트리 수업내용 요약

- 트리의 정의
- 이진트리의 정의/종류/특징/표현
- 일반 트리의 운행 방법
- 이진트리 운행과 방법
- 이진탐색트리

이진트리의 성질

◆ 이진 트리의 성질 정리

• **정리** [**최대노드수**]

(1) **레벨** i **의 최대 노드 수**: 2^{i-1} **(i** $\geq$ 1)

- ✓ i=1 **때 하나의 노드, 루트만 존재하므로 레벨** i**에서 최대 노드수는** $2^{i-1} = 2^0 = 1$
- ✓ **모든** j (1 $\leq$ j < i) **에 대해 레벨** j **에서의 최대 노드 수는** 2^{j-1} **이라 가정**
- ✓ **레벨** j+1 **의 최대 노드 수는** $2^{j-1} * 2 = 2^{(j+1)-1} = 2^j$

- ✓ **각 노드의 링크수는 최대차수인** 2**이므로**

  레벨 i**의 최대 노드 수는 레벨** i−1**에서의 최대노드수의** 2**배**, 2^{i-1}

(2) **깊이** k **인 이진 트리가 가질 수 있는 최대 노드수** : $2^k - 1$ **(k** $\geq$ 1)

- ✓ $\sum$ **(레벨 i 의 최대 노드수)** = $\sum 2^{i-1} = 2^k - 1$

이진트리의 성질

◆ 이진 트리의 성질 정리

- **정의**

  깊이가 k인 포화이진 트리는, 노드수가 $2^k - 1$ ($k \geq 0$)인 이진 트리

- **정의**

  깊이가 k 이고 노드수가 n인 이진트리가, 깊이 k인 포화이진트리 1부터 n까지 번호를 붙인 노드들과 일치하면 완전이진트리

- **정리 [단말 노드수와 차수가 2인 노드수와의 상관관계]**

  모든 이진트리 T에 대하여 n0는 단말 노드수, n2는 차수가 2인 노드수 라면

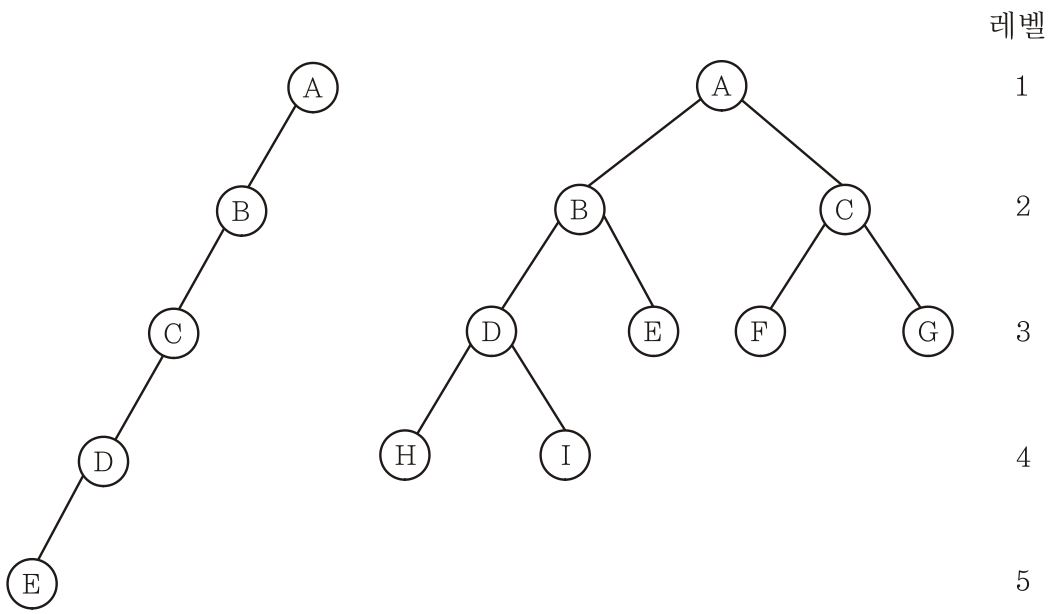
  n0 = n2 + 1

  노드의 수 = 링크의 수 + 1

  n = n0 + n1 + n2 = n1 + 2 * n2 + 1

이진트리의 표현

◆ 배열 표현



0	
1	A
2	B
3	
4	C
5	
6	
7	
8	D
9	
10	
11	
12	
13	
14	
15	
16	E

0	
1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H
9	I
10	
11	
12	
13	
14	
15	

이진트리의 표현

◆ 배열 표현

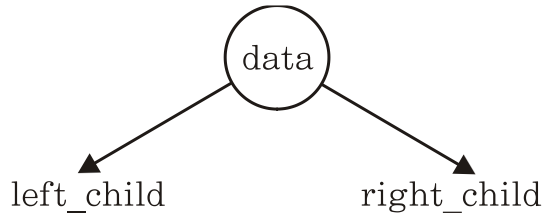
- 공간 낭비 크다
- 모든 이진 트리 표현 가능
- 완전 이진 트리는 낭비되는 공간 없다.
- 트리 중간에 노드 삽입, 삭제 시 다른 노드들의 많은 이동 발생
- 경사트리는 반이하의 기억장소만 사용.
- 최악의 경우, 깊이 k인 경사트리는 $2^k - 1$ 개의 기억장소 필요한데,
  실제로 k개만을 사용

이진트리의 표현

◆ 링크 표현

- 단순 연결 리스트를 사용하여 구현

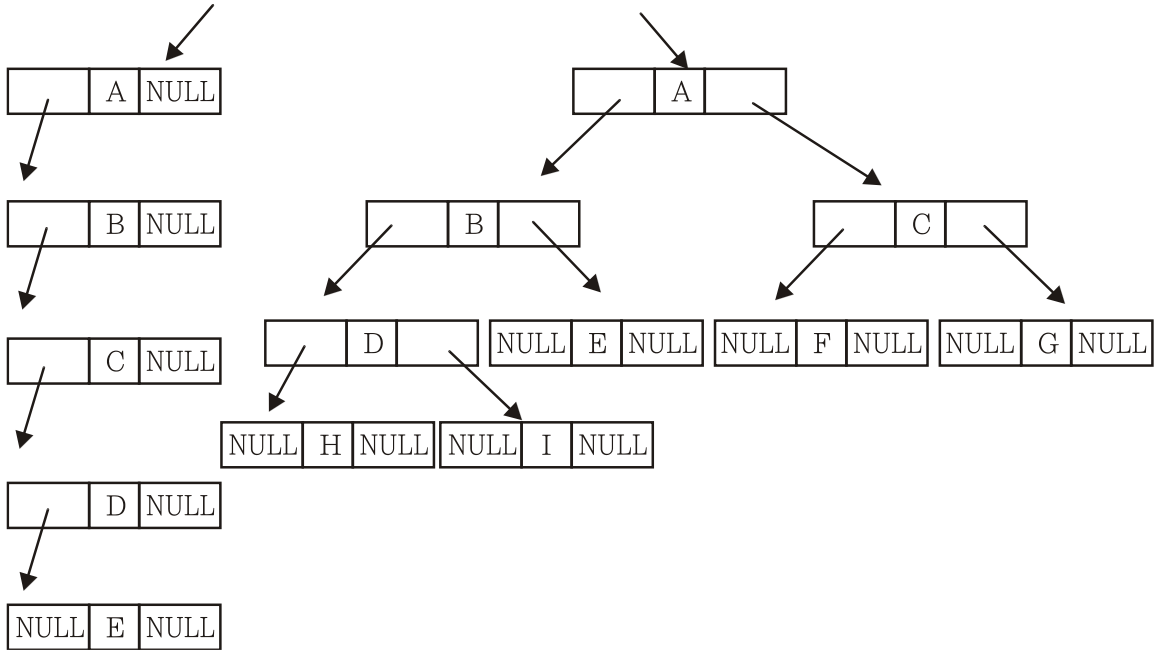
  - 이진 트리의 모든 노드는 2개의 자식 노드를 가지므로 일정한 구조의 노드를 사용할 수 있다.



  - 이진 트리의 노드 구조에 대한 C 구조체 정의

```
typedef struct tnode *tree_pointer ;
struct tnode {
        int data;
        tree_pointer left_child, right_child;
};
```

이진트리의 표현


A
NULL
B
NULL
C
NULL
D
NULL
NULL
E
NULL
A
B
C
D
NULL
E
NULL
NULL
F
NULL
NULL
G
NULL
NULL
H
NULL
NULL
I
NULL

일반트리의 운행 (Traversal)

◆ 트리 운행

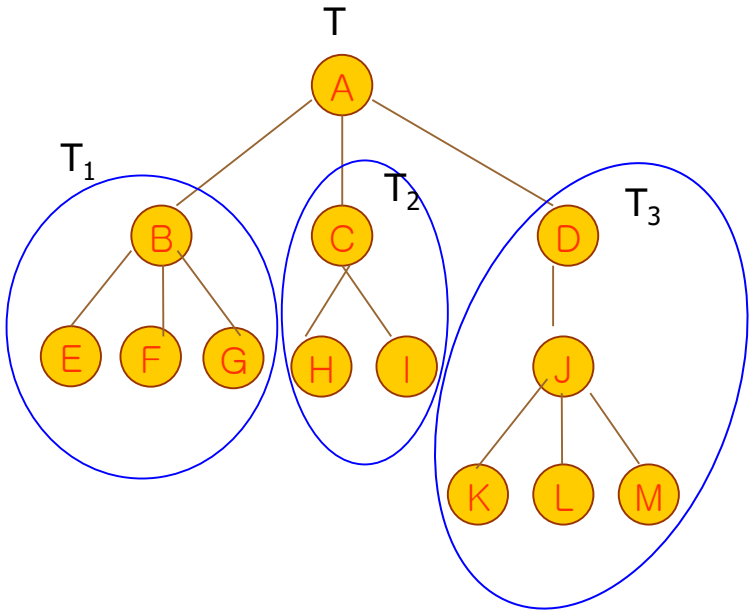
- 각 노드를 중복되지 않게 한번에 한 노드씩 전체 노드를 전부 검사하는 것

◆ 일반 트리의 운행 방법

- 레벨순 (level order)
- 전위 운행 (preorder traversal)
- 후위 운행 (postorder traversal)
- 족보순 (family order)

레벨순 운행 (Level Order Traversal)

- 트리의 각 노드를 레벨순으로 검사, 동일 레벨 노드는 왼쪽에서 오른쪽으로 검사
- 하향식(top-down) : 왼 → 오 , 위 → 아래
- 상향식(bottom-up) : 왼 → 오, 아래 → 위

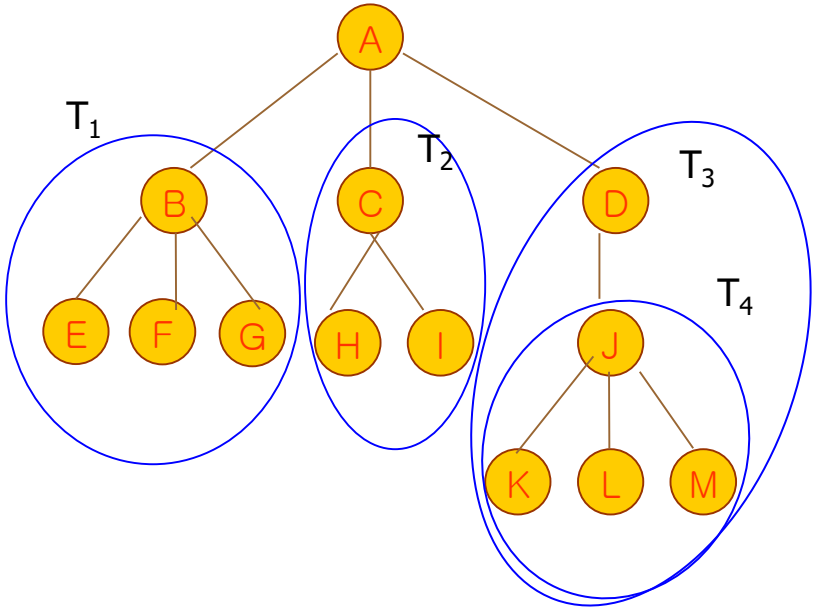


레벨순 운행	운행순서
하향식	A BCD EFG HI J KLM
상향식	KLM EFG HI J BCD A

전위운행 (Preorder Traversal)

- **루트 → left subtree → right subtree**

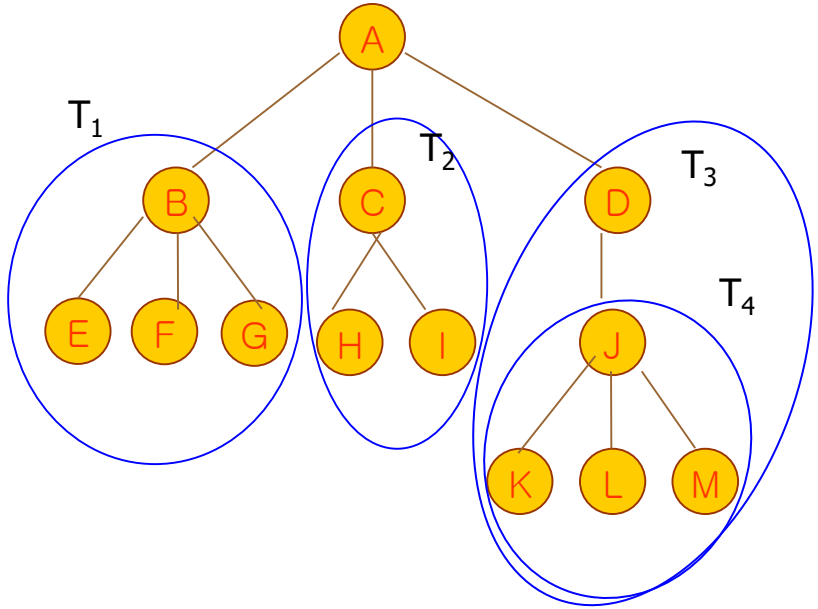
내용	노드검사
운행방법	A (T1) (T2) (T3) A (BEFG) (CHI) (D T4) A BEFG CHI D (JKLM)
운행순서	A BEFG CHI D JKLM



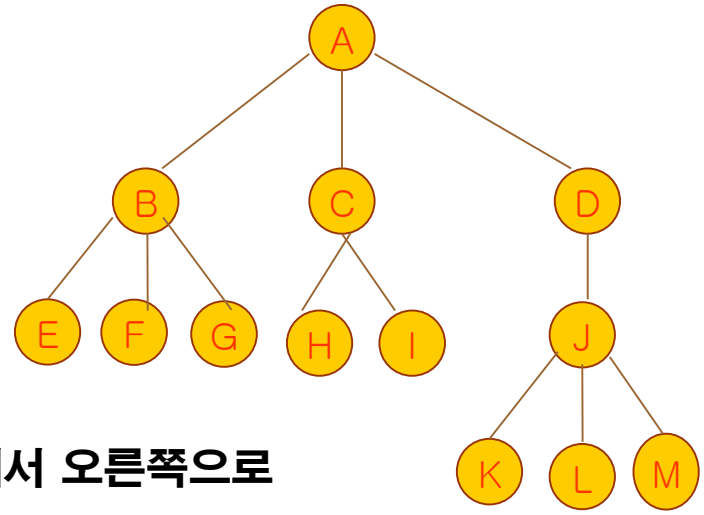
후위운행 (Postorder Traversal)

- **left subtree → right subtree → 루트**

내용	노드검사
운행방법	(T1) (T2) (T3) A (EFGB) (HIC) (T4 D) A EFGB HIC (KLMJ) D A
운행순서	EFGB HIC KLMJ D A



족보순 운행 (Family Order Traversal)



루트 → left subtree → right subtree

스택을 이용하여 가장 늦게 방문한 자식에서 같은 과정 반복수행

- 스택 이용

① 루트노드를 방문하고 스택에 push

② 한 개의 노드를 pop한후 , pop된 노드의 자식노드를 왼쪽에서 오른쪽으로 방문하고 스택에 push한다.

③ 자식노드가 더 이상 없으면 가장 마지막으로 push한 노드를 pop한 후 pop된 노드의 자식노드를 왼쪽부터 오른쪽으로 push 한다.

④ ③에서 pop된 노드의 자식노드가 없으면 스택의 노드를 pop하고, pop한 노드의 자식노드를 왼쪽부터 오른쪽으로 순으로 스택에 push한다.

이진트리의 운행

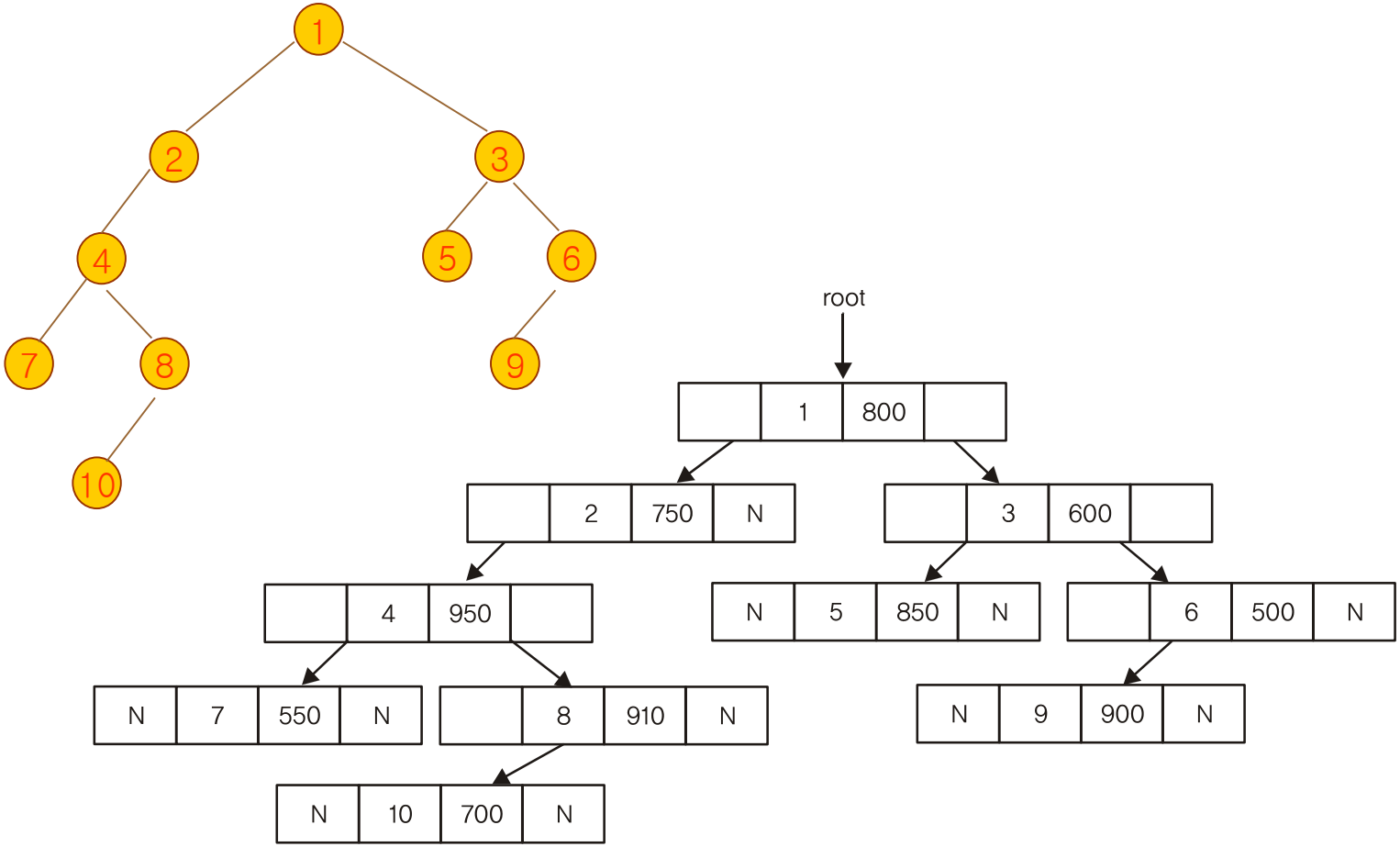
◆ 이진 트리가 순환적(recursive)으로 정의되어 있으므로,
운행작업도 서브트리에 대해서 순환적으로 반복하여 완성한다.

◆ 왼쪽 서브트리에 대한 운행을 오른쪽 서브트리 보다 먼저 수행한다.

◆ 운행의 종류

- 중위 운행(inorder traversal)
- 전위 운행(preorder traversal)
- 후위 운행(postorder traversal)

이진트리의 운행


1
2
3
4
5
6
7
8
9
10
root
1
800
2
750
N
3
600
4
950
N
5
850
N
6
500
N
N
7
550
N
8
910
N
N
9
900
N
N
10
700
N

중위 운행(inorder traversal)

◆ 수행 방법 (왼쪽서브트리->루트->오른쪽 서브트리 순으로)

- 널 노드에 도달할 때까지 왼쪽방향으로 이동
- 널에 도착하면 널 노드의 위 레벨 노드를 방문, 운행은 오른쪽으로
- 오른쪽으로 이동할 수 없는 경우는 바로 위 레벨의 방문하지 않은 노드에서 운행 시작

◆ 노드 구조

```
typedef struct tnode *tree_pointer;
struct tnode {
      tree_pointer left_child;
      int num;
      int score;
      tree_pointer right_child;
};
```

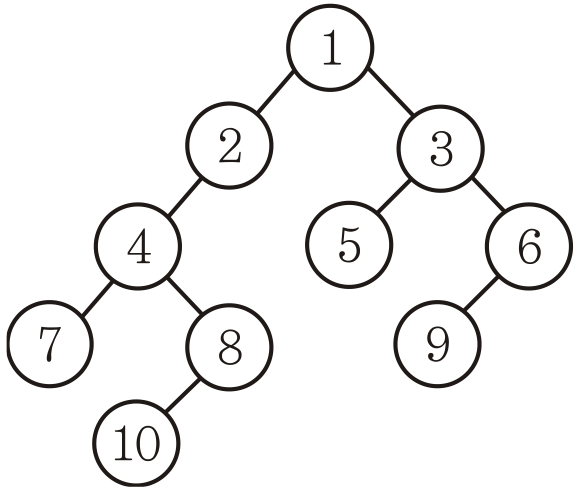
중위 운행(inorder traversal)

• 중위 운행 알고리즘

```
void inorder1(tree_pointer ptr)
{
    if (ptr) {
          inorder1 (ptr -> left_child);
          printf("%d", ptr -> num);
          inorder1 (ptr -> right_child);
     }
}
```

```
void inorder2(tree_pointer ptr)
{
    if (ptr) {
          inorder2 (ptr -> left_child);
          if(ptr->score >= 800)
                printf("%d    %d₩n", ptr -> num, ptr->score);
          inorder2 (ptr -> right_child);
     }
}
```

중위 운행(inorder traversal)의 예



- **inorder1 => 7 4 10 8 2 1 5 3 9 6**
- **inorder2 =>**

  4 950

  8 910

  1 800

  5 850

  9 900

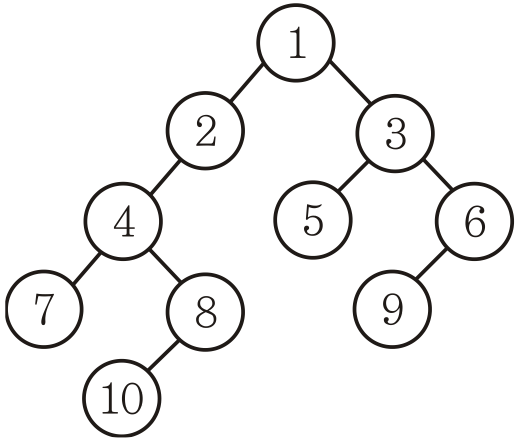
전위운행(preorder traversal)

◆ 수행 방법 (루트 -> 왼쪽서브트리 ->오른쪽 서브트리 순으로)

- 현재 루트를 처리
- 왼쪽 서브트리 처리
- 오른쪽 서브트리 처리

```
void preorder(tree_pointer ptr)
{
    if (ptr) {
          printf("%d", ptr -> num);
          preorder (ptr -> left_child);
          preorder (ptr -> right_child);
     }
}
```

전위운행의 예



preorder → **1 2 4 7 8 10 3 5 6 9**

후위운행(postorder traversal)

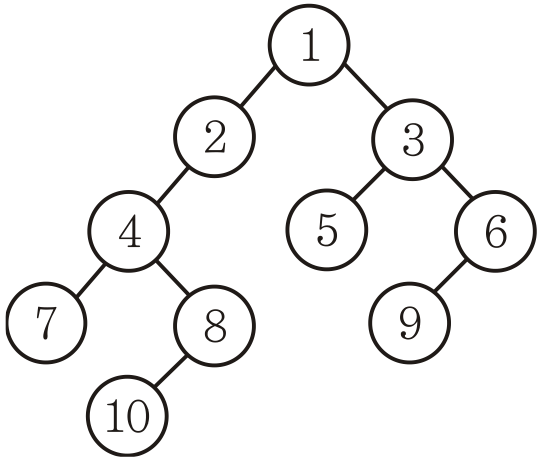
◆ 수행 방법 (왼쪽 서브트리 ->오른쪽 서브트리 -> 루트 순으로)

- 왼쪽 서브트리 먼저 방문
- 오른쪽 서브트리 방문.
- 루트 방문

◆ 후위 운행 알고리즘

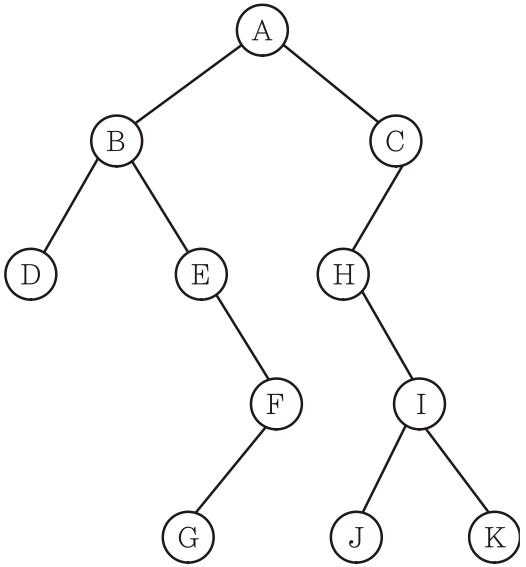
```
void postorder(tree_pointer ptr)
{
    if (ptr) {
          postorder (ptr -> left_child);
          postorder (ptr -> right_child);
          printf("%d", ptr -> num);
     }
}
```

후위운행의 예



postorder → 7 10 8 4 2 5 9 6 3 1

이진트리의 운행예제



- **중위 운행 :**

  D B E G F A H J I K C

- **전위 운행 :**

  A B D E F G C H I J K

- **후위 운행 :**

  D G F E B J K I H C A

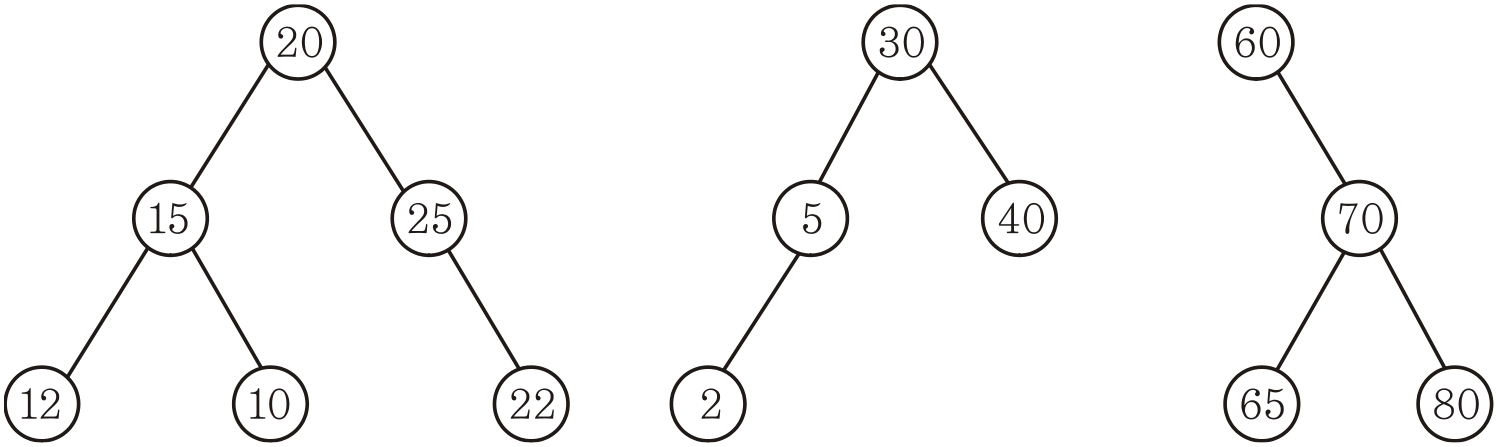
이진탐색트리

- **이진 탐색 트리 (Binary Search Tree)의 정의**

BST는 다음의 성질을 만족하는 이진트리이다.

- **모든 노드(원소)는 키값을 갖는다.**
- **BST의 왼쪽 서브트리의 키값은 루트의 키 값보다 작아야 한다.**
- **BST의 오른쪽 서브트리의 키값은 루트의 키 값보다 커야 한다.**
- **왼쪽과 오른쪽 서브 트리도 또한 BST이다.**

이진탐색트리



(a) BST 아님 (b) BST (c) BST

이진탐색트리의 연산

- 탐색
  - 찾으려는 skey 값을 현재 root의 key값과 비교
  - skey < root의 값이면 go to left
  - skey > root의 값이면 go to right (NULL을 만나면 탐색실패)

- 삽입
  - 탐색 후 실패한 위치에 삽입
  - ex) ikey <- 40

이진탐색트리 문제

300, 170, 250, 550, 110, 700, 630, 600의 순으로 이진 탐색 트리에 노드를 삽입하여 트리를 구축할 때 다음 물음에 답하시오.

(1) 구축된 이진 탐색 트리를 그리시오.

(2) 구축된 이진 탐색 트리의 깊이는?

(3) 구축된 이진 탐색 트리를 중위 운행한 결과를 쓰시오.

이진탐색트리 문제

(4) 구축된 이진 트리의 각 노드를 위하여 다음의 노드 포인터 tree_pointer와 inoder함수를 참조하여 key값의 오름차순으로 출력하는 함수 void asort(tree_pointer root)를 작성하시오.

```
typedef struct bsinode *tree_pointer;
struct bstnode{
    tree_pointer left_child;
    int key;
   tree_pointer right_child;
};
```

```
void inorder(tree_pointer ptr)
{
      if (ptr) {
        inorder(ptr -> left_child);
        printf ("%d ", ptr -> num);
        inorder(ptr -> right_child);
      }
}
```