**Lab 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 | 2021.11.19.금 |

|  |
| --- |
| /\* main.cpp \*/  /\* Description  \* Hash Dictionary 설계 및 구현  \* Programmed by J. H. Kim  \* Last updated : 2021-11-19 \*/  #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\*> myVocaDict;  HashDict<string, MyVoca\*>::Iterator itr;  HashDict<string, MyVoca\*>::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.operator\*().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.operator\*().getValue();  fout << \*pVoca << endl;  }  fout << endl;  fout.close();  } |
| /\* 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 = 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  }; |
| /\* 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 |
| /\* 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 101  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); // 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  #endif |
| /\* HashMap.cpp \*/  #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());  }  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;  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)  {  Iterator p(B, bkt, bkt->begin());  MyVoca\* pVoca;  while (p.ent != bkt->end())  {  pVoca = p.operator\*().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;  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 << endl << "Max\_ent(" << setw(2) << max\_ent << "), min\_ent(" << setw(2)  << 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);  } |
| /\* HashDictionary.h \*/  #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  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 = 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 |
| /\* 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 |
| /\* 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 |
|  |

**2. 2021-2 객체지향형 프로그래밍과 자료구조 실습 Oral Test**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 학번 | 21812158 | 성명 | 김주환 | 점수 |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **(1) 문자열 (string) 자료형의 키워드에 대한 Hash code 계산에서 주로 많이 사용되는 Cyclic Shift Hash Code 대하여 상세하게 설명하라.**   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **키워드가 “Yeungnam”일 때 hash code 값의 각 단계별 계산** | | | | | | | **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 | 0000 0000 | 0000 0000 | 0000 0000 | 59 | 0000 0059 | | 1 | 0000 0059 | 0000 0B20 | 0000 0000 | 65 | 0000 0B85 | | 2 | 0000 0B85 | 0001 70A0 | 0000 0000 | 75 | 0001 7115 | | 3 | 0001 7115 | 002E 22A0 | 0000 0000 | 6E | 002E 230E | | 4 | 002E 230E | 05C4 61CD | 0000 0000 | 67 | 05C4 6227 | | 5 | 05C4 6227 | B88C 44E0 | 0000 0000 | 6E | BB8C 454E | | 6 | B88C 454E | 1188 A9C0 | 0000 0017 | 61 | 1188 AA38 | | 7 | 1188 AA38 | 3115 4700 | 0000 0002 | 6D | 3115 476F |   Hash에서는 주어진 문자열을 Hash 함수로 계산하여 고르게 분산된 값을 구해야한다.    우측 사진과 같이 두 단계로 구성되는데 우선 Hash Function으로 Hash Code를 구하고 Compression Function을 통해 Hash Value를 구한다.  여기서는 Hash Function인 Cyclic Shift Hash Code를 알아보겠다.  Cyclic Shift는 우측 코드와 같이 h에 지정된 BIT\_SHIFT만큼 왼쪽 비트이동 시킨 것과 (BIT\_INT – BIT\_SHIFT) 만큼 우측 비트이동 시킨 값의 Or을 h에 저장한다. 이후 해당 index의 ASCII 값(unsigned int)을 h에 더해주는 과정이 해당 문자열 전체에 반복된다.  추가로 BIT\_SHIFT는 5로 지정했다. 왜냐하면, 5로 지정했을 경우에 가장 분산이 잘 되어있고 충돌이 적었기 때문이다. |
| **(2) Hash Map을 STL vector 와 STL list 로 구현하는 경우, 내부 구조를 그림으로 표현하고, 구현하는 방법에 대하여 상세하게 설명하라.**  (1 ~ 2) Bucket과 Bucket Array의 구성   |  | | --- | |  | | Hash Map은 stl vector와 stl list를 이용한 2차원 구조로 구현된다.  Bucket에 해당하는 부분은 std::list<T\_Entry>이고,  Bucket Array에 해당하는 부분은 std::vector<Bucket>입니다.  또한 자료에 접근하기 위해 Iterator를 선언해야한다.  Bucket Array의 포인터를 ba, 어떤 Bucket에 접근할지에 대한 Iterator를 BItor, Bucket 내부의 어떤 Entry에 접근할지에 대한 Iterator를 EItor로 선언한다. |   (3 ~ 5) 키워드에 대한 Hash Value의 계산 & 해당 Bucket 탐색 & Bucket 내부 Entry 탐색   |  | | --- | |  | | Hash Value는 두 단계를 통해 구할 수 있다.  우선, Key String을 Cyclic Shift Hash Code로 연산하여 Hash Code를 구한다.  주어진 Hash Code에 Compression Function을 계산해서 구할 수 있다.  Compression은 두가지 방법을 쓴다.  1. modulo : h = k % N  - N은 소수로 구성한다.  - 주로 사용함  2. MAD (Multiply, Add and Divide) : h = (a \* k + b) % N  - a와 b는 음수가 아니어야한다.  - 주로 사용하지 않는다.  Compression된 Hash Value가 Bucket의 Iterator (BItor)가 된다. | | 키워드에 해당하는Bucket 내부 Entry를 탐색하기 위해 Hash map의 Iterator를 선언합니다.  while문을 통해  1. Bucket의 끝이 아닌 경우  2. 원하는 Key가 아닌 경우에 EItor를 ++연산해줌으로써 원하는 Key를 찾을 때까지 Iterator를 증가시킵니다.  원하는 Key를 찾으면 해당 p에서, 못 찾으면 end를 반환한다. | |
| **(3) Hash Map에서 사용하는 class Iterator 구조와 제공 연산자 오버로딩에 대하여 상세하게 설명하라.**  (1 ~ 2) class HashMap의 class Iterator 구조와 생성자   |  | | --- | |  | | class Iterator는 class HashMap의 public에 포함되며, friend로 구현되어 class HashMap의 private 멤버를 이름을 통해 직접 접근할 수 있다. 그리고 class HashMap의 data에 접근하기 위해 사용된다.  private 영역에는 Bucket Array의 포인터를 ba, 어떤 Bucket에 접근할지에 대한 Iterator를 BItor, Bucket 내부의 어떤 Entry에 접근할지에 대한 Iterator를 EItor로 선언한다.  public 영역에는 생성자와 연산자 오버로딩 멤버함수가 있다.  생성자는 ba, BItor, EItor를 전달받아 객체를 생성한다. 이때 EItor는 따로 지정하지 않는경우 EItor의 기본생성자가 호출된다. EItor의 기본 생성자가 호출된 Bucket은 비어있다. |   (3) class Iterator의 operator\*() 연산자 오버로딩   |  | | --- | |  | | 연산자 오버로딩 operator\*()는 Entry&를 반환하는 연산자다. |   (4) class Iterator의 operator++() 연산자 오버로딩   |  | | --- | |  | | 연산자 오버로딩 operator++()는 Iterator를 하나씩 증가시키는 연산자다.  EItor를 증가시키면서 Bucket의 마지막에 도달하면, BItor를 증가시킨다.  이때, 마지막 Bucket이 아니거나 Bucket이 비어 있으면, 다시 BItor를 증가시킨다.  만약, 마지막 Bucket에 도달했으면, 해당 \*this를 반환한다.  그렇지 않다면, 다음 Bucket으로 넘어간 것이기 때문에 EItor를 Bucket의 begin으로 설정한다.  \*this를 반환한다. | |
| **(4) HashDict에서 사용하는 class Range의 구조와 데이터 멤버 및 멤버함수에 대하여 상세 하게 설명하라.**  (1) class HashDict의 class Range 구조 설명   |  | | --- | |  | | class Range는 하나의 Key에 대응되는 Value의 범위에 대한 data를 가지는 class다.  private 영역은 Iterator 자료형의 \_begin과 \_end를 데이터 멤버로 가진다.  이는 범위의 시작과 끝을 나타낸다.  public 영역은 생성자와 begin()과 end()를 가진다.  생성자는 \_begin과 \_end를 전달받아 생성한다.  begin()과 end()는 해당 값을 반환하는 기능을 수행한다. |   (2) class HashDict의 findAll() 멤버함수 실행에서 range의 \_begin과 \_end가 결정되는 과정 설명   |  | | --- | |  | | findAll() 멤버 함수는 같은 Key를 가지는 Entry의 범위를 반환하는 함수다.  우선 Key의 Iterator를 찾아서 b에 저장한다. (begin)  end가 아니고 Key가 아니면, p를 증가시킨다.  p는 Key에 해당하는 수만큼 더해졌을 것이다. (end)  b와 p를 Range에 전달해서 객체를 생성하여 반환한다. |   (3) range.begin()과 range.end()를 사용하여 해당 구간의 vocabulary들을 출력하는 방법 설명   |  | | --- | |  | | 테스트할 단어를 선정한다.  HashDict의 findAll()을 통해 Range객체를 반환받는다.  반복문의 반복을 결정하는 인자로 Range의 begin()과 end()를 사용하겠다.  우선, Iterator itr을 선언한다. 해당 itr을 index로 사용하겠다.  range.begin()을 통해 itr을 range의 \_begin으로 초기화한다.  이 반복문은 itr이 \_end가 아니게 될 때까지 itr을 증가시키며 반복한다.  (이때, \_end는 range.end()를 통해 반환)  반복문 내부에서는 itr의 Value (MyVoca\*)를 전달받는다.  이때 \*pVoca (MyVoca)로 출력한다. |   추가로 Range는 list 내부에 a, a, ab가 있는 경우(같은 해시 값) 구분해주는 역할도 수행한다. |