**자료구조 트리 과제 코드**

202112936 엄정석

-문제(LMS)

|  |
| --- |
| 8장 이진트리 클래스에 아래 함수 추가    IsFull()함수: 포화이진트리(Full Binary Tree)인지 검사하는 함수    IsBalanced()함수: 왼쪽서브트리와 오른쪽서브트리의 높이의 차이가 2보다 작으면 균형잡혀있다(balanced)고 한다. 트리가 균형잡혀 있는지 검사하는 함수    reverse() 함수: 왼쪽서브트리와 오른쪽 서브트리를 바꾸는 함수  주요부분에 주석 추가해주세요.  몇가지 \*\*서로 다른 테스트 케이스에 대한 예제\*\*로 주요 알고리즘을 발표자료로 만들기.  \*\*\* 5월16일(화) 수업때 발표는 위 3개 함수로 할께요~  제출일: 5월15일(월) 오후11시59분까지  레포트 게시판: 숙제5\_이진트리함수 (\*\*게시판 이전 버전 선택\*\*)  프로그램명: pg5\_이름\_BinaryTree.(h, cpp)  알고리즘 설명: pg5\_이름\_BinaryTree.ppt |

-코드(C++)

|  |
| --- |
| #include <cstdio>  #include <cstdlib>  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) { }  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  bool isnonLeaf() { return left != NULL || right != NULL; }  };  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 {  BinaryNode\* root; //루트 포인터  public:  //=====================================================================================  // 자료구조 과제를 위한 함수 3개 추가  //=====================================================================================  //(1)번: 이진트리가 포화이진트리인지 검사하는 함수  bool isFull() {  if (!isEmpty()) {  CircularQueue q;  q.enqueue(root); //루트를 큐에 추가  while (!q.isEmpty()) {  BinaryNode\* n = q.dequeue();  //말단노드가 아니고 서브 트리가 한쪽 자식만 가지면 포화이진트리 X  if (!(n->isLeaf())  && n->getLeft() == NULL  && n->getRight() != NULL) { return false; }  else if (!(n->isLeaf())  && n->getLeft() != NULL  && n->getRight() == NULL) { return false; }  //잎 노드가 아니면 자식 노드를 따라가고, 자식 노드를 큐에 추가  if (n != NULL) {  if (n->getLeft())  q.enqueue(n->getLeft());  if (n->getRight())  q.enqueue(n->getRight());  }  }  return true;  //다 돌고 나서도 false가 나오지 않으면 true값 반환  }  }  //(2)번: 균형이 잡혀있는지를 검사하는 함수 - 왼쪽 서브트리와 오른쪽 서브트리의 높이의 차이가 2보다 작은지를 검사  bool isBalanced() { return isBalanced(root); }  bool isBalanced(BinaryNode\* node) {  int leftHeight;  int rightHeight;  if (node == NULL) { return true; } //공백 트리일 때  if (node->isLeaf()) { return true; } //잎 노드일 때  leftHeight = getHeight(node->getLeft());  rightHeight = getHeight(node->getRight());  //왼쪽 서브트리와 오른쪽서브트리의 높이차가 2이상 나지 않고  //둘 다 균형이 잡혀있을 경우  //true 반환  if ((leftHeight - rightHeight) <= 1  && (leftHeight - rightHeight) >= -1  && isBalanced(node->getLeft())  && isBalanced(node->getRight())) { return true; }  return false;  }  //(3)번: 좌우 서브트리를 교체하는 함수(좌우 대칭) - 함수 오버로딩  void reverse() { reverse(root); }  void reverse(BinaryNode\* node) {  if (node == NULL) { return; }  else {  //재귀  reverse(node->getLeft());  reverse(node->getRight());  //leftNode와 rightNode를 서로 바꾼다  BinaryNode\* temp = node->getLeft();  node->setLeft(node->getRight());  node->setRight(temp);  }  }  //=====================================================================================  // 자료구조 과제를 위한 함수 3개 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("inorder: "); inorder(root); }  void inorder(BinaryNode\* node) {  //중위 순회: LVR  if (node != NULL) { //루트 노드가 0이 아닐 때  inorder(node->getLeft()); //왼쪽 서브트리  printf(" [%c] ", 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(" [%c] ", 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(" [%c] ", 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(" [%c] ", 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()));  }  };  int main() {  BinaryTree tree;  //각 테스트 케이스를 보고 싶다면 별 주석 해제  //각 테스트 케이스를 보고 싶지 않다면 별 주석으로 감싸기  //단, 한 번에 하나만 가능(같은 문자가 있어 재정의되기 때문)    /\*  //===========================================================================================  // 테스트 케이스 1  // 검사하는 함수: 전형적인 포화 이진트리 (트리의 높이가 3)  // A  // B C  // D E F G  //===========================================================================================  BinaryNode\* d = new BinaryNode('D', NULL, NULL);  BinaryNode\* e = new BinaryNode('E', NULL, NULL);  BinaryNode\* f = new BinaryNode('F', NULL, NULL);  BinaryNode\* g = new BinaryNode('G', NULL, NULL);  BinaryNode\* b = new BinaryNode('B', d, e);  BinaryNode\* c = new BinaryNode('C', f, g);  BinaryNode\* a = new BinaryNode('A', b, c);  tree.setRoot(a);  printf("===================================================================================\n");  printf("테스트 케이스 1: 전형적인 포화 이진트리 =>");  tree.inorder(); //중위 순회  //tree.preorder();  //tree.postorder();  //tree.levelorder();  printf("\n\n");  //포화 이진트리 여부  if (tree.isFull()) { printf("이 트리는 포화 이진트리입니다.\n"); }  else { printf("이 트리는 포화 이진트리가 아닙니다.\n"); }  printf("\n");  //균형이 잡힌 이진트리 여부  if (tree.isBalanced()) { printf("이 트리는 균형이 잡혀있습니다.\n"); }  else { printf("이 트리는 균형이 잡혀있지 않습니다.\n");; }  printf("\n");  //뒤바꾸기 수행  tree.reverse();  printf("좌우 대칭 결과. =>"); tree.inorder();  printf("\n");  printf("===================================================================================\n");  printf("\n");  //===========================================================================================  \*/  /\*  //===========================================================================================  // 테스트 케이스 2  // 검사하는 함수: 전형적인 포화 이진트리 (트리의 높이가 4)  // A  // B C  // D E F G  // H I J K L M N O  //===========================================================================================  BinaryNode\* h = new BinaryNode('H', NULL, NULL);  BinaryNode\* i = new BinaryNode('I', NULL, NULL);  BinaryNode\* j = new BinaryNode('J', NULL, NULL);  BinaryNode\* k = new BinaryNode('K', NULL, NULL);  BinaryNode\* l = new BinaryNode('L', NULL, NULL);  BinaryNode\* m = new BinaryNode('M', NULL, NULL);  BinaryNode\* n = new BinaryNode('N', NULL, NULL);  BinaryNode\* o = new BinaryNode('O', NULL, NULL);  BinaryNode\* d = new BinaryNode('D', h, i);  BinaryNode\* e = new BinaryNode('E', j, k);  BinaryNode\* f = new BinaryNode('F', l, m);  BinaryNode\* g = new BinaryNode('G', n, o);  BinaryNode\* b = new BinaryNode('B', d, e);  BinaryNode\* c = new BinaryNode('C', f, g);  BinaryNode\* a = new BinaryNode('A', b, c);  tree.setRoot(a);  printf("===================================================================================\n");  printf("테스트 케이스 2: 전형적인 포화 이진트리 =>");  tree.inorder(); //중위 순회  //tree.preorder();  //tree.postorder();  //tree.levelorder();  printf("\n\n");  //포화 이진트리 여부  if (tree.isFull()) { printf("이 트리는 포화 이진트리입니다.\n"); }  else { printf("이 트리는 포화 이진트리가 아닙니다.\n"); }  printf("\n");  //균형이 잡힌 이진트리 여부  if (tree.isBalanced()) { printf("이 트리는 균형이 잡혀있습니다.\n"); }  else { printf("이 트리는 균형이 잡혀있지 않습니다.\n");; }  printf("\n");  //뒤바꾸기 수행  tree.reverse();  printf("좌우 대칭 결과. =>"); tree.inorder();  printf("\n");  printf("===================================================================================\n");  printf("\n");  //===========================================================================================  \*/    /\*  //===========================================================================================  // 테스트 케이스 3  // 검사하는 함수: 전형적인 완전 이진트리 (트리의 높이가 3)  // A  // B C  // D E  //===========================================================================================  BinaryNode\* d = new BinaryNode('D', NULL, NULL);  BinaryNode\* e = new BinaryNode('E', NULL, NULL);  BinaryNode\* b = new BinaryNode('B', d, e);  BinaryNode\* c = new BinaryNode('C', NULL, NULL);  BinaryNode\* a = new BinaryNode('A', b, c);  tree.setRoot(a);  printf("===================================================================================\n");  printf("테스트 케이스 3: 전형적인 완전 이진트리 =>");  tree.inorder(); //중위 순회  //tree.preorder();  //tree.postorder();  //tree.levelorder();  printf("\n\n");  //포화 이진트리 여부  if (tree.isFull()) { printf("이 트리는 포화 이진트리입니다.\n"); }  else { printf("이 트리는 포화 이진트리가 아닙니다.\n"); }  printf("\n");  //균형이 잡힌 이진트리 여부  if (tree.isBalanced()) { printf("이 트리는 균형이 잡혀있습니다.\n"); }  else { printf("이 트리는 균형이 잡혀있지 않습니다.\n");; }  printf("\n");  //뒤바꾸기 수행  tree.reverse();  printf("좌우 대칭 결과. =>"); tree.inorder();  printf("\n");  printf("===================================================================================\n");  printf("\n");  //===========================================================================================  \*/    /\*  //===========================================================================================  // 테스트 케이스 4  // 검사하는 함수: 전형적인 완전 이진트리 (트리의 높이가 4)  // A  // B C  // D E F G  // H I  //===========================================================================================  BinaryNode\* h = new BinaryNode('H', NULL, NULL);  BinaryNode\* i = new BinaryNode('I', NULL, NULL);  BinaryNode\* d = new BinaryNode('D', h, i);  BinaryNode\* e = new BinaryNode('E', NULL, NULL);  BinaryNode\* f = new BinaryNode('F', NULL, NULL);  BinaryNode\* g = new BinaryNode('G', NULL, NULL);  BinaryNode\* b = new BinaryNode('B', d, e);  BinaryNode\* c = new BinaryNode('C', f, g);  BinaryNode\* a = new BinaryNode('A', b, c);  tree.setRoot(a);  printf("===================================================================================\n");  printf("테스트 케이스 4: 전형적인 완전 이진트리 =>");  tree.inorder(); //중위 순회  //tree.preorder();  //tree.postorder();  //tree.levelorder();  printf("\n\n");  //포화 이진트리 여부  if (tree.isFull()) { printf("이 트리는 포화 이진트리입니다.\n"); }  else { printf("이 트리는 포화 이진트리가 아닙니다.\n"); }  printf("\n");  //균형이 잡힌 이진트리 여부  if (tree.isBalanced()) { printf("이 트리는 균형이 잡혀있습니다.\n"); }  else { printf("이 트리는 균형이 잡혀있지 않습니다.\n");; }  printf("\n");  //뒤바꾸기 수행  tree.reverse();  printf("좌우 대칭 결과. =>"); tree.inorder();  printf("\n");  printf("===================================================================================\n");  printf("\n");  //===========================================================================================  \*/  /\*  //===========================================================================================  // 테스트 케이스 5  // 검사하는 함수: 교재 340p 1번 그림: 포화 이진 트리 X, 균형 O, reverse 가능  // A  // B E  // C D F  //===========================================================================================    BinaryNode\* c = new BinaryNode('C', NULL, NULL);  BinaryNode\* d = new BinaryNode('D', NULL, NULL);  BinaryNode\* b = new BinaryNode('B', c, d);  BinaryNode\* f = new BinaryNode('F', NULL, NULL);  BinaryNode\* e = new BinaryNode('E', NULL, f);  BinaryNode\* a = new BinaryNode('A', b, e);  tree.setRoot(a);  printf("===================================================================================\n");  printf("테스트 케이스 5: 교재 340p =>");  tree.inorder(); //중위 순회  //tree.preorder();  //tree.postorder();  //tree.levelorder();  printf("\n\n");  //포화 이진트리 여부  if (tree.isFull()) { printf("이 트리는 포화 이진트리입니다.\n"); }  else { printf("이 트리는 포화 이진트리가 아닙니다.\n"); }  printf("\n");  //균형이 잡힌 이진트리 여부  if (tree.isBalanced()) { printf("이 트리는 균형이 잡혀있습니다.\n");}  else { printf("이 트리는 균형이 잡혀있지 않습니다.\n");; }  printf("\n");  //뒤바꾸기 수행  tree.reverse();  printf("좌우 대칭 결과. =>"); tree.inorder();  printf("\n");  printf("===================================================================================\n");  printf("\n");    //===========================================================================================  \*/  /\*  //===========================================================================================  // 테스트 케이스 6  // 검사하는 함수: 루트 노드 입장에서는 균형잡혔지만, 서브트리에서는 균형적이지 않은 경우  // A  // B C  // D E G  // H  // A의 관점: 서브 트리 레벨이 각각 2, 3이므로 균형잡힘  // B의 관점: 서브 트리 레벨이 각각 1, 1이므로 균형잡힘  // C의 관점: 서브 트리 레벨이 각각 0, 2이므로 균형잡히지 않음  //===========================================================================================  BinaryNode\* h = new BinaryNode('H', NULL, NULL);  BinaryNode\* d = new BinaryNode('D', NULL, NULL);  BinaryNode\* e = new BinaryNode('E', NULL, NULL);  BinaryNode\* g = new BinaryNode('G', h, NULL);  BinaryNode\* b = new BinaryNode('B', d, e);  BinaryNode\* c = new BinaryNode('C', NULL, g);  BinaryNode\* a = new BinaryNode('A', b, c);  tree.setRoot(a);  printf("===================================================================================\n");  printf("테스트 케이스 6: 서브트리에서는 균형적이지 않은 이진트리 =>");  tree.inorder(); //중위 순회  //tree.preorder();  //tree.postorder();  //tree.levelorder();  printf("\n\n");  //포화 이진트리 여부  if (tree.isFull()) { printf("이 트리는 포화 이진트리입니다.\n"); }  else { printf("이 트리는 포화 이진트리가 아닙니다.\n"); }  printf("\n");  //균형이 잡힌 이진트리 여부  if (tree.isBalanced()) { printf("이 트리는 균형이 잡혀있습니다.\n"); }  else { printf("이 트리는 균형이 잡혀있지 않습니다.\n");; }  printf("\n");  //뒤바꾸기 수행  tree.reverse();  printf("좌우 대칭 결과. =>"); tree.inorder();  printf("\n");  printf("===================================================================================\n");  printf("\n");  //===========================================================================================  \*/    /\*  //===========================================================================================  // 테스트 케이스 7  // 검사하는 함수: 직관적으로 균형잡히지 않은 이진트리  // A  // B  // C D  //===========================================================================================  BinaryNode\* d = new BinaryNode('D', NULL, NULL);  BinaryNode\* c = new BinaryNode('C', NULL, NULL);  BinaryNode\* b = new BinaryNode('B', c, d);  BinaryNode\* a = new BinaryNode('A', b, NULL);  tree.setRoot(a);  printf("===================================================================================\n");  printf("테스트 케이스 7: 직관적으로 균형잡히지 않은 이진트리 =>");  tree.inorder(); //중위 순회  //tree.preorder();  //tree.postorder();  //tree.levelorder();  printf("\n\n");  //포화 이진트리 여부  if (tree.isFull()) { printf("이 트리는 포화 이진트리입니다.\n"); }  else { printf("이 트리는 포화 이진트리가 아닙니다.\n"); }  printf("\n");  //균형이 잡힌 이진트리 여부  if (tree.isBalanced()) { printf("이 트리는 균형이 잡혀있습니다.\n"); }  else { printf("이 트리는 균형이 잡혀있지 않습니다.\n");; }  printf("\n");  //뒤바꾸기 수행  tree.reverse();  printf("좌우 대칭 결과. =>"); tree.inorder();  printf("\n");  printf("===================================================================================\n");  printf("\n");  //===========================================================================================  \*/    /\*  //===========================================================================================  // 테스트 케이스 8  // 검사하는 함수: 직관적으로 균형잡히지 않은 이진트리  // A  // B C  // D E  // F  //===========================================================================================  BinaryNode\* f = new BinaryNode('F', NULL, NULL);  BinaryNode\* d = new BinaryNode('D', NULL, NULL);  BinaryNode\* e = new BinaryNode('E', NULL, f);  BinaryNode\* c = new BinaryNode('C', NULL, NULL);  BinaryNode\* b = new BinaryNode('B', d, e);  BinaryNode\* a = new BinaryNode('A', b, c);  tree.setRoot(a);  printf("===================================================================================\n");  printf("테스트 케이스 8: 직관적으로 균형잡히지 않은 이진트리 =>");  tree.inorder(); //중위 순회  //tree.preorder();  //tree.postorder();  //tree.levelorder();  printf("\n\n");  //포화 이진트리 여부  if (tree.isFull()) { printf("이 트리는 포화 이진트리입니다.\n"); }  else { printf("이 트리는 포화 이진트리가 아닙니다.\n"); }  printf("\n");  //균형이 잡힌 이진트리 여부  if (tree.isBalanced()) { printf("이 트리는 균형이 잡혀있습니다.\n"); }  else { printf("이 트리는 균형이 잡혀있지 않습니다.\n");; }  printf("\n");  //뒤바꾸기 수행  tree.reverse();  printf("좌우 대칭 결과. =>"); tree.inorder();  printf("\n");  printf("===================================================================================\n");  printf("\n");  //===========================================================================================  \*/    /\*  //===========================================================================================  // 테스트 케이스 9  // 검사하는 함수: 경사 이진트리(skewed binary tree)  // A  // B  // C  // D  //===========================================================================================  BinaryNode\* d = new BinaryNode('D', NULL, NULL);  BinaryNode\* c = new BinaryNode('C', d, NULL);  BinaryNode\* b = new BinaryNode('B', c, NULL);  BinaryNode\* a = new BinaryNode('A', b, NULL);  tree.setRoot(a);  printf("===================================================================================\n");  printf("테스트 케이스 9: 경사 이진트리 =>");  tree.inorder(); //중위 순회  //tree.preorder();  //tree.postorder();  //tree.levelorder();  printf("\n\n");  //포화 이진트리 여부  if (tree.isFull()) { printf("이 트리는 포화 이진트리입니다.\n"); }  else { printf("이 트리는 포화 이진트리가 아닙니다.\n"); }  printf("\n");  //균형이 잡힌 이진트리 여부  if (tree.isBalanced()) { printf("이 트리는 균형이 잡혀있습니다.\n"); }  else { printf("이 트리는 균형이 잡혀있지 않습니다.\n");; }  printf("\n");  //뒤바꾸기 수행  tree.reverse();  printf("좌우 대칭 결과. =>"); tree.inorder();  printf("\n");  printf("===================================================================================\n");  printf("\n");  //===========================================================================================  \*/  /\*  //===========================================================================================  // 테스트 케이스 10  // 검사하는 함수: 경사 이진트리(skewed binary tree)  // A  // B  // C  // D  //===========================================================================================  BinaryNode\* d = new BinaryNode('D', NULL, NULL);  BinaryNode\* c = new BinaryNode('C', NULL, d);  BinaryNode\* b = new BinaryNode('B', NULL, c);  BinaryNode\* a = new BinaryNode('A', NULL, b);  tree.setRoot(a);  printf("===================================================================================\n");  printf("테스트 케이스 10: 경사 이진트리 =>");  tree.inorder(); //중위 순회  //tree.preorder();  //tree.postorder();  //tree.levelorder();  printf("\n\n");  //포화 이진트리 여부  if (tree.isFull()) { printf("이 트리는 포화 이진트리입니다.\n"); }  else { printf("이 트리는 포화 이진트리가 아닙니다.\n"); }  printf("\n");  //균형이 잡힌 이진트리 여부  if (tree.isBalanced()) { printf("이 트리는 균형이 잡혀있습니다.\n"); }  else { printf("이 트리는 균형이 잡혀있지 않습니다.\n");; }  printf("\n");  //뒤바꾸기 수행  tree.reverse();  printf("좌우 대칭 결과. =>"); tree.inorder();  printf("\n");  printf("===================================================================================\n");  printf("\n");  //===========================================================================================  \*/    /\*  //===========================================================================================  // 테스트 케이스 11  // 검사하는 함수: 편향 이진트리(degenerated binary tree)  // A  // B  // C  // D  //===========================================================================================  BinaryNode\* d = new BinaryNode('D', NULL, NULL);  BinaryNode\* c = new BinaryNode('C', d, NULL);  BinaryNode\* b = new BinaryNode('B', NULL, c);  BinaryNode\* a = new BinaryNode('A', b, NULL);  tree.setRoot(a);  printf("===================================================================================\n");  printf("테스트 케이스 9: 경사 이진트리 =>");  tree.inorder(); //중위 순회  //tree.preorder();  //tree.postorder();  //tree.levelorder();  printf("\n\n");  //포화 이진트리 여부  if (tree.isFull()) { printf("이 트리는 포화 이진트리입니다.\n"); }  else { printf("이 트리는 포화 이진트리가 아닙니다.\n"); }  printf("\n");  //균형이 잡힌 이진트리 여부  if (tree.isBalanced()) { printf("이 트리는 균형이 잡혀있습니다.\n"); }  else { printf("이 트리는 균형이 잡혀있지 않습니다.\n");; }  printf("\n");  //뒤바꾸기 수행  tree.reverse();  printf("좌우 대칭 결과. =>"); tree.inorder();  printf("\n");  printf("===================================================================================\n");  printf("\n");  //===========================================================================================  \*/    //===========================================================================================  // 테스트 케이스 12  // 검사하는 함수: 공백 트리 (루트 포인터가 NULL, 곧 노드 1개가 NULL만 있는 경우)  // 공백 트리도 정의상 이진트리  //===========================================================================================  BinaryNode\* a = new BinaryNode(NULL, NULL, NULL);  tree.setRoot(a);  printf("===================================================================================\n");  printf("테스트 케이스 12: 공백 트리 =>");  tree.inorder(); //중위 순회  //tree.preorder();  //tree.postorder();  //tree.levelorder();  printf("\n\n");  //포화 이진트리 여부  if (tree.isFull()) { printf("이 트리는 포화 이진트리입니다.\n"); }  else { printf("이 트리는 포화 이진트리가 아닙니다.\n"); }  printf("\n");  //균형이 잡힌 이진트리 여부  if (tree.isBalanced()) { printf("이 트리는 균형이 잡혀있습니다.\n"); }  else { printf("이 트리는 균형이 잡혀있지 않습니다.\n");; }  printf("\n");  //뒤바꾸기 수행  tree.reverse();  printf("좌우 대칭 결과. =>"); tree.inorder();  printf("\n");  printf("===================================================================================\n");  printf("\n");  //===========================================================================================  return 0;  } |

-난점: 딱히 없으나, 테스트 케이스를 많이 만들어야 함. + 정의를 정확히 아는 것이 중요합니다.