최범수_8일차_과제

1. HW_001

```
#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
```

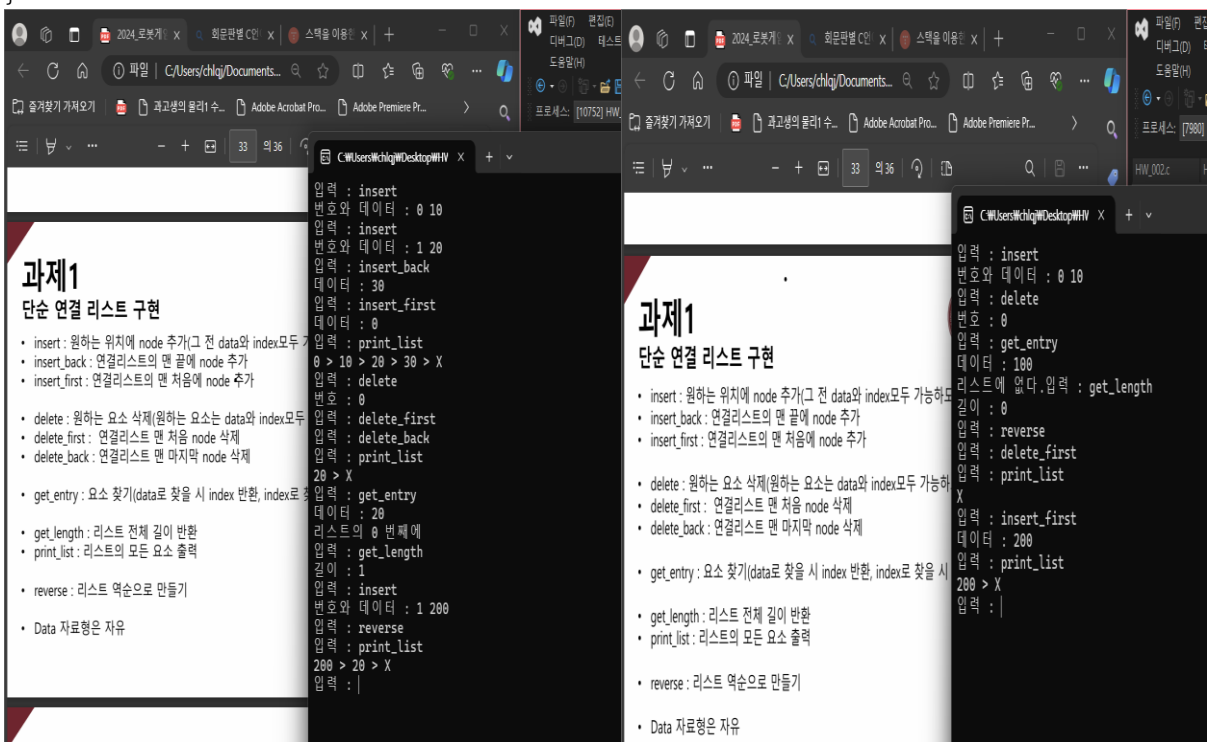
```
// 무한반복 시키기
while (1)
{
    printf("입력 : ");
    scanf("%s", sentence);

    // 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
}
```

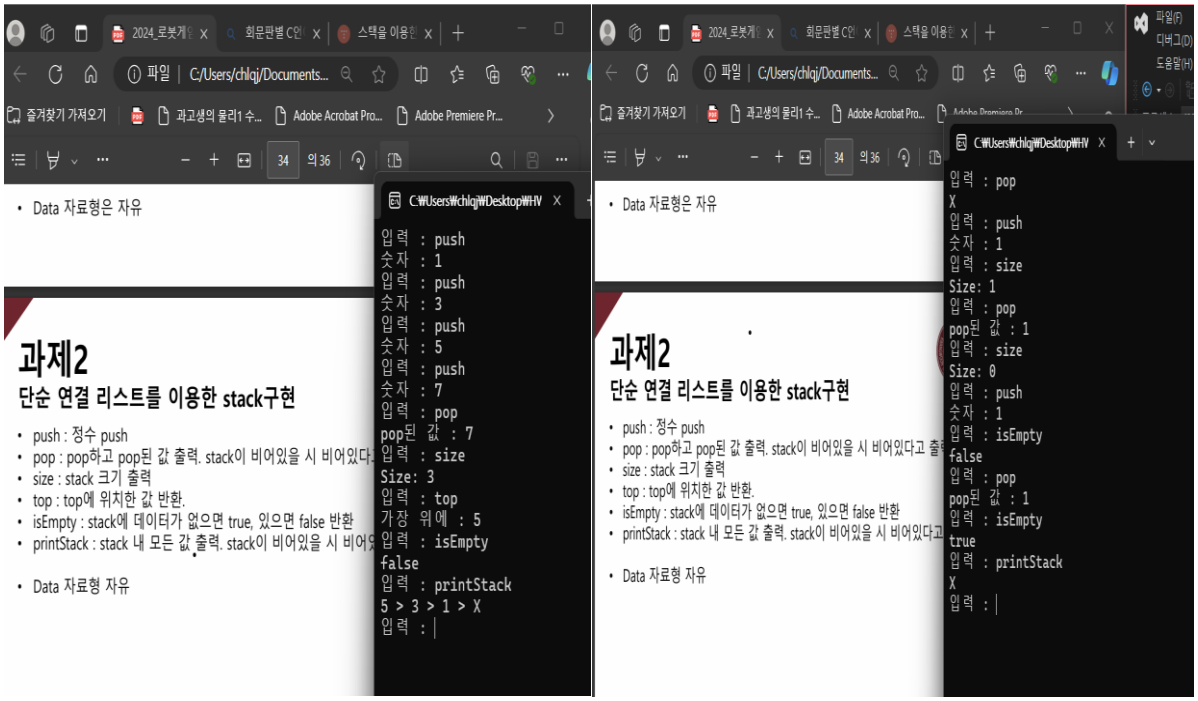


2. HW_002

```
#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;
//head 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;
}

//stack 구현
void push(int data)
{
    Node* next_node = node_node(data);//새로운 노드 만들고 다음에 저장
    next_node->next = head;
    head = next_node;//다음칸으로 옮겨주기
}
//pop 구현
int pop()
{
    if (head == NULL) {
        printf("X₩n");
        return -10; //아무것도 없을때
    }

    int pop_data = head->data;
    Node* temp = head;
    head = head->next;//head를 다음칸으로 저장
    free(temp);
    return pop_data;//pop시킨 데이터를 출력하기 위해 반환
}

int size() // size 함수
{
    int cnt = 0;
```

```
    Node* temp = head;
    while (temp != NULL) {
        cnt++;
        temp = temp->next;
    }
    return cnt;//cnt를 증가시키면서 크기를 측정
}

int isEmpty()
{
    return head == NULL;//head랑 NULL이 같으면 1 아니면 0
}

int top()
{
    if (head == NULL)
    {
        printf("X₩n");//아무것도 없으면 X출력
        return -10;
    }
    return head->data;
}

void printStack() // 리스트 출력하는 함수
{
    Node* tmp = head;
    while (tmp != NULL)
    {
        printf("%d > ", tmp->data);
        tmp = tmp->next;//임시로 저장한 값을 계속 다음 노드로 옮기면서 print를 계속해주는 과정
    }
    printf("비어 있다.₩n");
}

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

    while (1)
    {
        printf("입력 : ");
        scanf("%s", sentence);

        // 입력받은 것과 함수 명 비교하기
        if (strcmp(sentence, "push") == 0)
        {
            printf("숫자 : ");
            scanf("%d", &num);
            push(num);
        }
        else if (strcmp(sentence, "pop") == 0)
        {
            int pop_num = pop();
            if (pop_num != -10)//아까 -10 반환하면 없는거라서 아닌 경우에 추력
```

```
            {
                printf("pop된 값 : %d₩n", pop_num);
            }
        }
        else if (strcmp(sentence, "size") == 0)
        {
            printf("Size: %d₩n", size());
        }
        else if (strcmp(sentence, "top") == 0)
        {
            int top_num = top();
            if (top_num != -10) {//10은 없을 때
                printf("가장 위에 : %d₩n", top_num);//가장 위에값 출력,
            }
        }
        else if (strcmp(sentence, "isEmpty") == 0)
        {
            if (isEmpty() == 1)
                printf("true₩n");//1이면 비어있다
            else
                printf("false₩n");//아니면 채워져있다.
        }
        else if (strcmp(sentence, "printStack") == 0)
        {
            printStack();
        }
        else
            printf("다시 시도해주세요.₩n");
    }

    free(sentence);
    return 0;
}
```



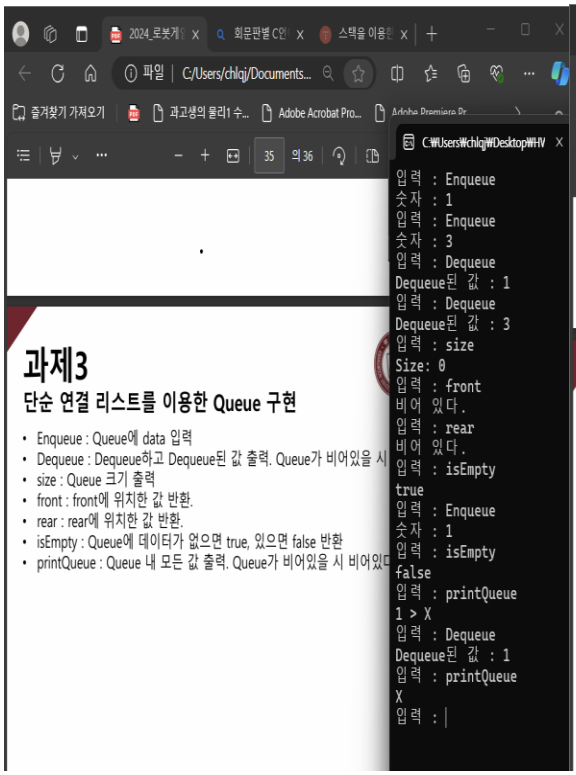
3. HW_003

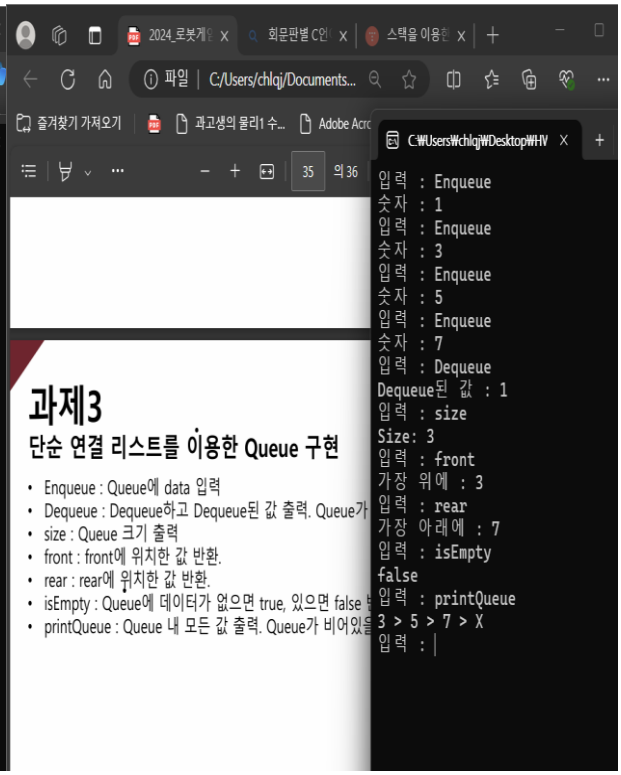
```
#define _CRT_SECURE_NO_WARNINGS

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

//Node에 data와 그 다음 노드를 만들 구조체 선언
typedef struct Node
{
    int data;
    struct Node* next;
} Node;
//head 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;
}
//Enqueue 구현
void Enqueue(int data)
{
    Node* next_node = node_node(data);
    if (head == NULL) {
        head = next_node;
    }
    else {
        Node* temp = head;//새로운 노드 만들고 다음에 저장
        while (temp->next != NULL) {
            temp = temp->next;//다음칸으로 옮겨주기
        }
        temp->next = next_node;
    }
}
//Dequeue 구현
int Dequeue()
{
    if (head == NULL) {
        printf("X₩n");
        return -10; //아무것도 없을때
    }

    int dequeue_data = head->data;
    Node* temp = head;//head를 다음칸으로 저장
    head = head->next;
```

```
    free(temp);
    return dequeue_data;//Dequeue시킨 데이터를 출력하기 위해 반환
}

int size() // size 함수
{
    int cnt = 0;
    Node* temp = head;
    while (temp != NULL) {
        cnt++;
        temp = temp->next;
    }
    return cnt;//cnt를 증가시키면서 크기를 측정
}

int front()
{
    if (head == NULL) {
        printf("비어 있다.₩n");
        return -10; //비어있으면 -10 반환
    }
    return head->data;//아니면 첫번째 반환
}

int rear()
{
    if (head == NULL) {
        printf("비어 있다.₩n");
        return -10;//비어있으면 -10 반환
    }

    Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    return temp->data;//아니면 맨 마지막 데이터 반환
}

int isEmpty()
{
    return head == NULL;//head랑 NULL이 같으면 1 아니면 0
}

void printQueue()
{
    Node* tmp = head;
    while (tmp != NULL)
    {
        printf("%d > ", tmp->data);
        tmp = tmp->next;
    }
    printf("X₩n");//아무것도 없으면 X출력
}

int main()
```

```
{
    char* sentence = (char*)malloc(100 * sizeof(char));
    int num;

    while (1)
    {
        printf("입력 : ");
        scanf("%s", sentence);

        // 입력받은 것과 함수 명 비교하기
        if (strcmp(sentence, "Enqueue") == 0)
        {
            printf("숫자 : ");
            scanf("%d", &num);
            Enqueue(num);
        }
        else if (strcmp(sentence, "Dequeue") == 0)
        {
            int result = Dequeue();
            if (result != -10) {
                printf("Dequeue된 값 : %d₩n", result);
            }
        }
        else if (strcmp(sentence, "size") == 0)
        {
            printf("Size: %d₩n", size());
        }
        else if (strcmp(sentence, "front") == 0)
        {
            int front_num = front();
            if (front_num != -10)//-10은 없을 때
            {
                printf("가장 위에 : %d₩n", front_num);
            }
        }
        else if (strcmp(sentence, "rear") == 0)
        {
            int rear_num = rear();
            if (rear_num != -10) //아까 -10 반환하면 없는거라서 아닌 경우에 출력
            {
                printf("가장 아래에 : %d₩n", rear_num);
            }
        }
        else if (strcmp(sentence, "isEmpty") == 0)
        {
            if (isEmpty() == 1)
                printf("true₩n");//비어있으면 아까 반환값이 1 아니면 0
            else
                printf("false₩n");
        }
        else if (strcmp(sentence, "printQueue") == 0)
        {
            printQueue();
        }
        else
```

```
            printf("다시 시도해주세요₩n");
        }

        free(sentence);// 초기화
        return 0;
}
```



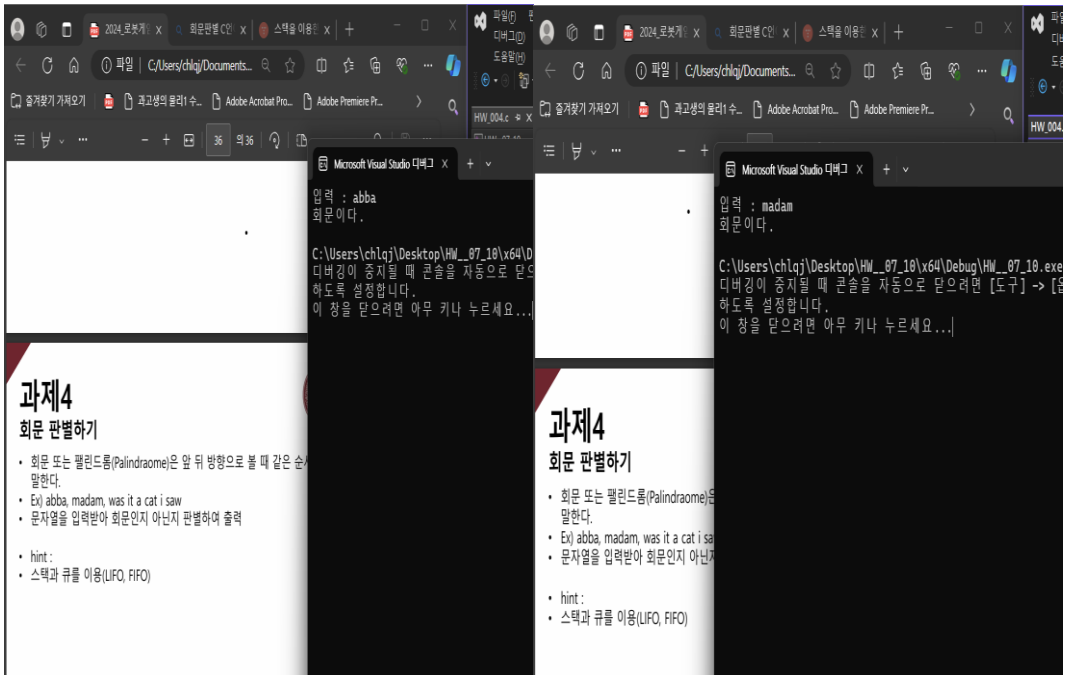


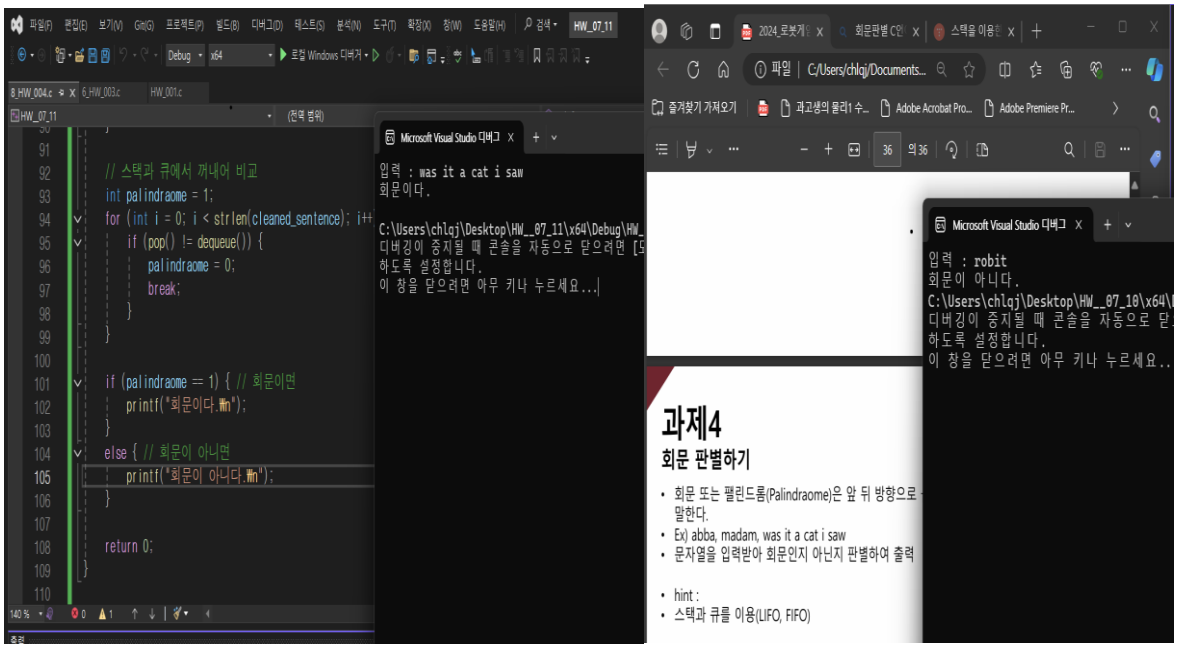
4. HW_004

```
#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 {
    char data;
    struct Node* next;
} Node;

// stack, queue 초기화
Node* stack_stack = NULL;
Node* queue_queue = NULL;

Node* node_node(char 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;
}

void push(char num) { // 스택에 push
    Node* next_node = node_node(num); // 새로운 노드 만들고 다음에 저장
    next_node->next = stack_stack;
    stack_stack = next_node; // 다음 칸으로 옮겨주기
}

char pop() { // 스택에서 팝
    if (stack_stack == NULL) {
        return '₩0';
    }
    char pop_num = stack_stack->data;
    Node* tmp = stack_stack;
    stack_stack = stack_stack->next;
    free(tmp);
    return pop_num; // pop 한 숫자를 반환
}

void enqueue(char num) { // 큐에 enqueue
    Node* next_node = node_node(num); // 새로운 노드 만들고 다음에 저장
    if (queue_queue == NULL) {
        queue_queue = next_node; // queue에 next노드 저장
    }
    else {
        Node* tmp = queue_queue; // 0이 아니면 그 다음 임시 노드 만들고 임시 노드 다음의 노드에
다음 노드 값 저장
```

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

char dequeue() { // stack에 pop처럼 이번에는 반대로 큐에 dequeue, 각각 맨 첫번째, 맨 뒤번째의 숫자를 비교하기 위해 사용
    if (queue_queue == NULL) {
        return '₩0';
    }
    char dequeue_num = queue_queue->data;
    Node* tmp = queue_queue;
    queue_queue = queue_queue->next;
    free(tmp);
    return dequeue_num; // dequeue된 숫자 반환
}

int main() {
    char sentence[1000] = "";
    char cleaned_sentence[1000] = "";
    int num = 0;

    printf("입력 : ");
    scanf("%[^₩n]", sentence); // 문장 입력받고, 띄어쓰기 포함

    // 띄어쓰기 제거하여 cleaned_sentence에 저장
    for (int i = 0; i < strlen(sentence); i++) {
        if (sentence[i] != ' ') {
            cleaned_sentence[num++] = sentence[i];
        }
    }
    cleaned_sentence[num] = '₩0'; // 문자열의 끝을 알리는 null 문자 추가

    // cleaned_sentence의 각 문자를 스택과 큐에 넣기
    for (int i = 0; i < strlen(cleaned_sentence); i++) {
        push(cleaned_sentence[i]); // push, enqueue에 각각 알파벳 입력
        enqueue(cleaned_sentence[i]);
    }

    // 스택과 큐에서 꺼내어 비교
    int palindraome = 1;
    for (int i = 0; i < strlen(cleaned_sentence); i++) {
        if (pop() != dequeue()) {
            palindraome = 0;
            break;
        }
    }

    if (palindraome == 1) { // 회문이면
        printf("회문이다.₩n");
    }
    else { // 회문이 아니면
        printf("회문이 아니다.₩n");
```

```
    }

    return 0;
}
```


입력 : abba
회문이다.
입력 : madam
회문이다.
과제4
회문 판별하기
• 회문 또는 팰린드롬(Palindraome)은 앞 뒤 방향으로 볼 때 같은 순
말한다.
• Ex) abba, madam, was it a cat i saw
• 문자열을 입력받아 회문인지 아닌지 판별하여 출력
• hint :
• 스택과 큐를 이용(LIFO, FIFO)


// 스택과 큐에서 꺼내어 비교
int palindraome = 1;
for (int i = 0; i < strlen(cleaned_sentence); i++
if (pop() != dequeue()) {
palindraome = 0;
break;
}
}
if (palindraome == 1) { // 회문이면
printf("회문이다.₩n");
}
else { // 회문이 아니면
printf("회문이 아니다.₩n");
}
return 0;
}
입력 : was it a cat i saw
회문이다.
입력 : robit
회문이 아니다.
과제4
회문 판별하기
• 회문 또는 팰린드롬(Palindraome)은 앞 뒤 방향으로
말한다.
• Ex) abba, madam, was it a cat i saw
• 문자열을 입력받아 회문인지 아닌지 판별하여 출력
• hint :
• 스택과 큐를 이용(LIFO, FIFO)