #include "Exceptions.h"  #include "CyclicShiftHashCode.h"  #include "MyVoca.h"  #define DEFAULT\_HASH\_TABLE\_SIZE 64  template <typename K, typename V> // key, value, hash  class HashMap {  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;  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, Iterator itr); // 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();  }  private:  int num\_entry; // number of entries  BktArray B; // bucket array (Hash Table)  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  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());  }  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  }  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  }  template <typename K, typename V> // find utility  typename HashMap<K, V>::Iterator HashMap<K, V>::\_find(const K& k)  {  CyclicShiftHashCode hash;  unsigned 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).getKey() != 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  }  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  }  }  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  }  template <typename K, typename V>  void HashMap<K, V>::fprintBucket(ostream& fout, BItor bkt)  {  CyclicShiftHashCode hash;  Iterator p(this->B, bkt, bkt->begin());  MyVoca\* pVoca;  unsigned int h = 0;  while (p.ent != bkt->end())  {  pVoca = p.operator\*().getValue();  h = hash(pVoca->getKeyWord());  fout << "["<< setw(15) << pVoca->getKeyWord();  fout << ", " << setw(5) << h % DEFAULT\_HASH\_TABLE\_SIZE << "], ";  ++p.ent;  }  }  template <typename K, typename V>  void HashMap<K, V>::fprintBucketSizes(ostream& fout, Iterator itr)  {  BItor ptor = itr.bkt;  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;  num\_bkts = B.size();  for (int bkt = 0; bkt < num\_bkts; bkt++)  {  bkt\_size = B[bkt].size();  fout << "Bucket[" << setw(3) << bkt << "] : ";  fprintBucket(fout, ptor++);  if (bkt\_size > max\_ent)  {  max\_ent = bkt\_size;  max\_bkt = bkt;  }  if (bkt\_size < min\_ent)  min\_ent = bkt\_size;  total += bkt\_size;  fout << endl;  }  }  #endif |
| /\* CyclicShiftHashCode.h \*/  #ifndef CSHC\_H  #define CSHC\_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 // !CSHC\_H |
| /\* Entry.h \*/  #ifndef E\_H  #define E\_H  #include <iostream>  #include <string>  #include <iomanip>  using namespace std;  template<typename K, typename V>  class Entry  {  friend ostream& operator<<(ostream& fout, Entry<K, V>& entry)  {  if (entry.getValue() != NULL)  fout << "(" << setw(8) << entry.getKey() << ": " << \*(entry.getValue()) << ")";  return fout;  }  public:  Entry(K key, V value)  : \_key(key), \_value(value) { }  Entry()  : \_key(), \_value() { } // 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()));  }  const Entry<K, V>& operator=(const Entry<K, V>& right); // 뭐야, 뭐가 문제야?  void fprint(ostream fout); // 레퍼런스로 해야하나?  private:  K \_key;  V \_value;  };  template<typename K, typename V>  const Entry<K, V>& Entry<K, V>::operator=(const Entry<K, V>& right)  {  this->\_key = right.getKey();  this->\_value = right.getValue();  return \*this;  }  template<typename K, typename V>  void Entry<K, V>::fprint(ostream fout)  {  fout << "[Key:" << setw(2) << this->getKey() << ", " << \*(this->getValue()) << "]";  }  #endif // !E\_H |
| /\* 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)  {  const string wd\_ty[] = { "n", "v", "adj", "adv", "prepos" };  list<string>::iterator itr;  fout << mv.keyWord << "(" << wd\_ty[mv.type] << "):" << endl;  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()  : keyWord(), type(), thesaurus(), usages()  {} // 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 = 130; // 1, 3번 20개, 2번 : 130개  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("aptitude", NOUN, {"gift", "talent"}, {"My grandson and granddaughter have a special aptitude that makes me happy in its own way."}),  MyVoca("certificate", NOUN, {"license", "authorization"}, {"If you use this certificate for escape the danger just for you, you will regret for use."}),  MyVoca("certificate", VERB, {"autheticate", "authorize"}, {"The teacher who teaches safe driving wants to certificate the ability about his student's safe driving."}),  MyVoca("eligible", ADJ, {"fit", "acceptable", "qualified", "suitable"}, {"That eligible guy who wants to be a husband with you send a bunch of flowers with his hand letter."}),  MyVoca("gratitude", NOUN, {"appreciation", "thankfulness"}, {"If your gratitude about last work is real, you can not do it like this."}),  MyVoca("hesitant", ADJ, {"uncertain", "doubtful", "undecided"}, {"That hesitant guy who can not decide everything well go to the counseling center and wants to solve that problem about it."}),  MyVoca("proficient", ADJ, {"adept", "able", "skilled"}, {"that guy who is talking a lot is proficient for his job that is dealing with customers."}),  MyVoca("recruit", NOUN, {"draftee", "newcomer", "rookie"}, {"That recruit who is drafted yesterday looks like nervous and about to make a lot of accidents."}), // 10  MyVoca("recruit", VERB, {"draft", "conscript", "employ"}, {"ministry of national defense of south Korea recruit youth guys who is healthy and responsible."}),  MyVoca("resume", NOUN, {"summary", "schema"}, {"Your resume about job you had before is clear what are you want to do."}),  MyVoca("resume", VERB, {"continue", "restart"}, {"You played game last 10 hours, so you must resume after you take a nap."}),  MyVoca("sophisticate", NOUN, {"trend setter", "socialite"}, {"That sophisticate who knows the way of the world is deal badly with a person."}),  MyVoca("sophisticate", VERB, {"improve", "adulterate", "corrupt"}, {"That guy who looks pure enough is sophisticate on his every talking that is not real."}),  MyVoca("stipulate", VERB, {"rule", "decide on conditions", "specify"}, {"That guy who does not know me can not stipulate my job and I will do my job freely."}),  MyVoca("deteriorate", VERB, {"decay", "decline"}, {"The man who has a big and sharp nose deteriorate his friend's health by decayed food."}),  MyVoca("initiative", NOUN, {"action", "leadership"}, {"The man who really want to go travel has an initiative about travel with his friends."}),  MyVoca("refrain", NOUN, {"melody", "theme"}, {"The woman who is talented on compose music makes a refrain about her friend's song."}),  MyVoca("refrain", VERB, {"avoid", "abstain"}, {"The woman who wants to diet due to next month refrains to eat noodle or etc."}),  MyVoca("correspond", VERB, {"agree", "communicate in writing"}, {"The man who is in army that is far from his hometown corresponds with his girlfriend."}),  MyVoca("compatible", ADJ, {"agreeable", "adaptable"}, {"The woman who got a part time job is compatible with her school life and study."}),  MyVoca("elaborate", ADJ, {"intricate", "involved"}, {"The woman who makes a graduation product can make it more elaborate with her ability."}),  MyVoca("elaborate", VERB, {"make detailed", "expand"}, {"The man who makes a deal with his customer has to elaborate his new product."}),  MyVoca("entail", VERB, {"require", "involve"}, {"The woman who really wants to be a leader has to entail leadership and ability."}), // 20  MyVoca("endure", VERB, {"suffer", "sustain"}, {"The man who is training in army has to endure mental and physical pain on his own."}),  MyVoca("succinct", ADJ, {"brief", "blunt"}, {"The woman who is about to do a presentation has to talk succinct about her story or argument."}),  MyVoca("premise", NOUN, {"hypothesis", "argument"}, {"The man who talked before on the stage wants his premise are correct and good."}),  MyVoca("premise", VERB, {"hypothesize", "precede"}, {"The woman who really wants to be a Hollywood star premises she has to make a history about her."}),  MyVoca("abide", VERB, {"remain", "stay"}, {"A man who travels in New York city, US for a first time abides for a month."}),  MyVoca("obligate", VERB, {"require", "constrain"}, {"That guy who believes me so much obligate me to be a good leader of a team."}),  MyVoca("resolve", VERB, {"fix", "determine"}, {"That guy who is important to me resolved to be a responsible man for me."}),  MyVoca("resolve", NOUN, {"resolution", "determination"}, {"That woman who makes a resolve is going to be a doctor until twenty-five years old."}),  MyVoca("administer", VERB, {"manage", "direct"}, {"The man who is manager of baseball team really wants to administer his team as he wants."}),  MyVoca("substitute", NOUN, {"surrogate", "make shift"}, {"The man who does not have ability about his job want to use substitute about work"}),  MyVoca("substitute", VERB, {"replace", "displace"}, {"The man who got a talent about his job want to be a substitute about his colleague."}),  MyVoca("adverse", ADJ, {"arbitrate", "facilitate"}, {"The women who adverse face-to-face study in college wants to do Non-face-to-face study at home."}),  // 추가  MyVoca("mediate", VERB, {"arbitrate", "facilitate"}, {"The woman who wants to mediate her friends is throw a party for her friends."}),  MyVoca("accommodate", VERB, {"oblige", "serve"}, {"The man who is president of some country must accommodate the public of his country."}), // 30  MyVoca("frequent", ADJ, {"common", "normal"}, {"Fighting with his best friend who is not good due to not important things is frequent."}),  MyVoca("frequent", VERB, {"visit", "haunt"}, {"The man who really like travelling frequents a tourist attraction in all over of KOREA."}),  MyVoca("revenue", NOUN, {"income", "proceeds"}, {"The man who is the richest person of KOREA makes the most revenue on his job."}),  MyVoca("procure", VERB, {"acquire", "obtain"}, {"The man who has to procure many stuffs is a manager of his baseball team."}),  MyVoca("diverse", ADJ, {"different", "various"}, {"The woman who got a diverse ability form god is going to be an entertainer."}),  MyVoca("prerequisite", NOUN, {"condition", "necessity"}, {"The man who has to keep his prerequisite of project is a student at university."}),  MyVoca("endorse", VERB, {"sanction", "approve"}, {"The woman who endorses her friend for class president is vice president of her class."}),  MyVoca("rectify", VERB, {"remedy", "cure"}, {"The man who rectifies his English essay is student of lecture that is English communication skill."}),  MyVoca("mandatory", ADJ, {"obligatory", "required"}, {"The game shop has a mandatory policy of all game stuff are sample such as just case of game."}),  MyVoca("consistent", ADJ, {"reliable", "steady"}, {"We always go to only one rice soup store because of consistent taste and service."}),  MyVoca("dominate", VERB, {"control", "rule"}, {"The woman who dominates her country is the late queen who got a wise lord."}), // 40  MyVoca("retail", NOUN, {"trade", "selling"}, {"The retail store of my city has everything what I want such as water and beer."}),  MyVoca("retail", VERB, {"sell", "peddle"}, {"The store in my city retails all of store that are spread on all of country."}),  MyVoca("invoice", NOUN, {"statement", "demand"}, {"If you want to check the position of your stuff, you have to check your invoice number int the website."}),  MyVoca("invoice", VERB, { "bill", "charge" }, { "If you send your stuff to your friend, you have to invoice on your box." }),  MyVoca("deduct", VERB, { "remove", "withhold" }, { "By deducting my phone fee from SK telecom, I will pay it on a half." }),  MyVoca("accrue", VERB, { "accumulate", "ensue" }, { "By growing my stack itself, my money is accrued as much as Warren Buffett’s money." }),  MyVoca("liquidate", VERB, { "settle", "discharge" }, { "He wants to liquidate his money that is his the most important retirement fund and installment saving." }),  MyVoca("plummet", NOUN, { "thepiece of lead", "bob of a plumb line" }, { "The plummet of value of a company led to the collapse of the company." }),  MyVoca("plummet", VERB, { "fall", "plunge" }, { "A man who wants to plummet into that pond usually do a water game with his friends." }),  MyVoca("subsidiary", ADJ, { "lesser", "secondary" }, { "A man who is subsidiary professor goes to a lecture room for teach his students." }),  MyVoca("subsidiary", NOUN, { "company", "firm" }, { "A man who is subsidiary about a lecture has to prepare devices about that lecture." }),  MyVoca("bilateral", ADJ, { "two-sided", "joint" }, { "A man who is bilateral friend is good to me and also bad to me." }),  MyVoca("audit", NOUN, { "review", "check" }, { "A man who is surprised by audit has to organize his data about his company." }),  MyVoca("audit", VERB, { "check", "review" }, { "A man who has to audit a company has to prepare data about an audit." }),  MyVoca("reconcile", VERB, { "settle", "reunite" }, { "A man who did a fault with his sister reconcile to a girl with his sister." }), // 50  MyVoca("commit", VERB, { "obligate", "pledge" }, { "A woman who commits to her friend with her sister apologizes to her friend with her sister." }),  MyVoca("yield", NOUN, { "harvest", "profit" }, { "A man who calculates yield about harvest is going to be a manager about a company." }),  MyVoca("yield", VERB, { "generate", "return" }, { "A man who yields his own thing makes his own money with his partner who is his friend." }),  MyVoca("affiliate", VERB, { "associate", "unite", "join", "link" }, { "A man who works in an affiliate earns less than his friend who works in a head office." }),  MyVoca("affiliate", NOUN, { "partner", "member", "colleague" }, { "A man who is Boss of a company affiliates his company with his partner’s company." }),  MyVoca("intervention", NOUN, { "interference", "intercession", "mediation" }, { "A woman who got a good intervention skill is good friend to all people including me." }),  MyVoca("scrutiny", NOUN, { "inspection", "examination", "analysis", "enquiry", "study" }, { "A man who makes a scrutiny with his partner will get an A grade on his lecture." }),  MyVoca("consolidate", VERB, { "combine", "strengthen", "merge" }, { "A woman who is leader of team consolidates her study group with her partner's study group." }),  MyVoca("weave", VERB, { "interlace", "knit", "plait", "intertwine" }, { "A man who got a good weave skill will makes good clothes to all customers." }),  MyVoca("weave", NOUN, { "pile", "texture", "nap" }, { "A woman who weaves her friend's clothes with her material is thanked by her all friends." }),  MyVoca("fluctuate", VERB, { "vary", "swing", "oscillate", "waver" }, { "A man who is boss of his company fluctuates his product price frequently with his partner." }),  MyVoca("legitimate", ADJ, { "lawful", "legal", "reasonable", "valid", "genuine" }, { "A woman who makes legitimate policy is already known to all companies including our company." }),  MyVoca("legitimate", VERB, { "legalize", "legitimize", "authorize" }, { "A man who legitimates his talking is good at talk with his friend and partner." }),  MyVoca("collborate", VERB, { "cooperate", "participate", "team up" }, { "A woman who collaborates team project about lecture with her partner is good at her area." }), // 60  MyVoca("commodity", NOUN, { "product", "item", "service", "goods" }, { "A man who makes commodity such as pencil, eraser, and post-it is leader of start-up." }),  MyVoca("apprehensive", ADJ, { "anxious", "nervous", "edgy", "uneasy" }, { "A woman who is apprehensive girl is nervous on her all things such as studying and meeting." }),  MyVoca("compromise", NOUN, { "cooperation", "settlement", "concession", "conciliation" }, { "The man who feels angry and his friend made a compromise and will eat lunch together." }),  MyVoca("compromise", VERB, { "cooperate", "bargain", "negotiate", "concede" }, { "The woman who wants to eat bread and milk compromised by eating rice with soup." }),  MyVoca("secure", ADJ, { "safe", "confident", "dependable", "firm" }, { "The woman who got a weak mind is not secure from other people and society." }),  MyVoca("secure", VERB, { "obtain", "lock", "fix", "ensure", "guarantee" }, { "The man who made a promise with his elder friend secures it hard on himself." }),  MyVoca("relinquish", VERB, { "surrender", "abandon", "renounce" }, { "The man who did fault to his friend relinquishes himself to be a friend with his friend again." }),  MyVoca("coordinate", NOUN, { "coordination", "equal", "altitude and longitude" }, { "Coordinate is important in all situations such as writing essays, talk with friends, and wear clothes." }),  MyVoca("coordinate", VERB, { "organize", "manage", "align" }, { "The woman who makes several friends who do not know each other cannot coordinate their relationship." }),  MyVoca("presume", VERB, { "believe", "venture", "assume", "guess" }, { "The man who studies hard presumes that he will get an A grade in the lecture." }),  MyVoca("subjective", ADJ, { "personal", "slanted", "individual" }, { "The woman who made an opinion about her lecture is subjective on her own argument." }),  MyVoca("assort", VERB, { "classify", "separate", "divide" }, { "The man who assorts his own pieces of stuff makes it tidy like a department store." }),  MyVoca("incorporate", VERB, { "join", "merge", "combine", "contain" }, { "The woman who incorporates her own companies can be the richest person in her country." }), // 70  MyVoca("exquisite", ADJ, { "beautiful", "excellent", "sensitive", "intense", "dainty" }, { "The man who is exquisite in his sense is sharp to his job such as research and study." }),  MyVoca("regulate", VERB, { "control", "moderate", "order", "adjust" }, { "The woman who regulates her son to stay at home stays beside her son together." }),  MyVoca("prohibit", VERB, { "forbid", "exclude", "outlaw", "ban" }, { "The convenience store prohibits to sell liquor like soju and beer to non-adult, but it can sell to adult." }),  MyVoca("expanse", NOUN, { "area", "spread", "stretch", "span" }, { "You can expanse your knowledge by studying what you want to learn, do, and read." }),  MyVoca("comprehend", VERB, { "realize", "understand", "know", "grasp" }, { "The man who comprehends what he has to do is good at his every job." }),  MyVoca("punctual", ADJ, { "prompt", "timely" }, { "The woman who is punctual on her promises is popular to her all friends and colleagues." }),  MyVoca("accommodate", VERB, { "house", "lodge", "adapt", "contain", "assist" }, { "The break room is good enough to accommodate our needs to eat lunch and study." }),  MyVoca("incidental", ADJ, { "related", "attendant", "nonessential", "secondary" }, { "The argument what you said is fact, but it is incidental thing of my argument." }),  MyVoca("incidental", NOUN, { "extras", "contingencies", "expenses" }, { "Incidentals like what your PC is down, and power is over can make your projects collapsed." }),  MyVoca("itinerary", NOUN, { "route", "journey", "circuit" }, { "You have to change your itinerary when you get more project from your supervisor or CEO." }),  MyVoca("substantial", ADJ, { "considerable", "solid", "real", "extensive" }, { "The bus company had a substantially lower review for provided service than our competitors had." }), // 80  MyVoca("amenity", ADJ, { "facility", "pleasantness", "comfort" }, { "The convenience store gives us many amenities such as every meal, life stuffs, and new things." }),  MyVoca("quote", VERB, { "cite", "repeat", "estimate", "offer" }, { "The man who quoted what his father said before to him lives on his father's way." }),  MyVoca("attainment", NOUN, { "achievement", "accomplishment" }, { "The man who works for the company received a lot of gratitude for his many attainments." }),  MyVoca("indulge", VERB, { "spoil", "pamper" }, { "The man who has first shopping in his life indulges to spend his entire money." }),  MyVoca("improvise", VERB, { "extemporize", "create" }, { "The woman who is the best singer in her country improvises her song for her parents." }),  MyVoca("instinct", NOUN, { "aptitude", "feeling" }, { "The man who has unbalanced music taste has the instinct to choose music that he wants to listen to." }),  MyVoca("conjunction", NOUN, { "combination", "aggregation" }, { "Conjunctions such as \"although\", \"but\", and \" and \" usually used in connecting sentences in several ways." }),  MyVoca("disseminate", VERB, { "spread", "circulate" }, { "The internet that disseminates a lot of pieces of information made by various people is helpful to live." }),  MyVoca("description", NOUN, { "account", "explanation" }, { "The description of the project did not match what we saw in his oral presentation." }),  MyVoca("fascinate", VERB, { "captivate", "charm" }, { "The fascination of her boyfriend is bigger and bigger by receiving a birthday present by her boyfriend." }), // 90  MyVoca("broaden", VERB, { "widen", "extend" }, { "If my kid eats various food in balance, his taste will broaden like a universe." }),  MyVoca("investigate", VERB, { "consider", "examine" }, { "The police officer needs to investigate the fact about an accident and clear up the situation." }),  MyVoca("assess", VERB, { "measure", "tax", "evaluate", "charge", "judge" }, { "The neighboring country’s government has assessed the largest tax ever to the nation’s top companies." }),  MyVoca("fatigue", NOUN, { "exhaustion", "tiredness", "weariness", "weakness" }, { "The national team and other athletes are under endless fatigue after important competitions such as the Olympics." }),  MyVoca("abate", VERB, { "decrease", "subside", "decline", "fail", "stop" }, { "The coronavirus has abated people’s population movement and reduced the economy at a very high rate of infection." }),  MyVoca("nutrution", NOUN, { "nourishment", "diet", "food", "sustenance" }, { "Eating breakfast and morning nutrition is very important to our health and daily lives." }),  MyVoca("pertinent", ADJ, { "relevant", "related", "apposite", "appropriate", "germane" }, { "Anyone pertinent in yesterday’s crime should go to the police station for questioning and then go home." }),  MyVoca("diagnose", VERB, { "identify", "analyze", "spot", "detect", "establish" }, { "The doctor who diagnosed my wife’s illness yesterday told me to come to the hospital again." }),  MyVoca("meager", ADJ, { "scanty", "insufficient", "inadequate", "stingy", "skimpy" }, { "My new member of the gym is a too meager body, eat lots of rice and exercise a lot." }),  MyVoca("aggravate", VERB, { "worsen", "exacerbate", "annoy", "irritate", "exaggerate" }, { "The worst friend I have ever had in my life is bad-natured and it aggravates everyone, including me." }), // 100  MyVoca("deficient", ADJ, { "lacking", "faulty", "scarce", "flawed", "underprovided" }, { "I will study a lot at university because I am deficient in my skill to learn new technology in the future." }),  MyVoca("prescribe", VERB, { "recommend", "set", "fix", "suggest", "stipulate" }, { "I had a fever because the medicine prescribed by the doctor at the hospital I went to yesterday did not work." }), // 102  //{ "-1", NOUN, { "" }, { "" } }, // end sentinel  }; |
| /\* Exceptions.h \*/  #ifndef EXCP\_H  #define EXCP\_H  #include <string>  class NonexistentElement  {  public:  NonexistentElement()  : err\_mail(NULL) { }  NonexistentElement(string em)  : err\_mail(em) { }  private:  string err\_mail;  };  #endif // !EXCP\_H |
|  |
| Cyclic Shift Hash Code 함수를 통해 주어진 영어 단어들에 대한 32–bit hash code를 생성한 후, 모듈로 연산의 압축함수를 사용하여 6-bit 크기의 hash value를 계산했다.  계산 결과에 충돌이 발생한 경우가 있으며, 이는 각 단어가 다른 hash code를 가지더라도 압축 과정에서 같은 hash value가 나올 수 있기 때문에 충돌이 발생했다. |

**11.3 SkipList**

|  |
| --- |
| /\* main.cpp \*/  /\* Description  \* Hash Dictionary 설계 및 구현  \* Programmed by J. H. Kim  \* Last updated : 2021-11-24 \*/  #include <iostream>  #include <fstream>  #include <string>  #include "GenericSkipList.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;  SkipList<string, MyVoca\*> myVocaSL;  typedef SkipList<string, MyVoca\*>::Position Pos\_planet;  unsigned int h = 0;  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;  Pos\_planet pPlanet;  string id;  fout << " Inserting myVocas to";  fout << "myVoca SkipList ..." << endl;  for (int i = 0; i < NUM\_MY\_TOEIC\_VOCA; i++)  {  fout << "Inserting [";  fout << i << "]: " << myToeicVocaList[i] << endl;  id = myToeicVocaList[i].getKeyWord();  myVocaSL.SkipListInsert(fout, id, &myToeicVocaList[i]);  myVocaSL.PrintSkipList(fout);  fout << endl;  }  fout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* " << endl;  for (int i = 0; i < NUM\_MY\_TOEIC\_VOCA; i++)  {  fout << "Search planetSkipList and delete";  fout << " the planet(";  fout << myToeicVocaList[i] << ") : ";  pPlanet = myVocaSL.SkipListSearch(myToeicVocaList[i].getKeyWord());  if (pPlanet != Pos\_planet(NULL))  {  fout << \*pPlanet << endl;  myVocaSL.SkipListRemove(fout, myToeicVocaList[i].getKeyWord());  }  myVocaSL.PrintSkipList(fout);  fout << endl;  }  } |
| /\*\* Generic Skip List.h (1) \*/  #ifndef GSL\_H  #define GSL\_H  #include <iostream>  #include <stdlib.h>  #include <iomanip>  #include <time.h>  #include <limits>  #include <string>  #define PROBABILITY\_OF\_ABOVE\_LEVEL 2  #define PLUS\_INF "~"  #define MINUS\_INF ""  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;  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  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(const Position& p) { pQN->down = p.pQN; }  void setAbove(const Position& p) { pQN->up = p.pQN; }  void setAfter(const Position& p) { pQN->next = p.pQN; }  void setBefore(const Position& p)  {  pQN->prev = p.pQN;  }  bool operator==(const Position& q)  {  return pQN == q.pQN;  }  bool operator!=(const Position& q)  {  return this->pQN != q.pQN;  }  Element& operator\*() { return pQN->element(); }  friend class SkipList;  }; // end of class Position  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));  }  Position SkipListSearch(const 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;  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;  }  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;  }  }  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;  }  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;  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;  }  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(11) << pxy.key();  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(12);  fout << pxy.key();  }  px0 = px0.after();  }  if (pxy.key() == PLUS\_INF)  fout << "₩ + oo";  else  fout << setw(11) << pxy.key();  fout << endl;  p0y = p0y.below();  pxy = p0y;  }  fout << endl;  }  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  {  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 |
| /\* 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)  {  const string wd\_ty[] = { "n", "v", "adj", "adv", "prepos" };  list<string>::iterator itr;  fout << mv.keyWord << "(" << wd\_ty[mv.type] << "):" << endl;  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()  : keyWord(), type(), thesaurus(), usages()  {} // 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 = 20; // 1, 3번 20개, 2번 : 130개  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("aptitude", NOUN, {"gift", "talent"}, {"My grandson and granddaughter have a special aptitude that makes me happy in its own way."}),  MyVoca("certificate", NOUN, {"license", "authorization"}, {"If you use this certificate for escape the danger just for you, you will regret for use."}),  MyVoca("certificate", VERB, {"autheticate", "authorize"}, {"The teacher who teaches safe driving wants to certificate the ability about his student's safe driving."}),  MyVoca("eligible", ADJ, {"fit", "acceptable", "qualified", "suitable"}, {"That eligible guy who wants to be a husband with you send a bunch of flowers with his hand letter."}),  MyVoca("gratitude", NOUN, {"appreciation", "thankfulness"}, {"If your gratitude about last work is real, you can not do it like this."}),  MyVoca("hesitant", ADJ, {"uncertain", "doubtful", "undecided"}, {"That hesitant guy who can not decide everything well go to the counseling center and wants to solve that problem about it."}),  MyVoca("proficient", ADJ, {"adept", "able", "skilled"}, {"that guy who is talking a lot is proficient for his job that is dealing with customers."}),  MyVoca("recruit", NOUN, {"draftee", "newcomer", "rookie"}, {"That recruit who is drafted yesterday looks like nervous and about to make a lot of accidents."}), // 10  MyVoca("recruit", VERB, {"draft", "conscript", "employ"}, {"ministry of national defense of south Korea recruit youth guys who is healthy and responsible."}),  MyVoca("resume", NOUN, {"summary", "schema"}, {"Your resume about job you had before is clear what are you want to do."}),  MyVoca("resume", VERB, {"continue", "restart"}, {"You played game last 10 hours, so you must resume after you take a nap."}),  MyVoca("sophisticate", NOUN, {"trend setter", "socialite"}, {"That sophisticate who knows the way of the world is deal badly with a person."}),  MyVoca("sophisticate", VERB, {"improve", "adulterate", "corrupt"}, {"That guy who looks pure enough is sophisticate on his every talking that is not real."}),  MyVoca("stipulate", VERB, {"rule", "decide on conditions", "specify"}, {"That guy who does not know me can not stipulate my job and I will do my job freely."}),  MyVoca("deteriorate", VERB, {"decay", "decline"}, {"The man who has a big and sharp nose deteriorate his friend's health by decayed food."}),  MyVoca("initiative", NOUN, {"action", "leadership"}, {"The man who really want to go travel has an initiative about travel with his friends."}),  MyVoca("refrain", NOUN, {"melody", "theme"}, {"The woman who is talented on compose music makes a refrain about her friend's song."}),  MyVoca("refrain", VERB, {"avoid", "abstain"}, {"The woman who wants to diet due to next month refrains to eat noodle or etc."}),  MyVoca("correspond", VERB, {"agree", "communicate in writing"}, {"The man who is in army that is far from his hometown corresponds with his girlfriend."}),  MyVoca("compatible", ADJ, {"agreeable", "adaptable"}, {"The woman who got a part time job is compatible with her school life and study."}),  MyVoca("elaborate", ADJ, {"intricate", "involved"}, {"The woman who makes a graduation product can make it more elaborate with her ability."}),  MyVoca("elaborate", VERB, {"make detailed", "expand"}, {"The man who makes a deal with his customer has to elaborate his new product."}),  MyVoca("entail", VERB, {"require", "involve"}, {"The woman who really wants to be a leader has to entail leadership and ability."}), // 20  //{ "-1", NOUN, { "" }, { "" } }, // end sentinel  }; |
|  |