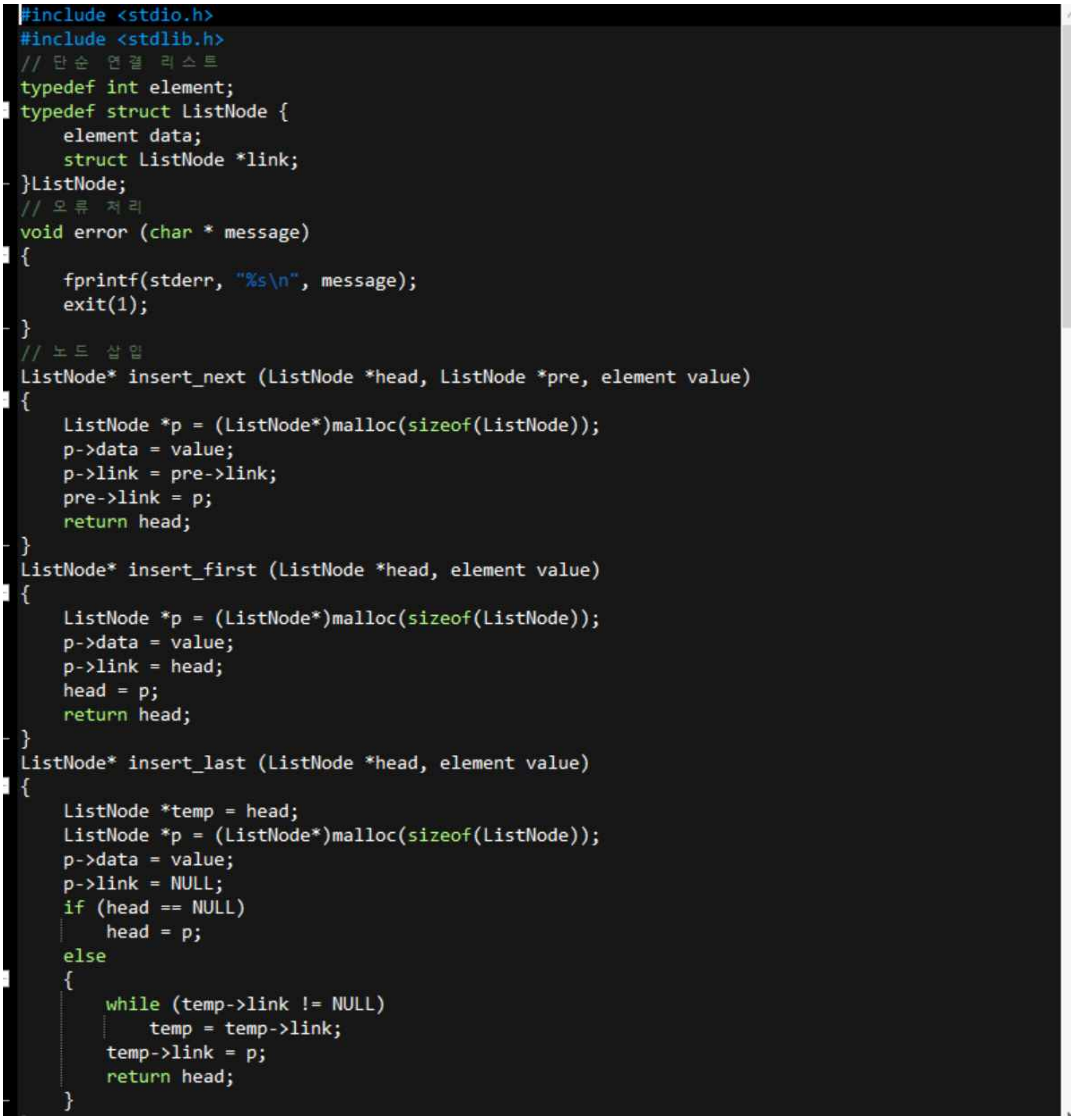
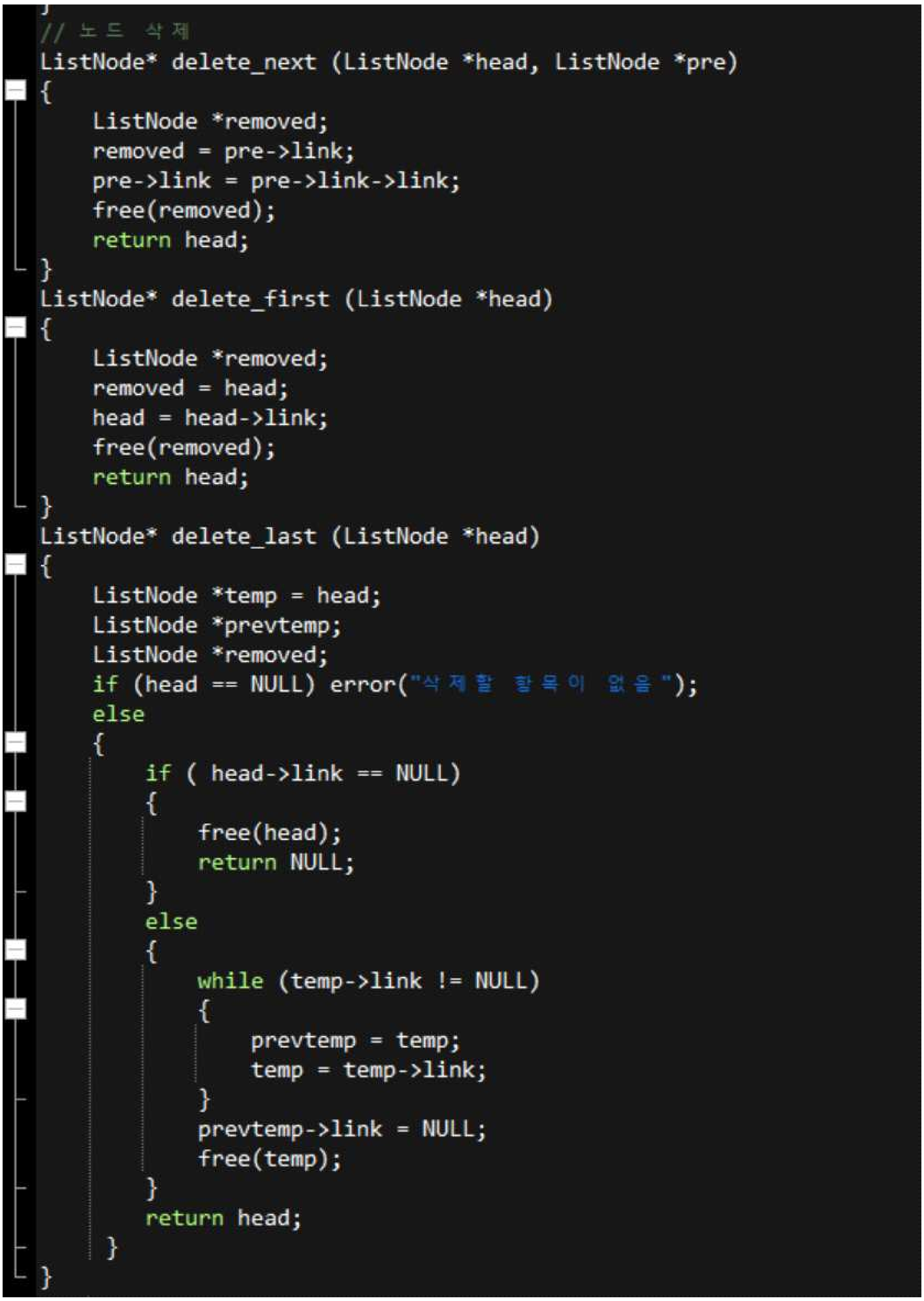
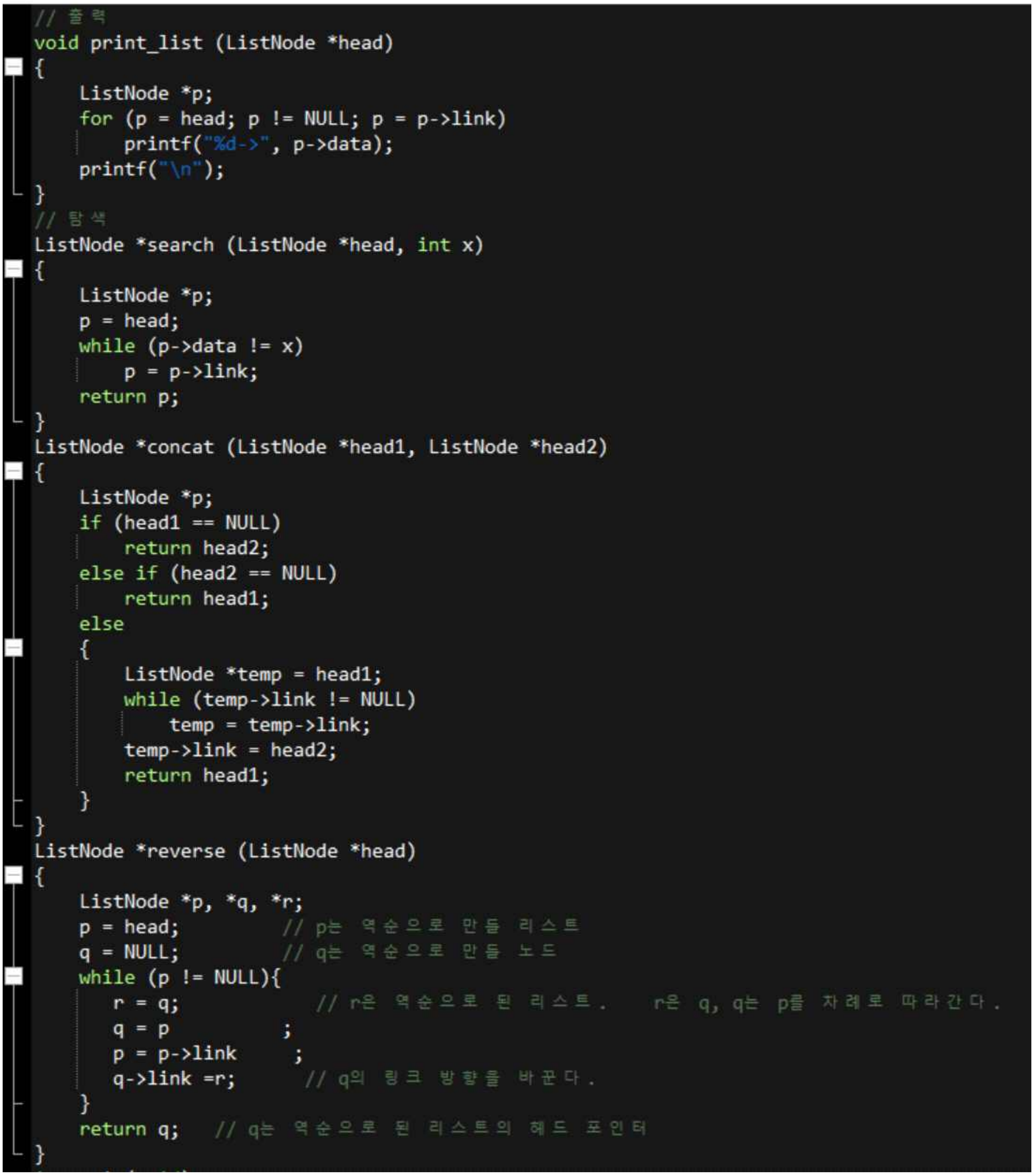
// 단순연결리스트



```c
#include <stdio.h>
#include <stdlib.h>
// 단순 연결 리스트
typedef int element;
typedef struct ListNode {
    element data;
    struct ListNode *link;
}ListNode;
// 오류 처리
void error (char * message)
{
    fprintf(stderr, "%s\n", message);
    exit(1);
}
// 노드 삽입
ListNode* insert_next (ListNode *head, ListNode *pre, element value)
{
    ListNode *p = (ListNode*)malloc(sizeof(ListNode));
    p->data = value;
    p->link = pre->link;
    pre->link = p;
    return head;
}
ListNode* insert_first (ListNode *head, element value)
{
    ListNode *p = (ListNode*)malloc(sizeof(ListNode));
    p->data = value;
    p->link = head;
    head = p;
    return head;
}
ListNode* insert_last (ListNode *head, element value)
{
    ListNode *temp = head;
    ListNode *p = (ListNode*)malloc(sizeof(ListNode));
    p->data = value;
    p->link = NULL;
    if (head == NULL)
        head = p;
    else
    {
        while (temp->link != NULL)
            temp = temp->link;
        temp->link = p;
        return head;
    }
```

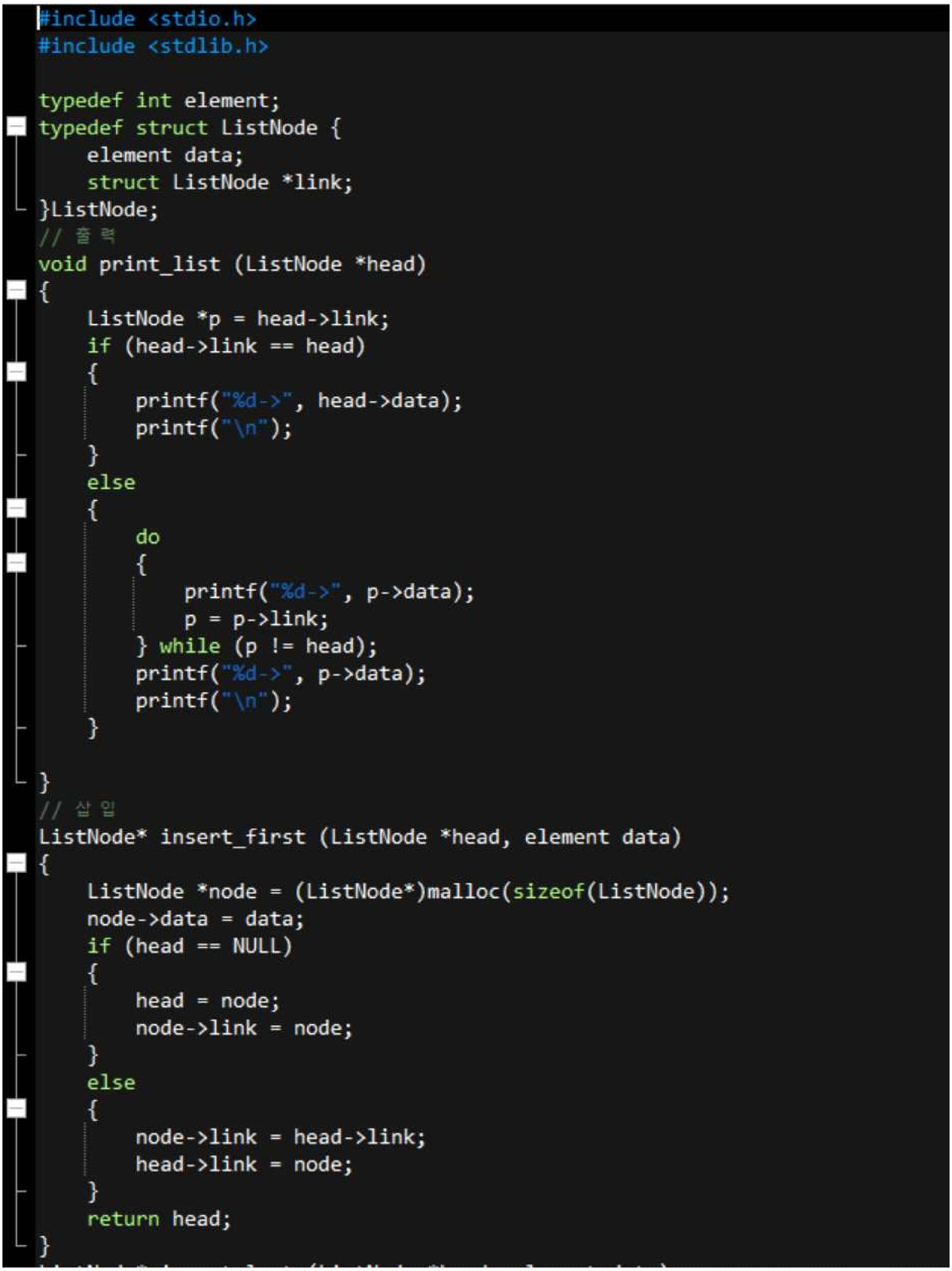


```c
}
// 노드 삭제
ListNode* delete_next (ListNode *head, ListNode *pre)
{
    ListNode *removed;
    removed = pre->link;
    pre->link = pre->link->link;
    free(removed);
    return head;
}
ListNode* delete_first (ListNode *head)
{
    ListNode *removed;
    removed = head;
    head = head->link;
    free(removed);
    return head;
}
ListNode* delete_last (ListNode *head)
{
    ListNode *temp = head;
    ListNode *prevtemp;
    ListNode *removed;
    if (head == NULL) error("삭제할 항목이 없음");
    else
    {
        if ( head->link == NULL)
        {
            free(head);
            return NULL;
        }
        else
        {
            while (temp->link != NULL)
            {
                prevtemp = temp;
                temp = temp->link;
            }
            prevtemp->link = NULL;
            free(temp);
        }
        return head;
     }
}
```

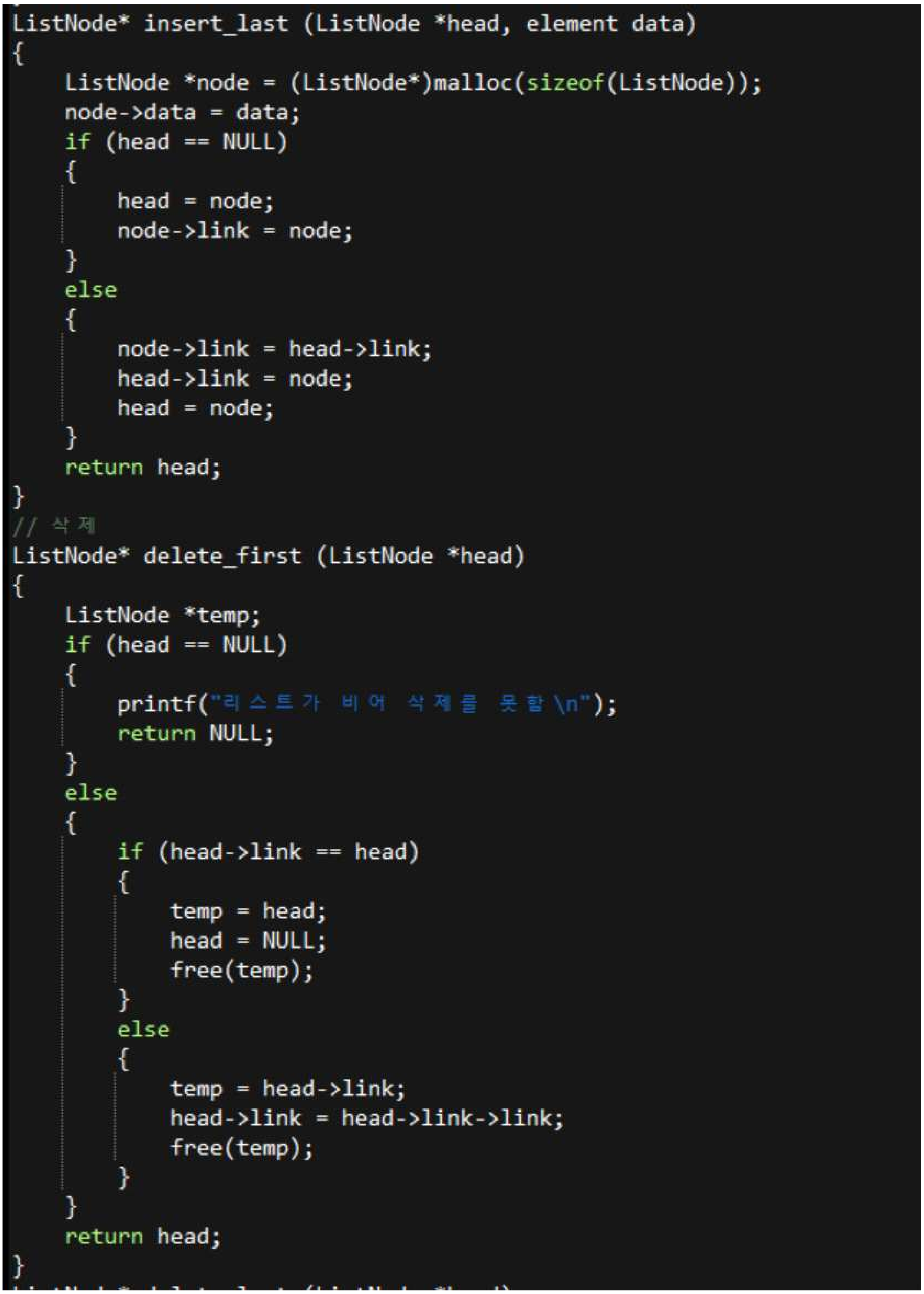


```c
// 출력
void print_list (ListNode *head)
{
    ListNode *p;
    for (p = head; p != NULL; p = p->link)
        printf("%d->", p->data);
    printf("\n");
}
// 탐색
ListNode *search (ListNode *head, int x)
{
    ListNode *p;
    p = head;
    while (p->data != x)
        p = p->link;
    return p;
}
ListNode *concat (ListNode *head1, ListNode *head2)
{
    ListNode *p;
    if (head1 == NULL)
        return head2;
    else if (head2 == NULL)
        return head1;
    else
    {
        ListNode *temp = head1;
        while (temp->link != NULL)
            temp = temp->link;
        temp->link = head2;
        return head1;
    }
}
ListNode *reverse (ListNode *head)
{
    ListNode *p, *q, *r;
    p = head;           // p는 역순으로 만들 리스트
    q = NULL;           // q는 역순으로 만들 노드
    while (p != NULL){
       r = q;             // r은 역순으로 된 리스트.   r은 q, q는 p를 차례로 따라간다.
       q = p         ;
       p = p->link     ;
       q->link =r;      // q의 링크 방향을 바꾼다.
    }
    return q;   // q는 역순으로 된 리스트의 헤드 포인터
}
```

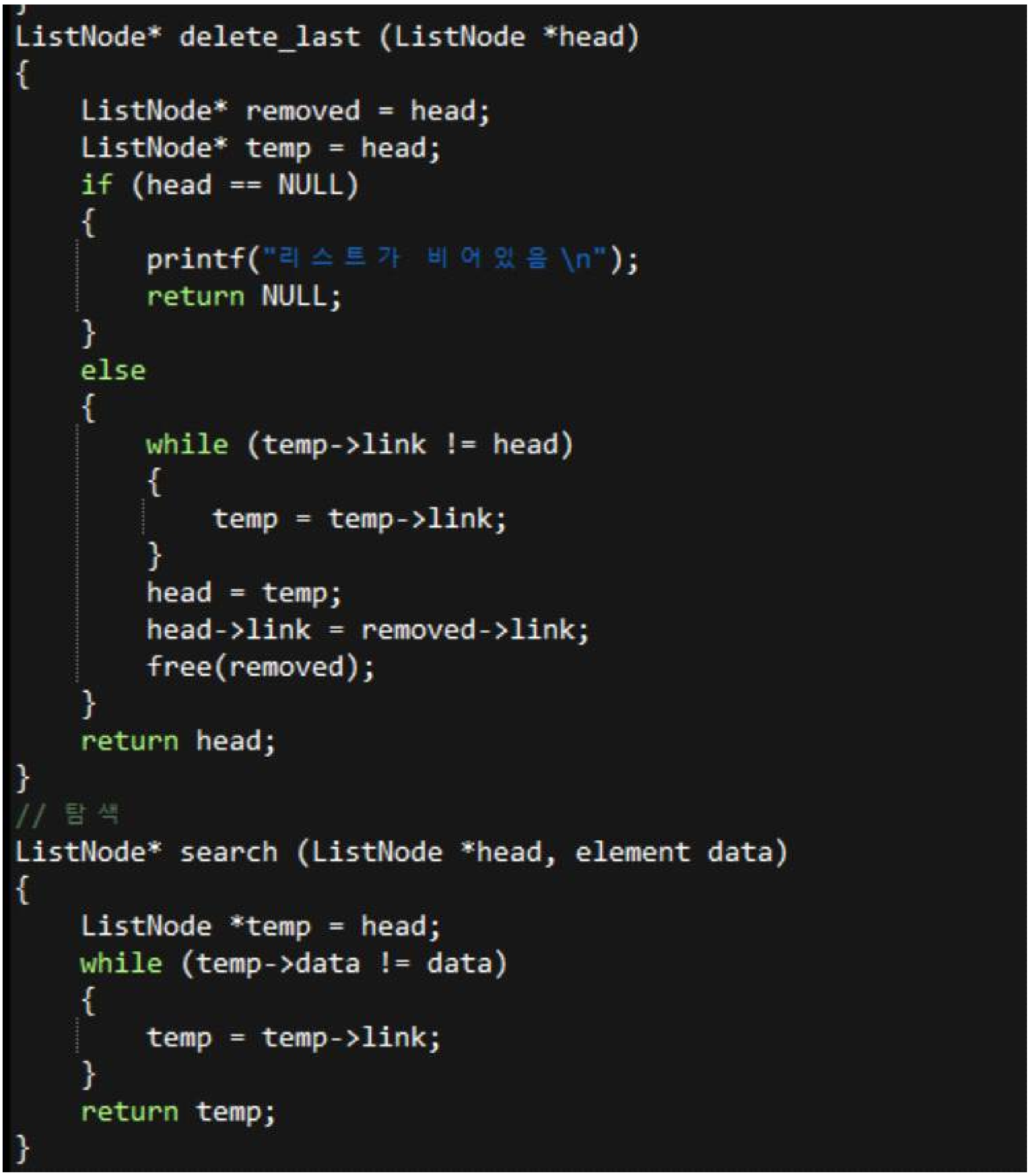
// 원형 연결 리스트



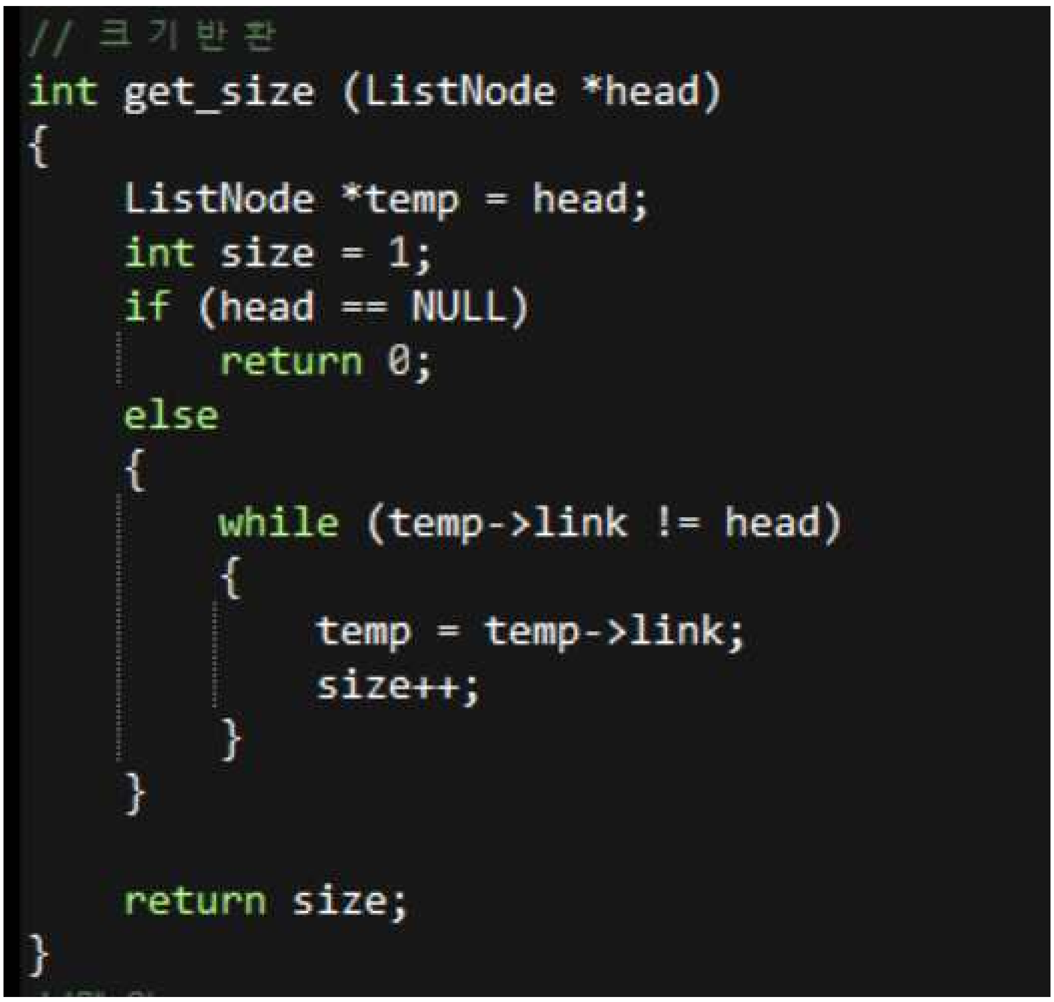
```c
#include <stdio.h>
#include <stdlib.h>

typedef int element;
typedef struct ListNode {
    element data;
    struct ListNode *link;
}ListNode;
// 출력
void print_list (ListNode *head)
{
    ListNode *p = head->link;
    if (head->link == head)
    {
        printf("%d->", head->data);
        printf("\n");
    }
    else
    {
        do
        {
            printf("%d->", p->data);
            p = p->link;
        } while (p != head);
        printf("%d->", p->data);
        printf("\n");
    }

}
// 삽입
ListNode* insert_first (ListNode *head, element data)
{
    ListNode *node = (ListNode*)malloc(sizeof(ListNode));
    node->data = data;
    if (head == NULL)
    {
        head = node;
        node->link = node;
    }
    else
    {
        node->link = head->link;
        head->link = node;
    }
    return head;
}
```



```c
ListNode* insert_last (ListNode *head, element data)
{
    ListNode *node = (ListNode*)malloc(sizeof(ListNode));
    node->data = data;
    if (head == NULL)
    {
        head = node;
        node->link = node;
    }
    else
    {
        node->link = head->link;
        head->link = node;
        head = node;
    }
    return head;
}
// 삭제
ListNode* delete_first (ListNode *head)
{
    ListNode *temp;
    if (head == NULL)
    {
        printf("리스트가 비어 삭제를 못함\n");
        return NULL;
    }
    else
    {
        if (head->link == head)
        {
            temp = head;
            head = NULL;
            free(temp);
        }
        else
        {
            temp = head->link;
            head->link = head->link->link;
            free(temp);
        }
    }
    return head;
}
```



```c
}
ListNode* delete_last (ListNode *head)
{
    ListNode* removed = head;
    ListNode* temp = head;
    if (head == NULL)
    {
        printf("리스트가 비어있음 \n");
        return NULL;
    }
    else
    {
        while (temp->link != head)
        {
            temp = temp->link;
        }
        head = temp;
        head->link = removed->link;
        free(removed);
    }
    return head;
}
// 탐색
ListNode* search (ListNode *head, element data)
{
    ListNode *temp = head;
    while (temp->data != data)
    {
        temp = temp->link;
    }
    return temp;
}
```



```c
// 크기반환
int get_size (ListNode *head)
{
    ListNode *temp = head;
    int size = 1;
    if (head == NULL)
        return 0;
    else
    {
        while (temp->link != head)
        {
            temp = temp->link;
            size++;
        }
    }

    return size;
}
```