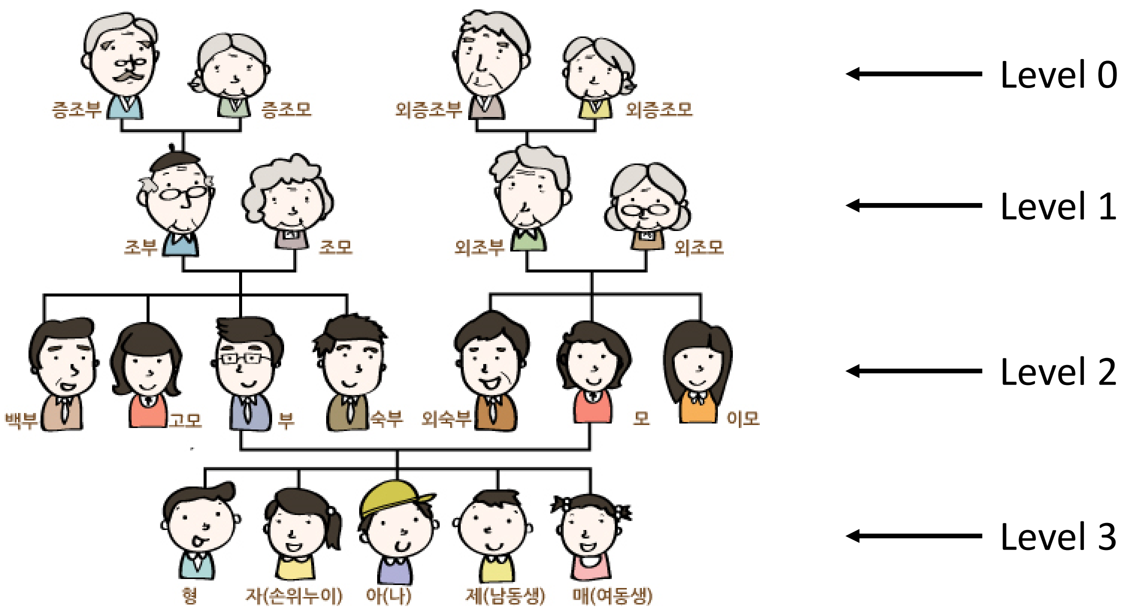
트리

10-11 주차-강의

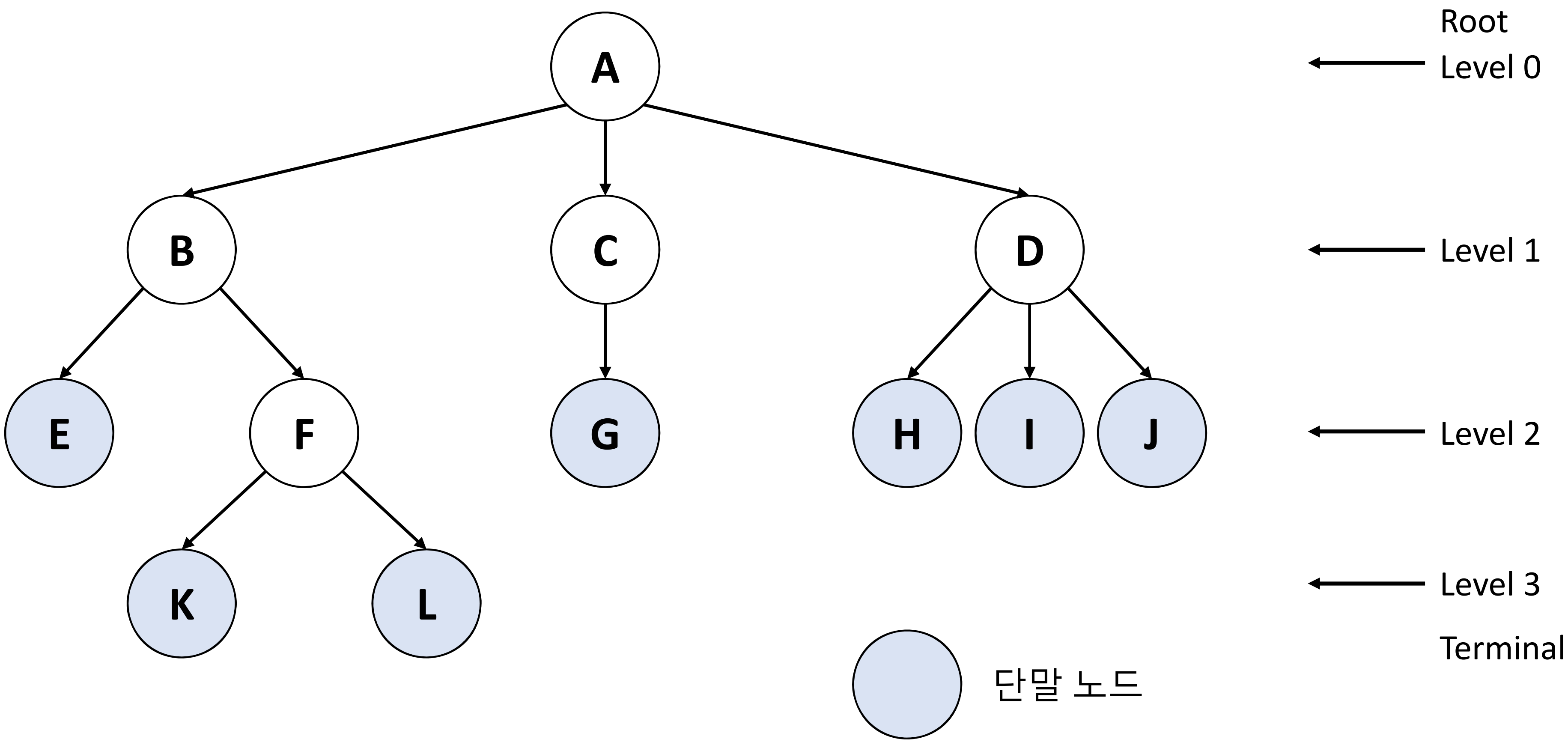
남춘성

Tree 의 개념적 이해 - I

- **트리(tree)**
  - 원소들 간에 1:n 관계를 가지는 비선형 자료구조
  - 원소들 간에 계층관계를 가지는 계층형 자료구조
  - 상위 원소에서 하위 원소로 내려가면서 확장되는 트리(나무)모양의 구조
  - 트리 자료구조의 예 – 가계도
    - 가계도의 자료 : 가족 구성원
    - 자료를 연결하는 선 : 부모-자식 관계 표현



- 트리 A


Root
Level 0
A
B
C
D
Level 1
E
F
G
H
I
J
Level 2
K
L
Level 3
Terminal
단말 노드

Tree 의 개념적 이해 - III

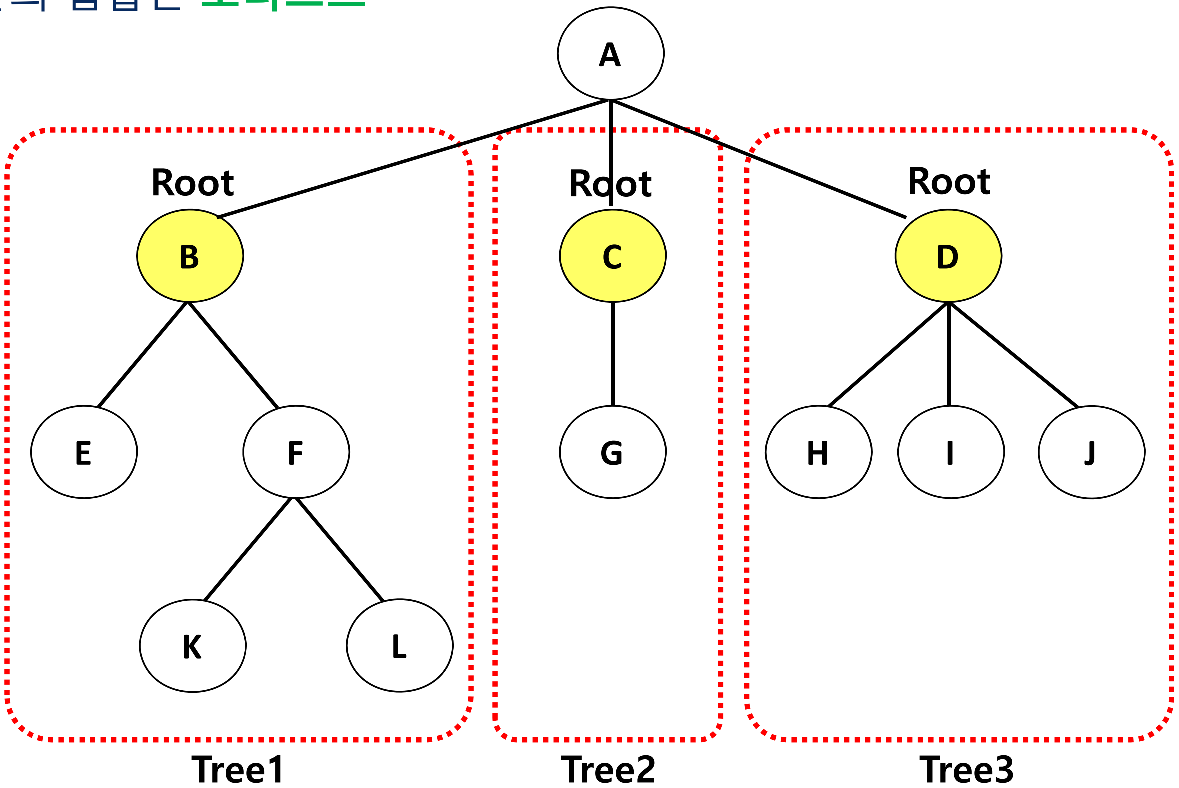
- 트리 A
  - **노드(node)**: 트리의 원소
    - 트리 A의 노드: A,B,C,D,E,F,G,H,I,J,K,L
  - **루트 노드(root node)**: 트리의 시작 노드
    - 트리 A의 루트노드: A
  - **내부 노드(internal node)**: 트리의 중간 노드
    - 단말 노드를 제외한 모든 노드 : B, C, D
  - **간선(edge)**: 노드를 연결하는 선. 부모 노드와 자식 노드를 연결
  - **형제 노드(sibling node)**: 같은 부모 노드의 자식 노드들
    - B,C,D는 형제 노드
  - **조상 노드**: 간선을 따라 루트 노드까지 이르는 경로에 있는 모든 노드들
    - K의 조상 노드 : F, B, A
  - **서브 트리(subtree)**: 부노 노드와 연결된 간선을 끊었을 때 생성되는 트리
    - 각 노드는 자식 노드의 개수 만큼 서브 트리를 가진다.
  - **자손 노드**: 서브 트리에 있는 하위 레벨의 노드들
    - B의 자손 노드: E,F,K,L

Tree 의 개념적 이해 - IV

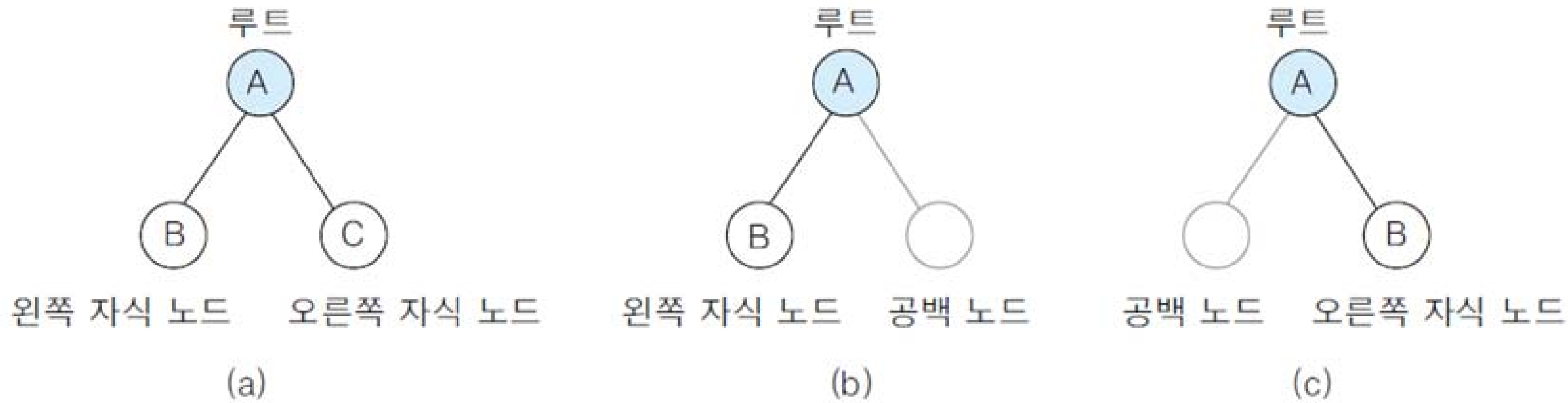
- 트리 A
  - **차수**(degree)
    - **노드의 차수** : 노드에 연결된 자식 노드의 수.
      - A의 차수=3, B의 차수=2, C의 차수=1
    - **트리의 차수** : 트리에 있는 노드의 차수 중에서 가장 큰 값
      - 트리 A의 차수=3
  - 단말 노드(리프 노드) : 차수가 0인 노드. 자식 노드가 없는 노드
- 높이
  - **노드의 높이** : 루트에서 노드에 이르는 간선의 수. 노드의 레벨
    - B의 높이=1, F의 높이=2
  - **트리의 높이** : 트리에 있는 노드의 높이 중에서 가장 큰 값. 최대 레벨
    - 트리 A의 높이=3

Tree 의 개념적 이해 - V

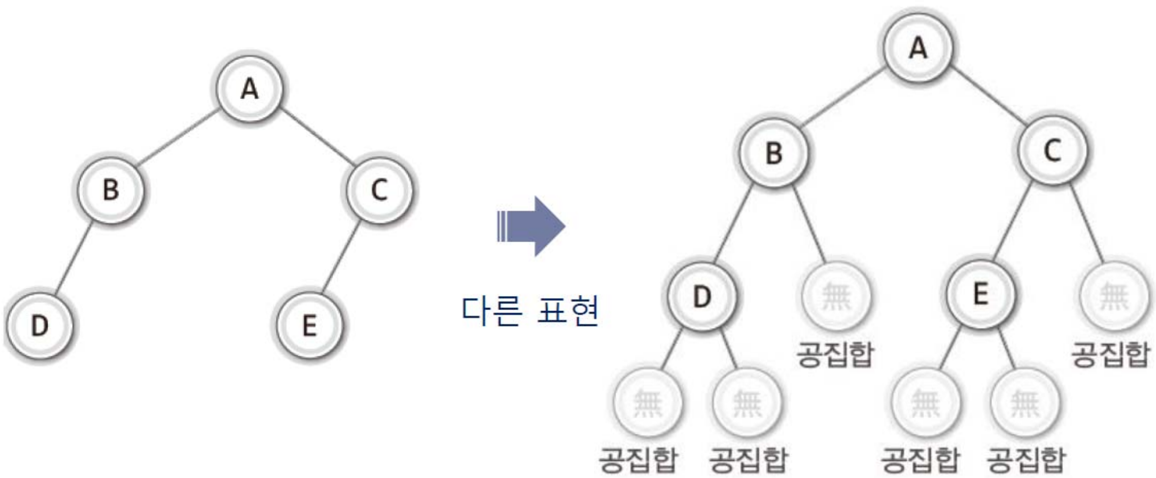
- **포리스트(forest)** : 서브트리의 집합
  - 트리A에서 노드 A를 제거하면
    - A의 자식 노드 B, C, D에 대한 **서브 트리**가 생성
    - 이들의 집합은 **포리스트**

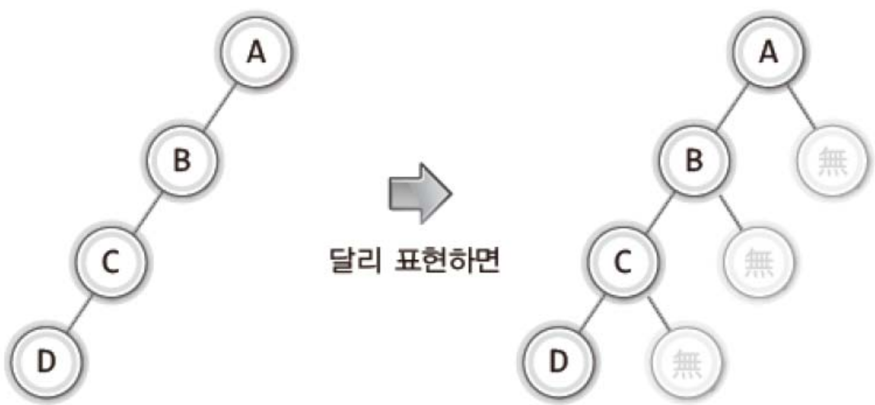


- **이진트리**
  - 트리의 노드 구조를 일정하게 정의하여 트리의 구현과 연산이 쉽도록 정의한 트리
  - 이진 트리의 모든 노드는 **왼쪽 자식 노드**와 **오른쪽 자식 노드** 만을 가진다.
    - 부모 노드와 자식 노드 수와의 관계 → 1:2
    - 공백 노드도 자식 노드로 취급한다.
    - 0 ≤ 노드의 차수 ≤ 2



- **이진트리**
  - 공집합도 이진 트리에서는 노드로 간주함!
  - 즉, 하나의 노드에 두 개의 노드가 달려있는 형태의 트리는 모두 이진트리





이진트리의 개념적 이해 - III

- **이진트리의 특성**

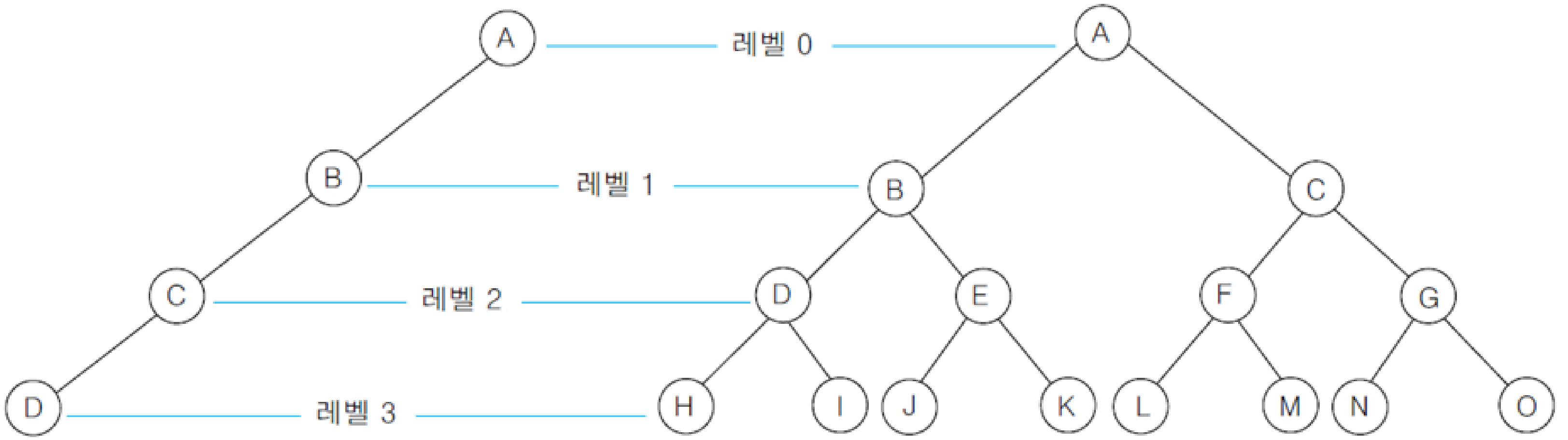
정의1) n개의 노드를 가진 이진 트리는 항상 (n-1)개의 간선을 가진다.

- 루트를 제외한 (n-1)개의 노드가 부모 노드와 연결되는 한 개의 간선을 가짐

정의2) 높이가 h인 이진 트리가 가질 수 있는 노드의 최소 개수는 (h+1)개가 되며, 최대 개수는 (2^{h+1}-1)개가 된다.

- 이진 트리의 높이가 h가 되려면 한 레벨에 최소한 한 개의 노드가 있어야 하므로 높이가 h인 이진 트리의 최소 노드의 개수는 (h+1)개
- 하나의 노드는 최대 2개의 자식 노드를 가질 수 있으므로 레벨 i에서의 노드의 최대 개수는 2i개 이므로 높이가 h인 이진 트리 전체의 노드 개수는
- $\sum 2i = 2^{h+1} - 1$ 개

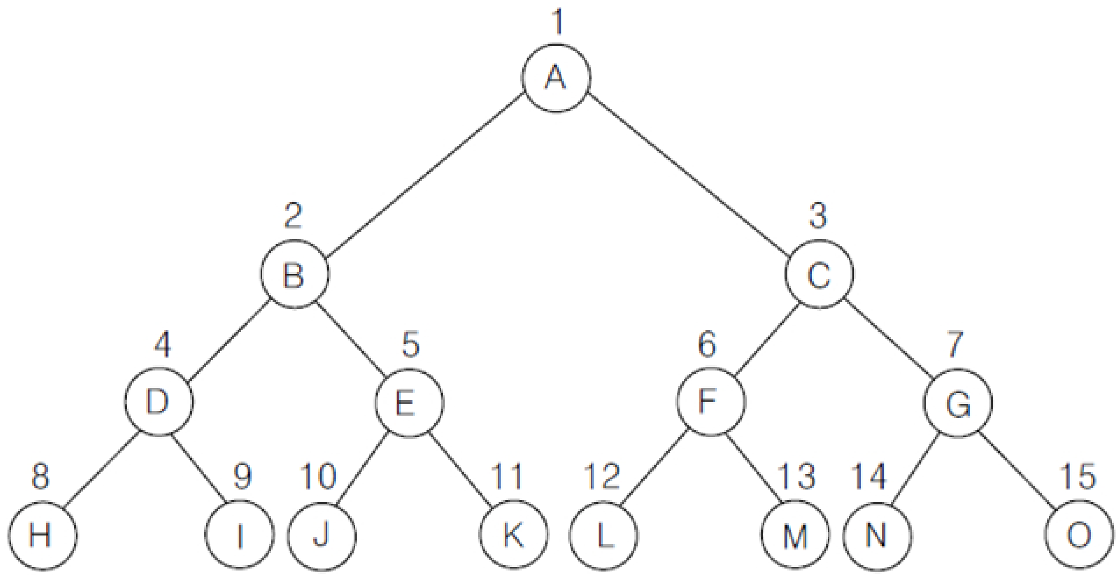
- **이진트리의 특성**
  - 높이가 3이면서 최소의 노드를 갖는 이진트리와 최대의 노드를 갖는 이진트리



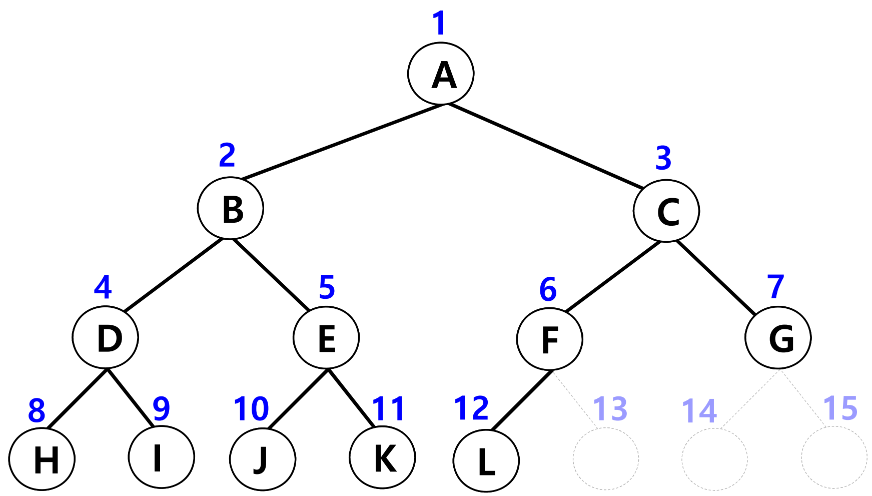
(a) 최소의 노드를 갖는 이진 트리 (b) 최대의 노드를 갖는 이진 트리

이진트리의 개념적 이해 - V

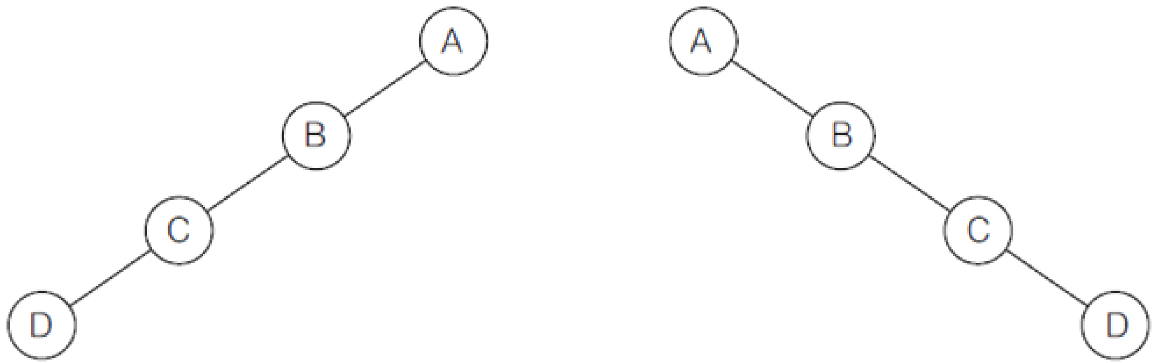
- **이진 트리의 종류**
  - 포화 이진 트리(Full Binary Tree)
    - 모든 레벨에 노드가 포화상태로 차 있는 이진 트리
    - 높이가 h일 때, 최대의 노드 개수인 $(2^{h+1}-1)$ 의 노드를 가진 이진 트리
    - 루트를 1번으로 하여 $2^{h+1}-1$까지 정해진 위치에 대한 노드 번호를 가짐



- **완전 이진 트리(Complete Binary Tree)**
  - 높이가 h이고 노드 수가 n개일 때 (단, $h+1 \le n < 2^{h+1}-1$), 포화 이진 트리의 노드 번호 1번부터 n번까지 빈 자리가 없는 이진 트리(왼쪽부터 차곡차곡 채워진 상태) 예) 노드가 12개인 완전 이진 트리



- **편향 이진 트리(Skewed Binary Tree)**
  - 높이 h에 대한 최소 개수의 노드를 가지면서 한쪽 방향의 자식 노드만을 가진 이진 트리
  - 왼쪽 편향 이진 트리
    - 모든 노드가 왼쪽 자식 노드만을 가진 편향 이진 트리
  - 오른쪽 편향 이진 트리
    - 모든 노드가 오른쪽 자식 노드만을 가진 편향 이진 트리



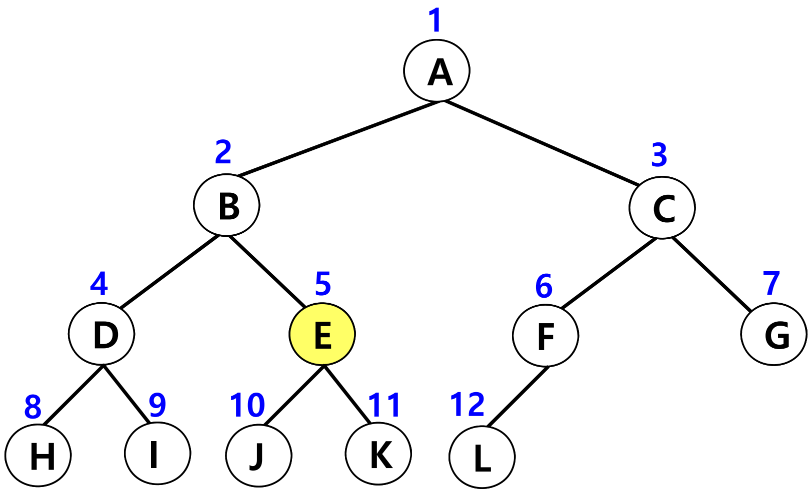
(a) 왼쪽 편향 이진 트리 (b) 오른쪽 편향 이진 트리

배열을 이용한 이진트리 개념 - I

- 1차원 배열의 순차 자료구조 사용
  - 높이가 h인 포화 이진 트리의 노드번호를 배열의 인덱스로 사용
  - 인덱스 0번 : 실제로 사용하지 않고 비워둠.
  - 인덱스 1번 : 루트 저장

배열을 이용한 이진트리 개념 - II

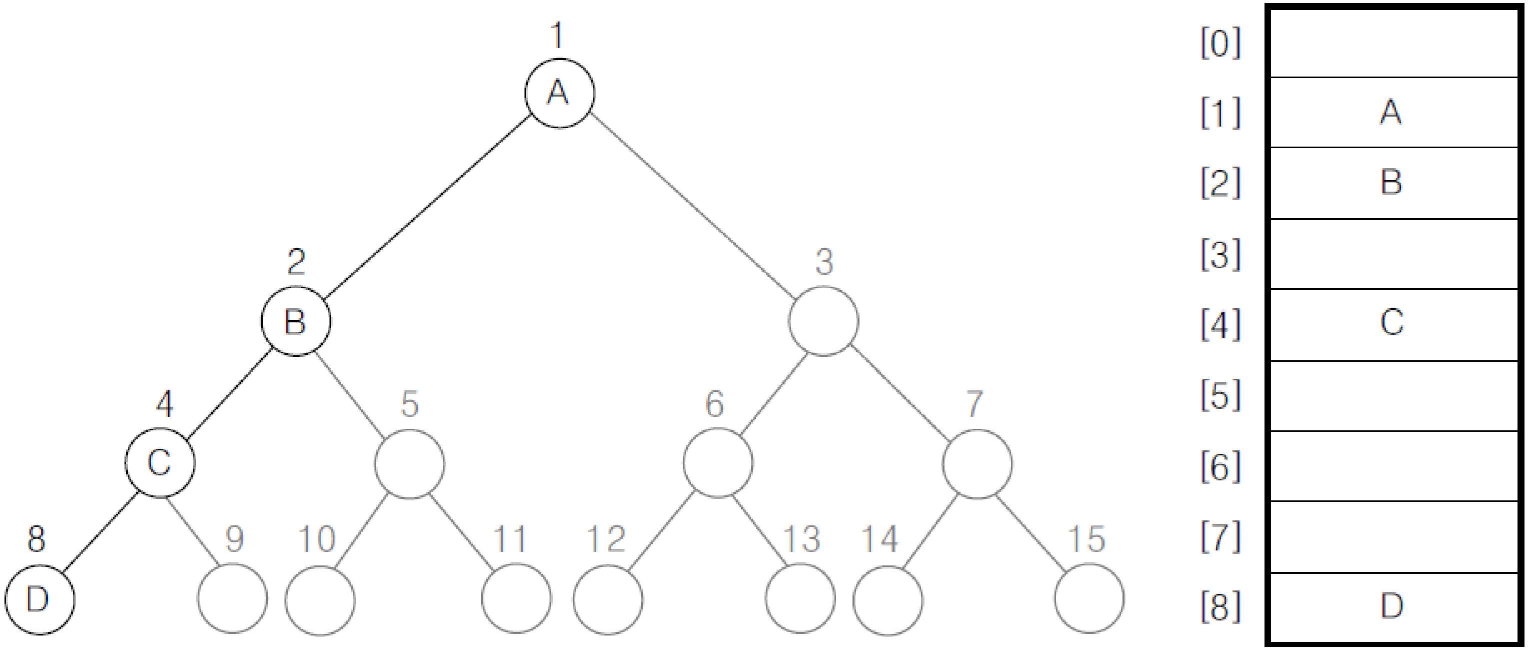
- 완전 이진 트리의 1차원 배열 표현



[0]	
[1]	A
[2]	B
[3]	C
[4]	D
[5]	E
[6]	F
[7]	G
[8]	H
[9]	I
[10]	J
[11]	K
[12]	L

배열을 이용한 이진트리 개념 - III

- **왼쪽 편향 이진 트리의 1차원 배열 표현**



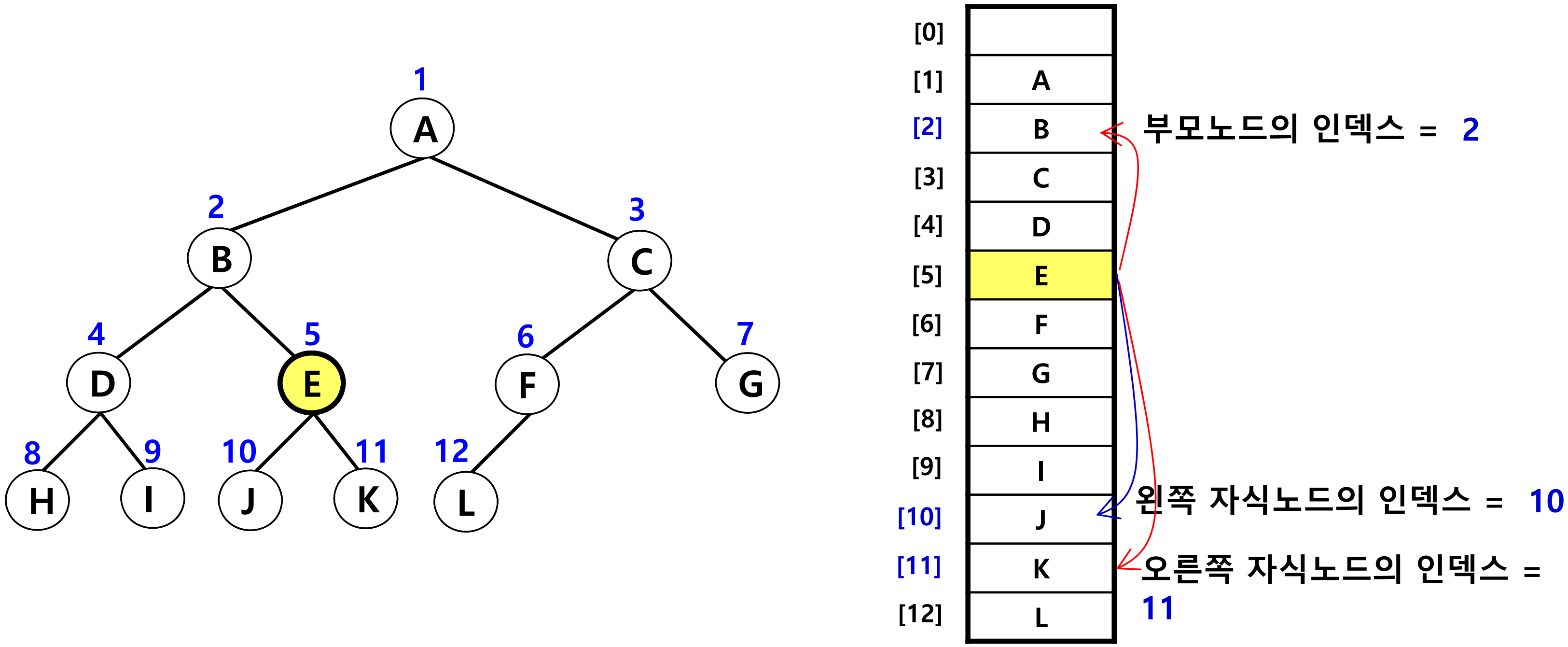
[그림 8-11] 편향 이진 트리의 배열 표현

배열을 이용한 이진트리 개념 - IV

• 이진 트리의 1차원 배열에서의 인덱스 관계

노드	인덱스	성립 조건
노드 i의 부모 노드	$\lfloor i/2 \rfloor$	i > 1
노드 i의 왼쪽 자식 노드	2 x i	(2 x i) ≤ n
노드 i의 오른쪽 자식 노드	(2 x i) + 1	(2 x i + 1) ≤ n
루트 노드	1	n > 0

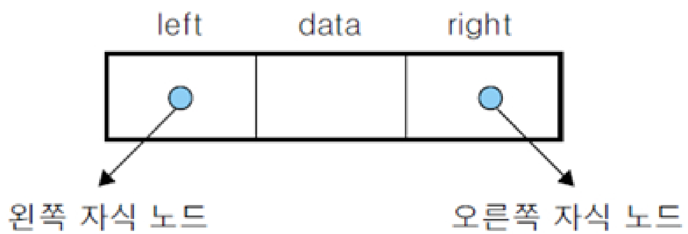
전체 노드의 수 = n
I = 노드의 수



연결리스트를 이용한 이진트리 개념 - I

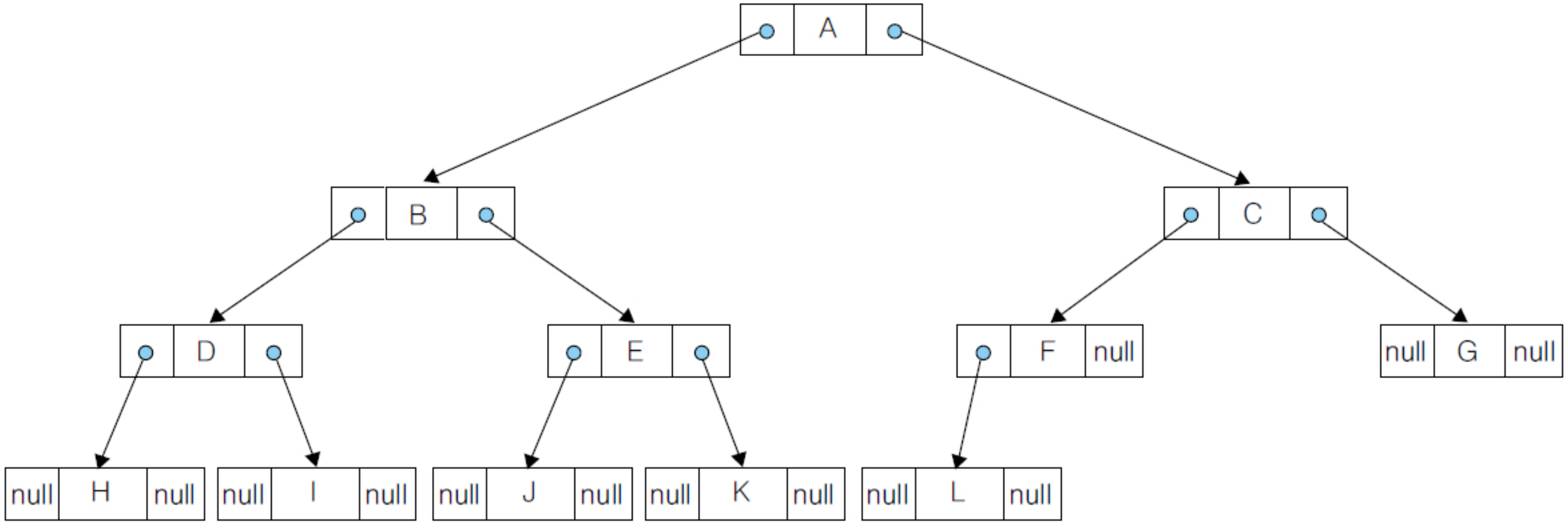
- **이진 트리의 순차 자료구조 표현의 단점**
  - 편향 이진 트리의 경우에 사용하지 않는 배열 원소에 대한 메모리 공간 낭비 발생
  - 트리의 원소 삽입/삭제에 대한 배열의 크기 변경 어려움

- **연결 자료구조를 이용한 이진트리의 구현**
  - 단순 연결 리스트를 사용하여 구현
  - 이진 트리의 모든 노드는 최대 2개의 자식 노드를 가지므로 일정한 구조의 단순 연결 리스트 노드를 사용하여 구현



연결리스트를 이용한 이진트리 개념 - III

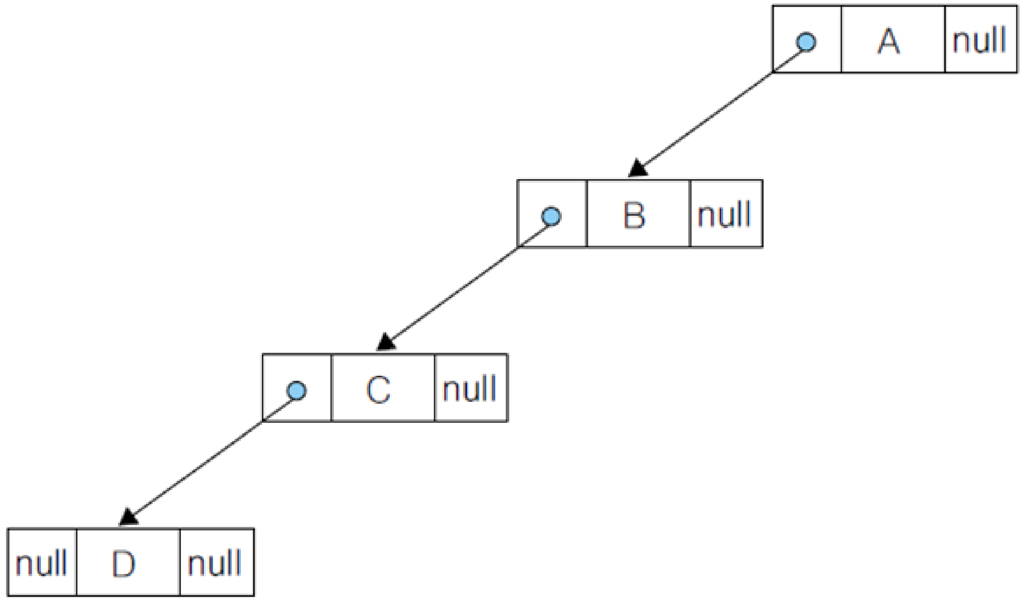
- **완전 이진 트리의 단순 연결 리스트 표현**



(a) 완전 이진 트리

연결리스트를 이용한 이진트리 개념 - IV

- **왼쪽 편향 이진 트리의 단순 연결 리스트 표현**



(b) 편향 이진 트리

연결리스트를 이용한 이진트리 구현 - 헤더파일 정의

- 이진 트리 노드를 표현한 구조체
  - 이진 트리가 모든 노드가 직/간접적으로 연결되어 있기 때문에 루트 노드의 주소값만 안다면, 이진 트리 전체를 알 수 있음
  - 하나의 노드는 그 자체로 이진 트리
    - 구조체를 정의하기 위해서는 실제로 노드를 표현한 구조체 자체가 이진 트리를 표현한 구조체여야 함

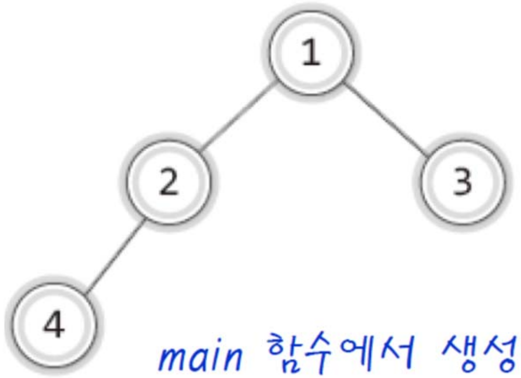
```
typedef struct _bTreeNode{
	BTData data;
	struct _bTreeNode * left;
	struct _bTreeNode * right;
}BTreeNode;
```

연결리스트를 이용한 이진트리 구현 - ADT

- BTreeNode * MakeBTreeNode(void);
  - 노드의 생성 (데이터는 setData를 통해, right와 left는 Null로 자동초기화)
- BTData GetData(BTreeNode *bt);
  - 노드에 저장된 데이터를 반환
- Void setData(BTreeNode * bt, BTData data);
  - 노드에 데이터를 저장
- BTreeNode * GetLeftSubTree(BTreeNode * bt);
  - 왼쪽 서브 트리의 주소 값 반환
- BTreeNode * GetRightSubTree(BTreeNode * bt);
  - 오른쪽 서브 트리의 주소 값 반환
- void MakeLeftSubTree(BTreeNode * main, BTreeNode * sub);
  - Main의 서브 왼쪽 서브 트리로 sub를 연결
- Void MakeRightSubTree(BTreeNode * main, BTreeNode * sub);
  - Main의 오른쪽 서브 트리로 sub를 연결

연결리스트를 이용한 이진트리 구현 - main 함수

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

        SetData(bt1, 1);//bt1에 1 저장
        SetData(bt2, 2);//bt2에 2 저장
        SetData(bt3, 3);//bt3에 3 저장
        SetData(bt4, 4);//bt4에 4 저장

        MakeLeftSubTree(bt1, bt2);//bt2를 bt1의 왼쪽 자식 노드로
        MakeRightSubTree(bt1, bt3);//bt3를 bt1의 왼쪽 자식 노드로
        MakeLeftSubTree(bt2, bt4);//bt4를 bt2의 왼쪽 자식 노드로

        // bt의 왼쪽 자식 노드 데이터 출력
        printf("%d \n", GetData(GetLfetSubTree(bt1)));
        // bt의 왼쪽 자식 노드의 왼쪽 자식 노드의 데이터 출력
        printf("%d \n", GetData(GetLfetSubTree(GetLeftSubTree(bt1))));

        return 0
}
```



main 함수에서 생성하는 트리

연결리스트를 이용한 이진트리 구현 - 기능구현 I

SUNG KYUN KWAN UNIVERSITY

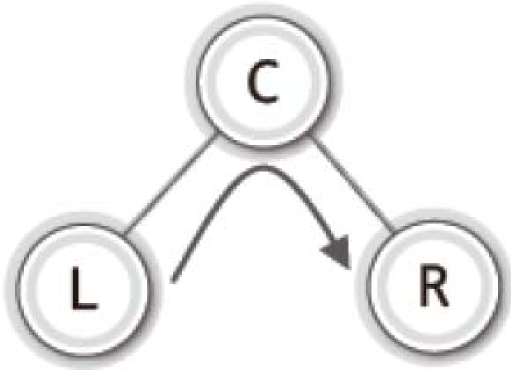
```
BTreeNode * MakeBTreeNode(void) {
	BTreeNode * nd = (BTreeNode*)malloc(sizeof(BTreeNode));
	nd->left = NULL;
	nd->right = NULL;
	return nd;
}

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

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

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

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

연결리스트를 이용한 이진트리 구현 - 기능구현 II

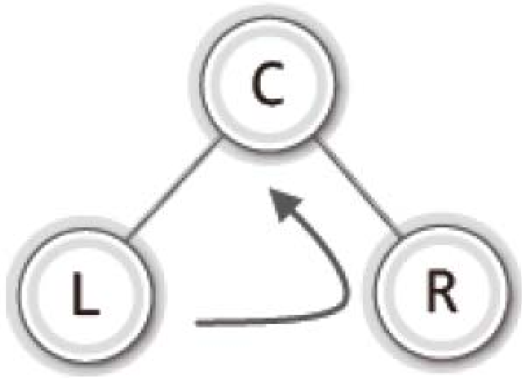
```
void MakeLeftSubTree(BTreeNode * main, BTreeNode * sub) {
	if (main->left != NULL)			// 기존에 연결된 노드가 있으면 삭제되게 구현한 코드
		free(main->left);			// subtree가 하나의 노드면 괜찮지만 그렇지 않으면 문제가 됨

	main->left = sub;
}


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

	main->right = sub;
}
```

이진 트리 순회 방법 : 3가지 (중위, 후위, 전위)

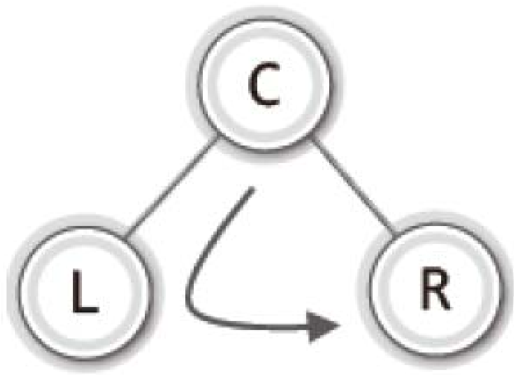
- 기준
  - 루트 노드를 언제 방문하느냐?
    - 처음 : 전위, 중간 : 중위, 마지막, 후위



중위



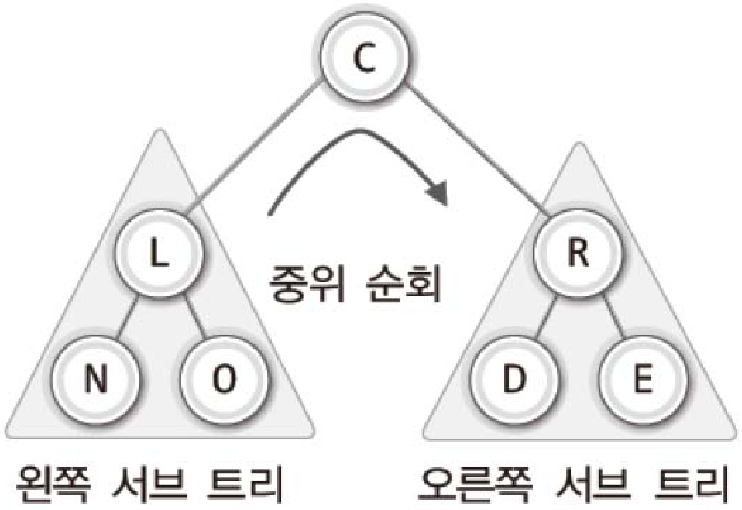
후위



전위

이진 트리 순회 방법 : 중위순회

- 단계
  - 1단계 : 왼쪽 서브트리 순회
  - 2단계 : 루트 노드 방문
  - 3단계 : 오른쪽 서브트리 순회




```
void InorderTraverse(BTreeNode *bt){
	if (bt == NULL)
		return;

	InorderTraverse(bt->left);       //1단계
	printf( " %d \n " , bt->data);  //2단계
	InorderTraverse(bt->right);     //3단계
}
```

이진 트리 순회 방법 : 전위순회

- 단계
  - 1단계 : 루트 노드 방문
  - 2단계 : 왼쪽 서브트리 순회
  - 3단계 : 오른쪽 서브트리 순회

```
void PreorderTraverse(BTreeNode *bt){
	if (bt == NULL)
		return;

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

이진 트리 순회 방법 : 후위순회

SUNG KYUN KWAN UNIVERSITY

- 단계
  - 1단계 : 왼쪽 서브트리 순회
  - 2단계 : 오른쪽 서브트리 순회
  - 3단계 : 루트 노드 방문

```
void PostorderTraverse(BTreeNode *bt){
    if (bt == NULL)
        return;

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

노드의 방문 자유롭게 구성

- 함수 포인터 형 VisitFuncPtr 정의

```
typedef void VisitFuncPtr(BTData data);  // 헤더에 정의
```

- Action이 가리키는 함수를 통해 방문을 진행

```
void InorderTraverse(BTreeNode *bt, VisitFuncPtr action){
        if (bt == NULL)
                return;

        InorderTraverse(bt->left);
        action(bt->data);
        InorderTraverse(bt->right);
}
```

- VisitFuncPrt형을 기준으로 정의된 함수 정의

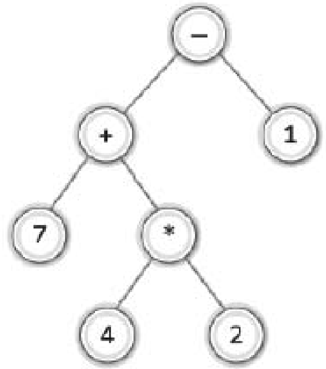
```
void ShowIntData(int data){
        printf("%d ", data);
}
```

수식 트리의 개념적 이해 - I

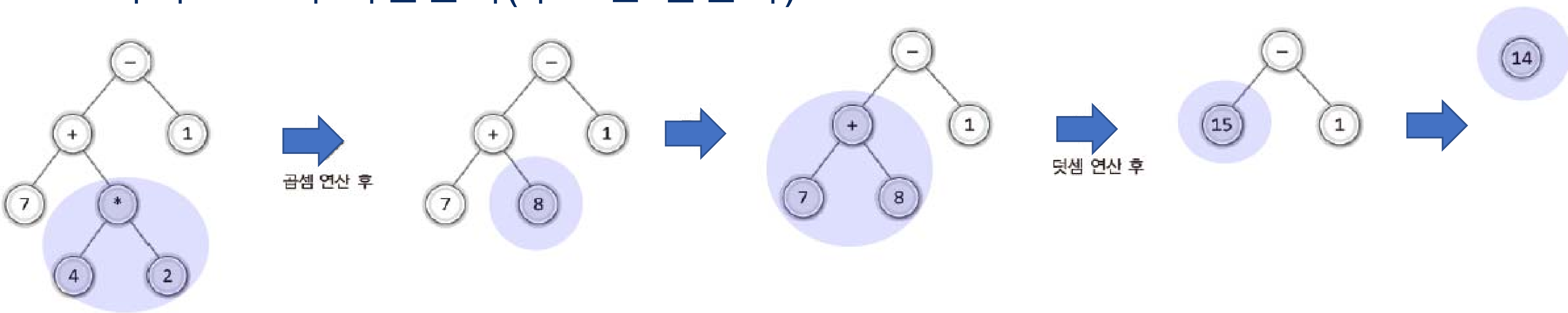
- 수식 트리
  - 이진 트리를 이용해서 수식을 표현해 놓은 방법
  - 컴퓨터 컴파일러는 중위 표기법의 수식을 수식 트리 형태로 재구성
    - 해석이 쉽고, 연산의 과정에서 우선순위를 고려하지 않아도 됨(미리 되어 있으니)

```
int main(void)
{
    int result = 0;
    result = 7 + 4 * 2 – 1;
    . . . .
}
```

수식 트리



- 수식 트리 계산과정
  - 자식 노드가 피연산자(부모는 연산자)



- 수식 트리를 만드는 절차
  - 중위 표기법의 수식 → 후위 표기법의 수식 → **수식 트리**
  - 이전 ConvToRPNExp에서 후위 표기법으로 전환이 되었으니 이를 수식트리로 만으로 전환만 하면 됨

수식 트리 구현 - 헤더파일 및 ADT

- 이전 헤더파일 _bTreeNode를 그대로 사용
- BTreeNode * MakeExpTree(char exp[])
  - 후위 표기법의 수식을 인자로 받아서 수식 트리를 구성
  - 루트 노드의 주소 값을 반환
- Int EvaluateExpTree(BTreeNode *bt)
  - 수식 트리의 수식을 계산하여 그 결과를 반환
- Void ShowPrefixTypeExp(BTreeNode *bt)
  - 전위 표기법 기반 출력(전위 순회)
- Void ShowInfixTypeExp(BTreeNode *bt)
  - 중위 표기법 기반 출력(중위 순회)
- Void ShowPostfixTypeExp(BTreeNode *bt)3
  - 후위 표기법 기반 출력(후위 순회)

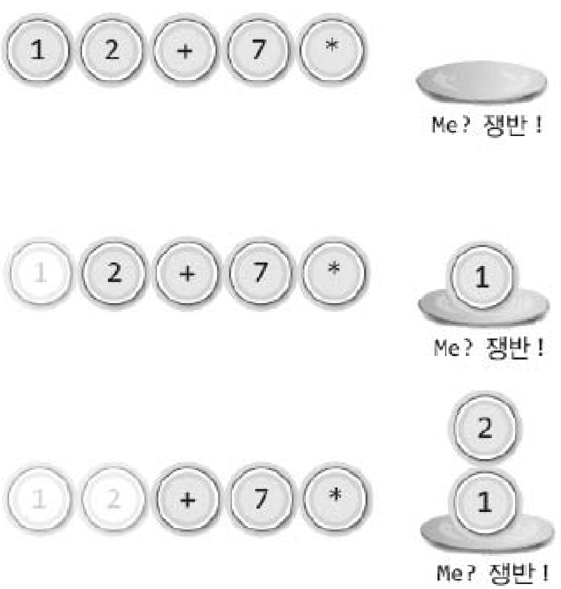
수식 트리 구현 - 트리 만들기(MakeExpTree) 구현 I

- 후위 표기법의 수식에서 먼저 등장하는 피연산자와 연산자를 이용해서 트리의 하단부터 구성→ 점진적으로 윗부분 구성

- **피연산자를 만나면 무조건 스택으로 옮김**
- **연산자를 만나면 스택에서 두 개의 피연산자를 꺼내어 자식 노드로 연결**
- **자식 노드를 연결해서 만들어진 트리는 다시 스택으로 옮김**

수식 트리 구현 - 트리 만들기(MakeExpTree) 구현 II

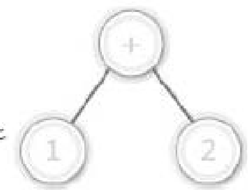
- Stack을 이용하여 구성방법

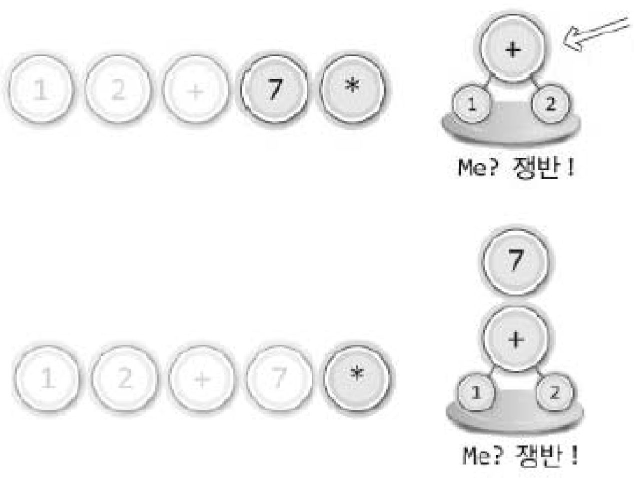


피연산자는 스택에 저장

연산자를 만나면 스택에서 피연산자 두 개를 꺼내어 트리 구성

형성된 트리는 다시 스택으로





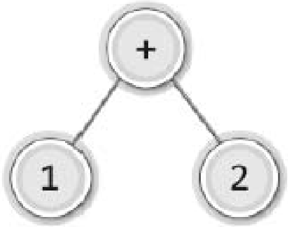
같은 방법으로 스택에서 꺼냄

수식 트리 구현 - 트리 만들기(MakeExpTree) 구현 III

```
BTreeNode * MakeExpTree(char exp[]) {
    Stack stack;            //피연산자를 위한 스택 생성
    BTreeNode * pnode;          //트리를 위한 트리 변수 정의
    int expLen = strlen(exp);
    StackInit(&stack);

    for(i=0; i<expLen; i++) {
        pnode = MakeBTreeNode();   // 노드 생성
        if(isdigit(exp[i]) {
            SetData(pnode, exp[i]-'0');
        } else {
            makeRightSubTree(pnode, SPop(&stack));    //오른쪽 자식에 저장
            makeLeftSubTree(pnode, SPop(&stack));     //왼쪽 자식에 저장
            SetData(pnode, exp[i]);         //자신의 값 저장
        }
        SPush(&stack, pnode);               //스택에 저장
    }

    return SPop(&Stack);            //마지막 남은 원소(루트) 반환
}
```

수식 트리 구현 - 계산(EvaluateExpTreee) I

```
int EvaluateExpTree(BTreeNode *bt) {
	int op1, op2;
	op1 = GetData(GetLeftSubTree(bt));
	op2 = GetData(GetRightSubTree(bt));

	switch(GetData(bt)) {
		case '+':
			return op1+op2;
		case '-':
			return op1-op2;
		case '*':
			return op1*op2;
		case '/':
			return op1/op2;
	}
	return 0;
}
```



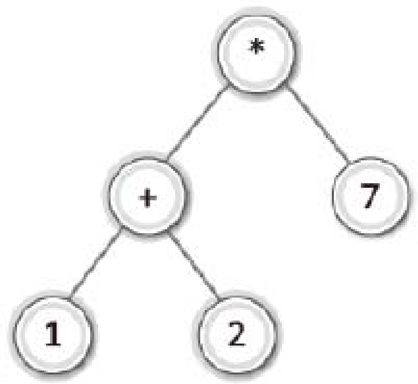
피연산자가 아닌 서브 트리가 달린 경우 문제가 됨.
즉, 재귀적 문제를 가짐
재귀는 반드시 종료 조건을 명시함

수식 트리 구현 - 계산(EvaluateExpTreee) II

```
int EvaluateExpTree(BTreeNode *bt) {
    int op1, op2;

    if(GetLeftSubTree(bt)==NULL && GetRightSubTree(bt)==NULL)
        return GetData(bt);

    op1 = EvaluateExpTree(GetLeftSubTree(bt));
    op2 = EvaluateExpTree(GetRightSubTree(bt));

    switch(GetData(bt)) {
        case '+':
            return op1+op2;
        case '-':
            return op1-op2;
        case '*':
            return op1*op2;
        case '/':
            return op1/op2;
    }
    return 0;
}
```

재귀함수 종료 조건으로 단말노드일 경우

재귀를 통해 서브트리 문제를 해결



수식 트리 구현 - 순회

SUNG KYUN KWAN UNIVERSITY

```
void ShowPrefixTypeExp(BTreeNode *bt) {
	PreorderTraverse(bt, ShowNodeData);
}

void ShowInfixTypeExp(BTreeNode *bt) {
	InrderTraverse(bt, ShowNodeData);
}

void ShowPostfixTypeExp(BTreeNode *bt) {
	PostorderTraverse(bt, ShowNodeData);
}

void ShowNodeData(int data) {
	if(0<=data && data <= 9)
		printf("%d ", data);
	else
		prinft("%d ", data);
}
```

수식 트리 구현 - main

```
int main(void) {
	char exp[] = "12+7*";
	BTreeNode * eTree = MakeExpTree(exp);

	printf("전위 표기법의 수식: ");
	ShowPrefixTypeExp(eTree); printf("\n");

	printf("전위 표기법의 수식: ");
	ShowPrefixTypeExp(eTree); printf("\n");

	printf("전위 표기법의 수식: ");
	ShowPrefixTypeExp(eTree); printf("\n");

	printf("연산의 결과: %d \n", EvaluateExpTree(eTree));

	return 0;
}
```

```
전위 표기법의 수식 : * + 1 2 7
중위 표기법의 수식 : 1 + 2 * 7
후위 표기법의 수식 : 1 2 + 7 *
연산의 결과 : 21
```