

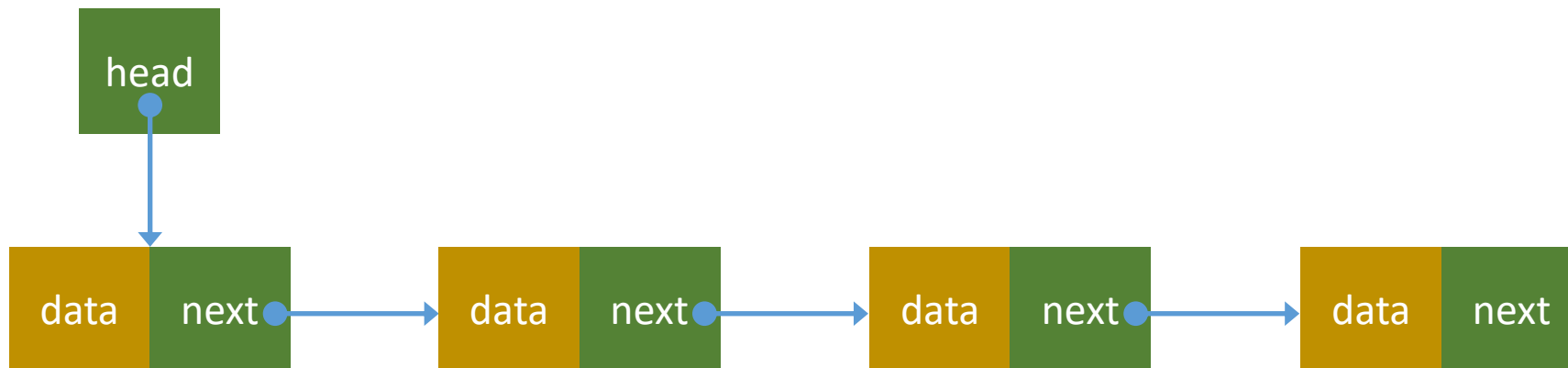
고급C프로그래밍

12 연결 리스트(Linked List)

연결 리스트 개요

- **데이터 요소들을 순차적으로 저장하는 자료구조**

- ✓ 노드들의 집합으로 이루어지며 구성 요소는 데이터 + 다음 노드를 가리키는 포인터
- ✓ 데이터 : 단일 데이터 형식의 데이터 또는 데이터들의 집합 (배열, 구조체 등 자료구조)
- ✓ 단일 연결 리스트(또는 단방향 연결 리스트)
 - ➔ 단방향으로 노드들이 연결되어 있으며, 각 노드는 다음 노드를 가리킴



연결 리스트 vs. 배열

1. 유연한 메모리 할당

- ✓ 고정된 크기의 메모리를 사용하지 않고 데이터에 따라 동적으로 메모리 할당/해제

2. 데이터 추가 및 삭제의 용이성

- ✓ 리스트 중간에 데이터를 추가하거나 삭제하는 것이 편리함

3. 데이터 저장에 있어 가변적이고 확장적임

- ✓ 저장하는 데이터의 양에 따라 리스트의 크기를 줄이거나 늘리기가 용이함

4. 메모리 사용이 효율적임

- ✓ 배열은 물리적으로 연속된 메모리 공간을 사용하나, 연결 리스트는 독립적인 메모리 블록을 연결하여 사용

5. 데이터 구조의 확장성

- ✓ 데이터 탐색 효율성 등을 고려하여 다양한 구조로 확장이 편리함

단일 연결 리스트 만들기 (1/13)

<1단계> 노드 정의

- ✓ 노드 구성요소 : 데이터 + 다음 노드를 가리키는 포인터

```
struct Node {  
    int data;                //배열, 구조체 등 다양한 자료구조 사용  
                             //복수의 자료구조 사용 가능  
  
    struct Node* next;       //다음 노드를 가리키는 포인터  
};
```

단일 연결 리스트 만들기 (2/13)

<2단계> 새로운 노드 만들기

//새로운 노드에 저장할 데이터를 매개변수로 전달
//만들어진 노드의 주소를 반환

```
struct Node* create_node(int data) {
```

//동적 메모리 할당으로 노드를 생성

```
struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
```

//매개변수로 전달받은 데이터를 노드에 저장

```
new_node->data = data;
```

//다음 노드를 가리킬 포인터는 NULL로 초기화

```
new_node->next = NULL;
```

```
return new_node;
```

```
}
```

단일 연결 리스트 만들기 (3/13)

<3단계> 연결 리스트에 노드를 추가하기

//head는 연결 리스트의 시작 노드를 가리킴

```
Struct Node* insert_node(struct Node* head, int data) {
```

//연결 리스트에 추가할 노드를 생성

```
    struct Node* new_node = create_node(data);
```

//기존 연결 리스트에 노드가 하나도 없을 경우

```
    if (head == NULL) { head = new_node; }
```

//마지막 노드를 찾아서 새로 만든 노드를 추가

```
    else {
```

1

```
        struct Node* temp = head;
```

```
        while (temp->next != NULL) { temp = temp->next; }
```

```
        temp->next = new_node;
```

```
    }
```

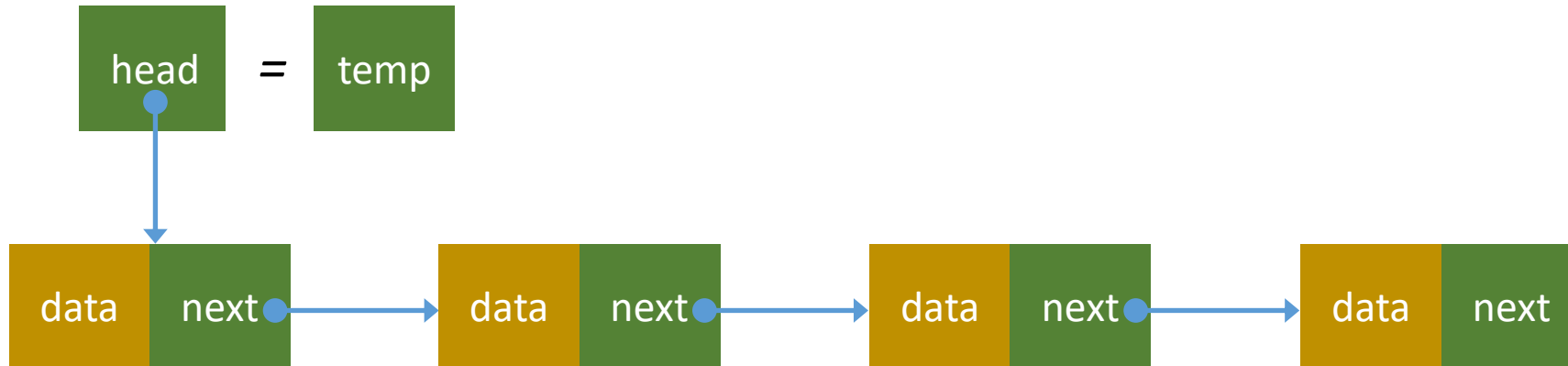
2

```
    return head; }
```

단일 연결 리스트 만들기 (4/13)

<3단계 - 1> 연결 리스트 탐색 하기

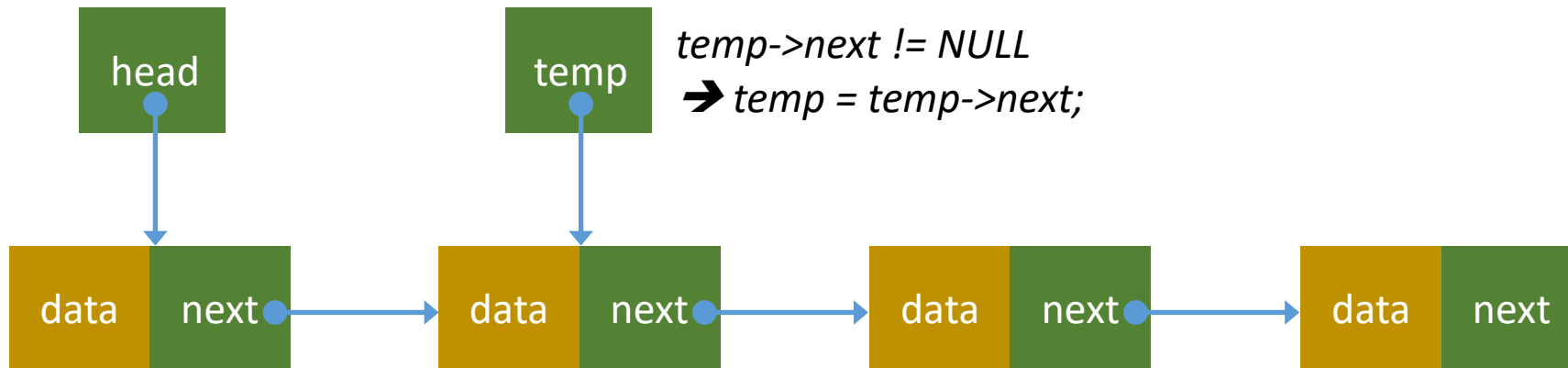
- ✓ temp가 탐색자 역할을 수행
- ✓ head를 이용해서 연결 리스트의 첫번째 노드를 가리킴
- ✓ 반복문이 반복될 때 마다 조건에 맞는 노드를 찾아서 다음 노드로 이동



단일 연결 리스트 만들기 (5/13)

<3단계 - 1> 연결 리스트 탐색 하기

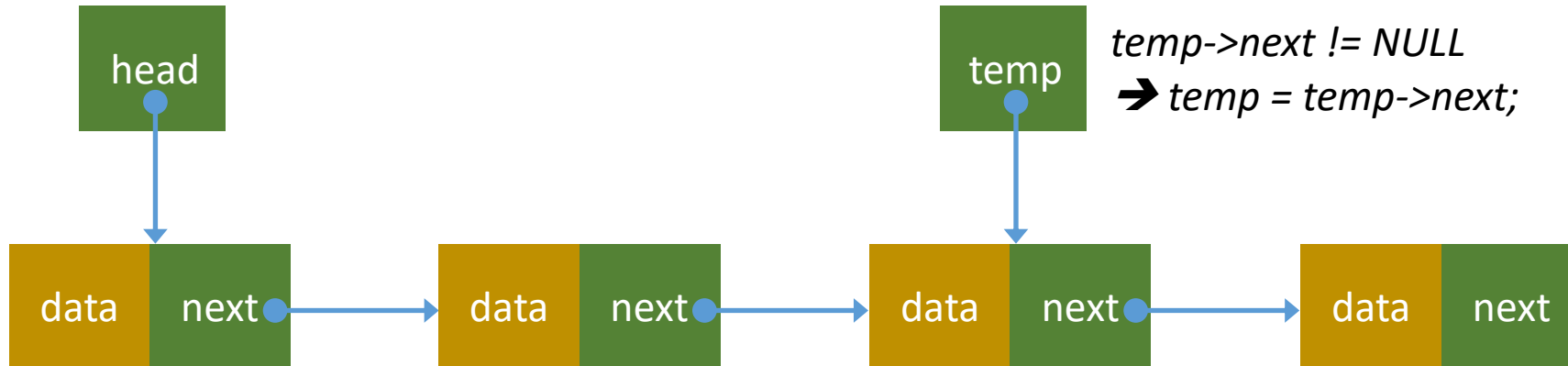
- ✓ temp가 탐색자 역할을 수행
- ✓ head를 이용해서 연결 리스트의 첫번째 노드를 가리킴
- ✓ 반복문이 반복될 때 마다 조건에 맞는 노드를 찾아서 다음 노드로 이동



단일 연결 리스트 만들기 (6/13)

<3단계 - 1> 연결 리스트 탐색 하기

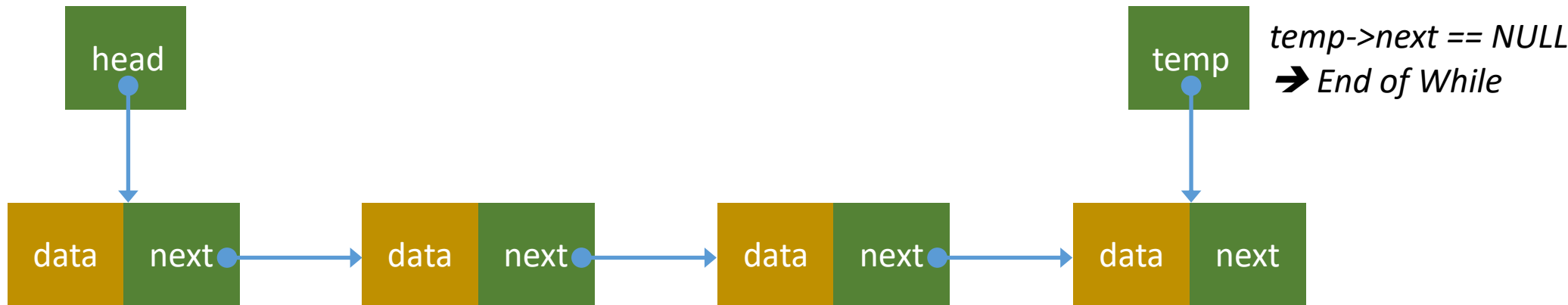
- ✓ temp가 탐색자 역할을 수행
- ✓ head를 이용해서 연결 리스트의 첫번째 노드를 가리킴
- ✓ 반복문이 반복될 때 마다 조건에 맞는 노드를 찾아서 다음 노드로 이동



단일 연결 리스트 만들기 (7/13)

<3단계 - 1> 연결 리스트 탐색 하기

- ✓ temp가 탐색자 역할을 수행
- ✓ head를 이용해서 연결 리스트의 첫번째 노드를 가리킴
- ✓ 반복문이 반복될 때 마다 조건에 맞는 노드를 찾아서 다음 노드로 이동



단일 연결 리스트 만들기 (8/13)

<3단계 - 2> head를 반환하는 이유

✓ `struct Node* head` 매개변수 전달방식 : 값으로 전달

➔ 포인터가 매개 변수지만, 함수 내에서 주소 자체를 변경하기 때문에 함수가 종료되면 변경값이 사라짐. 즉 주소 자체가 값으로 사용!

단일 연결 리스트 만들기 (9/13)

<3단계 - 2> head를 반환하는 이유

✓ 주소로 전달 : 포인터의 포인터를 사용

```
void insert_node(struct Node** head, int data) {  
    struct Node* new_node = create_node(data);  
  
    if (*head == NULL) { *head = new_node; }  
  
    else {  
        struct Node* temp = *head;  
        while (temp->next != NULL) { temp = temp->next; }  
        temp->next = new_node;  
    }  
    return head; }
```

//호출 시

```
struct Node *head = NULL;  
insert_node(&head, 1);
```

단일 연결 리스트 만들기 (10/13)

<4단계> 연결 리스트에 노드를 삭제하기

//head를 반환, target은 삭제할 노드의 데이터

```
struct Node* delete_node(struct Node* head, int target) {
```

//노드 삭제를 위해서는 2개의 노드 포인터가 필요

```
    struct Node* current = head;
```

```
    struct Node* prev = NULL;
```

// 첫 번째 노드가 삭제 대상인 경우 head를 다음 노드로 변경하고 노드를 삭제

```
    if (current != NULL && current->data == target) {
```

```
        head = current->next;
```

```
        free(current);
```

```
        return head; }
```

단일 연결 리스트 만들기 (11/13)

<4단계> 연결 리스트에 노드를 삭제하기 (계속)

// 삭제 대상 노드를 찾을 때 까지 노드를 건너가며 계속 탐색

```
while (current != NULL && current->data != target) {
```

//prev는 현재 노드를, current는 다음 노드를 가리킴

```
    prev = current;
```

```
    current = current->next; }
```

//삭제 대상 노드가 없을 경우

```
if (current == NULL) {
```

```
    printf("삭제할 노드가 없습니다.\n");
```

```
    return head; }
```

// 삭제 대상 노드를 연결에서 분리하고 메모리 해제

1

```
prev->next = current->next;
```

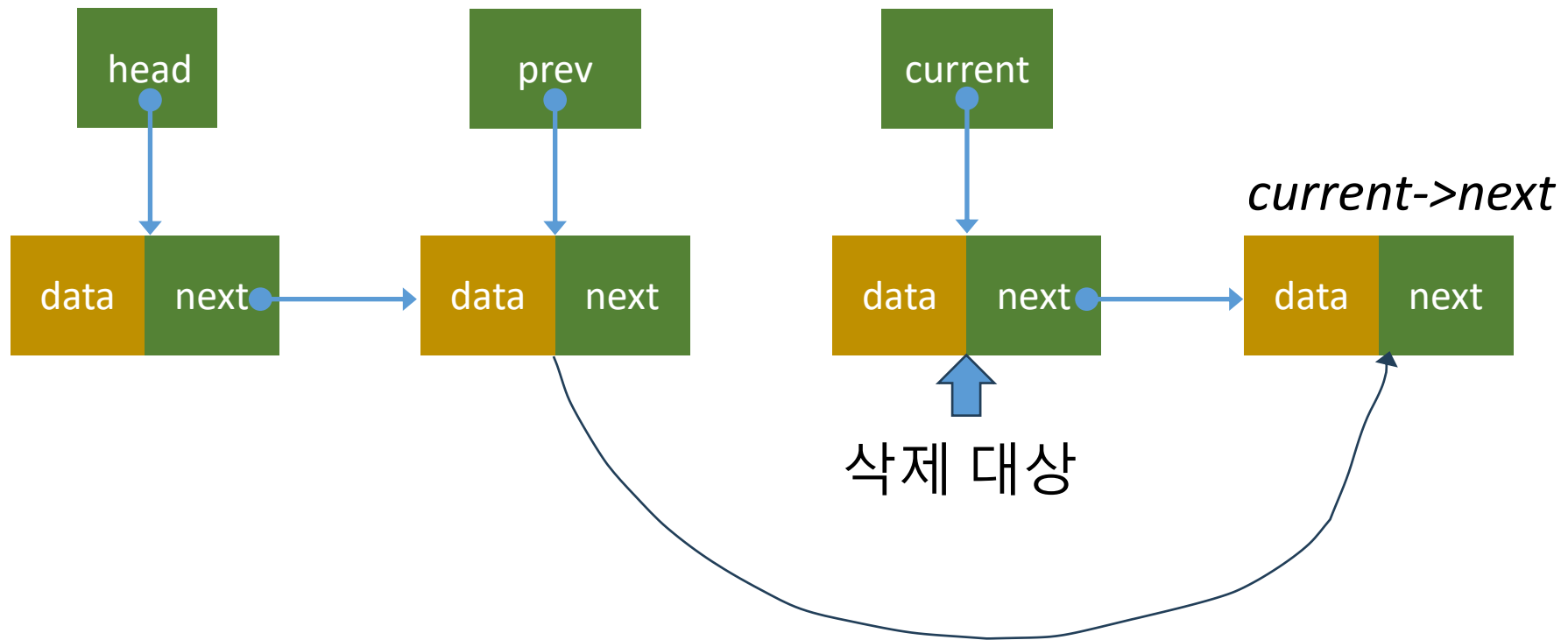
```
free(current);
```

```
return head; }
```

단일 연결 리스트 만들기 (12/13)

<4단계 - 1> 노드 삭제하기

- ✓ 이전 노드(prev)가 다음 노드(current->next)를 가리키도록 하고, 현재 노드는 삭제



단일 연결 리스트 만들기 (13/13)

<> 연결 리스트의 데이터 처리 : 데이터 출력 (예)

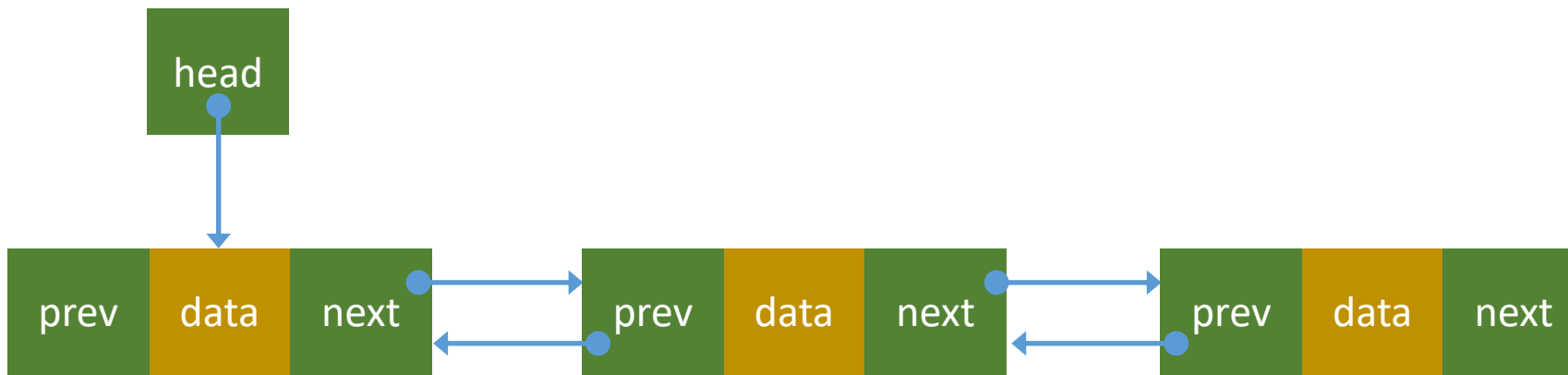
//노드를 탐색하며, 각 노드의 데이터값을 출력

```
void print_list(struct Node* head) {  
    struct Node* current = head;  
  
    while (current != NULL) {  
        printf("%d -> ", current->data);  
        current = current->next; }  
  
    printf("NULL\n");  
}
```


이중 연결 리스트 (양방향 연결 리스트) (1/7)

- **앞, 뒤 노드와 연결을 유지**

- ✓ 기존 단일 연결 리스트에서 앞 노드와의 연결을 추가
- ✓ 노드 탐색 및 리스트 정렬 등에서 단일 연결 리스트 대비 장점을 보유



이중 연결 리스트 (양방향 연결 리스트) (2/7)

<1단계> 노드 정의

- ✓ 노드 구성요소 : 데이터 + 이전, 다음 노드를 가리키는 포인터

```
struct Node {  
    int data;  
    struct Node* prev;    //이전 노드를 가리키는 포인터  
    struct Node* next;    //다음 노드를 가리키는 포인터  
};
```

이중 연결 리스트 (양방향 연결 리스트) (3/7)

<2단계> 새로운 노드 만들기

//새로운 노드에 저장할 데이터를 매개변수로 전달

//만들어진 노드의 주소를 반환

```
struct Node* create_node(int data) {
```

//동적 메모리 할당으로 노드를 생성

```
struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
```

//매개변수로 전달받은 데이터를 노드에 저장

```
new_node->data = data;
```

//이전, 다음 노드를 가리킬 포인터는 NULL로 초기화

```
new_node->prev = NULL;
```

```
new_node->next = NULL;
```

```
return new_node;
```

```
}
```

이중 연결 리스트 (양방향 연결 리스트) (4/7)

<3단계> 연결 리스트에 노드를 추가하기

//head는 연결 리스트의 시작 노드를 가리킴

```
Struct Node* insert_node(struct Node* head, int data) {
```

//연결 리스트에 추가할 노드를 생성

```
    struct Node* new_node = create_node(data);
```

//기존 연결 리스트에 노드가 하나도 없을 경우

```
    if (head == NULL) { head = new_node; }
```

//마지막 노드를 찾아서 새로 만든 노드를 추가

```
    else {
```

```
        struct Node* temp = head;
```

```
        while (temp->next != NULL) { temp = temp->next; }
```

```
        temp->next = new_node;
```

```
        new_node->prev = temp;
```

```
    } return head; }
```

이중 연결 리스트 (양방향 연결 리스트) (5/7)

<4단계> 연결 리스트에 노드를 삭제하기

```
struct Node* delete_node(struct Node* head, int target) {
```

//노드 삭제를 위해서는 1개의 노드 포인터가 필요

```
    struct Node* current = head;
```

```
    struct Node* prev = NULL;
```

// 첫 번째 노드가 삭제 대상인 경우 head를 다음 노드로 변경하고 노드를 삭제

```
    if (current != NULL && current->data == target) {
```

```
        head = current->next;
```

```
        head->prev = NULL;
```

```
        free(current);
```

```
        return head; }
```

이중 연결 리스트 (양방향 연결 리스트) (6/7)

<4단계> 연결 리스트에 노드를 삭제하기 (계속)

// 삭제 대상 노드를 찾을 때 까지 노드를 건너가며 계속 탐색

```
while (current != NULL && current->data != target) {
```

```
prev = current;
```

```
current = current->next; }
```

//삭제 대상 노드가 없을 경우

```
if (current == NULL) {
```

```
printf("삭제할 노드가 없습니다.\n");
```

```
return head; }
```

// 삭제 대상 노드가 마지막 노드일 경우

```
if (current->next == NULL) {
```

```
current->prev->next = NULL;
```

```
free(current);
```

```
return head; }
```

이중 연결 리스트 (양방향 연결 리스트) (7/7)

<4단계> 연결 리스트에 노드를 삭제하기 (계속)

// 삭제 대상 노드가 중간에 위치할 경우

```
current->prev->next = current->next;  
current->next->prev = current->prev;  
free(current);  
return head; }
```

실습

[실습 1] 간단한 연결 리스트 생성 프로그램 작성

- **다음의 5가지 정수값을 저장하는 연결 리스트 생성**

- ✓ 1, 3, 5, 7, 9 등 5개의 정수값을 저장하는 5개의 노드로 구성된 연결 리스트 생성
- ✓ 사용자로부터 정수값을 입력받고, 0을 입력할 때까지 입력받은 정수값을 저장하는 노드를 추가함
- ✓ 사용자가 0을 입력하면, 저장하고 있는 정수값들을 모두 출력

[실습 2] 간단한 연결 리스트 생성 프로그램 확장

- **[실습 1]의 프로그램에서 노드 삭제 기능을 확장**
 - ✓ 사용자가 음수를 입력하면 노드를 삭제하는 기능을 추가 구현

[실습 3] 도서관 도서 대여 프로그램을 연결 리스트로 확장

- 10주차 [실습 3]의 프로그램을 단일 연결 리스트로 확장
 - ✓ 보유 도서 리스트, 대여 도서 리스트를 연결 리스트로 구현
 - ✓ 보유 도서의 보유량이 0이 되면 보유 도서 리스트에서 삭제

[실습 4] 도서관 도서 대여 프로그램을 이중 연결 리스트로 변경

- **[실습 3]의 프로그램을 이중 연결 리스트로 변경**

Q & A