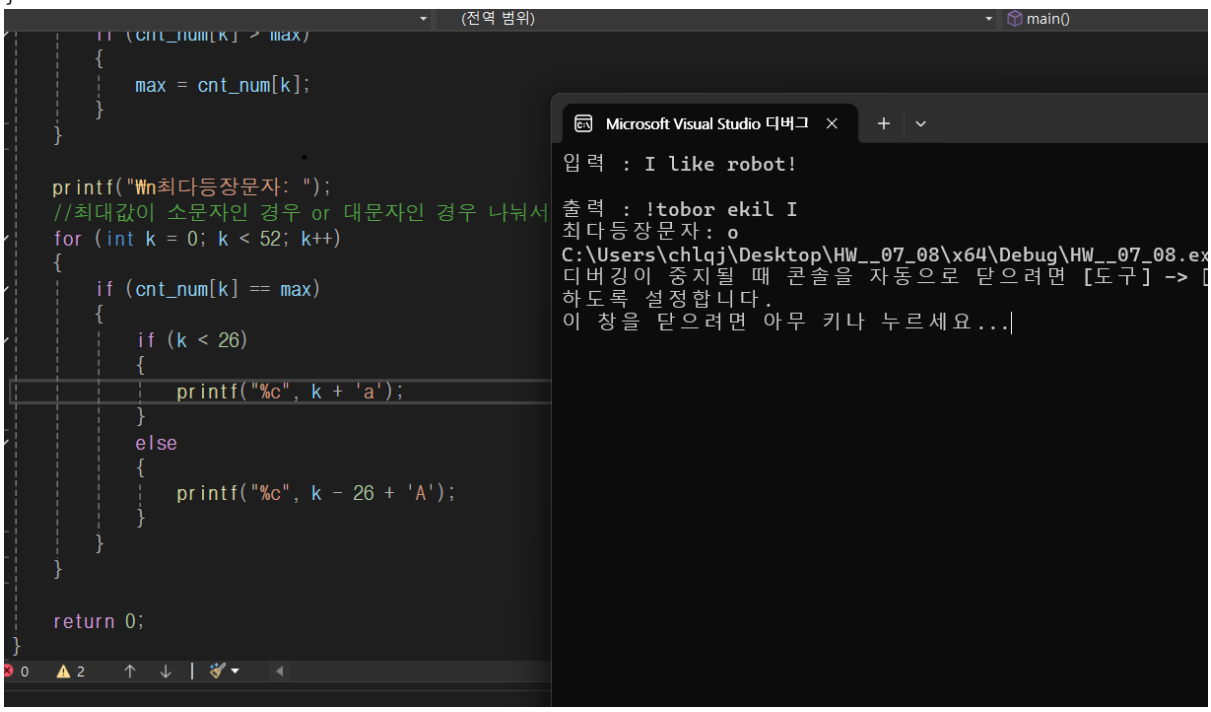
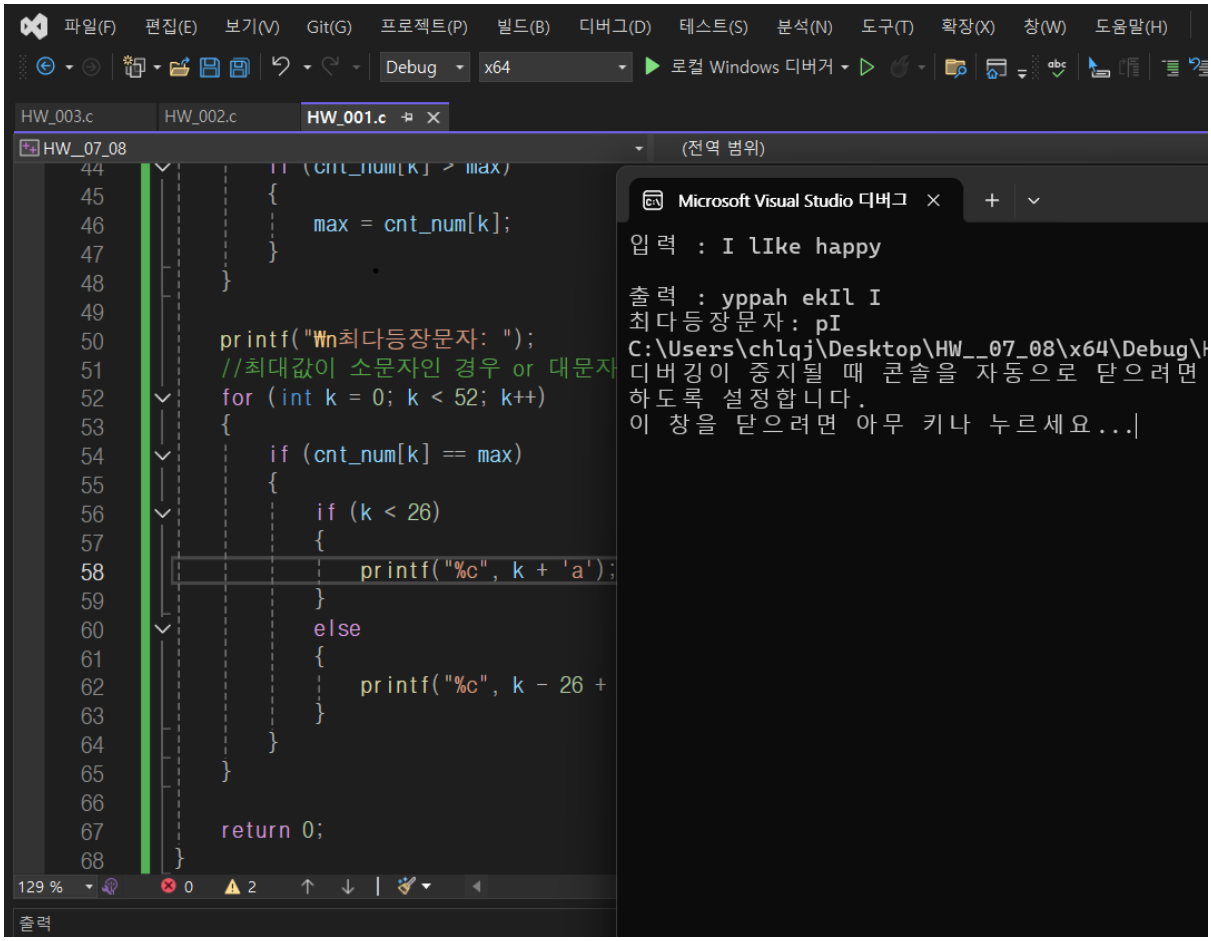
최범수 6일차 과제

1. HW_001

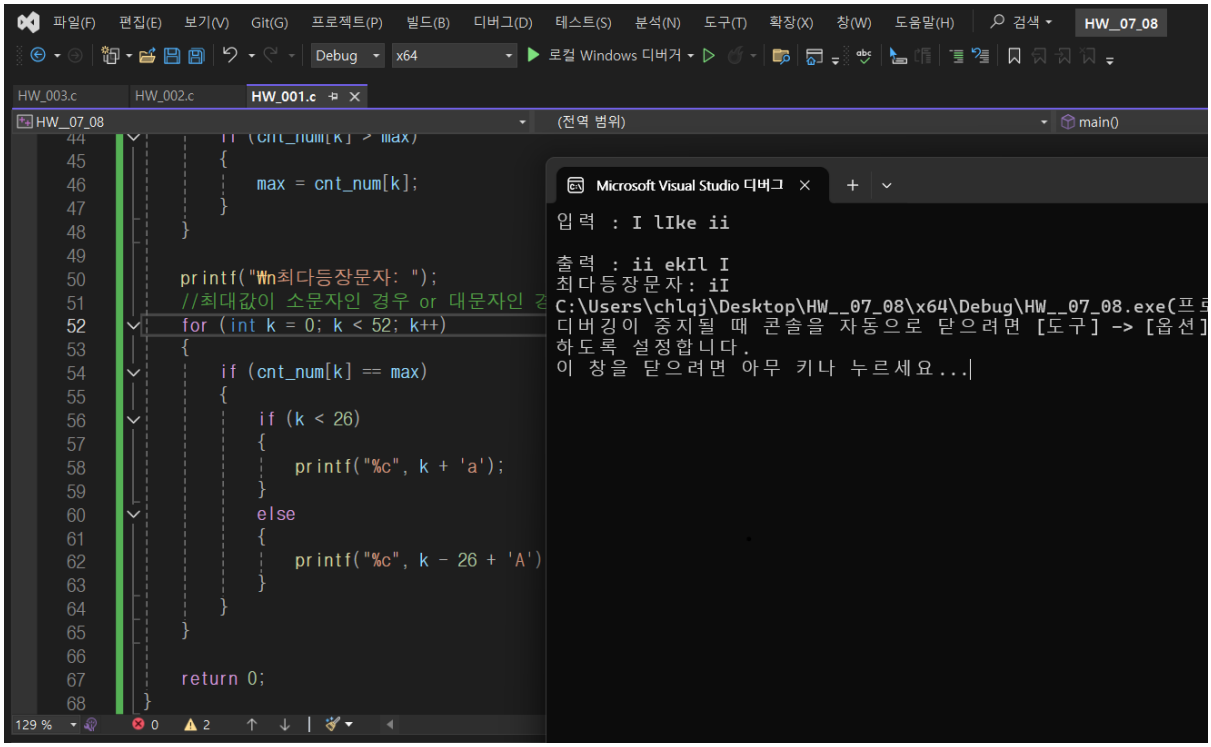
```
#define _CRT_SECURE_NO_WARNINGS

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

int main()
{
    //문장 입력받을 배열 생성
    char sentence[100];
    char* ps;
    ps = sentence;

    int max = 0;
    int cnt_num[52] = { 0, };//알파벳 대소문자 총 52개

    printf("입력 : ");
    //띄어쓰기 포함에서 문장 입력받는 기호
    scanf("%[^₩n]s", sentence);

    printf("₩n출력 : ");
    //배열의 크기에서 1만큼 작은것부터 0까지 출력
    for (int i = strlen(ps) - 1; i >= 0; --i)
    {
        printf("%c", ps[i]);
    }

    //소문자인 경우 0~26번째 cnt_num에 입력받기
    for (int i = 0; i < strlen(ps); i++)
    {
        if (ps[i] >= 'a' && ps[i] <= 'z')
        {
            cnt_num[ps[i] - 'a']++;
        }
        //대문자인 경우 27 ~ 52번째 cnt_num에 입력받기
        else if (ps[i] >= 'A' && ps[i] <= 'Z')
        {
            cnt_num[ps[i] - 'A' + 26]++;
        }
    }

    //0~52까지 대소문자 포함해서 가장 많이 나온 문자를 max에 저장하기
    for (int k = 0; k < 52; k++)
    {
        if (cnt_num[k] > max)
        {
            max = cnt_num[k];
        }
    }

    printf("₩n최다등장문자: ");
```

```
    //최대값이 소문자인 경우 or 대문자인 경우 나눠서 출력
    for (int k = 0; k < 52; k++)
    {
        if (cnt_num[k] == max)
        {
            if (k < 26)
            {
                printf("%c", k + 'a');
            }
            else
            {
                printf("%c", k - 26 + 'A');
            }
        }
    }

    return 0;
}
```





두 글자가 중복으로 가장 많이 나오면 두 문자 p, I모두 출력되게 구현

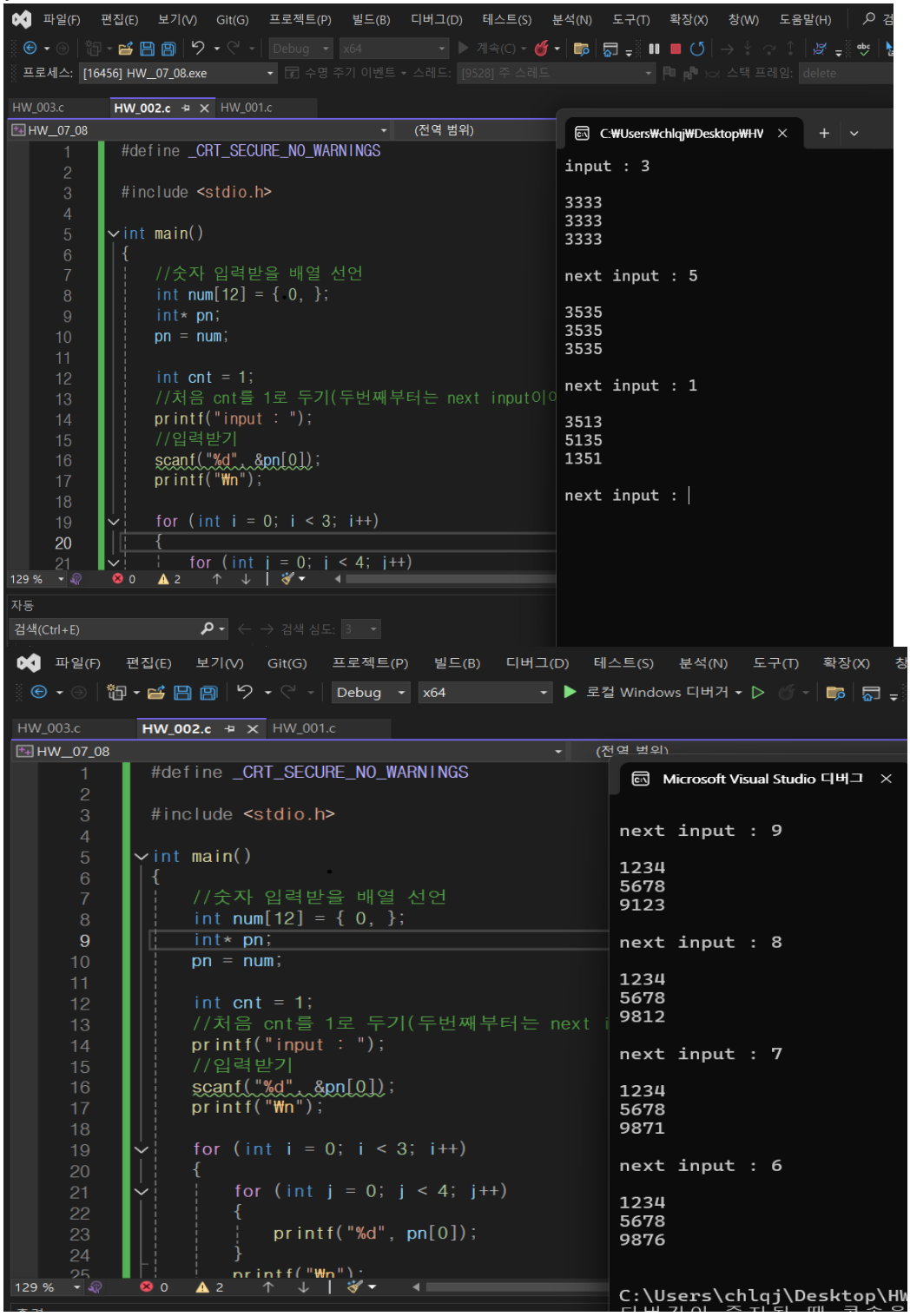


대소문자 구분해서 i와 I 둘다 출력

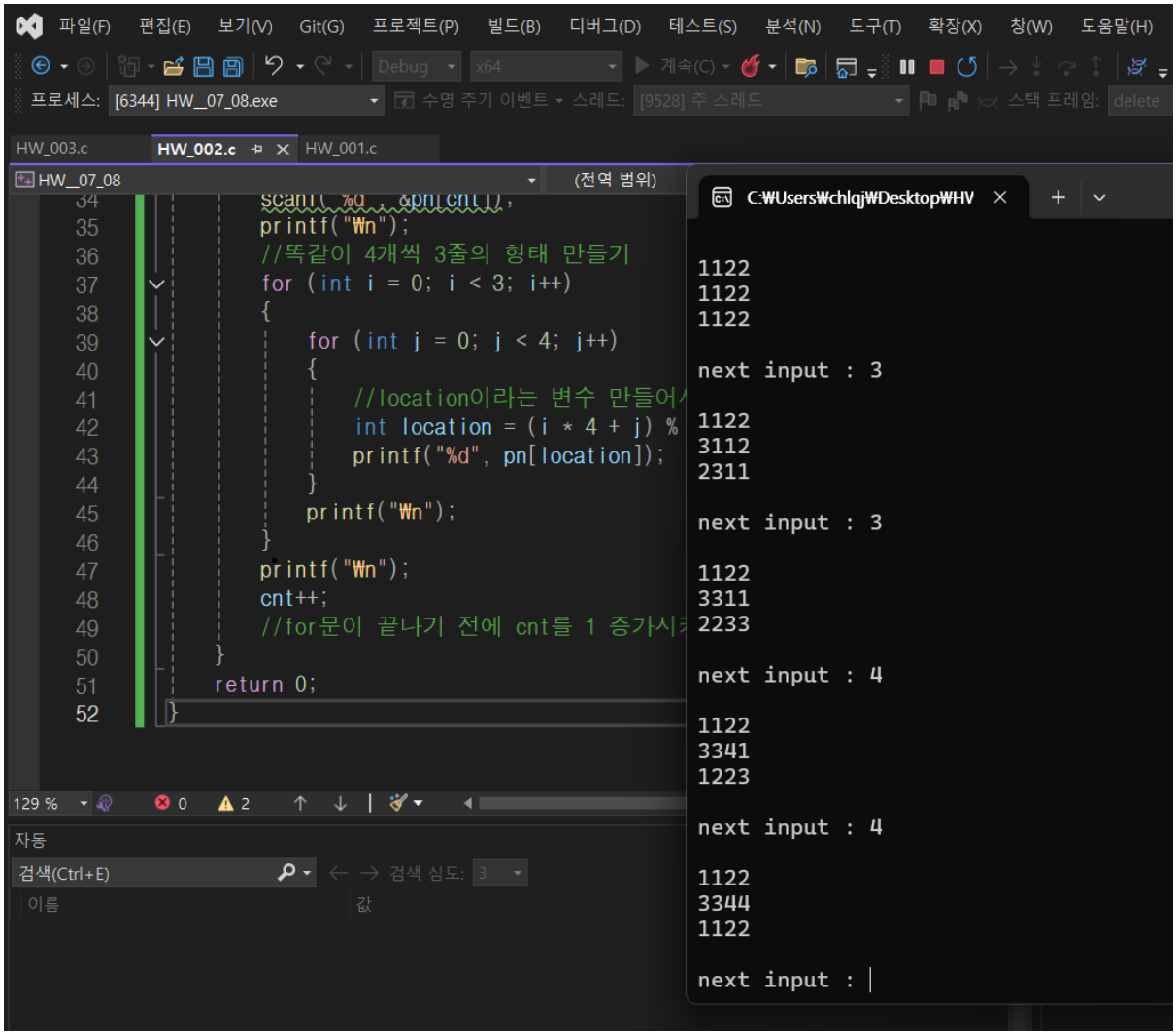
2. HW_002

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int main()
{
    //숫자 입력받을 배열 선언
    int num[12] = { 0, };
    int* pn;
    pn = num;

    int cnt = 1;
    //처음 cnt를 1로 두기(두번째부터는 next input이여서 처음한번은 그냥 출력하고 나머지를
반복문 안에 넣었습니다.)
    printf("input : ");
    //입력받기
    scanf("%d", &pn[0]);
    printf("₩n");

    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            printf("%d", pn[0]);
        }
        printf("₩n");
    }
    printf("₩n");
    //3을 4개씩 3줄로 출력
    //이미 한번 돌렸으니 11번만 반복
    while (cnt <= 11)
    {
        //입력받는 부분 선언, cnt는 1씩 증가하면서 입력받는 중
        printf("next input : ");
        scanf("%d", &pn[cnt]);
        printf("₩n");
        //똑같이 4개씩 3줄의 형태 만들기
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 4; j++)
            {
                //location이라는 변수 만들어서 1, 5번째를 0*4+1, 1*4+1의 형태로 나누어서
생각하고 그걸cnt+1로 나눈 나머지를 저장하고 출력
                int location = (i * 4 + j) % (cnt + 1);  //cnt+1인 이유는 cnt가 1일 때 입력받은
것은 2개니까 맞춰주기 위해서 1을 더함
                printf("%d", pn[location]);
            }
            printf("₩n");
        }
        printf("₩n");
        cnt++;
        //for문이 끝나기 전에 cnt를 1 증가시키고 반복
    }
```

```
    return 0;
}
```



숫자 최대 개수를 12로 지정해서 12번이 되면 프로그램이 종료되게 설정, 1부터 9까지 입력하고 다시 6까지 입력한 테스트 케이스



같은 숫자를 연속해서 작성하면 다음과 같은 결과로 출력되게 설정

3. HW_003

```c
#define _CRT_SECURE_NO_WARNINGS

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

//Node에 data와 그 다음 노드를 만들 구조체 선언
typedef struct _Node
{
    int data;
    struct _Node* next;
} Node;

//제일 머리부분 초기화
Node* head = NULL;

Node* node_node(int data) // 노드 만들기
{
    Node* next_node = (Node*)malloc(sizeof(Node));
    if (next_node == NULL)
        return NULL;

    next_node->data = data; // 새로운 노드 만들기, 먼저 data를 가리키게 해야 함
    next_node->next = NULL; // 그 후 다음 노드를 가리키게 설정

    return next_node;//만들어진 노드 반환
}

//함수를 아래에서 만들어서 위에서 선언
void insert(int num, int data);
void insert_back(int data);
void insert_first(int data);
void delete(int num);
void delete_back();
void delete_first();
int get_entry(int data);
int get_length();
void print_list();
void reverse();

int main()
{
    // 동적할당을 이용해서 sentence에 문장 입력받기
    char* sentence = (char*)malloc(100 * sizeof(char));
    int* num = (int*)malloc(sizeof(int));
    int* data = (int*)malloc(sizeof(int));


    //char sort[100];//get_entry에서 data를 찾을지 index로 찾을지 종류 선택

    // 무한반복 시키기
    while (1)
    {
```

```
printf("입력 : ");
scanf("%s", sentence);

// 입력받은 것과 함수 명 비교하기
if (strcmp(sentence, "insert") == 0)
{
    printf("번호와 데이터 : ");
    scanf("%d %d", num, data);
    insert(*num, *data);//insert 함수
}
else if (strcmp(sentence, "insert_back") == 0)
{
    printf("데이터 : ");
    scanf("%d", data);
    insert_back(*data);
}
else if (strcmp(sentence, "insert_first") == 0)
{
    printf("데이터 : ");
    scanf("%d", data);
    insert_first(*data);
}
else if (strcmp(sentence, "delete") == 0)
{
    printf("번호 : ");
    scanf("%d", num);
    delete(*num);
}
else if (strcmp(sentence, "delete_first") == 0)
{
    delete_first();
}
else if (strcmp(sentence, "delete_back") == 0)
{
    delete_back();
}
else if (strcmp(sentence, "get_entry") == 0)
{
    printf("데이터 : ");
    scanf("%d", data);
    int num = get_entry(*data);


    if (num != -10)
    {
        printf("리스트의 %d 번째에₩n", num);
    }
    else
    {
        printf("리스트에 없다.");
    }
}
else if (strcmp(sentence, "get_length") == 0)
{
    printf("길이 : %d₩n", get_length());
```

```
        }
        else if (strcmp(sentence, "print_list") == 0)
        {
            print_list();
        }
        else if (strcmp(sentence, "reverse") == 0)
        {
            reverse();
        }
        else
        {
            printf("잘못된 방법입니다.₩n");
        }
    }

    free(sentence);
    free(num);
    free(data);

    return 0;
}

void insert(int num, int data) // insert 함수
{
    Node* next_node = node_node(data);//새로운 node 만들기

    if (head == NULL) {
        // 리스트가 비어있는 경우
        if (num == 0) {
            head = next_node;
        }//다시 실행하게 돌림
        return;
    }
    Node* now = head;
    for (int i = 0; now->next != NULL && i < num; i++) {
        now = now->next;//다음 노드를 현재 노드에 넣어서 새로운 노드에 데이터와 번호 받기
    }

    next_node->next = now->next;
    now->next = next_node;
}

void insert_back(int data) // insert_back 함수
{
    Node* next_node = node_node(data);
    if (head == NULL) {
        head = next_node;
    }//헤드가 없는게 아니라면
    else {
        Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = next_node;
        //null이 아닐때까지 돌려서 마지막을 찾고 거기에 새로운 노드 만들기
```

```
    }
}

void insert_first(int data) // insert_first 함수
{
    Node* next_node = node_node(data);
    next_node->next = head;
    head = next_node;
    //그냥 바로 처음 부분에 새로운 노드 만들면 된다.
}

void delete(int num) // delete 함수
{
    if (head == NULL)
        return;//아무것도 없을 때

    if (num == 0) { // 첫 번째 노드를 삭제하는 경우
        Node* temp = head;
        head = head->next;
        free(temp);
        return;//delete_first 사용 불가능, 원래 delete_first를 썼으나, 안돼서 조건문으로 다시
만들어줌
    }

    Node* tmp = head;
    for (int i = 0; i < num - 1; ++i)
    {
        if (tmp->next == NULL)
            return;
        tmp = tmp->next;//다음이 null이라면 다시 실행
    }

    Node* delete_delete = tmp->next;//다음 노드 지우기

    if (delete_delete == NULL)
        return; //삭제할 게 없을 때
    tmp->next = delete_delete->next;//임시로 저장한 노드에 지울 노드 입력
    free(delete_delete);//초기화
}


void delete_back() // delete_back 함수
{
    if (head == NULL)//없을 때
        return;

    if (head->next == NULL)
    {
        free(head);
        head = NULL;
        return;
    }

    Node* tmp = head;
    while (tmp->next->next != NULL)
```

```
    {
        tmp = tmp->next;
    }//임시 다음다음이 null이 아닐 때까지 임시에 다음걸 저장

    free(tmp->next);
    tmp->next = NULL;//그리고 삭제
}

void delete_first() // delete_first 함수
{
    if (head == NULL) return;
    Node* tmp = head;
    head = head->next;
    free(tmp);
}//insert_first와 마찬가지로 그냥 바로 처음꺼 지우기

int get_entry(int data) // get_entry 함수
{
    Node* tmp = head;//맨 처음으로 저장
    int num = 0;
    while (tmp != NULL) //임시가 널이 아닌 동안
    {
        if (tmp->data == data)//저장한것과 data가 같으면 그 숫자 반환
        {
            return num;
        }
        tmp = tmp->next;
        num++;//넘을 계속 증가시켜서 전부 확인
    }
    return -10; // 데이터가 없는 경우
}

int get_length() // get_length 함수
{
    int cnt = 0;//count하는것

    Node* tmp = head;
    while (tmp != NULL) //임시가 널이 아닌동안
    {
        cnt++;
        tmp = tmp->next;//카운트 증가, 임시에는 계속 다음 노드를 저장
    }
    return cnt;//그렇게 반환된 카운트를 반환해서 길이를 알게 함
}

void print_list() // 리스트 출력하는 함수
{
    Node* tmp = head;
    while (tmp != NULL)//널이 아닐 때까지
    {
        printf("%d > ", tmp->data);
        tmp = tmp->next;//data를 출력하고 그 다음 노드로 이동
    }
    printf("X₩n");//맨 마지막에 X로 없음을 나타낸다.
}
```

```
void reverse() // 리스트 뒤집는 함수
{
    Node* left = NULL;
    Node* cur = head;
    Node* right = NULL;

    while (cur != NULL)//중심이 널이 아니면
    {
        right = cur->next;//오른쪽 데이터에 다음 노드 저장
        cur->next = left;//다음 노드에 왼쪽 데이터
        left = cur;//왼쪽은 현재
        cur = right;//현재는 오른쪽 데이터 저장
    }
    head = left;//그리고 한칸씩 왼쪽으로 이동
}
```