Data Structures

Lab # 09

Data & Knowledge Engineering Laboratory
Kyung Hee University, Korea

Chapter 8 Programming Exercise

Lab 09

- 1. Exercise 21
- 2. Exercise 22
- 3. Exercise 29
- 4. Exercise 30
- 5. Exercise 31

실습 준비

- **샘플 코드에 구현된 트리를 사용**
  - ...₩labplus_CRLF₩labplus₩Lab, C++ 3rd₩Chapter8₩Recursive Tree
  - Chapter8중 Recursive 방식을 사용한 트리를 사용

- **실습 문제에 해당하는 함수를 클래스 선언문에(TreeType.h) 추가하고, TreeType.cpp에 해당 함수를 구현**

- **사용할 파일**
  - QueType.h, QueType.cpp, TreeType.h, TreeType.h

1. Exercise 21

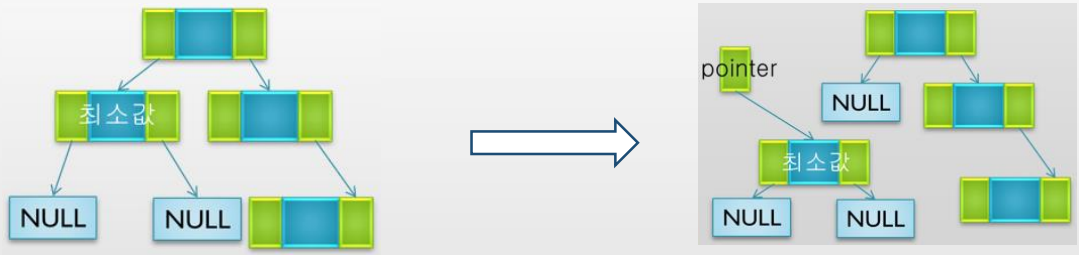
■ 문 제

- ❖ 트리 내의 가장 작은 키를 가진 노드를 찾고 트리에서 그 노드의 연결을 제거한 뒤 연결이 제거된 노드를 가리키는 포인터를 반환하는 PtrToSuccessor 함수를 작성하라

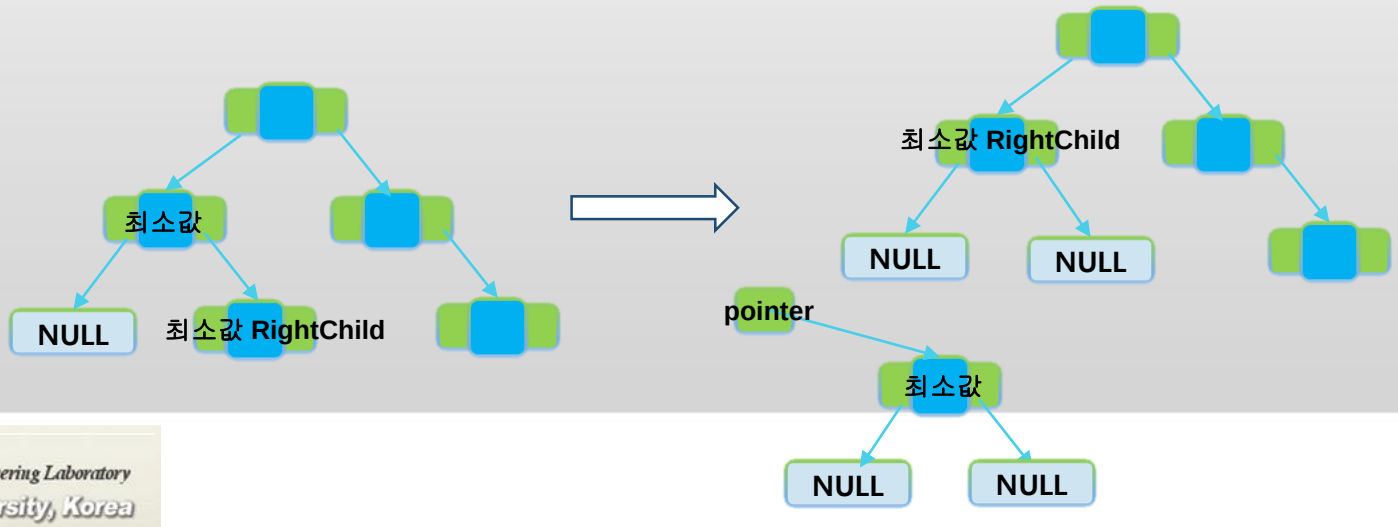
■ 예제

PtrToSuccessor 수행 후

Case 1:



Case 2:



1-help slides

```
// 입력 받은 노드부터 시작되는 서브트리의 왼쪽 노드만 따라가 가장 작은 값을 찾는다.
// 아래 두 가지 버전을 모두 해 볼 것

// Recursive version:
TreeNode* TreeType::PtrToSuccessor(_____________ tree)  // 파라메터 타입: (1) TreeNode* or (2) TreeNode*&
{
        if(tree->left != NULL) //왼쪽 노드가 NULL이 아니면
                _____________________; // general case
        else {      // base case
                TreeNode *tempPtr = ___________; // tree 값을 backup
                // tree가 tree의 right child를 가리키도록 수정
                //right child가 null인 경우 자연스럽게 case1을 만족
                // tempPtr을 리턴
        }
}

// Nonrecursive version:
TreeNode* TreeType::PtrToSuccessor(_____________ tree)  // 파라메터 타입: (1) TreeNode* or (2) TreeNode*&
{
        while (_____________) // 제일 왼쪽 노드까지 내려간다
                tree = tree->left;

        TreeNode *tempPtr = ___________; // tree 값을 backup
        // tree가 tree의 right child를 가리키도록 수정
        // tempPtr을 리턴
}
```

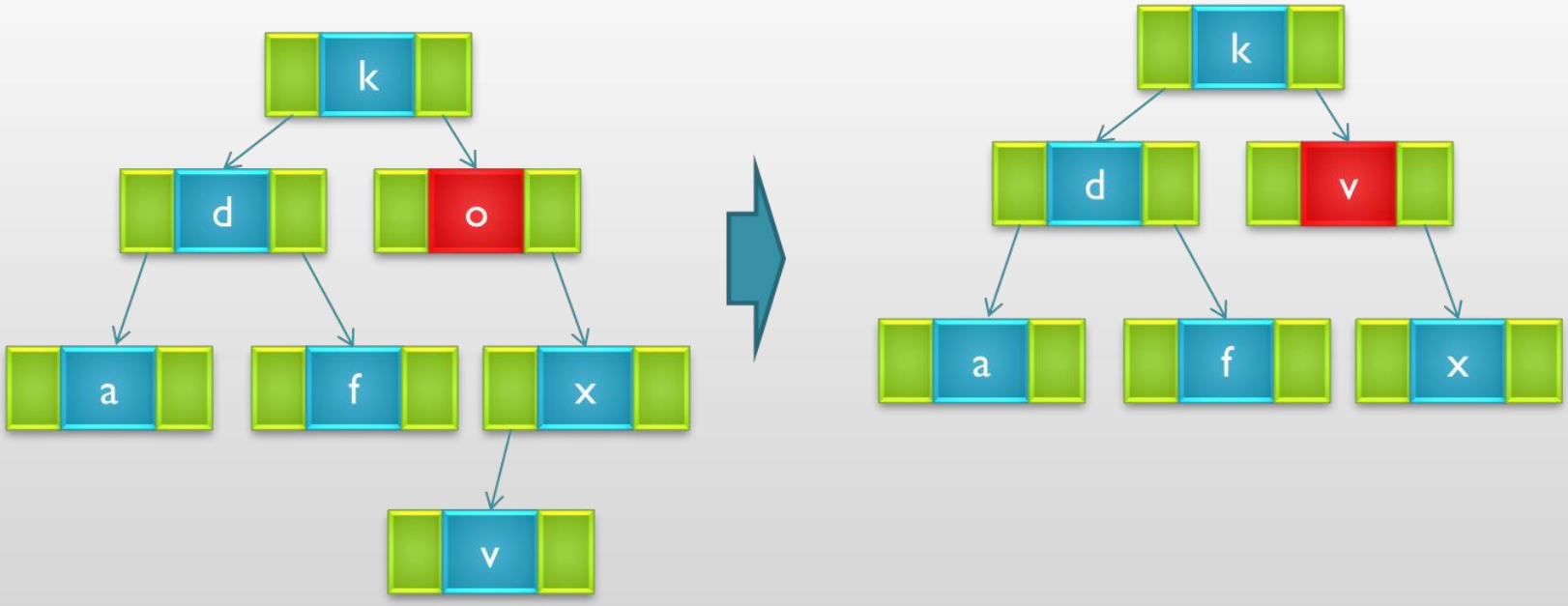
2. Exercise 22

■ 문 제

❖ 두 자식 노드를 가진 노드를 삭제할 경우에 삭제되는 값의 중간 후임자(전임자가 아닌)를 사용하도록 이 장의 DeleteNode 함수를 수정하라. 또한 이전 연습문제에서 작성한 PtrToSuccessor 함수를 호출하라.

- 중간 후임자 (immediate successor) 삭제하려는 값 다음으로 작은 값

■ 예 제



❖ O를 삭제한 경우 o의 위치에 중간 후임자가 위치하게 한다.

2-help slide

- 삭제하려는 노드의 right에 위치한 서브 트리에서 가장 작은 값을 찾으시오

```
void DeleteNode(TreeNode*& tree)  //이미 구현된 소스에 수정하세요.
{
  ItemType data;
  TreeNode* tempPtr;

  tempPtr = tree;
  if (tree->left == NULL)
  {
    tree = tree->right;
    delete tempPtr;
  }
  else if (tree->right == NULL)
  {
    tree = tree->left;
    delete tempPtr;
  }
  else
  {
        //이 부분을 전임자가 아닌 중간 후임자의 값이 들어가도록 수정.
        //삭제하려는 노드의 오른쪽에서 PtrToSuccessor()를 사용한다.
        //값을 대치하고 노드 삭제.
  }
}
```

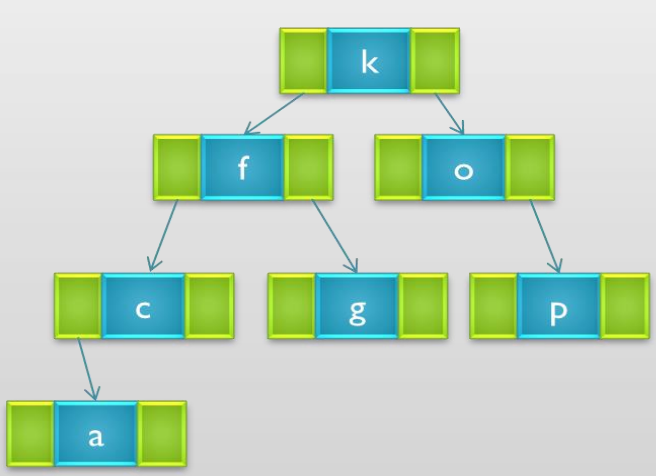
1번 문제에서는 PtrToSuccessor()를 멤버함수로 구현하였는데,
DeleteNode() 함수는 멤버함수가 아니므로 멤버함수를 호출할 수 없다.
따라서, 1번 문제의 구현을 비 멤버함수로 변경하여 사용해야 한다.

3. Exercise 29

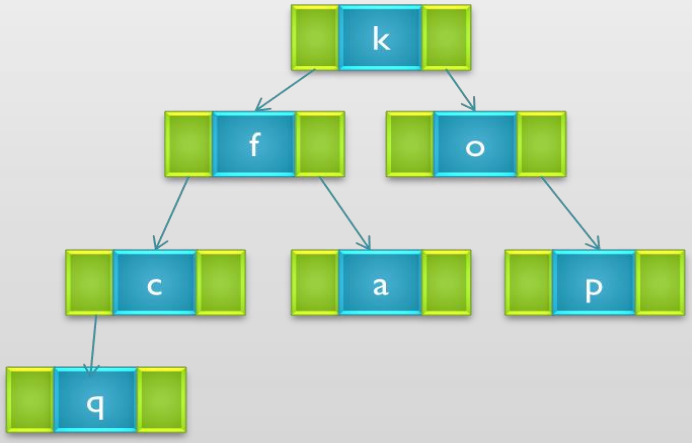
■ 문 제

- ❖ 이진 트리가 이진 검색 트리인지를 결정하는 부울 멤버 함수 IsBST를 TreeType 클래스에 추가하라.
  - 이진 검색 트리: root 노드를 기준으로 값이 작으면 왼쪽, 크면 오른쪽에 위치한 트리의 형태
- ❖ a. 적당한 주석을 포함하여 IsBST 함수의 선언을 작성하라.
- ❖ b. 이 함수의 재귀적 구현을 작성하라.

■ 예 제



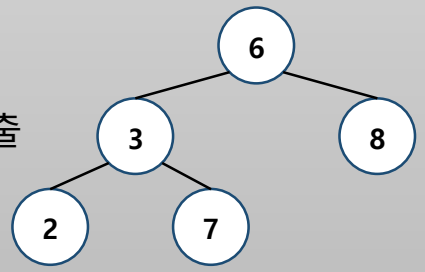
return true



return false

3-help slide

```
// 노드를 재귀적으로 검사하면서 노드가 가지고 있는 값을 비교하여 이진 검색 트리인지 아닌지 검사한다.

bool Imp_IsBST(TreeNode* tree);

bool TreeType::IsBST() // 클래스에 IsBST 함수를 선언하세요.
{
        return Imp_IsBST (root);
}

bool Imp_IsBST(TreeNode* tree)
{
        bool BST = true;        // 기본값(마지막 NULL일 경우)
        if(tree != NULL){       // 트리가 비지 않거나, 마지막 노드가 아니라면
                //왼쪽 노드가 NULL이 아니고, 왼쪽 노드의 값이 현재 노드의 값보다 클 경우 BST는 fals
                if(...)

                //오른쪽노드가 NULL이 아니고, 오른쪽 노드의 값이 현재 노드의 값보다 작을 경우 BST
if(...)

                BST = Imp_IsBST (tree->left);  // 왼쪽에 대해 재귀 호출
                BST = Imp_IsBST (tree->right);   // 오른쪽에 대해 재귀 호출
        }
        return BST;
}
```



이렇게 작성하면 틀림!!! 뭐가 잘못 됐을까?

Is BST or not?

3-help slide

```
// 노드를 재귀적으로 검사하면서 노드가 가지고 있는 값을 비교하여 이진 검색 트리인지 아닌지 검사한다.
bool Imp_IsBST(TreeNode* tree, ItemType &min, ItemType &max);

bool TreeType::IsBST() // 클래스에 IsBST 함수를 선언하세요.
{
        ItemType min, max;
        return Imp_IsBST (root, min, max);
}

bool Imp_IsBST(TreeNode* tree, ItemType &min, ItemType &max) // min, max: returns the value range of the tree
{
        bool isBST;

        if(tree = NULL) return true; // emptry tree는 BST

        //왼쪽 노드가 NULL이 아니면, 왼쪽 서브트리가 BST인지 체크하고 tree->info와 비교
        if(...) {
            isBST = Imp_IsBST(tree->left, left_min, left_max);
            // 왼쪽 서브트리가 BST가 아니거나 tree->info가 왼쪽 서브트리 값보다 작은 경우
            if (!isBST || tree->info <= _______) return false;
        }
        //오른쪽 노드가 NULL이 아니면, 오른쪽 서브트리가 BST인지 체크하고 tree->info와 비교
        if(...) {
                    // 왼쪽 서브트리 코드 참조하여 작성
        }

        min = (tree->left == NULL) ? __ : __ ;      max = ...; // min, max의 값을 설정
        return true;
}
```

4. Exercise 30

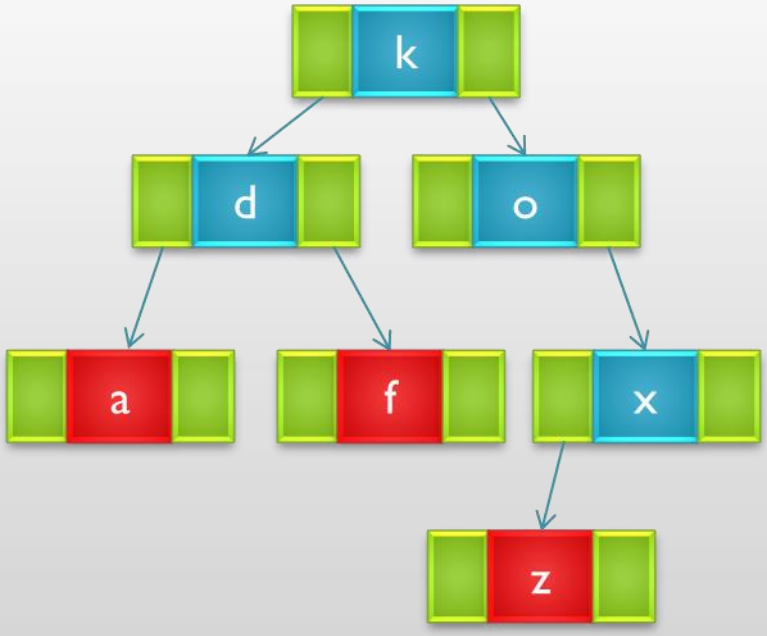
■ 문 제

❖ 트리의 리프 노드 수를 반환하는 LeafCount 멤버 함수를 포함하도록 이진 검색 트리 ADT를 확장하라.

- 노드의 left와 right가 모두 NULL인 경우가 LeafNode이다.

■ 예 제

❖ LeafCount() = 3



4-help slide

```
//노드의 좌우가 NULL인 노드를 찾을 때까지 반복하면서, 찾으면 1을 반환하여 노드의
//개수를 카운트 한다.

int Imp_LeafCount(TreeNode *tree);

int TreeType::LeafCount()
{
        return Imp_LeafCount(root);
}

int Imp_LeafCount (TreeNode *tree)
 {
        if(tree==NULL) //리프 노드가 아닐 경우.
                return 0;
        else if(…) //노드의 좌우가 NULL일 경우
                return 1;
        else
                //노드의 좌우를 재귀 호출하여 더한다.
}
```

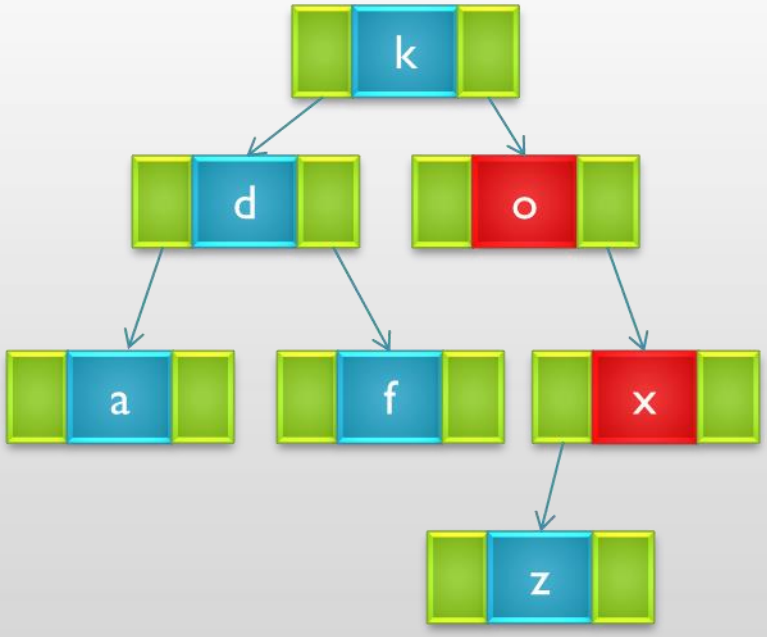
5. Exercise 30

문 제

- 트리 내의 노드 중 하나의 자식을 가진 노드 수를 반환하는 SingleParentCount 멤버 함수를 포함하도록 이진 검색 트리 ADT를 확장하라.
  - 자식노드가 1개인 노드의 개수를 반환

예 제

- SingleParentCount() = 2



5-help slide

```
//4가지 경우로 나누어서 재귀 호출을 할 수 있다.
//1. 트리가 비었거나, 마지막 노드인 경우
//2. 왼쪽에만 노드가 있을 경우
//3. 오른쪽에만 노드가 있을 경우
//4. 노드를 2개 가지고 있을 경우

int Imp_SingleParentCount(TreeNode *tree);

int TreeType::SingleParentCount()
{
        return Imp_SingleParentCount(root);
}

int Imp_SingleParentCount(TreeNode *tree)
{
        if(tree==NULL)
                //0을 리턴.
        else if(tree->left == NULL && tree->right != NULL)
                //1개의 노드를 가지므로 오른쪽 노드를 재귀 호출하고 1을 더하여 리턴.
        else if(tree->right == NULL && tree->left != NULL)
                //1개의 노드를 가지므로 왼쪽 노드를 재귀 호출하고 1을 더하여 리턴.
        else
                //노드의 양쪽을 재귀 호출하여 더한다.
}
```