

제3장

연결 리스트

연결 리스트

개념과 기본 연산

리스트 (List)

❶ 리스트

- 기본적인 연산: 삽입(insert), 삭제(remove), 검색(search) 등
- 리스트를 구현하는 대표적인 두 가지 방법: **배열, 연결리스트**

❷ 배열의 단점

- 크기가 고정 - reallocation이 필요
- 리스트의 중간에 원소를 삽입하거나 삭제할 경우 다수의 데이터를 옮겨야

❸ 연결리스트

- 다른 데이터의 이동없이 중간에 삽입이나 삭제가 가능하며,
- 길이의 제한이 없음
- 하지만 **랜덤 액세스가 불가능**

연결리스트

100	Monday
101	Tuesday
102	Friday
103	Saturday
104	
105	
106	
107	
108	

Array

100		
101		
102	Tuesday	107
103	Saturday	null
104		
105	Monday	102
106		
107	Friday	103
108		

Linked List

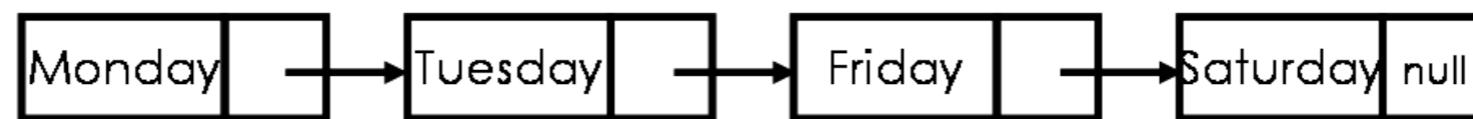
삽입과 삭제

100		
101		
102	Tuesday	108
103	Saturday	null
104		
105	Monday	102
106		
107	Friday	103
108	Thursday	107

Adding Thursday

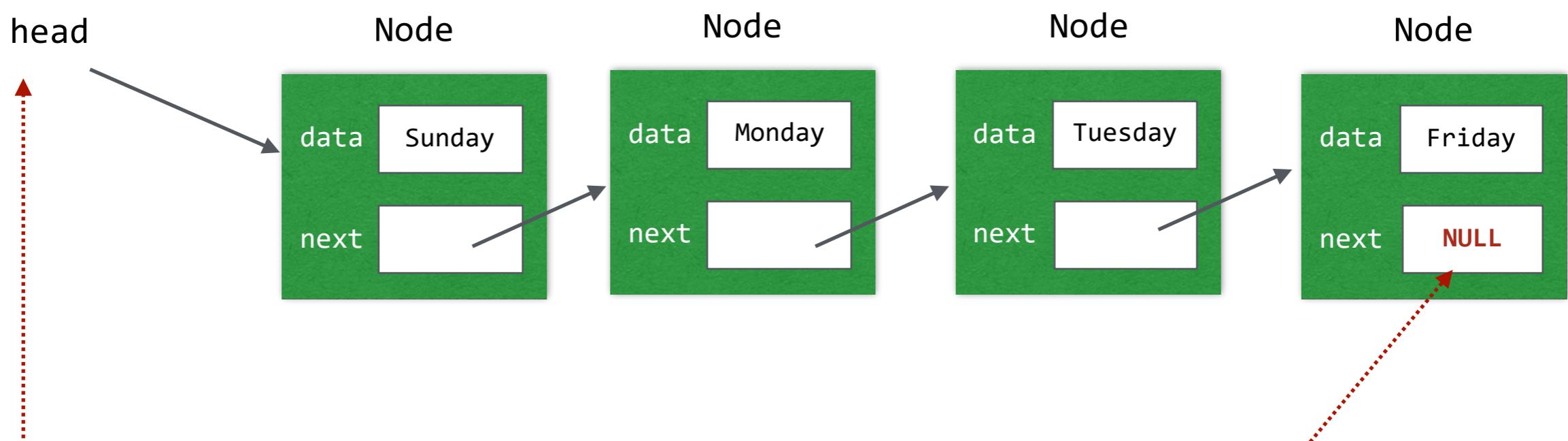
100		
101		
102		
103	Saturday	null
104		
105	Monday	108
106		
107	Friday	103
108	Thursday	107

Removing Tuesday



노드

- 각각의 노드는 필요한 데이터 필드와 하나 혹은 그 이상의 링크 필드로 구성
- 링크 필드는 다음 노드의 주소를 저장
- 첫 번째 노드의 주소는 따로 저장해야



연결리스트의 첫 번째 노드의 주소는 따로 저장해야하며 절대 잊어버려서는 안된다.

마지막 노드의 next 필드에는 NULL을 저장하여 연결리스트의 끝임을 표시한다.

Node

연결 리스트에서 하나의 노드를 표현하기 위한 구조체이다.
각 노드에 저장될 데이터는 하나의 문자열이라고 가정하자.

```
struct node {  
    char *data;
```

```
    struct node *next;
```

다음 노드의 주소를 저장할 필드이다.

← 이렇게 자신과 동일한 구조체에 대한 포인터를 멤버로 가진다는 의미에서
“자기참조형 구조체”라고 부르기도 한다.

```
}
```

```
typedef struct node Node;
```

```
Node *head = NULL;
```

← 연결리스트의 첫 번째 노드의 주소를 저장할 포인터이다.

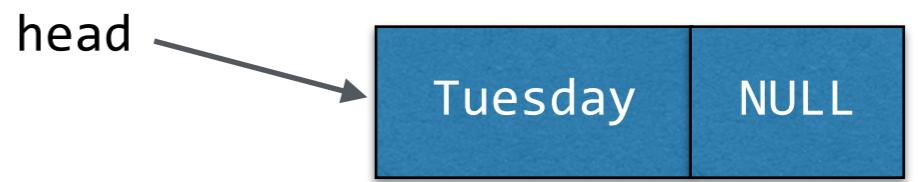
예제 프로그램

```
int main()
{
    Node *head = (Node *)malloc(sizeof(Node));
    head->data = "Tuesday";
    head->next = NULL;

    Node *q = (Node *)malloc(sizeof(Node));
    q->data = "Thursday";
    q->next = NULL;
    head->next = q;

    q = (Node *)malloc(sizeof(Node));
    q->data = "Monday";
    q->next = head;
    head = q;

    Node *p = head;
    while(p!=NULL) {
        printf("%s\n", p->data);
        p = p->next;
    }
}
```



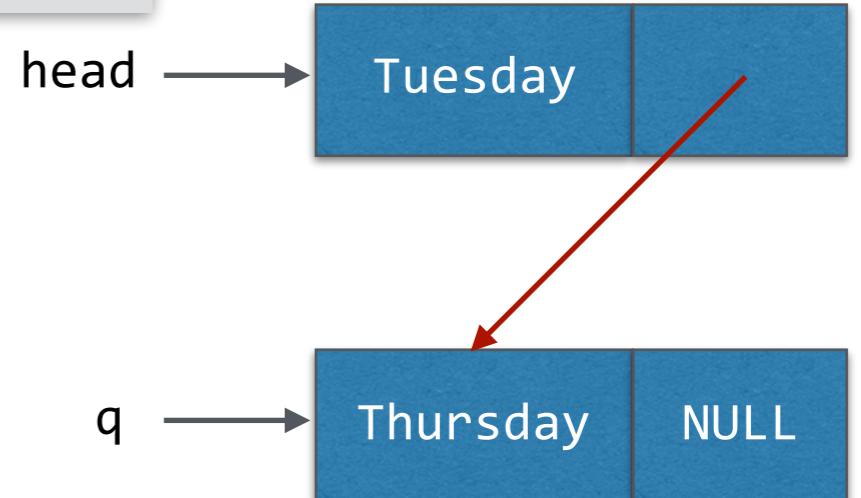
예제 프로그램

```
int main()
{
    Node *head = (Node *)malloc(sizeof(Node));
    head->data = "Tuesday";
    head->next = NULL;

    Node *q = (Node *)malloc(sizeof(Node));
    q->data = "Thursday";
    q->next = NULL;
    head->next = q;

    q = (Node *)malloc(sizeof(Node));
    q->data = "Monday";
    q->next = head;
    head = q;

    Node *p = head;
    while(p!=NULL) {
        printf("%s\n", p->data);
        p = p->next;
    }
}
```



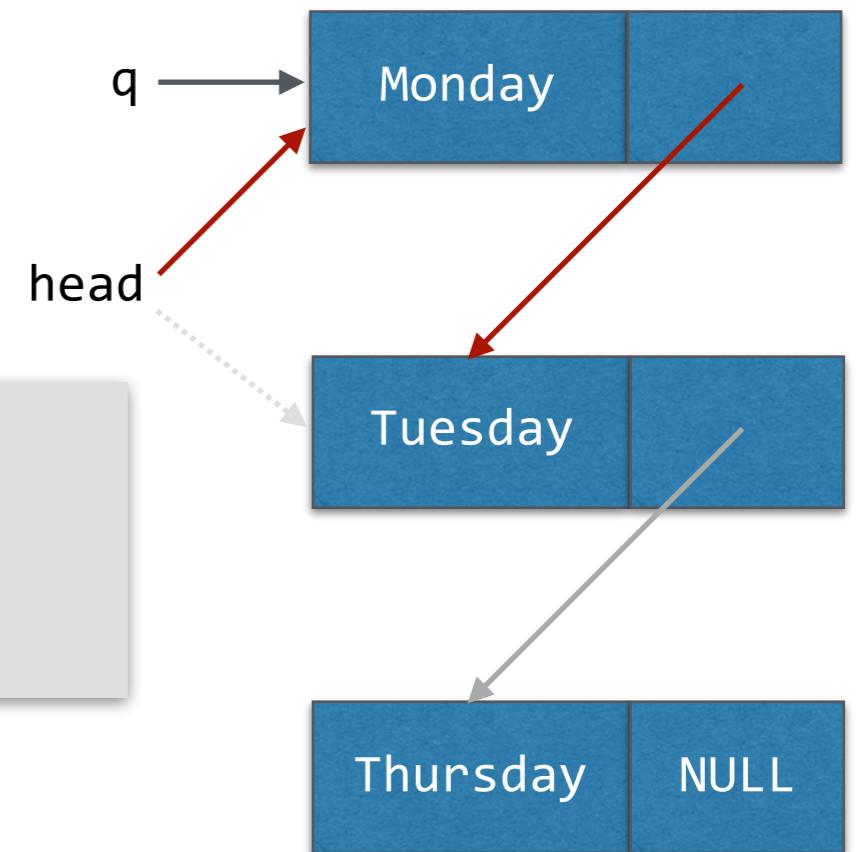
예제 프로그램

```
int main()
{
    Node *head = (Node *)malloc(sizeof(Node));
    head->data = "Tuesday";
    head->next = NULL;

    Node *q = (Node *)malloc(sizeof(Node));
    q->data = "Thursday";
    q->next = NULL;
    head->next = q;

    q = (Node *)malloc(sizeof(Node));
    q->data = "Monday";
    q->next = head;
    head = q;

    Node *p = head;
    while(p!=NULL) {
        printf("%s\n", p->data);
        p = p->next;
    }
}
```



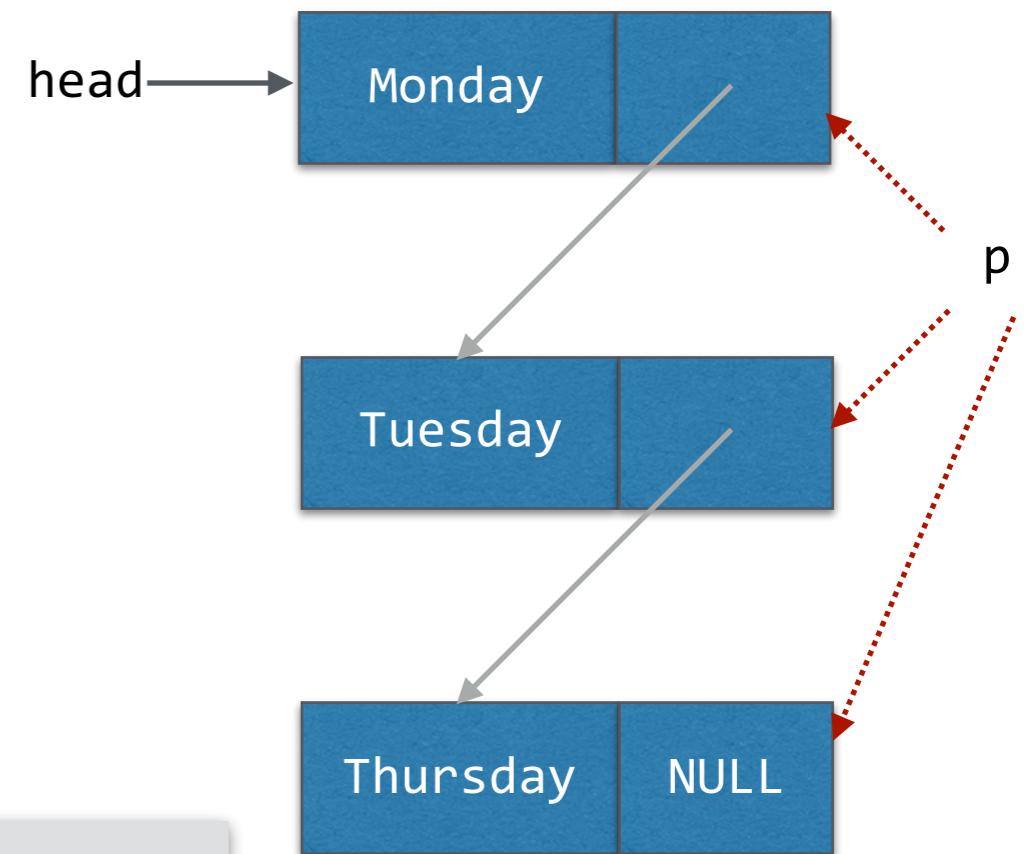
예제 프로그램

```
int main()
{
    Node *head = (Node *)malloc(sizeof(Node));
    head->data = "Tuesday";
    head->next = NULL;

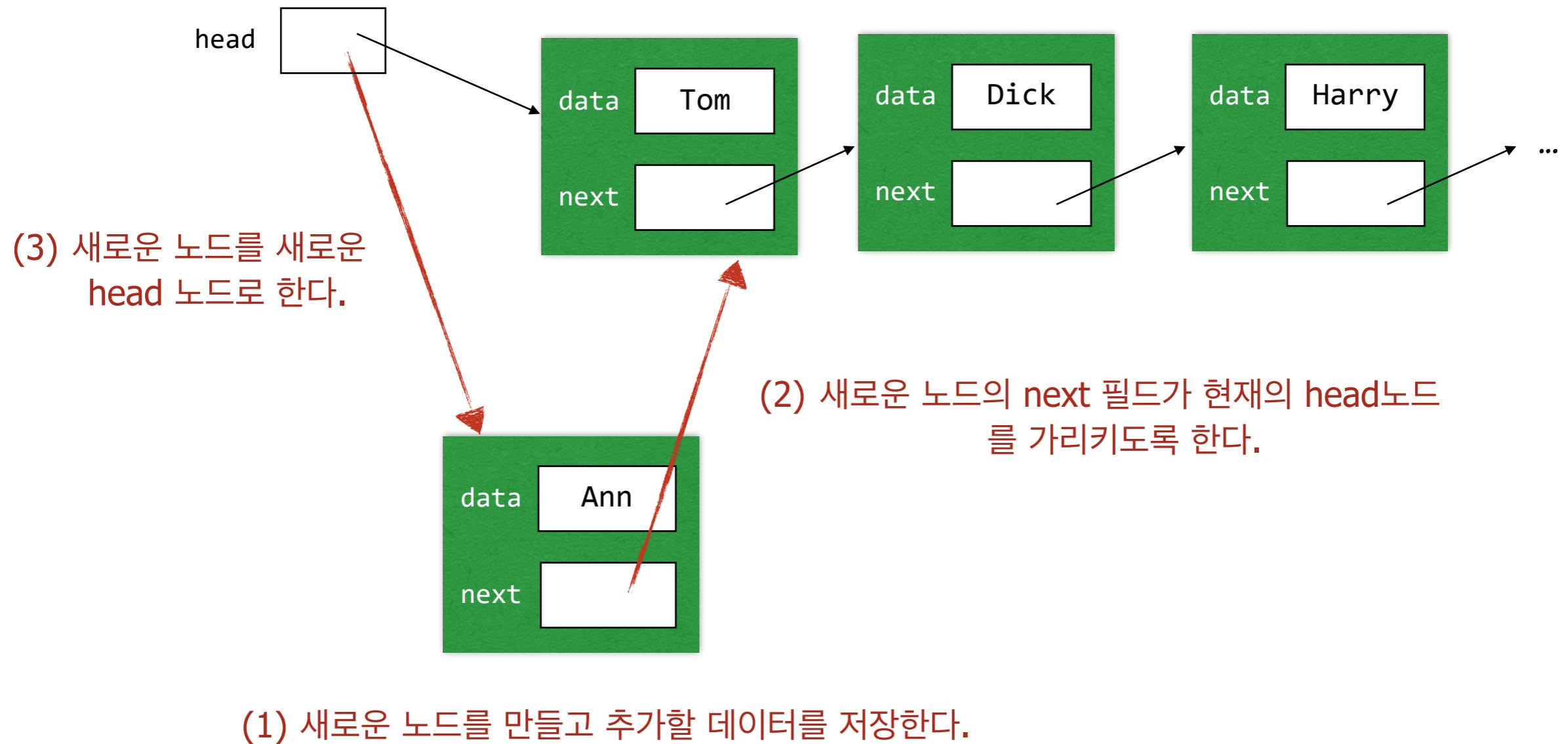
    Node *q = (Node *)malloc(sizeof(Node));
    q->data = "Thursday";
    q->next = NULL;
    head->next = q;

    q = (Node *)malloc(sizeof(Node));
    q->data = "Monday";
    q->next = head;
    head = q;

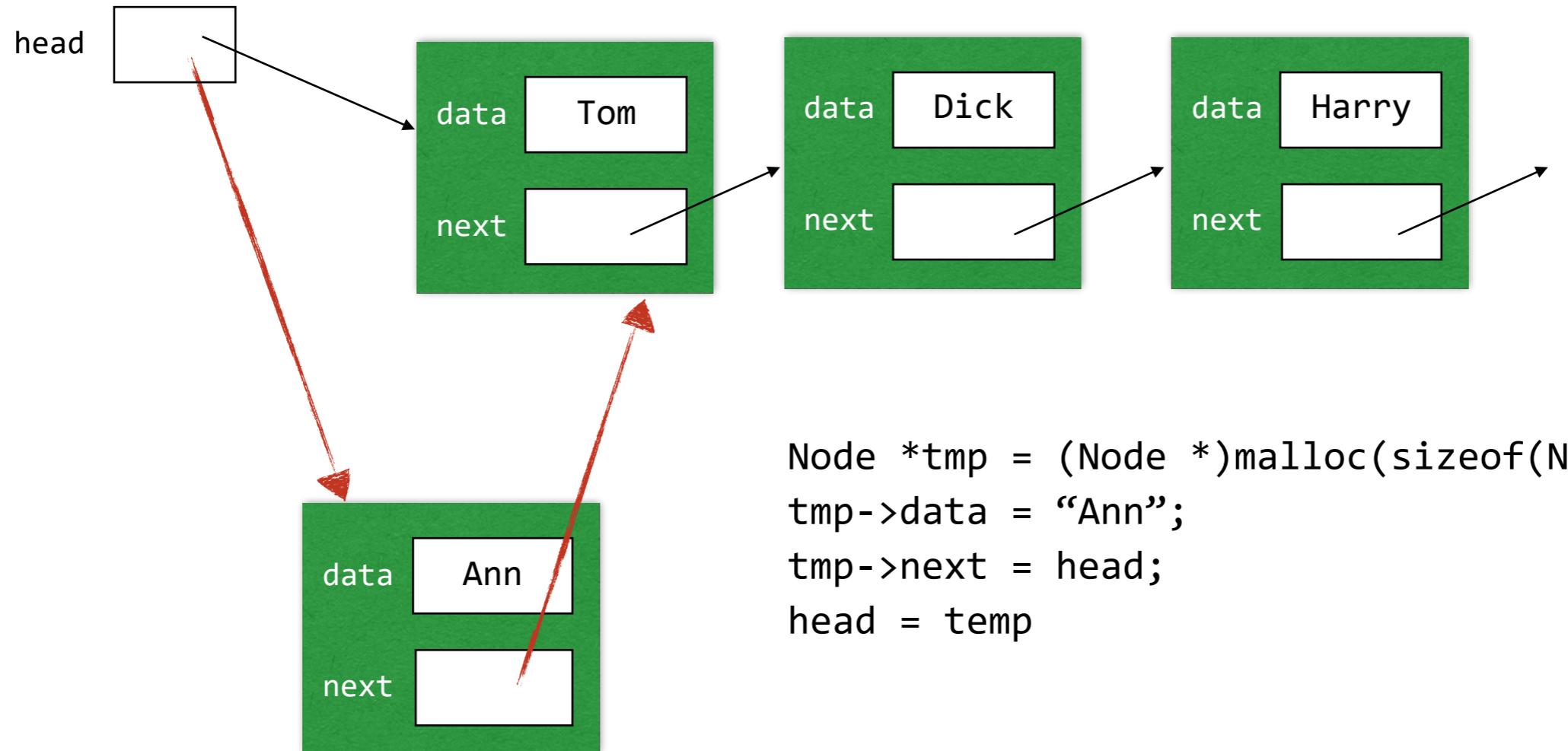
    Node *p = head;
    while(p!=NULL) {
        printf("%s\n", p->data);
        p = p->next;
    }
}
```



연결리스트의 맨 앞에 새로운 노드 삽입하기



연결리스트의 맨 앞에 새로운 노드 삽입하기



연결 리스트를 다루는 프로그램에서 가장 주의할 점은 내가 작성한 코드가 일반적인 경우만이 아니라 특수한 경우에도 문제 없이 작동하는지 확인하는 것이다. 이 경우에는 기존의 연결 리스트의 길이가 0인 경우, 즉 head가 NULL인 경우에도 문제가 없는지 확인해야 한다. 문제가 없는가?

연결리스트의 맨 앞에 새로운 노드 삽입하기

첫번째 노드를 가리키는 포인터 head가 전역변수인 경우

```
void add_first(char *item)
{
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->data = item;
    temp->next = head;
    head = temp;
}
```

연결리스트의 맨 앞에 새로운 노드 삽입하기

포인터 변수 head 의 주소를 매개변수로
받는다.



```
void add_first(Node **ptr_head, char *item)
{
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->data = item;
    temp->next = *ptr_head;
    *ptr_head = temp; ← 바뀐 head노드의 주소를 포인터를 이용하여 변수 head에 쓴다.
}
```

첫번째 노드를 가리키는 포인터 head가 전역변수가 아닌 경우

이렇게 구현할 경우 이 함수는 다음과 같이 호출해야 한다.

```
add_first( &head, item_to_store );
```

연결리스트의 맨 앞에 새로운 노드 삽입하기

첫번째 노드를 가리키는 포인터 head가 전역변수가 아닌 경우

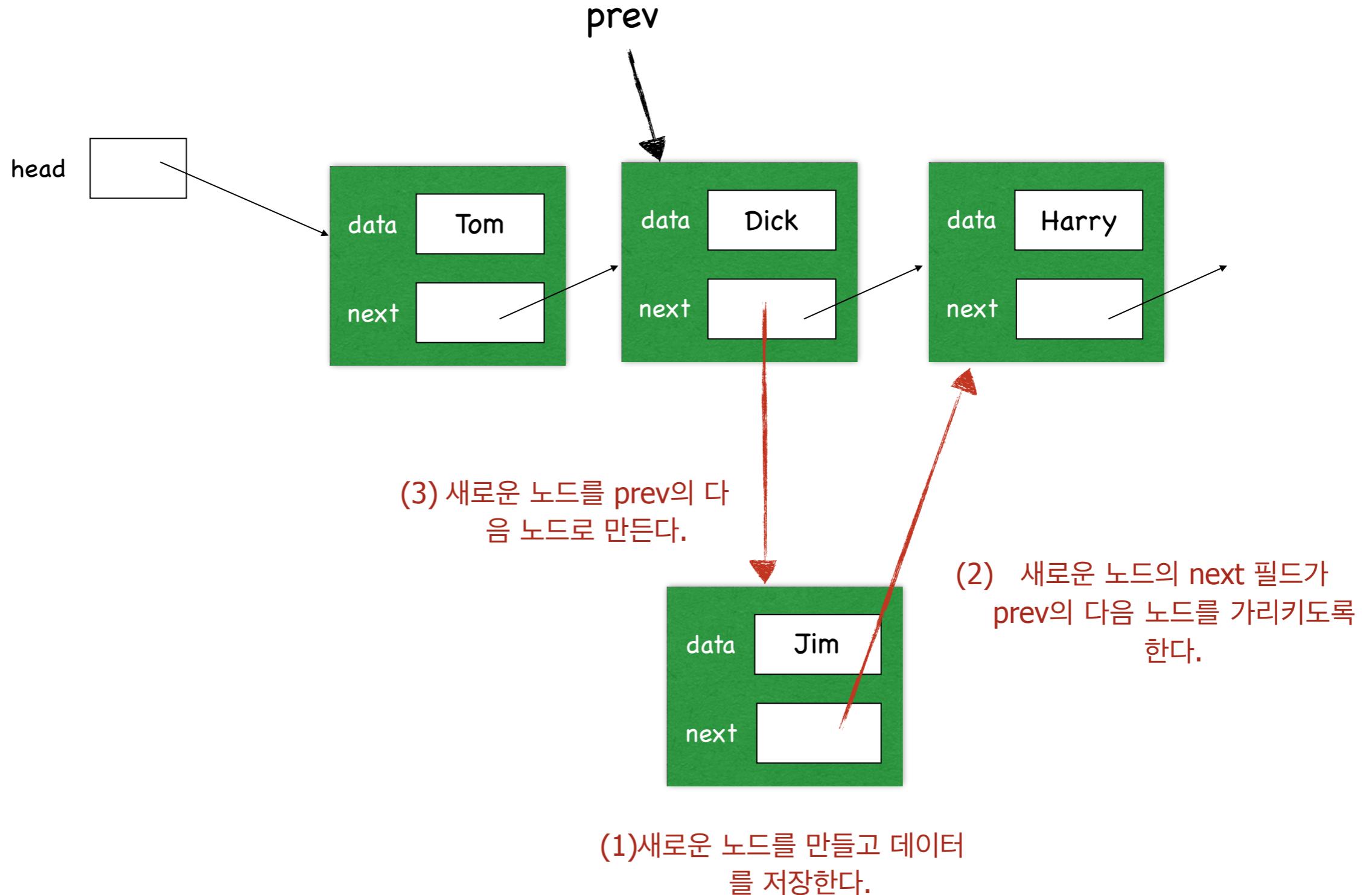
```
Node *add_first(Node *head, char *item)
{
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->data = item;
    temp->next = head;
    return temp; ← 새로운 head 노드의 주소를 return한다.
}
```

이렇게 구현할 경우 이 함수는 다음과 같은 식으로 호출해야 한다.

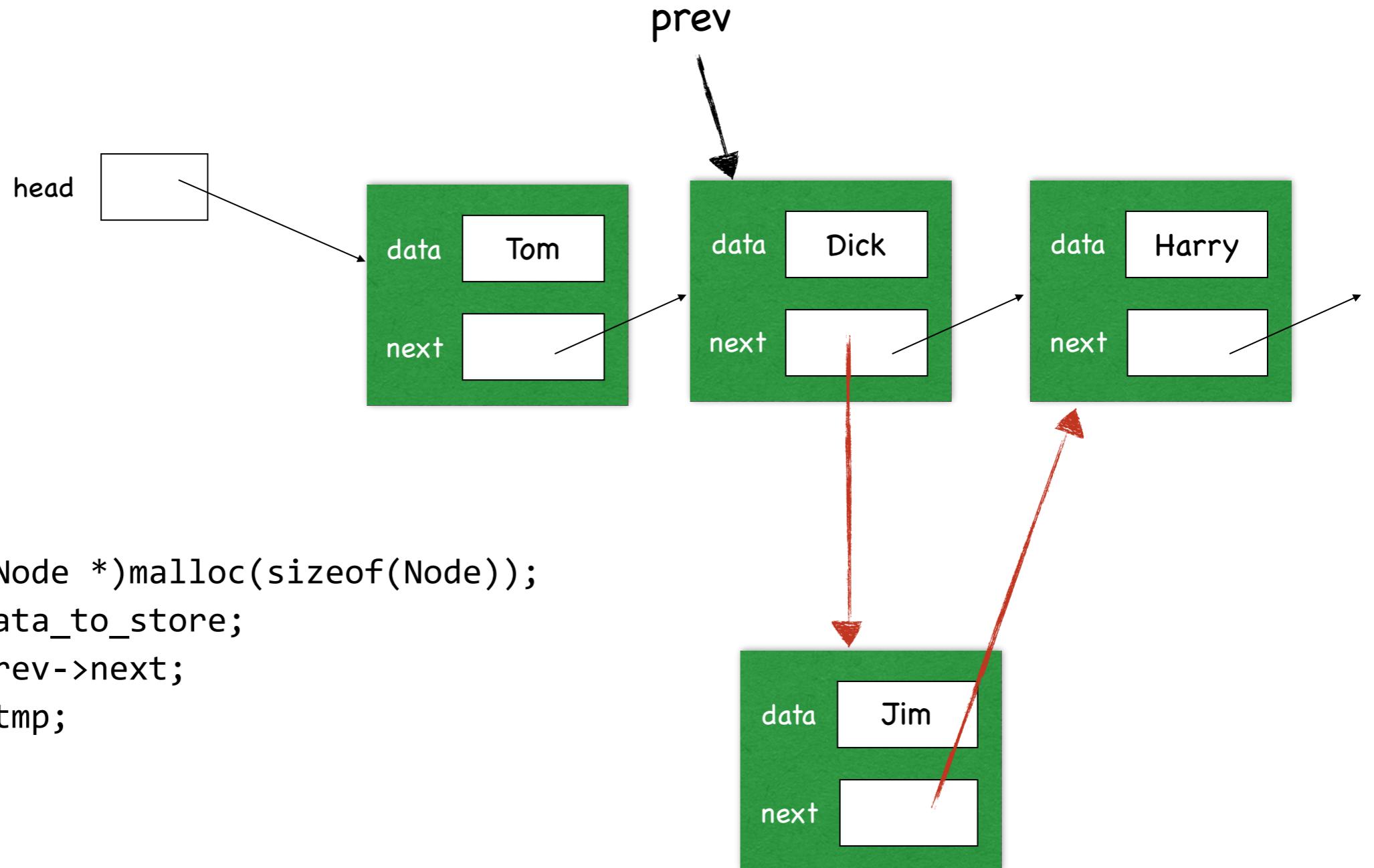
```
head = add_first(head, item_to_store);
```

어떤 노드 뒤에 새로운 노드 삽입하기

포인터 prev가 가리키는 노드 바로 뒤에 새로운 노드를 만들어 삽입한다.



어떤 노드 뒤에 새로운 노드 삽입하기



insert_after()

삽입에 성공하면 1, 아니면 0을 반환한다.

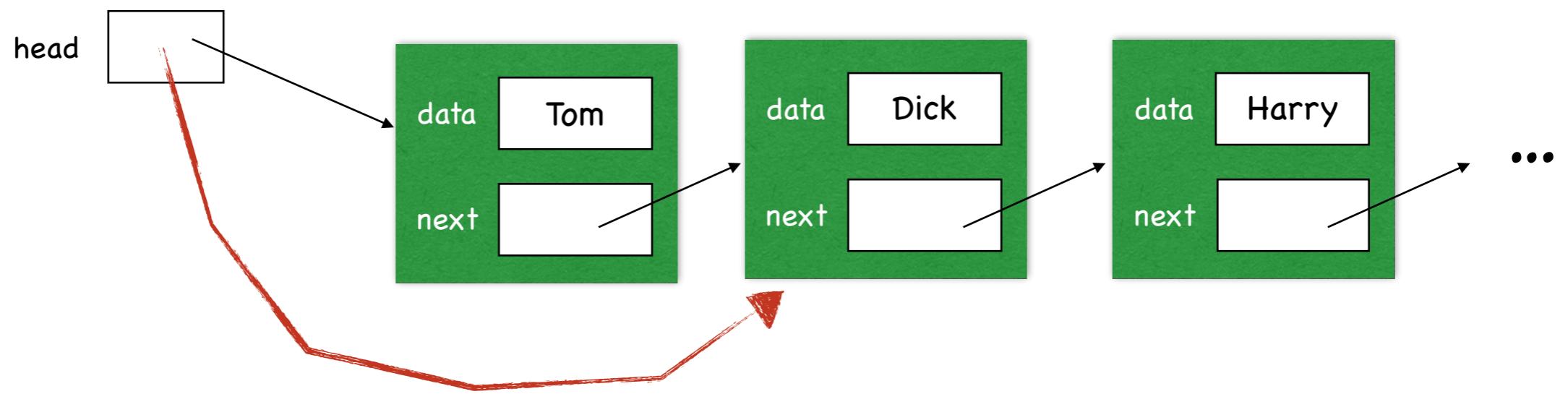


```
int add_after(Node *prev, char *item)  
{
```

```
}
```

연결리스트에 새로운 노드를 삽입할 때 삽입할 위치의 바로 앞 노드의 주소가 필요하다. 즉 어떤 노드의 뒤에 삽입하는 것은 간단하지만, 반대로 어떤 노드의 앞에 삽입하는 것은 간단하지 않다.

첫번째 노드의 삭제



head가 현재 head 노드의 다음 노드를 가리키게 만든다.

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

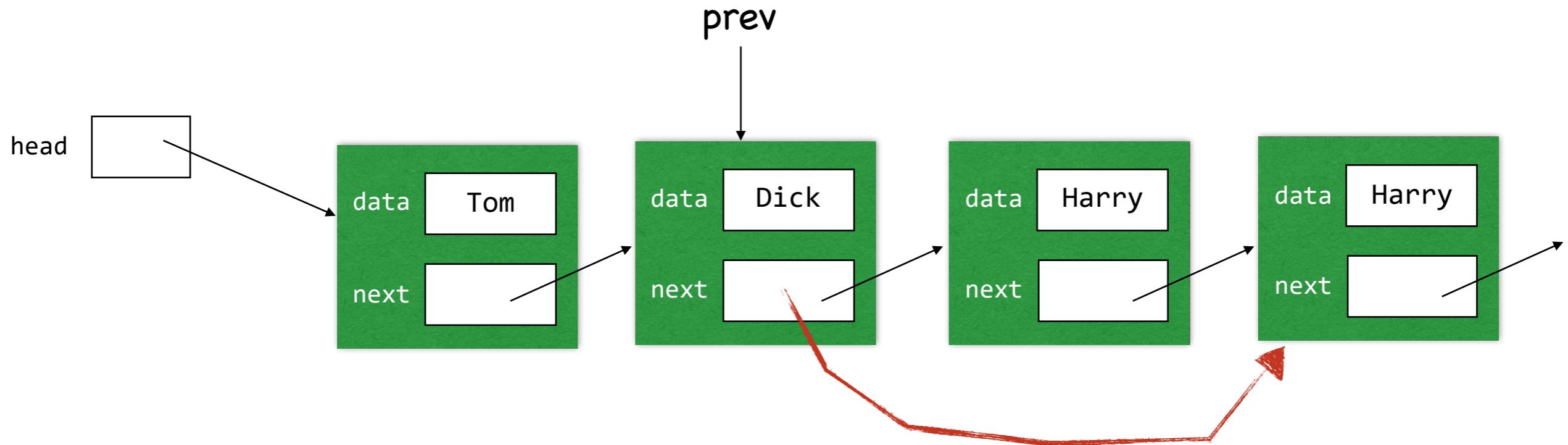
remove_first()

연결리스트의 첫번째 노드를 삭제하고 그 노드의 주소를
반환한다.

```
Node * remove first () {  
}  
}
```

어떤 노드의 다음 노드 삭제하기

prev가 가리키는 노드의 다음 노드를 삭제한다.



prev의 다음 노드가 null이 아니라면 prev의 next 필드
가 현재 next노드의 다음 노드를 가리키게 만든다.

```
if (prev->next != NULL)  
    prev->next = prev->next->next;
```

remove_after()

```
Node *remove_after (Node *prev) {  
}  
}
```

단순연결리스트에 어떤 노드를 삭제할 때는 삭제할 노드의 바로 앞 노드의 주소가 필요하다. 삭제할 노드의 주소만으로는 삭제할 수 없다.

연결리스트 순회하기

```
Node *find(char *word) {  
    Node *p = head;  
    while (p != NULL) {  
        if (strcmp(p->data, word)==0)  
            return p;  
        p = p->next;  
    }  
    return NULL;  
}
```

연결리스트의 노드들을 처음부터 순서대로 방문하는 것을 순회(traverse)한다고 말한다. 이 함수는 입력된 문자열 word와 동일한 단어를 저장한 노드를 찾아서 그 노드의 주소를 반환한다. 그것을 위해서 연결리스트를 순회한다.

연결리스트 순회하기

연결리스트의 index번째 노드의 주소를 반환한다.

```
Node *get_node(int index) {  
    if (index < 0)  
        return NULL;  
    Node *p = head;  
    for (int i=0; i<index && p!=NULL; i++)  
        p = p->next;  
    return p;  
}
```

void add(int index,...)

연결리스트의 index번째 위치에 새로운 노드를
만들어서 삽입한다.

```
int add(int index, char *item) {  
  
}  
}
```

char *remove(int index,...)

index번째 노드를 삭제하고, 그 노드에 저장된 데이터를 반환한다.

```
Node *remove(int index) {  
}  
}
```

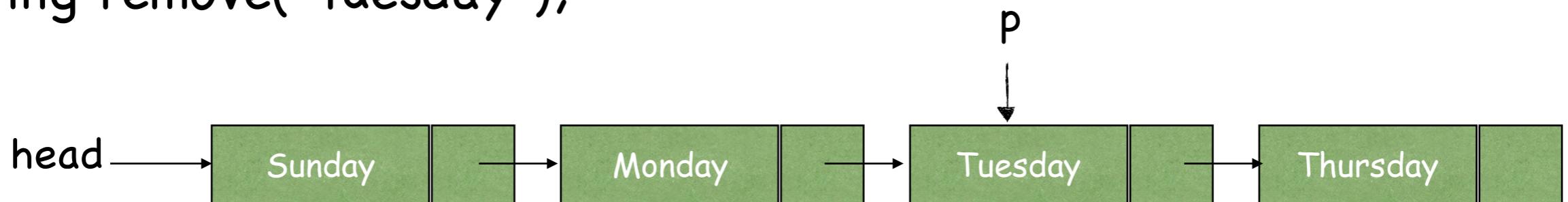
char *remove(char *item)

입력된 스트링을 저장한 노드를 찾아 삭제한다.
삭제된 노드에 저장된 스트링을 반환한다.

```
Node *remove(char *item) {  
    Node *p = head;  
    while (p!=NULL && strcmp(p->data, item)!=0)  
        p=p->next;  
  
    ???  
}
```

삭제할 노드를 찾았음. 하지만 노드를 삭제하기 위해서는 삭제할 노드가 아니라 직전 노드의 주소가 필요함.

trying remove("Tuesday");

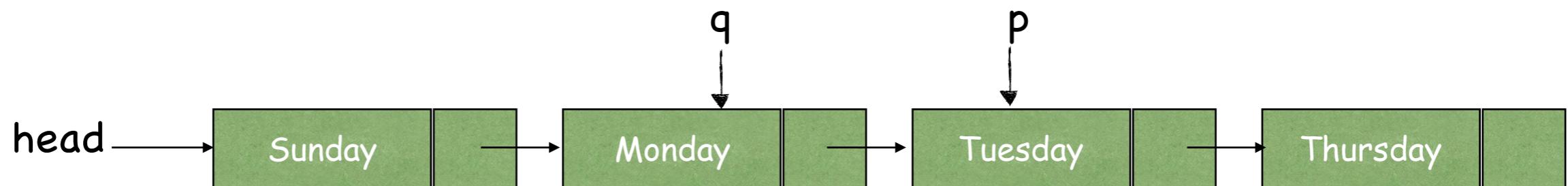


char *remove(char *item)

```
Node *remove(char * item) {
```

```
}
```

q는 항상 p의 직전 노드를 가리킴. p가 첫번째 노드일 경우 q는 NULL임



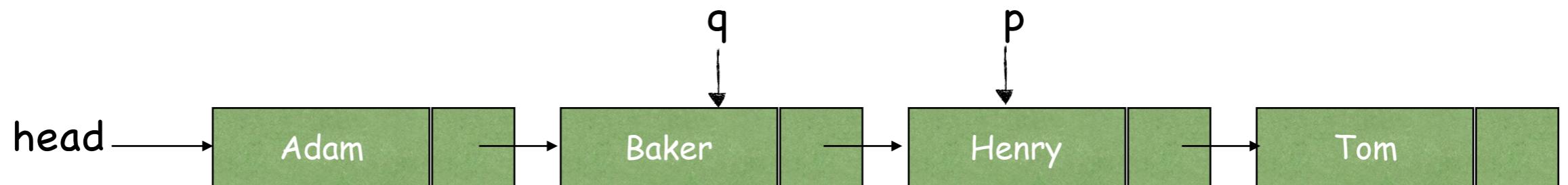
void add_to_ordered_list(char * item)

연결리스트에 데이터들이 오름차순으로 정렬되어 있다
는 가정하에서 새로운 데이터를 삽입한다.

```
void add_to_ordered_list(char *item) {
```

```
}
```

trying add_to_ordered_list("David");



다항식
polynomial

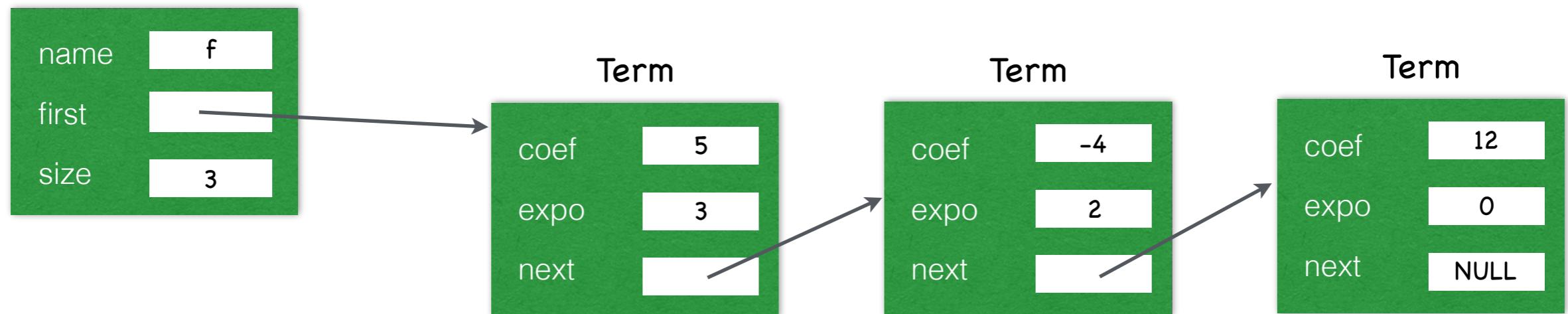
Use Case

```
$ f= 2x^5 + 3x -4x^2- 8  
$ calc f 1  
-7  
$ g = x^3 + 2x^2 + 1  
$ g= f+ g  
$ print g  
g = 2x^5+x^3-2x^2+3x-7  
$ f = x^2-x + 1  
$ print f  
f = x^2-x+1  
$ exit
```

1. 1원 다항식을 정의할 수 있다. 다항식의 이름은 x를 제외한 영문 소문자이다 (예: f, g 등).
2. 변수는 항상 x이다.
3. 각 항의 지수는 음이 아닌 정수이고, 계수는 정수이다.
4. =, +, - 등의 앞뒤로 하나 이상의 공백이 있을 수도 있고 없을 수도 있다.
5. 항들이 반드시 차수에 대해 내림차순으로 입력되는 것은 아니며, 동일 차수의 항이 여럿 있을 수도 있다.
6. 함수는 항상 차수에 대한 내림차순으로 정렬되어 저장되고 출력되어야 한다.
7. 동일 이름의 함수를 새로 정의할 수도 있다. 이 경우 기존의 함수는 지워진다.

- 연결리스트를 이용하여 하나의 다항식을 표현하는 구조체 **Polynomial**을 정의 한다.
- 다항식을 항들의 연결 리스트로 표현한다.
- 항들을 차수에 대해서 내림차순으로 정렬하여 저장하며, 동일 차수의 항을 2개 이상 가지지 않게 한다. 또한 계수가 0인 항이 존재하지 않게 한다.
- 하나의 항은 계수와 지수에 의해 정의된다. 하나의 항을 표현하기 위해 구조체 **Term**을 정의한다.
- 변수 **x**의 값이 주어질 때 다항식의 값을 계산하는 함수를 제공한다.

Polynomial



$$f(x) = 5x^3 - 4x^2 + 12$$

구조체 Term

```
struct term {  
    int coef;  
    int expo;  
    struct term *next;  
};  
  
typedef struct term Term;
```

구조체 Polynomial

```
typedef struct polynomial {  
    char name;  
    Term *first;  
    int size;  
} Polynomial;
```

```
Polynomial *polys[MAX_POLYS];  
int n = 0;
```



polys는 다항식들에 대한 포인터의 배열이다.



저장된 다항식의 개수이다.

create instance

```
Term *create_term_instance() {  
    Term *t = (Term *)malloc(sizeof(Term));  
    t->coef = 0;  
    t->expo = 0;  
    t->next = NULL;  
    return t;  
}
```

다항식 객체를 생성할 때 이름을 지정해주도록
만들어 보았다.

```
Polynomial *create_polynomial_instance(char name) {  
    Polynomial *ptr_poly = (Polynomial *)malloc(sizeof(Polynomial));  
    ptr_poly->name = name;  
    ptr_poly->size = 0;  
    ptr_poly->first = NULL;  
    return ptr_poly;  
}
```

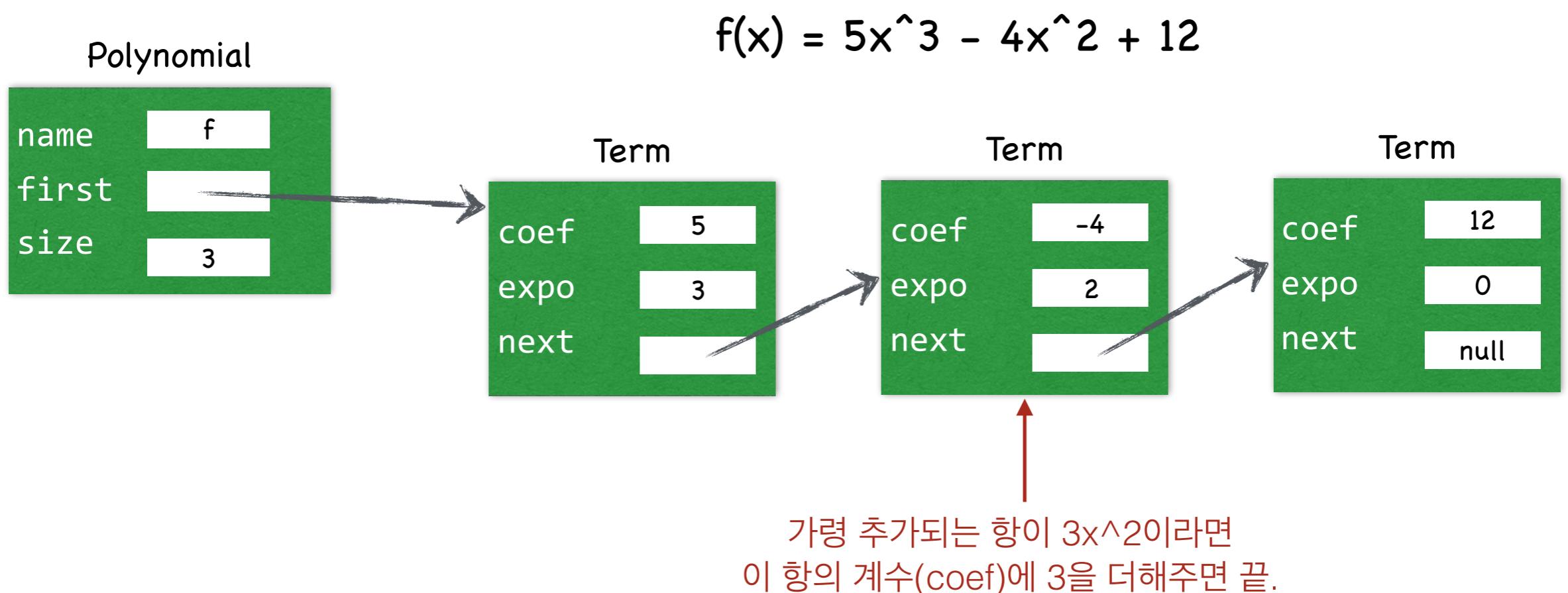
동적으로 생성된 객체를 적절하게 초기화해주지 않는 것이
종종 프로그램의 오류를 야기한다.
이렇게 객체를 생성하고 기본값으로 초기화해주는 함수를
따로 만들어 사용하는 것은 좋은 방법이다.

add_term(int c, int e, Polynomial *poly)

- **poly가 가리키는 다항식에 새로운 하나의 항 (c,e)를 추가하는 함수**

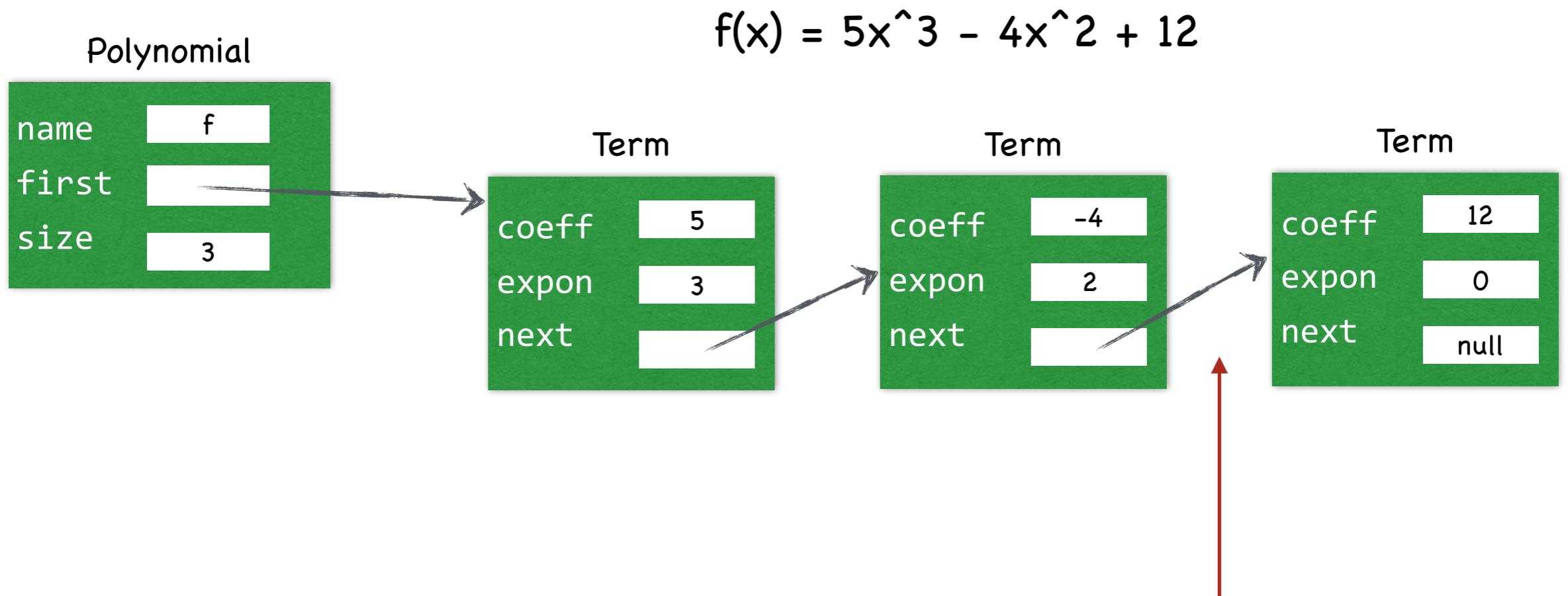
- **두 가지 경우**

- 추가하려는 항과 동일 차수의 항이 이미 있는 경우: 기존 항의 계수만 변경
- 그렇지 않은 경우: 새로운 항을 삽입 (항들은 차수의 내림차순으로 항상 정렬 됨)



add_term(int c, int e, Polynomial *poly)

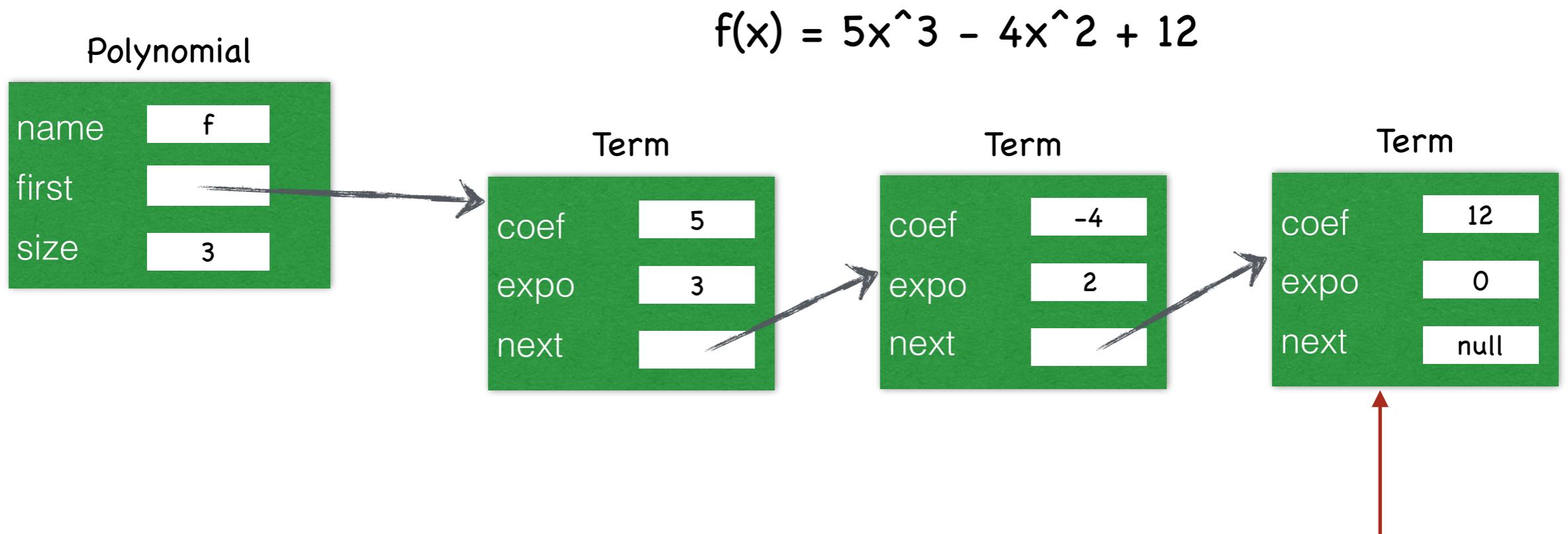
- 새로운 항을 삽입해야하는 경우: 예를 들어 $5x$ 를 삽입한다고 가정



새로운 항이 삽입될 위치는 여기이다. 이 위치를 찾기 위해서는 새로운 항보다 차수가 작거나 같은 항이 나올때 까지 첫 번째 항부터 순서대로 검사해야 한다.

add_term(int c, int e, Polynomial *poly)

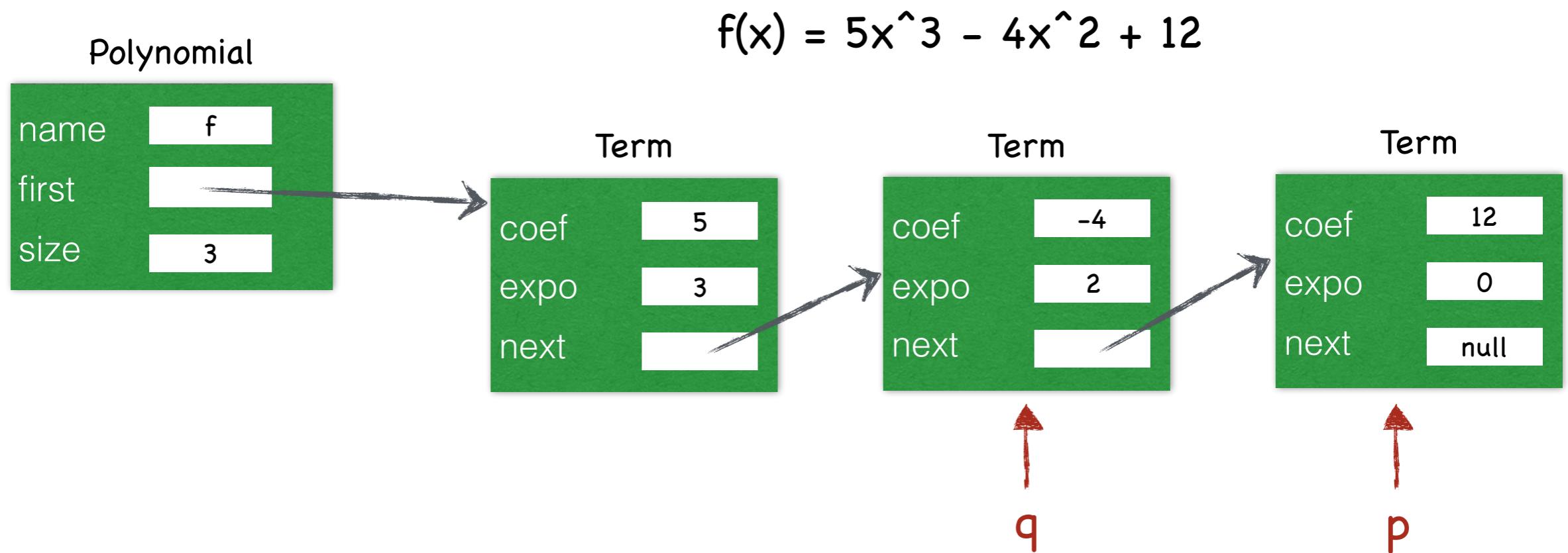
- ➊ 새로운 항을 삽입해야하는 경우: 예를 들어 $5x$ 를 삽입한다고 가정



새로운 항보다 차수가 작거나 같은 항이 나올때 까지 순서대로 검사하면 이 항에 도착하게 된다. 하지만 연결리스트에서 노드를 삽입하기 위해서는 이전 노드의 주소가 필요하다.

add_term(int c, int e, Polynomial *poly)

- 새로운 항을 삽입해야하는 경우: 예를 들어 $5x$ 를 삽입한다고 가정



```
Term p = first, q = null;  
while (p != null && p.expo > e) {  
    q = p; // q는 항상 p의 한발 뒤를 따라감  
    p = p.next;  
}
```

add_term(int c, int e, Polynomial *poly)

```
void add_term(int c, int e, Polynomial *poly) {
```

← 동일 차수의 항이 존재하는 경우

← 더했더니 계수가 0이 되는 경우

← q의 다음 노드를 삭제한다. 단, q가 NULL이면
첫번째 노드를 삭제한다.

← 불필요해진 노드 p를 free시킨다.

```
}
```

add_term(int c, int e, Polynomial *poly)

- ← 맨 앞에 삽입하는 경우
- ← q의 뒤, p의 앞에 삽입 (p는 null일 수도 있음)

}

eval(Polynomial *poly, int x)

```
int eval(Polynomial *poly, int x) { ← 다항식의 값을 계산하는 함수  
    ← 각각의 항의 값을 계산하여 더한다.  
}  
  
int eval(Term *term, int x) { ← 하나의 항의 값을 계산하는 함수  
}  
}
```

print_poly(Polynomial *poly)

```
void print_poly(Polynomial *p)
{
```

```
}
```

```
void print_term(Term *pTerm) {
    printf("%dx^%d", pTerm->coef, pTerm->expo);
}
```

이 함수는 완벽하지 않다.
개선하는 것은 과제로 남겨 둔다.

process_command()

```
void process_command()
{
    char command_line[BUFFER_LENGTH];
    char copied[BUFFER_LENGTH];
    char *command, *arg1, *arg2;

    while(1) {
        printf("$ ");
        if (read_line(stdin, command_line, BUFFER_LENGTH) <= 0)
            continue;
        strcpy(copied, command_line); ←———— 입력 라인을 복사해 둔다.
        command = strtok(command_line, " ");
        if (strcmp(command, "print") == 0) {
            arg1 = strtok(NULL, " ");
            if (arg1 == NULL) {
                printf("Invalid arguments.\n");
                continue;
            }
            handle_print(arg1[0]);
        }
    }
}
```

process_command()

```
else if (strcmp(command, "calc") == 0) {
    arg1 = strtok(NULL, " ");
    if (arg1 == NULL) {
        printf("Invalid arguments.\n");
        continue;
    }
    arg2 = strtok(NULL, " ");
    if (arg2 == NULL) {
        printf("Invalid arguments.\n");
        continue;
    }
    handle_calc(arg1[0], arg2);
}
else if (strcmp(command, "exit") == 0)
    break;
else {
    handle_definition(copied);           ←———— 다행식을 입력받아 정의하는 일을 한다.
}
}
```

handle_print()

```
void handle_print(char name) {  
    /* left as exercise */  
}
```

handle_calc()

```
void handle_calc(char name, char *x_str) {  
    /* left as exercise */  
}
```

handle_definition()

```
void handle_definition(char *expression) {
```

← 모든 공백 문자들을 제거한다.

handle_definition()

handle_definition()

}

erase_blanks

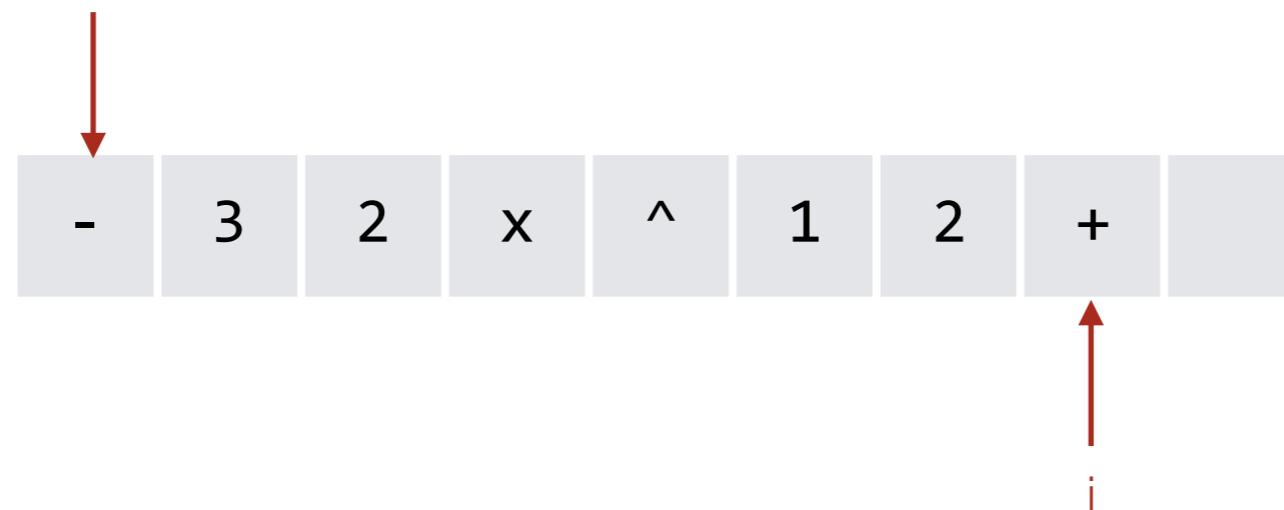
```
void erase_blanks(char *expression) {  
    /* left as exercise */  
}  
→
```

문자배열 expression에서 모든 공백
문자들을 제거하여 압축한다.
문자열의 끝에 '\0'를 추가해준다.

create_by_parse_polynomial

create_by_parse_polynomial

begin_term



```
int result = parse_and_add_term(body, begin_term, i, ptr_poly);
```

맨 첫번째 항은 +혹은 - 기호로 시작되지 않을수도 있음

parse_and_add_term

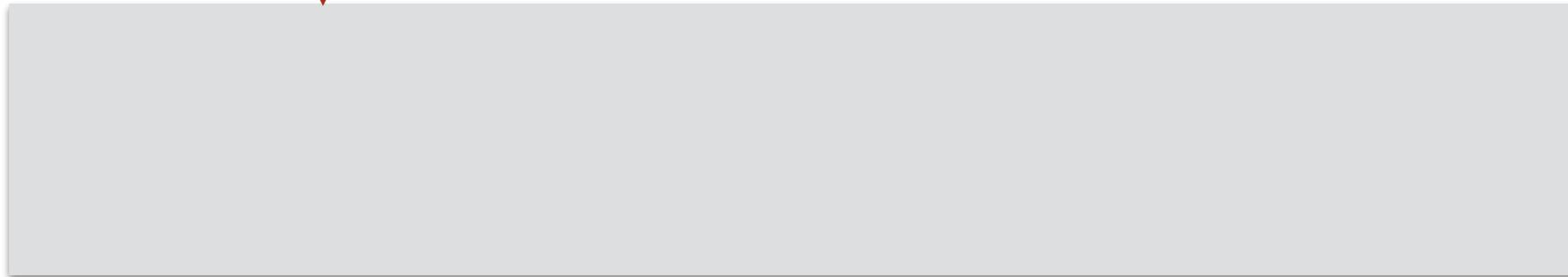
```
int parse_and_add_term(char *expr, int begin, int end, Polynomial *p_poly)
{
    int i=begin;
    int sign_coef = 1, coef = 0, expo = 1;
```

+ 혹은 - 기호로 부터 계수의 부호를 결정한다. 하지만 +혹은 -기호가 없을 수도 있다(첫 번째 항의 경우)

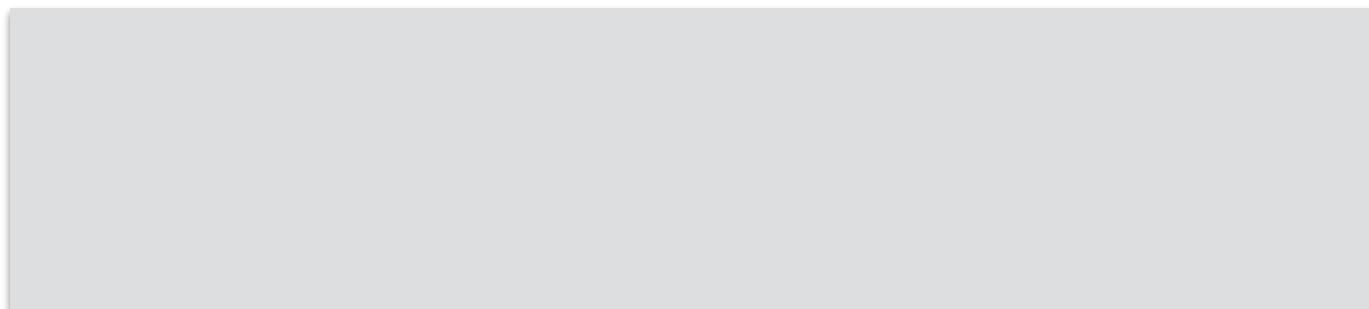


parse_and_add_term

digit들을 읽어서 계수의 절대값(coef)를 계산한다. 하지만 digit
들이 하나도 없을 수도 있다(계수가 1혹은 -1인 경우)



← coef가 0인 경우 계수는 0이 아니라 1
혹은 -1이다.



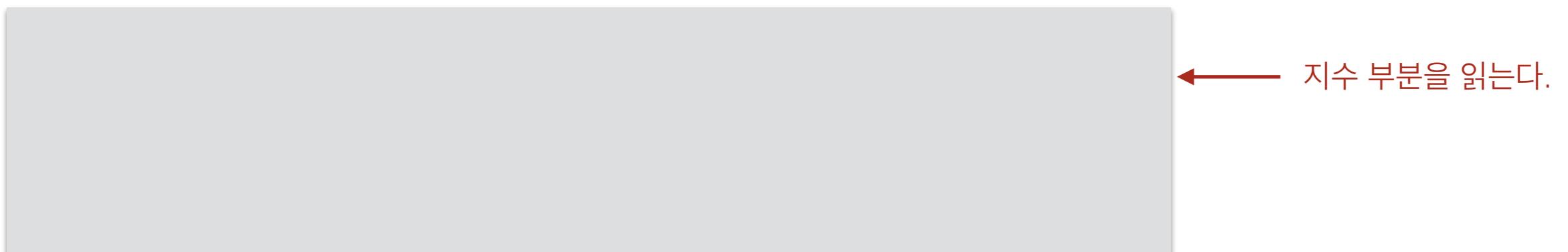
← 더 이상 항을 구성하는 문자가 없다면
상수항이라는 의미이다.

parse_and_add_term

```
if (expr[i] != 'x')      ← 계수 다음에 x가 아닌 다른 문자가 등장해서는 안된다.  
    return 0;  
i++;
```



```
if (expr[i] != '^')      ← x 다음에 ^가 아닌 다른 문자가 등장해서는 안된다.  
    return 0;  
i++;
```



```
add_term(coef, expo, p_poly);  
return 1;  
}
```

insert_polynomial

```
void insert_polynomial(Polynomial *ptr_poly) {
    for (int i=0; i<n; i++) {
        if (polys[i]->name == ptr_poly->name) {
            destroy_polynomial(polys[i]);
            polys[i] = ptr_poly;
            return;
        }
    }
    polys[n++] = ptr_poly;
}
```

다항식을 덮어쓸 경우에는 기존의 다항식
객체를 free시켜줘야 한다.

destroy_polynomial

```
void destroy_polynomial(Polynomial *ptr_poly)
{
    if (ptr_poly == NULL)
        return;
    Term *t = ptr_poly->first, *tmp;
    while (t!=NULL) {
        tmp = t;
        t = t->next;      ← 다행식에 속한 모든 항들을 free시킨다.
        free(tmp);
    }
    free(ptr_poly);     ← ptr_poly가 가리키는 다행식 객체를 free시킨다.
}
```

create_by_add_two_polynomials

```
Polynomial *create_by_add_two_polynomials(char name, char f, char g) {  
    /* left as exercise */  
}
```

새로운 empty 다항식을 만든 후 두 다항식의
모든 항들을 add해주면 된다.

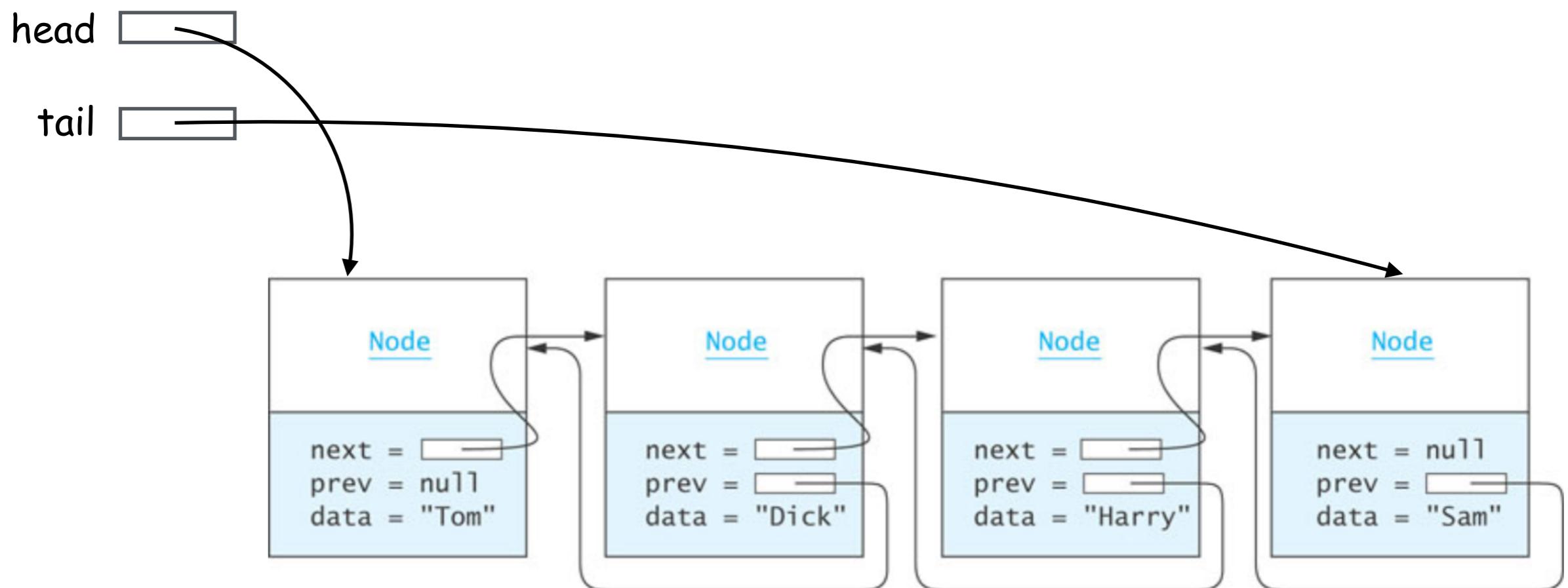
단방향 연결 리스트(single linked list)의 한계:

- 어떤 노드의 앞에 새로운 노드를 삽입하기 어려움
- 삭제의 경우에 항상 삭제할 노드의 앞 노드가 필요
- 단방향의 순회만이 가능

이중 연결 리스트

- 각각의 노드가 다음(next) 노드와 이전(previous) 노드의 주소를 가지는 연결 리스트
- 양방향의 순회(traverse)가 가능

이중연결리스트



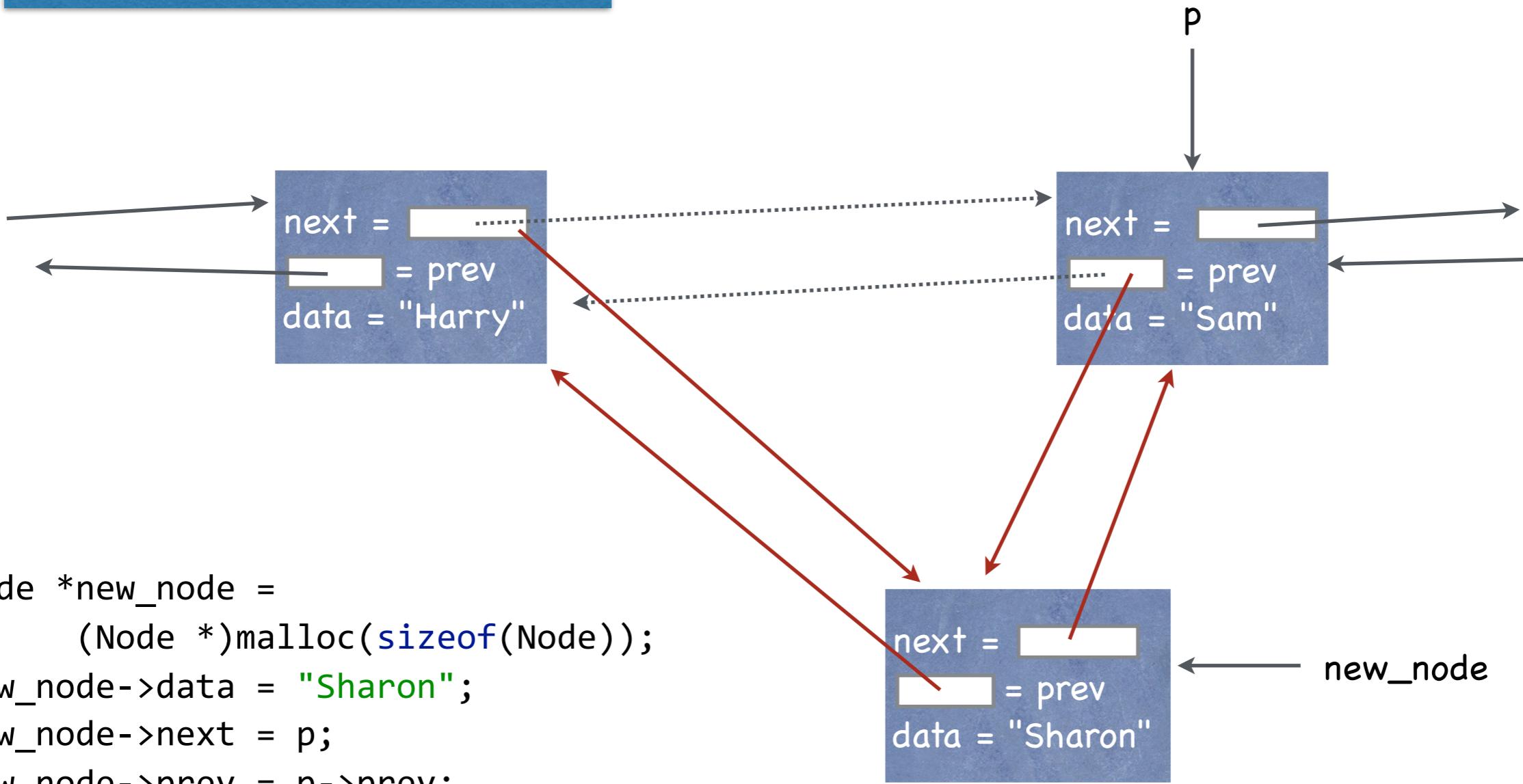
Node

각 노드에 하나의 문자열이 저장된다고 가정하자.

```
struct node {  
    char *data;  
    struct node *next;  
    struct node *prev;  
};  
  
typedef struct node Node;  
  
Node *head;  
Node *tail;  
int size = 0;
```

노드 삽입

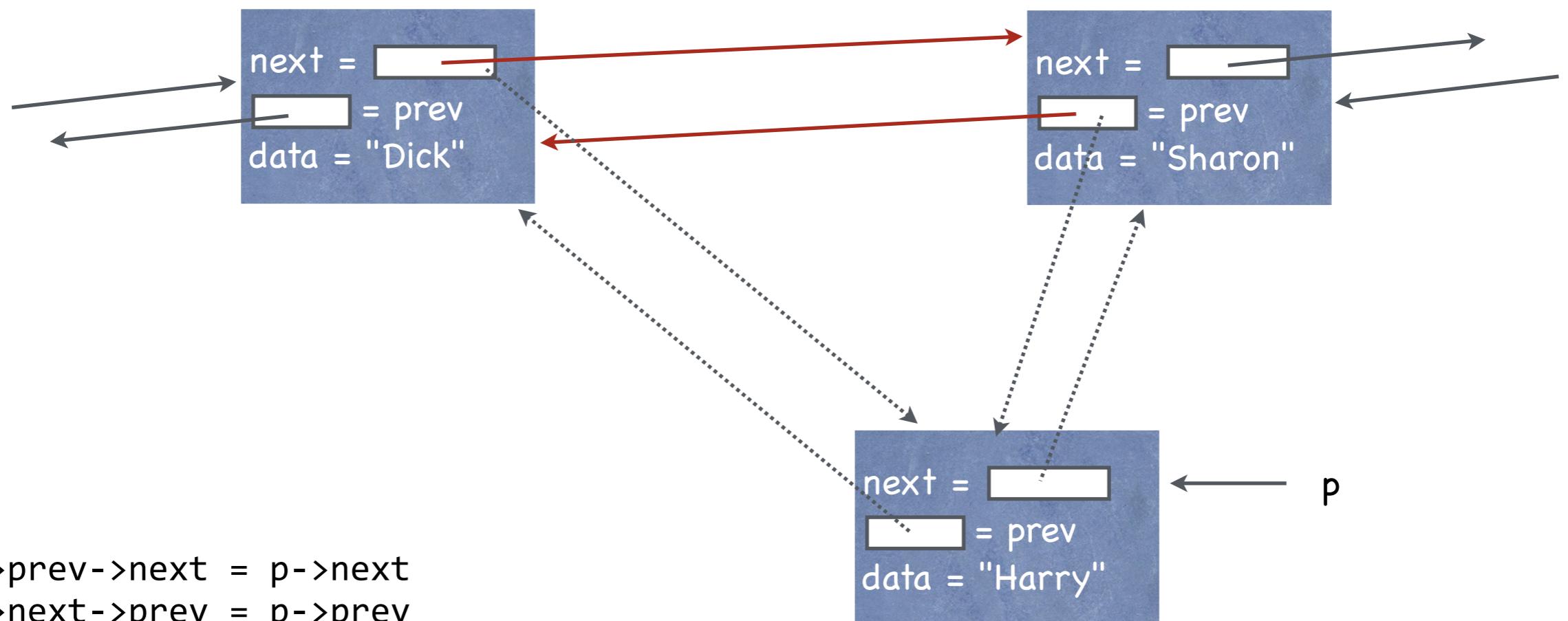
p가 가리키는 노드 앞에 새로운 노드를 삽입하는 경우



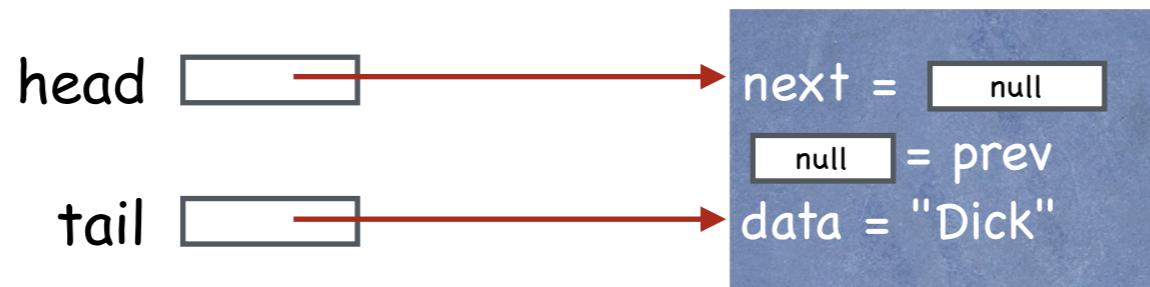
```
Node *new_node =
    (Node *)malloc(sizeof(Node));
new_node->data = "Sharon";
new_node->next = p;
new_node->prev = p->prev;
p->prev->next = new_node;
p->prev = new_node;
```

노드 삭제

p가 가리키는 노드를 삭제하는 경우



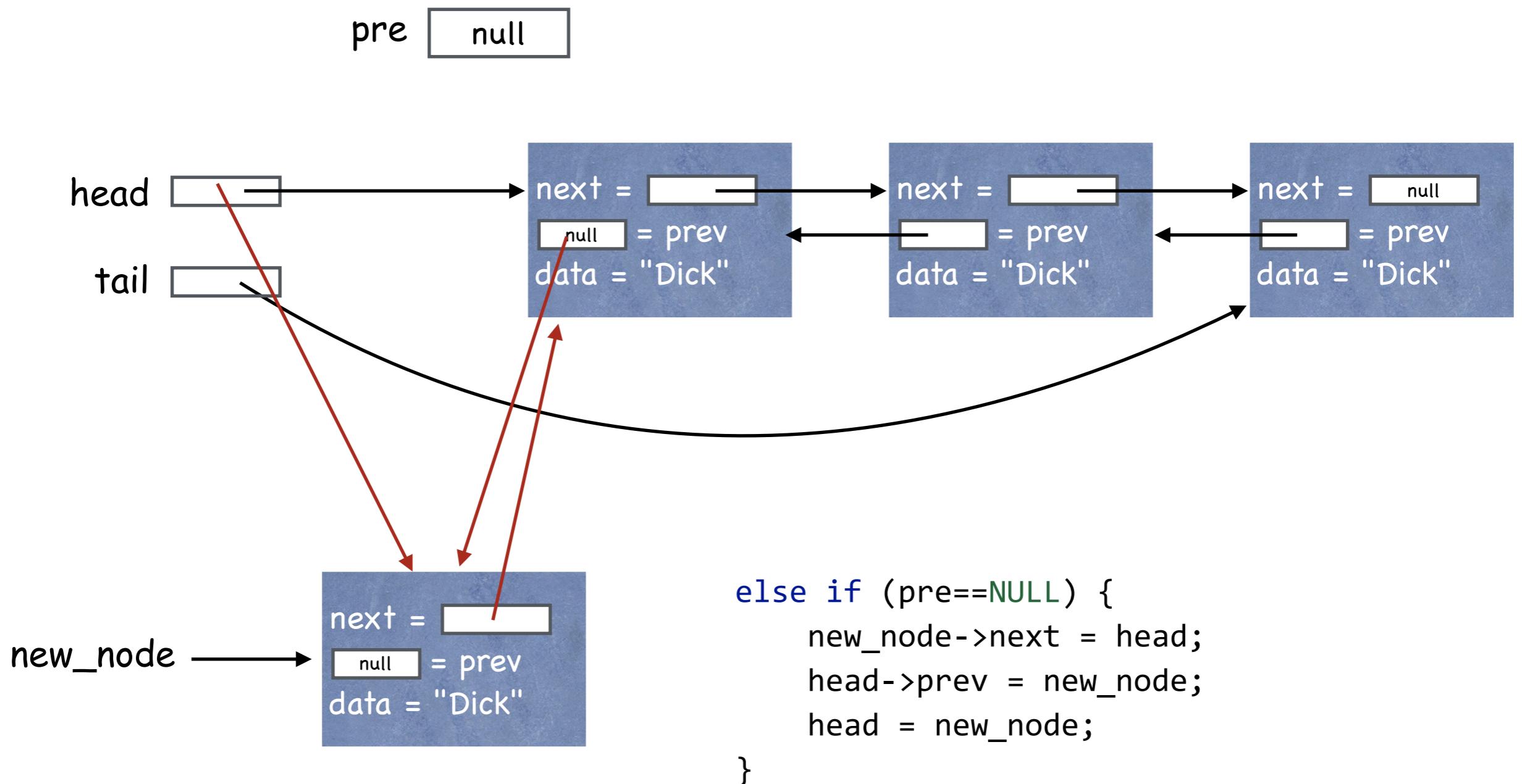
add_after(Node *pre, char *item) - to empty list



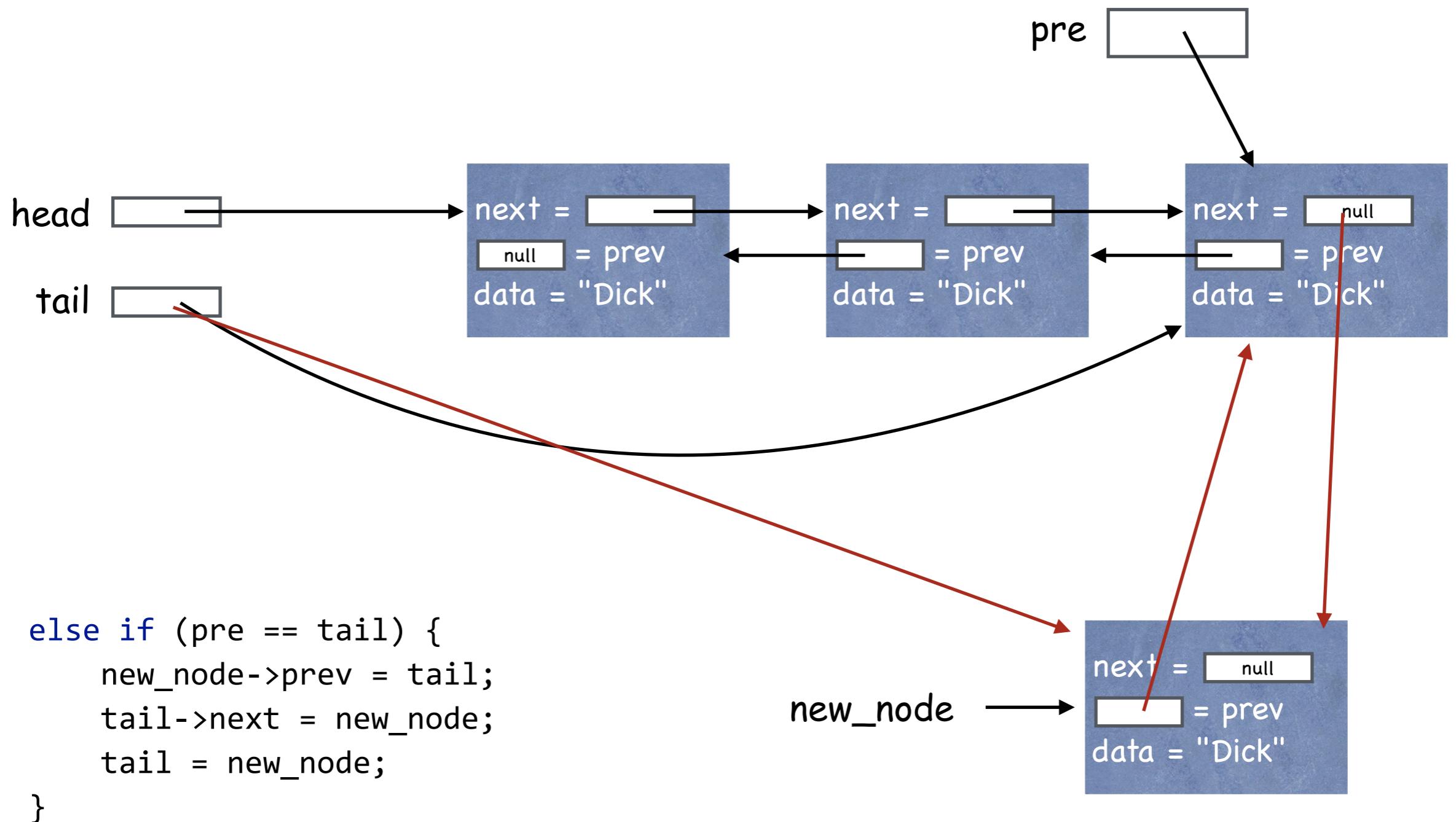
```
void add_after(Node *pre, char *item)
{
    Node *new_node = (Node *)malloc(sizeof(Node));
    new_node->data = item;
    new_node->prev = NULL;
    new_node->next = NULL;

    if (pre == NULL && head == null) {
        head = node_node;
        tail = node_node;
    }
    ...
}
```

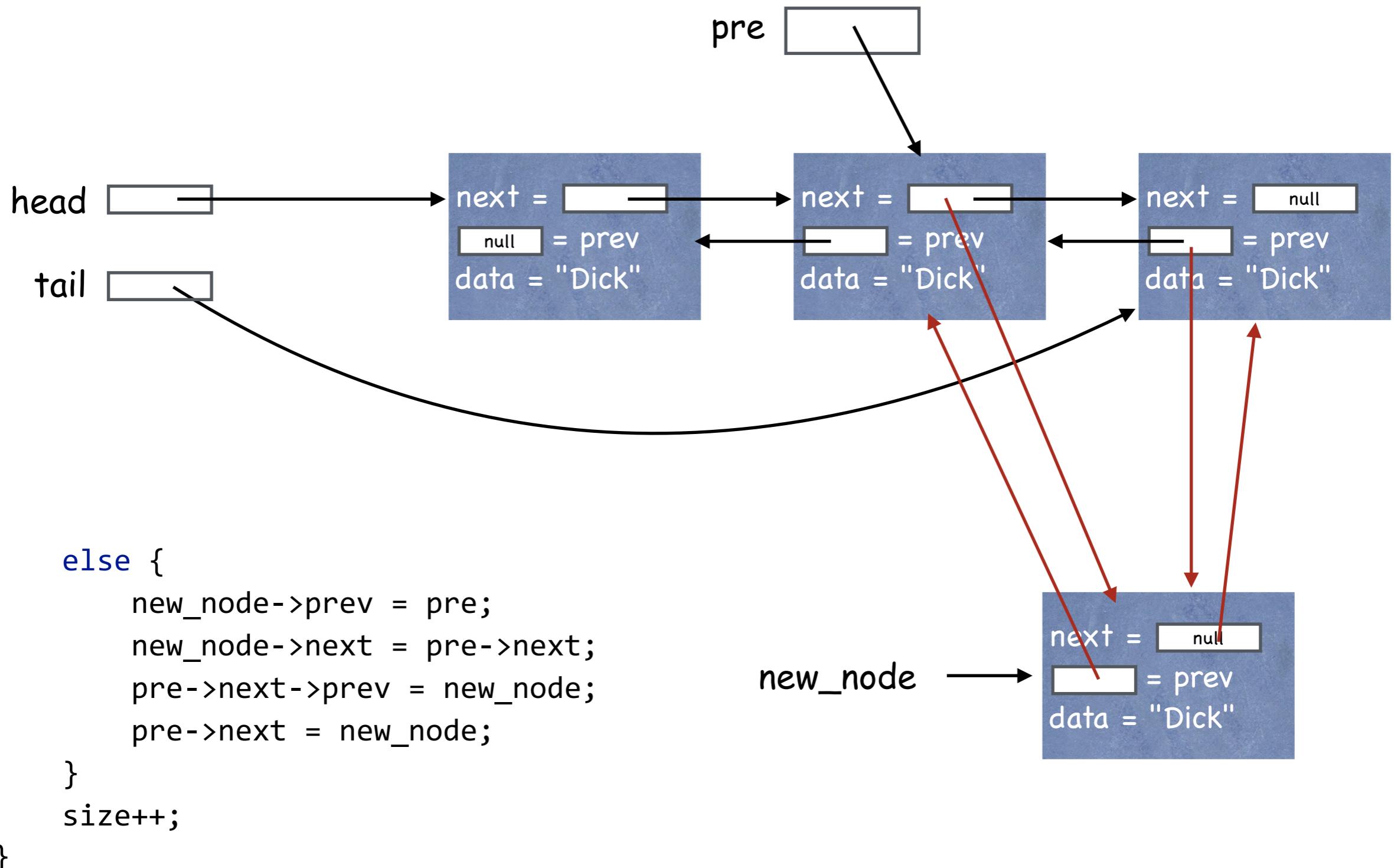
add_after(Node *pre, char *item) - at the head



add_after(Node *pre, char *item) - at the tail



add_after(Node *pre, char *item) - in the middle



remove(Node *p)

☞ **4가지 경우로 나누어서 처리한다.**

1. p가 유일한 노드인 경우
2. p가 head인 경우
3. p가 tail인 경우
4. 그밖의 일반적인 경우

add_ordered_list(char *item)

```
void add_ordered_list(char *item) {  
    Node *p = tail;  
    while (p!=NULL && strcmp(item, p->data) < 0)  
        p = p->prev;  
    add_after(p, item);  
}
```

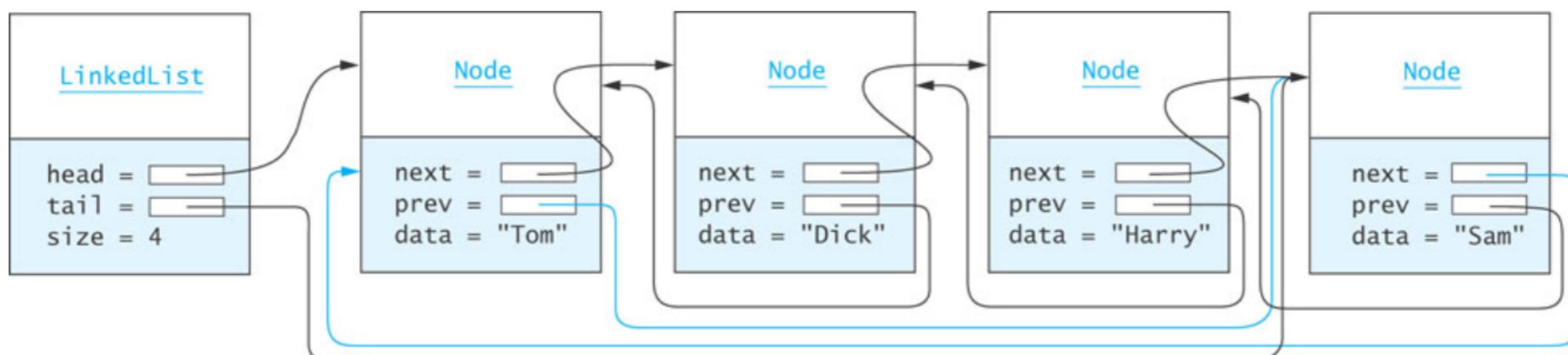
원형(circular) 리스트

원형 연결리스트

- 마지막 노드의 다음 노드가 첫번째 노드가 되는 단순 연결리스트

원형 이중연결 리스트:

- 마지막 노드의 다음 노드가 첫번째 노드가 되고
- 첫 노드의 이전 노드가 마지막 노드가 됨



MP3 관리 프로그램

MP3 Management Program

실행 예

Data file name ? my_collection.txt

프로그램을 실행하면 어떤 데이터 파일을 load할지 물어본다. 그냥 Enter를 치면 load하지 않는다.

Data file loaded.

\$ status



저장된 모든 노래의 번호, 가수, 제목, 파일의 경로명을 이렇게 출력한다.

2: AOA, 심쿵해, C:\\Music\\AOA\\심쿵해.mp3

1: BIGBANG, BAE BAE, C:\\Music\\BIGBANG\\BAE BAE.mp3

5: BIGBANG, LOSER, C:\\Music\\BIGBANG\\LOSER.mp3

6: BIGBANG, 맨정신, C:\\Music\\BIGBANG\\맨정신.mp3

4: 아이유, 무릎, C:\\Music\\아이유\\무릎.mp3

3: 아이유, 새 신발, C:\\Music\\아이유\\새 신발.mp3

가수 이름에 대해서 알파벳 순으로, 그리고 노래 제목에 대해서도 알파벳 순으로 출력된다.

\$ add

Artist: 아이유

가수 이름과 노래 제목은 2단어 이상일 수 있다.

Title: Twenty three



File: C:\\Music\\아이유\\twenty-three.mp3



파일은 지정하지 않아도 상관없다.

\$ search

Artist: AOA

Title: 심쿵해



가수 이름과 노래 제목으로 검색한다.

Found:

2: AOA, 심쿵해, C:\\Music\\AOA\\심쿵해.mp3

\$ search

Artist: 아이유



제목 없이 가수 이름만으로 검색한다.

Title:

Found:

7: 아이유, Twenty three, C:\\Music\\아이유\\twenty-three.mp3

4: 아이유, 무릎, C:\\Music\\아이유\\무릎.mp3

3: 아이유, 새 신발, C:\\Music\\아이유\\새 신발.mp3

실행 예

\$ play 4 ← 4번 노래를 play한다.

아이유, 무릎 is now playing.

\$ remove 6 ← 6번 노래를 목록에서 삭제한다.

6: BIGBANG, 맨정신 is deleted from the list.

\$ save as my_collection.txt ← 목록을 파일에 저장한다.

Saved.

\$ exit

파일 형식

존재하지 않는 항목의 경우 하나의 공백문자로 표시한다.

'#' 문자를 필드들 간의 구분자로 사용한다.

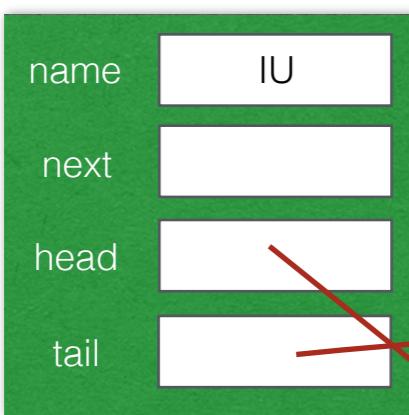
모든 라인은 반드시 구분자로 끝난다.

```
Beatles#Let it be#C:\\song_collections\\beatles\\let-it-be.mp3#
Girls Generation#GGG# #
IU#Twenty three#C:\\song_collections\\iu\\twenty-three.mp3#
```

한 줄에 한 개씩 저장한다.

자료구조

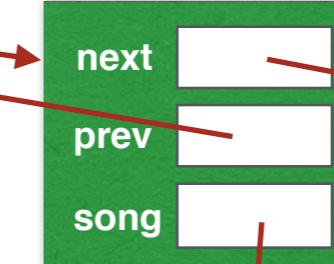
Artist



SNode



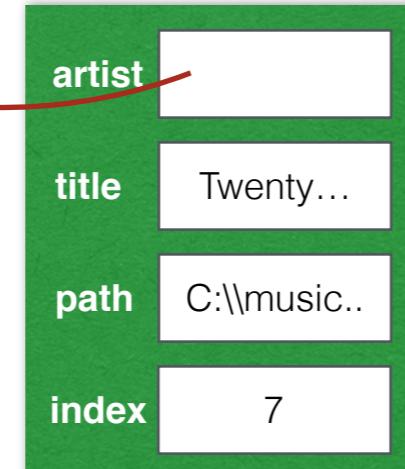
SNode



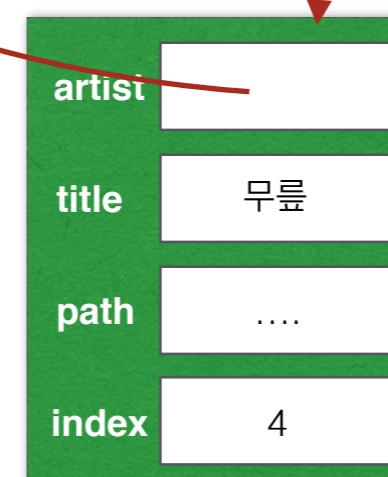
SNode



Song



Song

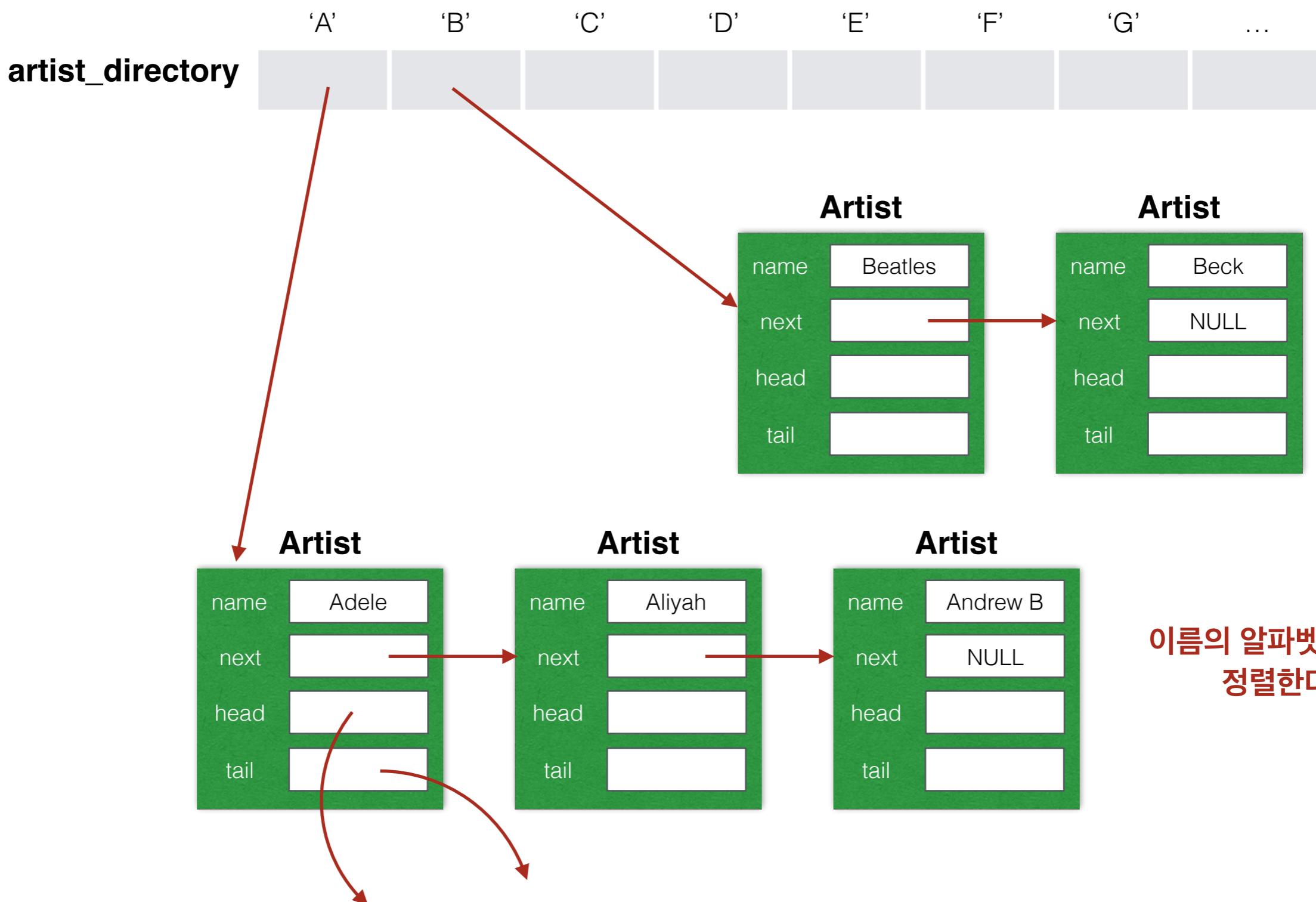


Song

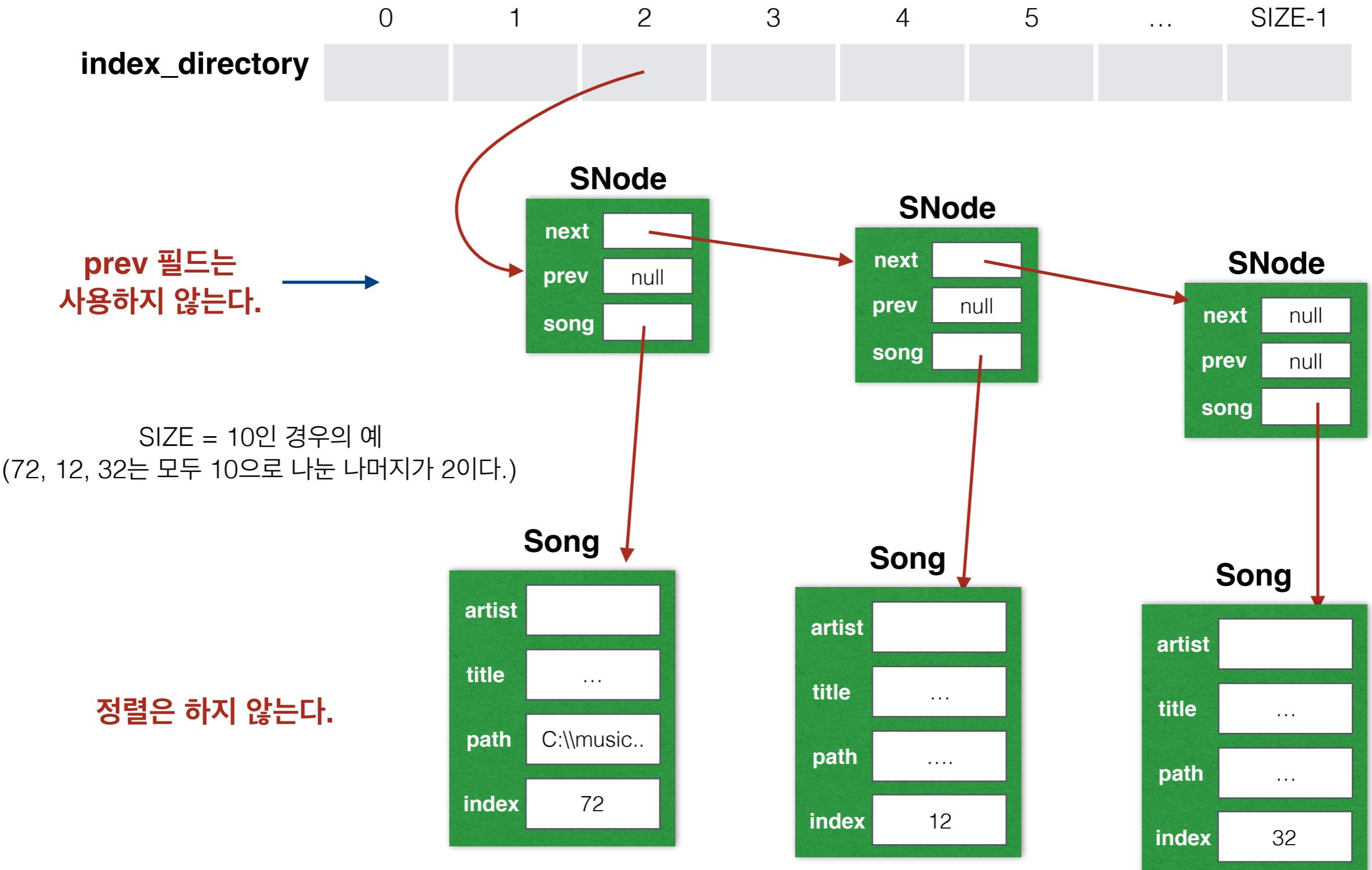


노래들은 이중 연결리스트에 제목
의 알파벳 순으로 저장한다.

artist들을 이니셜에 따라 분류해서
각 그룹을 하나의 단방향 연결리스트로 저장한다.



song들을 “index를 SIZE로 나눈 나머지”가 동일한 것들끼리
분류하여 각 그룹을 하나의 단방향 연결리스트로 저장한다.



프로그램을 여러 개의 소스파일로 구성하기

- ⦿ C 프로그램은 여러 개의 소스 파일들로 구성된다.
- ⦿ 관습적으로 각각의 소스파일은 확장자 .c를 가진다.
- ⦿ 하나의 소스파일은 **main**이라는 이름의 함수를 가져야 한다.
- ⦿ 프로그램을 여러 개의 소스 파일로 분할할 경우 장점:
 - ⦿ 서로 연관된 함수들과 변수들이 하나의 파일에 있으므로 프로그램의 구조가 좀더 알기 쉽고 명료해진다 (modularity).
 - ⦿ 각각의 소스 파일들을 개별적으로 컴파일 할 수 있으므로 컴파일 시간이 절약된다.
 - ⦿ 소프트웨어 재사용이 용이하다.

소스파일 구성

● 서로 연관된 함수들과 변수들을 하나의 파일에 넣는다.

1. 입력을 라인단위로 읽거나 문자열이나 토큰의 처리와 관련된 함수들을 하나의 소스 파일 `string_tools.c`에 넣는다.
2. 음원 데이터를 추가, 검색, 삭제, 관리하는 일을 하는 함수들을 다른 하나의 소스 파일 `library.c`에 넣는다.
3. `main`함수와 사용자의 명령어를 처리하는 `process_command`함수 등은 또 다른 하나의 파일 `main.c`에 둔다.

`string_tools.c`

```
int read_line()
{ ... }
int compose_name()
{...}
```

`library.c`

```
struct song { ... };
int add_song()
{ ... }
void find_song()
{...}
int remove_song()
{...}
```

`main.c`

```
char *buffer[100];
int main()
{...}
void
process_command()
{...}
handle_add()
{...}
```

여러 개의 소스 파일로 분할할 경우 해결해야 할 문제점

- 어떻게 다른 파일에 정의되어 있는 함수를 호출할 수 있는가?
- 어떻게 다른 파일에 정의되어 있는 변수(**external variable**)를 사용할 것인가?
- 어떻게 서로 다른 파일들이 매크로나 타입 정의를 공유할 것인가?

Header File과 include 지시어



매크로와 타입정의의 공유

다른 소스 파일과 공유할
매크로와 타입정의를
이렇게 해더파일에 넣는다.

library.h

```
#define MAX 100
typedef struct song Song;
struct song {
    ...
};

void add_song();
void find_song();
int remove_song();
```

library.c

```
#include "library.h"

void add_song()
{ ... }
void find_song()
{...}
int remove_song()
{...}
...
```

library.h를 include한 파일
에서는 매크로 MAX와 구조
체 Song을 사용 할 수 있다.

main.c

```
#include "library.h"

void process_command()
{
    ...
    add_song(...);
    ...
}
```

함수의 공유

library.h

```
#define MAX 100
typedef struct song Song;
struct song {
    ...
};

void add_song();
void find_song();
int remove_song();
```

다른 소스 파일과 공유할 함수는 이렇게 해더파일에는 prototype을 넣고, 실제 구현은 소스파일에 넣는다.

library.c

```
#include "library.h"

void add_song()
{ ... }

void find_song()
{...}
int remove_song()
{ }
```

library.h를 include한 파일에서는 함수 add_song을 사용할 수 있다.

main.c

```
#include "library.h"

void process_command()
{
    ...
    add_song(...);
    ...
}
```

서로 다른 파일 간의 변수의 공유

변수의 선언(**declaration**)과 정의(**definition**)의 구분

- 선언: 컴파일러에게 변수의 존재를 알려 줌
 - 정의: 실제로 메모리를 할당
- 변수는 여러 번 선언될 수 있다. 하지만 정의는 한번만 해야 한다.
- 선언과 정의를 동시에 하는 방법:

```
int i;
```

- 키워드 **extern**을 이용하여 변수를 정의하지 않고 선언만 할 수 있음. 즉, 컴파일러에게 변수 **i**와 배열 **a**가 다른 파일에 정의되어 있음을 알려주는 역할

```
extern int i;
```

```
extern int a[];
```

서로 다른 파일 간의 변수의 공유

- 공유 변수의 선언은 해더 파일에 둔다.
- 공유 변수를 사용하는 모든 소스 파일은 해더 파일을 **include**한다.
- 소스 파일 중 오직 한 곳에서 공유 변수를 정의한다.

file.h

```
extern int global_variable; /* Declaration of the variable */
```

file.c

```
#include "file.h"
```

```
int global_variable; /* Definition of the variable */
```

```
...
```

```
printf("%d\n", global_variable++); /* Use the variable */
```

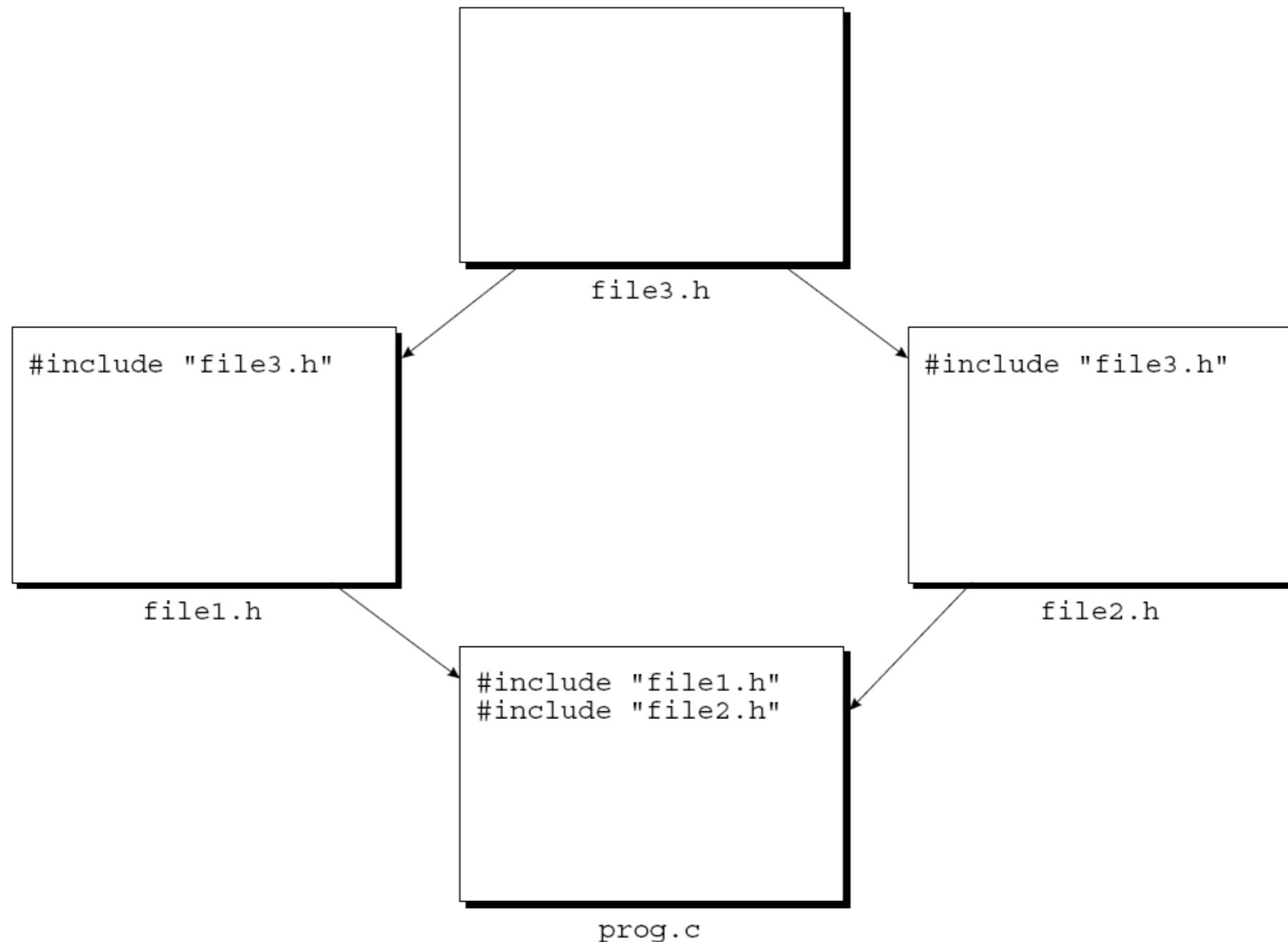
file2.c

```
#include "file.h"
```

```
...
```

```
printf("%d\n", global_variable++); /* Use the variable */
```

중첩된 Include



중복된 해더 파일

- 중복된 해더 파일이 항상 오류인 것인 아니다.
- 매크로 정의, 함수 프로토타입, 그리고 외부(**extern**) 변수의 선언은 여러 번 중복되어도 상관 없다.
- 하지만 타입 정의가 중복되는 것은 컴파일러 오류를 야기한다.

해더 파일의 중복 방지

- 중복된 해더 파일 문제를 해결하기 위해 **#ifndef - #endif** 지시어를 이용한다.
- 가령 **boolean.h** 파일이 중복 **include**되는 것을 방지하기 위해서

```
#ifndef BOOLEAN_H
#define BOOLEAN_H

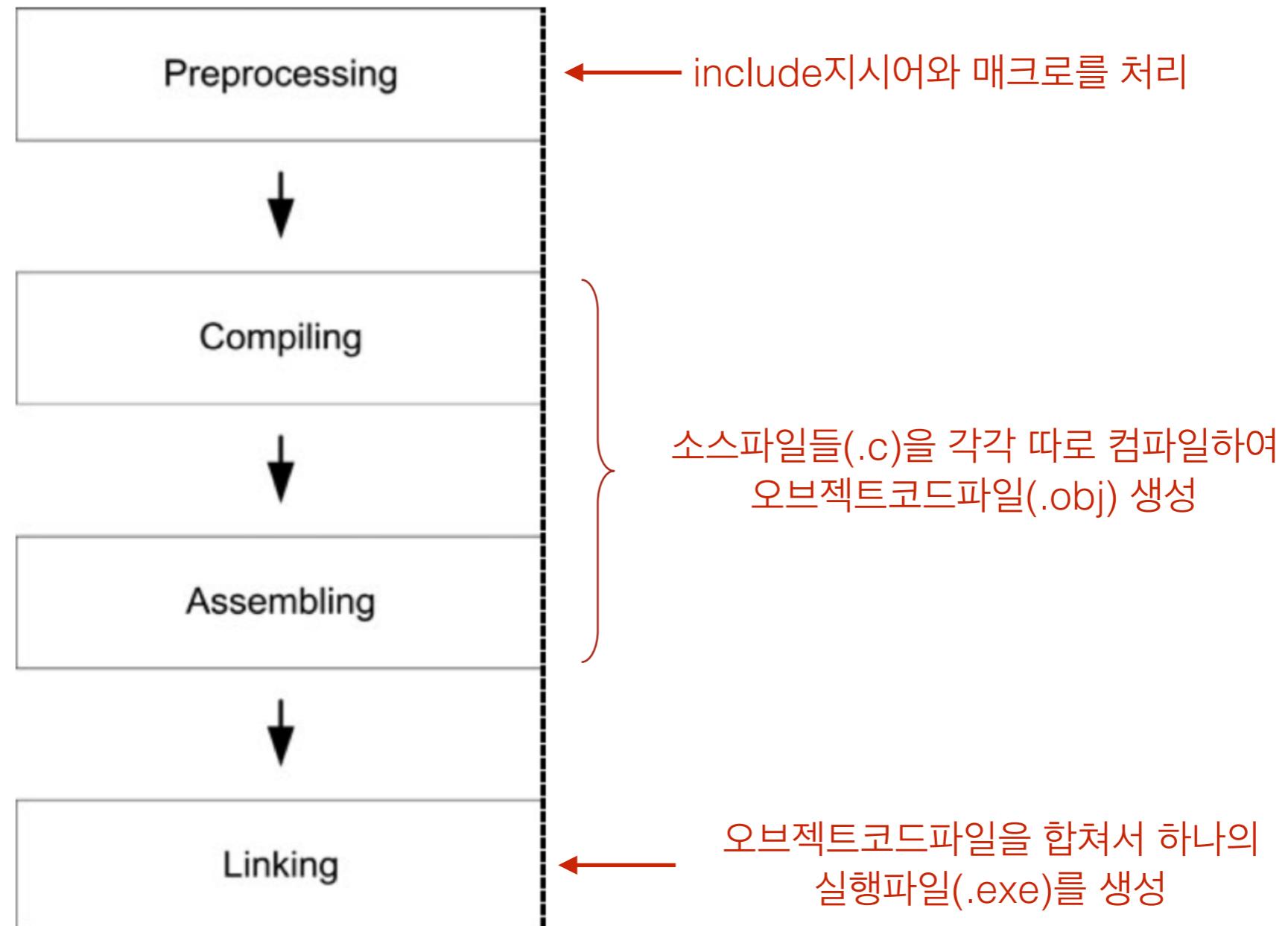
#define TRUE 1
#define FALSE 0
typedef int Bool;
#endif
```

Music Library Program

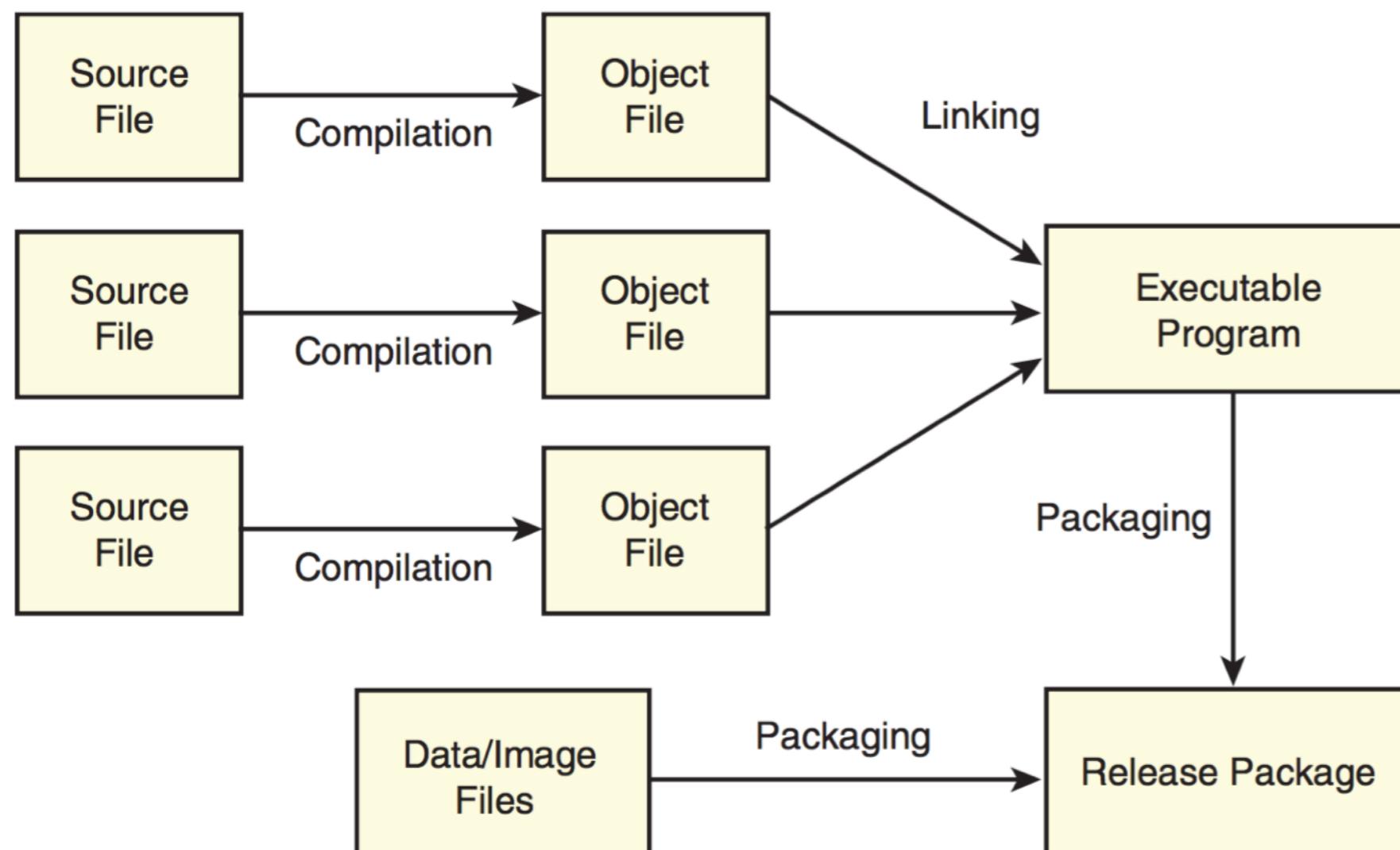
Watch videos !!

Q&A

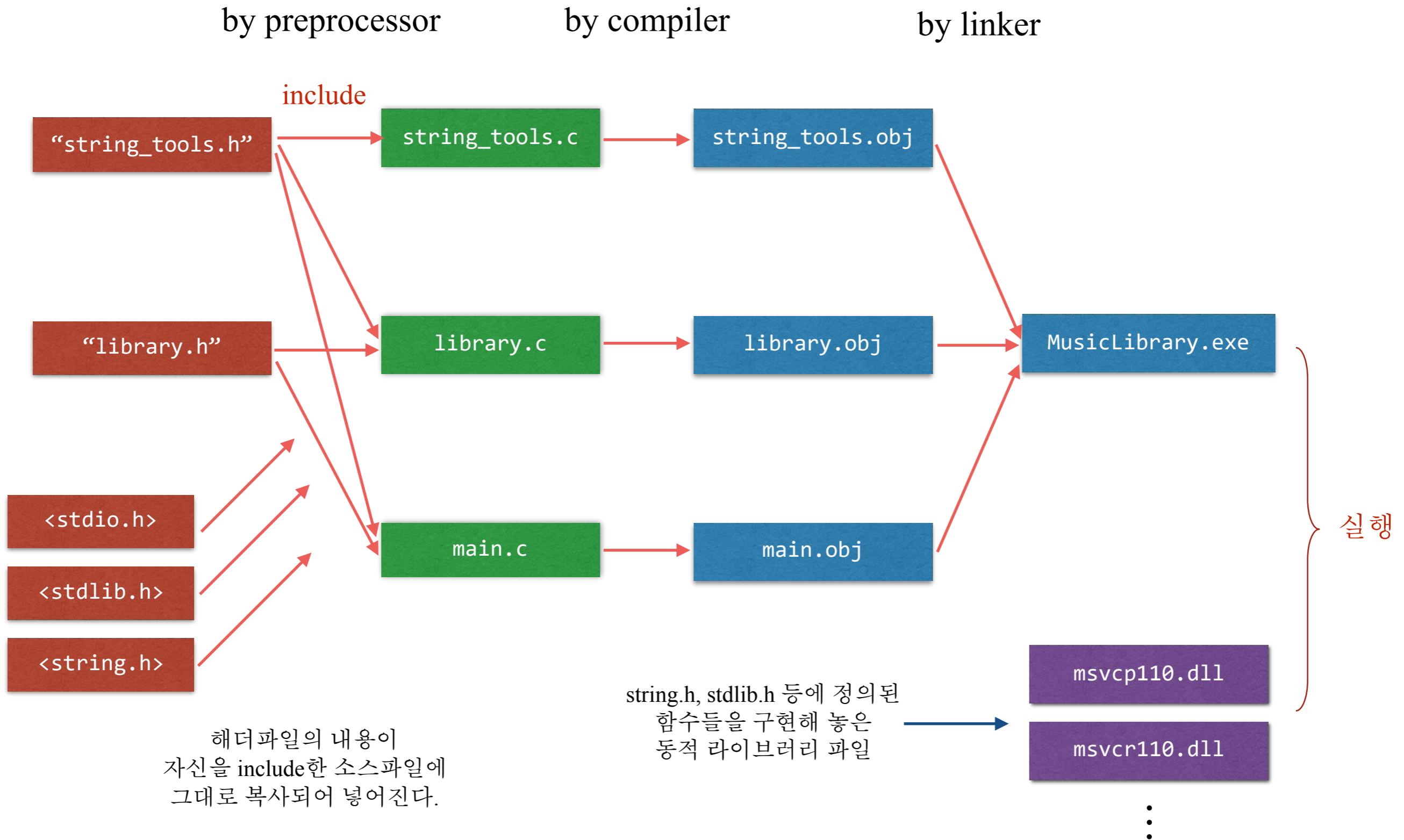
빌드 과정



빌드 과정



빌드 과정



Exploring the project directory

실행파일(.exe)이다.

Name	Date modified	Type	Size
MusicLibrary.exe	2015-11-04 오전 1...	Application	43 KB
MusicLibrary.ilk	2015-11-04 오전 1...	Incremental Linker File	394 KB
MusicLibrary.pdb	2015-11-04 오전 1...	Program Debug Database	804 KB

```
Command Prompt - MusicLibrary.exe
C:\Users\Wohheum\Dropbox\USProjects\MusicLibrary>cd De
'cdDe' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\Wohheum\Dropbox\USProjects\MusicLibrary>cd Debug

C:\Users\Wohheum\Dropbox\USProjects\MusicLibrary\Debug>dir
Volume in drive C is BOOTCAMP
Volume Serial Number is 50AD-907A

Directory of C:\Users\Wohheum\Dropbox\USProjects\MusicLibrary\Debug

2015-11-05  오전 10:24    <DIR>      .
2015-11-05  오전 10:24    <DIR>      ..
2015-11-05  오전 10:24           46,080 MusicLibrary.exe
2015-11-05  오전 10:24           336,520 MusicLibrary.ilk
2015-11-05  오전 10:24           659,456 MusicLibrary.pdb
               3 File(s)     1,042,056 bytes
               2 Dir(s)   95,135,068,160 bytes free

C:\Users\Wohheum\Dropbox\USProjects\MusicLibrary\Debug>MusicLibrary.exe
Data file name ? ../MusicLibrary/list.txt
$
```

실행파일을
command 창에서
실행하였다.

Exploring the project directory

> MusicLibrary > MusicLibrary >

Name	Date modified	Type	Size
Debug	2015-11-04 오전 1...	File folder	
library.cpp	2015-11-04 오전 1...	C++ Source	10 KB
library.h	2015-11-04 오전 1...	C/C++ Header	1 KB
list.txt	2015-10-29 오전 1...	Text Document	7 KB
main.cpp	2015-11-04 오전 1...	C++ Source	3 KB
MusicLibrary.vcxproj	2015-11-04 오전 9:...	VC++ Project	7 KB
MusicLibrary.vcxproj.filters	2015-11-04 오전 9:...	FILTERS File	2 KB
string_tools.cpp	2015-10-28 오후 3:...	C++ Source	1 KB
string_tools.h	2015-10-28 오후 3:...	C/C++ Header	1 KB
tmpdata.txt	2015-11-04 오전 1...	Text Document	7 KB

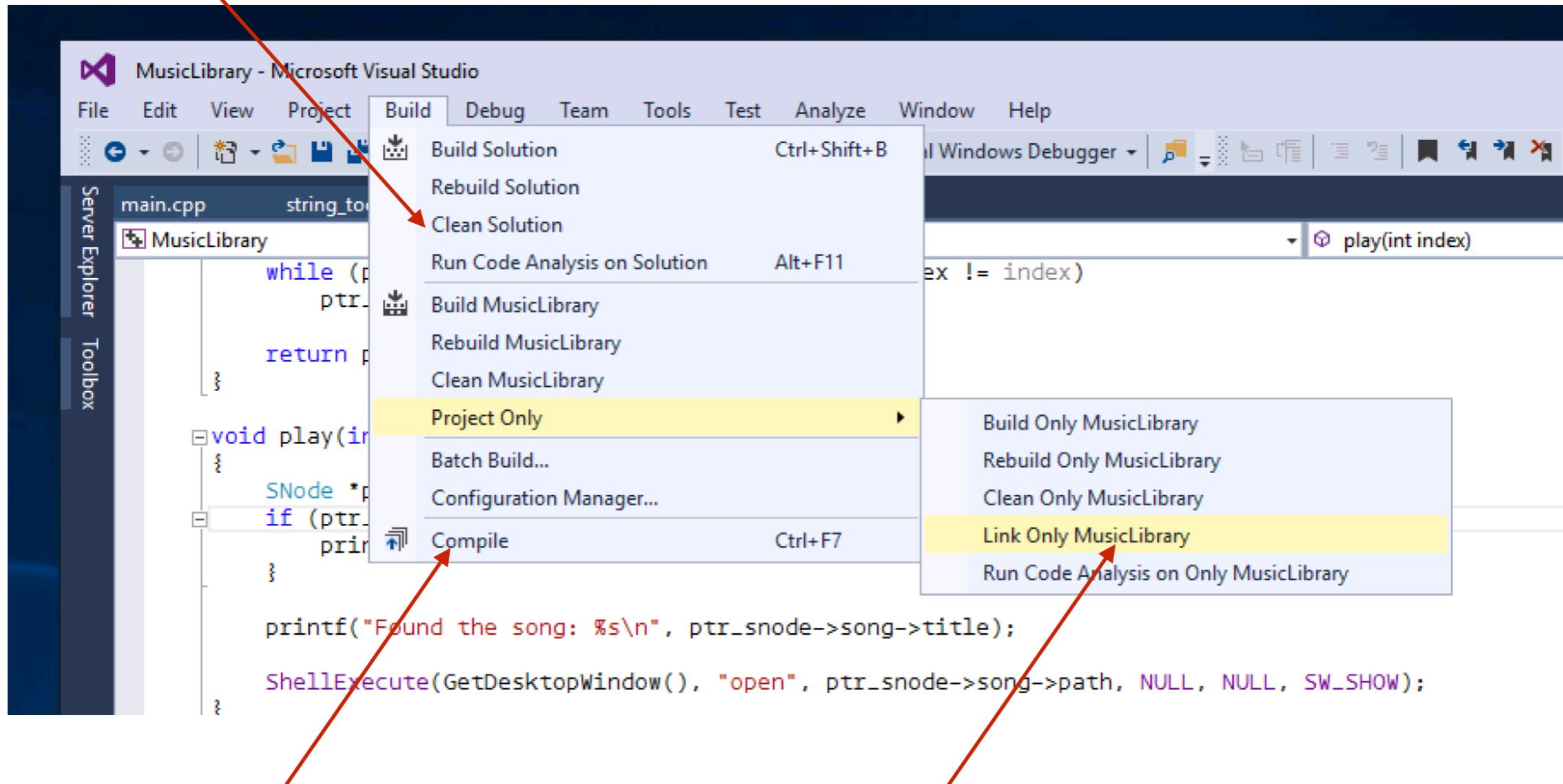
MusicLibrary > MusicLibrary > Debug >

Name	Date modified	Type	Size
MusicLibrary.tlog	2015-11-04 오전 1...	File folder	
library.obj	2015-11-04 오전 1...	Object File	49 KB
main.obj	2015-11-04 오전 1...	Object File	21 KB
MusicLibrary.log	2015-11-04 오전 1...	Text Document	1 KB
string_tools.obj	2015-10-28 오후 4:...	Object File	5 KB
vc140.idb	2015-11-04 오전 1...	VC++ Minimum Rebuild Dependency File	659 KB
vc140.pdb	2015-11-04 오전 1...	Program Debug Database	164 KB

3개의 .obj
파일이 있다.

각 단계를 따로 해보기

(1) clean solution을 한다.



(2) 각각의 소스파일을 따로따로 컴파일 한다.

(3) 링크한다.

각 단계마다 어떤 파일이
생성 되는지 살펴본다.

char, unsigned char, int

The screenshot shows a debugger interface with the following details:

Source.cpp (Global Scope) main()

```
#include <stdbool.h>
#include <stdio.h>
#include <string.h>

int main()
{
    char name[] = "喜";
    int len = strlen(name);
    int size = sizeof(name);

    int index = name[0];
    int index2 = (unsigned char)name[0];

    printf("%d %d", len, size); ≤1ms elapsed

    return 0;
}
```

Autos window:

Name	Value	Type
index	0xfffffff8	int
index2	0x000000c8	int
len	0x00000002	int
name	0x0116f9e8 "喜"	char[0x00000003]
[0x00000000]	0xc8 '?'	char
[0x00000001]	0xab '?'	char
[0x00000002]	0x00 '\0'	char
name[0]	0xc8 '?'	char
size	0x00000003	int

Bottom navigation bar: Autos, Locals, Watch 1

Ready