Chapter 08.
트리

13주차

트리
(계층적인 관계를 표현하는 비선형 자료구조)

트리는 데이터의 저장, 검색, 삭제 보다는

데이터의 표현을 위한 도구이다!

트리 관련 용어 (1/3)

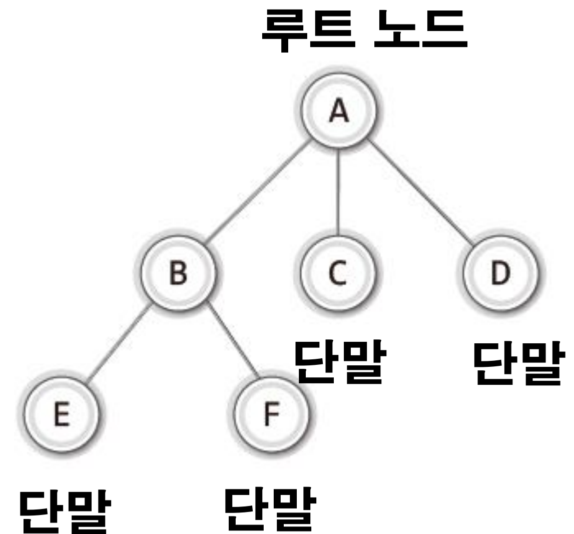
- **노드: node**
  트리의 구성요소에 해당하는 A, B, C, D, E, F와 같은 요소

- **간선: edge**
  노드와 노드를 연결하는 연결선

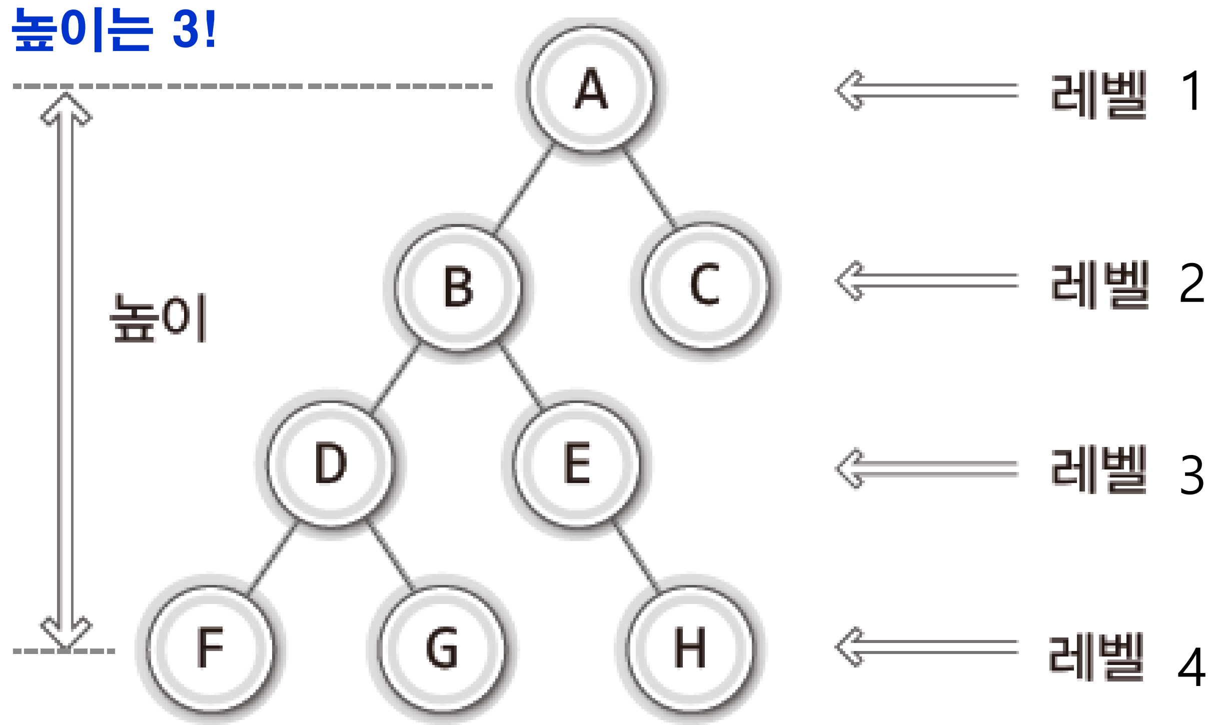
- **루트 노드: root node**
  트리 구조에서 최상위에 존재하는 A와 같은 노드

- **단말 노드: terminal node**
  아래로 또 다른 노드가 연결되어 있지 않은 E, F, C, D와 같은 노드

- **내부 노드: internal node**
  단말 노드를 제외한 모든 노드로 A, B와 같은 노드



트리 관련 용어 (2/3)



트리의 높이와 레벨의 최대 값은 같다!

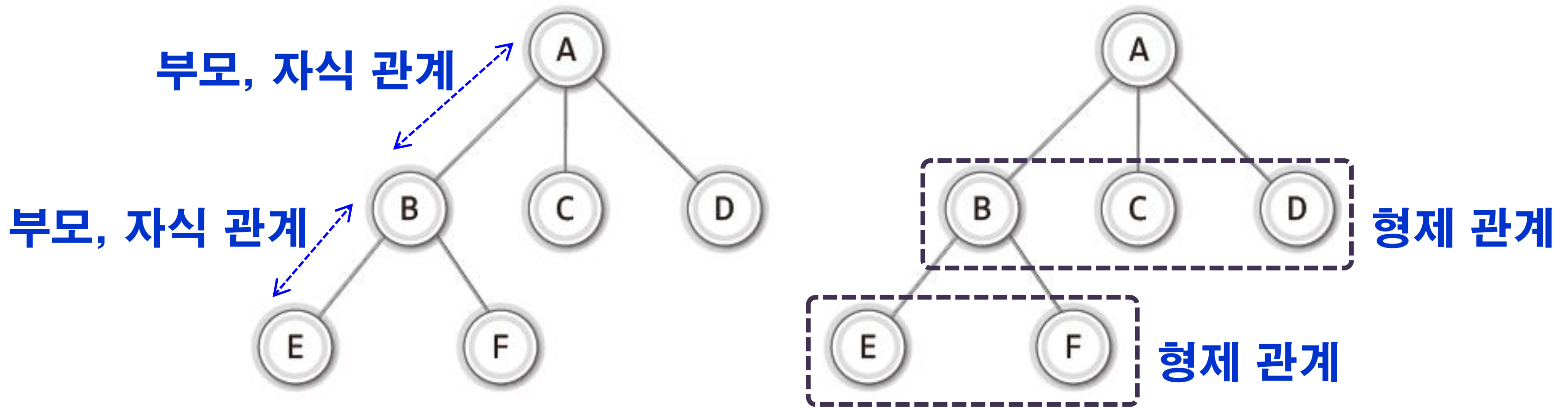
서브트리 : A의 서브트리 2개 : B를 루트로 하는 트리와 C를 루트로 하는 트리.
B의 서브트리 2개 : D를 루트로 하는 트리와 E를 루트로 하는 트리.

[트리의 정의]

루트 노드와 N개의 서브 트리로 이루어진다.

서브트리 역시 트리이다. (재귀적인 정의)

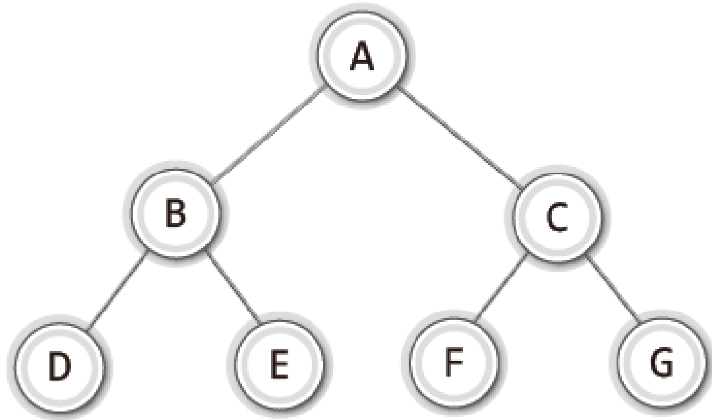
트리 관련 용어 (3/3)



부모노드 : 루트 노드와의 최단 경로 상의 레벨이 1 낮은 노드.
자식노드 : 직접 연결되어 있으면서 레벨이 1 높은 노드.
형제노드 : 부모가 같은 노드들.

- 노드 A는 노드 B, C, D의 부모 노드(parent node)이다.
- 노드 B, C, D는 노드 A의 자식 노드(child node)이다.
- 노드 B, C, D는 부모 노드가 같으므로, 서로가 서로에게 형제 노드(sibling node)이다.

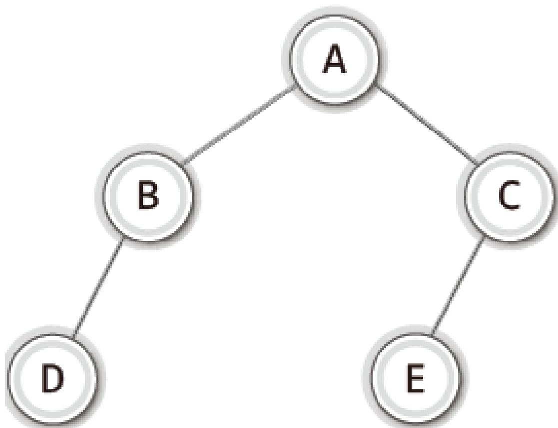
이진 트리



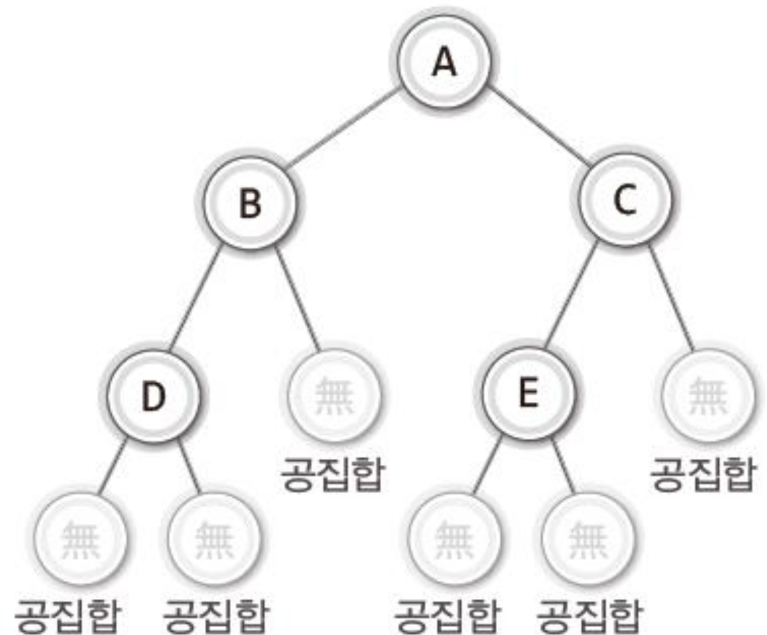
이진 트리의 예

[이진 트리의 조건]

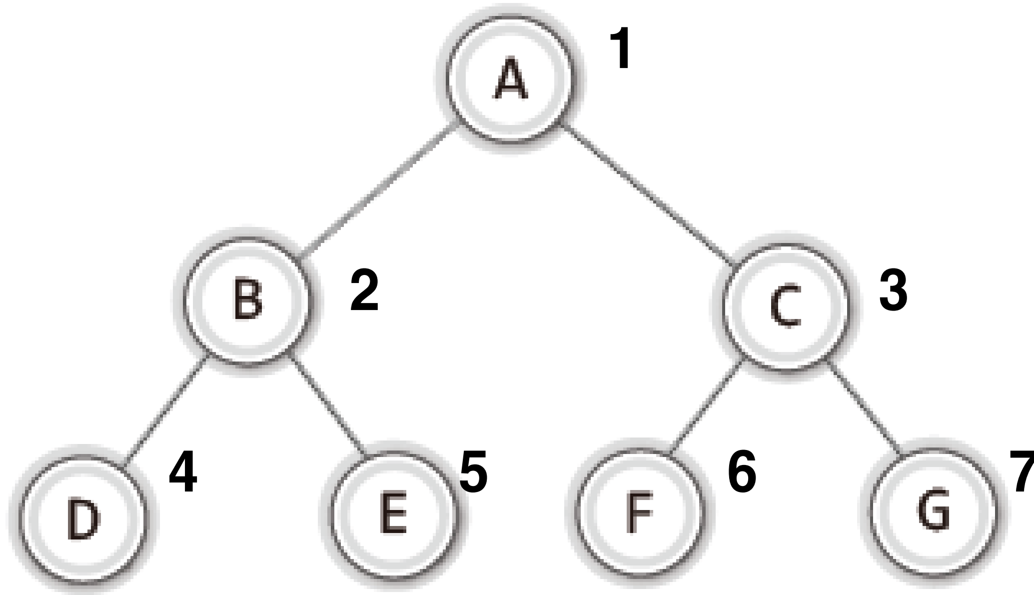
- 루트 노드를 중심으로 두 개의 서브 트리로 나뉘어진다.
- **왼쪽 자식과 오른쪽 자식을 구분한다**.
- 두 개의 서브 트리도 모두 이진 트리이어야 한다. (재귀적인 정의!)
- 공집합도 노드로 간주된다. (공집합도 이진트리)



다른 표현



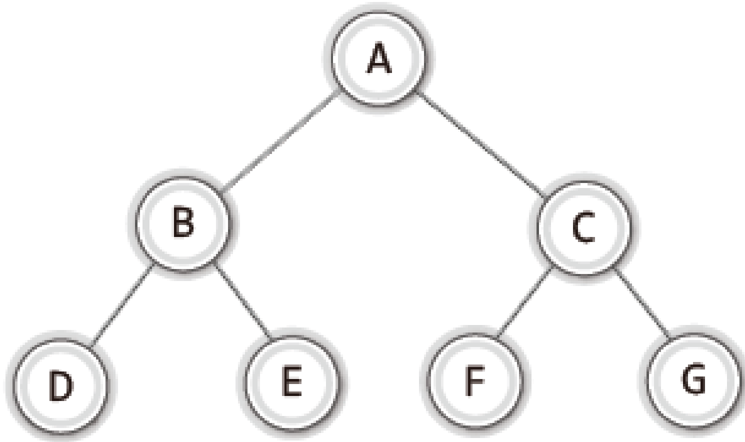
이진 트리의 노드 번호 매기기



- 루트 노드 번호는 1.
- 위에서 아래로, 같은 레벨에서는 왼쪽에서 오른쪽으로 진행.
- 이진 트리의 모든 노드가 있다고 생각하고 번호를 하나씩 증가시키며 붙임

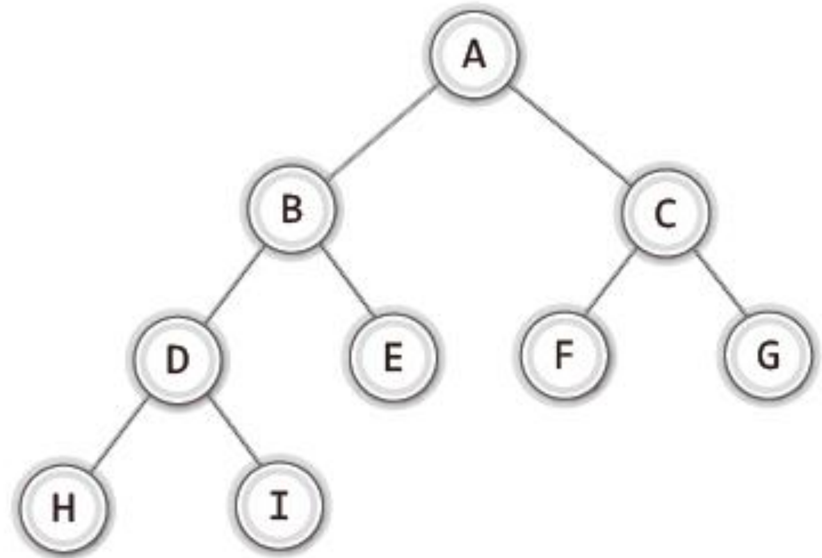
** 번호는 노드에 붙여지는 것이 아니고 자리에 붙여지는 것!

포화이진트리와 완전이진트리



모든 레벨에 최대 수의 노드를 가진 이진트리

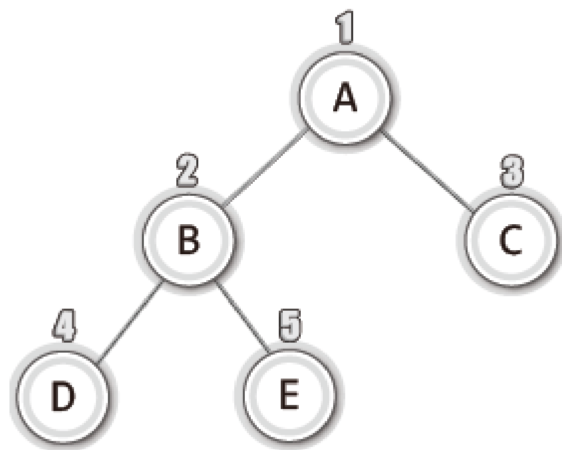
포화 이진 트리



빈 번호가 없는 이진 트리

완전 이진 트리

이진 트리의 구현
(배열 기반)



Index	Value
[0]	
[1]	A
[2]	B
[3]	C
[4]	D
[5]	E
[6]	·
[7]	·

- 노드 번호가 배열의 index가 됨.
- 구현 편의상 배열의 첫 번째 요소는 사용하지 않음.

[i] 노드의 부모 위치 : [i/2]
[i] 노드의 왼쪽 자식 위치 : i * 2
[i] 노드의 오른쪽 자식 위치 : i * 2 + 1

이진 트리의 배열기반 구현

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <stdlib.h>

typedef int BTData;
#define COUNT   100

// 0 이상의 정수만 저장한다고 전제하고 모든 노드를 -1로 초기화.
void InitTree(BTData *tree, int cnt) {
        int i;
        for (i = 0; i < cnt; i++)
                tree[i] = -1;
}
```

이진 트리의 배열기반 구현

```
void MakeRootNode(BTData *tree, int cnt, BTData data) {
        tree[1] = data;
}

void MakeLeftSubTree(BTData *tree, int cnt, int index, BTData data)
{
        int leftindex = index * 2;
        if (leftindex >= cnt)
                return;
        return tree[leftindex] = data;
}

void MakeRightSubTree(BTData *tree, int cnt, int index, BTData data)
{
        int rightindex = index * 2 + 1;
        if (rightindex >= cnt)
                return;
        return tree[rightindex] = data;
}
```

이진 트리의 배열기반 구현

```
BTData GetData(BTData *tree, int cnt, int index) {
	if (index >= cnt || tree[index] < 0)
		return -1;
	return tree[index];
}

BTData GetLeftSubTree(BTData *tree, int cnt, int index)
{
	return index * 2;
}

BTData GetRightSubTree(BTData *tree, int cnt, int index)
{
	return index * 2 + 1;
}
```

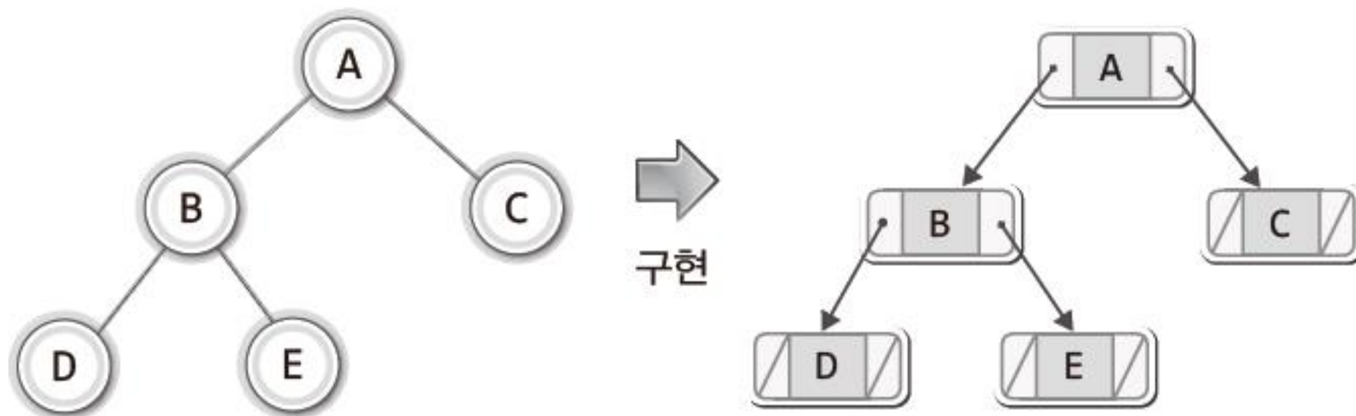
이진 트리의 배열기반 구현

```
int main(void) {
    BTData BTree[COUNT];

    InitTree(BTree, COUNT);
    MakeRootNode(BTree, COUNT, 1);          // 루트 노드에 1 넣기.
    MakeLeftSubTree(BTree, COUNT, 1, 2 );   // 1번 노드의 왼쪽 자식으로 2 넣기
    MakeRightSubTree(BTree, COUNT, 1, 3 );  // 1번 노드의 오른쪽 자식으로 2 넣기
    MakeLeftSubTree(BTree, COUNT, 2, 4 );   // 2번 노드의 왼쪽 자식으로 4 넣기

    printf("%d ₩n", GetData(BTree, COUNT, GetLeftSubTree(BTree, COUNT, 1)));
    printf("%d ₩n", GetData(BTree, COUNT, GetLeftSubTree(BTree, COUNT,
GetLeftSubTree(BTree, COUNT, 1))));
    return 0;
}
```

루트의 왼쪽 자식의 값

루트의 왼쪽 자식의 왼쪽 자식의 값

이진 트리의 구현
(연결리스트 기반)



- 연결 리스트 기반에서는 트리의 구조와 리스트의 연결 구조가 일치함.
- 따라서 구현과 관련된 직관적인 이해가 더 좋음.

이진 트리의 구현
(노드구조체를 트리구조체로도 사용)

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <stdlib.h>

typedef int BTData;

typedef struct _bTreeNode
{
	BTData data;
	struct _bTreeNode * left;
	struct _bTreeNode * right;
} BTreeNode;
```

이진 트리의 구현
(노드 생성, 노드 데이터 get/set)

```
BTreeNode * MakeBTreeNode(void)
{
	BTreeNode * nd = (BTreeNode*)malloc(sizeof(BTreeNode));

	nd->left = NULL;
	nd->right = NULL;
	return nd;
}

BTData GetData(BTreeNode * bt)
{
	return bt->data;
}

void SetData(BTreeNode * bt, BTData data)
{
	bt->data = data;
}
```

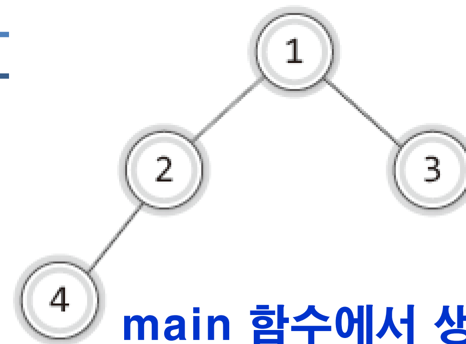
이진 트리의 구현
(왼쪽/오른쪽 서브트리 get)

```
BTreeNode * GetLeftSubTree(BTreeNode * bt)
{
		return bt->left;
}

BTreeNode * GetRightSubTree(BTreeNode * bt)
{
		return bt->right;
}
```

이진 트리의 구현
(왼쪽/오른쪽 서브트리 set)

```
void MakeLeftSubTree(BTreeNode * main, BTreeNode * sub)
{
	if (main->left != NULL)
		free(main->left);

	main->left = sub;
}

void MakeRightSubTree(BTreeNode * main, BTreeNode * sub)
{
	if (main->right != NULL)
		free(main->right);

	main->right = sub;
}
```

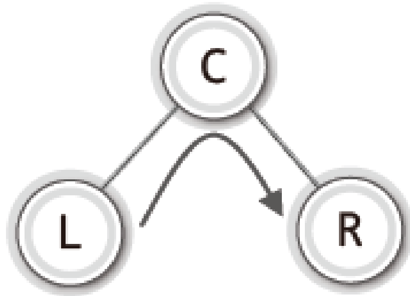
이진 트리의 구현
(main 함수)



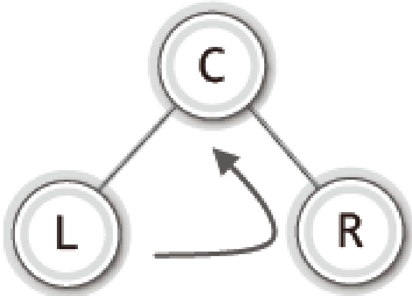

```
int main(void) {
	BTreeNode * bt1 = MakeBTreeNode();
	BTreeNode * bt2 = MakeBTreeNode();
	BTreeNode * bt3 = MakeBTreeNode();
	BTreeNode * bt4 = MakeBTreeNode();

	SetData(bt1, 1); SetData(bt2, 2); SetData(bt3, 3); SetData(bt4, 4);

	MakeLeftSubTree(bt1, bt2);
	MakeRightSubTree(bt1, bt3);
	MakeLeftSubTree(bt2, bt4);

	printf("%d ₩n", GetData(GetLeftSubTree(bt1)));
	printf("%d ₩n", GetData(GetLeftSubTree(GetLeftSubTree(bt1))));

	return 0;
}
```

이진 트리의 순회
(순회의 3가지 방법. 루트 노드를 언제 방문하느냐에 따라)

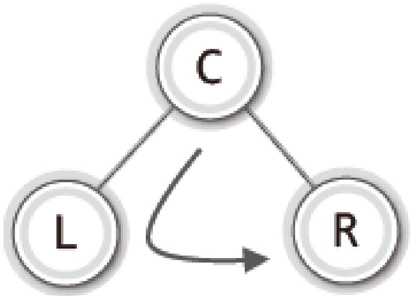
C : 루트노드 방문
L R : 왼쪽 또는 오른쪽 노드를 방문하러 이동.



▶ [그림 08-21: 중위 순회]



▶ [그림 08-22: 후위 순회]



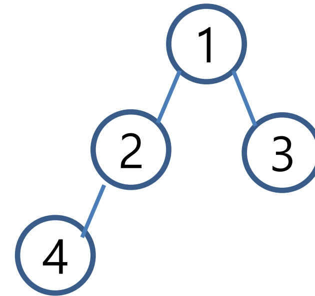
▶ [그림 08-23: 전위 순회]

이진 트리의 구현
(중위 운행 : 이전 소스에 추가)

```
void InorderTraverse(BTreeNode * bt)
{
	if (bt == NULL)    // bt가 NULL이면 재귀 탈출!
		return;

	InorderTraverse(bt->left);
	printf("%d ₩n", bt->data);
	InorderTraverse(bt->right);
}
```

이진 트리의 구현
(중위 운행 main 함수 : 이전 소스의 main함수 수정)

```
int main(void) {
	BTreeNode * bt1 = MakeBTreeNode();
	BTreeNode * bt2 = MakeBTreeNode();
	BTreeNode * bt3 = MakeBTreeNode();
	BTreeNode * bt4 = MakeBTreeNode();

	SetData(bt1, 1);
	SetData(bt2, 2);
	SetData(bt3, 3);
	SetData(bt4, 4);

	MakeLeftSubTree(bt1, bt2);
	MakeRightSubTree(bt1, bt3);
	MakeLeftSubTree(bt2, bt4);

	InorderTraverse(bt1);
	return 0;
}
```



이진 트리의 구현
(전위/후위 운행 : 이전 소스에 추가)

```
void PreorderTraverse(BTreeNode * bt)
{
	if (bt == NULL)    // bt가 NULL이면 재귀 탈출!
		return;

	printf("%d ₩n", bt->data);
	InorderTraverse(bt->left);
	InorderTraverse(bt->right);
}

void PostorderTraverse(BTreeNode * bt)
{
	if (bt == NULL)    // bt가 NULL이면 재귀 탈출!
		return;

	InorderTraverse(bt->left);
	InorderTraverse(bt->right);
	printf("%d ₩n", bt->data);
}
```

이진 트리의 구현
(중위 운행 main 함수 : 이전 소스의 main함수 수정)

```
int main(void) {
        BTreeNode * bt1 = MakeBTreeNode();
        BTreeNode * bt2 = MakeBTreeNode();
        BTreeNode * bt3 = MakeBTreeNode();
        BTreeNode * bt4 = MakeBTreeNode();
        BTreeNode * bt5 = MakeBTreeNode();
        BTreeNode * bt6 = MakeBTreeNode();

        SetData(bt1, 1); SetData(bt2, 2); SetData(bt3, 3);
        SetData(bt4, 4); SetData(bt5, 5); SetData(bt6, 6);

        MakeLeftSubTree(bt1, bt2); MakeRightSubTree(bt1, bt3);
        MakeLeftSubTree(bt2, bt4); MakeRightSubTree(bt2, bt5);
        MakeRightSubTree(bt3, bt6);

        PreorderTraverse(bt1); printf("₩n");
        InorderTraverse(bt1); printf("₩n");
        PostorderTraverse(bt1); printf("₩n");
        return 0;
}
```