

Data Structures

Lab # 10



- 1. Exercise 26
- 2. Exercise 27
- 3. Exercise 32
- 4. Exercise 35
- 5. Exercise 36

■ 샘플 코드에 구현된 트리를 사용

- ❖ ...\\labplus_CRLF\\labplus\\Lab, C++ 3rd\\Chapter8\\Recursive Tree
- ❖ Chapter8중 **Recursive 방식**을 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.cpp

■ 4,5번 문제 : 첨부파일 확인

- ❖ TreeType.h와 QueType.h 상단에 typedef **char** ItemType -> typedef **int** ItemType으로 변경->변경안해도 무관

- ❖ 4번문제 : SortedType.h파일 사용 5번문제 : UnSortedType.h파일 사용

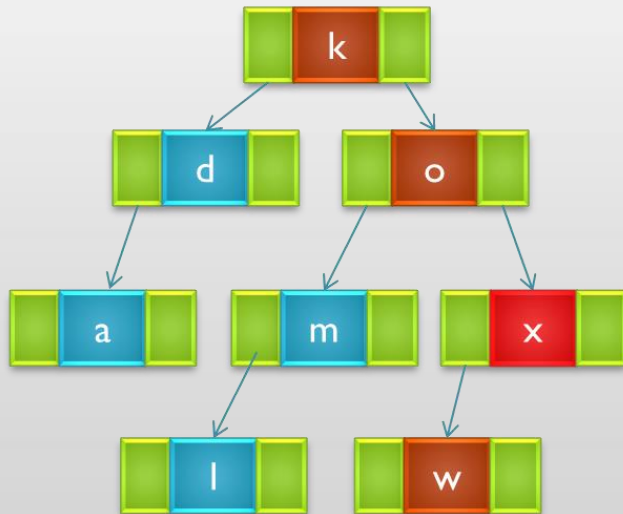
1. Exercise 26

■ 문 제

- ❖ TreeType멤버 함수로 value값을 기준으로 조상노드에 저장된 모든 값을 출력하는 Ancestors 함수를 작성하라 (단, value값은 출력하지 않음)
- ❖ 함수 프로토타입을 정의하고, 반드시 반복 버전(Iterative)으로 작성하시오.

■ 예 제

- ❖ Value = 'x', tree.Ancestors(Value)



k, o 를 출력
(x는 출력하지 않음에 주목)

1-help slide

//입력 받은 값을 만날 때까지 노드를 따라가면서 내용을 출력해준다.

```
void TreeType::Ancestors(ItemType value)
{
    bool found=false;
    QueType path;

    TreeNode *currentNode = root;

    while(_____) // value를 가진 노드를 찾거나 트리 끝에 도달할 때까지
    {
        if(_____) // value를 가진 노드를 찾으면
            found = true;
        else {
            path.Enqueue(_____); // path에 현재 노드의 값을 삽입
            if(_____)
                _____; // 왼쪽 서브트리로 이동
            else
                _____; // 오른쪽 서브트리로 이동
        }
    }

    if (found)
        // path에 있는 내용을 출력
    else
        // value를 찾을 수 없다고 출력
}
```

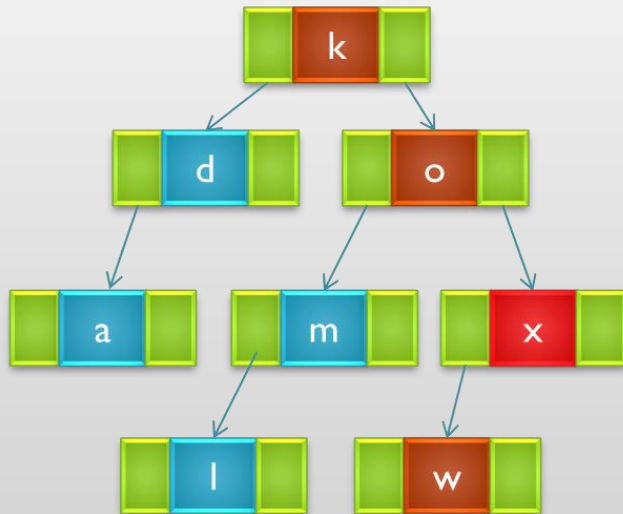
2. Exercise 27

■ 문 제

- ❖ 1번 문제에서 작성한 Ancestors함수를 재귀 함수로 작성하시오.

■ 예 제

- ❖ Value = 'x', tree.Ancestors(Value)



k, o 를 출력
(x는 출력하지 않음에 주목)

2-help slides

```
bool Ancestors_recursive(TreeNode* tree, ItemType item)
{
    bool found = false;
    if(tree == NULL)
        return false;

    if ( _____ ) // value를 찾은 경우
        return true;

    if ( _____ )
        found = Ancestors_recursive(____, item); // 왼쪽 서브트리 호출
    else
        found = Ancestors_recursive(____, item); // 오른쪽 서브트리 호출

    if(found) // 재귀호출 후 value값을 찾았으면
        // 현재 노드의 값을 출력

    return found;
}

void TreeType::Ancestors_re(ItemType value)
{
    Ancestors_recursive(root, value);
}
```

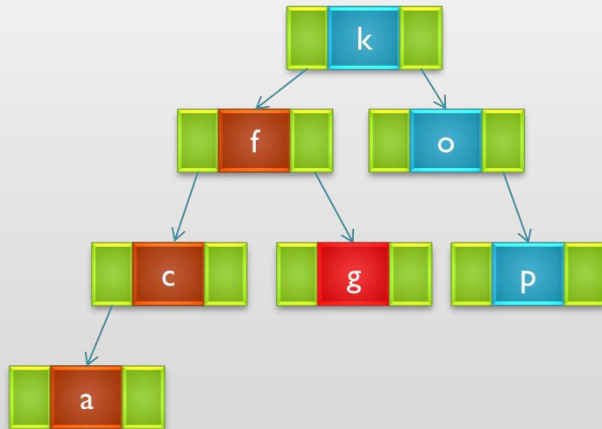
3. Exercise 32

■ 문 제

- ❖ 트리내에서 Value 보다 작은 값을 갖는 노드의 수를 반환하는 클라이언트 함수 **Smaller**를 작성하라.

■ 예 제

- ❖ Value = 'g', Smaller(Value)



‘g’를 입력 받을 경우 g보다 작은 a, c, f의 개수인 3이 리턴되어야 한다.

//트리로부터 아이템을 순차적으로 얻으면서 그 값의 개수를 카운트 한다.
//주어진 value보다 큰 아이템을 얻을 경우 즉시 루프를 벗어나도록 한다.

```
int Smaller(TreeType tree, int value)
{
    ItemType item;
    bool finished = false;
    int count=0;

    tree.ResetTree(IN_ORDER);

    do {
        tree.GetNextItem(_____, _____);

        if (_____)
            count++;
        else
            finished = true;
    } while (!finished);

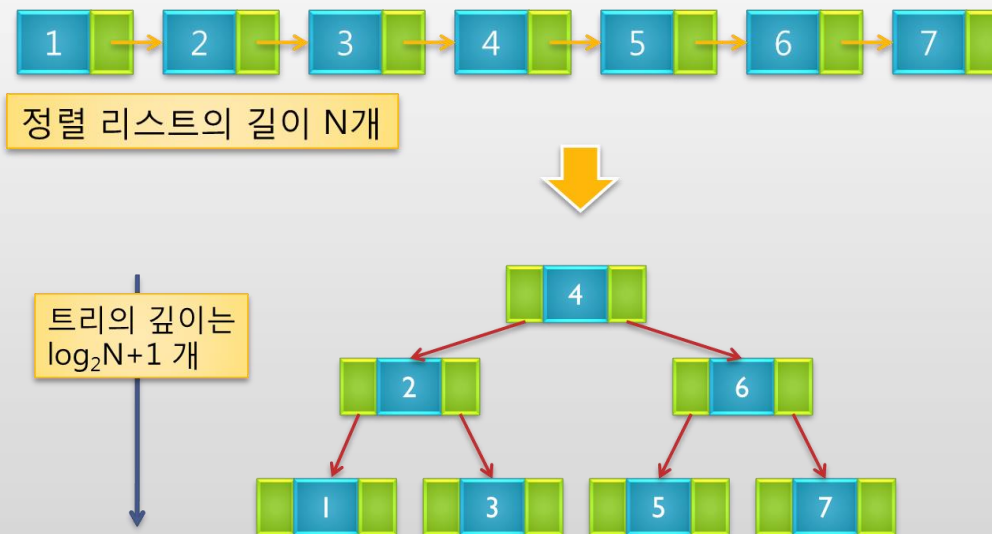
    return count;
}
```

4. Exercise 35

■ 문 제

- ❖ 정수형의 정렬 리스트 요소로부터 이진 검색 트리를 생성하는 **MakeTree** 클라이언트 함수를 작성하라.
- 만약 순차적으로 입력한다면 N레벨의 트리를 만들게 된다. 이점을 유의하여 최대 $\log_2 N + 1$ 레벨을 가지는 트리를 만들어야 한다.

■ 예 제



4-help slide

//리스트가 가지는 아이템을 배열로 옮긴 뒤, 배열에서 입력 순서를 계산하여 트리에 입력한다.

```
void AddElement(TreeType& tree,int Array[],int from,int to);  
void MakeTree(TreeType &tree, SortedType<int> &list);
```

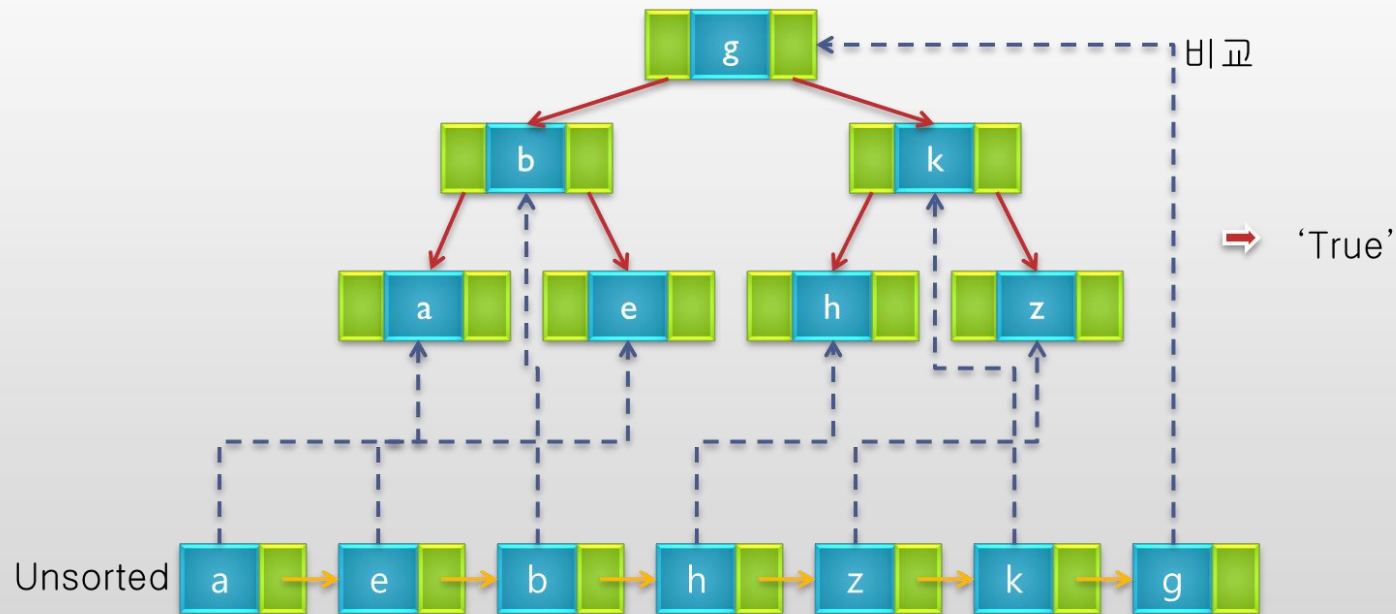
```
void MakeTree(TreeType &tree, SortedType<int> &list)  
{  
  
    int length = list.LengthIs(); //리스트 크기를 얻는다  
    int * array = new int[length]; //동적 배열 할당  
    int item_info;  
    int i;  
  
    list.ResetList();  
  
    for(i=0; i < length; i++)  
    {  
        list.GetNextItem(item_info);  
        array[i] = item_info;  
    }  
  
    AddElement(tree, array, 0, length-1);  
  
    delete [] array; // 동적 배열 삭제  
}  
void AddElement(TreeType& tree,int Array[],int front, int rear)  
{  
    int mid; //중간 값을 기록하는 변수  
    if(_____) { //함수의 종료조건  
        mid = _____;  
        tree.InsertItem(Array[mid]);  
        AddElement(tree,Array,_____,_____);  
        AddElement(tree,Array,_____,_____);  
    }  
}
```

5. Exercise 36

■ 문 제

- ❖ 이진 검색 트리와 비정렬 순차 리스트가 동일한 값들을 가지는지를 결정하는 **MatchingItem** 클라이언트 함수를 작성하라.

■ 예 제



List와 tree의 아이템을 꺼내가면서 비교한다

5-help slide

```
bool MatchingItem_Unsorted(TreeType &tree, UnsortedType<ItemType> &list)
{
    int list_length = list.LengthIs();
    int tree_length = tree.LengthIs();

    if(list_length != tree_length) //길이를 먼저 비교, 같은지 검사한다.
    {
        return false;
    }else
    {
        ItemType item;
        bool found;

        _____; //list에 iterator를 사용할 준비를 한다
        for (int i = 0; i < list_length; i++) {
            _____; // list에서 하나의 아이템을 가져온다
            tree.RetrieveItem(item, found); //트리에 해당 아이템이 있는지 검색. O(logN)이 걸림.
            if (!found) _____;
        }
        return true;
    }
}

//아래 구현과 비교해 볼 때 수행시간 면에서 무슨 차이가 있을까?
tree.ResetTree(IN_ORDER); // tree에 iterator를 사용할 준비를 한다
for (int i = 0; i < list_length; i++) {
    tree.GetNextItem(item, IN_ORDER, found); // tree에서 하나의 아이템을 가져온다
    list.RetrieveItem(item, found); // list에 해당 아이템이 있는지 검색. O(N)이 걸림.
    if (!found) _____;
}
```