**Homework**

|  |
| --- |
| 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 | 2021.12.01.수 |

**12. 일부 입력된 단어로 구성될 수 있는 예측구문 (predictive text)을 찾아주기 위한 trie 자료구조를 C++ 프로그램으로 구현하고 기능을 시험하라.**

|  |
| --- |
| /\* main.cpp \*/  /\* Description  \* trie 자료구조 구현 및 기능 검사  \* Programmed by J. H. Kim  \* Last updated : 2021-12-01 \*/  #include <iostream>  #include <fstream>  #include <list>  #include <string>  #include "MyVoca.h"  #include "MyVocaList.h"  #include "Trie.h"  #include "TrieNode.h"  using namespace std;  #define MAX\_WORD\_LENGTH 100  #define NUM\_TEST\_VOCAS 7  //#define TEST\_SIMPLE\_TRIE  void test\_simple\_trie(ostream& fout);  void test\_trie\_myVoca(ostream& fout);  void main()  {  ofstream fout;  fout.open("output.txt");  if (fout.fail())  {  printf("Error in opening output file !\n");  exit;  }  Trie<MyVoca\*> trie\_myVoca("Trie\_MyVoca");  TrieNode<MyVoca\*>\* pTN;  MyVoca\* pVoca;  int word\_count;  string keyStr;  char keyWord[MAX\_WORD\_LENGTH];  List\_pVoca predictVocas;  List\_pVoca\_Iter itr;  /\* Testing Basic Operation in trie \*/  MyVoca testVocas[NUM\_TEST\_VOCAS] =  {  MyVoca("xyz", NOUN, { "" }, { "" }),  MyVoca("ABCD", NOUN, { "" }, { "" }),  MyVoca("ABC", NOUN, { "" }, { "" }),  MyVoca("AB", NOUN, { "" }, { "" }),  MyVoca("A", NOUN, { "" }, { "" }),  MyVoca("xy", NOUN, { "" }, { "" }),  MyVoca("x", NOUN, { "" }, { "" }),  };  fout << "Testing basic operations of trie inserting ..... " << endl;  for (int i = 0; i < NUM\_TEST\_VOCAS; i++)  {  trie\_myVoca.insert(testVocas[i].getKeyWord(), &testVocas[i]);  }  trie\_myVoca.fprintTrie(fout);  /\*Destroy the trie\*/  fout << "\nTesting TrieDestroy...\n";  trie\_myVoca.eraseTrie();  trie\_myVoca.fprintTrie(fout);  /\* inserting keyWords into trie \*/  fout << "Inserting My Vocabularies to myVocaDict . . . " << endl;  word\_count = 0;  pVoca = &myToeicVocaList[0];  while (pVoca->getKeyWord() != "-1")  {  keyStr = pVoca->getKeyWord();  trie\_myVoca.insert(keyStr, pVoca);  pVoca++;  }  fout << "Total " << trie\_myVoca.size() << " words in trie\_myVoca .." << endl;  trie\_myVoca.fprintTrie(fout);  /\* testing keyWord search in trie \*/  while (1)  {  cout << "\nInput any prefix to search in trie (. to finish) : ";  cin >> keyStr;  if (keyStr == string("."))  break;  predictVocas.clear();  trie\_myVoca.findPrefixMatch(keyStr, predictVocas);  cout << "list of predictive words with prefix (" << keyStr << ") :" << endl;  itr = predictVocas.begin();  for (int i = 0; i < predictVocas.size(); i++)  {  pVoca = \*itr;  cout << \*pVoca << endl;  ++itr;  }  }  cout << endl;  /\* Testing TrieDeleteKey() \*/  printf("\nTesting trie\_delete\_key for %d keywords from trie data structure.\n", NUM\_MY\_TOEIC\_VOCA);  for (int i = 0; i < NUM\_MY\_TOEIC\_VOCA / 2; i++) {  pVoca = &myToeicVocaList[i];  if ((pVoca == NULL) || (\*pVoca->getKeyWord().c\_str() == '\0'))  continue;  fout << "Trie-Deleting (key : " << pVoca->getKeyWord() << ") ...\n";  trie\_myVoca.deleteKeyStr(pVoca->getKeyWord());  //trieStr.fprintTrie(fout);  }  //trieStr.fprintTrie(fout);  cout << "\nErasing trie\_myVoca ...." << endl;  fout << "\nErasing trie\_myVoca ...." << endl;  trie\_myVoca.eraseTrie();  fout.close();  } |
| /\* Trie.h (1) \*/  #ifndef Trie\_H  #define Trie\_H  #include <iostream>  #include <string>  #include "TrieNode.h"  #define MAX\_STR\_LEN 50  //#define DEBUG  using namespace std;  typedef list<MyVoca\*> List\_pVoca;  typedef list<MyVoca\*>::iterator List\_pVoca\_Iter;  enum SearchMode { FULL\_MATCH, PREFIX\_MATCH };  template <typename E>  class Trie  {  public:  Trie(string name); // constructor  int size() { return num\_keys; }  void insert(string keyStr, E value);  void insertExternalTN(TrieNode<E>\* pTN, string keyStr, E value);  TrieNode<E>\* find(string keyStr);  void findPrefixMatch(string prefix, List\_pVoca& predictWords);  void deleteKeyStr(string keyStr);  void eraseTrie();  void fprintTrie(ostream& fout);  protected:  TrieNode<E>\* \_find(string keyStr, SearchMode sm);  void \_traverse(TrieNode<E>\* pTN, List\_pVoca& list\_keywords);  private:  TrieNode<E>\* \_root; // \_root trie node  int num\_keys;  string trie\_name;  };  template<typename E>  Trie<E>::Trie(string name)  {  trie\_name = name;  \_root = new TrieNode<E>('₩0', NULL);  \_root->setKey('₩0');  \_root->setPrev(NULL);  \_root->setNext(NULL);  \_root->setParent(NULL);  \_root->setChild(NULL);  num\_keys = 0;  }  template<typename E>  void Trie<E>::insertExternalTN(TrieNode<E>\* pTN, string keyStr, E value)  {  TrieNode<E>\* pTN\_New = NULL;  pTN\_New = new TrieNode<E>('₩0', value);  pTN->setChild(pTN\_New);  (pTN->getChild())->setParent(pTN);  pTN\_New->setValue(value);  //cout << "key (" << keyStr << ") is inserted ₩n";  }  template<typename E>  void Trie<E>::insert(string keyStr, E value)  {  TrieNode<E>\* pTN = NULL, \* pTN\_New = NULL;  char\* keyPtr = (char\*)keyStr.c\_str();  if (keyPtr == NULL)  return;  /\* Firstly, check any possible duplicated key insertion \*/  if (\_find(keyStr, FULL\_MATCH) != NULL)  {  cout << "The given key string (" << keyStr << ") is already existing; just return !!" << endl;  return;  }  pTN = this->\_root;  while ((pTN != NULL) && (\*keyPtr != '\0'))  {  if ((pTN->getKey() < \*keyPtr) && (pTN->getNext() == NULL) && (\*keyPtr != '\0'))  break;  while ((pTN->getKey() < \*keyPtr) && (pTN->getNext() != NULL))  pTN = pTN->getNext();  while ((pTN != NULL) && (pTN->getKey() == \*keyPtr) && (\*keyPtr != '\0'))  {  pTN = pTN->getChild();  keyPtr++;  }  if ((pTN->getKey() > \*keyPtr) && (\*keyPtr != '\0'))  break;  } // end while for positioning  /\* Secondly, the given key string is a sub-string of an existing key \*/  /\* e.g.) trying to insert "abc" while "abcde" is already exisiting. \*/  if ((pTN->getKey() != '\0') && (\*keyPtr == '\0'))  {  /\* there was a longer key string already !! \*/  /\* break the longer key string, and connected to the separated key strings \*/  pTN\_New = new TrieNode<E>('\0', value);  pTN\_New->setParent(pTN->getParent());  (pTN->getParent())->setChild(pTN\_New);  pTN\_New->setNext(pTN);  pTN->setPrev(pTN\_New);  //cout << "key (" << keyWord << ") is inserted" << endl;  this->num\_keys++;  return;  }  else if ((pTN->getKey() < \*keyPtr) && (pTN->getNext() == NULL) && (\*keyPtr != '\0'))  {  /\* at this level, a new substring is inserted as the last nodes \*/  pTN\_New = new TrieNode<E>(\*keyPtr, VALUE\_INTERNAL\_NODE);  pTN\_New->setParent(pTN->getParent());  pTN\_New->setPrev(pTN);  pTN->setNext(pTN\_New);  pTN = pTN\_New;  keyPtr++;  while (\*keyPtr != '\0')  {  pTN\_New = new TrieNode<E>(\*keyPtr, VALUE\_INTERNAL\_NODE);  pTN->setChild(pTN\_New);  (pTN->getChild())->setParent(pTN);  pTN = pTN->getChild();  keyPtr++;  }  if (\*keyPtr == '\0')  {  insertExternalTN(pTN, keyStr, value);  this->num\_keys++;  return;  }  }  else if ((pTN->getKey() > \*keyPtr) && (\*keyPtr != '\0'))  {  /\* insert between two existing trie nodes \*/  pTN\_New = new TrieNode<E>(\*keyPtr, VALUE\_INTERNAL\_NODE);  pTN\_New->setNext(pTN);  pTN\_New->setParent(pTN->getParent());  if (pTN->getPrev() == NULL)  { /\* this pTN\_new becomes the new first in this level \*/  if (pTN->getParent() != NULL)  (pTN->getParent())->setChild(pTN\_New);  }  else {  (pTN->getPrev())->setNext(pTN\_New);  }  pTN\_New->setPrev(pTN->getPrev());  pTN->setPrev(pTN\_New);  pTN = pTN\_New;  keyPtr++;  while (\*keyPtr != '\0')  {  pTN\_New = new TrieNode<E>(\*keyPtr, VALUE\_INTERNAL\_NODE);  pTN->setChild(pTN\_New);  (pTN->getChild())->setParent(pTN);  pTN = pTN->getChild();  keyPtr++;  }  if (\*keyPtr == '\0')  {  insertExternalTN(pTN, keyStr, value);  this->num\_keys++;  return;  }  }  }  template<typename E>  TrieNode<E>\* Trie<E>::find(string keyStr)  {  TrieNode<E>\* pTN = NULL;  pTN = \_find(keyStr, FULL\_MATCH);  return pTN;  }  template<typename E>  TrieNode<E>\* Trie<E>::\_find(string keyStr, SearchMode sm)  {  char\* keyPtr;  TrieNode<E>\* pTN = NULL;  TrieNode<E>\* found = NULL;  if (&keyStr == NULL)  return NULL;  keyPtr = (char\*)keyStr.c\_str();  pTN = this->\_root;  while ((pTN != NULL) && (\*keyPtr != '\0'))  {  while ((pTN != NULL) && (pTN->getKey() < \*keyPtr))  { // 없다면 next로 이동하지 못한다.  if (pTN->getNext() == NULL)  return NULL;  pTN = pTN->getNext();  }  if ((pTN != NULL) && (pTN->getKey() > \*keyPtr))  {  // key not found  return NULL;  }  else if ((pTN == NULL) && (\*keyPtr != '\0'))  {  // key not found  return NULL;  }  else if ((pTN->getKey() == \*keyPtr) && (\*keyPtr != '\0'))  {  pTN = pTN->getChild();  keyPtr++;  if (\*keyPtr == '\0')  {  /\* key or prefix found \*/  if (sm == FULL\_MATCH)  {  if (pTN->getKey() == '\0')  {  /\* found the key string as a full-match \*/  return pTN;  }  else // (pTN->getKey() != '\0')  {  /\* found the key string as a substring of a longer existing string \*/  return NULL;  }  }  else if (sm == PREFIX\_MATCH)  {  /\* found the key string as a full-match or as a substring of a longer existing  string \*/  return pTN;  }  }  else if ((pTN->getKey() == '\0') && (\*keyPtr != '\0'))  {  if (pTN->getNext() != NULL)  {  pTN = pTN->getNext();  continue;  }  else  return NULL;  }  else  {  continue;  }  }  } // end while  }  template<typename E>  void Trie<E>::\_traverse(TrieNode<E>\* pTN, List\_pVoca& list\_keywords)  {  if (pTN == NULL)  return;  if (pTN->getChild() == NULL)  {  list\_keywords.push\_back(pTN->getValue());  }  else  {  \_traverse(pTN->getChild(), list\_keywords);  }  if (pTN->getNext() != NULL)  {  \_traverse(pTN->getNext(), list\_keywords);  }  }  template<typename E>  void Trie<E>::findPrefixMatch(string keyStr, List\_pVoca& predictWords)  {  TrieNode<E>\* pPtr = NULL;  const char\* keyPtr;  TrieNode<E>\* pTN = NULL;  TrieNode<E>\* found = NULL;  keyPtr = (char\*)keyStr.c\_str();  if (keyPtr == NULL)  return;  pTN = this->\_root;  pTN = \_find(keyStr, PREFIX\_MATCH);  \_traverse(pTN, predictWords);  //printf("Error in TrieSearch (key: %s) !!\n", keyWord);  }  template<typename E>  void Trie<E>::deleteKeyStr(string keyStr)  {  TrieNode<E>\* pTN = NULL, \* \_root;  TrieNode<E>\* tmp = NULL;  int trie\_val;  \_root = this->\_root;  if (NULL == \_root || "" == keyStr)  return;  pTN = \_find(keyStr, FULL\_MATCH);  if (pTN == NULL)  {  cout << "Key [" << keyStr << "] not found in trie" << endl;  return;  }  while (1)  {  if (pTN == NULL)  break;  if (pTN->getPrev() && pTN->getNext())  {  tmp = pTN;  (pTN->getNext())->setPrev(pTN->getPrev());  (pTN->getPrev())->setNext(pTN->getNext());  free(tmp);  break;  }  else if (pTN->getPrev() && !(pTN->getNext()))  {  tmp = pTN;  (pTN->getPrev())->setNext(NULL);  free(tmp);  break;  }  else if (!(pTN->getPrev()) && pTN->getNext())  {  tmp = pTN;  (pTN->getParent())->setChild(pTN->getNext());  pTN = pTN->getNext();  pTN->setPrev(NULL);  free(tmp);  break;  }  else  {  tmp = pTN;  pTN = pTN->getParent();  if (pTN != NULL)  pTN->setChild(NULL);  free(tmp);  if ((pTN == \_root) && (pTN->getNext() == NULL) && (pTN->getPrev() == NULL))  {  cout << "Now, the trie is empty !!" << endl;  break;  }  }  }  this->num\_keys--;  }  template<typename E>  void Trie<E>::eraseTrie()  {  TrieNode<E>\* pTN;  TrieNode<E>\* pTN\_to\_be\_deleted = NULL;  if (this->\_root == NULL)  return;  pTN = this->\_root;  /\* delete the last key word first \*/  while (pTN != NULL)  {  while ((pTN != NULL) && (pTN->getNext()))  pTN = pTN->getNext();  while (pTN->getChild())  {  if (pTN->getNext())  break;  pTN = pTN->getChild();  }  if (pTN->getNext())  continue;  if (pTN->getPrev() && pTN->getNext())  {  pTN\_to\_be\_deleted = pTN;  (pTN->getNext())->setPrev(pTN->getPrev());  (pTN->getPrev())->setNext(pTN->getNext());  pTN = pTN->getNext();  free(pTN\_to\_be\_deleted);  }  else if (pTN->getPrev() && !(pTN->getNext()))  {  pTN\_to\_be\_deleted = pTN;  (pTN->getPrev())->setNext(NULL);  pTN = pTN->getPrev();  free(pTN\_to\_be\_deleted);  }  else if (!(pTN->getPrev()) && pTN->getNext())  {  pTN\_to\_be\_deleted = pTN;  (pTN->getParent())->setChild(pTN->getNext());  (pTN->getNext())->setPrev(NULL);  pTN = pTN->getNext();  free(pTN\_to\_be\_deleted);  }  else  {  pTN\_to\_be\_deleted = pTN;  if (pTN == this->\_root)  {  /\* \_root \*/  this->num\_keys = 0;  return;  }  if (pTN->getParent() != NULL)  {  pTN = pTN->getParent();  pTN->setChild(NULL);  }  else  {  pTN = pTN -> getPrev();  }  free(pTN\_to\_be\_deleted);  } // end if - else  } // end while  }  template<typename E>  void Trie<E>::fprintTrie(ostream& fout)  {  TrieNode<E>\* pTN;  int line = 1, indent = 0;  fout << "trie ( " << this->trie\_name << ") with "  << this->num\_keys << " trie\_nodes" << endl;  if (this->num\_keys == 0)  {  fout << "Empty trie !" << endl;  return;  }  pTN = this->\_root;  pTN->\_fprint(fout, pTN, indent);  }  #endif |
| /\* TrieNode.h (1) \*/  #ifndef TRIE\_NODE\_H  #define TRIE\_NODE\_H  #include <iostream>  #include <string>  #include <list>  #define VALUE\_INTERNAL\_NODE NULL  using namespace std;  typedef list<string> STL\_list;  template <typename E>  class TrieNode  {  public:  TrieNode() {} // default constructor  TrieNode(char k, E v) : key(k), value(v) { prev = next = parent = child = NULL; }  void setKey(char k) { key = k; }  void setValue(E v) { value = v; }  void setNext(TrieNode<E>\* nxt) { next = nxt; }  void setPrev(TrieNode<E>\* pv) { prev = pv; }  void setParent(TrieNode<E>\* pr) { parent = pr; }  void setChild(TrieNode<E>\* chld) { child = chld; }  char getKey() { return key; }  E getValue() { return value; }  TrieNode<E>\* getPrev() { return prev; }  TrieNode<E>\* getNext() { return next; }  TrieNode<E>\* getParent() { return parent; }  TrieNode<E>\* getChild() { return child; }  void \_fprint(ostream& fout, TrieNode<E>\* pTN, int indent);  private:  char key;  E value;  TrieNode<E>\* prev;  TrieNode<E>\* next;  TrieNode<E>\* parent;  TrieNode<E>\* child;  };  template<typename E>  void TrieNode<E>::\_fprint(ostream& fout, TrieNode<E>\* pTN, int indent)  {  if (pTN == NULL)  {  fout << endl;  return;  }  else  {  if(pTN->key != '0') fout << pTN->key;  \_fprint(fout, pTN->child, indent + 1);  if (pTN->next == NULL)  return;  for (int i = 0; i < indent; i++)  fout << " ";  \_fprint(fout, pTN->next, indent);  }  }  #endif |
| /\*\* 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)  {  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() {} // 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  #include "MyVoca.h"  int NUM\_MY\_TOEIC\_VOCA = 130;  MyVoca myToeicVocaList[]; // defined in MyVocaList.cpp  #endif |
| /\* MyVocaList.cpp \*/  #include "MyVoca.h"  #define TEST\_FULL\_SET  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("offer", NOUN, { "proposal" }, { "He accepted out offer to write the business plan." }),  MyVoca("compromise", NOUN, { "give-and-take", "bargaining", "accommodation" }, { "The couple made a compromise and ordered food to take out." }),  MyVoca("delegate", NOUN, { "representative", "agent", "substitute" }, { "" }),  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  MyVoca("-1", NOUN, { "" }, { "" }) // end sentinel  }; |
|  |
|  |
|  |