자료 구조 (Tree) 분석

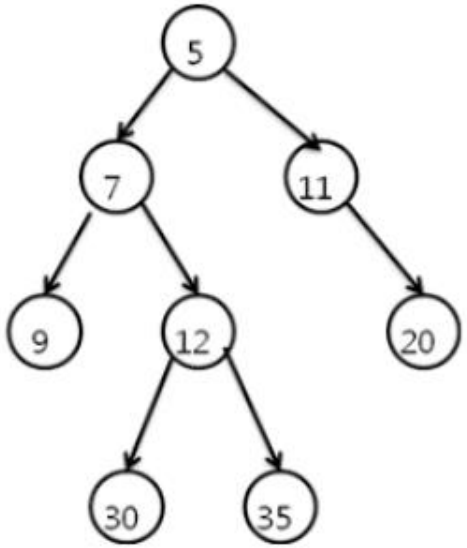
(미완성)

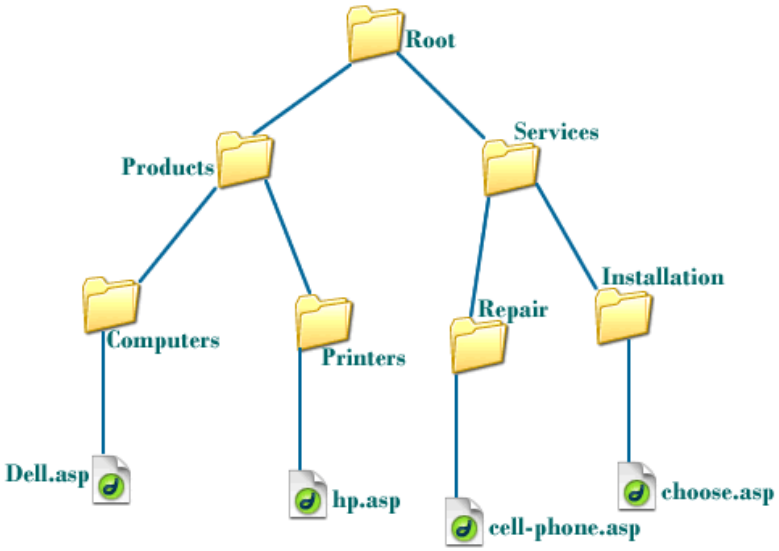
홍정완

(HTTPS://GITHUB.COM/JUNG9928)

- 목 차 -

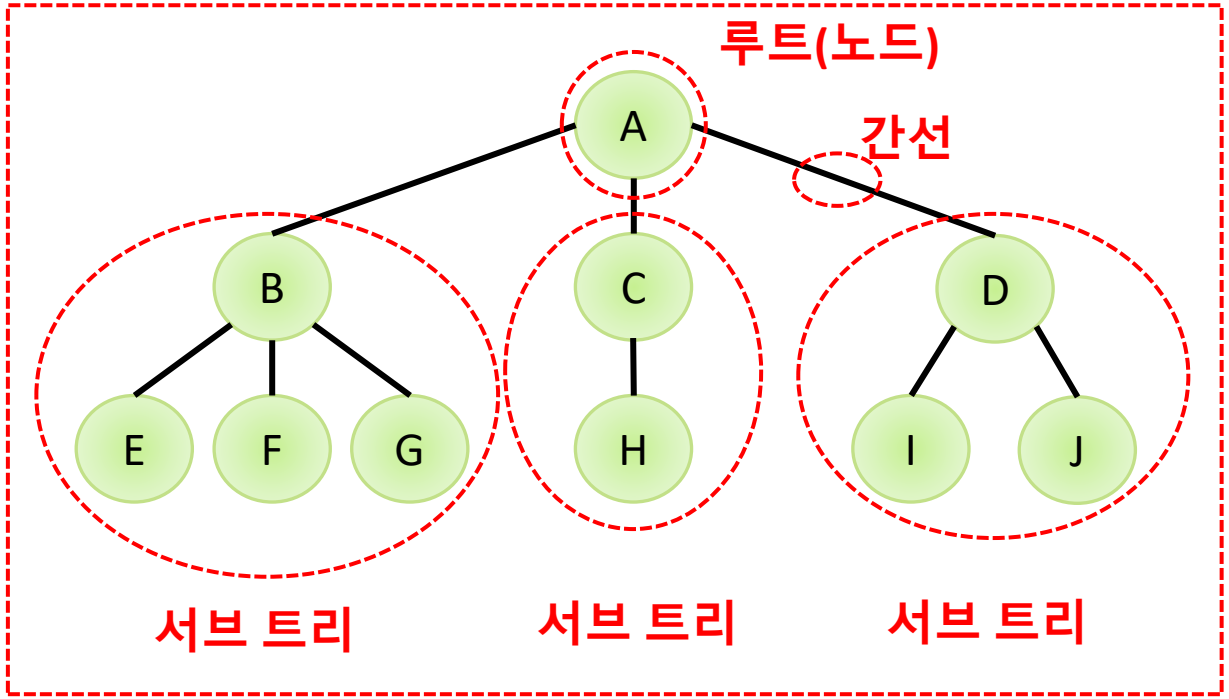
1. 트리(Tree) 란?





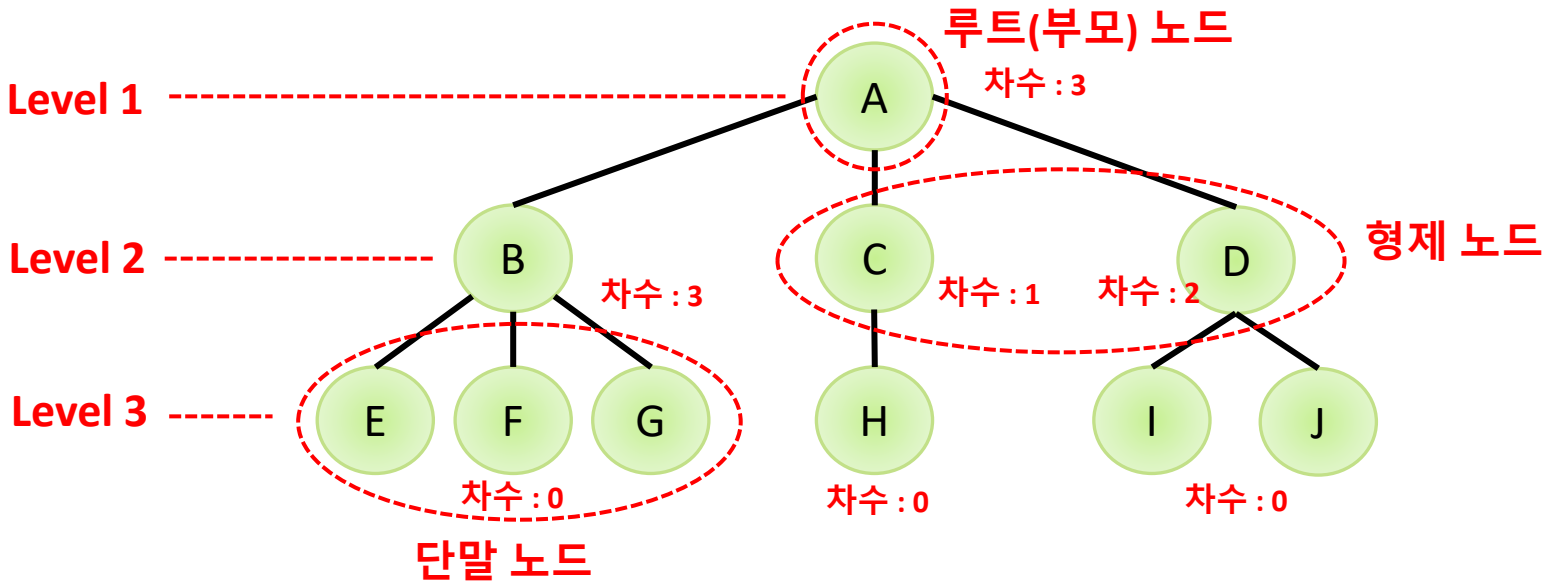
- 왜 Tree라고 불리는가? ➔ 실제 나무를 거꾸로 엎어 놓은 것 같은 모양이기 때문.
- 비선형 자료 구조이며, 계층적인 자료를 표현하는데 적합.

1. 트리(Tree) 에서 사용되는 용어



- 1개 이상의 노드로 이루어진 유한 집합.
- A~J까지 각각을 “노드”라 한다.
- 루트 노드를 제외한 나머지 노드들 : 서브 트리

1. 노드 간의 관계



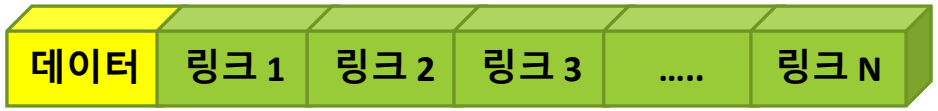
- 루트 노드 : A
- B의 자식 노드 : E, F, G
- 형제 노드 : B, C, D
- 단말 노드 : E, F, G, H, I, J
- 비 단말 노드 : A, B, C, D
- 트리의 깊이 : 3

부모 노드 : 서브 트리를 가지는 노드
형제 노드 : 부모가 같은 자식 노드
단말 노드 : 자식 노드가 없는 노드들
비 단말 노드 : 단말 노드의 반대.
차수 : 어떤 노드가 가지고 있는 자식 노드의 개수.
내부 노드 : 차수가 1 이상인 노드

레벨(Level) : 루트 노드들 로부터 깊이(루트 노드의 레벨 = 1)
조상(Ancestor) : 노드의 부모 노드들의 총 집합
자손(Descendant) : 노드의 부속 트리에 있는 모든 노드들
트리의 깊이(depth of tree) : 트리에 속한 노드의 최대 레벨
자식(child) : 부모에 속하는 부속 노드

1. 트리가 표현되는 방법

N-링크 방식 (일반적인 트리)

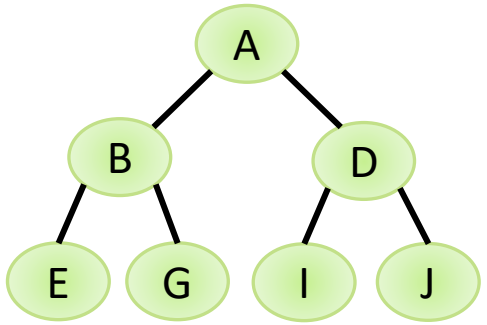


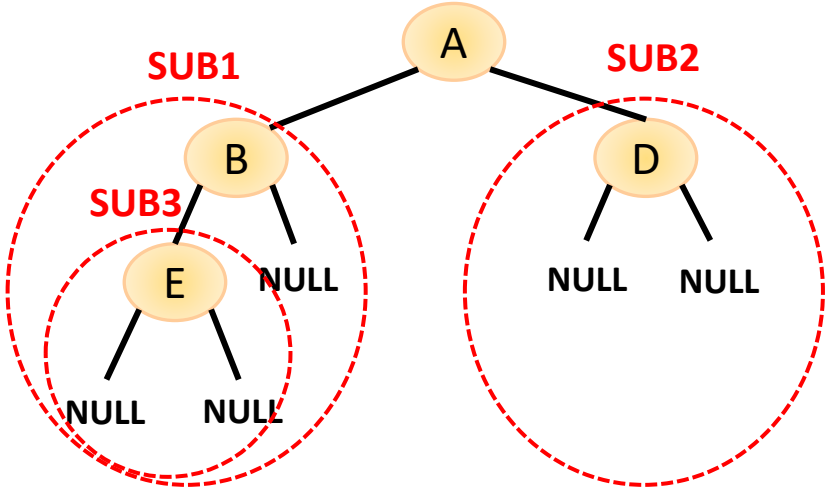
- 서로 다른 개수의 자식 노드(링크 필드)를 가지게 됨
- 노드에 따라서 링크 필드 개수가 다를 수 있음
- 자식 노드의 개수에 제한이 없다.
- **단점** : **노드의 크기가 고정되지 않음. => 복잡해짐**

이진 트리 방식

- 자식 노드 개수를 2개로 한정

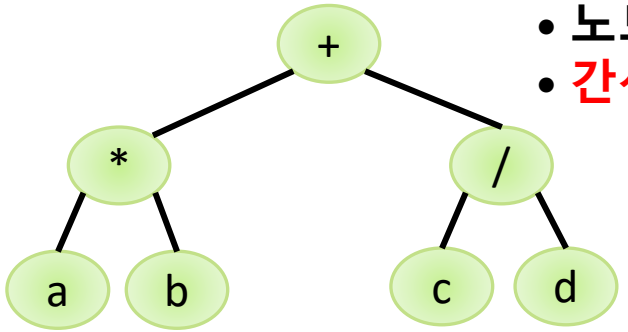
2. 이진 트리





- **모든 노드가 2개의 서브 트리**를 가지고 있는 트리
  즉, 이진 트리의 서브 트리는 모두 이진 트리 여야 한다.
- 서브 트리는 공집합(NULL)일 수 있다.
- 최대 2개의 자식 노드 소유 가능.
- 모든 노드의 차수 2 이하
- **주의** : 공집합(NULL)도 이진 트리이다.

2. 이진 트리 성질

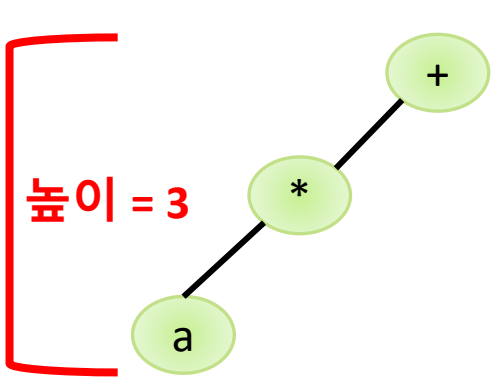


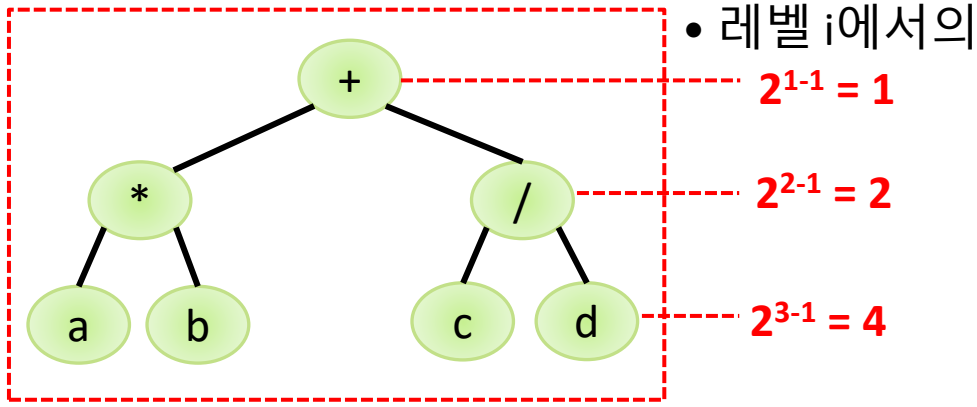
- **노드의 개수 : 7**
- **간선의 개수 : 6**

- n개의 노드를 가진 이진 트리는 **n-1개의 간선**을 가진다.
  (루트를 제외하면 정확하게 하나의 부모 노드를 가짐)
  (부모와 자식 간, 하나의 간선만이 존재)

- 높이가 h인 이진 트리의 경우, 최소 h개의 노드를 가지며, 최대(트리 전체) **2^h-1개**의 노드를 가진다.

- 레벨 i에서의 노드의 최대 개수는 **2^{i-1}.**





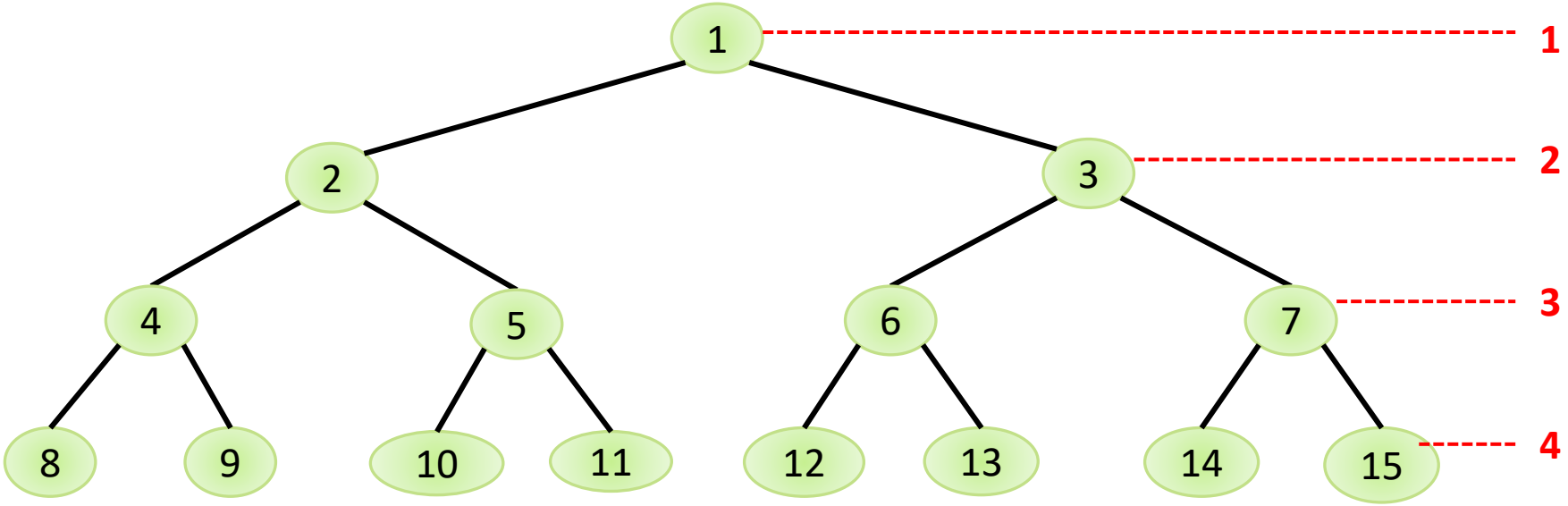
최소 노드 개수 = 3

최대 노드 개수 = $2^{1-1} + 2^{2-1} + 2^{3-1} = 1 + 2 + 4 = 7$

 # n개의 노드를 가지는 이진 트리의 높이 #

최대 : n
최소 : $[\log_2(n+1)]$

2. 포화 이진 트리



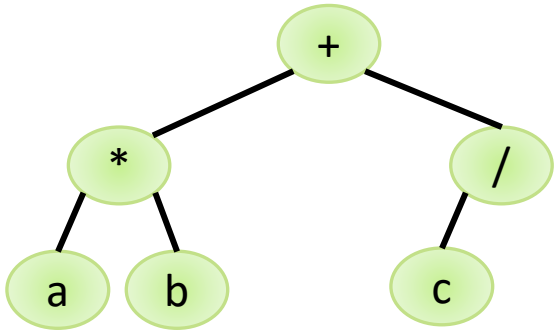
- 트리의 각 레벨에 노드가 꽉 차 있는 이진 트리

.

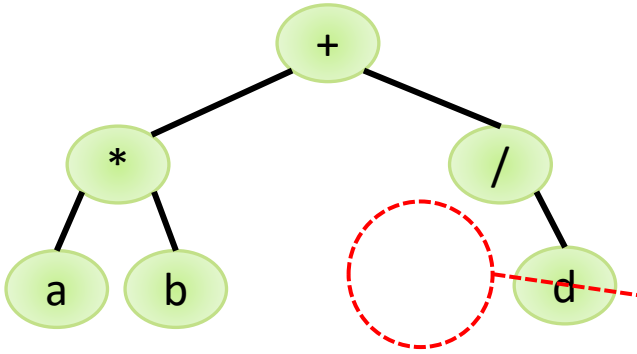
- 높이가 k인 포화 이진 트리는 2k-1개의 노드를 가짐

- **전체 노드 개수** = $2^{1-1} + 2^{2-1} + 2^{3-1} + 2^{4-1} = 2^4-1 =$ **15**

2. 완전 이진 트리



(a) 완전 이진트리



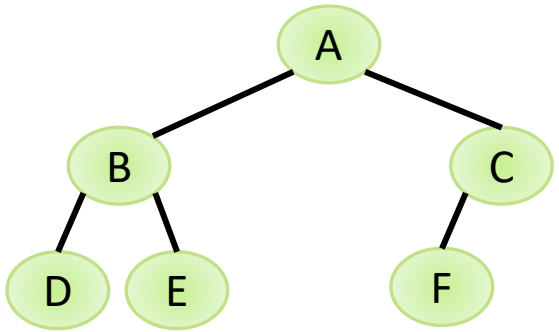
(a) 완전 이진트리가 아님

- 높이가 k일 때, 레벨 1부터 k-1까지는 노드가 모두 채워져 있고 마지막 레벨 k에서는 왼쪽부터 오른쪽으로 노드가 순서대로 채워져 있는 이진 트리.
- 마지막 레벨에서는 노드가 꽉 차 있지 않아도 되지만 중간에 빈 곳이 있어서는 안됨.

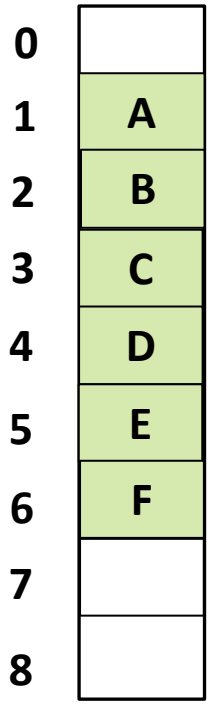
  즉, 포화 이진 트리는 항상 완전 이진 트리이지만 그 반대는 성립하지 않는다.

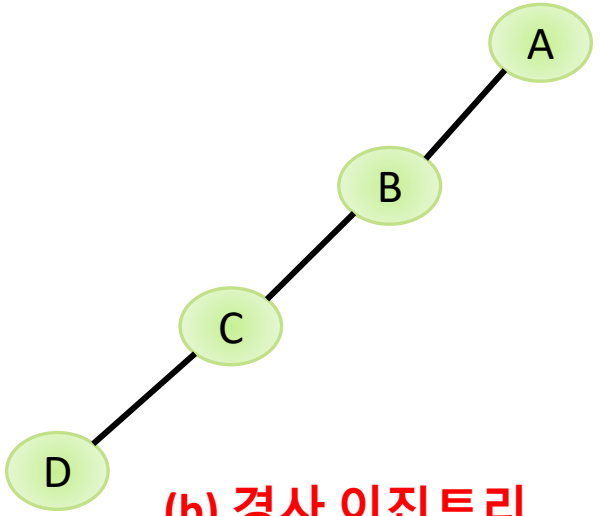
중간에 비어 있으므로 완전 이진 트리가 아님.

2. 완전 이진 트리의 배열 표현법

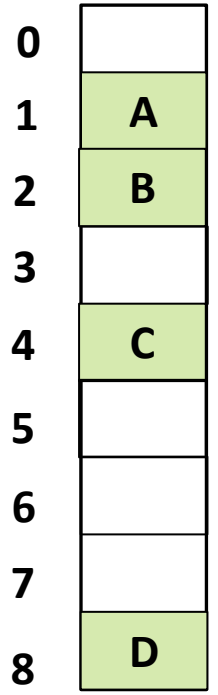


(a) 완전 이진트리



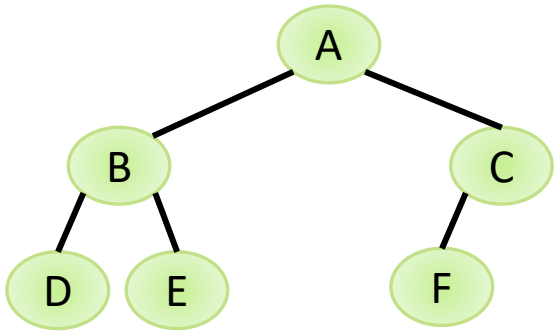


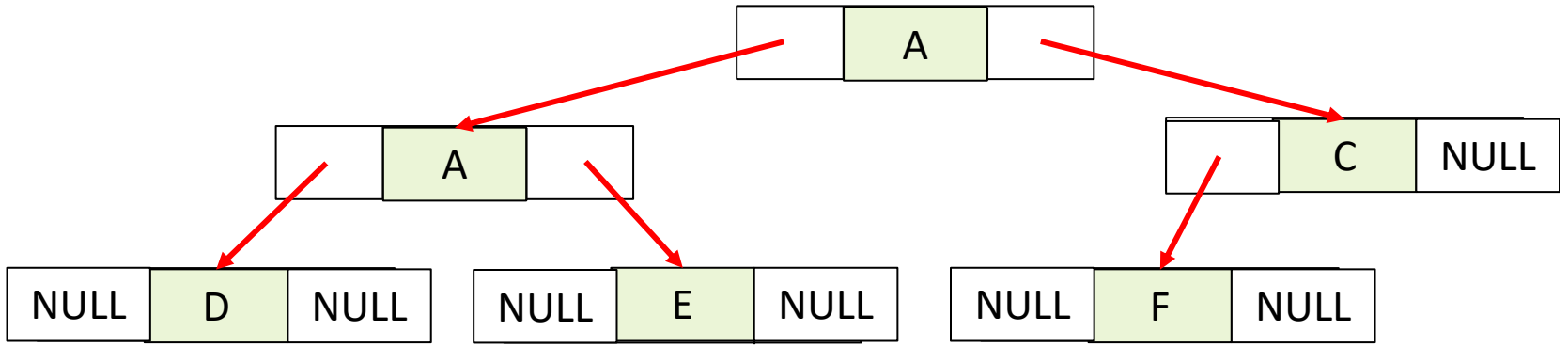
(b) 경사 이진트리



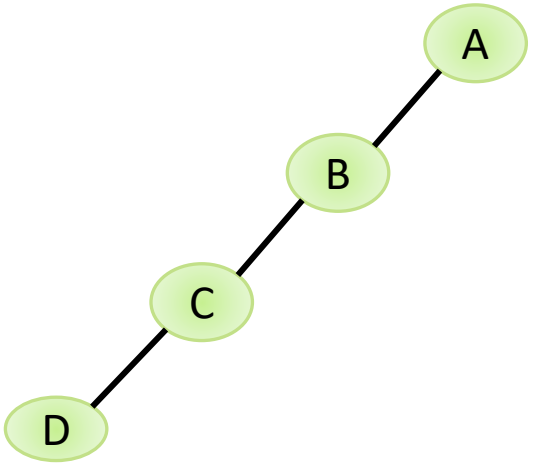
- 노드 i의 **부모 노드** 인덱스 = **i / 2**
- 노드 i의 **왼쪽 자식 노드** 인덱스 = **2i**
- 노드 i의 **오른쪽 자식 노드** 인덱스 = **2i + 1**

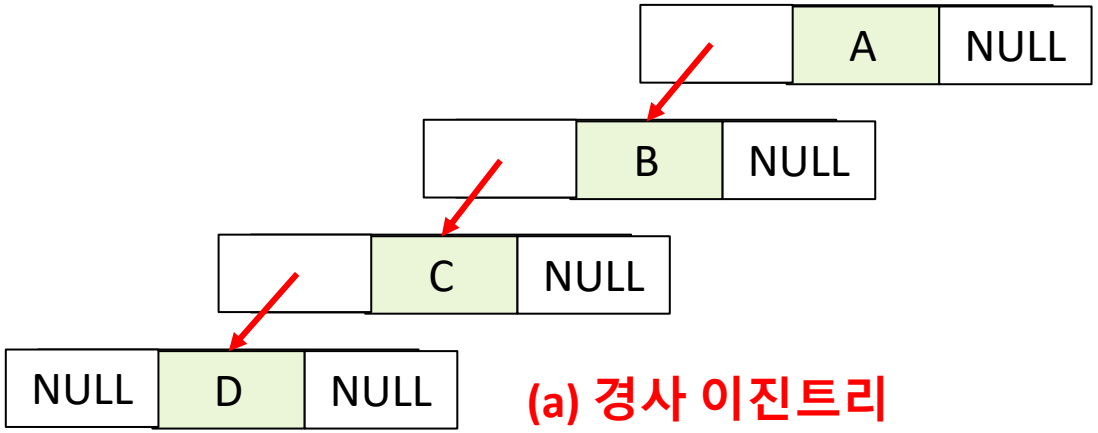
2. 이진 트리의 링크 표현법





(a) 완전 이진트리





(a) 경사 이진트리

2. 이진 트리의 링크 표현법

```
typedef struct TreeNode {
    int data;
    struct TreeNode *left, *right;
} TreeNode;
```

- 노드는 구조체로 표현한다.
- 링크는 포인터로 표현한다
- 루트 노드를 가리키는 포인터만 있으면 트리 안의 모든 노드들에 접근이 가능하다.

2. 이진 트리의 순회(Traversal)

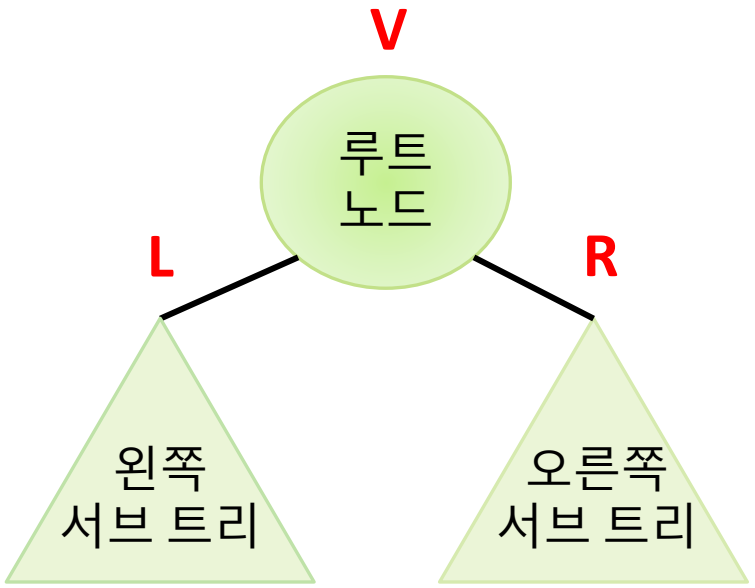
- 이진 트리에 속하는 **모든 노드를 한번 씩 방문**하는 것.

트리의 순회 방법

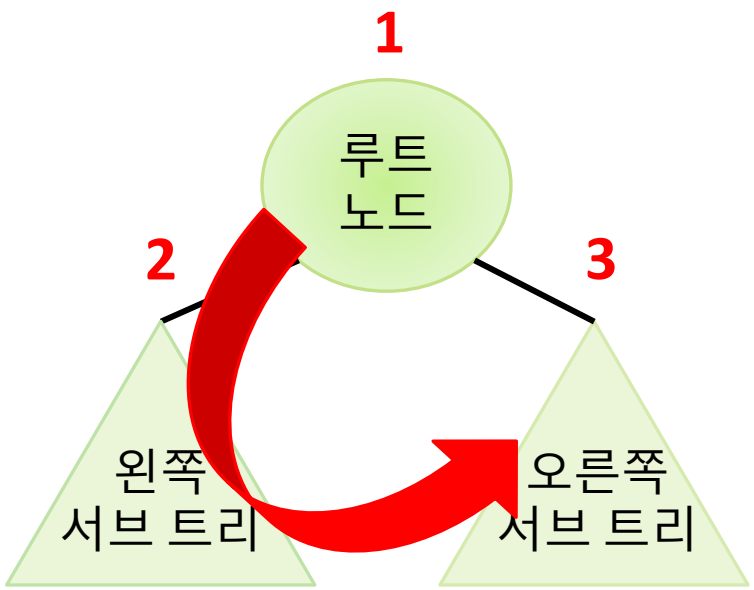
1. **전위 순회**
   => 루트를 먼저 방문

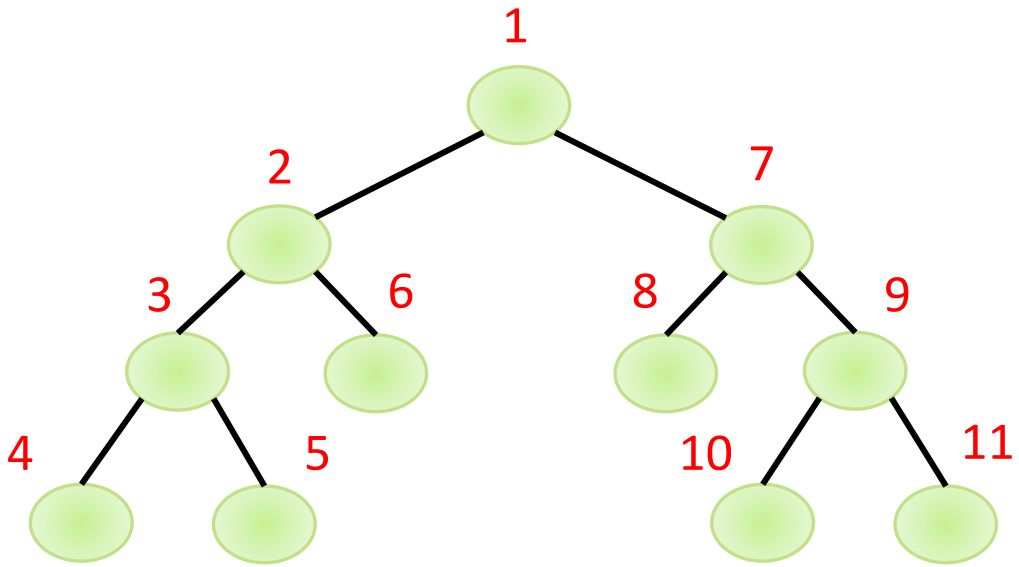
2. **중위 순회**
   => 루트를 왼쪽과 오른쪽 서브 트리 중간에 방문

3. **후위 순회**
   => 루트를 마지막에 방문.



2. 이진 트리의 전위순회(Preorder)



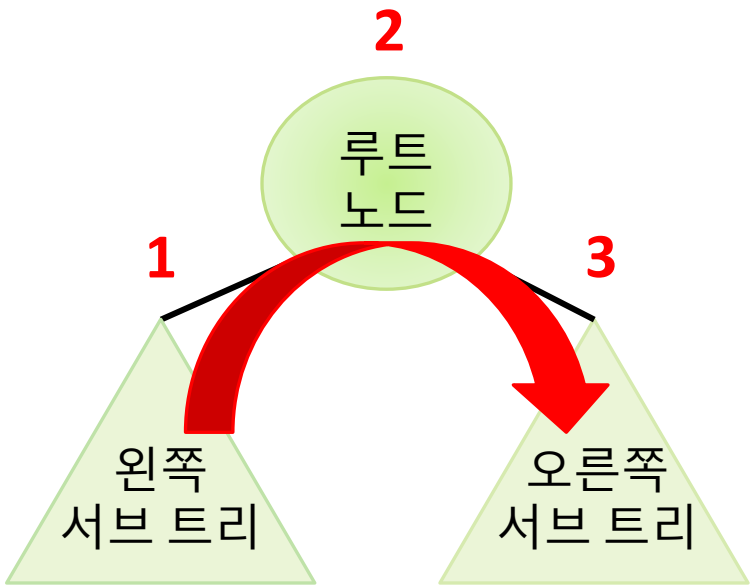


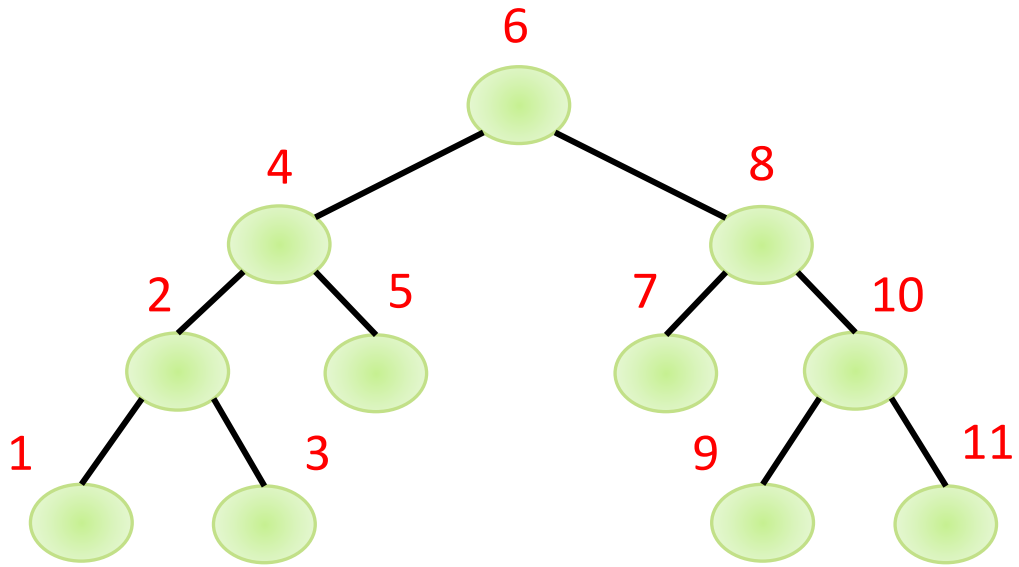
① 루트 노드를 방문한다.

② 왼쪽 서브 트리를 방문한다.

③ 오른쪽 서브 트리를 방문한다.

2. 이진 트리의 중위순회(Inorder)





① 왼쪽 서브 트리를 방문한다.

② 루트 노드를 방문한다.

③ 오른쪽 서브 트리를 방문한다.

2. 이진 트리의 중위순회(Inorder)

```c
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>

typedef struct TreeNode {
    int data;
    struct TreeNode* left, * right;
} TreeNode;

/*
        15
    4       20
1       16  25

*/

TreeNode n1 = {1, NULL, NULL};
TreeNode n2 = {4, &n1, NULL};
TreeNode n3 = {16, NULL, NULL};
TreeNode n4 = {25, NULL, NULL};
TreeNode n5 = {20, &n3, &n4};
TreeNode n6 = {15, &n2, &n5};
TreeNode* root = &n6;

void inorder(TreeNode *root)
{
    if (root)
    {
        inorder(root->left);            // 왼쪽 서브트리 순회
        printf("%d\n", root->data);     // 노드 방문
        inorder(root->right);           // 오른쪽 서브트리 순회
    }
}
```

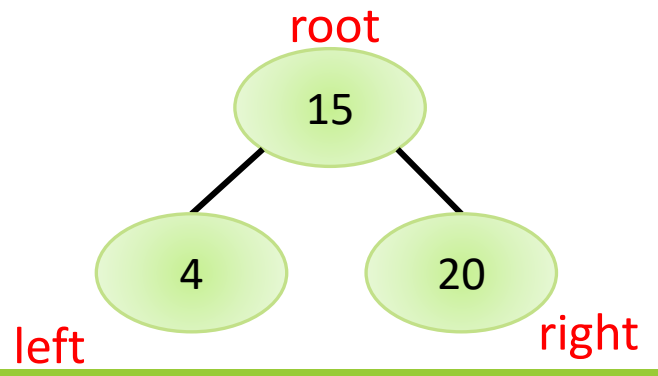
①

②

① 트리에 사용할 노드의 구조체 선언

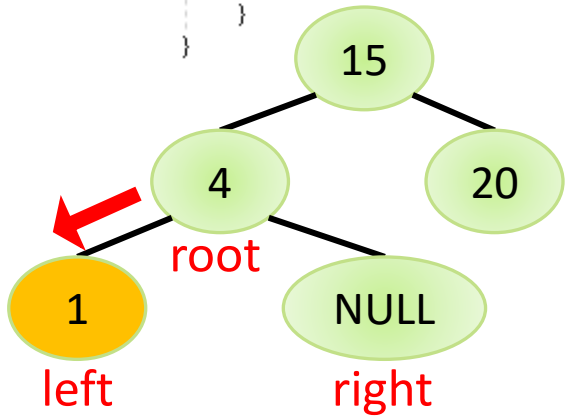
② 중위 순회 실행

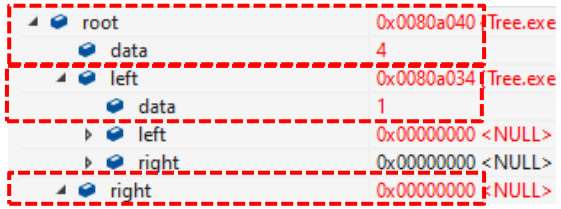
root	0x0080a070	{Tree.exe!TreeNode n6} {data=15 left=0x0080a...	TreeNode *
data	15		int
left	0x0080a040	{Tree.exe!TreeNode n2} {data=4 left=0x0080a0...	TreeNode *
data	4		int
left	0x0080a034	{Tree.exe!TreeNode n1} {data=1 left=0x000000...	TreeNode *
right	0x00000000	<NULL>	TreeNode *
right	0x0080a064	{Tree.exe!TreeNode n5} {data=20 left=0x0080a...	TreeNode *
data	20		int
left	0x0080a04c	{Tree.exe!TreeNode n3} {data=16 left=0x00000...	TreeNode *
right	0x0080a058	{Tree.exe!TreeNode n4} {data=25 left=0x00000...	TreeNode *



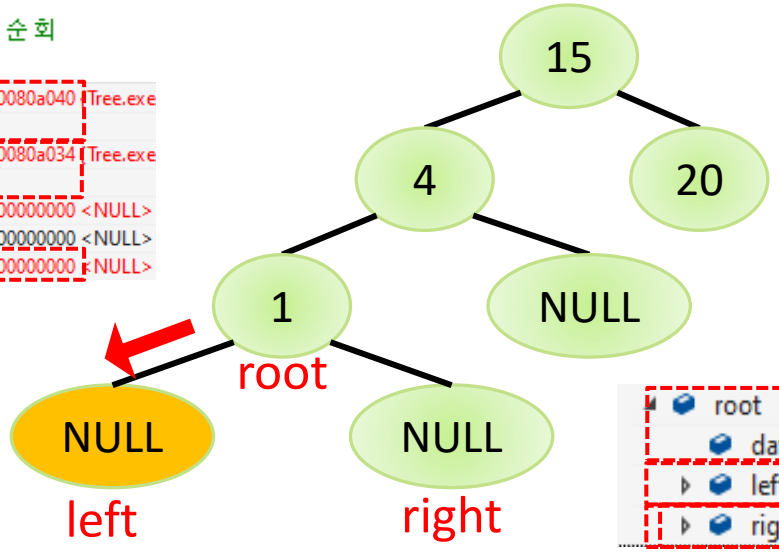
2. 이진 트리의 중위순회(Inorder)

```
void inorder(TreeNode *root)
{
    if (root)
    {
        inorder(root->left);  ①      // 왼쪽 서브트리 순회
        printf("%d\n", root->data);   // 노드 방문
        inorder(root->right);         // 오른쪽 서브트리 순회
    }
}
```





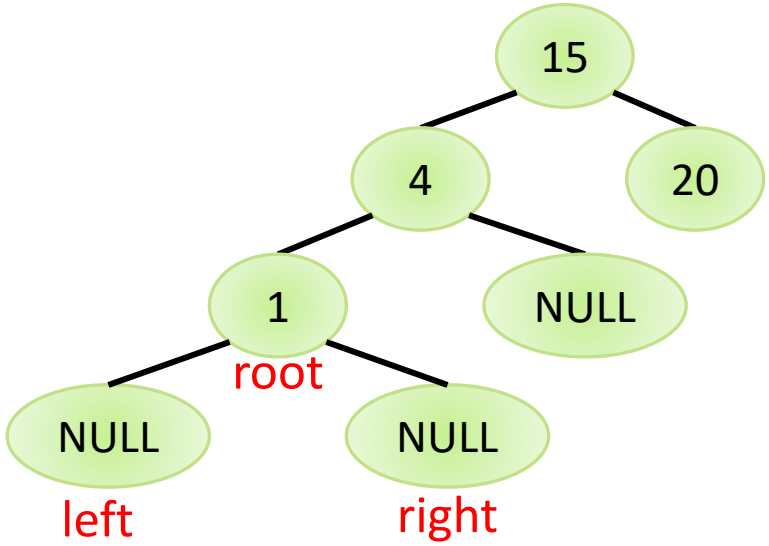
① 루트의 왼쪽 서브 트리(자식 노드)가 NULL이 아닌 '참 ' 값이라면 왼쪽 방문

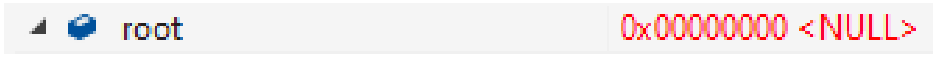


2. 이진 트리의 중위순회(Inorder)

```
void inorder(TreeNode *root)
{
    if (root)
    {
        inorder(root->left); ①          // 왼쪽 서브트리 순회
        printf("%d\n", root->data);      // 노드 방문
        inorder(root->right);            // 오른쪽 서브트리 순회
    }
}
```

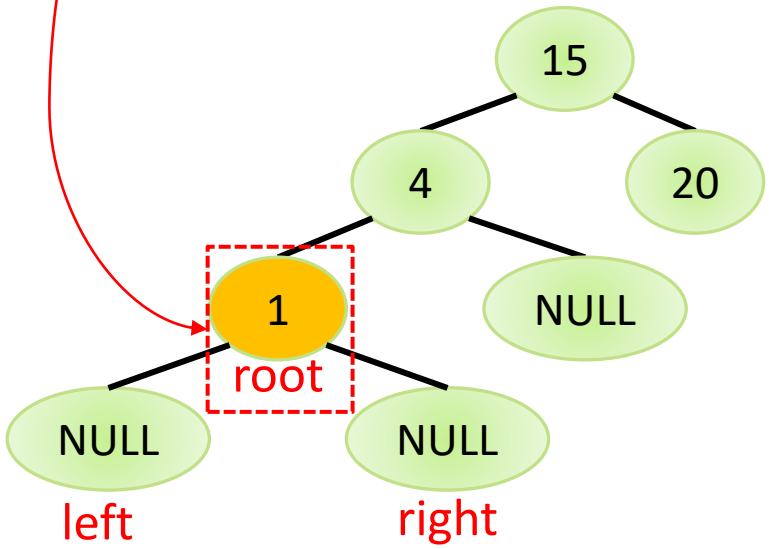
① root가 NULL이므로 return 한다.





2. 이진 트리의 중위순회(Inorder)

```
void inorder(TreeNode *root)
{
    if (root)
    {
        inorder(root->left);              // 왼쪽 서브트리 순회
        printf("%d₩n", root->data);  ②   // 노드 방문
        inorder(root->right);             // 오른쪽 서브트리 순회
    }
}
```



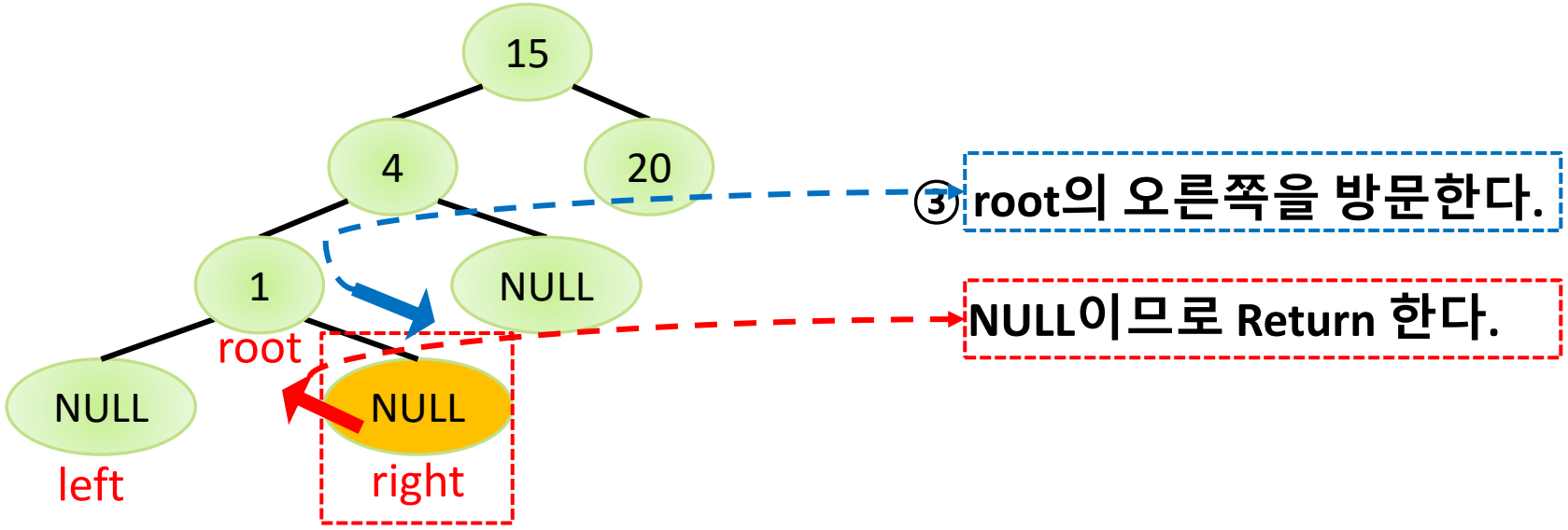
② root의 data를 출력.

```
#### 중위 순회 ####
[1]
```

2. 이진 트리의 중위순회(Inorder)

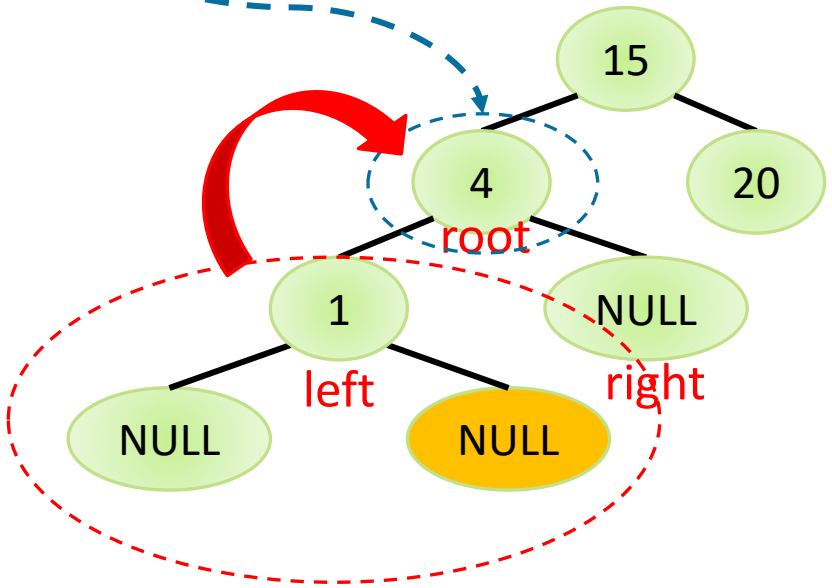
```
void inorder(TreeNode *root)
{
    if (root)
    {
        inorder(root->left);            // 왼쪽 서브트리 순회
        printf("%d₩n", root->data);     // 노드 방문
        inorder(root->right);           // 오른쪽 서브트리 순회
    }
}
```

③



2. 이진 트리의 중위순회(Inorder)

```
void inorder(TreeNode *root)
{
    if (root)
    {
        inorder(root->left);          // 왼쪽 서브트리 순회
        printf("%d₩n", root->data);   ④ // 노드 방문
        inorder(root->right);         // 오른쪽 서브트리 순회
    }
}
```



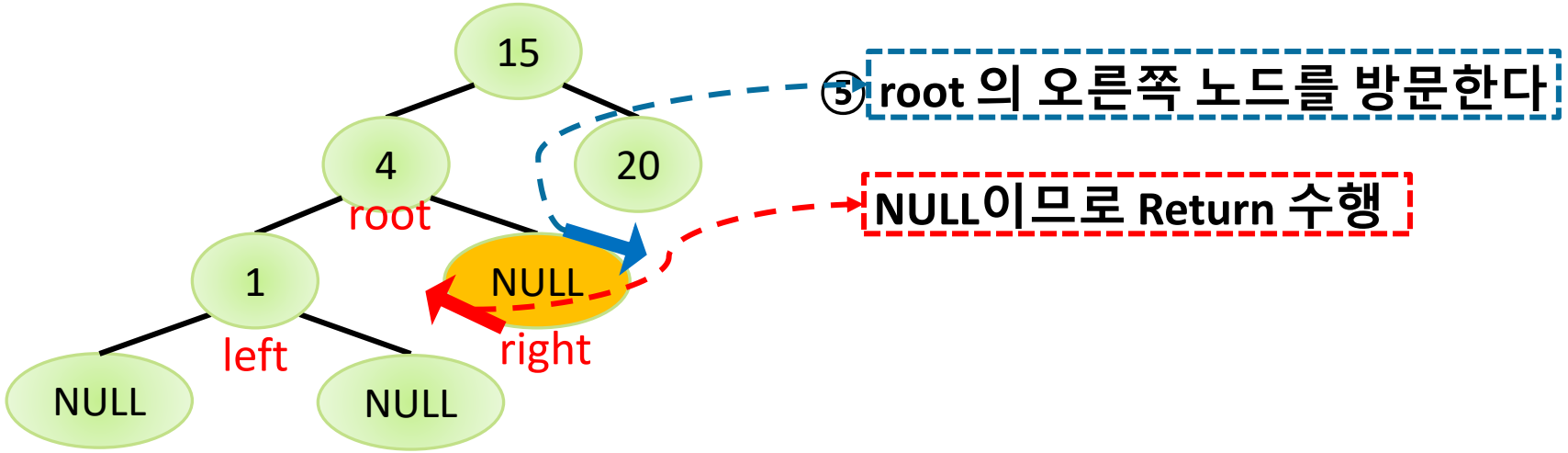
④ inorder(left) 함수 수행 후

root 노드의 data 출력

```
#### 중위 순회 ####
[1] [4]
```

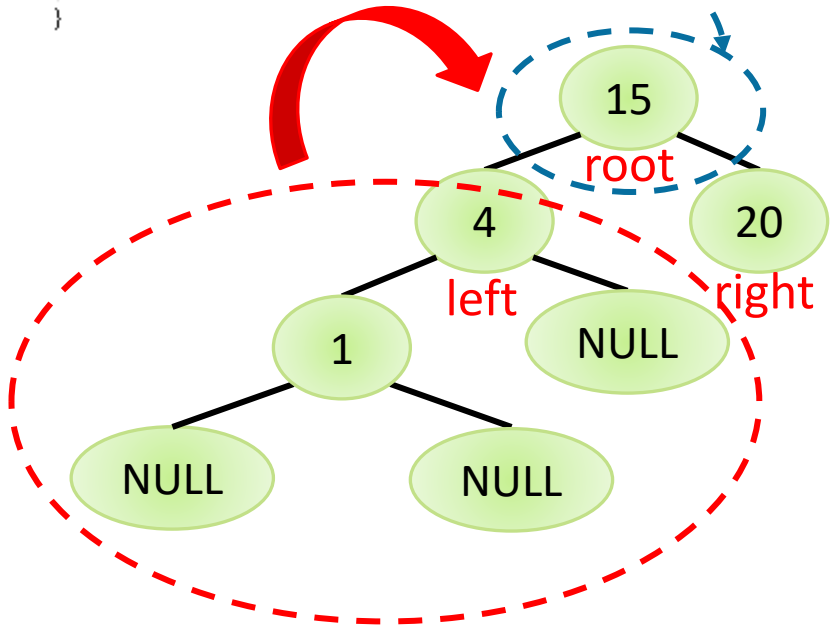
2. 이진 트리의 중위순회(Inorder)

```
void inorder(TreeNode *root)
{
    if (root)
    {
        inorder(root->left);            // 왼쪽 서브트리 순회
        printf("%d₩n", root->data);     // 노드 방문
        inorder(root->right); ⑤         // 오른쪽 서브트리 순회
    }
}
```



2. 이진 트리의 중위순회(Inorder)

```
void inorder(TreeNode *root)
{
    if (root)
    {
        inorder(root->left);          // 왼쪽 서브트리 순회
⑥       printf("%d₩n", root->data);  // 노드 방문
        inorder(root->right);         // 오른쪽 서브트리 순회
    }
}
```



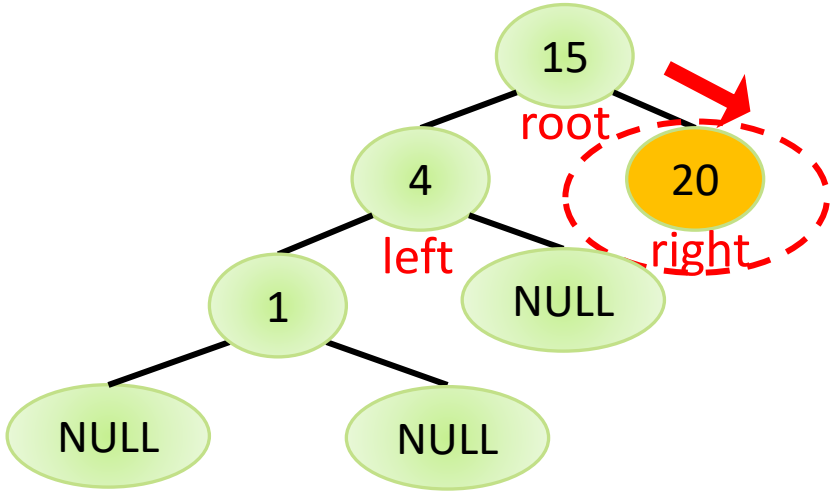
⑥ inorder(left) 수행.

root의 data를 출력

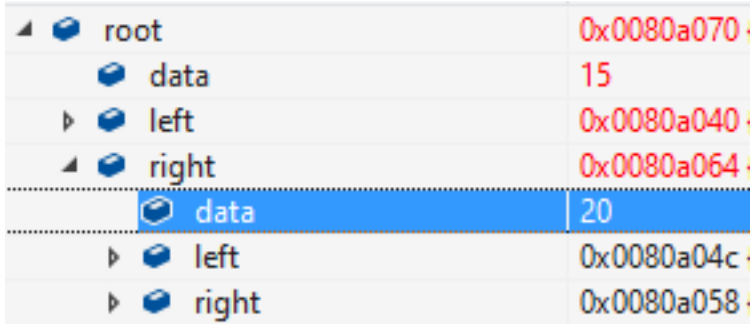
```
#### 중위 순회 ####
[1] [4] [15]
```

2. 이진 트리의 중위순회(Inorder)

```
void inorder(TreeNode *root)
{
    if (root)
    {
        inorder(root->left);            // 왼쪽 서브트리 순회
        printf("%d₩n", root->data);     // 노드 방문
        inorder(root->right);           // 오른쪽 서브트리 순회
    }
}
```



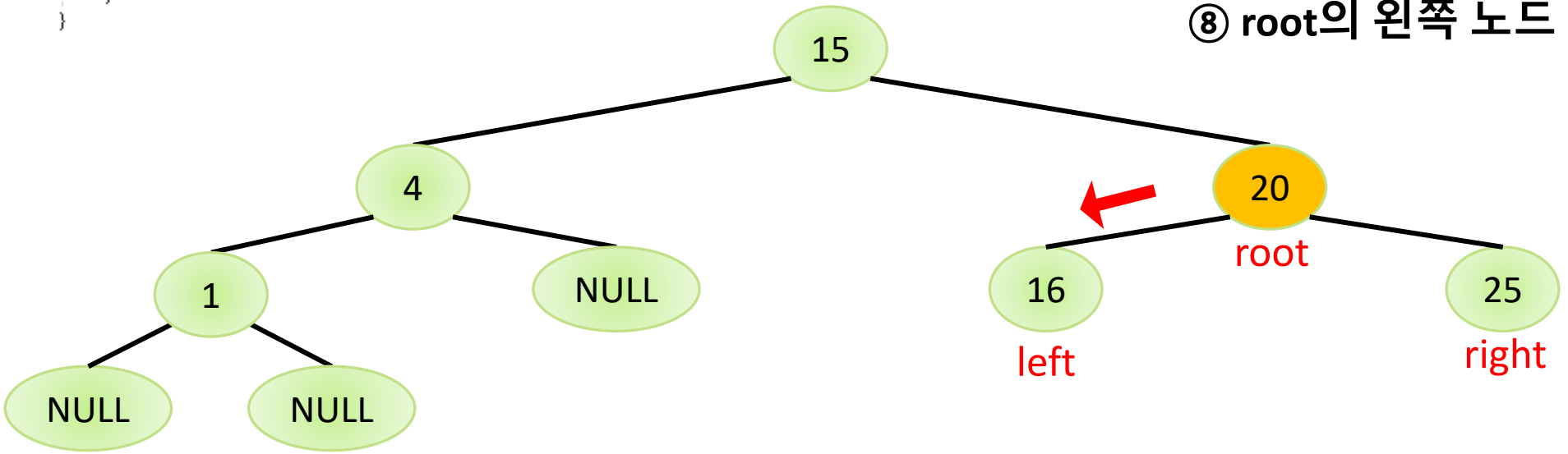
⑦ root의 오른쪽 노드를 방문.



2. 이진 트리의 중위순회(Inorder)

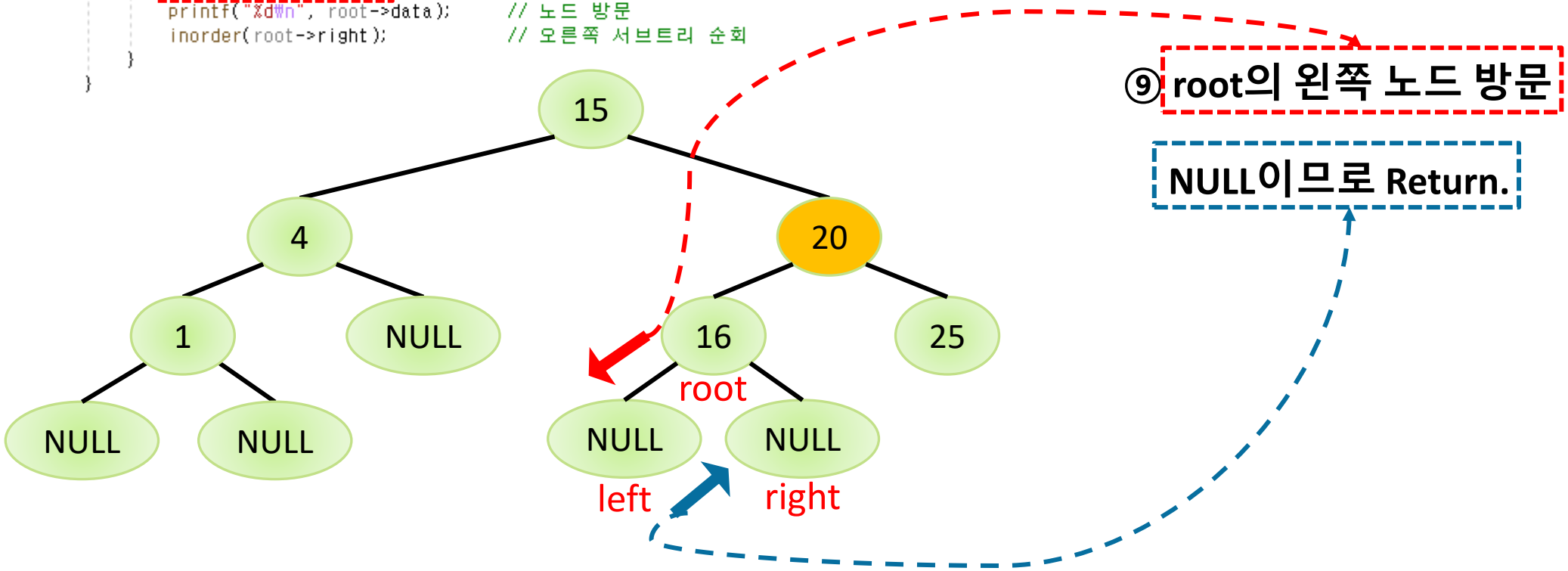
```
void inorder(TreeNode *root)
{
    if (root)
    {
        inorder(root->left);          // 왼쪽 서브트리 순회
        printf("%d\n", root->data);   // 노드 방문
        inorder(root->right);         // 오른쪽 서브트리 순회
    }
}
```

⑧ root의 왼쪽 노드 방문.



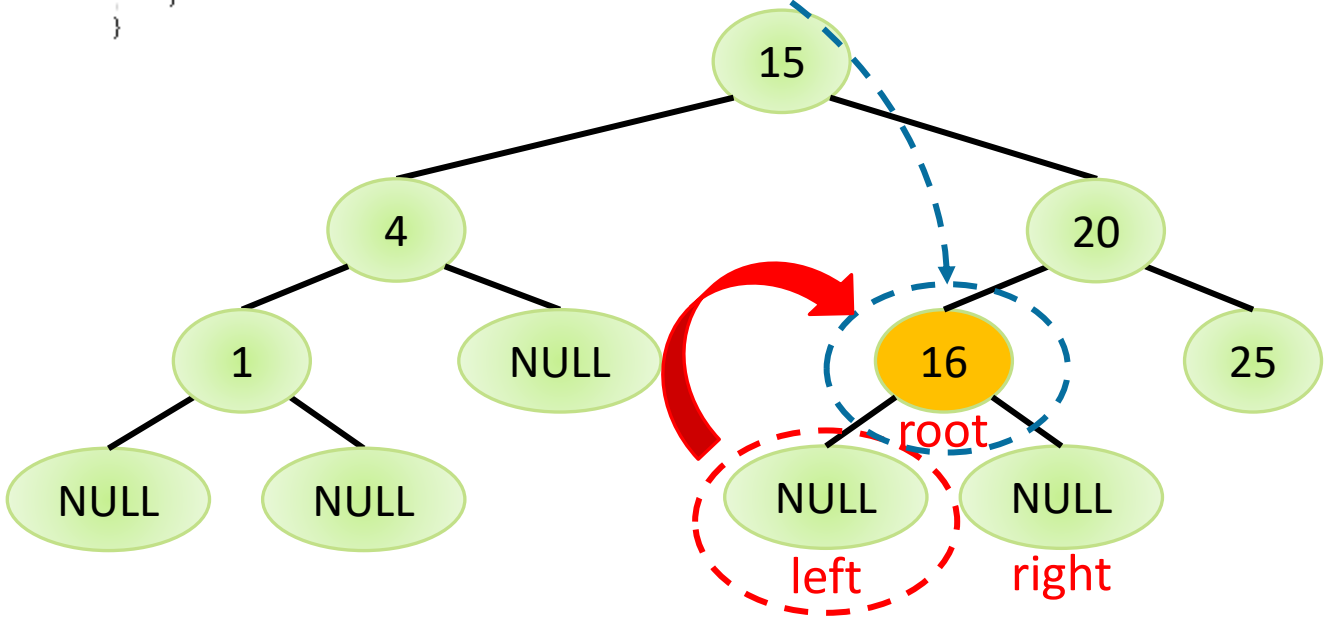
2. 이진 트리의 중위순회(Inorder)

```
void inorder(TreeNode *root)
{
    if (root)
    {
        inorder(root->left);          // 왼쪽 서브트리 순회
        printf("%d\n", root->data);   // 노드 방문
        inorder(root->right);         // 오른쪽 서브트리 순회
    }
}
```



2. 이진 트리의 중위순회(Inorder)

```
void inorder(TreeNode *root)
{
    if (root)
    {
        inorder(root->left);          // 왼쪽 서브트리 순회
        printf("%d₩n", root->data);   // 노드 방문
        inorder(root->right);         // 오른쪽 서브트리 순회
    }
}
```



⑩ 이진 트리에서 inorder(left) 수행 후

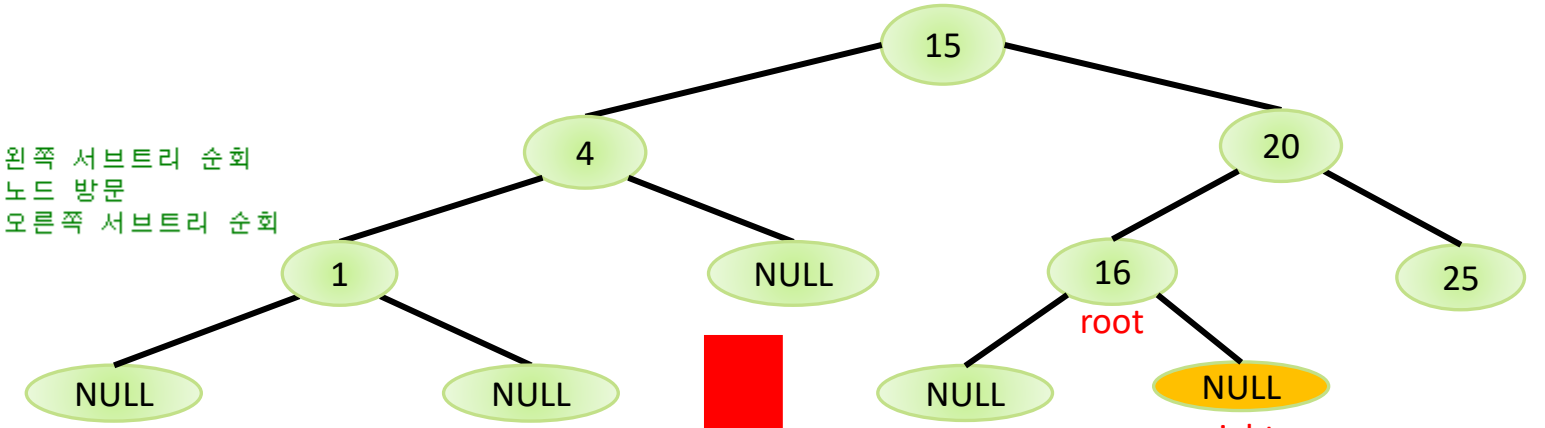
root의 data를 출력.

```
##### 중위 순회 #####
[1] [4] [15] [16]
```

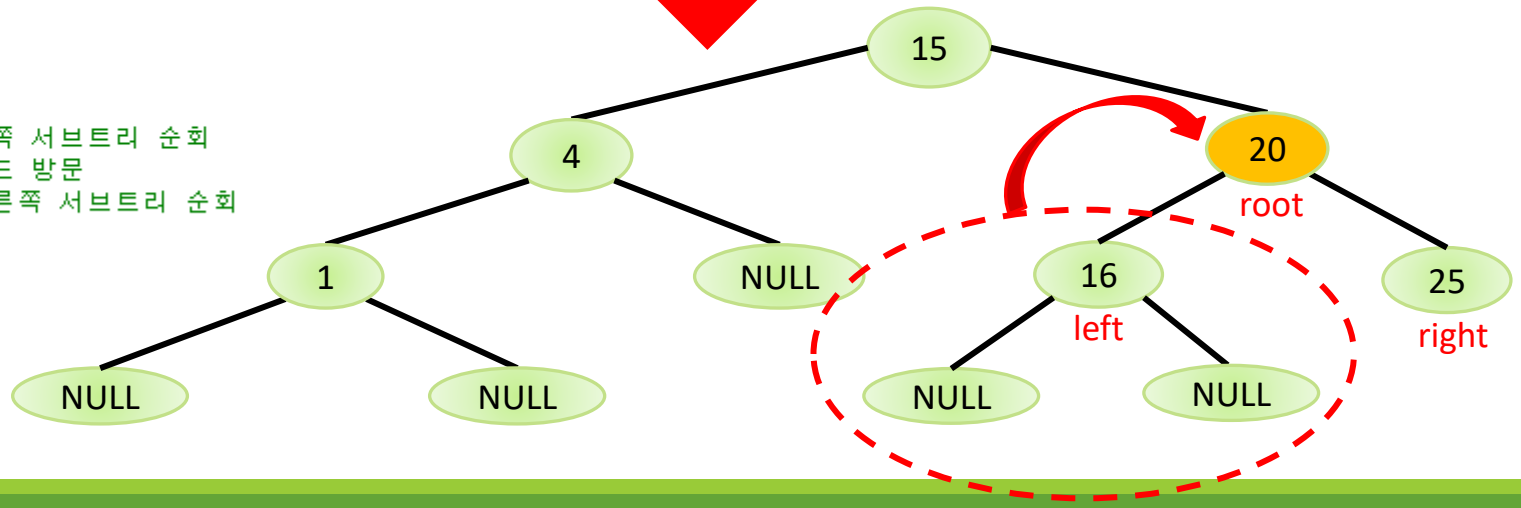
root	0x0080a04c {Tree.exe
data	16
left	0x00000000 <NULL>
right	0x00000000 <NULL>

2. 이진 트리의 중위순회(Inorder)

```
void inorder(TreeNode *root)
{
    if (root)
    {
        inorder(root->left);          // 왼쪽 서브트리 순회
        printf("%d₩n", root->data);   // 노드 방문
        inorder(root->right);         // 오른쪽 서브트리 순회
    }
}
```



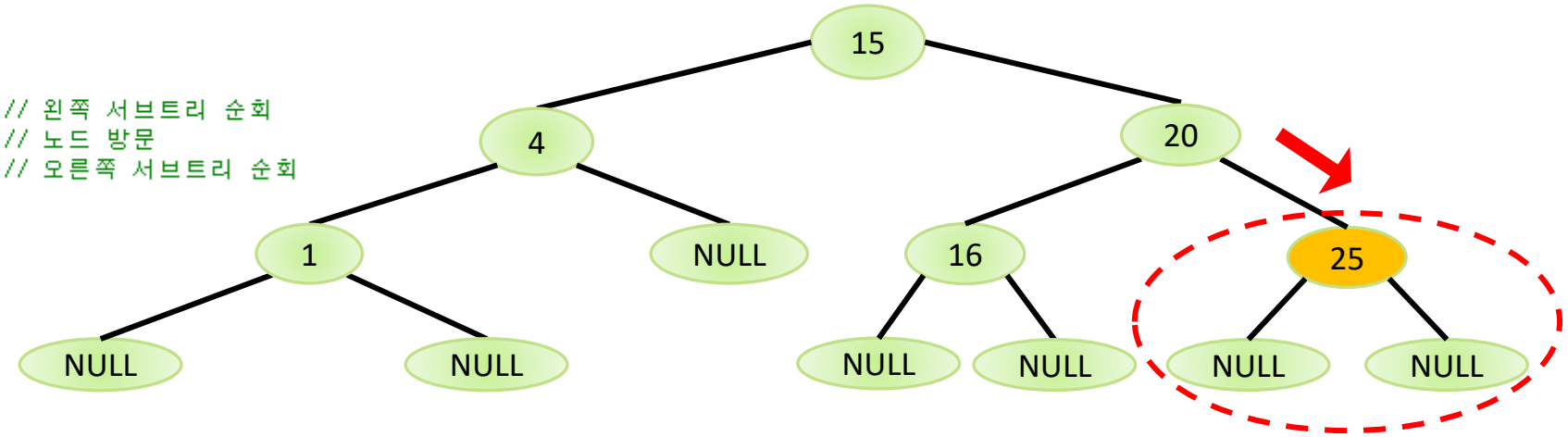

```
void inorder(TreeNode *root)
{
    if (root)
    {
        inorder(root->left);          // 왼쪽 서브트리 순회
        printf("%d₩n", root->data);   // 노드 방문
        inorder(root->right);         // 오른쪽 서브트리 순회
    }
}
```

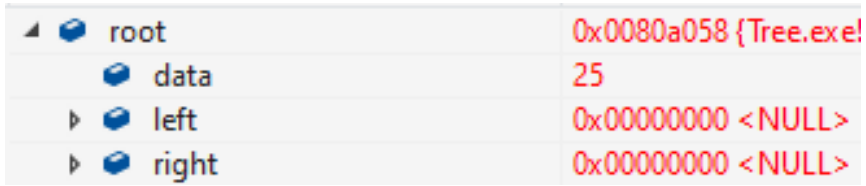



```
#### 중위 순회 ####
[1] [4] [15] [16] [20]
```

2. 이진 트리의 중위순회(Inorder)

```
void inorder(TreeNode *root)
{
    if (root)
    {
        inorder(root->left);          // 왼쪽 서브트리 순회
        printf("%d₩n", root->data);   // 노드 방문
        inorder(root->right);         // 오른쪽 서브트리 순회
    }
}
```



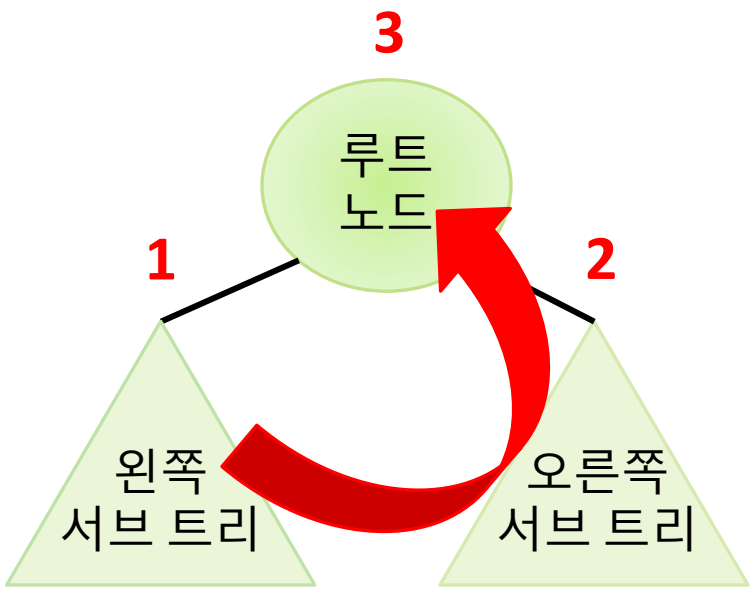


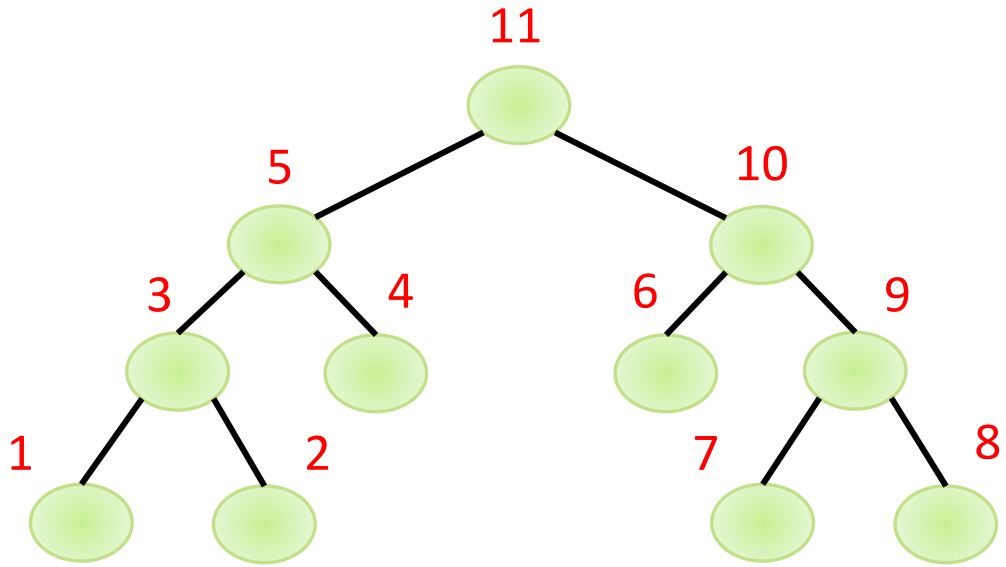

```
#### 중위 순회 ####
[1] [4] [15] [16] [20] [25]
```

root의 왼쪽 노드를 방문 후, root의 데이터를 출력한다.

그 후, 오른쪽 노드를 방문하는데 NULL이므로 Return 한다.

2. 이진 트리의 후위순회(Postorder)





① 왼쪽 서브 트리를 방문한다.

② 오른쪽 서브 트리를 방문한다.

③ 루트 노드를 방문한다.