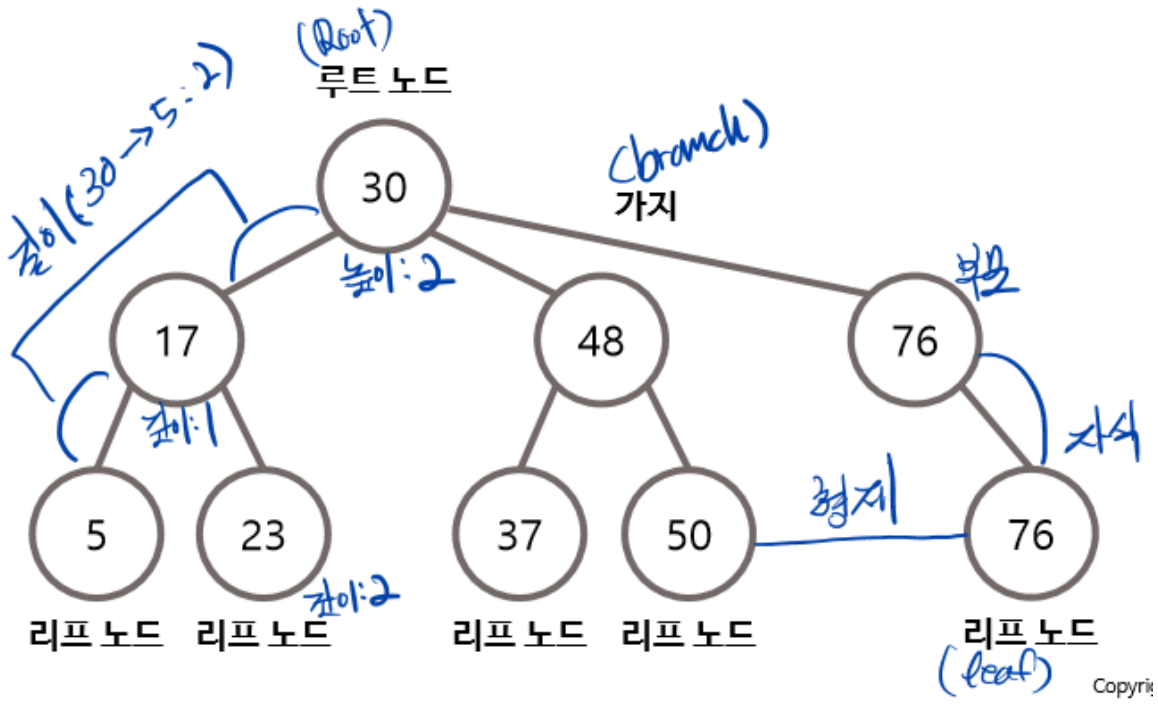
# Binary Tree (이진 트리)

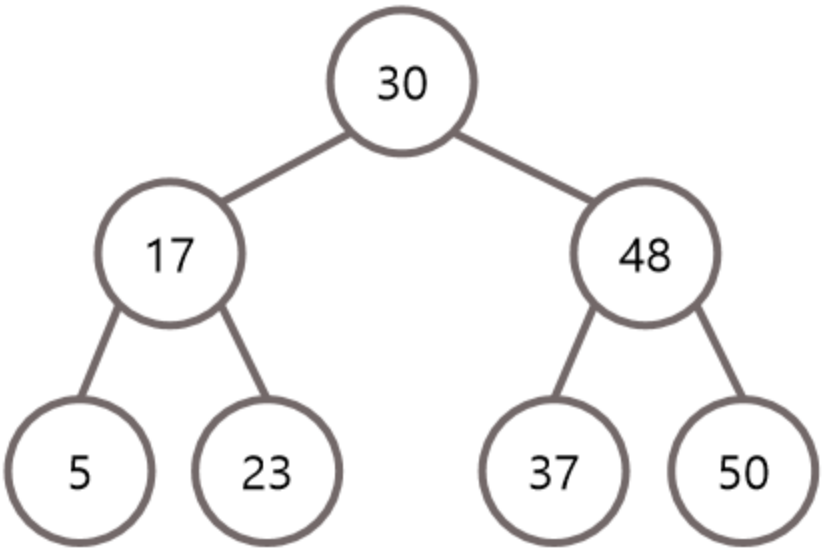
## 개념

1. Tree : 뒤집은 나무 형태의 자료구조

트리

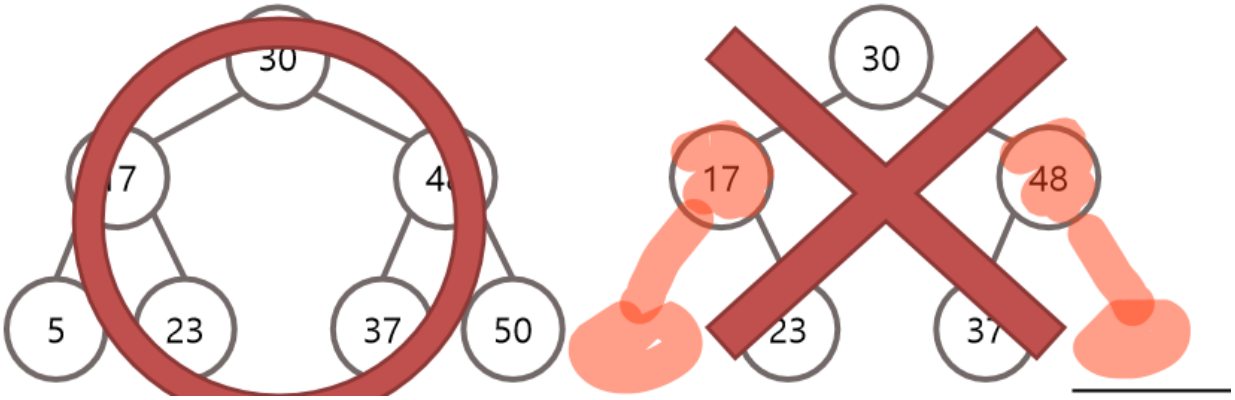



2. Binary Tree : 최대 2개의 자식을 가질 수 있는 트리

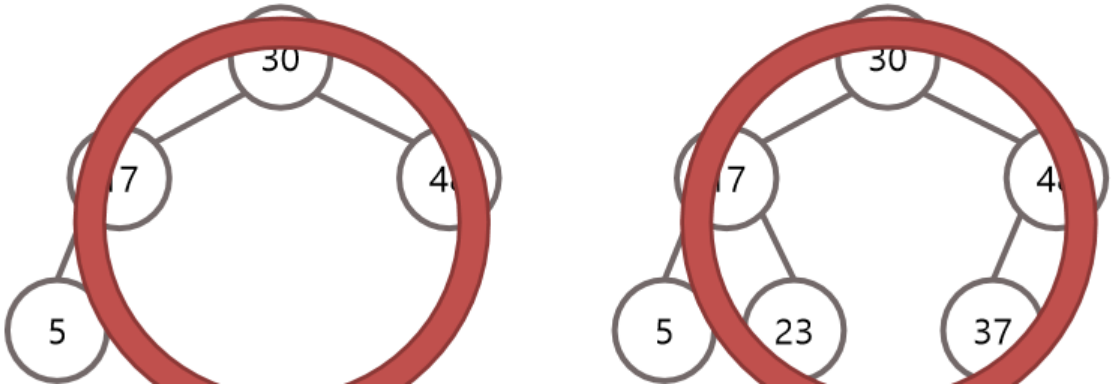


## 3. Type of Binary Tree

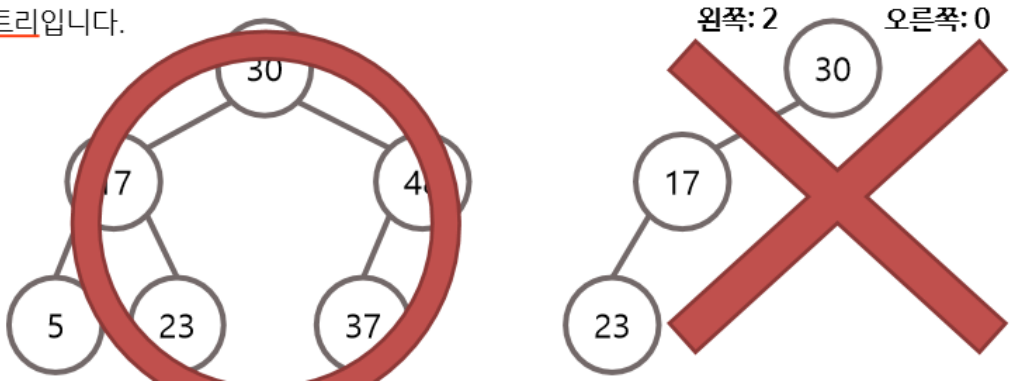
포화 이진 트리(Full Binary Tree)는 리프 노드를 제외한 모든 노드가 두 자식을 가지고 있는 트리입니다.




완전 이진 트리(Complete Binary Tree)는 모든 노드들이 왼쪽 자식부터 차근차근 채워진 노드입니다.




높이 균형 트리(Height Balanced Tree)는 왼쪽 자식 트리와 오른쪽 자식 트리의 높이가 1 이상 차이 나지 않는 트리입니다.



# 구현

## 1. struct

```
typedef struct ST_Node
{
        int nData;
        struct ST_Node *pLeftChild;
        struct ST_Node *pRightChild;
} Node;

Node* InitNode(int nData, Node* pLeftChild, Node* pRightChild)
{
        Node* pNode = (Node*)malloc(sizeof(Node));
        pNode->nData                = nData;
        pNode->pLeftChild        = pLeftChild;
        pNode->pRightChild        = pRightChild;

        return pNode;
}
```

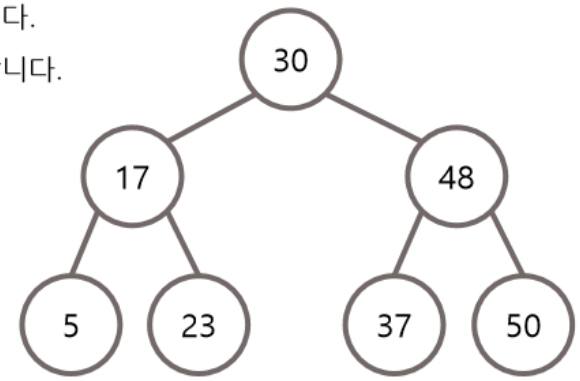
## 2. 출력

### 1) 전위

이진 트리의 전위 순회

1) 자기 자신을 출력합니다.
2) 왼쪽 자식을 방문합니다.
3) 오른쪽 자식을 방문합니다.

**출력 내용:** 30 - 17 - 5 - 23 - 48 - 37 - 50





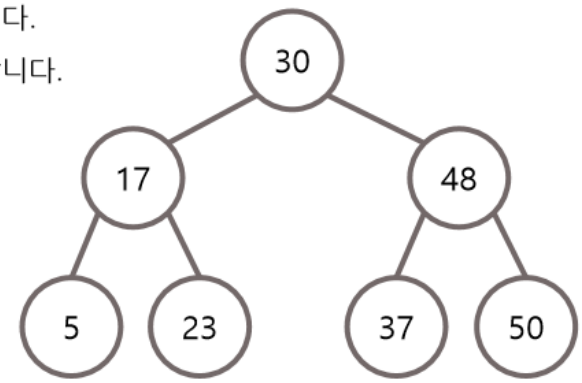

```
void PreOrder(Node *pNode) {
        if (pNode) {
                printf("%d ", pNode->nData);
                PreOrder(pNode->pLeftChild);
                PreOrder(pNode->pRightChild);
        }
}
```

**2) 중위**

이진 트리의 중위 순회

1) 왼쪽 자식을 방문합니다.
2) 자기 자신을 출력합니다.
3) 오른쪽 자식을 방문합니다.

**출력 내용:** 5 - 17 - 23 - 30 - 37 - 48 - 50






```
void InOrder(Node *pNode) {
```

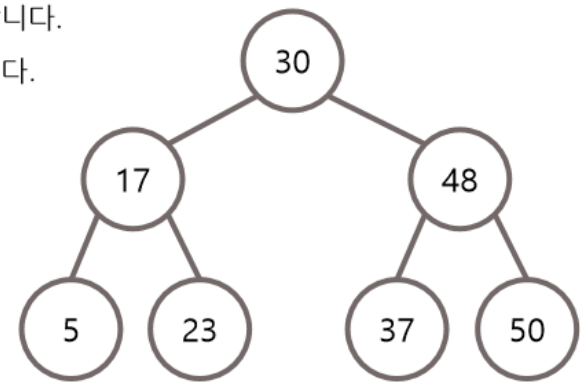
```
        if (pNode) {
                InOrder(pNode->pLeftChild);
                printf("%d ", pNode->nData);
                InOrder(pNode->pRightChild);
        }
}
```

3. 후위

이진 트리의 후위 순회

1) 왼쪽 자식을 방문합니다.
2) 오른쪽 자식을 방문합니다.
3) 자기 자신을 출력합니다.

**출력 내용: 5** - 23 - 17 - 37 - 50 - 48 - 30






```
void PostOrder(Node *pNode) {
        if (pNode) {
                PostOrder(pNode->pLeftChild);
                PostOrder(pNode->pRightChild);
                printf("%d ", pNode->nData);
        }
}
```