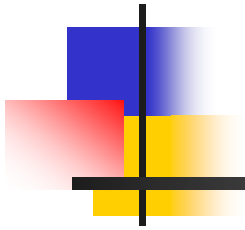
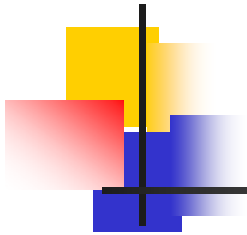


세번째 강의

트리

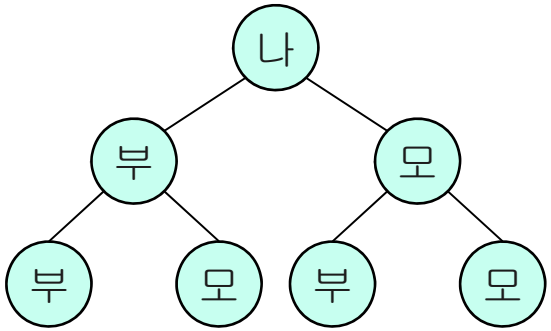


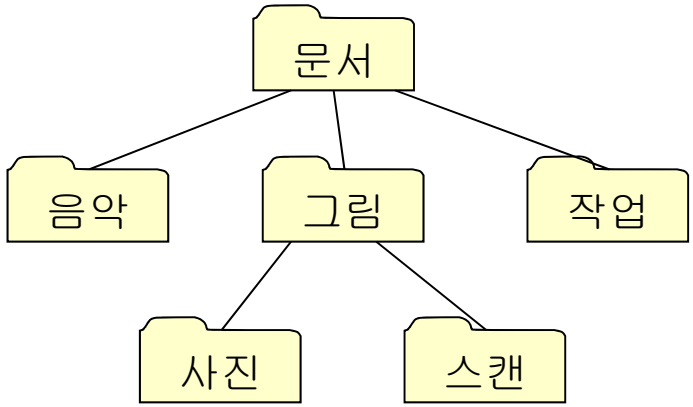
목차

- 트리**(Tree)**
- 이진 트리**(Binary Tree)**
- 이진 트리에 적용 가능한 연산
- 이진 검색트리 **(Binary Search Tree)**
- 실습
- 스스로 프로그래밍
- 첨삭 지도

1. 트리의 정의

- 트리란?
  - 계층적 구조의 자료를 표현할 때 사용
    - 가계도, 회사의 조직, 폴더 구조 등

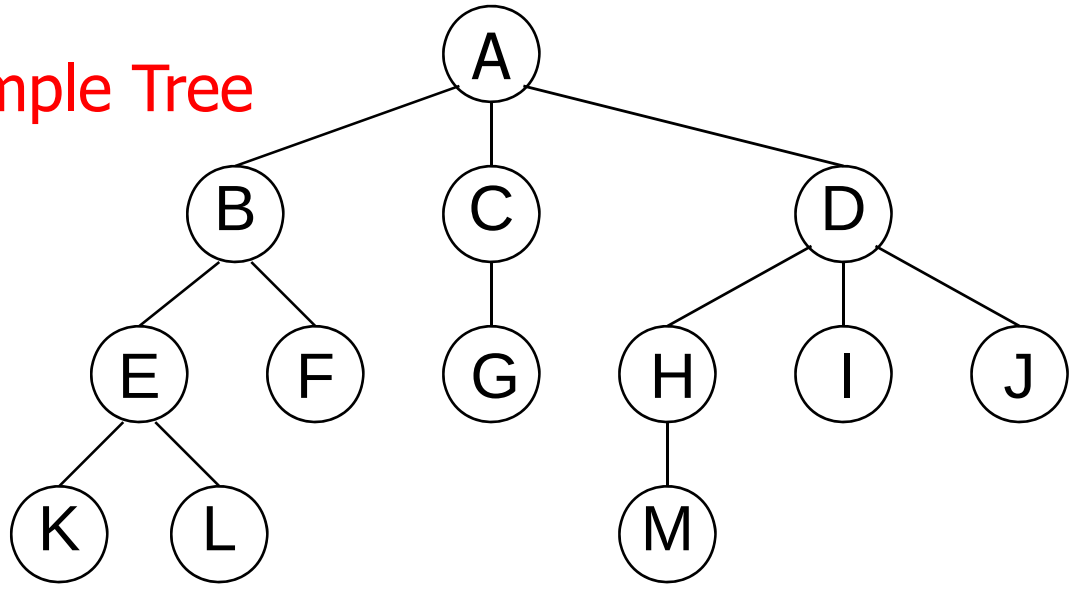




- 트리의 정의
  - 하나 이상의 노드로 이루어진 유한집합
  - Root라고 하는 노드가 하나 존재
  - 나머지 노드들은 n (≥ 0)개의 집합 $T_1, \cdots, T_n$으로 분할
    - $T_1, \cdots, T_n$: 분리된 트리 (Root의 서브 트리)

트리에 관련된 용어들

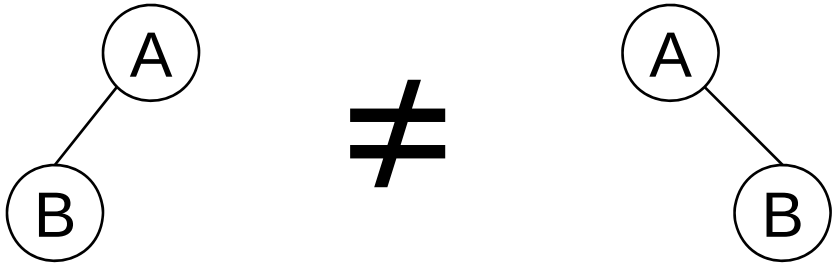
Sample Tree



- 노드의 차수(degree) (A: 3), 트리의 차수(degree) (3)
- 단말 노드(leaf or terminal node) (K, L, F, G, M, I, J)
- Parent (E: B), Children (B: E & F), Siblings (E & F)
- Ancestor (M: H, D, A), Descendants (B: E, F, K, L)
- Level (Root: 1), Height or Depth (4)

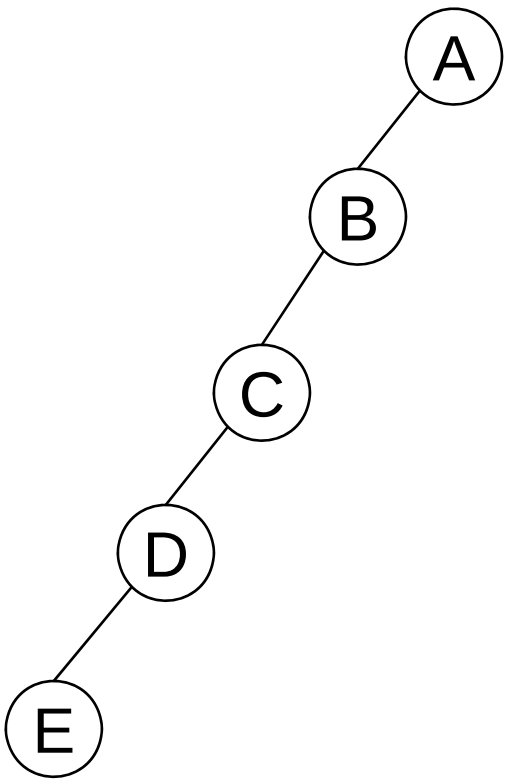
2. 이진 트리(Binary Tree)

- 이진 트리의 주요 특징
  - 모든 노드의 차수(degree)는 2를 넘지 않는다
  - 왼쪽 서브트리와 오른쪽 서브트리가 구분

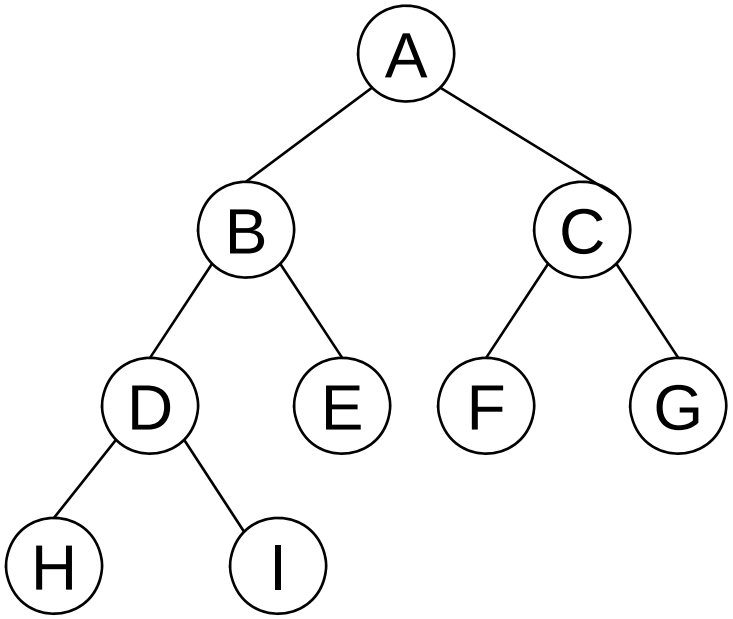


- 이진 트리의 정의
  - 유한 개의 노드들의 집합으로서
    - 노드 수는 0이 될 수 있으며,
    - 하나의 root 노드와 왼쪽 서브트리, 그리고 오른쪽 서브트리로 구성
    - 각 서브트리는 다시 이진 트리이다.

편향 트리와 완전 이진 트리



편향(skewed)
이진 트리



완전(complete)
이진 트리

Level
1
2
3
4
5

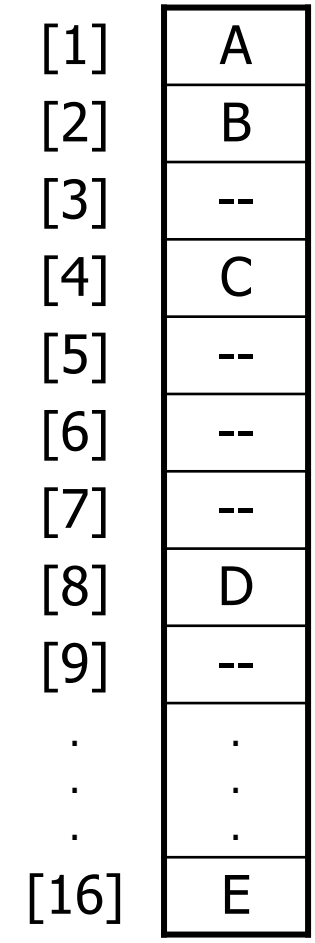
트리의 표현

- **1**차원 배열
  - parent(i) = $\lfloor i/2 \rfloor$ if i ≠ 1. If i = 1 (root), no parent.
  - lchild(i) = 2i if 2i ≤ n. If 2i > n, i has no left child.
  - rchild(i) = 2i + 1 if 2i + 1 ≤ n. If 2i + 1 > n, no right child.

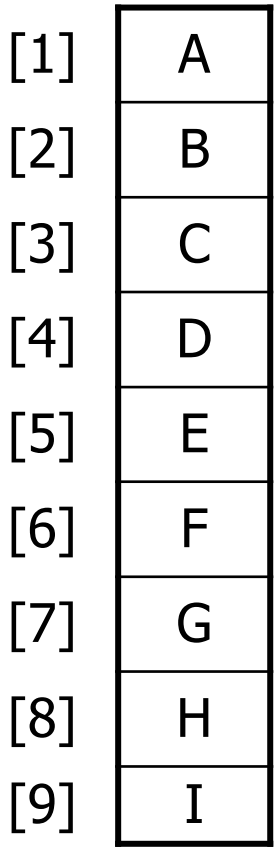
- 링크

```
struct node {
    int      data;
    struct node   *lchild;
    struct node   *rchild;
};
```

배열로 구현한 예

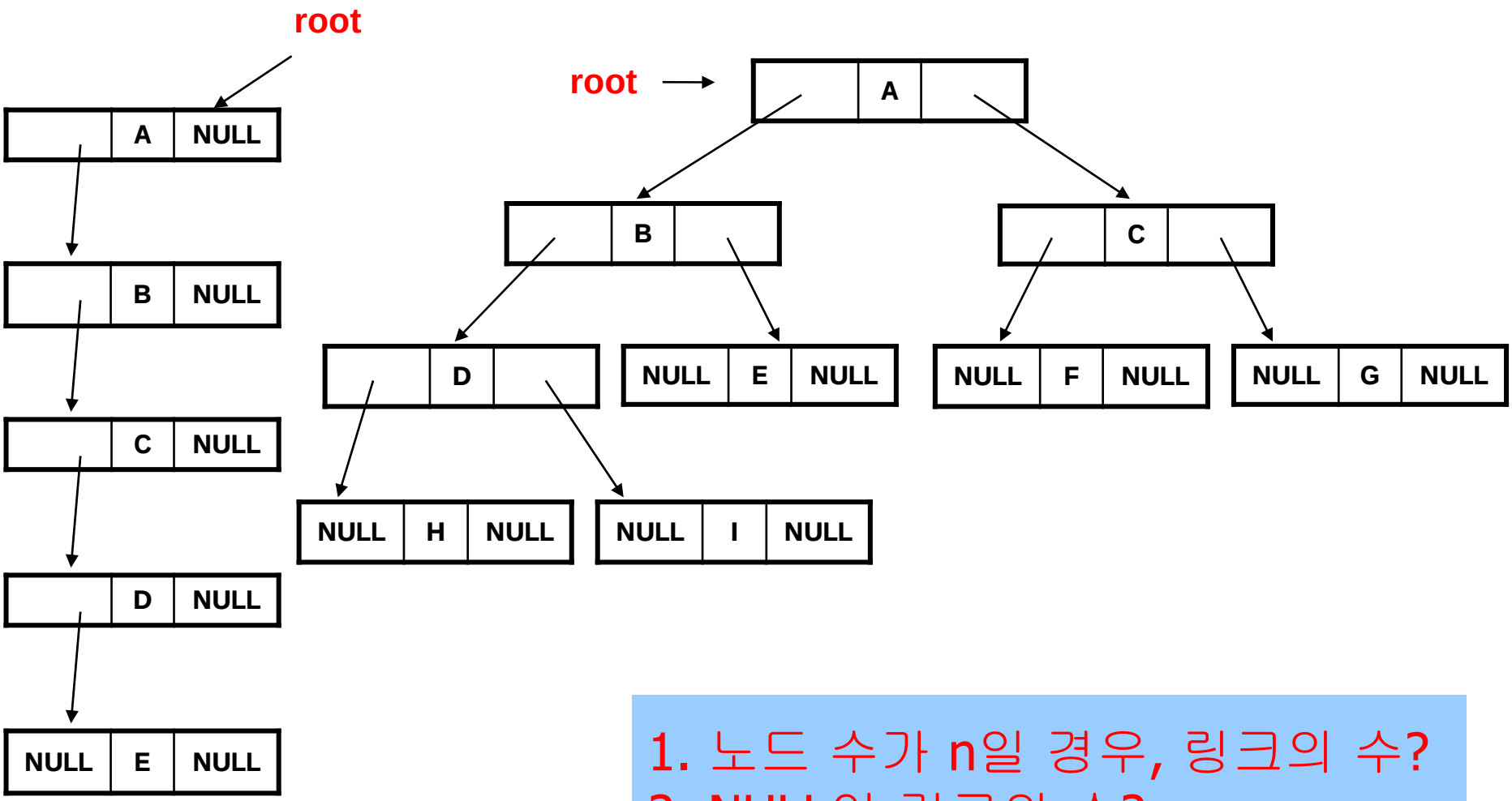


편향 이진 트리

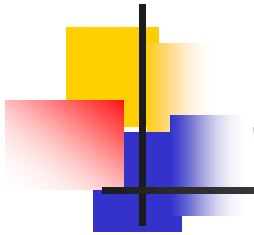


완전 이진 트리

링크로 구현한 예



1. 노드 수가 n일 경우, 링크의 수?
2. NULL인 링크의 수?



3. 이진트리에 적용가능한 연산

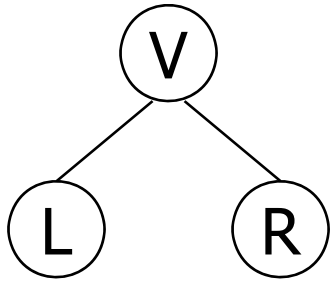
- 가장 기본적인 알고리즘: 순회
  - Inorder
  - Preorder
  - Postorder

- 순회 알고리즘의 응용
  - 이진 트리의 복사
  - 두 개의 이진 트리가 동일한지 검사
  - 이진 트리에서 노드 수 계산

이진 트리 순회(Binary Tree Traversal)

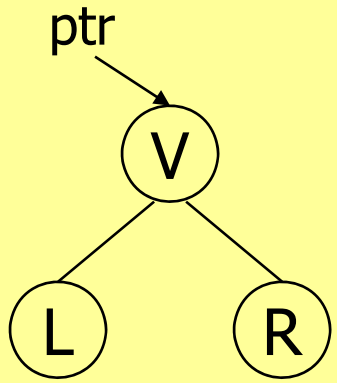
- 문제 정의
  - 이진 트리의 모든 노드를 한번씩 방문
  - 트리에 있는 노드의 순서를 결정

- 세 가지 순회 방법
  - 중위 순회(inorder traversal)
    - L → V → R
  - 전위 순회(preorder traversal)
    - V → L → R
  - 후위 순회(postorder traversal)
    - L → R → V



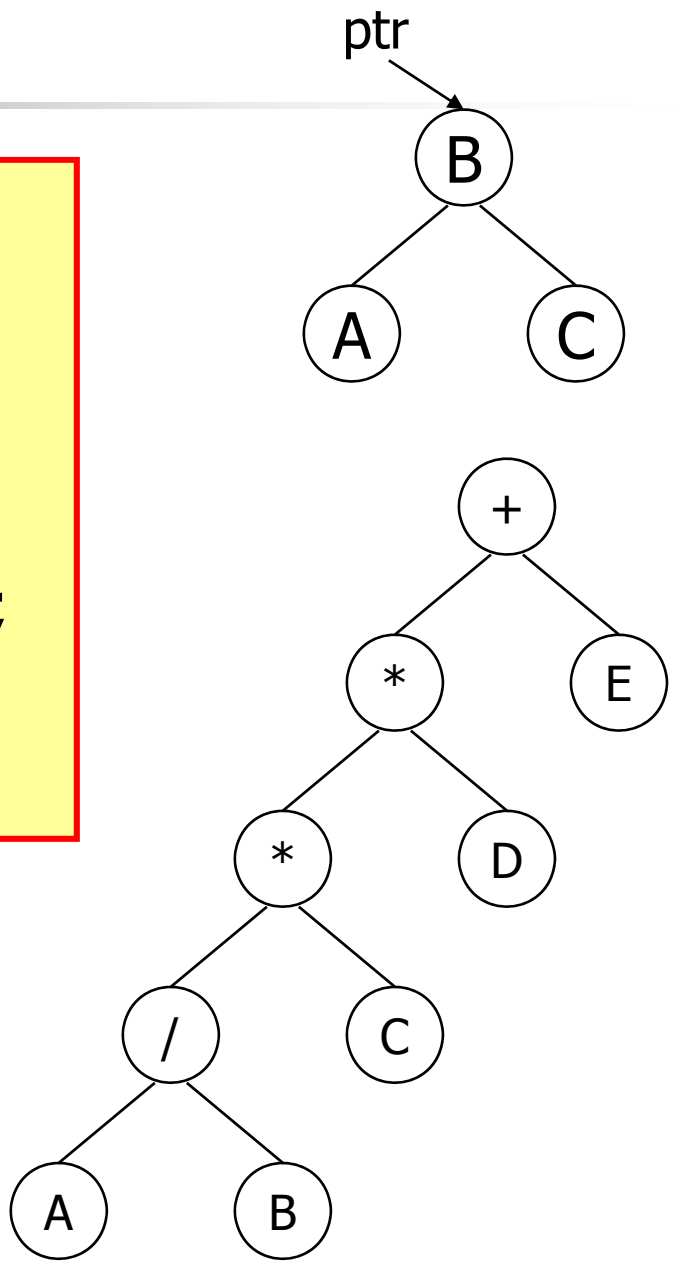
Inorder Traversal

```
void inorder(struct node *ptr)
{
    if (ptr) {
        inorder(ptr→left_child);
        printf("%d", ptr→data);
        inorder(ptr→right_child);
    }
}
```



Preorder Traversal

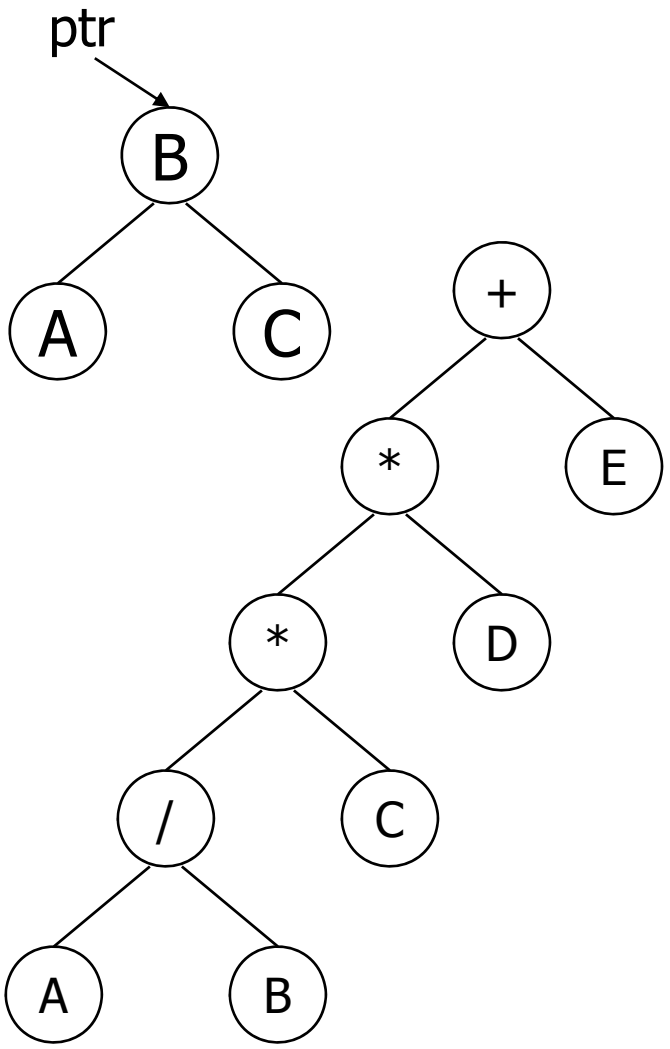
```
void preorder(struct node *ptr)
{
        if (ptr) {
            printf("%d", ptr→data);
            preorder(ptr→left_child);
            preorder(ptr→right_child);
        }
}
```



Output: + * * / A B C D E

Postorder Traversal

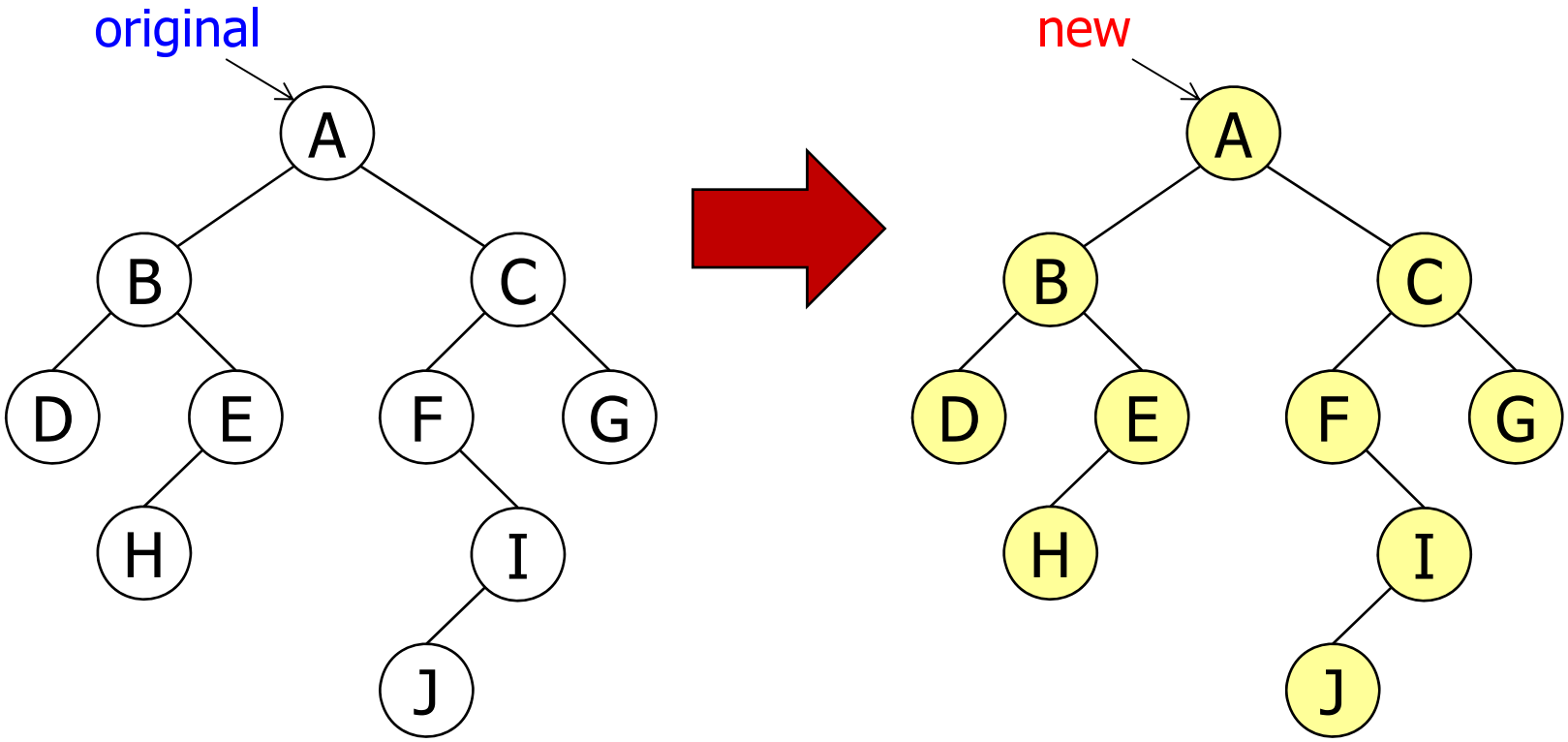
```
void postorder(struct node *ptr)
{
        if (ptr) {
            postorder(ptr→left_child);
            postorder(ptr→right_child);
            printf("%d", ptr→data);
        }
}
```



Output: A B / C * D * E +

이진 트리의 복사

- 문제 설명
  - 입력 이진 트리의 노드 구조와 동일한 새로운 이진 트리를 생성하여 루트 노드의 주소를 반환
  - 후위 순회 알고리즘을 응용

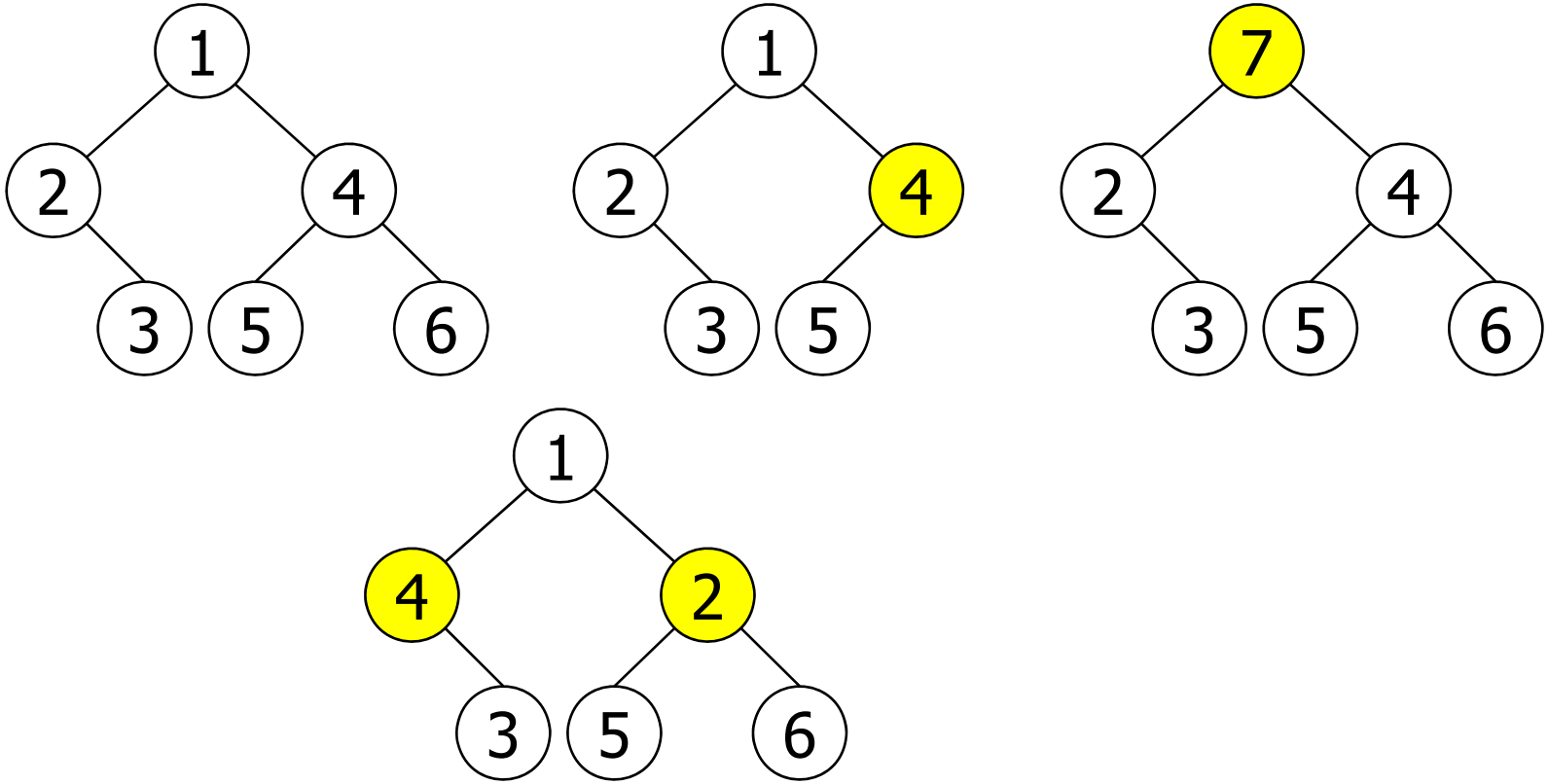


이진트리의 복사

```
tree_pointer copy(struct node *original)
{  // original 트리를 복사한 새로운 이진 트리를 반환
    struct node *temp;
    if (original) {
       temp = (struct node *) malloc(sizeof(struct node));
       temp→left_child = copy(original→left_child);
       temp→right_child = copy(original→right_child);
       temp→data = original→data;
       return temp;
    }
    return NULL;
}
```

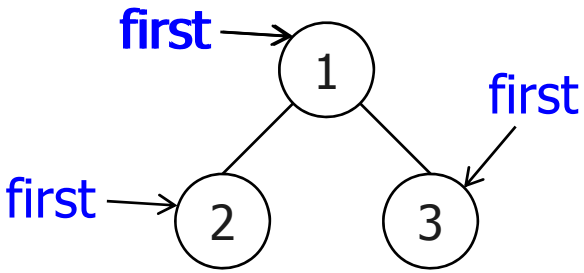
이진 트리의 동일성 검사

- 문제 설명
  - 두 개의 이진 트리가 동일한 데이터와 동일한 구조(부모-자식, 형제 등)를 갖는지를 검사
  - 전위 순회 알고리즘을 응용

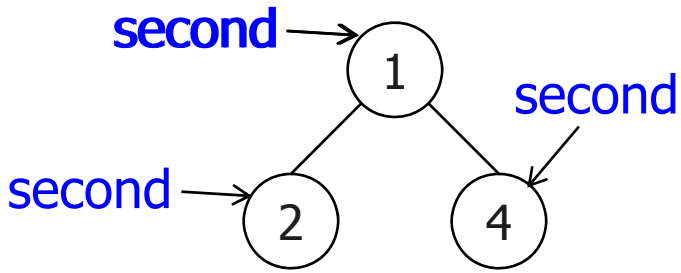


이진트리의 동일성 검사

```
int equal(struct node *first, struct node *second)
{
    /* first와 second 트리가 다를 경우 FALSE를 반환.
       트리가 동일할 경우, TRUE를 반환 */

    return ((!first && !second) ||  (first && second &&
        (first→data == second→data) &&
        equal(first→left_child, second→left_child) &&
        equal(first→right_child, second→right_child)));
}
```



first = NULL



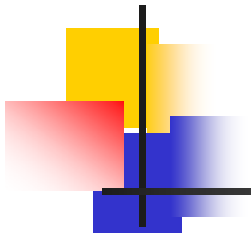
second = NULL

이진 트리의 노드 수 계산

- 접근 방법
  - 루트 노드가 NULL이면 0을 반환
  - NULL이 아니면, "1+ 왼쪽 서브트리의 노드 수 + 오른쪽 서브트리의 노드 수" 를 반환
    - 서브트리의 노드 수? → 순환 알고리즘

```
int get_node_count(struct node *ptr)
{
    int count = 0
    if (ptr != NULL)
        count = 1 + get_node_count(ptr→left_child) +
                    get_node_count(ptr→right_child);
    return count;
}
```

응용 문제: 트리의 높이 계산



이진 트리의 단말 노드 수 계산

- 접근 방법
  - 루트 노드가 NULL이면, 0을 반환
  - 단말 노드이면, 1을 반환
  - 자식이 있을 경우, "왼쪽 서브트리의 단말 노드 수 + 오른쪽 서브트리의 단말 노드 수" 를 반환
    - 서브트리의 단말 노드 수? → 순환 알고리즘

단말 노드 수 계산 알고리즘

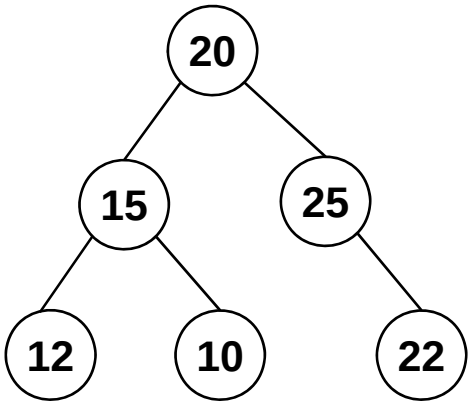
```
int get_leaf_count(struct node *ptr)
{
    int count = 0;

    if (ptr != NULL) {
        if (ptr→left_child == NULL &&
            ptr→right_child == NULL)        // 단말 노드
                return 1;
        else  count = get_leaf_count(ptr→left_child) +
                      get_leaf_count(ptr→right_child);
    }
    return count;
}
```

4. 이진 검색 트리(Binary Search Trees)

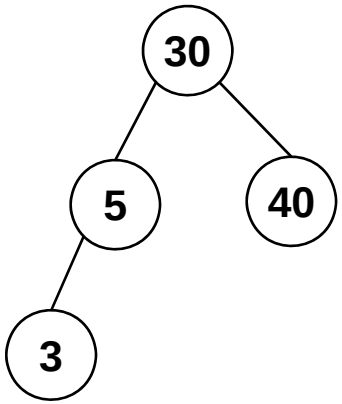
- 이진 검색 트리란?
  - 트리 내에서 특정 데이터(키 값)를 효율적으로 찾도록 하는 트리

- 정의 [이진 검색 트리]
  - 모든 노드는 유일한 키 값을 가지고 있다.
  - 왼쪽 서브트리에 저장된 키 값 < Root node의 키 값
  - 오른쪽 서브트리에 저장된 키 값 > Root node의 키 값
  - 왼쪽/오른쪽 서브트리도 이진 검색 트리이다.

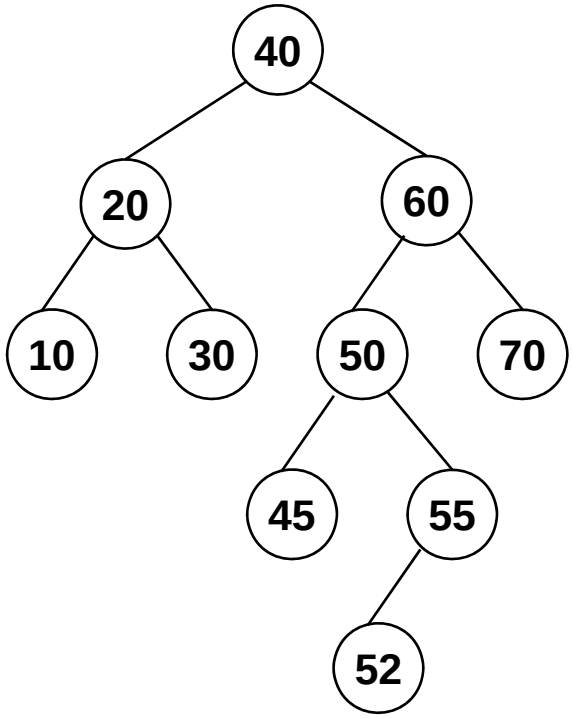
이진 트리와 이진 검색 트리



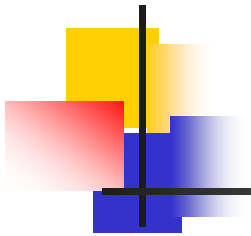
(a)



(b)



(c)



이진 검색 트리에서 검색 연산

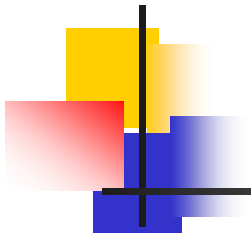
- 기본 개념
  - If (key == root→key) return(root)
  - If (key < root→key) search(root→left_child)
  - If (key > root→key) search(root→right_child)

Recursive Search

```
struct node *search (struct node *root, int key)
{
   // key를 포함하고 있는 노드의 포인터를 return
   // 해당되는 노드가 없을 경우, return NULL.
   // Recursive version

   if (!root) return NULL;
   if (key == root→data) return root;
   if (key < root→data)
         return search (root→left_child, key);
   return search (root→right_child, key);
}
```

Iterative Search

```
struct node *iterSearch(struct node *tree, int key)
{
   // key를 포함하고 있는 노드의 포인터를 return
   // 해당되는 노드가 없을 경우, return NULL.
   // Iterative version

        while (tree) {
                if (key == tree→data) return tree;
                if (key < tree→data)
                        tree = tree→left_child;
                else
                        tree = tree→right_child;
        }
        return NULL;
}
```

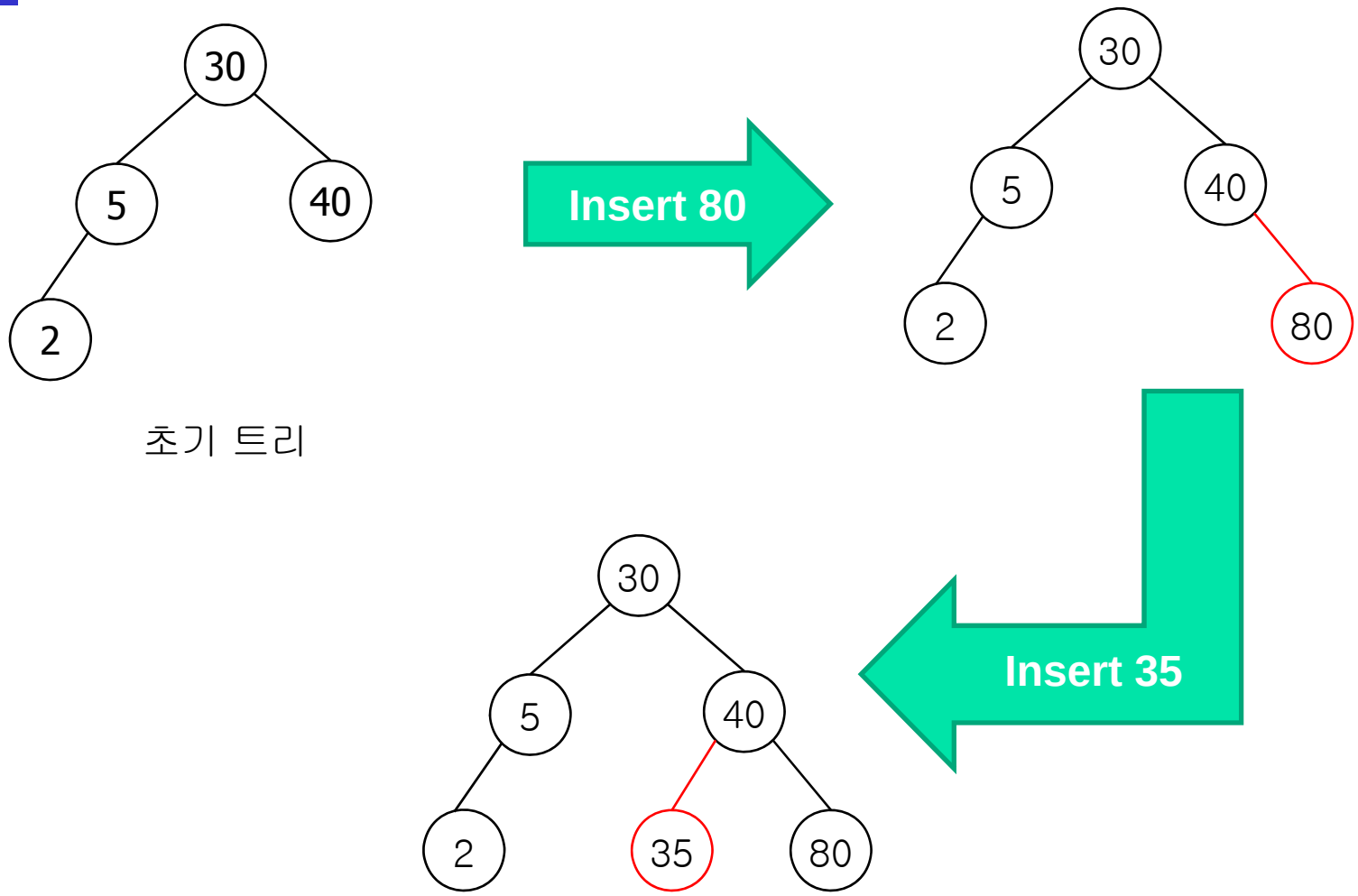


이진 검색 트리에 노드 추가

- 기본 개념
  - 추가할 키 값이 트리에 이미 존재하는지 확인
    - 키의 유일성을 보장하여야 함.
  - 검색 알고리즘 수행 후, 알고리즘이 종료되는 곳에 새로운 노드 추가

- modified_search(root, key)
  - key가 존재할 경우, return(NULL)
  - Otherwise, 검색 알고리즘에서 방문한 마지막 노드에 대한 포인터를 return

노드 추가의 예

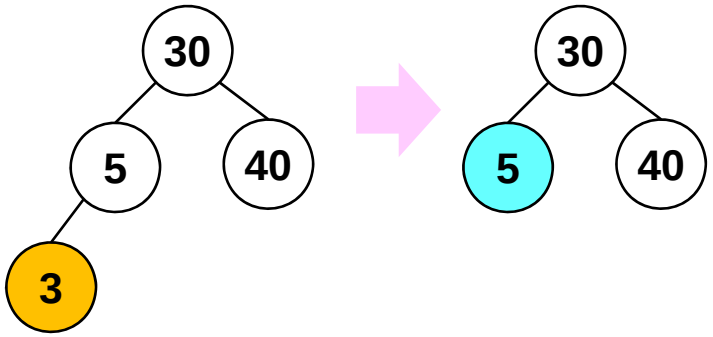


이진 검색트리에 노드 추가

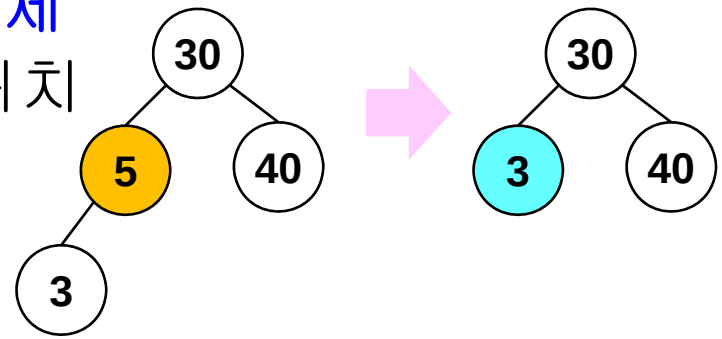
```
void insert_node (struct node **root, int num)
{   // node: root, num: 추가할 키 값.

    struct node *ptr, parent = modified_search (*root, num);

    if (parent || !(*root)) {    // num이 tree에 존재하지 않음.
        ptr = (struct node *) malloc(sizeof(struct node));
        ptr→data = num;  // num을 키 값으로 하는 노드 생성
        ptr→left_child = ptr→right_child = NULL;

        if (*root)      // parent의 child node로 삽입
            if (num < parent→data)
                parent→left_child = ptr;
            else
                parent→right_child = ptr;
        else *root = ptr;    // 트리가 empty일 경우, root로 등록
    }
}
```

이진 검색 트리에서 노드 삭제

- 리프 노드의 삭제
  - parent→left_child = NULL

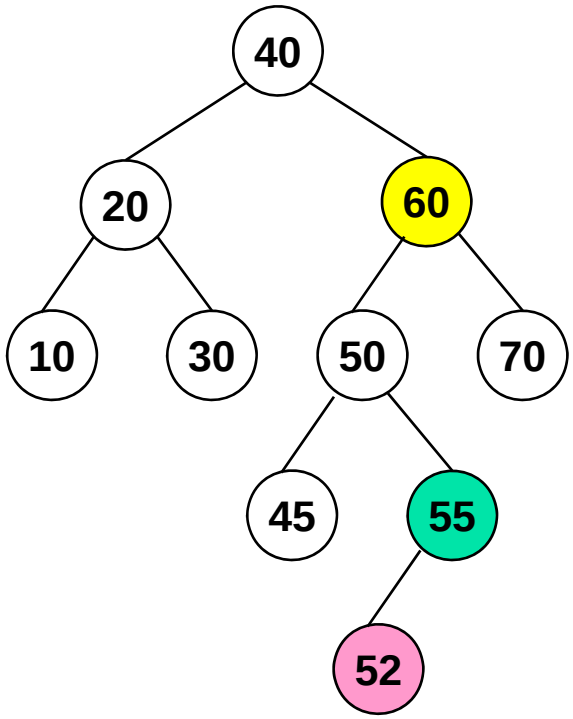


- **Child**가 하나밖에 없는 노드의 삭제
  - 삭제된 자리에 child node를 위치

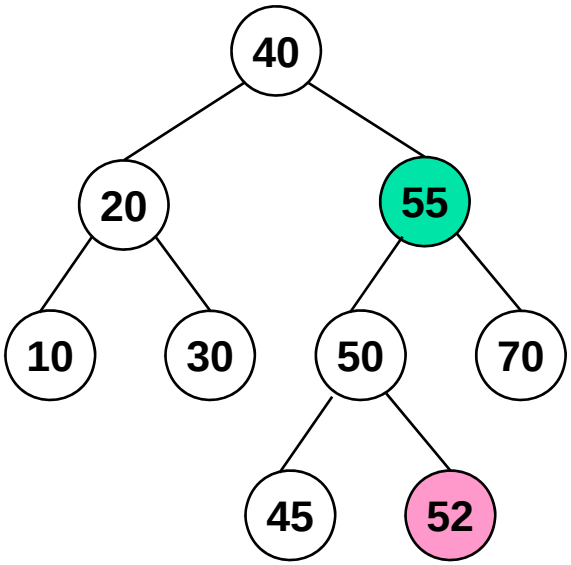


- 두 개의 **children**을 갖는 노드의 삭제
  - 왼쪽 서브트리에서 가장 큰 노드나, 오른쪽 서브트리에서 가장 작은 노드를 삭제된 자리에 위치시킴

복잡한 삭제의 경우



(a) tree before deletion of 60

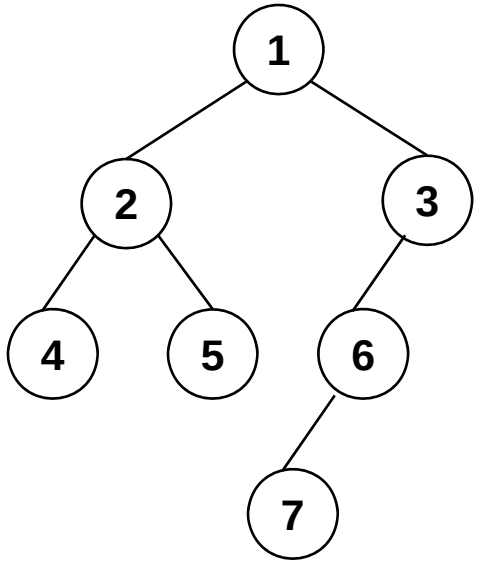


(b) tree after deletion of 60

트리 – 실습 1

- **struct node** 구조체를 이용하여 오른쪽과 같은 이진 트리를 생성하라.

```
struct node {
        int         data;
        struct node         *lchild;
        struct node         *rchild;
};
```

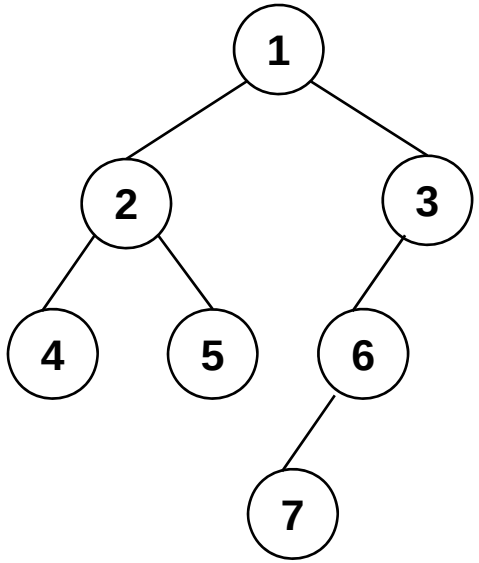


- 생성된 트리에 대해 **inorder, preorder, postorder** 알고리즘을 실행하라.

트리 – 실습 2

- **struct node** 구조체를 이용하여 오른쪽과 같은 이진 트리를 생성하라.

```
struct node {
        int         data;
        struct node         *lchild;
        struct node         *rchild;
};
```

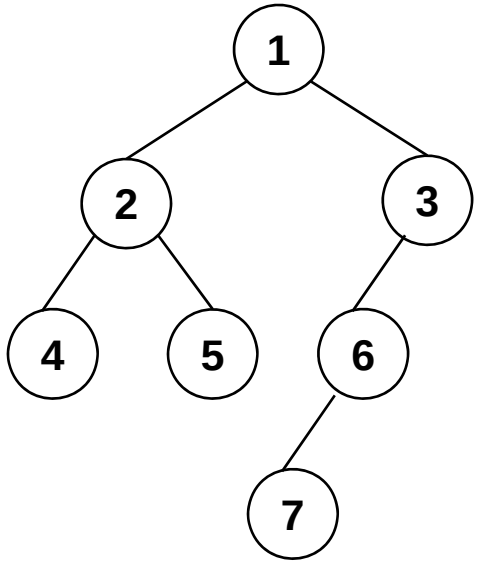


- 생성된 트리를 또 다른 트리로 복사하되, 왼쪽 자식과 오른쪽 자식을 바꾸어서 복사하라.
- 복사한 트리에 대해서도 **inorder, preorder, postorder** 알고리즘을 실행하라.

트리 – 실습 3

- **struct node** 구조체를 이용하여 오른쪽과 같은 이진 트리를 생성하라.

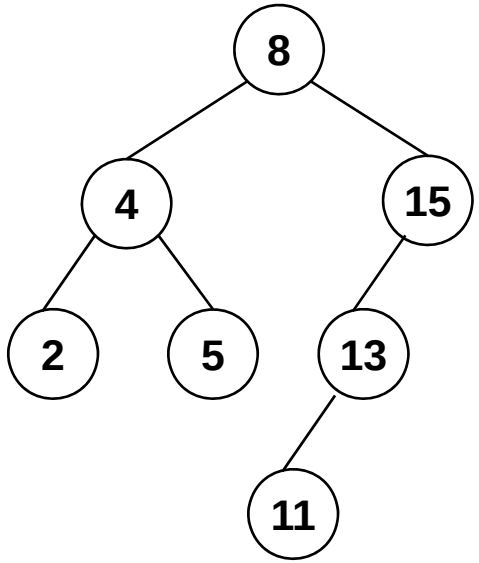
```
struct node {
    int         data;
    struct node     *lchild;
    struct node     *rchild;
};
```



- 생성된 트리에 대해 **degree**가 **0, 1, 2**인 노드 수를 각각 출력하라.
- 트리에 저장된 데이터 중에서 가장 큰 값을 출력하는 함수를 작성하고 테스트해 보라.

트리 – 실습 4

- 이진 검색트리가 되도록 데이터를 아래와 같이 수정하라.



- 데이터 x를 입력받아, x를 갖는 노드의 포인터를 반환하는 함수 search(struct node root, int x)를 작성하라.
- 데이터 x를 입력받아, x가 저장될 위치의 부모 노드를 반환하는 함수 modified_search(root, x)를 작성하라.