**자료구조 이진탐색트리 레포트**

**(C++, C)**

**202112936 엄정석**

**A. 이진 탐색 트리 – 균형 잡힌 트리(성공)**

-문제

|  |
| --- |
| **[자료구조] 이진 탐색 트리 - 균형 잡힌 트리**    삽입, 삭제, 검색 기능을 지원하는 이진탐색트리에서 트리가 균형 (balanced) 잡혀 있는지 확인할 수 있는 isBalanced() 함수를 구현하고자 한다.  이진 탐색 트리에서 균형 잡힘의 정의는 다음과 같다.  \* 현재 노드를 기준으로 왼쪽 서브트리는 balanced여야 한다.  \* 현재 노드를 기준으로 오른쪽 서브트리는 balanced여야한다. \* 현재 노드를 기준으로 왼쪽 서브트리와 오른쪽 서브트리의 높이 (height)의 차이가 1이하인 상태를 말한다.  예를 들어, 다음의 트리는 높이의 차이가 1이하로 균형 잡혀 있다.  A /      \ B         E  /     \         \     C      D         F    **Input**  입력의 첫 줄에는 하고자 하는 작업의 수 n(1 <= n <= 20)이 들어온다.  다음의 n 줄에는 다음과 같은 작업 명령이 들어온다.  I x : key x를 이진 탐색 트리에 삽입한다. (value는 key와 같다)  D x : key x를 이진 탐색 트리에서 삭제한다. (value는 key와 같다)    **Output**  트리가 균형 잡혀있는지를 출력한다.  균형이 잡혀있을 경우 Balanced를 출력하고, 그렇지 않은경우 Unbalanced를 출력한다.    **Sample Input 1**  9 I 35 I 18 I 7 I 3 I 26 D 18 I 12 I 30 I 68  **Sample Ouput 1**  Unbalanced    **Sample Input 2**  5 I 10 I 5 I 2 I 7 I 20  **Sample Ouput 2**  Balanced |
| [자료구조] 리트머스 숙제 Sample 입출력에 대한 추가 설명    이진탐색 트리 - 균형 잡힌 트리  Sample Output1 설명     35   / \   26 68   / \   7 30   /   3    26을 value로 가지는 node와 68을 value로 가지는 노드가 높이 차이가 2임으로 Unbalanced가 출력된다.  (자손 노드가 2개인 노드를 삭제할 시 자기보다 큰 자손 중 가장 작은 자손을 선택함.)    Sample Output2 설명     10   / \   5 20   / \   2 7     모든 노드가 Balanced 함으로 Balanced가 출력한다. |

-코드(C++, 성공)

|  |
| --- |
| #include <cstdio>  #include <cstdlib>  #include <iostream>  #include <stack>  #include <algorithm>  #include <climits> //INT\_MAX, INT\_MIN을 사용하기 위한 헤더  using namespace std;  #define MAX\_QUEUE\_SIZE 100  inline void error(const char\* str) {  fprintf(stderr, "%s\n", str);  exit(1);  }  class BinaryNode {  protected:  int data; //트리에 저장할 데이터  BinaryNode\* left; //왼쪽 자식 노드의 포인터  BinaryNode\* right; //오른쪽 자식 노드의 포인터  public:  BinaryNode(int val = 0, BinaryNode\* l = NULL, BinaryNode\* r = NULL)  : data(val), left(l), right(r) { }  //키 값으로 노드를 탐색하는 함수(노드 클래스에서 순환으로 구현)  BinaryNode\* search(int key) {  if (key == data)  return this;  else if (key < data && left != NULL)  return left->search(key);  else if (key > data && right != NULL)  return right->search(key);  else  return NULL;  }  void setData(int val) { data = val; }  void setLeft(BinaryNode\* l) { left = l; }  void setRight(BinaryNode\* r) { right = r; }  int getData() { return data; } //노드값 반환  BinaryNode\* getLeft() { return left; } //왼쪽 노드값 반환  BinaryNode\* getRight() { return right; } //오른쪽 노드값 반환  bool isLeaf() { return left == NULL && right == NULL; } //잎 노드 여부 확인  };  class CircularQueue {  int front;  int rear;  BinaryNode\* data[MAX\_QUEUE\_SIZE];  public:  CircularQueue() { front = rear = 0; }  bool isEmpty() { return front == rear; }  bool isFull() { return ((rear + 1) % MAX\_QUEUE\_SIZE) == front; }  void enqueue(BinaryNode\* n) {  if (isFull()) { error(" Error: 큐가 포화상태입니다.\n"); }  else {  rear = (rear + 1) % MAX\_QUEUE\_SIZE;  data[rear] = n;  }  }  BinaryNode\* dequeue() {  if (isEmpty()) { error(" Error: 큐가 공백상태입니다.\n"); }  else {  front = (front + 1) % MAX\_QUEUE\_SIZE;  return data[front];  }  }  };  class BinaryTree {  public:  BinaryNode\* root; //루트 포인터  //=====================================================================================  // 자료구조 과제를 위한 함수 중 bool isBalanced() 추가  //=====================================================================================  //균형이 잡혀있는지를 검사하는 함수 - 왼쪽 서브트리와 오른쪽 서브트리의 높이의 차이가 2보다 작은지를 검사  //O(n)의 시간 복잡도를 수행 - 재귀를 없애고, 스택을 이용하여 반복적으로 수행  bool isBalanced() {  if (root == NULL)  return true; // 공백 트리는 균형 잡혀있다고 간주  stack<BinaryNode\*> nodes; //노드를 위한 스택  stack<int> heights; //높이를 위한 스택  nodes.push(root);  heights.push(1); //높이의 초기값은 1  int minimum\_height = INT\_MAX; //무한히 큰 수(설정)  int maximum\_height = INT\_MIN; //무한히 작은 수(설정)  while (!nodes.empty()) {  //스택에서 원소를 뽑아 옴  BinaryNode\* current\_state = nodes.top();  int height = heights.top();  nodes.pop();  heights.pop();  if (current\_state->isLeaf()) {  // 잎 노드인 경우 최소 높이와 최대 높이 갱신  minimum\_height = min(minimum\_height, height);  maximum\_height = max(maximum\_height, height);  //이 과정은 지금 현재의 높이와 현재 기록된 최소/최대 높이 중 최소/최대의 값을 고르는 과정!  }  else {  // 왼쪽 자식 노드가 있는 경우 스택에 추가  if (current\_state->getLeft() != NULL) {  nodes.push(current\_state->getLeft());  heights.push(height + 1); //뽑은 값에서 1 추가  }  // 오른쪽 자식 노드가 있는 경우 스택에 추가  if (current\_state->getRight() != NULL) {  nodes.push(current\_state->getRight());  heights.push(height + 1); //뽑은 값에서 1 추가  }  }  }  return (maximum\_height - minimum\_height) <= 1;  }  //=====================================================================================  // 자료구조 과제를 위한 함수 끝  //=====================================================================================  BinaryTree() : root(NULL) { }  void setRoot(BinaryNode\* node) { root = node; }  BinaryNode\* getRoot() { return root; } //루트 노드값 반환  bool isEmpty() { return root == NULL; } //빈 트리 여부 확인  //이진트리의 순회 연산  void inorder() { printf("\n inorder: "); inorder(root); }  void inorder(BinaryNode\* node) {  //중위 순회: LVR  if (node != NULL) { //루트 노드가 0이 아닐 때  inorder(node->getLeft()); //왼쪽 서브트리  printf(" [%d] ", node->getData()); //루트 노드 처리  inorder(node->getRight()); //오른쪽 서브트리  }  }  void preorder() { printf("\n preorder: "); preorder(root); }  void preorder(BinaryNode\* node) {  //전위 순회: VLR  if (node != NULL) { //루트 노드가 0이 아닐 때  printf(" [%d] ", node->getData()); //루트 노드 처리  preorder(node->getLeft()); //왼쪽 서브트리  preorder(node->getRight()); //오른쪽 서브트리  }  }  void postorder() { printf("\n postorder: "); postorder(root); }  void postorder(BinaryNode\* node) {  //후위 순회: LRV  if (node != NULL) { //루트 노드가 0이 아닐 때  postorder(node->getLeft()); //왼쪽 서브트리  postorder(node->getRight()); //오른쪽 서브트리  printf(" [%d] ", node->getData()); //루트 노드 처리  }  }  //레벨 순회(BFS: 깊이 우선 탐색)  void levelorder() {  printf("\nlevelorder: ");  if (!isEmpty()) {  CircularQueue q;  q.enqueue(root);  //먼저 루트를 큐에 넣는다.  while (!q.isEmpty()) {  BinaryNode\* n = q.dequeue();  //큐가 비어있지 않으면 큐에서 삭제  if (n != NULL) {  //n이 NULL이 아닐 때  printf(" [%d] ", n->getData()); //n을 표시하고  q.enqueue(n->getLeft());//왼쪽 노드를 큐에 넣는다.  q.enqueue(n->getRight());//오른쪽 노드를 큐에 넣는다.  }  //공백 상태가 될 때까지 while문 반복  }  }  printf("\n");  }  //이진트리의 추가 연산  //트리의 노드 개수를 구하는 함수  int getCount() { return isEmpty() ? 0 : getCount(root); }  //순환 호출에 의해 node를 루트로 하는 서브트리의 노드 수 계산 함수  int getCount(BinaryNode\* node) {  if (node == NULL) { return 0; }  return (1 + getCount(node->getLeft()) + getCount(node->getRight()));  }  //트리의 높이를 구하는 함수  int getHeight() { return isEmpty() ? 0 : getHeight(root); }  int getHeight(BinaryNode\* node) {  if (node == NULL) { return 0; }  int hLeft = getHeight(node->getLeft());  int hRight = getHeight(node->getRight());  return ((hLeft > hRight) ? hLeft + 1 : hRight + 1);  //問: 왜 다른 곳과 달리, hLeft, hRight를 쓰는 것일까?  //答: 편의상 하는 것, 굳이 없어도 return만 잘 구현하면 상관 없음  }  //트리의 잎 노드(단말노드) 개수를 구하는 함수  int getLeafCount() { return isEmpty() ? 0 : getLeafCount(root); }  //순환 호출에 의해 node를 루트로 하는 서브트리의 단말 노드 수 계산 함수  int getLeafCount(BinaryNode\* node) {  if (node == NULL) { return 0; }  if (node->isLeaf()) { return 1; } //잎 노드!  else return (getLeafCount(node->getLeft()) + getLeafCount(node->getRight()));  }  };  class BinSrchTree : public BinaryTree {  public:  BinSrchTree(void) { }  ~BinSrchTree(void) { }  //이진 탐색 트리의 탐색 연산  BinaryNode\* search(int key) {  BinaryNode\* node = searchRecur(root, key);  if (node != NULL) {  printf("탐색 성공: 키값이 %d인 노드 = 0x%x\n", node->getData(), node);  }  else {  printf("키값이 %d인 노드 없음\n", key);  }  return node;  }  //키 값으로 노드를 탐색하는 함수(순환적 방법)  //일반 함수로 구현(BinSrchTree의 멤버 함수로 넣어도 됨)  BinaryNode\* searchRecur(BinaryNode\* n, int key) {  if (n == NULL) { return NULL; } //못 찾았다.  if (key == n->getData()) { return n; }  else if (key < n->getData()) { return searchRecur(n->getLeft(), key); }  else return searchRecur(n->getRight(), key);  }  //키 값으로 노드를 탐색하는 함수(반복적인 방법)  //일반 함수로 구현(BinSrchTree의 멤버 함수로 넣어도 됨)  BinaryNode\* SearchIter(BinaryNode\* n, int key) {  while (n != NULL) {  if (key == n->getData()) { return n; }  else if (key < n->getData()) { n = n->getLeft(); }  else { n = n->getRight(); }  }  return n;  }  //=============================================================  //이진 탐색 트리의 삽입 연산  void insert(BinaryNode\* n) {  if (n == NULL) { return; }  if (isEmpty()) { root = n; }  else { insertRucur(root, n); }  }  //이진 탐색 트리의 삼입 함수  void insertRucur(BinaryNode\* r, BinaryNode\* n) {  //root와 key가 같으면 return  if (n->getData() == r->getData()) { return; }  //root보다 key가 작을 때  else if (n->getData() < r->getData()) {  if (r->getLeft() == NULL) { r->setLeft(n); } //root의 왼쪽 자식이 없으면 n이 왼쪽 자식  else { insertRucur(r->getLeft(), n); } //root에 왼쪽 자식이 있으면 순환 호출  }  //root보다 key가 클 때  else {  if (r->getRight() == NULL) { r->setRight(n); } //root의 오른쪽 자식이 없으면 n이 오른쪽 자식  else { insertRucur(r->getRight(), n); } //root에 오른쪽 자식이 있으면 순환 호출  }  }  //=============================================================  //이진 탐색 트리의 삭제 연산  void remove(int key) {  if (isEmpty()) { return; } //빈 트리이면 return  //없앨 노드와 그 노드의 부모 노드를 찾는다.  BinaryNode\* parent = NULL; //부모 노드  BinaryNode\* node = root; //루트 노드  while (node != NULL && node->getData() != key) {  parent = node;  node = (key < node->getData()) ? node->getLeft() : node->getRight();  }  //없앨 노드가 트리에 없음  if (node == NULL) {  printf("Error: 키가 트리에 없습니다.\n");  return;  }  //없앨 노드가 트리에 있음  else { remove(parent, node); }  }  void remove(BinaryNode\* parent, BinaryNode\* node) {  //case 1: 삭제하려는 노드가 단말 노드인 경우 ->단말 노드의 링크를 끊는다.  if (node->isLeaf()) {  if (parent == NULL) { root = NULL; } //node == root이면 루트만 있는 상태  else { //node != root인 경우  if (parent->getLeft() == node) {  parent->setLeft(NULL);  }  else {  parent->setRight(NULL);  }  }  }  //case 2: 삭제하려는 노드가 왼쪽이나 오른쪽 자식만 갖는 경우  else if (node->getLeft() == NULL || node->getRight() == NULL) {  //삭제할 노드의 유일한 자식 노드 => child  BinaryNode\* child = (node->getLeft() != NULL) ?  node->getLeft() : node->getRight();  //삭제할 노드가 루트이면 ==> child가 새로운 root가 됨  if (node == root) { root = child; }  else {  if (parent->getLeft() == node) {  parent->setLeft(child);  }  else {  parent->setRight(child);  }  }  }  //case 3: 삭제하려는 노드가 두 개의 자식이 모두 있는 경우  else {  //삭제하려는 노드의 오른쪽 서브트리에서 가장 작은 노드를 탐색  //succp => 후계 노드의 부모 노드  //succ => 후계 노드: 오른쪽 서브트리에서 가장 key가 작은 노드  BinaryNode\* succp = node;  BinaryNode\* succ = node->getRight(); //왼쪽부터 수행시 getLeft()  while (succ->getLeft() != NULL) { //후계 노드 탐색  succp = succ; //후계 노드의 부모 노드  succ = succ->getLeft(); //후계 노드  }  //후계 노드의 부모와 후계 노드의 오른쪽 자식을 직접 연결  if (succp->getLeft() == succ) {  succp->setLeft(succ->getRight());  }  else { //후계 노드가 삭제할 노드의 바로 오른쪽 자식인 경우  succp->setRight(succ->getRight());  //후계 노드 정보를 삭제할 노드에 복사  }  node->setData(succ->getData());  //삭제할 노드를 후계 노드로 변경: 실제로는 후계 노드가 제거됨  node = succ;  }  delete node; //메모리 동적 해제  }  };  BinSrchTree tree; //객체 생성  void switchInput(string order, int key) {  if (order == "I") { //삽입 연산  tree.insert(new BinaryNode(key));  }  else if (order == "D") { //삭제 연산  tree.remove(key);  }  else { return; }  }  int main() {  int n; //작업의 수  cin >> n;  string order; //명령(I, D)  int key; //key값  for (int j = 0; j < n; j++) {  cin >> order >> key;  switchInput(order, key);  }  if (tree.isBalanced() == true) { printf("Balanced"); }  else { printf("Unbalanced");; }  return 0;  } |

-난점: 없음(트리 과제를 수행했다면, 그 코드를 바탕으로 작성하면 쉽게 구성 가능), 다만 문자열 파싱은 필요없음.(getline 같은 함수 안 쓰셔도 됩니다. 어차피 변수 설정하면 변수마다 공백 단위로 입력되기 때문에, 굳이 복잡하게 구현할 필요는 없습니다.) 그리고 iterative하게 구현하는 것도 시간이 좀 걸립니다…

**B. 이진 탐색 트리 – 합 경로 세기(성공)**

-문제

|  |
| --- |
| **[자료구조] 이진 탐색 트리 - 합 경로 세기**    이진 탐색 트리와 숫자 X가 주어졌을 때, 루트 노드부터 시작하는 경로 (path)에서 연속된 노드들이 가지고 있는 숫자들의 합이 X가 되는 모든 경로의 수를 찾는 프로그램을 작성하시오.  예를 들어서 다음과 같은 이진 탐색 트리가 있다고 하자.  10 /      \  7         12  /     \         \     5      8         30  여기에서 X가 22이면 위 트리에서 22인 경로는 10-7-5와 10-12로 2개 존재한다.    **Input**  입력의 첫 줄에는 하고자 하는 작업의 수 n(1 <= n <= 20)와 숫자 X가 주어진다.  다음의 n 줄에는 다음과 같은 작업 명령이 들어온다.  I x : key x를 이진 탐색 트리에 삽입한다. (value는 key와 같다)  D x : key x를 이진 탐색 트리에서 삭제한다. (value는 key와 같다)    **Output**  루트 노드에서 시작하는 경로중 노드가 가지는 숫자의 합이 X가 되는 경로의 개수를 찾아 출력한다.    **Sample Input**  6 22  I 10  I 7  I 12  I 5  I 8  I 30    **Sample Output**  2    **Sample Input**  10 91  I 35  I 18  I 7  I 3  I 26  D 18  I 12  I 30  I 68  I 100  **Sample Output**  1 |
| [자료구조] 리트머스 숙제 - 이진 탐색 트리 - 합 경로 세기   Sample 입출력에 대한 추가 설명    Sample Output1 설명   10   / \   7 12   / \ \   5 8 30  경로 합이 22인 경로는 10-7-5 와 10-12 총 2개가 존재한다.    Sample Output2 설명   35   / \   26 68   / \ \   7 30 100   / \   3 12   경로 합이 91인 경로는 35-26-30 총 1개가 존재한다. |

-코드(C++, 성공)

|  |
| --- |
| #include <cstdio>  #include <cstdlib>  #include <iostream>  #include <stack>  #include <algorithm>  #include <climits> //INT\_MAX, INT\_MIN을 사용하기 위한 헤더  using namespace std;  #define MAX\_QUEUE\_SIZE 100  inline void error(const char\* str) {  fprintf(stderr, "%s\n", str);  exit(1);  }  class BinaryNode {  protected:  int data; //트리에 저장할 데이터  BinaryNode\* left; //왼쪽 자식 노드의 포인터  BinaryNode\* right; //오른쪽 자식 노드의 포인터  public:  BinaryNode(int val = 0, BinaryNode\* l = NULL, BinaryNode\* r = NULL)  : data(val), left(l), right(r) { }  //키 값으로 노드를 탐색하는 함수(노드 클래스에서 순환으로 구현)  BinaryNode\* search(int key) {  if (key == data)  return this;  else if (key < data && left != NULL)  return left->search(key);  else if (key > data && right != NULL)  return right->search(key);  else  return NULL;  }  void setData(int val) { data = val; }  void setLeft(BinaryNode\* l) { left = l; }  void setRight(BinaryNode\* r) { right = r; }  int getData() { return data; } //노드값 반환  BinaryNode\* getLeft() { return left; } //왼쪽 노드값 반환  BinaryNode\* getRight() { return right; } //오른쪽 노드값 반환  bool isLeaf() { return left == NULL && right == NULL; } //잎 노드 여부 확인  };  class CircularQueue {  int front;  int rear;  BinaryNode\* data[MAX\_QUEUE\_SIZE];  public:  CircularQueue() { front = rear = 0; }  bool isEmpty() { return front == rear; }  bool isFull() { return ((rear + 1) % MAX\_QUEUE\_SIZE) == front; }  void enqueue(BinaryNode\* n) {  if (isFull()) { error(" Error: 큐가 포화상태입니다.\n"); }  else {  rear = (rear + 1) % MAX\_QUEUE\_SIZE;  data[rear] = n;  }  }  BinaryNode\* dequeue() {  if (isEmpty()) { error(" Error: 큐가 공백상태입니다.\n"); }  else {  front = (front + 1) % MAX\_QUEUE\_SIZE;  return data[front];  }  }  };  class BinaryTree {  public:  BinaryNode\* root; //루트 포인터  //=====================================================================================  // 자료구조 과제를 위한 함수 중 bool isBalanced() 추가  //=====================================================================================  //균형이 잡혀있는지를 검사하는 함수 - 왼쪽 서브트리와 오른쪽 서브트리의 높이의 차이가 2보다 작은지를 검사  //O(n)의 시간 복잡도를 수행 - 재귀를 없애고, 스택을 이용하여 반복적으로 수행  bool isBalanced() {  if (root == NULL)  return true; // 공백 트리는 균형 잡혀있다고 간주  stack<BinaryNode\*> nodes; //노드를 위한 스택  stack<int> heights; //높이를 위한 스택  nodes.push(root);  heights.push(1); //높이의 초기값은 1  int minimum\_height = INT\_MAX; //무한히 큰 수(설정)  int maximum\_height = INT\_MIN; //무한히 작은 수(설정)  while (!nodes.empty()) {  //스택에서 원소를 뽑아 옴  BinaryNode\* current\_state = nodes.top();  int height = heights.top();  nodes.pop();  heights.pop();  if (current\_state->isLeaf()) {  // 잎 노드인 경우 최소 높이와 최대 높이 갱신  minimum\_height = min(minimum\_height, height);  maximum\_height = max(maximum\_height, height);  //이 과정은 지금 현재의 높이와 현재 기록된 최소/최대 높이 중 최소/최대의 값을 고르는 과정!  }  else {  // 왼쪽 자식 노드가 있는 경우 스택에 추가  if (current\_state->getLeft() != NULL) {  nodes.push(current\_state->getLeft());  heights.push(height + 1); //뽑은 값에서 1 추가  }  // 오른쪽 자식 노드가 있는 경우 스택에 추가  if (current\_state->getRight() != NULL) {  nodes.push(current\_state->getRight());  heights.push(height + 1); //뽑은 값에서 1 추가  }  }  }  return (maximum\_height - minimum\_height) <= 1;  }  //=====================================================================================  // 자료구조 과제를 위한 함수 끝  //=====================================================================================  BinaryTree() : root(NULL) { }  void setRoot(BinaryNode\* node) { root = node; }  BinaryNode\* getRoot() { return root; } //루트 노드값 반환  bool isEmpty() { return root == NULL; } //빈 트리 여부 확인  //이진트리의 순회 연산  void inorder() { printf("\n inorder: "); inorder(root); }  void inorder(BinaryNode\* node) {  //중위 순회: LVR  if (node != NULL) { //루트 노드가 0이 아닐 때  inorder(node->getLeft()); //왼쪽 서브트리  printf(" [%d] ", node->getData()); //루트 노드 처리  inorder(node->getRight()); //오른쪽 서브트리  }  }  void preorder() { printf("\n preorder: "); preorder(root); }  void preorder(BinaryNode\* node) {  //전위 순회: VLR  if (node != NULL) { //루트 노드가 0이 아닐 때  printf(" [%d] ", node->getData()); //루트 노드 처리  preorder(node->getLeft()); //왼쪽 서브트리  preorder(node->getRight()); //오른쪽 서브트리  }  }  void postorder() { printf("\n postorder: "); postorder(root); }  void postorder(BinaryNode\* node) {  //후위 순회: LRV  if (node != NULL) { //루트 노드가 0이 아닐 때  postorder(node->getLeft()); //왼쪽 서브트리  postorder(node->getRight()); //오른쪽 서브트리  printf(" [%d] ", node->getData()); //루트 노드 처리  }  }  //레벨 순회(BFS: 깊이 우선 탐색)  void levelorder() {  printf("\nlevelorder: ");  if (!isEmpty()) {  CircularQueue q;  q.enqueue(root);  //먼저 루트를 큐에 넣는다.  while (!q.isEmpty()) {  BinaryNode\* n = q.dequeue();  //큐가 비어있지 않으면 큐에서 삭제  if (n != NULL) {  //n이 NULL이 아닐 때  printf(" [%d] ", n->getData()); //n을 표시하고  q.enqueue(n->getLeft());//왼쪽 노드를 큐에 넣는다.  q.enqueue(n->getRight());//오른쪽 노드를 큐에 넣는다.  }  //공백 상태가 될 때까지 while문 반복  }  }  printf("\n");  }  //이진트리의 추가 연산  //트리의 노드 개수를 구하는 함수  int getCount() { return isEmpty() ? 0 : getCount(root); }  //순환 호출에 의해 node를 루트로 하는 서브트리의 노드 수 계산 함수  int getCount(BinaryNode\* node) {  if (node == NULL) { return 0; }  return (1 + getCount(node->getLeft()) + getCount(node->getRight()));  }  //트리의 높이를 구하는 함수  int getHeight() { return isEmpty() ? 0 : getHeight(root); }  int getHeight(BinaryNode\* node) {  if (node == NULL) { return 0; }  int hLeft = getHeight(node->getLeft());  int hRight = getHeight(node->getRight());  return ((hLeft > hRight) ? hLeft + 1 : hRight + 1);  //問: 왜 다른 곳과 달리, hLeft, hRight를 쓰는 것일까?  //答: 편의상 하는 것, 굳이 없어도 return만 잘 구현하면 상관 없음  }  //트리의 잎 노드(단말노드) 개수를 구하는 함수  int getLeafCount() { return isEmpty() ? 0 : getLeafCount(root); }  //순환 호출에 의해 node를 루트로 하는 서브트리의 단말 노드 수 계산 함수  int getLeafCount(BinaryNode\* node) {  if (node == NULL) { return 0; }  if (node->isLeaf()) { return 1; } //잎 노드!  else return (getLeafCount(node->getLeft()) + getLeafCount(node->getRight()));  }  };  class BinSrchTree : public BinaryTree {  public:  BinSrchTree(void) { }  ~BinSrchTree(void) { }  //이진 탐색 트리의 탐색 연산  BinaryNode\* search(int key) {  BinaryNode\* node = searchRecur(root, key);  if (node != NULL) {  printf("탐색 성공: 키값이 %d인 노드 = 0x%x\n", node->getData(), node);  }  else {  printf("키값이 %d인 노드 없음\n", key);  }  return node;  }  //키 값으로 노드를 탐색하는 함수(순환적 방법)  //일반 함수로 구현(BinSrchTree의 멤버 함수로 넣어도 됨)  BinaryNode\* searchRecur(BinaryNode\* n, int key) {  if (n == NULL) { return NULL; } //못 찾았다.  if (key == n->getData()) { return n; }  else if (key < n->getData()) { return searchRecur(n->getLeft(), key); }  else return searchRecur(n->getRight(), key);  }  //키 값으로 노드를 탐색하는 함수(반복적인 방법)  //일반 함수로 구현(BinSrchTree의 멤버 함수로 넣어도 됨)  BinaryNode\* SearchIter(BinaryNode\* n, int key) {  while (n != NULL) {  if (key == n->getData()) { return n; }  else if (key < n->getData()) { n = n->getLeft(); }  else { n = n->getRight(); }  }  return n;  }  //=============================================================  //이진 탐색 트리의 삽입 연산  void insert(BinaryNode\* n) {  if (n == NULL) { return; }  if (isEmpty()) { root = n; }  else { insertRucur(root, n); }  }  //이진 탐색 트리의 삼입 함수  void insertRucur(BinaryNode\* r, BinaryNode\* n) {  //root와 key가 같으면 return  if (n->getData() == r->getData()) { return; }  //root보다 key가 작을 때  else if (n->getData() < r->getData()) {  if (r->getLeft() == NULL) { r->setLeft(n); } //root의 왼쪽 자식이 없으면 n이 왼쪽 자식  else { insertRucur(r->getLeft(), n); } //root에 왼쪽 자식이 있으면 순환 호출  }  //root보다 key가 클 때  else {  if (r->getRight() == NULL) { r->setRight(n); } //root의 오른쪽 자식이 없으면 n이 오른쪽 자식  else { insertRucur(r->getRight(), n); } //root에 오른쪽 자식이 있으면 순환 호출  }  }  //=============================================================  //이진 탐색 트리의 삭제 연산  void remove(int key) {  if (isEmpty()) { return; } //빈 트리이면 return  //없앨 노드와 그 노드의 부모 노드를 찾는다.  BinaryNode\* parent = NULL; //부모 노드  BinaryNode\* node = root; //루트 노드  while (node != NULL && node->getData() != key) {  parent = node;  node = (key < node->getData()) ? node->getLeft() : node->getRight();  }  //없앨 노드가 트리에 없음  if (node == NULL) {  printf("Error: 키가 트리에 없습니다.\n");  return;  }  //없앨 노드가 트리에 있음  else { remove(parent, node); }  }  void remove(BinaryNode\* parent, BinaryNode\* node) {  //case 1: 삭제하려는 노드가 단말 노드인 경우 ->단말 노드의 링크를 끊는다.  if (node->isLeaf()) {  if (parent == NULL) { root = NULL; } //node == root이면 루트만 있는 상태  else { //node != root인 경우  if (parent->getLeft() == node) {  parent->setLeft(NULL);  }  else {  parent->setRight(NULL);  }  }  }  //case 2: 삭제하려는 노드가 왼쪽이나 오른쪽 자식만 갖는 경우  else if (node->getLeft() == NULL || node->getRight() == NULL) {  //삭제할 노드의 유일한 자식 노드 => child  BinaryNode\* child = (node->getLeft() != NULL) ?  node->getLeft() : node->getRight();  //삭제할 노드가 루트이면 ==> child가 새로운 root가 됨  if (node == root) { root = child; }  else {  if (parent->getLeft() == node) {  parent->setLeft(child);  }  else {  parent->setRight(child);  }  }  }  //case 3: 삭제하려는 노드가 두 개의 자식이 모두 있는 경우  else {  //삭제하려는 노드의 오른쪽 서브트리에서 가장 작은 노드를 탐색  //succp => 후계 노드의 부모 노드  //succ => 후계 노드: 오른쪽 서브트리에서 가장 key가 작은 노드  BinaryNode\* succp = node;  BinaryNode\* succ = node->getRight(); //왼쪽부터 수행시 getLeft()  while (succ->getLeft() != NULL) { //후계 노드 탐색  succp = succ; //후계 노드의 부모 노드  succ = succ->getLeft(); //후계 노드  }  //후계 노드의 부모와 후계 노드의 오른쪽 자식을 직접 연결  if (succp->getLeft() == succ) {  succp->setLeft(succ->getRight());  }  else { //후계 노드가 삭제할 노드의 바로 오른쪽 자식인 경우  succp->setRight(succ->getRight());  //후계 노드 정보를 삭제할 노드에 복사  }  node->setData(succ->getData());  //삭제할 노드를 후계 노드로 변경: 실제로는 후계 노드가 제거됨  node = succ;  }  delete node; //메모리 동적 해제  }  };  BinSrchTree tree; //객체 생성  void switchInput(string order, int key) {  if (order == "I") { //삽입 연산  tree.insert(new BinaryNode(key));  }  else if (order == "D") { //삭제 연산  tree.remove(key);  }  else { return; }  }  int path\_sum\_from\_node(BinaryNode\* node, int X) {  int count = 0;    //Base case  if (node == NULL) { return 0; }  if (node->getData() == X) { count++; }  //Inductive step  count += path\_sum\_from\_node(node->getLeft(), X - node->getData());  count += path\_sum\_from\_node(node->getRight(), X - node->getData());  return count;  }  int path\_sum(BinaryNode\* node, int X) {  int count = 0;  //Base case  if (node == NULL) { return 0; }  count += path\_sum\_from\_node(node, X); // 현재 노드에서 시작하는 경로의 수  //Inductive case - 혹시나 해서 만든 부분  // 왼쪽 서브트리에서 시작하는 경로의 수  count += path\_sum(node->getLeft(), X);  // 오른쪽 서브트리에서 시작하는 경로의 수  count += path\_sum(node->getRight(), X);  return count;  }  int main() {  int n; //작업의 수  int X; //숫자(겅로의 합)  cin >> n >> X;  string order; //명령(I, D)  int key; //key값  for (int j = 0; j < n; j++) {  cin >> order >> key;  switchInput(order, key);  }  cout << path\_sum(tree.getRoot(), X);  return 0;  } |

-난점: 딱히 없음(다만, 프로그래밍 프로젝트에는 없는 문제여서, 인터넷에서 파이썬 코드를 공부하고 C++로 변환해야 한다는 문제점이 존재.)

**C. 이진탐색트리 – 한영/영한 사전 검색(성공)**

-문제

|  |
| --- |
| 이진탐색트리 – 한영/영한 사전 검색   한국어 단어 검색과 영어 단어 검색이 가능한 한-영 사전과 영-한 사전을 이진탐색트리로 구현하시오.  -      한영(K-E) 사전: 한국어 단어가 키(key)가 되고 영어 단어가 값(value)이 됨  -      영한(E-K) 사전: 영어 단어가 키(key)가 되고 한국어 단어가 값(value)이 됨  단, 키가 중복되는 경우는 없다. 한국어와 영어단어에 공백이 포함될 수 있다.    한영사전과 영한사전의 각 기능의 설명은 다음과 같다.   * 입력(i): <한국어 영어> 순서로 입력 받아서 2개의 사전트리에 삽입함. 한국어가 한 줄(line)로 입력되고, 영어가 한줄로 입력됨 * 한국어 단어 검색(k): 검색할 한국어 단어를 입력받아 K-E 사전에서 해당 영어 번역어를 출력함. 만약 사전에 없는 단어를 입력받았을 시 <한국어단어 UNKNOWN ENTRY>를 출력함 * 영어 단어 검색(e): 검색할 영어 단어를 입력받아 E-K 사전에서 해당 한국어 번역어를 출력함. 만약 사전에 없는 단어를 입력받았을시 <영어단어 UNKNOWN ENTRY>를 출력함 * 출력(p): K-E사전을 먼저 출력하고, E-K사전을 출력함. 이때, K-E사전은 한국어 단어(key) 순서로 출력, E-K사전은 영어 단어(key) 알파벳 순서로 출력함. 각 사전의 시작부분에 “K-E dictionary:”와 “E-K dictionary:”을 출력하고 내용을 출력한다. * 종료(q): 프로그램을 종료함     **Input**  각 메뉴와 그에 따른 입력을 받는다. 종료될 때까지 반복한다. 각 문자들의 길이는 100을 넘지 않는다.      **Ouput**  출력 커맨드를 입력 받을 때마다 해당 내용들을 출력한다    **Example:**  Input  i             // 사전 엔트리 입력  책        // 한국어 단어  book     // 영어 단어  i          // 사전 엔트리 입력  사과  apple  k            // 한영사전 검색  책  e            // 영한사전 검색  apple  i  흑해  black sea  p  q    Output  책 book            // “k 책”의 출력  apple 사과             // “e apple”의 출력  K-E dictionary:       // p  사과 apple          // K-E사전 내용 출력  책 book  흑해 black sea  E-K dictionary:  apple 사과             // E-K사전 내용 출력  black sea 흑해  book 책    **Example:**  Input  i             // 사전 엔트리 입력  책        // 한국어 단어  book     // 영어 단어  k        // 한영 단어 사전 검색  노트북  q      // 종료    Output  노트북 UNKNOWN ENTRY   // k "노트북" 에 대한 출력 |

-코드 ver.1(C) – 성공 코드(최종 제출본)

|  |
| --- |
| // 이진 탐색 트리를 사용한 영어 사전  #define \_CRT\_SECURE\_NO\_WARNINGS  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <memory.h>  #define MAX\_WORD\_SIZE 100  #define MAX\_MEANING\_SIZE 200  // 데이터 형식  typedef struct {  char word[MAX\_WORD\_SIZE]; // 키필드  char meaning[MAX\_MEANING\_SIZE];  } element;  // 노드의 구조  typedef struct TreeNode {  element key;  struct TreeNode\* left, \* right;  } TreeNode;  // 만약 e1 < e2 이면 -1 반환  // 만약 e1 == e2 이면 0 반환  // 만약 e1 > e2 이면 1 반환  int compare(element e1, element e2) {  int result = strcmp(e1.word, e2.word);  if (result < 0)  return -1;  else if (result > 0)  return 1;  else  return 0;  }  // 이진 탐색 트리 탐색 함수(더미 데이터)  TreeNode\* search(TreeNode\* root, element key) {  TreeNode\* p = root;  while (p != NULL) {  int cmp = compare(key, p->key);  if (cmp == 0)  return p;  else if (cmp < 0)  p = p->left;  else if (cmp > 0)  p = p->right;  }  return NULL; // 탐색에 실패했을 경우 NULL 반환  }  TreeNode\* new\_node(element item) {  TreeNode\* temp = (TreeNode\*)malloc(sizeof(TreeNode));  temp->key = item;  temp->left = NULL;  temp->right = NULL;  return temp;  }  TreeNode\* insert\_node(TreeNode\* node, element key) {  // 트리가 공백이면 새로운 노드를 반환한다.  if (node == NULL) return new\_node(key);  // 그렇지 않으면 순환적으로 트리를 내려간다.  if (compare(key, node->key) < 0)  node->left = insert\_node(node->left, key);  else if (compare(key, node->key) > 0)  node->right = insert\_node(node->right, key);  // 루트 포인터를 반환한다.  return node;  }  // 이진 탐색 트리 탐색 함수(단어 검색)  TreeNode\* search\_binary\_tree(TreeNode\* root, char\* word) {  TreeNode\* p = root;  while (p != NULL) {  int cmp = strcmp(word, p->key.word);  if (cmp == 0)  return p;  else if (cmp < 0)  p = p->left;  else if (cmp > 0)  p = p->right;  }  return NULL; // 탐색에 실패했을 경우 NULL 반환  }  // 이진 탐색 트리 출력 함수  void show\_word(TreeNode\* p) {  if (p != NULL) {  show\_word(p->left);  printf("%s %s\n", p->key.word, p->key.meaning);  show\_word(p->right);  }  }  void help() {  //printf("\*\*\*\* i: 입력, k: 한국어 단어 검색, e: 영어 단어 검색, p: 출력, q: 종료 \*\*\*\*: ");  }  // 이진 탐색 트리를 사용하는 영어 사전 프로그램  int main(void)  {  char command;  char list[MAX\_WORD\_SIZE]; //명령 리스트 확인  int count = 0; //카운트 변수    element e; //한-영 사전 구조체  element e2; //영한-사전 구조체  TreeNode\* root = NULL; //한-영 사전 저장 트리  TreeNode\* root2 = NULL; //영-한 사전 저장 트리  TreeNode\* tmp = NULL;    TreeNode\* korean\_copy[MAX\_WORD\_SIZE];  TreeNode\* english\_copy[MAX\_WORD\_SIZE];  int korean\_count = 0; //한국어 트리 배열 카운트  int english\_count = 0; //영어 트리 배열 카운트  char\* result\_korean[MAX\_WORD\_SIZE];  char\* result\_english[MAX\_WORD\_SIZE];  do {  help();  command = getchar();  getchar(); // 엔터키 제거  switch (command) {  case 'i': // 추가  //printf("한국어 단어:");  gets(e.word);  //printf("영어 단어:");  gets(e.meaning);  strcpy(e2.meaning, e.word);  strcpy(e2.word, e.meaning);  root = insert\_node(root, e);  root2 = insert\_node(root2, e2);  break;  case 'p': //출력  list[count] = 'p';  count++;  //printf("K-E dictionary:\n");  //show\_korean(root);  //printf("E-K dictionary:\n");  //show\_english(root);  break;  case 'k': //한국어 단어 검색  list[count] = 'k';  count++;  //printf("한국어 단어:");  gets(e.word);  tmp = search\_binary\_tree(root, e.word);  korean\_copy[korean\_count] = tmp;    if (tmp != NULL) {  //printf("%s %s\n", tmp->key.word, tmp->key.meaning);  }  else {  result\_korean[korean\_count] = e.word;  //printf("%s UNKNOWN ENTRY\n", result\_korean[korean\_count]);  }  korean\_count++;  break;  case 'e': //영어 단어 검색  list[count] = 'e';  count++;  //printf("영어 단어:");  gets(e2.word);  tmp = search\_binary\_tree(root2, e2.word);  english\_copy[english\_count] = tmp;  if (tmp != NULL) {  //printf("%s %s\n", tmp->key.meaning, tmp->key.word);  }  else {  result\_english[english\_count] = e2.word;  //printf("%s UNKNOWN ENTRY\n", result\_english[english\_count]);  }  english\_count++;  break;  }  } while (command != 'q'); //q: 종료  //출력  if (command == 'q') {  for (int a = 0; a < count; a++) {  if (list[a] == 'k') {  //printf("token for k\n");  for (int b = 0; b < korean\_count; b++) {  if (korean\_copy[b] != NULL) {  printf("%s %s\n", korean\_copy[b]->key.word, korean\_copy[b]->key.meaning);  }  else {  printf("%s UNKNOWN ENTRY\n", result\_korean[b]);  }  }    }  else if (list[a] == 'e') {  //printf("token for e\n");  for (int c = 0; c < english\_count; c++) {  if (english\_copy[c] != NULL) {  printf("%s %s\n", english\_copy[c]->key.word, english\_copy[c]->key.meaning);  }  else {  printf("%s UNKNOWN ENTRY\n", result\_english[c]);  }  }  }  else if (list[a] == 'p') {  if (root != NULL) {  printf("K-E dictionary:\n");  show\_word(root);  printf("E-K dictionary:\n");  show\_word(root2);  }  }  else { continue; }  }  }  return 0;  } |

-코드(C++, 컴파일 에러)

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS //strcpy 때문에 있어야 합니다.  #include <cstdio>  #include <cstdlib>  #include <cstring>  #include <iostream>  #include <conio.h> //getche()를 쓰기 위한 헤더  #include <iostream>  using namespace std;  #define MAX\_WORD\_SIZE 40  #define MAX\_MEANING\_SIZE 200  #define MAX\_QUEUE\_SIZE 100  //사전을 위한 레코드 클래스  class Record {  public:  char word[MAX\_WORD\_SIZE];  char meaning[MAX\_MEANING\_SIZE];  Record(const char\* w = "", const char\* m = "") { set(w, m); } //""이 들어가니 변수 유형 오류 발생!  void set(const char\* w, const char\* m) {  strcpy(word, w);  strcpy(meaning, m);  }  //현재 레코드의 word 필드와 문자열 w 비교  int compare(Record\* n) { return compare(n->word); }  int compare(char\* w) { return strcmp(w, word); }  //의미를 이용한 검색  int compareMeaning(char\* m) { return strcmp(meaning, m); }  //출력: "단어 : 의미"  void display() {  printf("%s %s\n", word, meaning);  }  //레코드 n의 내용을 자신에 복사  void copy(Record\* n) { set(n->word, n->meaning); }  };  class BinaryNode : public Record {  BinaryNode\* left;  BinaryNode\* right;  public:  BinaryNode(const char\* w, const char\* m) : Record(w, m), left(NULL), right(NULL) {}  //왼쪽 자식, 오른쪽 자식 설정  void setData(char\* val) { strcpy(word, val); } //빠진 부분 추가!  void setLeft(BinaryNode\* l) { left = l; }  void setRight(BinaryNode\* r) { right = r; }  //왼쪽 자식, 오른쪽 자식 반환  char\* getData() { return word; } //노드값 반환, 빠진 부분 추가!  char\* getMeaning() { return meaning; }  BinaryNode\* getLeft() { return left; }  BinaryNode\* getRight() { return right; }  //잎 노드(단말 노드) 반환: 왼쪽 자식과 오른쪽 자식 모두 없을 때 true  bool isLeaf() { return (left == NULL) && (right == NULL); }  };  class CircularQueue {  int front = 0;  int rear = 0;  BinaryNode\* data[MAX\_QUEUE\_SIZE] = { };  public:  CircularQueue() { front = rear = 0; }  bool isEmpty() { return front == rear; }  bool isFull() { return ((rear + 1) % MAX\_QUEUE\_SIZE) == front; }  void enqueue(BinaryNode\* n) {  if (isFull()) { printf(" Error: 큐가 포화상태입니다.\n"); }  else {  rear = (rear + 1) % MAX\_QUEUE\_SIZE;  data[rear] = n;  }  }  BinaryNode\* dequeue() {  if (isEmpty()) { printf(" Error: 큐가 공백상태입니다.\n"); }  else {  front = (front + 1) % MAX\_QUEUE\_SIZE;  return data[front];  }  }  };  class BinaryTree {  protected:  BinaryNode\* root;  public:  BinaryTree() : root(NULL) { }  BinaryNode\* getRoot() { return root; }  bool isEmpty() { return root == NULL; } //루트만 있는 트리인지의 여부 확인  //이진트리의 순회 연산  void inorder(BinaryNode\* node) {  //중위 순회: LVR  if (node != NULL) { //루트 노드가 0이 아닐 때  inorder(node->getLeft()); //왼쪽 서브트리  node->display(); //루트 노드 처리  inorder(node->getRight()); //오른쪽 서브트리  }  }  };  class BinSrchTree : public BinaryTree {  public:  //이진 탐색 트리의 탐색 연산  BinaryNode\* search(char\* key) { return searchRecur(root, key); }  //키 값으로 노드를 탐색하는 함수(순환적 방법)  //일반 함수로 구현(BinSrchTree의 멤버 함수로 넣어도 됨)  BinaryNode\* searchRecur(BinaryNode\* n, char\* key) {  if (n == NULL) { return NULL; }  if (n->compare(key) == 0) { return n; }  else if (n->compare(key) < 0) { return searchRecur(n->getLeft(), key); }  else return searchRecur(n->getRight(), key);  }    //루트 노드가 있는가의 여부  bool hasData() { return getRoot(); }  //이진 탐색 트리의 삽입 연산  void insert(BinaryNode\* n) {  if (n == NULL) { return; }  if (isEmpty()) { root = n; }  else { insertRucur(root, n); }  }  //이진 탐색 트리의 삼입 함수  void insertRucur(BinaryNode\* r, BinaryNode\* n) {  //root와 key가 같으면 return  if (n->compare(r) == 0) { return; }  //root보다 key가 작을 때  else if (n->compare(r) > 0) {  if (r->getLeft() == NULL) { r->setLeft(n); } //root의 왼쪽 자식이 없으면 n이 왼쪽 자식  else { insertRucur(r->getLeft(), n); } //root에 왼쪽 자식이 있으면 순환 호출  }  //root보다 key가 클 때  else {  if (r->getRight() == NULL) { r->setRight(n); } //root의 오른쪽 자식이 없으면 n이 오른쪽 자식  else { insertRucur(r->getRight(), n); } //root에 오른쪽 자식이 있으면 순환 호출  }  }  //=============================================================  //이진 탐색 트리의 삭제 연산  void remove(char\* key) {  if (isEmpty()) { return; } //빈 트리이면 return  //없앨 노드와 그 노드의 부모 노드를 찾는다.  BinaryNode\* parent = NULL;  BinaryNode\* node = root;  while (node != NULL && node->compare(key) != 0) {  parent = node;  node = (node->compare(key) < 0) ? node->getLeft() : node->getRight();  }  //없앨 노드가 트리에 없음  if (node == NULL) {  printf("Error: 키가 트리에 없습니다.\n");  return;  }  //없앨 노드가 트리에 있음  else { remove(parent, node); }  }  void remove(BinaryNode\* parent, BinaryNode\* node) {  //case 1: 삭제하려는 노드가 단말 노드인 경우 ->단말 노드의 링크를 끊는다.  if (node->isLeaf()) {  if (parent == NULL) { root = NULL; } //node == root이면 공백상태  else { //node != root인 경우  if (parent->getLeft() == node) {  parent->setLeft(NULL);  }  else {  parent->setRight(NULL);  }  }  }  //case 2: 삭제하려는 노드가 왼쪽이나 오른쪽 자식만 갖는 경우  else if (node->getLeft() == NULL || node->getRight() == NULL) {  //삭제할 노드의 유일한 자식 노드 => child  BinaryNode\* child = (node->getLeft() != NULL) ?  node->getLeft() : node->getRight();  //삭제할 노드가 루트이면 ==> child가 새로운 root가 됨  if (node == root) { root = child; }  else {  if (parent->getLeft() == node) {  parent->setLeft(child);  }  else {  parent->setRight(child);  }  }  }  //case 3: 삭제하려는 노드가 두 개의 자식이 모두 있는 경우  else {  //삭제하려는 노드의 오른쪽 서브트리에서 가장 작은 노드를 탐색  //succp => 후계 노드의 부모 노드  //succ => 후계 노드: 오른쪽 서브트리에서 가장 key가 작은 노드  BinaryNode\* succp = node;  BinaryNode\* succ = node->getRight();  while (succ->getLeft() != NULL) { //후계 노드 탐색  succp = succ; //후계 노드의 부모 노드  succ = succ->getLeft(); //후계 노드  }  //후계 노드의 부모와 후계 노드의 오른쪽 자식을 직접 연결  if (succp->getLeft() == succ) {  succp->setLeft(succ->getRight());  }  else { //후계 노드가 삭제할 노드의 바로 오른쪽 자식인 경우  succp->setRight(succ->getRight());  //후계 노드 정보를 삭제할 노드에 복사  }  node->setData(succ->getData());  //삭제할 노드를 후계 노드로 변경: 실제로는 후계 노드가 제거됨  //word만 반환해도 괜찮을까? -> 일단 나중에 살펴보자...ㅠㅠ  node = succ;  }  delete node; //메모리 동적 해제  }  };  class Dictionary : public BinSrchTree {  public:  //전위 순회하면서, 각 노드의 단어와 의미 출력  void printAllWords() {  //printf(" >> 나의 단어장:\n");  if (!isEmpty()) { inorder(root); }  }  //단어에 의한 검색 연산  BinaryNode\* searchWord(char\* word) {  BinaryNode\* node = search(word);  if (node != NULL) {  return node;  //printf(" >> ");  //node->display();  }  else {  //printf(" >> 등록되지 않은 의미: %s\n", word);  return NULL;  }  return NULL;  }  };  void help() {  //printf("[사용법] i-추가, k-한국어 단어 검색, e-영어 단어 검색, p-출력, q-종료 =>");  }  int main() {  ios::sync\_with\_stdio(false);  cin.tie(NULL);  cout.tie(NULL);  char\* result\_korean[MAX\_WORD\_SIZE];  char\* result\_english[MAX\_WORD\_SIZE];  int korean\_count = 0; //한국어 트리 배열 카운트  int english\_count = 0; //영어 트리 배열 카운트  char list[MAX\_WORD\_SIZE]; //명령 리스트 확인  int count = 0; //카운트 변수  char command;  char word[80]; //한영사전 한국어 단어  char meaning[200]; //한영사전 영어 단어  Dictionary tree; //한영사전  char word2[80]; //영한사전 영어 단어  char meaning2[200]; //영한사전 한국어 단어  Dictionary tree2; //영한사전  BinaryNode\* tmp = NULL; //값을 나르는 임시 변수(한영사전)  BinaryNode\* tmp2 = NULL; //값을 나르는 임시 변수(영한사전)  BinaryNode\* korean\_copy[MAX\_WORD\_SIZE];  BinaryNode\* english\_copy[MAX\_WORD\_SIZE];  do {  help();  cin >> command;  while (getchar() != '\n');  //command = \_getche(); //키보드를 치는 것을 감지하여 엔터키 없이 바로 한 키보드 문자 입력, \_getche()로 해야 빌드 오류 X  //printf("\n");  switch (command) {  case 'i':  //printf(" > 한국어 단어: ");  cin.getline(word, 80, '\n');//구형 gets 사용이 VS 2022에서 아예 인식 X  strcpy(meaning2, word);  //printf(" > 영어 단어: ");  cin.getline(meaning, 200, '\n');  strcpy(word2, meaning);  tree.insert(new BinaryNode(word, meaning));  tree2.insert(new BinaryNode(word2, meaning2));  break;  case 'k':  //printf(" > 검색 단어: ");  cin.getline(word, 80, '\n');  tmp = tree.searchWord(word);  korean\_copy[korean\_count] = tmp;  list[count] = 'k';  count++;  if (tmp != NULL) {  //printf("%s %s\n", tmp->getData(), tmp->getData());  }  else {  result\_korean[korean\_count] = word;  //printf("%s UNKNOWN ENTRY\n", result\_korean[korean\_count]);  }  korean\_count++;  break;  case 'e':  //printf(" > 검색 단어: ");  cin.getline(word2, 80, '\n');  tmp2 = tree2.searchWord(word2);  english\_copy[english\_count] = tmp2;  list[count] = 'e';  count++;  if (tmp2 != NULL) {  //printf("%s %s\n", tmp->key.meaning, tmp->key.word);  }  else {  result\_english[english\_count] = word2;  //printf("%s UNKNOWN ENTRY\n", result\_english[english\_count]);  }  english\_count++;  break;  case 'p':  //printf("K-E Dictionary");  //tree.printAllWords();  //printf("K-E Dictionary");  //tree2.printAllWords();  //printf("\n");  list[count] = 'p';  count++;  break;  }  } while (command != 'q');  if (command == 'q') {  for (int a = 0; a < count; a++) {  if (list[a] == 'k') {  //printf("token for k\n");  for (int b = 0; b < korean\_count; b++) {  if (korean\_copy[b] != NULL) {  printf("%s %s\n", korean\_copy[b]->getData(), korean\_copy[b]->getMeaning());  }  else {  printf("%s UNKNOWN ENTRY\n", result\_korean[b]);  }  }  }  else if (list[a] == 'e') {  //printf("token for e\n");  for (int c = 0; c < english\_count; c++) {  if (english\_copy[c] != NULL) {  printf("%s %s\n", english\_copy[c]->getData(), english\_copy[c]->getMeaning());  }  else {  printf("%s UNKNOWN ENTRY\n", result\_english[c]);  }  }  }  else if (list[a] == 'p') {  if (tree.hasData() == true && tree2.hasData()==true) {  printf("K-E dictionary:\n");  tree.printAllWords();  printf("E-K dictionary:\n");  tree2.printAllWords();  }  }  else { continue; }  }  }    return 0;  } |

-코드 ver.2(C++, 컴파일 에러, 최종 제출 X)

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS //strcpy 때문에 있어야 합니다.  #include <cstdio>  #include <cstdlib>  #include <cstring>  #include <iostream>  #include <conio.h> //getche()를 쓰기 위한 헤더  #include <iostream>  using namespace std;  #define MAX\_WORD\_SIZE 40  #define MAX\_MEANING\_SIZE 200  #define MAX\_QUEUE\_SIZE 100  //사전을 위한 레코드 클래스  class Record {  public:  char word[MAX\_WORD\_SIZE];  char meaning[MAX\_MEANING\_SIZE];  Record(const char\* w = "", const char\* m = "") { set(w, m); } //""이 들어가니 변수 유형 오류 발생!  void set(const char\* w, const char\* m) {  strcpy(word, w);  strcpy(meaning, m);  }  //현재 레코드의 word 필드와 문자열 w 비교  int compare(Record\* n) { return compare(n->word); }  int compare(char\* w) { return strcmp(w, word); }  //의미를 이용한 검색  int compareMeaning(char\* m) { return strcmp(meaning, m); }  //출력: "단어 : 의미"  void display() {  printf("%s %s\n", word, meaning);  }  //레코드 n의 내용을 자신에 복사  void copy(Record\* n) { set(n->word, n->meaning); }  };  class BinaryNode : public Record {  BinaryNode\* left;  BinaryNode\* right;  public:  BinaryNode(const char\* w, const char\* m) : Record(w, m), left(NULL), right(NULL) {}  //왼쪽 자식, 오른쪽 자식 설정  void setData(char\* val) { strcpy(word, val); } //빠진 부분 추가!  void setLeft(BinaryNode\* l) { left = l; }  void setRight(BinaryNode\* r) { right = r; }  //왼쪽 자식, 오른쪽 자식 반환  char\* getData() { return word; } //노드값 반환, 빠진 부분 추가!  char\* getMeaning() { return meaning; }  BinaryNode\* getLeft() { return left; }  BinaryNode\* getRight() { return right; }  //잎 노드(단말 노드) 반환: 왼쪽 자식과 오른쪽 자식 모두 없을 때 true  bool isLeaf() { return (left == NULL) && (right == NULL); }  };  class CircularQueue {  int front = 0;  int rear = 0;  BinaryNode\* data[MAX\_QUEUE\_SIZE] = { };  public:  CircularQueue() { front = rear = 0; }  bool isEmpty() { return front == rear; }  bool isFull() { return ((rear + 1) % MAX\_QUEUE\_SIZE) == front; }  void enqueue(BinaryNode\* n) {  if (isFull()) { printf(" Error: 큐가 포화상태입니다.\n"); }  else {  rear = (rear + 1) % MAX\_QUEUE\_SIZE;  data[rear] = n;  }  }  BinaryNode\* dequeue() {  if (isEmpty()) { printf(" Error: 큐가 공백상태입니다.\n"); }  else {  front = (front + 1) % MAX\_QUEUE\_SIZE;  return data[front];  }  }  };  class BinaryTree {  protected:  BinaryNode\* root;  public:  BinaryTree() : root(NULL) { }  BinaryNode\* getRoot() { return root; }  bool isEmpty() { return root == NULL; } //루트만 있는 트리인지의 여부 확인  //이진트리의 순회 연산  void inorder(BinaryNode\* node) {  //중위 순회: LVR  if (node != NULL) { //루트 노드가 0이 아닐 때  inorder(node->getLeft()); //왼쪽 서브트리  node->display(); //루트 노드 처리  inorder(node->getRight()); //오른쪽 서브트리  }  }  };  class BinSrchTree : public BinaryTree {  public:  //이진 탐색 트리의 탐색 연산  BinaryNode\* search(char\* key) { return searchRecur(root, key); }  //키 값으로 노드를 탐색하는 함수(순환적 방법)  //일반 함수로 구현(BinSrchTree의 멤버 함수로 넣어도 됨)  BinaryNode\* searchRecur(BinaryNode\* n, char\* key) {  if (n == NULL) { return NULL; }  if (n->compare(key) == 0) { return n; }  else if (n->compare(key) < 0) { return searchRecur(n->getLeft(), key); }  else return searchRecur(n->getRight(), key);  }    //루트 노드가 있는가의 여부  bool hasData() { return getRoot(); }  //이진 탐색 트리의 삽입 연산  void insert(BinaryNode\* n) {  if (n == NULL) { return; }  if (isEmpty()) { root = n; }  else { insertRucur(root, n); }  }  //이진 탐색 트리의 삼입 함수  void insertRucur(BinaryNode\* r, BinaryNode\* n) {  //root와 key가 같으면 return  if (n->compare(r) == 0) { return; }  //root보다 key가 작을 때  else if (n->compare(r) > 0) {  if (r->getLeft() == NULL) { r->setLeft(n); } //root의 왼쪽 자식이 없으면 n이 왼쪽 자식  else { insertRucur(r->getLeft(), n); } //root에 왼쪽 자식이 있으면 순환 호출  }  //root보다 key가 클 때  else {  if (r->getRight() == NULL) { r->setRight(n); } //root의 오른쪽 자식이 없으면 n이 오른쪽 자식  else { insertRucur(r->getRight(), n); } //root에 오른쪽 자식이 있으면 순환 호출  }  }  //=============================================================  //이진 탐색 트리의 삭제 연산  void remove(char\* key) {  if (isEmpty()) { return; } //빈 트리이면 return  //없앨 노드와 그 노드의 부모 노드를 찾는다.  BinaryNode\* parent = NULL;  BinaryNode\* node = root;  while (node != NULL && node->compare(key) != 0) {  parent = node;  node = (node->compare(key) < 0) ? node->getLeft() : node->getRight();  }  //없앨 노드가 트리에 없음  if (node == NULL) {  printf("Error: 키가 트리에 없습니다.\n");  return;  }  //없앨 노드가 트리에 있음  else { remove(parent, node); }  }  void remove(BinaryNode\* parent, BinaryNode\* node) {  //case 1: 삭제하려는 노드가 단말 노드인 경우 ->단말 노드의 링크를 끊는다.  if (node->isLeaf()) {  if (parent == NULL) { root = NULL; } //node == root이면 공백상태  else { //node != root인 경우  if (parent->getLeft() == node) {  parent->setLeft(NULL);  }  else {  parent->setRight(NULL);  }  }  }  //case 2: 삭제하려는 노드가 왼쪽이나 오른쪽 자식만 갖는 경우  else if (node->getLeft() == NULL || node->getRight() == NULL) {  //삭제할 노드의 유일한 자식 노드 => child  BinaryNode\* child = (node->getLeft() != NULL) ?  node->getLeft() : node->getRight();  //삭제할 노드가 루트이면 ==> child가 새로운 root가 됨  if (node == root) { root = child; }  else {  if (parent->getLeft() == node) {  parent->setLeft(child);  }  else {  parent->setRight(child);  }  }  }  //case 3: 삭제하려는 노드가 두 개의 자식이 모두 있는 경우  else {  //삭제하려는 노드의 오른쪽 서브트리에서 가장 작은 노드를 탐색  //succp => 후계 노드의 부모 노드  //succ => 후계 노드: 오른쪽 서브트리에서 가장 key가 작은 노드  BinaryNode\* succp = node;  BinaryNode\* succ = node->getRight();  while (succ->getLeft() != NULL) { //후계 노드 탐색  succp = succ; //후계 노드의 부모 노드  succ = succ->getLeft(); //후계 노드  }  //후계 노드의 부모와 후계 노드의 오른쪽 자식을 직접 연결  if (succp->getLeft() == succ) {  succp->setLeft(succ->getRight());  }  else { //후계 노드가 삭제할 노드의 바로 오른쪽 자식인 경우  succp->setRight(succ->getRight());  //후계 노드 정보를 삭제할 노드에 복사  }  node->setData(succ->getData());  //삭제할 노드를 후계 노드로 변경: 실제로는 후계 노드가 제거됨  //word만 반환해도 괜찮을까? -> 일단 나중에 살펴보자...ㅠㅠ  node = succ;  }  delete node; //메모리 동적 해제  }  };  class Dictionary : public BinSrchTree {  public:  //전위 순회하면서, 각 노드의 단어와 의미 출력  void printAllWords() {  //printf(" >> 나의 단어장:\n");  if (!isEmpty()) { inorder(root); }  }  //단어에 의한 검색 연산  BinaryNode\* searchWord(char\* word) {  BinaryNode\* node = search(word);  if (node != NULL) {  return node;  //printf(" >> ");  //node->display();  }  else {  //printf(" >> 등록되지 않은 의미: %s\n", word);  return NULL;  }  return NULL;  }  };  void help() {  //printf("[사용법] i-추가, k-한국어 단어 검색, e-영어 단어 검색, p-출력, q-종료 =>");  }  int main() {  ios::sync\_with\_stdio(false);  cin.tie(NULL);  cout.tie(NULL);  char\* result\_korean[MAX\_WORD\_SIZE];  char\* result\_english[MAX\_WORD\_SIZE];  int korean\_count = 0; //한국어 트리 배열 카운트  int english\_count = 0; //영어 트리 배열 카운트  char list[MAX\_WORD\_SIZE]; //명령 리스트 확인  int count = 0; //카운트 변수  char command;  char word[80]; //한영사전 한국어 단어  char meaning[200]; //한영사전 영어 단어  Dictionary tree; //한영사전  char word2[80]; //영한사전 영어 단어  char meaning2[200]; //영한사전 한국어 단어  Dictionary tree2; //영한사전  BinaryNode\* tmp = NULL; //값을 나르는 임시 변수(한영사전)  BinaryNode\* tmp2 = NULL; //값을 나르는 임시 변수(영한사전)  BinaryNode\* korean\_copy[MAX\_WORD\_SIZE];  BinaryNode\* english\_copy[MAX\_WORD\_SIZE];  do {  help();  cin >> command;  while (getchar() != '\n');  //command = \_getche(); //키보드를 치는 것을 감지하여 엔터키 없이 바로 한 키보드 문자 입력, \_getche()로 해야 빌드 오류 X  //printf("\n");  switch (command) {  case 'i':  //printf(" > 한국어 단어: ");  cin.getline(word, 80, '\n');//구형 gets 사용이 VS 2022에서 아예 인식 X  strcpy(meaning2, word);  //printf(" > 영어 단어: ");  cin.getline(meaning, 200, '\n');  strcpy(word2, meaning);  tree.insert(new BinaryNode(word, meaning));  tree2.insert(new BinaryNode(word2, meaning2));  break;  case 'k':  //printf(" > 검색 단어: ");  cin.getline(word, 80, '\n');  tmp = tree.searchWord(word);  korean\_copy[korean\_count] = tmp;  list[count] = 'k';  count++;  if (tmp != NULL) {  //printf("%s %s\n", tmp->getData(), tmp->getData());  }  else {  result\_korean[korean\_count] = word;  //printf("%s UNKNOWN ENTRY\n", result\_korean[korean\_count]);  }  korean\_count++;  break;  case 'e':  //printf(" > 검색 단어: ");  cin.getline(word2, 80, '\n');  tmp2 = tree2.searchWord(word2);  english\_copy[english\_count] = tmp2;  list[count] = 'e';  count++;  if (tmp2 != NULL) {  //printf("%s %s\n", tmp->key.meaning, tmp->key.word);  }  else {  result\_english[english\_count] = word2;  //printf("%s UNKNOWN ENTRY\n", result\_english[english\_count]);  }  english\_count++;  break;  case 'p':  //printf("K-E Dictionary");  //tree.printAllWords();  //printf("K-E Dictionary");  //tree2.printAllWords();  //printf("\n");  list[count] = 'p';  count++;  break;  }  } while (command != 'q');  if (command == 'q') {  for (int a = 0; a < count; a++) {  if (list[a] == 'k') {  //printf("token for k\n");  for (int b = 0; b < korean\_count; b++) {  if (korean\_copy[b] != NULL) {  printf("%s %s\n", korean\_copy[b]->getData(), korean\_copy[b]->getMeaning());  }  else {  printf("%s UNKNOWN ENTRY\n", result\_korean[b]);  }  }  }  else if (list[a] == 'e') {  //printf("token for e\n");  for (int c = 0; c < english\_count; c++) {  if (english\_copy[c] != NULL) {  printf("%s %s\n", english\_copy[c]->getData(), english\_copy[c]->getMeaning());  }  else {  printf("%s UNKNOWN ENTRY\n", result\_english[c]);  }  }  }  else if (list[a] == 'p') {  if (tree.hasData() == true && tree2.hasData()==true) {  printf("K-E dictionary:\n");  tree.printAllWords();  printf("E-K dictionary:\n");  tree2.printAllWords();  }  }  else { continue; }  }  }    return 0;  } |

-난점

교재의 코드는 입력된 데이터를 저장하지 않습니다.(C++의 경우)

클래스 2개로 분리하셔야 합니다! 그래야 Sample input 1의 영-한 사전 순서를 지킬 수 있습니다.

라인편집기와 마찬가지로, 입력 시기와 출력 시기를 엄격히 구분해야 합니다.

gets를 반드시 쓰셔야 하며, strcpy도 쓰셔야 합니다. 그래야 수월하게 코드를 작성할 수 있습니다.

\*C++ 한정: C언어로는 정답이 나오는데, C++로 하면 컴파일 에러가 뜹니다.