



시스템 프로그래밍 기초

Introduction to System Programming

ICT융합학부 조용우

10. Structures and List Processing



자기참조 구조체

- 자기참조 구조체는 자기 자신의형을 참조하는 포인터멤버를 가짐
- 이러한 자료구조를 동적자료구조라고 함
- 배열이나 단순변수는 일반적으로 블록을 진입할 때 메모리할당을 받지만, 동적자료구조는 기억장소관리루틴을 사용하여 명시적으로 메모리할당을 요구함

자기참조 구조체

■ 예제

```
struct list {  
    int data;  
    struct list *next;  
} a;
```

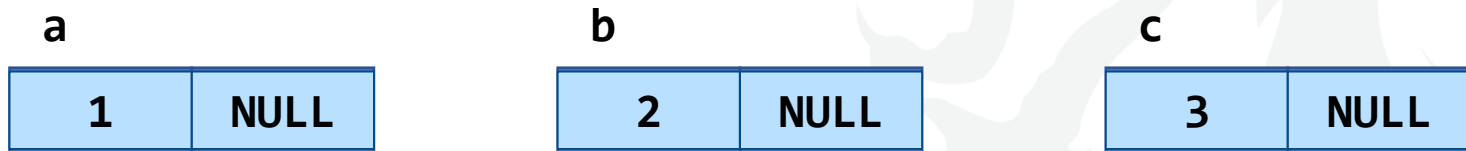


- 포인터 변수 `next`를 연결이라고 하고, 그림으로 나타낼 때 연결을 화살표로 표현하는 것이 좋음
- `next`는 다음 `list` 원소의 메모리주소나, `0`으로 정의되는 특별한 값 `NULL`을 가짐

자기참조 구조체

■ 동작 예제

```
struct list  a, b, c;  
a.data = 1;  
b.data = 2;  
c.data = 3;  
a.next = b.next = c.next = NULL;
```

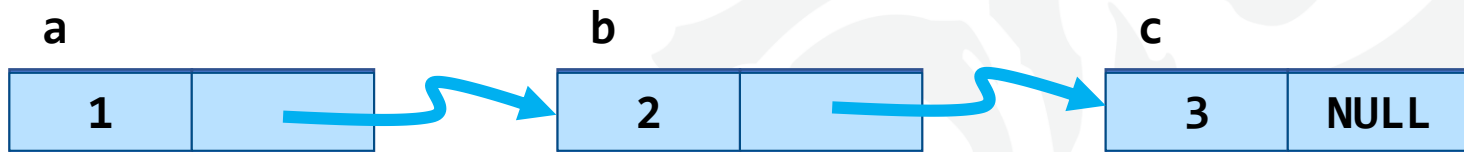


자기참조 구조체

■ 동작 예제

```
a.next = &b;
```

```
b.next = &c;
```



```
a.next -> data: 값2
```

```
a.next -> next -> data: 값3
```

선형 연결 리스트

- 선형 연결 리스트는 자료 구조체들이 순차적으로 매달려 있는 빨래줄과 같음
- 헤드 포인터는 이 리스트 상의 첫 번째 원소를 포인트하고, 각 원소는 다음 원소를 포인트 함
- 그리고 마지막 원소는 NULL값을 가짐

선형 연결 리스트 - 헤더 파일

```
/* In file list.h */

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

typedef char DATA; /* will use char in examples */

struct linked_list {
    DATA d;
    struct linked_list *next;
};

typedef struct linked_list ELEMENT;
typedef ELEMENT *LINK;
```


선형 연결 리스트

- 예제 - 세 개의 문자 n, e, w를 저장하는 선형연결 리스트를 동적으로 생성하는 코드

```
head = malloc(sizeof(ELEMENT));
```

```
head -> d = 'n';
```

```
head -> next = NULL;
```



선형 연결 리스트

- 예제 - 세 개의 문자 n, e, w를 저장하는 선형연결 리스트를 동적으로 생성하는 코드

```
head -> next = malloc(sizeof(ELEMENT));
```

```
head -> next -> d = 'e';
```

```
head -> next -> next = NULL;
```



선형 연결 리스트

- 예제 - 세 개의 문자 n, e, w를 저장하는 선형연결 리스트를 동적으로 생성하는 코드

```
head -> next -> next = malloc(sizeof(ELEMENT));
```

```
head -> next -> next -> d = 'w';
```

```
head -> next -> next -> next = NULL;
```



리스트 연산

■ 기본적인 선형 리스트 연산

1. 리스트 생성
2. 원소 개수 세기
3. 원소 탐색
4. 두 리스트 결합
5. 원소 삽입
6. 원소 삭제



주어진 문자열을 리스트로 변환 (재귀버전)

```
#include <stdlib.h>
#include "list.h"
LINK string_to_list(char s[]) {
    LINK head;
    if (s[0] == '\0') /* base case */
        return NULL;
    else {
        head = malloc(sizeof(ELEMENT));
        head -> d = s[0];
        head -> next = string_to_list(s + 1);
        return head;
    }
}
```

주어진 문자열을 리스트로 변환 (반복버전)

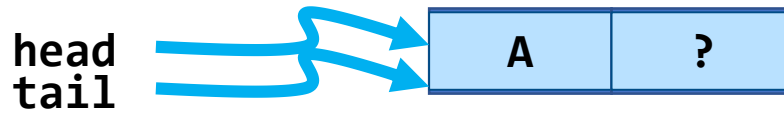
```
LINK s_to_l(char s[ ]){
    LINK    head = NULL, tail;
    int      i;
    if (s[0] != '\0') {                // first element
        head = malloc(sizeof(ELEMENT));
        head -> d = s[0];
        tail = head;
        for (i = 1; s[i] != '\0'; ++i) { // add to tail
            tail -> next = malloc(sizeof(ELEMENT));
            tail = tail -> next;
            tail -> d = s[i];
        }
        tail -> next = NULL;           // end of list
    }
    return head;
}
```

리스트 연산: `s_to_l("AB")` 호출에 의한 리스트 생성 단계



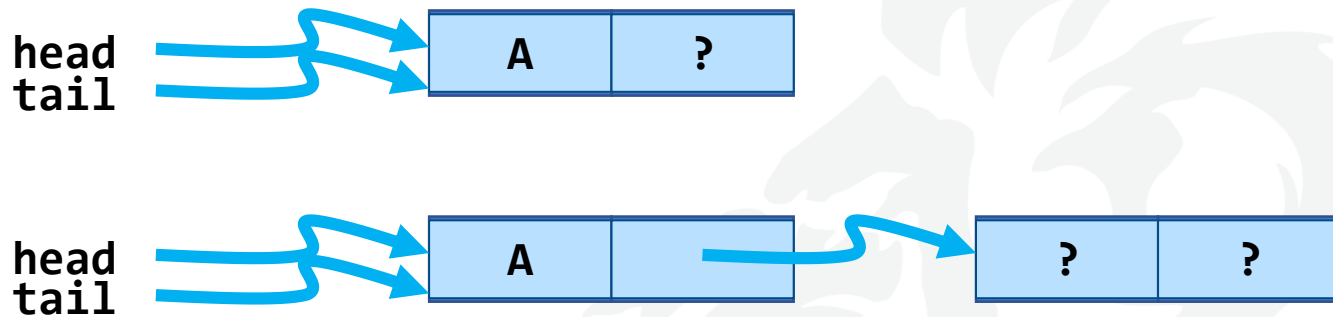
10.3 List Operations

리스트 연산: `s_to_l("AB")` 호출에 의한 리스트 생성 단계



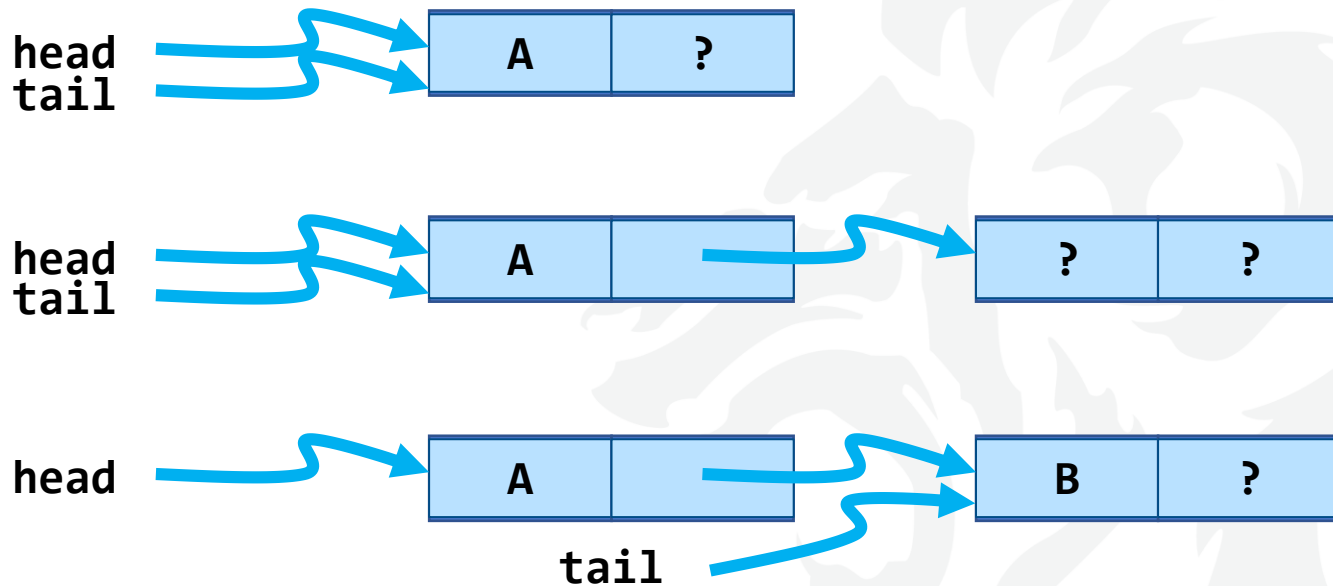
10.3 List Operations

리스트 연산: `s_to_l("AB")` 호출에 의한 리스트 생성 단계



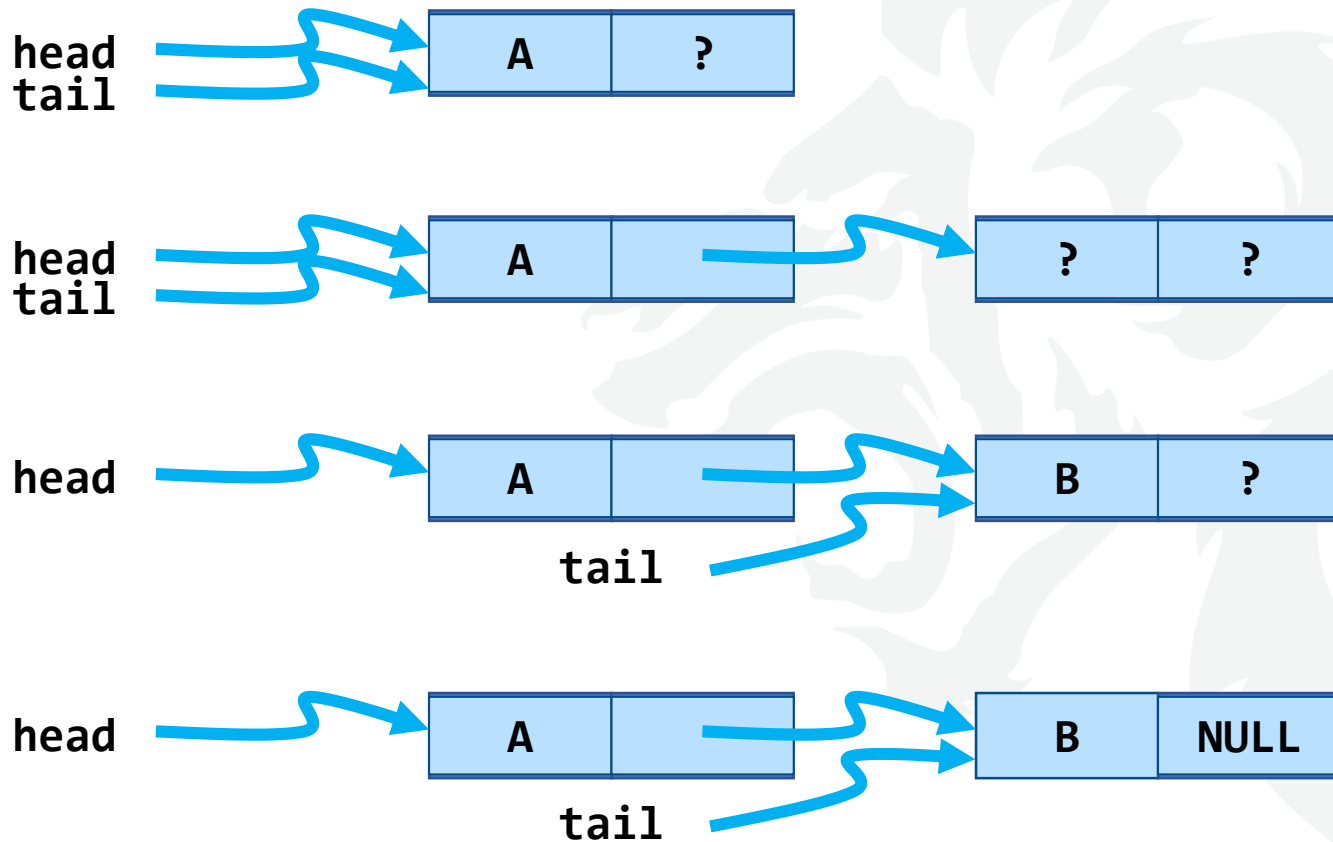
10.3 List Operations

리스트 연산: `s_to_l("AB")` 호출에 의한 리스트 생성 단계



10.3 List Operations

리스트 연산: `s_to_l("AB")` 호출에 의한 리스트 생성 단계



리스트 처리 함수 - 원소 수를 세는 함수 (재귀버전)

```
/* Count a list recursively. */  
  
int count(LINK head){  
    if (head == NULL)  
        return 0;  
    else  
        return (1 + count(head -> next));  
}
```

리스트 처리 함수 - 원소 수를 세는 함수 (반복버전)

```
/* Count a list iteratively. */  
  
int count_it(LINK head){  
    int cnt = 0;  
    for ( ; head != NULL; head = head -> next)  
        ++cnt;  
    return cnt;  
}
```

10.4 Some List Processing Functions

리스트 처리 함수 - 두 리스트를 연결하는 함수 (재귀버전)

```
/* Concatenate list a and b with a as head. */  
  
void concatenate(LINK a, LINK b){  
    assert(a != NULL);  
    if (a -> next == NULL)  
        a -> next = b;  
    else  
        concatenate(a -> next, b);  
}
```

10.4 Some List Processing Functions

리스트 처리 함수 - 자기참조 리스트를 처리하기 위한 재귀 함수의 일반적인 형태

```
void generic_recursion(LINK head){  
    if (head == NULL)  
        do the base case  
    else  
        do the general case and recur with  
        generic_recursion(head -> next)  
}
```

리스트 처리 함수 - 삽입

- 리스트에서 현재 위치에 새로운 원소를 삽입할 경우, 고정된 시간 내에 삽입을 할 수 있음
- 반대로 큰 배열에 새로운 값을 삽입할 경우, 배열 값의 순서를 유지해야 하기 때문에 삽입에 걸리는 시간은 평균적으로 배열 길이에 비례함

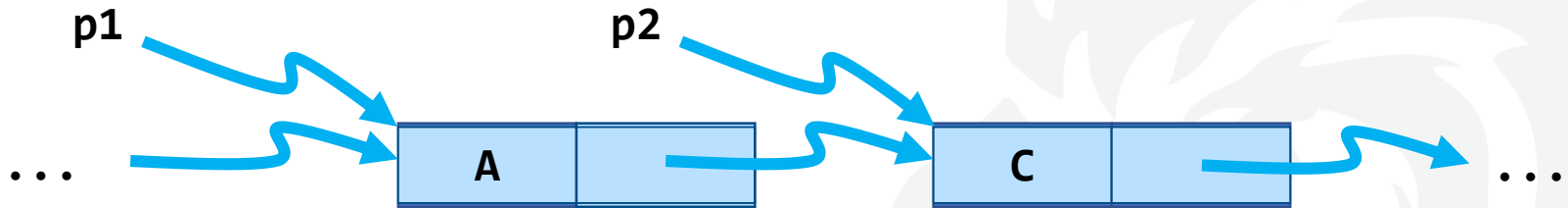
10.4 Some List Processing Functions

리스트에서 p1과 p2가 포인트하고 있는 인접한 두 원소 사이에 q가 포인트하고 있는 원소 삽입



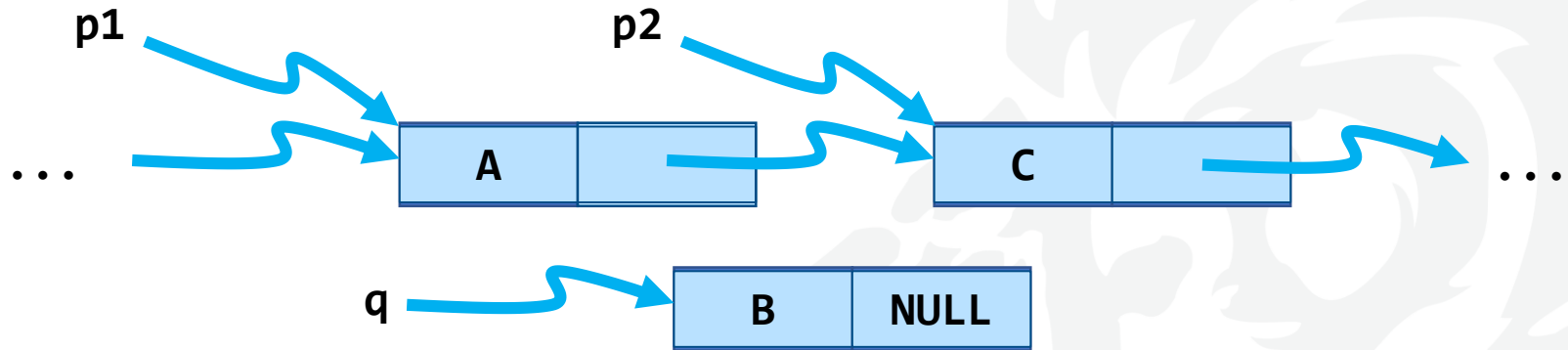
10.4 Some List Processing Functions

리스트에서 p1과 p2가 포인트하고 있는 인접한 두 원소 사이에 q가 포인트하고 있는 원소 삽입



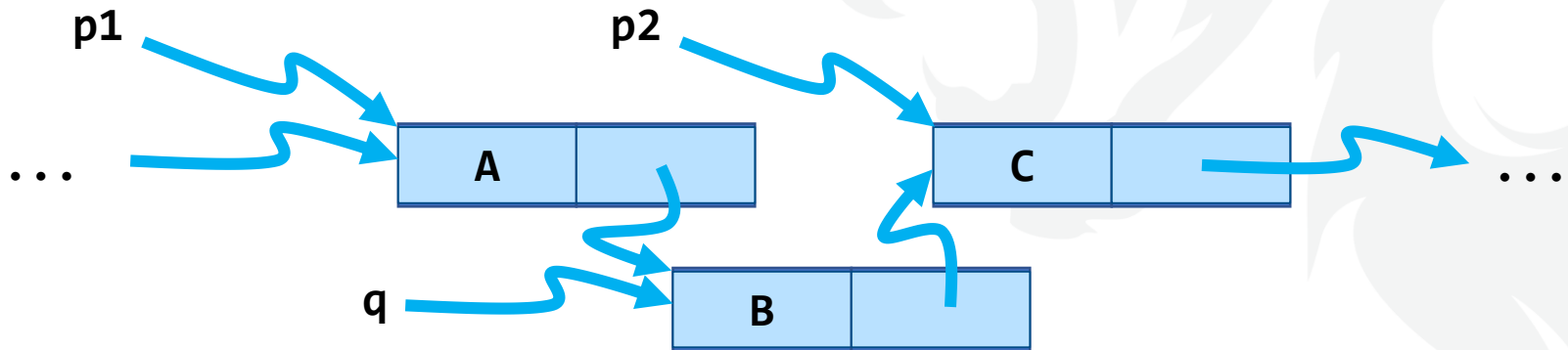
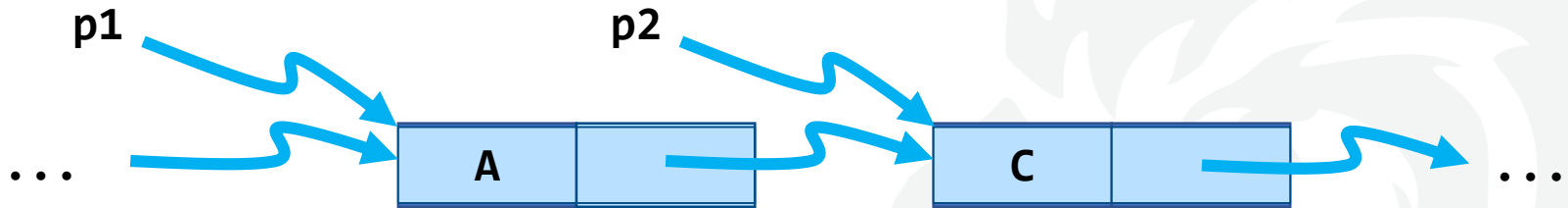
10.4 Some List Processing Functions

리스트에서 p1과 p2가 포인트하고 있는 인접한 두 원소 사이에 q가 포인트하고 있는 원소 삽입



10.4 Some List Processing Functions

리스트에서 p1과 p2가 포인트하고 있는 인접한 두 원소 사이에 q가 포인트하고 있는 원소 삽입

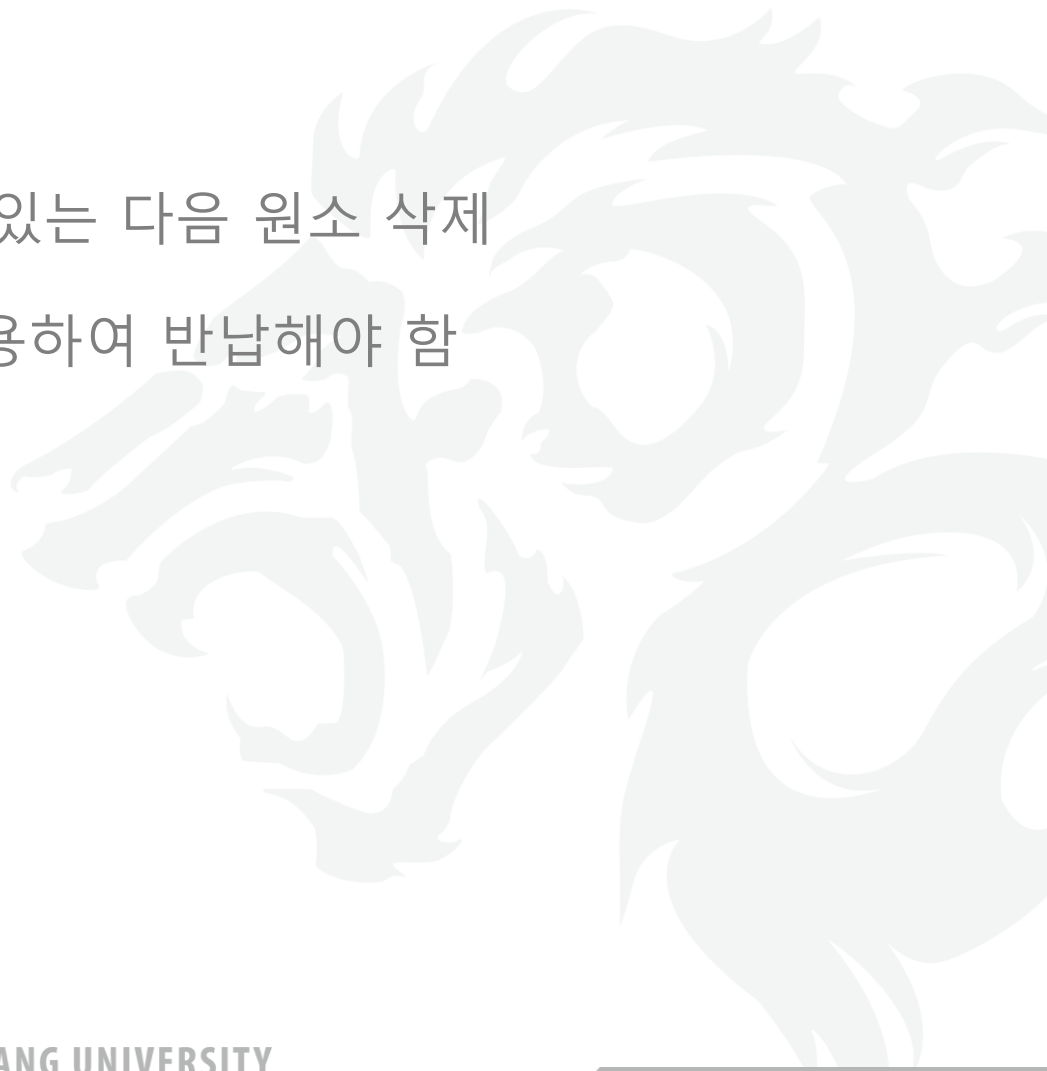


리스트 처리 함수 - 삽입 함수

```
/* Inserting an element in a linked list. */  
  
void insert(LINK p1, LINK p2, LINK q){  
    assert(p1 -> next == p2);  
    p1 -> next = q; /* insert */  
    q -> next = p2;  
}
```

리스트 처리 함수 - 삭제

- 리스트에서 p가 포인트하고 있는 다음 원소 삭제
- 삭제되는 원소를 free()를 사용하여 반납해야 함



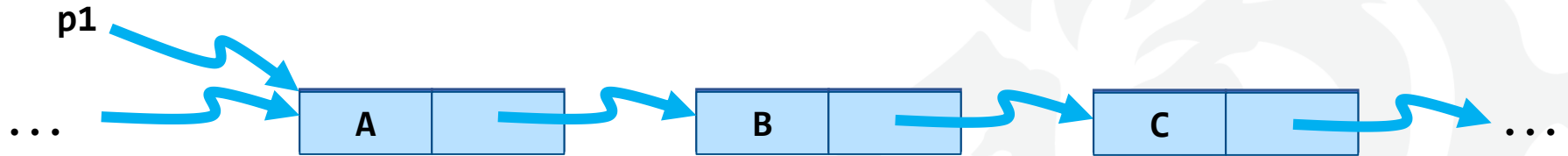
10.4 Some List Processing Functions

리스트에서 p 가 포인팅하고 있는 다음 원소 삭제



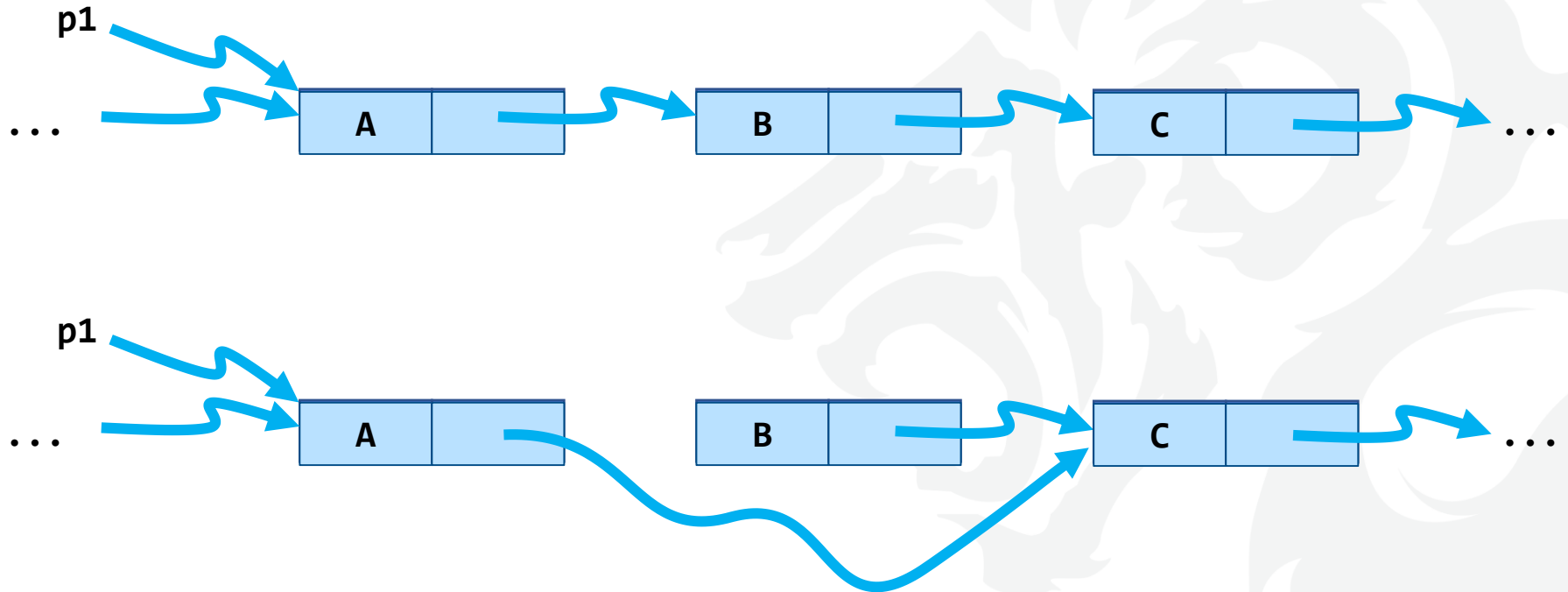
10.4 Some List Processing Functions

리스트에서 p가 포인트하고 있는 다음 원소 삭제



10.4 Some List Processing Functions

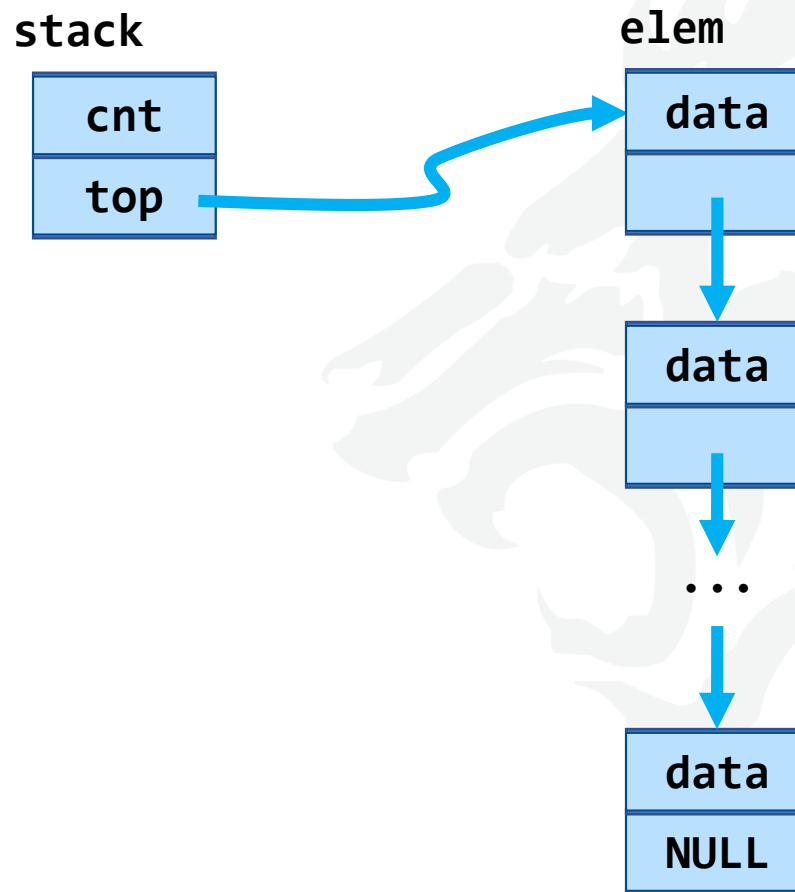
리스트에서 p가 포인트하고 있는 다음 원소 삭제



리스트 처리 함수 - 삭제 함수

```
/* Recursive deletion of a list. */  
  
void delete_list(LINK head) {  
    if (head != NULL) {  
        delete_list(head -> next);  
        free(head);    /* release storage */  
    }  
}
```

선형 연결 리스트로 구현한 스택



stack.h

```
#include <stdio.h>
#include <stdlib.h>
#define EMPTY 0
#define FULL 10000
typedef char data;
typedef enum {false, true} boolean;

struct elem {          // element on the stack
    data d;
    struct elem *next;
};
typedef struct elem elem;
```

stack.h

```
struct stack {  
    int    cnt;           // a count of the elements  
    elem  *top;          // ptr to the top element  
};  
typedef struct stack  stack;  
  
void      initialize(stack *stk);  
void      push(data d, stack *stk);  
data      pop(stack *stk);  
data      top(stack *stk);  
boolean   empty(const stack *stk);  
boolean   full(const stack *stk);
```

연산 구현

```
#include "stack.h"
void initialize(stack *stk){
    stk -> cnt = 0;
    stk -> top = NULL;
}
void push(data d, stack *stk){
    elem *p;
    p = malloc(sizeof(elem));
    p -> d = d;
    p -> next = stk -> top;
    stk -> top = p;
    stk -> cnt++;
}
```

연산 구현

```
data pop(stack *stk){
    data d;
    elem *p;
    d = stk -> top -> d;
    p = stk -> top;
    stk -> top = stk -> top -> next;
    stk -> cnt --;
    free(p);
    return d;
}
```

연산 구현

```
data top(stack *stk){  
    return (stk -> top -> d);  
}  
boolean empty(const stack *stk){  
    return ((boolean) (stk -> cnt == EMPTY));  
}  
boolean full(const stack *stk){  
    return ((boolean) (stk -> cnt == FULL));  
}
```


검사 프로그램

```
#include "stack.h"
int main(void){
    char  str[ ] = "My name is joanna Kelly!";
    int   i;      stack  s;
    initialize(&s); // initialize the stack
    printf(" In the string: %s\n", str);
    for (i = 0; str[i] != '\0'; ++i)
        if (!full(&s))
            push(str[i], &s); // push a char on the stack
    printf("From the stack: ");
    while (!empty(&s))
        putchar(pop(&s));    // pop a char off the stack
    putchar('\n');
    return 0;
}
```

```
In the string : My name is Jonna Kelly!
From the stack : !ylleK annoJ si eman yM
```

수식 표기법

- 중위 표기법
- $3 + 7$
- 전위 표기법
- $+, 3, 7$
- 후위 표기법 (폴리시 표기법, polish notation)



10.6 An Example: Polish Notation and Stack Evaluation

수식 표기법

- 중위 표기법

3 + 7

- 전위 표기법

+, 3, 7

- 후위 표기법 (폴리시 표기법, polish notation)

3, 7, +




중위 수식을 후위 수식으로 만드는 방법

1. 중위 수식을 연산자의 우선순위에 따라 완전한 괄호 표현으로 변환

예) $1 + 3 * 4 \rightarrow (1 + (3 * 4))$

2. 각 연산자를 그 연산자를 포함하는 괄호 중 제일 가까운 닫는 괄호 뒤로 보냄

예) $(1 + (3 * 4)) \rightarrow (1 (3, 4) *) +$



3. 3괄호 제거

예) $1, 3, 4, *, +$

폴리시 수식 계산

- 수식의 각 항목을 왼쪽에서 오른쪽으로 검사하며 다음과 같은 일을 반복한다.
 1. 현재 검사되는 항목이 숫자이면 다음 항목을 검사한다.
 2. 현재 검사되는 항목이 연산자이면 앞의 두 항목(숫자)을 이 연산자의 피연산자로 하여 계산한 후 새로운 항목으로 삽입한다.
 3. 이러한 일은 하나의 숫자가 남을 때까지 반복하고, 마지막 남은 수가 이 수식의 값이다.

폴리시 수식 계산 예제

■ 13, 4, -, 2, 3, *, +



10.6 An Example: Polish Notation and Stack Evaluation

폴리시 수식 계산 예제

■ 13, 4, -, 2, 3, *, +

① ② ③



폴리시 수식 계산 예제

- 13, 4, -, 2, 3, *, +

① ② ③

- 9, 2, 3, *, +

-



10.6 An Example: Polish Notation and Stack Evaluation

폴리시 수식 계산 예제

■ 13, 4, -, 2, 3, *, +

① ② ③

■ 9, 2, 3, *, +

④ ⑤ ⑥



10.6 An Example: Polish Notation and Stack Evaluation

폴리시 수식 계산 예제

■ 13, 4, -, 2, 3, *, +

① ② ③

■ 9, 2, 3, *, +

④ ⑤ ⑥

■ 9, 6, +

■



10.6 An Example: Polish Notation and Stack Evaluation

폴리시 수식 계산 예제

■ 13, 4, -, 2, 3, *, +

① ② ③

■ 9, 2, 3, *, +

④ ⑤ ⑥

■ 9, 6, +

■ 15



폴리시 수식 계산 - 2-스택 알고리즘

1. 폴리시 스택이 공백이면, 연산 스택의 top을 답으로 하고 중지한다.
2. 폴리시 스택이 공백이 아니면, 폴리시 스택에서 pop하여 그것을 d에 배정한다. (이 알고리즘에서는 자료 보관을 위해 d, d1, d2를 사용한다.)
3. d가 피연산자라면, 연산 스택에 d를 push한다.
4. d가 연산자라면, 연산 스택을 두 번 pop하여 처음 것을 d2에 배정하고 다음 것을 d1에 배정한다. d 연산자를 d1과 d2 피연산자에 적용하여 계산하고, 결과를 연산 스택에 push한다. 단계 1부터 다시 수행한다.

10.6 An Example: Polish Notation and Stack Evaluation

폴리시 수식 계산 - 2-스택 알고리즘 예제

폴리시 스택	연산 스택		
		13	
		4	
		-	
		2	
		3	
		*	
		+	



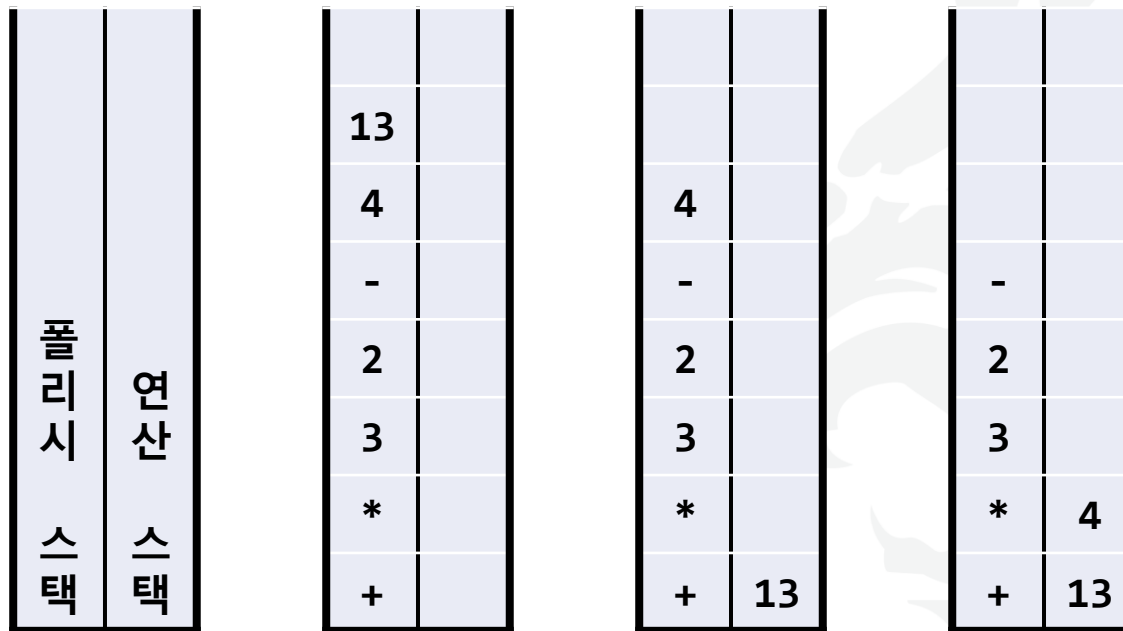
10.6 An Example: Polish Notation and Stack Evaluation

폴리시 수식 계산 - 2-스택 알고리즘 예제

폴리시 스택	연산 스택		
		13	
		4	
		-	
		2	
		3	
		*	
		+	
		4	
		-	
		2	
		3	
		*	
		+	13

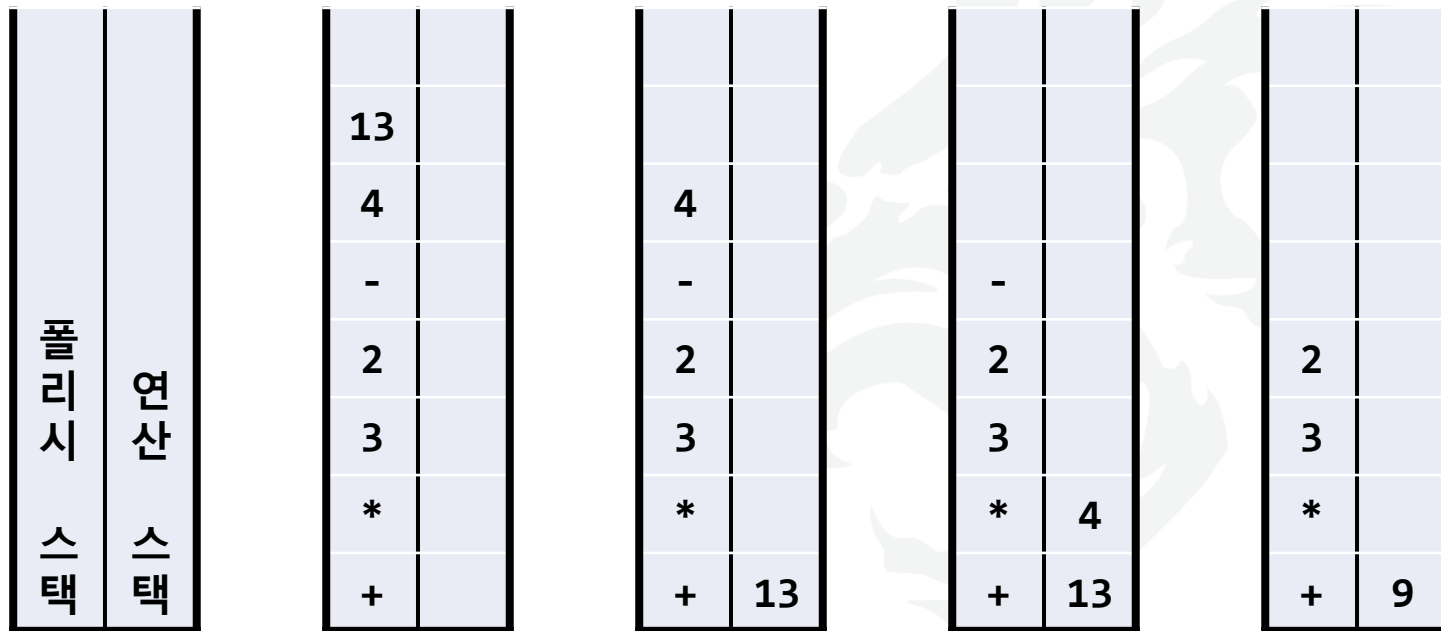
10.6 An Example: Polish Notation and Stack Evaluation

폴리시 수식 계산 - 2-스택 알고리즘 예제



10.6 An Example: Polish Notation and Stack Evaluation

폴리시 수식 계산 - 2-스택 알고리즘 예제



10.6 An Example: Polish Notation and Stack Evaluation

폴리시 수식 계산 - 2-스택 알고리즘 예제

2	
3	
*	
+	9



10.6 An Example: Polish Notation and Stack Evaluation

폴리시 수식 계산 - 2-스택 알고리즘 예제

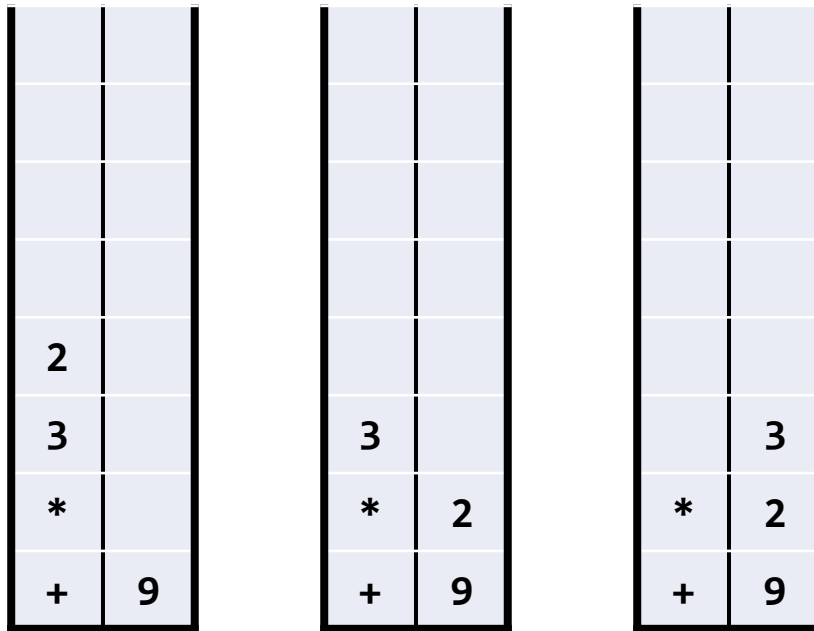
2			
3			
*			
+	9		

3			
*	2		
+	9		



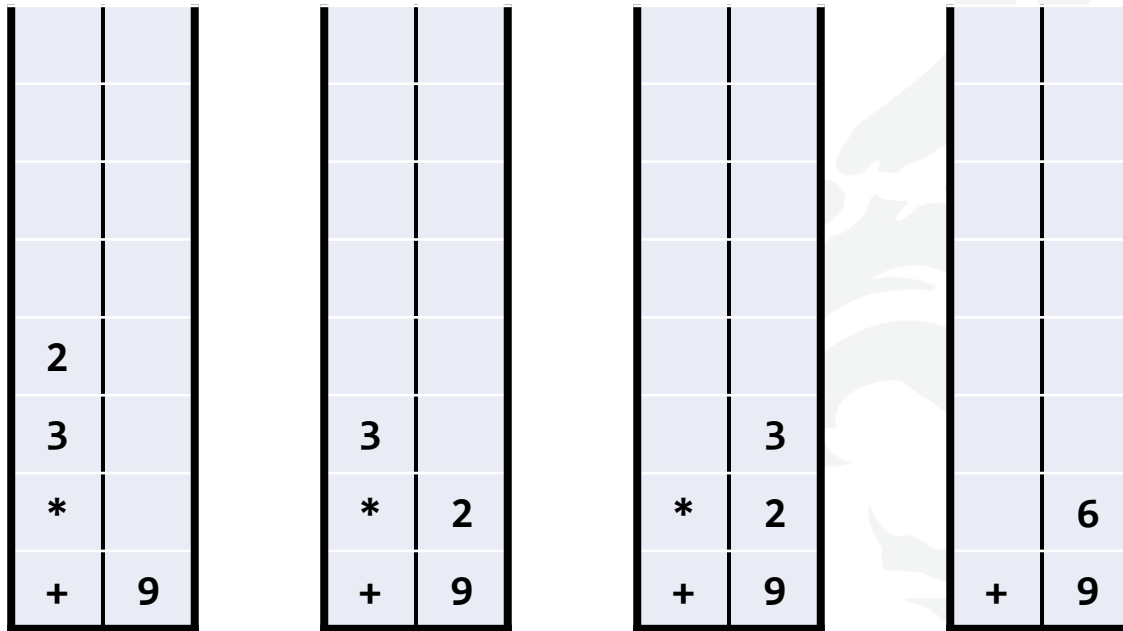
10.6 An Example: Polish Notation and Stack Evaluation

폴리시 수식 계산 - 2-스택 알고리즘 예제



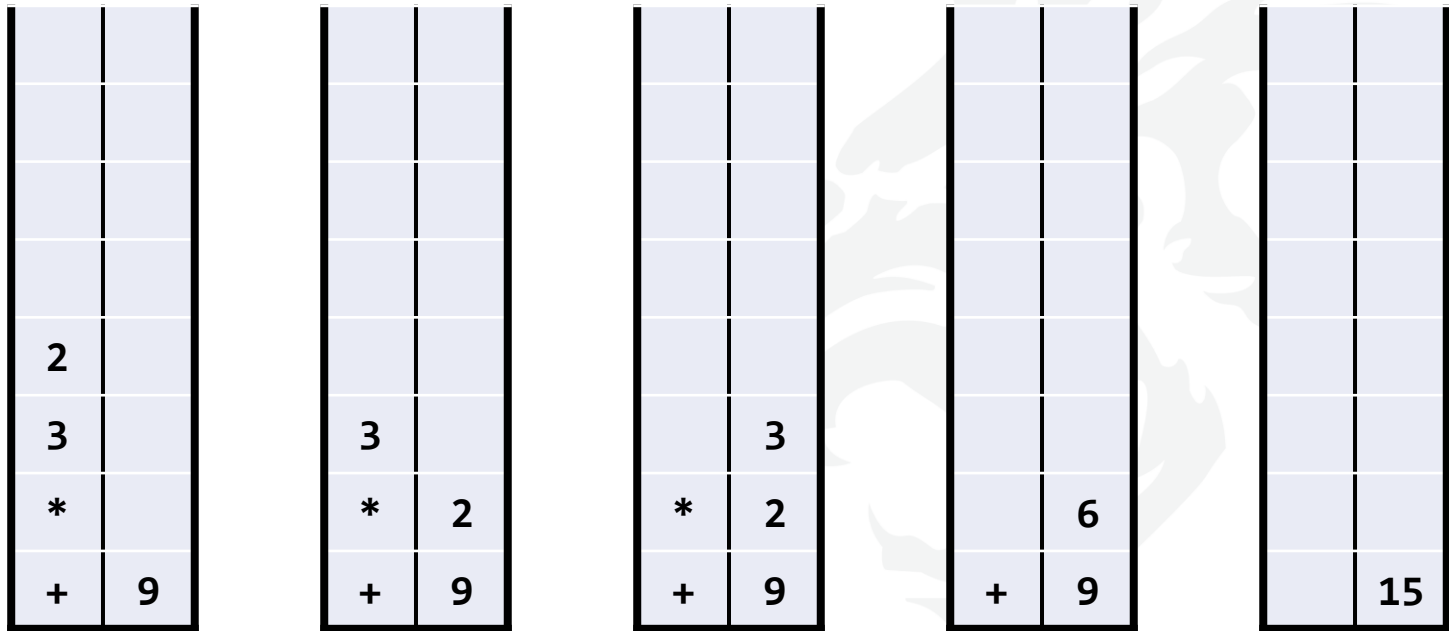
10.6 An Example: Polish Notation and Stack Evaluation

폴리시 수식 계산 - 2-스택 알고리즘 예제



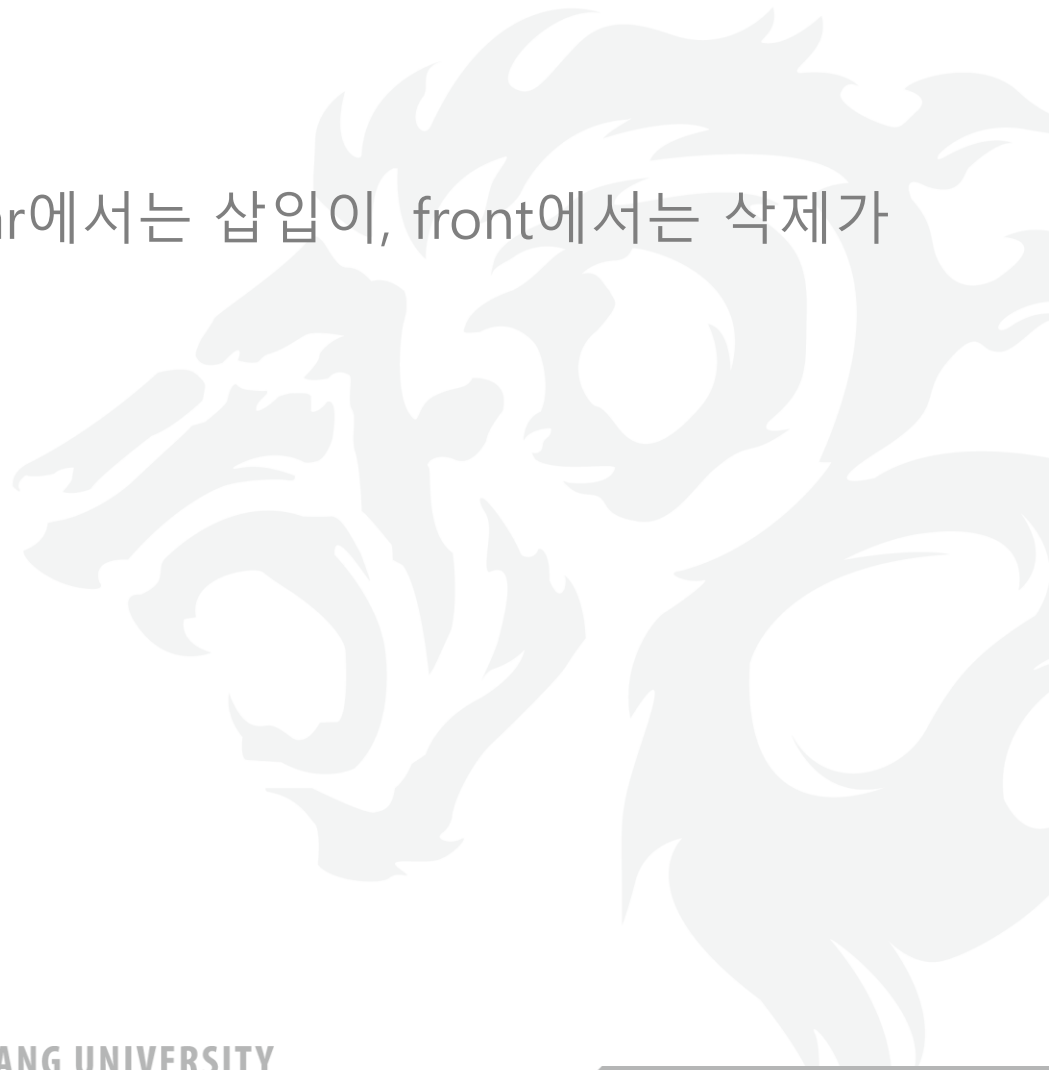
10.6 An Example: Polish Notation and Stack Evaluation

폴리시 수식 계산 - 2-스택 알고리즘 예제

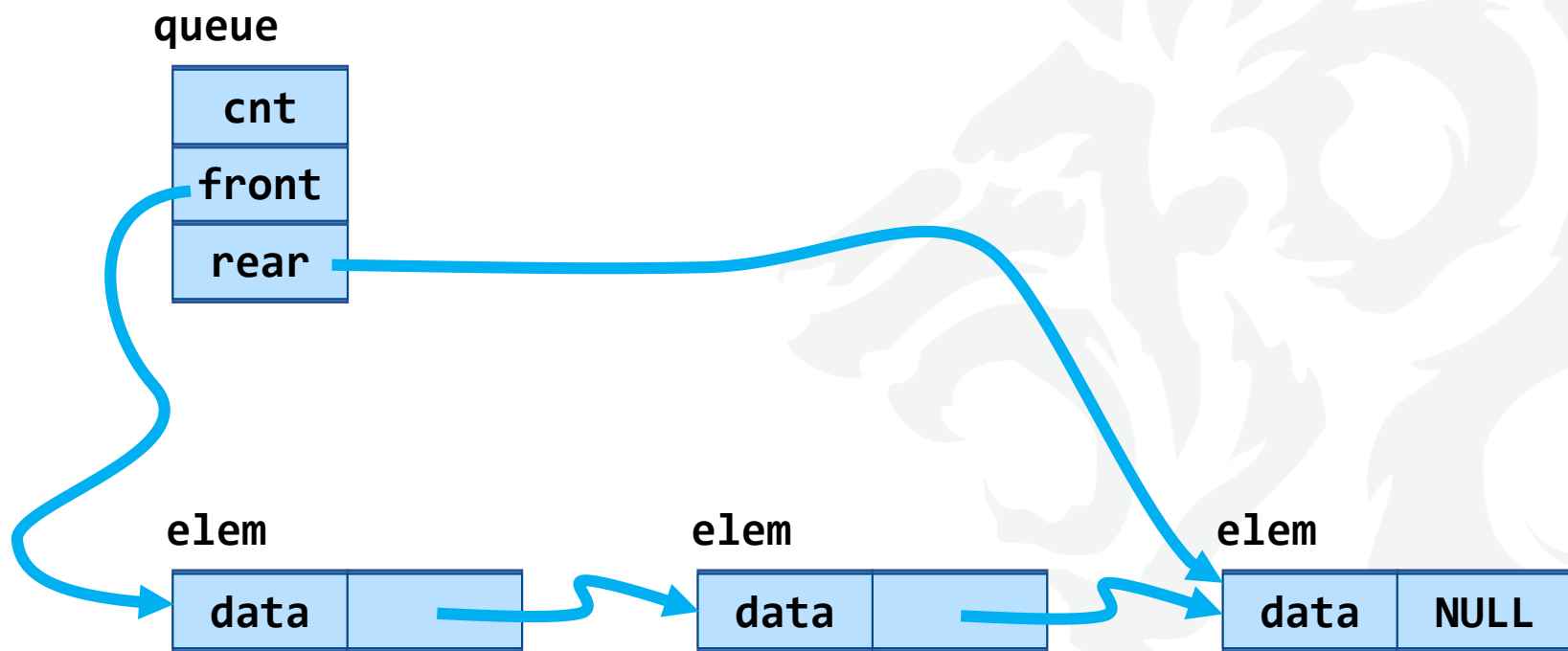


큐

- 큐는 front와 rear가 있고, rear에서는 삽입이, front에서는 삭제가 일어남



리스트로 구현한 큐



queue.h

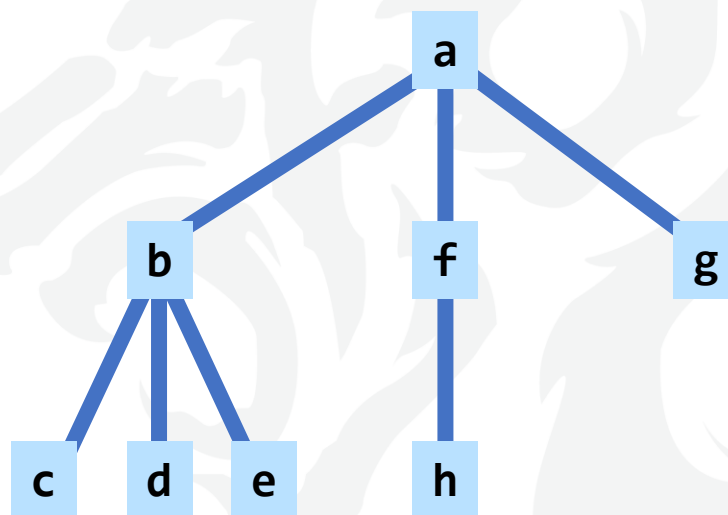
```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#define EMPTY      0
#define FULL       10000
typedef unsigned int    data;
typedef enum {false, true}  boolean;
struct elem {           // an element in the queue
    data      d;
    struct elem *next;
};
typedef struct elem elem;
```


queue.h

```
struct queue {  
    int    cnt;        // a count of the elements  
    elem   *front;     // ptr to the front element  
    elem   *rear;     // ptr to the rear element  
};  
typedef struct queue queue;  
void initialize(queue *q);  
void enqueue(data d, queue *q);  
data dequeue(queue *q);  
data front(const queue *q);  
boolean empty(const queue *q);  
boolean full(const queue *q);
```

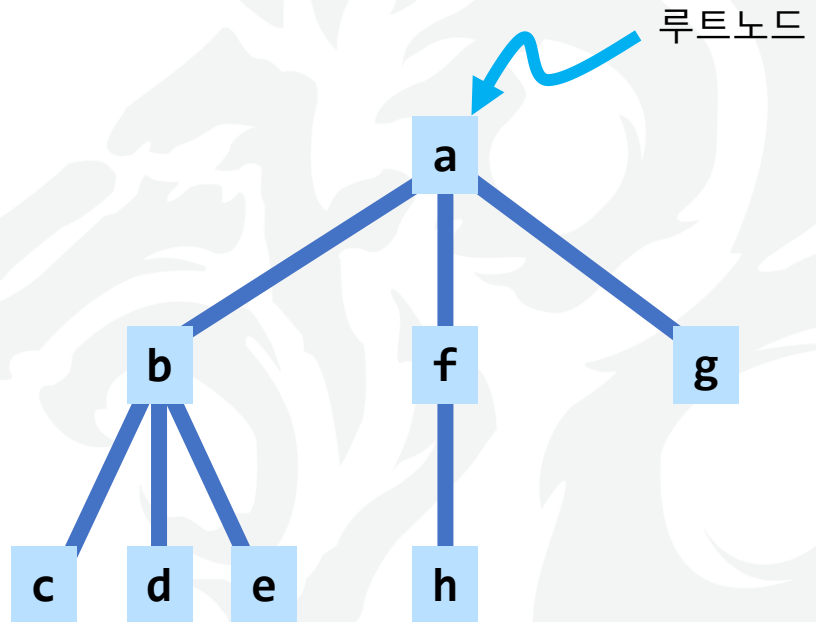
트리

- 노드라고 하는 원소의 유한 집합
- 루트, 리프 노드, 부트리



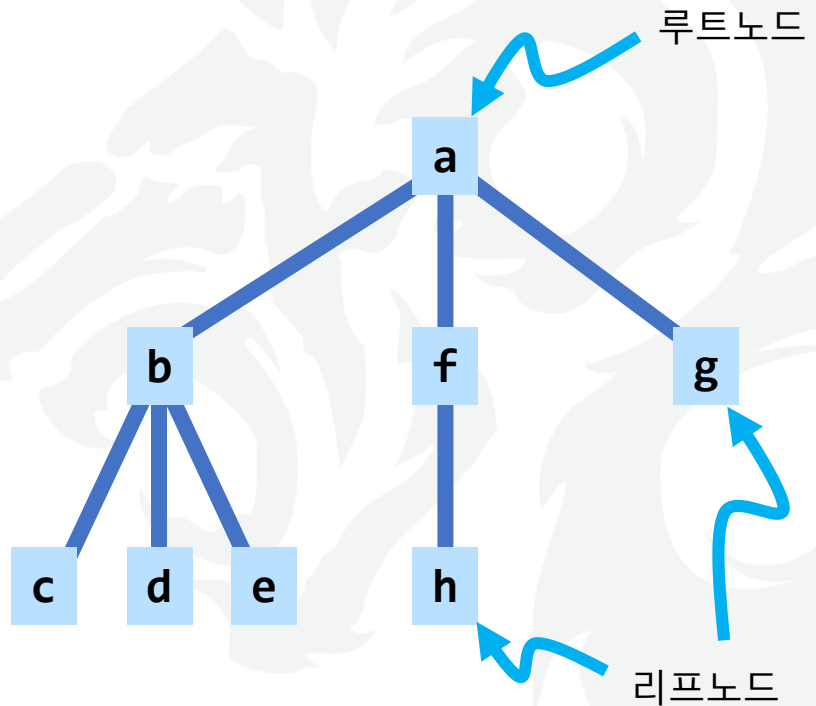
트리

- 노드라고 하는 원소의 유한 집합
- 루트, 리프 노드, 부트리



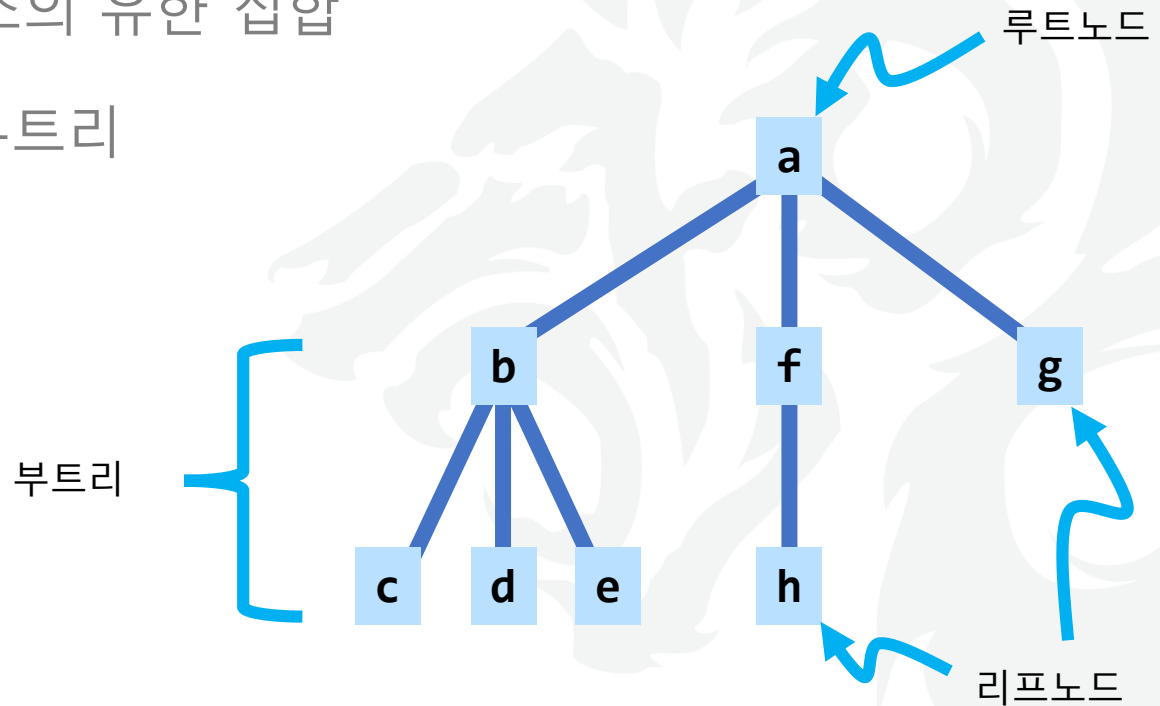
트리

- 노드라고 하는 원소의 유한 집합
- 루트, 리프 노드, 부트리



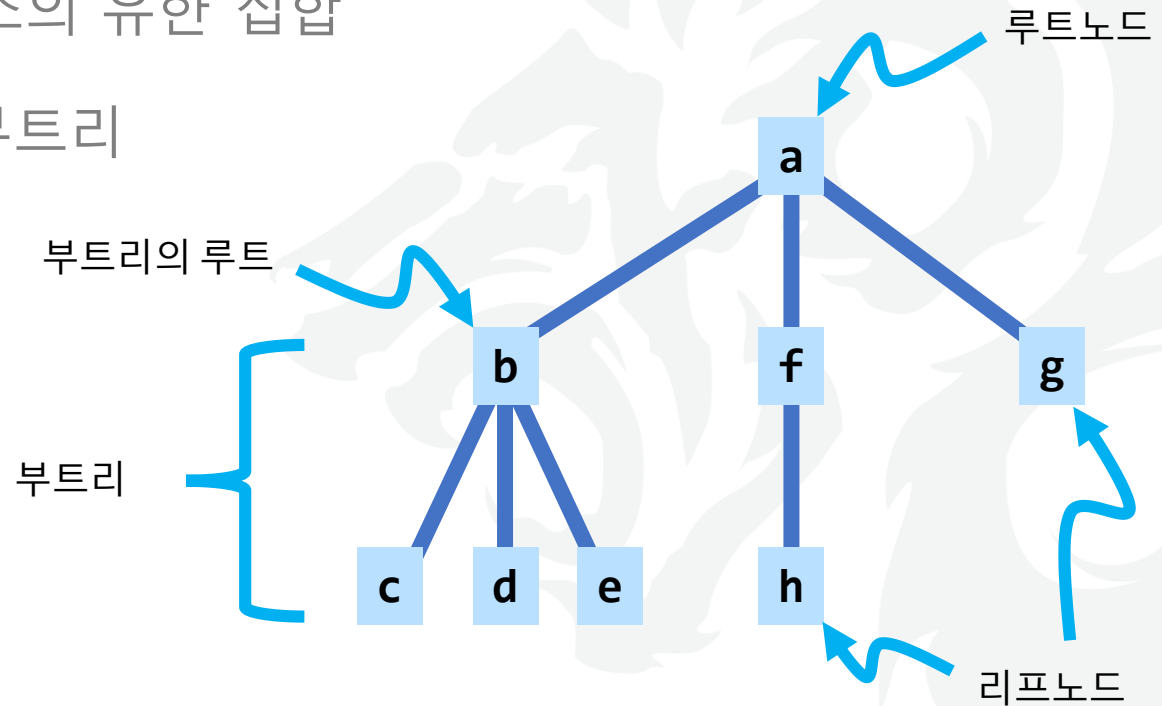
트리

- 노드라고 하는 원소의 유한 집합
- 루트, 리프 노드, 부트리



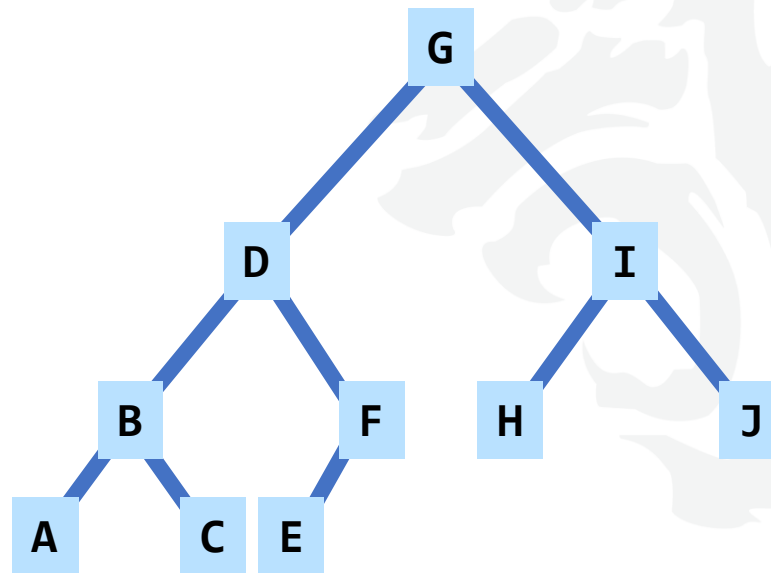
트리

- 노드라고 하는 원소의 유한 집합
- 루트, 리프 노드, 부트리



이진 트리

- 한 노드가 최대 2개의 자식 노드를 갖는 트리



이진 트리 - tree.h

```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
typedef char DATA;
struct node {
    DATA d;
    struct node *left;
    struct node *right;
};
typedef struct node NODE;
typedef NODE *BTREE;
#include "fct_proto.h" /* function prototypes */
```


이진 트리 순회

■ 중위순회

1. 왼쪽 부트리 방문
2. 루트 방문
3. 오른쪽 부트리 방문

■ 전위순회

1. 루트 방문
2. 왼쪽 부트리 방문
3. 오른쪽 부트리 방문

■ 후위순회

1. 왼쪽 부트리 방문
2. 오른쪽 부트리 방문
3. 루트 방문



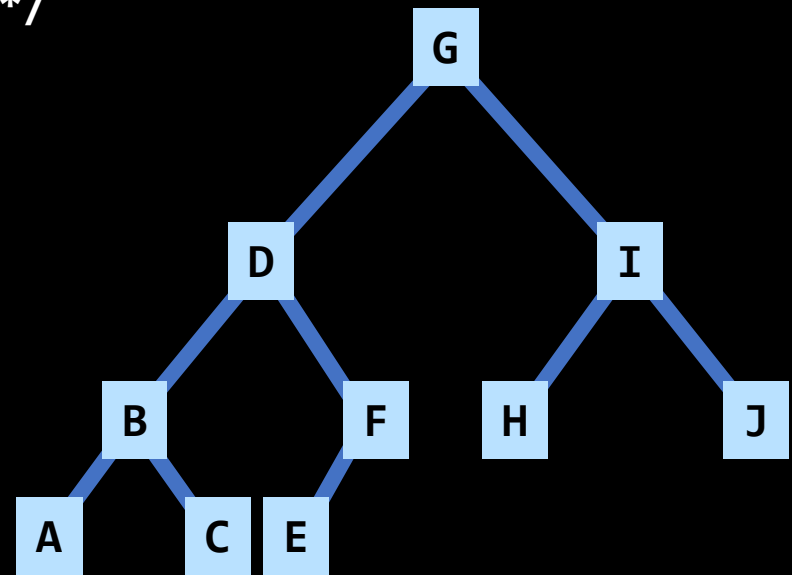
중위순회

```
/* Inorder binary tree traversal. */
```

```
void inorder(BTREE root){  
    if (root != NULL) {  
        inorder(root -> left);  
        printf("%c", root -> d);  
        inorder(root -> right);  
    }  
}
```

Output:

A B C D E F G H I J



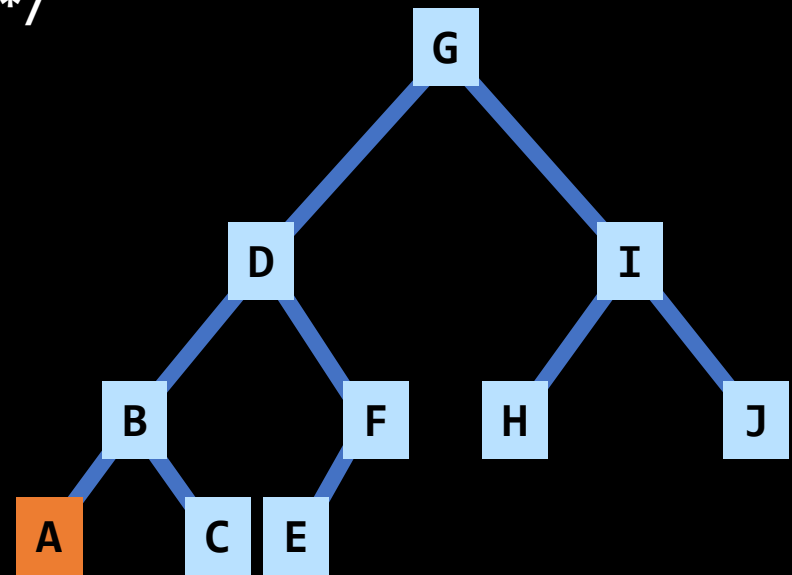
중위순회

```
/* Inorder binary tree traversal. */
```

```
void inorder(BTREE root){  
    if (root != NULL) {  
        inorder(root -> left);  
        printf("%c", root -> d);  
        inorder(root -> right);  
    }  
}
```

Output:

A B C D E F G H I J



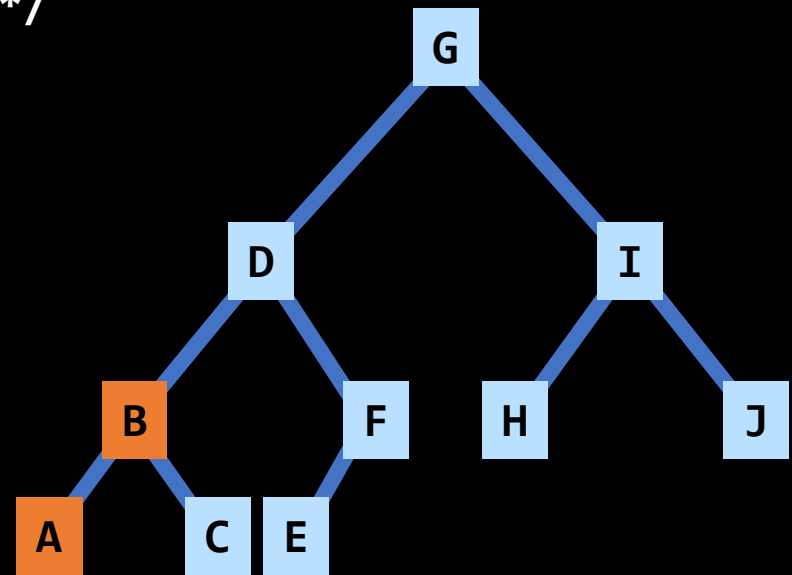
중위순회

```
/* Inorder binary tree traversal. */
```

```
void inorder(BTREE root){  
    if (root != NULL) {  
        inorder(root -> left);  
        printf("%c", root -> d);  
        inorder(root -> right);  
    }  
}
```

Output:

A B C D E F G H I J



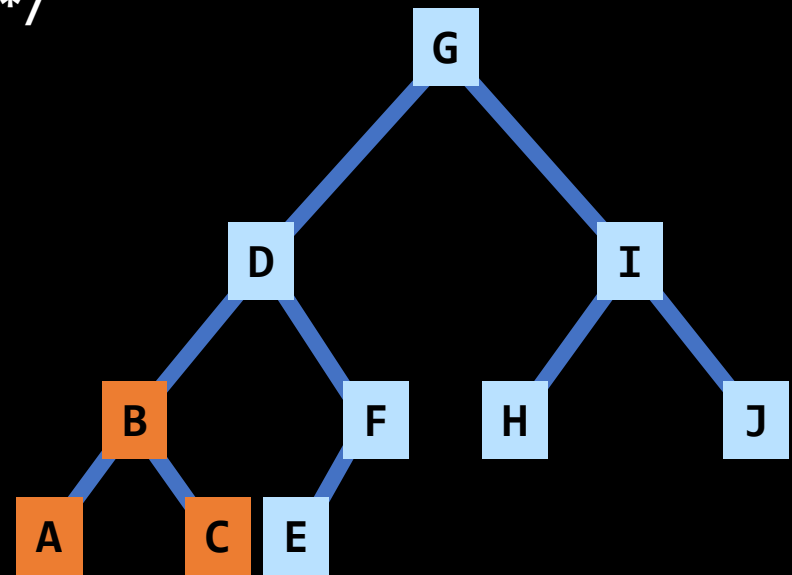
중위순회

```
/* Inorder binary tree traversal. */
```

```
void inorder(BTREE root){  
    if (root != NULL) {  
        inorder(root -> left);  
        printf("%c", root -> d);  
        inorder(root -> right);  
    }  
}
```

Output:

A B C D E F G H I J



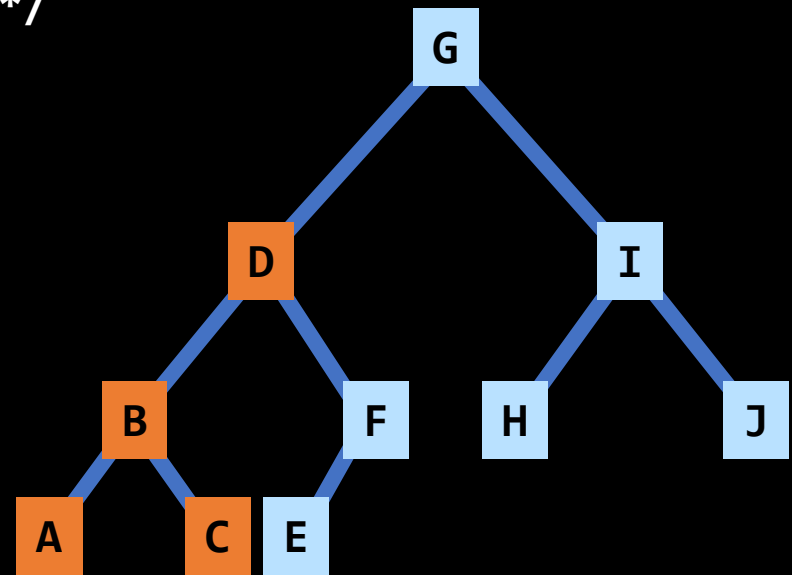
중위순회

```
/* Inorder binary tree traversal. */
```

```
void inorder(BTREE root){  
    if (root != NULL) {  
        inorder(root -> left);  
        printf("%c", root -> d);  
        inorder(root -> right);  
    }  
}
```

Output:

A B C D E F G H I J



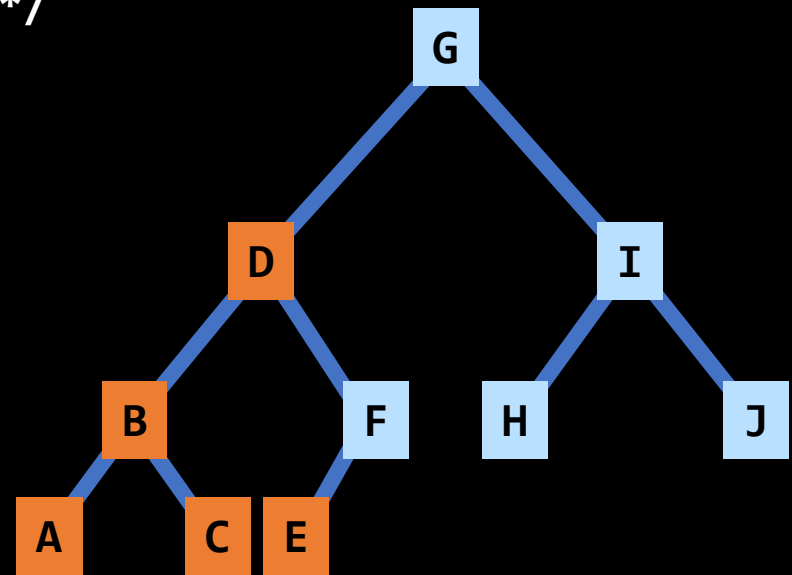
중위순회

```
/* Inorder binary tree traversal. */
```

```
void inorder(BTREE root){  
    if (root != NULL) {  
        inorder(root -> left);  
        printf("%c", root -> d);  
        inorder(root -> right);  
    }  
}
```

Output:

A B C D E F G H I J



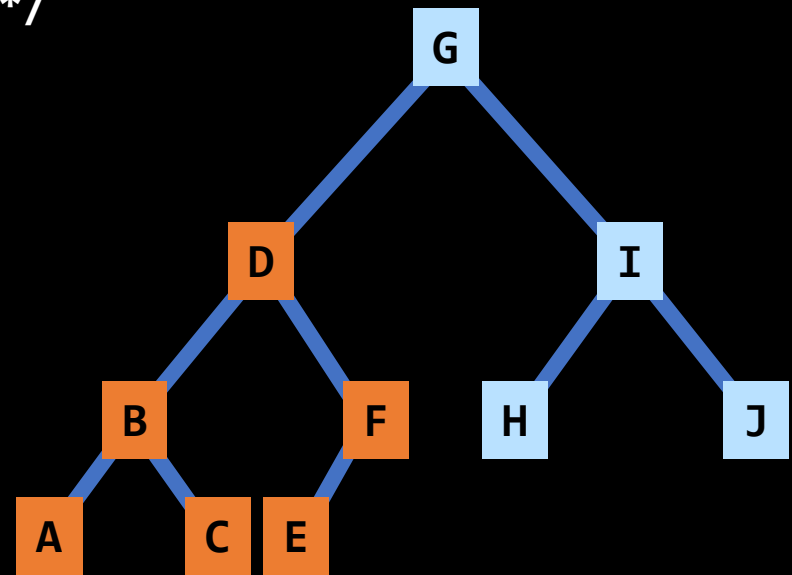
중위순회

```
/* Inorder binary tree traversal. */
```

```
void inorder(BTREE root){  
    if (root != NULL) {  
        inorder(root -> left);  
        printf("%c", root -> d);  
        inorder(root -> right);  
    }  
}
```

Output:

A B C D E F G H I J



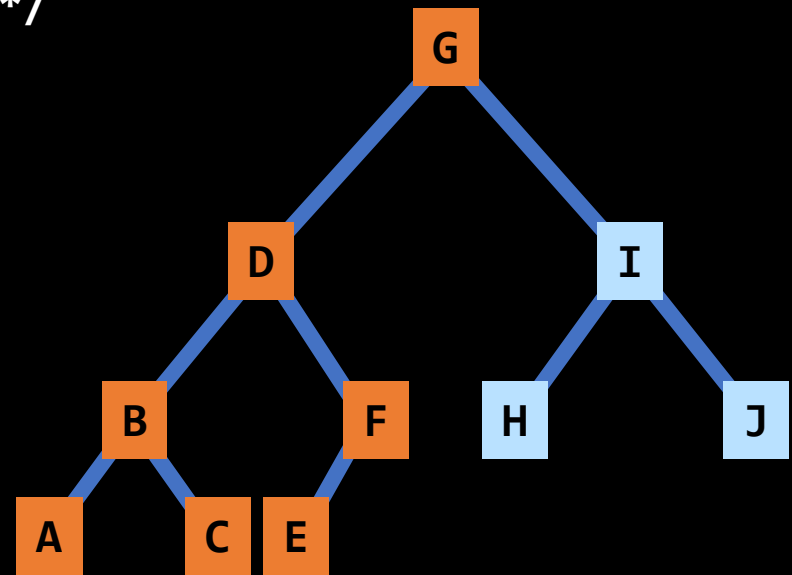
중위순회

```
/* Inorder binary tree traversal. */
```

```
void inorder(BTREE root){  
    if (root != NULL) {  
        inorder(root -> left);  
        printf("%c", root -> d);  
        inorder(root -> right);  
    }  
}
```

Output:

A B C D E F G H I J



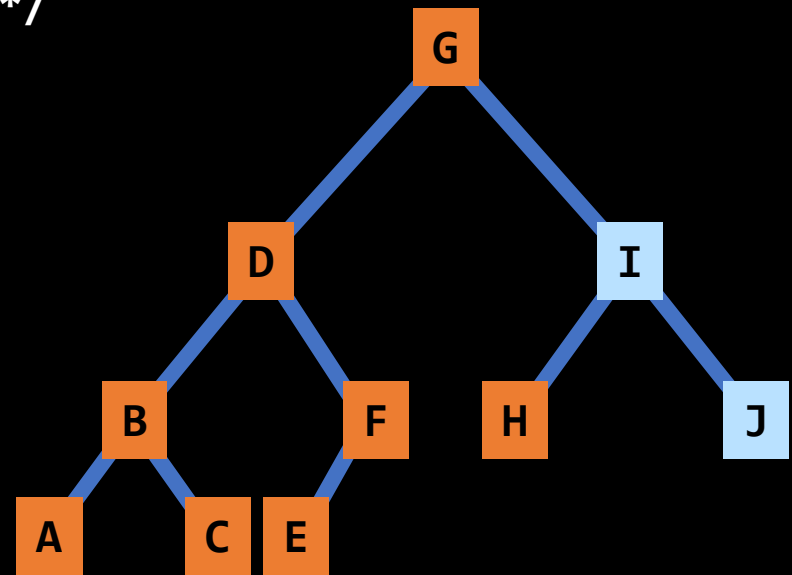
중위순회

```
/* Inorder binary tree traversal. */
```

```
void inorder(BTREE root){  
    if (root != NULL) {  
        inorder(root -> left);  
        printf("%c", root -> d);  
        inorder(root -> right);  
    }  
}
```

Output:

A B C D E F G H I J



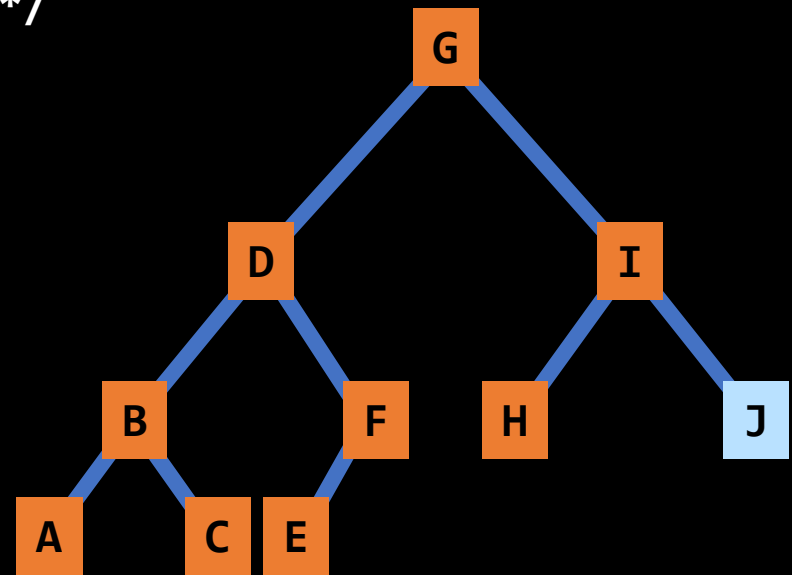
중위순회

```
/* Inorder binary tree traversal. */
```

```
void inorder(BTREE root){  
    if (root != NULL) {  
        inorder(root -> left);  
        printf("%c", root -> d);  
        inorder(root -> right);  
    }  
}
```

Output:

A B C D E F G H I J



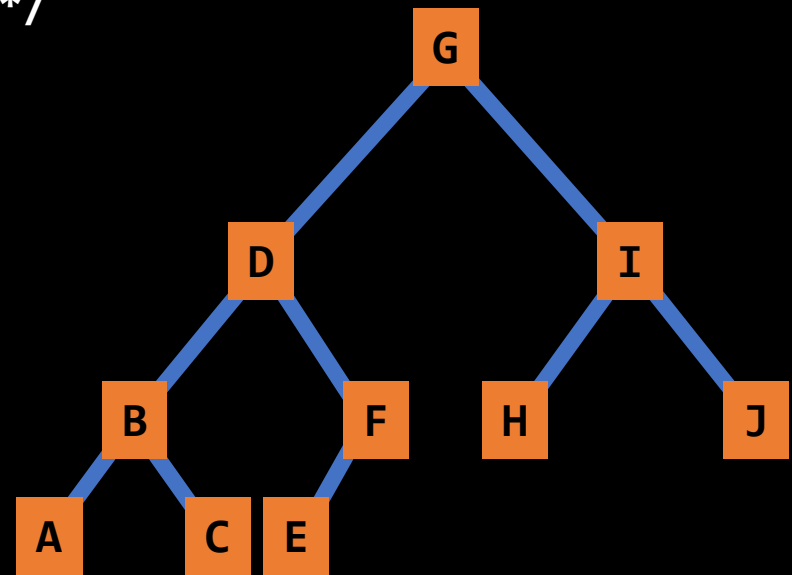
중위순회

```
/* Inorder binary tree traversal. */
```

```
void inorder(BTREE root){  
    if (root != NULL) {  
        inorder(root -> left);  
        printf("%c", root -> d);  
        inorder(root -> right);  
    }  
}
```

Output:

A B C D E F G H I J



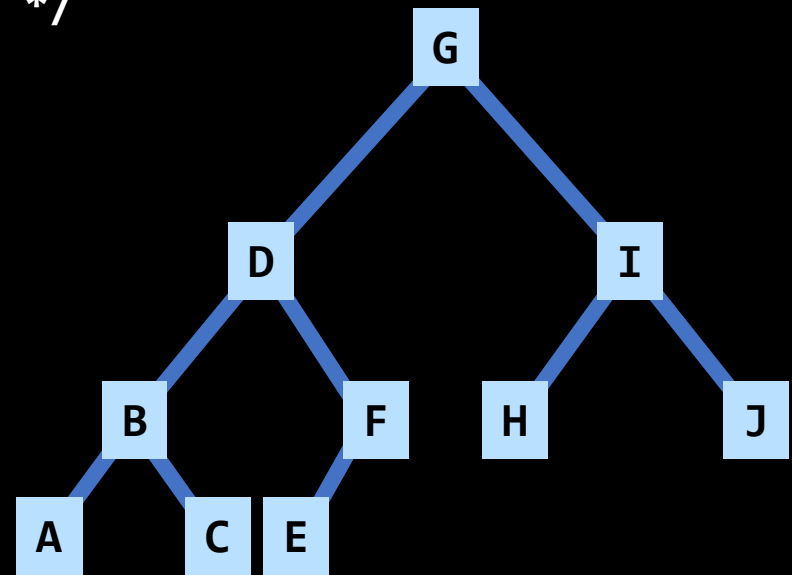
전위순회

```
/* Preorder binary tree traversal. */
```

```
void preorder(BTREE root){  
    if (root != NULL) {  
        printf("%c ", root -> d);  
        preorder(root -> left);  
        preorder(root -> right);  
    }  
}
```

Output:

G D B A C F E I H J



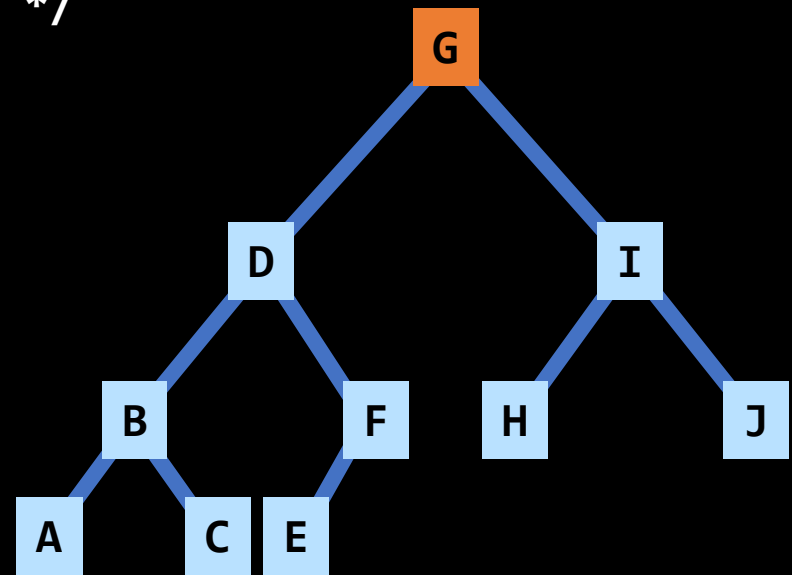
전위순회

```
/* Preorder binary tree traversal. */
```

```
void preorder(BTREE root){  
    if (root != NULL) {  
        printf("%c ", root -> d);  
        preorder(root -> left);  
        preorder(root -> right);  
    }  
}
```

Output:

G D B A C F E I H J



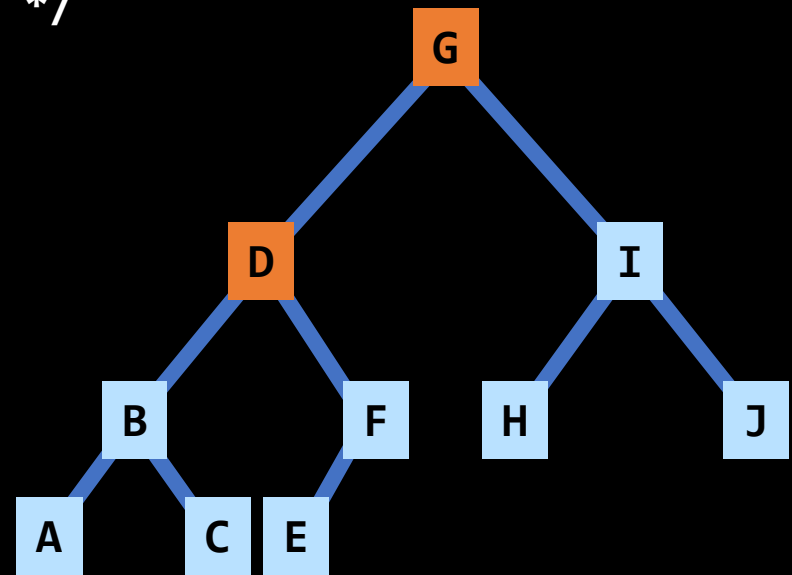
전위순회

```
/* Preorder binary tree traversal. */
```

```
void preorder(BTREE root){  
    if (root != NULL) {  
        printf("%c ", root -> d);  
        preorder(root -> left);  
        preorder(root -> right);  
    }  
}
```

Output:

G D B A C F E I H J



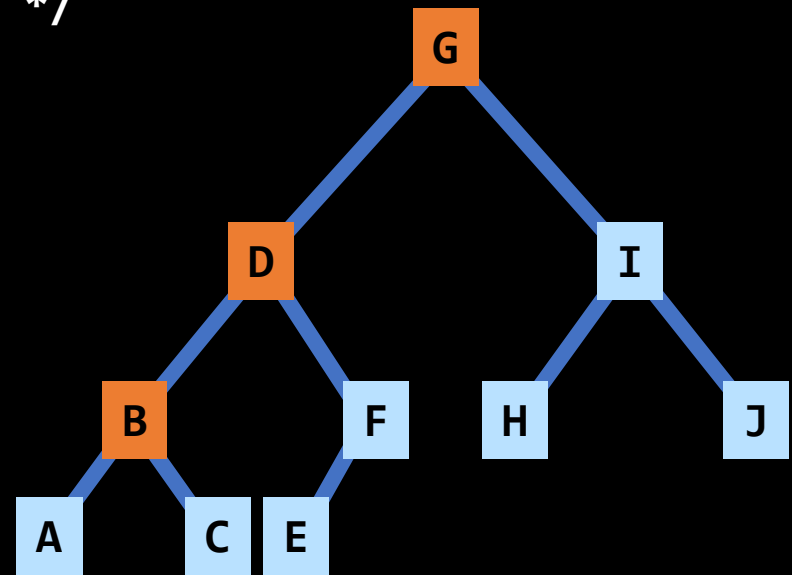
전위순회

```
/* Preorder binary tree traversal. */
```

```
void preorder(BTREE root){  
    if (root != NULL) {  
        printf("%c ", root -> d);  
        preorder(root -> left);  
        preorder(root -> right);  
    }  
}
```

Output:

G D B A C F E I H J



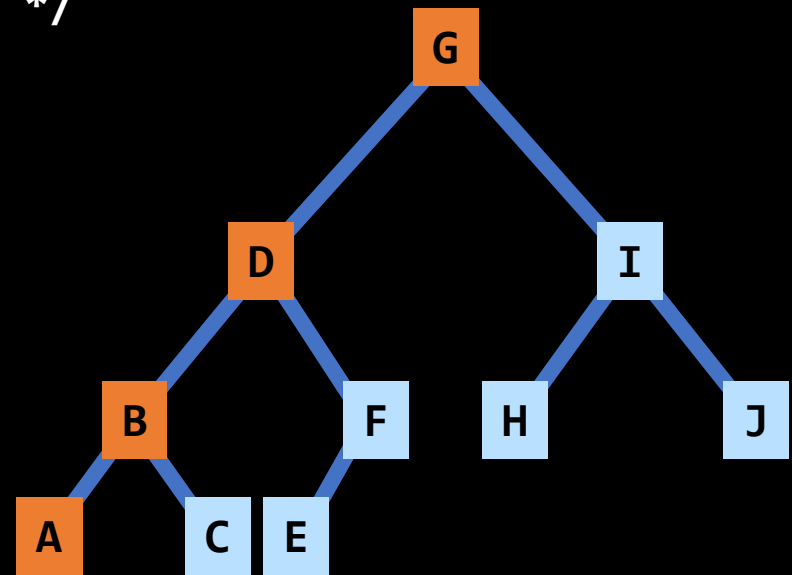
전위순회

```
/* Preorder binary tree traversal. */
```

```
void preorder(BTREE root){  
    if (root != NULL) {  
        printf("%c ", root -> d);  
        preorder(root -> left);  
        preorder(root -> right);  
    }  
}
```

Output:

G D B A C F E I H J



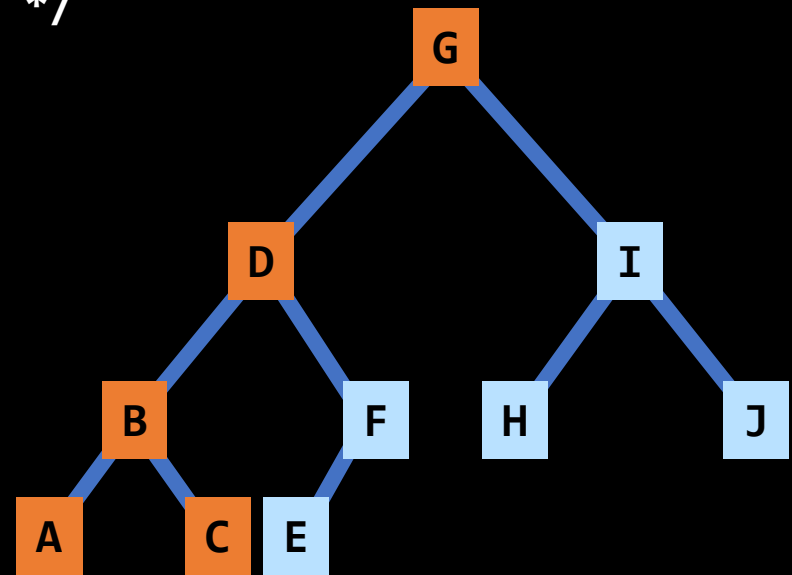
전위순회

```
/* Preorder binary tree traversal. */
```

```
void preorder(BTREE root){  
    if (root != NULL) {  
        printf("%c ", root -> d);  
        preorder(root -> left);  
        preorder(root -> right);  
    }  
}
```

Output:

G D B A C F E I H J



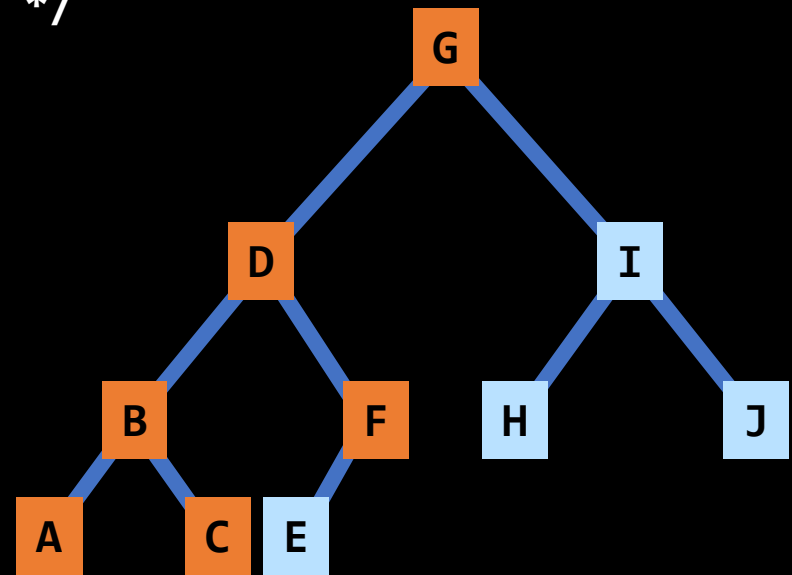
전위순회

```
/* Preorder binary tree traversal. */
```

```
void preorder(BTREE root){  
    if (root != NULL) {  
        printf("%c ", root -> d);  
        preorder(root -> left);  
        preorder(root -> right);  
    }  
}
```

Output:

G D B A C F E I H J



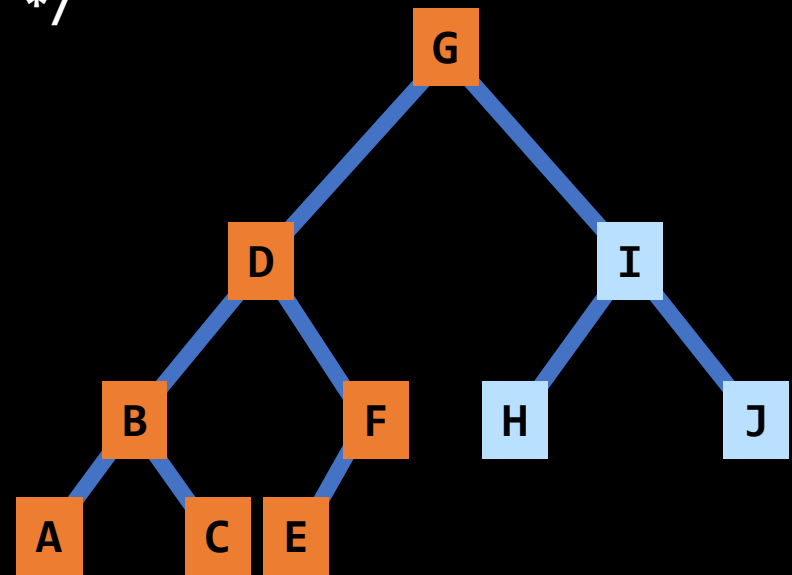
전위순회

```
/* Preorder binary tree traversal. */
```

```
void preorder(BTREE root){  
    if (root != NULL) {  
        printf("%c ", root -> d);  
        preorder(root -> left);  
        preorder(root -> right);  
    }  
}
```

Output:

G D B A C F E I H J



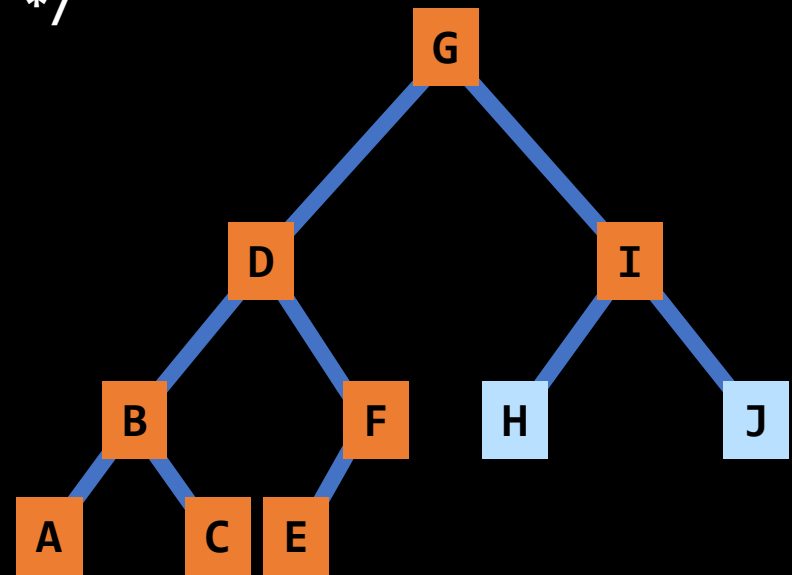
전위순회

```
/* Preorder binary tree traversal. */
```

```
void preorder(BTREE root){  
    if (root != NULL) {  
        printf("%c ", root -> d);  
        preorder(root -> left);  
        preorder(root -> right);  
    }  
}
```

Output:

G D B A C F E I H J



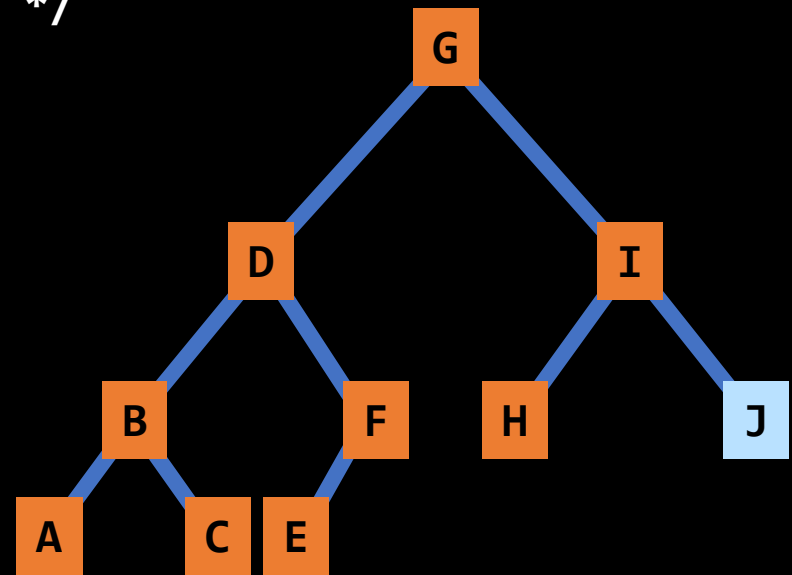
전위순회

```
/* Preorder binary tree traversal. */
```

```
void preorder(BTREE root){  
    if (root != NULL) {  
        printf("%c ", root -> d);  
        preorder(root -> left);  
        preorder(root -> right);  
    }  
}
```

Output:

G D B A C F E I H J



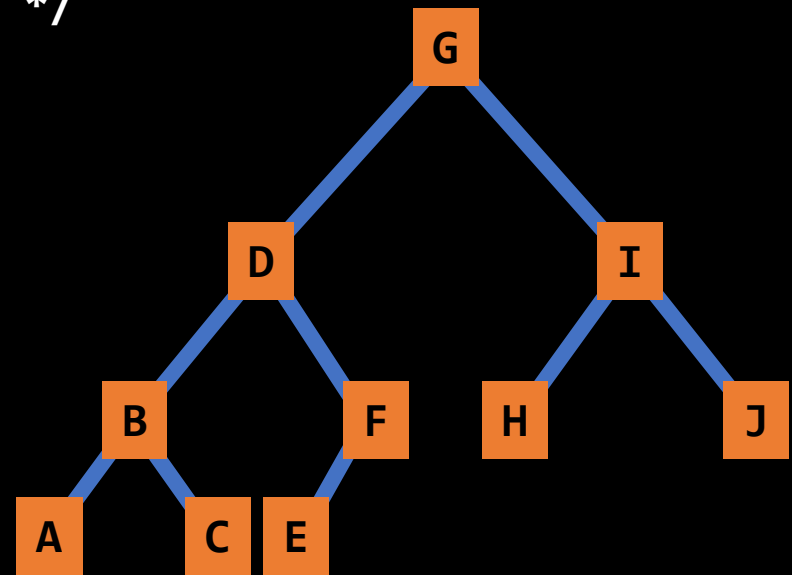
전위순회

```
/* Preorder binary tree traversal. */
```

```
void preorder(BTREE root){  
    if (root != NULL) {  
        printf("%c ", root -> d);  
        preorder(root -> left);  
        preorder(root -> right);  
    }  
}
```

Output:

G D B A C F E I H J



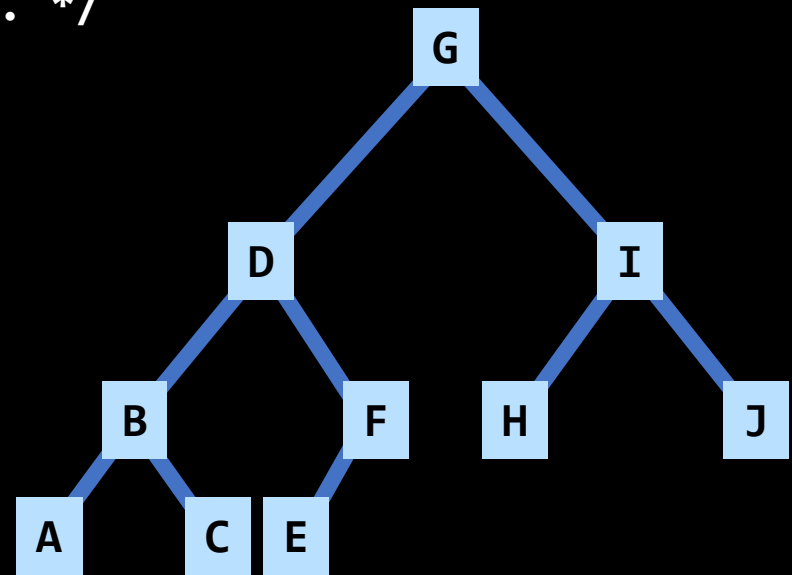
후위순회

```
/* Postorder binary tree traversal. */
```

```
void postorder(BTREE root){  
    if (root != NULL) {  
        postorder(root -> left);  
        postorder(root -> right);  
        printf("%c ", root -> d);  
    }  
}
```

Output:

A C B E F D H J I G



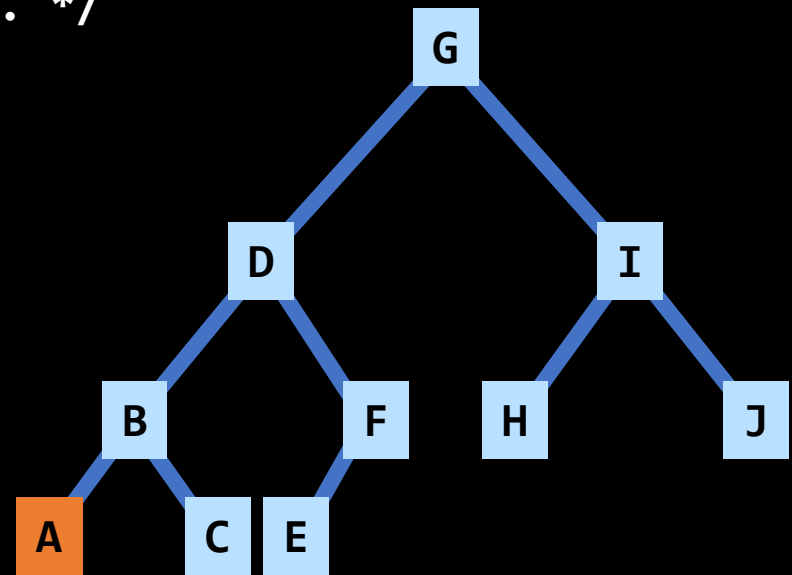
후위순회

```
/* Postorder binary tree traversal. */
```

```
void postorder(BTREE root){  
    if (root != NULL) {  
        postorder(root -> left);  
        postorder(root -> right);  
        printf("%c ", root -> d);  
    }  
}
```

Output:

A C B E F D H J I G



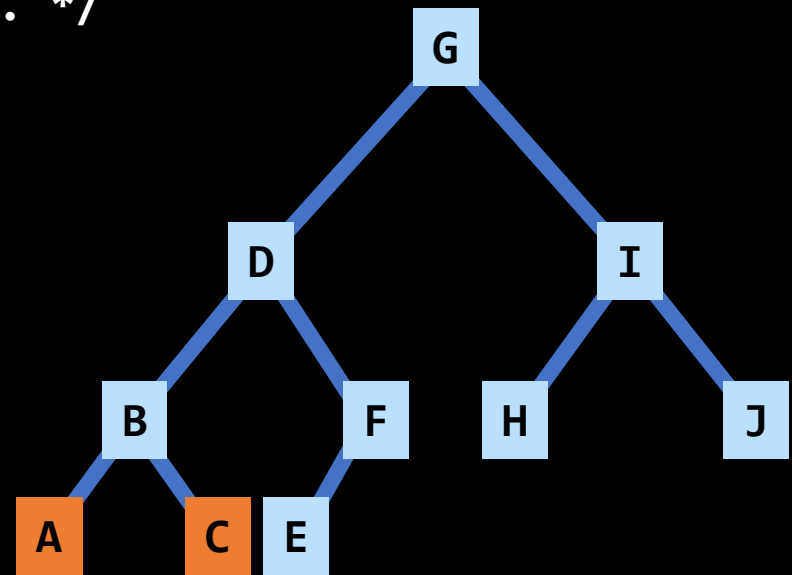
후위순회

```
/* Postorder binary tree traversal. */
```

```
void postorder(BTREE root){  
    if (root != NULL) {  
        postorder(root -> left);  
        postorder(root -> right);  
        printf("%c ", root -> d);  
    }  
}
```

Output:

A C B E F D H J I G



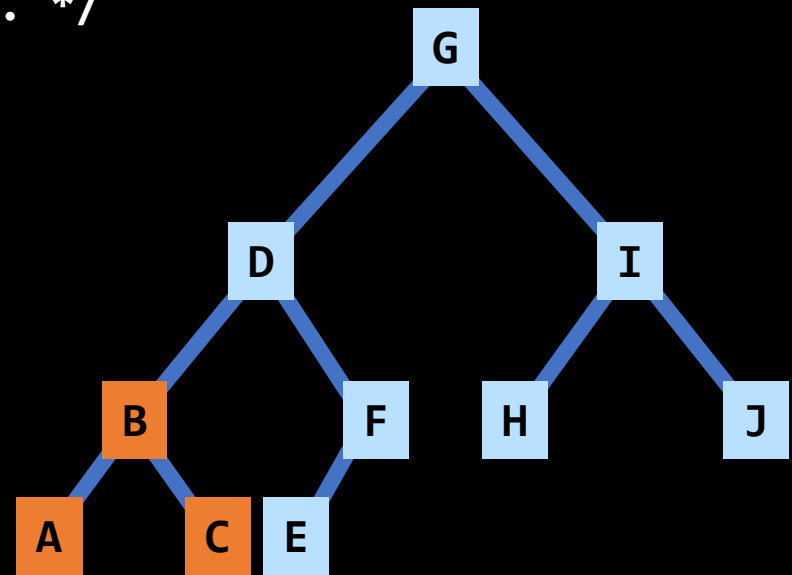
후위순회

```
/* Postorder binary tree traversal. */
```

```
void postorder(BTREE root){  
    if (root != NULL) {  
        postorder(root -> left);  
        postorder(root -> right);  
        printf("%c ", root -> d);  
    }  
}
```

Output:

A C B E F D H J I G



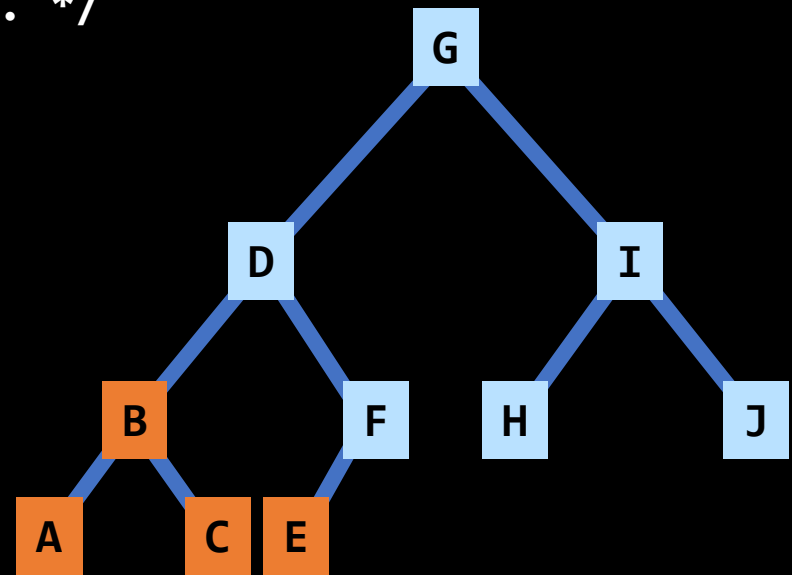
후위순회

```
/* Postorder binary tree traversal. */
```

```
void postorder(BTREE root){  
    if (root != NULL) {  
        postorder(root -> left);  
        postorder(root -> right);  
        printf("%c ", root -> d);  
    }  
}
```

Output:

A C B E F D H J I G



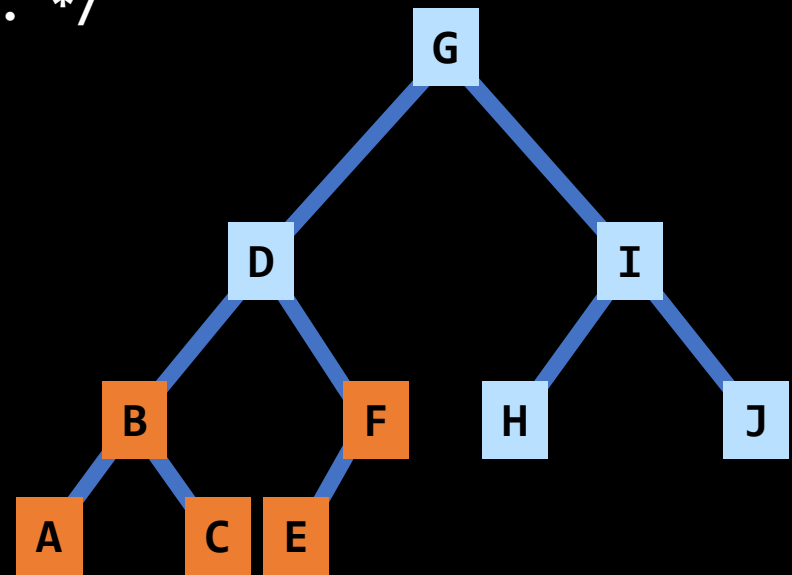
후위순회

```
/* Postorder binary tree traversal. */
```

```
void postorder(BTREE root){  
    if (root != NULL) {  
        postorder(root -> left);  
        postorder(root -> right);  
        printf("%c ", root -> d);  
    }  
}
```

Output:

A C B E F D H J I G



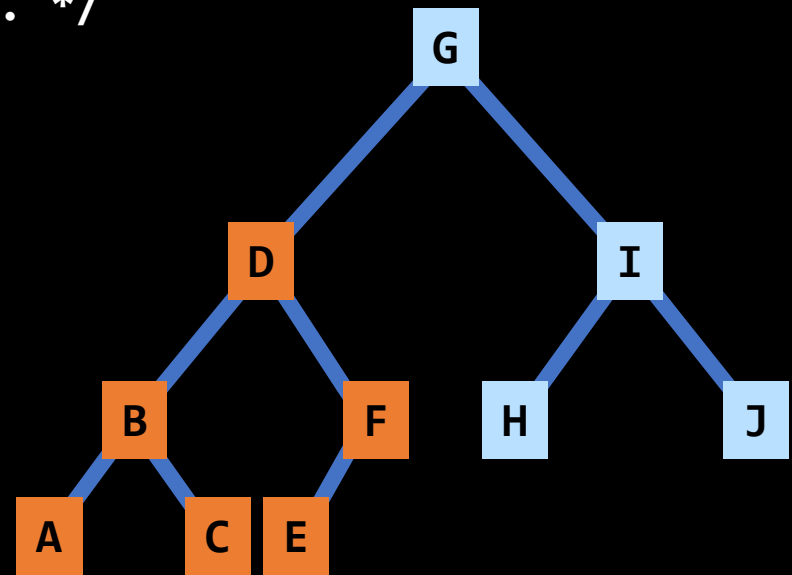
후위순회

```
/* Postorder binary tree traversal. */
```

```
void postorder(BTREE root){  
    if (root != NULL) {  
        postorder(root -> left);  
        postorder(root -> right);  
        printf("%c ", root -> d);  
    }  
}
```

Output:

A C B E F D H J I G



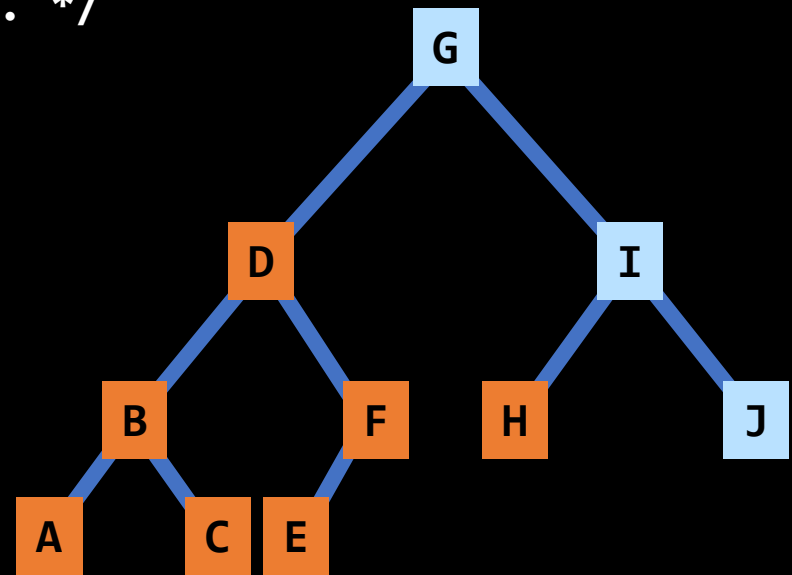
후위순회

```
/* Postorder binary tree traversal. */
```

```
void postorder(BTREE root){  
    if (root != NULL) {  
        postorder(root -> left);  
        postorder(root -> right);  
        printf("%c ", root -> d);  
    }  
}
```

Output:

A C B E F D H J I G



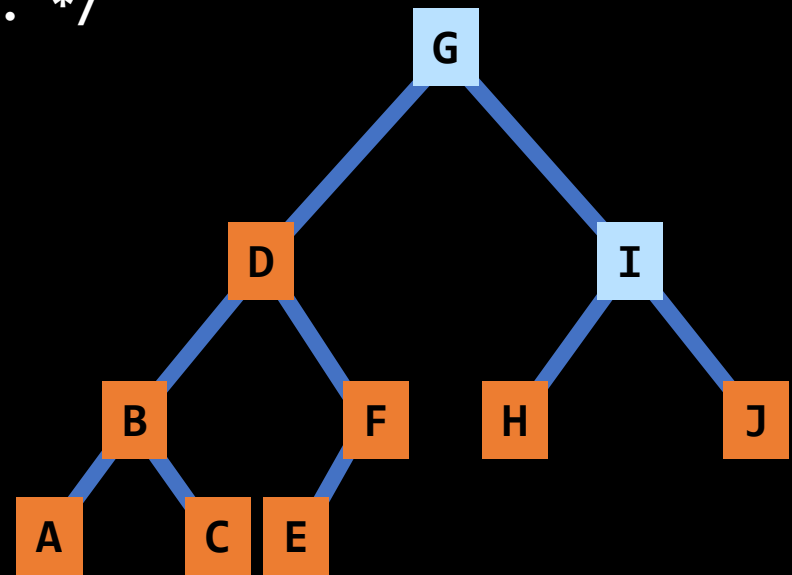
후위순회

```
/* Postorder binary tree traversal. */
```

```
void postorder(BTREE root){  
    if (root != NULL) {  
        postorder(root -> left);  
        postorder(root -> right);  
        printf("%c ", root -> d);  
    }  
}
```

Output:

A C B E F D H J I G



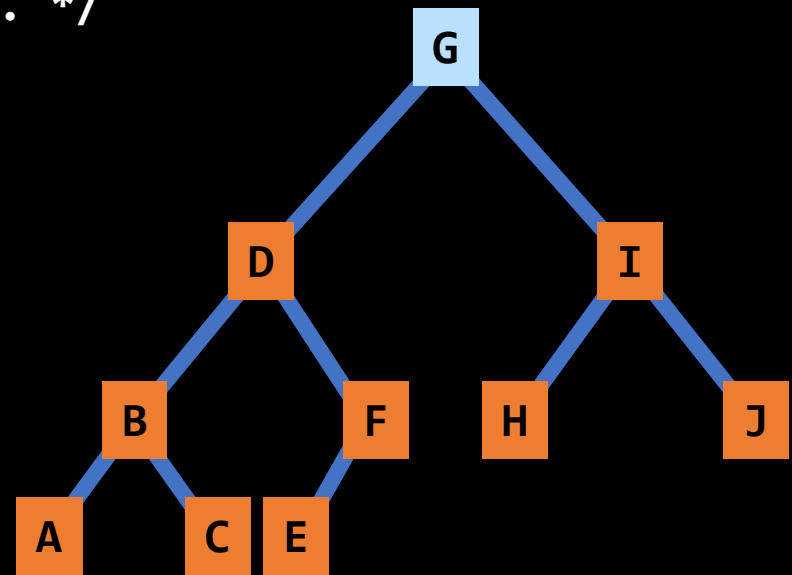
후위순회

```
/* Postorder binary tree traversal. */
```

```
void postorder(BTREE root){  
    if (root != NULL) {  
        postorder(root -> left);  
        postorder(root -> right);  
        printf("%c ", root -> d);  
    }  
}
```

Output:

A C B E F D H J I G



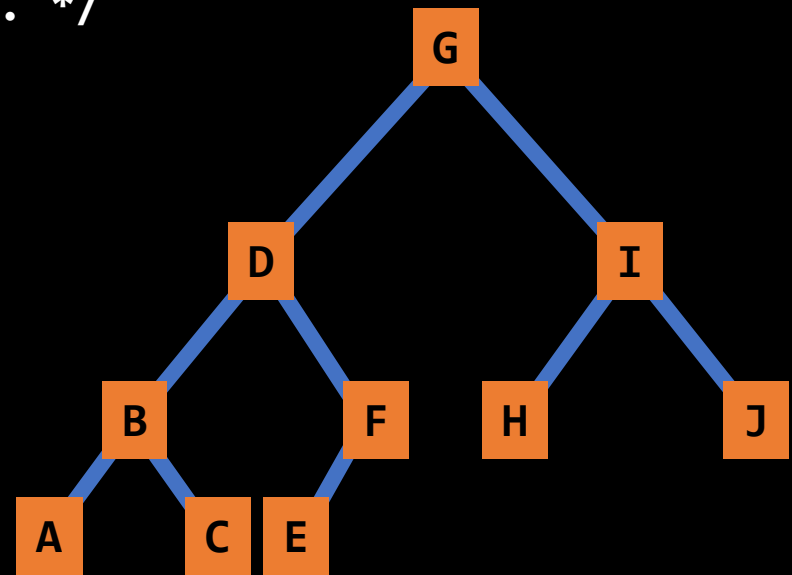
후위순회

```
/* Postorder binary tree traversal. */
```

```
void postorder(BTREE root){  
    if (root != NULL) {  
        postorder(root -> left);  
        postorder(root -> right);  
        printf("%c ", root -> d);  
    }  
}
```

Output:

A C B E F D H J I G



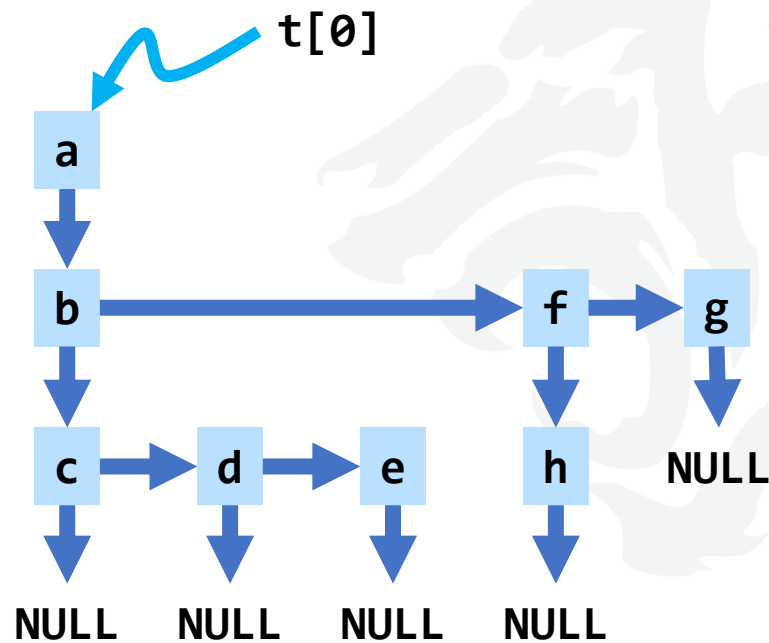
일반적인 연결 리스트

- 어떤 자료구조는 배열과 리스트를 결합하여 사용함



일반 트리

- 형제 노드는 선형 리스트로 자식 노드는 배열의 인덱스로 접근



일반 트리

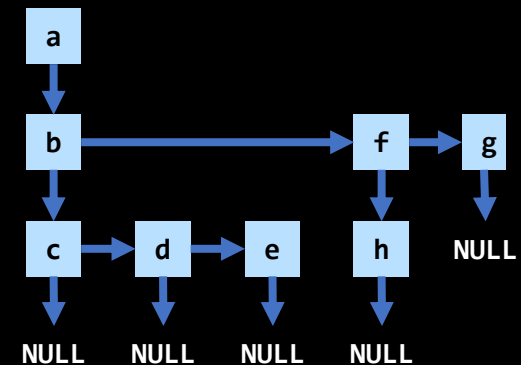
```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
typedef char DATA;
struct node {
    int      child_no;
    DATA    d;
    struct node *sib;
};
typedef struct node NODE;
typedef NODE *GTREE;
#include "fct_proto.h" /* function prototypes */
```

일반 트리 - 노드 생성 함수

```
/* Create a new node. */  
  
GTREE new_gnode(){  
    return (malloc(sizeof(NODE)));  
}  
  
GTREE init_gnode(DATA d1, int num, GTREE sibs){  
    GTREE temp;  
    temp = new_gnode();  
    temp -> d = d1;  
    temp -> child_no = num;  
    temp -> sib = sibs;  
    return temp;  
}
```

일반 트리 - 일반 트리 생성

```
t[0] = init_gnode('a', 1, NULL);  
t[1] = init_gnode('b', 2, NULL);  
t[1] -> sib = init_gnode('f', 6, NULL);  
t[1] -> sib -> sib = init_gnode('g', 7, NULL);  
t[2] = init_gnode('c', 3, NULL);  
t[2] -> sib = init_gnode('d', 4, NULL);  
t[2] -> sib -> sib = init_gnode('e', 5, NULL);  
t[3] = t[4] = t[5] = NULL;  
t[6] = init_gnode('h', 8, NULL);  
t[7] = t[8] = NULL;
```



일반 트리 - 전위 순회 함수

```
/* Preorder traversal of general trees. */  
  
void preorder_g(GTREE *t, int ind)  
{  
    GTREE tmp; /* tmp traverses the sibling list */  
    tmp = t[ind]; /* t[ind] is the root node */  
    while (tmp != NULL) {  
        printf("%c %d\n", tmp -> d, tmp -> child_no);  
        preorder_g(t, tmp -> child_no);  
        tmp = tmp -> sib;  
    }  
}
```


Homework

- Exercises #3, 8, 14, 19, 28, 32

