데이터구조 레포트 3

학년	2학년	학번	20211379
이름	장원영	제출일	2024.06.08

일체의 부정 행위 없이 스스로의 노력으로 작성한 레포트임을 확인합니다.

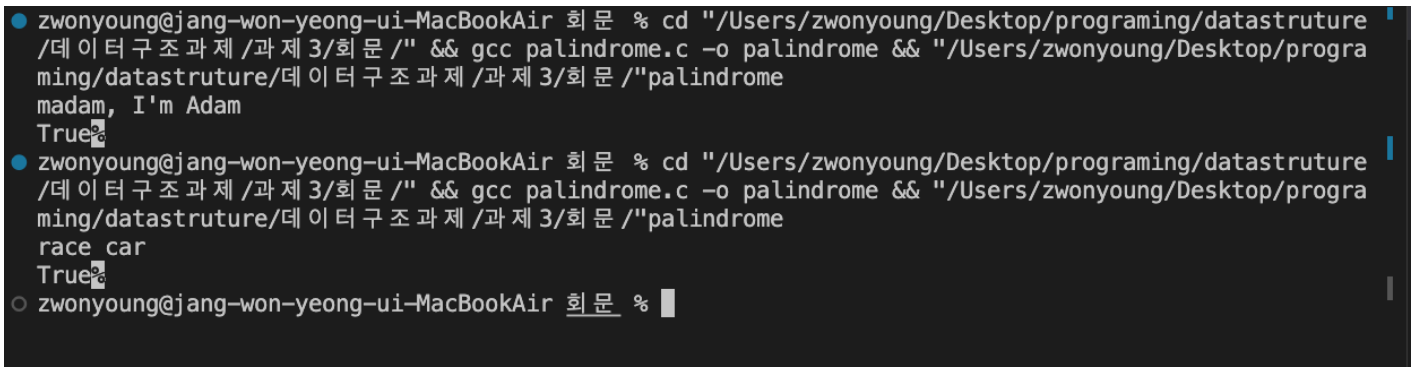
문제 1. 회문

코드 및 주석

```
#define _CRT_SECURE_NO_WARNINGS // scanf_s를 사용하지 않음으로 발생하는 오류 해결

#include <stdio.h> // 기본 입출력을 위한 stdio.h
#include <string.h> // 문자열의 길이 측정 함수 strlen을 사용하기 위한 string.h
#include <ctype.h> // 입력받은 아스키 문자를 대문자든 소문자든 소문자로 변경시켜주는 tolower함수를 사용하기 위한 ctype.h

#define Datatype char // Datatype이라는 글자를 컴파일 전에 전부 char로 치환할 것을 define 전처리기로 선언
#define MAX_SIZE 100 //MAX_SIZE라는 글자를 전부 컴파일 전에 100이라는 상수로 치환할 것을 define 전처리기 선언

typedef enum { // 순서있는 상수 배열 enum 선언
        // enum은 C언어의 예약어로서 enum을 통해 상수 배열을 선언 순서대로 0,1,...과 같이 저장,
        // FALSE는 0 TRUE는 1과 매칭되며 bool 자료형의 요소로 쓸 수도, FALSE, TRUE 따로 쓸 수도 있다.
        FALSE, TRUE
}Bool; // 이 enum의 이름을 Bool로 부를 것을 선언

typedef struct ArrayStack { // 스택의 정의
        Datatype data[MAX_SIZE]; // 스택을 정의한다
        int top; // 스택의 top 즉 인덱스이다
} ArrayStack; // 스택의 이름을 ArrayStack 자료형으로 부를 것임을 선언

void stack_init(ArrayStack* stack){ // 스택의 초기화 스택의 주소를 매개변수로 가진다 즉, 스택 변수를 포인터로 직접 조작할 것임을 알 수 있다.
        stack->top = -1; //stack의 top을 -1로 설정(stack은 포인터이므로 ->로 접근한다.) -1일때를 스택이 빈 것으로 판단하여 초기화시켜준다
}

int push_stack(ArrayStack* stack, Datatype data){ // 스택의 push 함수, 매개변수로 스택의 주소, 데이터를 매개변수로 받는다.
        if(stack->top < MAX_SIZE){ // 스택의 top이 데이터 사이즈보다 작다면 : 삽입 가능
                stack->data[++stack->top] = data; // stack의 top을 1 증가시키고(전위 연산자) 그걸 인덱스번호로 삽입한곳에 데이터 삽입
                return 1; // 삽입 성공을 의미하는 1 return
        } else { // 그렇지 않다면
                return 0; // stack overflow이므로 그것을 의미하는 0 return
        }
}

Datatype pop_stack(ArrayStack* stack){ // 스택의 pop 함수, 매개변수로 스택의 주소를 받는다.
        if(stack->top > -1){ // 만약 스택이 비어있는 스택의 의미인 top이 -1 보다 크다면
                Datatype data = stack->data[stack->top--]; // stack의 top을 인덱스번호로 삽입한곳의 데이터를 data 변수에 저장하고 top을 1 감소시킨다
(후위 연산자)
                return data; // 빼둔 data 반환
        } else { // 스택이 비어있다면
                return '\0'; // stack onderflow를 의미하는 널 문자 반환
        }
}

int palindrome(char in_str[]){ // 배열의 주소를 매개변수로 받는 회문 판단 함수 선언
        ArrayStack s; // 배열 스택 s 선언
        int i; // 반복문 변수 i 선언
        char ch, chs; // 문자열 in_str의 i번째 요소를 저장할 ch, 스택에서 나온 문자를 저장할 chs 선언
        int len = strlen(in_str); // in_str 문자열의 길이를 len에 저장

        ArrayStack stack; // stack형 구조체 stack 선언
        stack_init(&stack); // 스택 초기화 함수 stack_init을 호출하고 stack의 주소 넘겨주기

        for(i = 0; i < len; i++){ // 문자열의 길이만큼 반복
                ch = in_str[i]; // ch에 현재 인덱스의 원소를 집어넣기
                if( // 만약 ch가 스페이스바 혹은 구두점이면
                ch == ' ' || // 스페이스바
```

```
                ch == '"' || // 큰따옴표
                ch == '\'' || // 이스케이프 처리로 '를 정상적으로 인식하게 함 // 작은 따옴표
                ch == ',' || // 쉼표
                ch == '.' || // 마침표
                ch == '!' || // 느낌표
                ch == '?' || // 물음표
                ch == '`' || // 악센트
                ch == ':' || // 콜론
                ch == ';' // 세미콜론
                ){ // 스페이스이거나 구두점이면
                        continue; // 구문 넘기고 계속 반복문 진행
                }
                ch = tolower(ch); // 받아들인 문자를 영문 소문자로 모두 변경
                push_stack(&stack, ch); // ch를 스택에 집어넣기
        }
        for(i = 0; i < len; i++){ // 문자열의 길이만큼 반복
                ch = in_str[i];
                if( // 만약 ch가 스페이스바 혹은 구두점이면
                ch == ' ' || // 스페이스바
                ch == '"' || // 큰따옴표
                ch == '\'' || // 이스케이프 처리로 '를 정상적으로 인식하게 함 // 작은 따옴표
                ch == ',' || // 쉼표
                ch == '.' || // 마침표
                ch == '!' || // 느낌표
                ch == '?' || // 물음표
                ch == '`' || // 악센트
                ch == ':' || // 콜론
                ch == ';' // 세미콜론
                ){ // 스페이스이거나 구두점이면
                        continue; // 구문 넘기고 계속 반복문 진행
                }
                ch = tolower(ch); // ch를 소문자로 변경
                chs = pop_stack(&stack); // pop한 문자를 순서대로 저장한 문자열
                if(ch != chs){ // 만약 pop한 문자열과 원래 문자열이 한글자라도 일치하지 않으면
                        return FALSE; // 거짓 반환후 함수 종료 거짓은 == 0
                }
        }
        return TRUE; // 참 반환후 함수 종료 참은 == 1
}
int main(){
        char array[100]; // 입력받을 문자열의 공간 array를 선언하고 그 크기를 100으로 설정
        scanf("%[^\n]*c", array); // 문자열을 array에 입력받기
        if(palindrome(array) == 1){ // 입력받은 문자열의 주소를 파라미터로 넘긴 회문 함수가 참을 반환하면
                printf("True"); // True 출력
        } else { // 거짓이면
                printf("False"); // false 출력
        }
        return 0; // 함수 종료
}
```

출력 화면 및 설명



순서대로 설명드리겠습니다.

```
#define _CRT_SECURE_NO_WARNINGS
```

는 필자가 vscode에서 작성하였기에 비주얼 스튜디오에서 scanf_s를 사용하지 않아 발생하는 오류를 방지하기 위해 적어놓은 전처리기 구문입니다.

데이터 타입과 최대 크기의 정의

```
#define Datatype char
#define MAX_SIZE 100
```

이 둘은 #define문으로 정의되는데 앞에 #이 붙는 명령어는 전처리기로서 컴파일 되기 전에 작동하는 구문입니다.

이때 위와 같이 define 문을 선언하면 코드 내에 앞의 문장이 적혀 있는 내용은 뒤에 적혀있는 내용으로 자동으로 컴파일러가 변환시켜 준 후 컴파일을 진행하게 됩니다.

이 둘을 정의하고 스택에 처리할 데이터의 변수명을 Datatype으로, 스택 배열의 최대 사이즈를 MAX_SIZE로 적어서 코딩함으로서 코드 수정을 간편하게 만들어 줍니다.

FALSE와 TRUE 구현

문제 조건의 코드에 적혀있는 참,거짓 자료형인 boolean 자료형을 사용하기 위해 기본 제공 헤더파일인 <stdbool.h>를 사용할 수도 있지만 TRUE, FALSE와 같이 일부러 코드에 대문자로 표시한 것을 미루어보아 bool 형 자료형을 직접 구현해보라는 뜻으로 이해하고 직접 구현해 보았습니다.

우선 저는 c언어의 기본 기능인 enum을 사용하였는데 enum에 대해 간략하게 설명해 보자면
enum은 c언어 컴파일러가 미리 예약해둔 예약어로서
`enum{단어1, 단어2, 단어3}`
과 같이 사용하는 순간 각각의 단어1, 단어2, 단어3은 순서대로 0, 1, 2, 3의 '상수'가 배정되게 되는 구조라고 보시면 됩니다. 그것을 typedef로 감싸주어 명칭이 Bool인 타입으로 재정의 해주면서 우리가 원하는 true, false의 역할을 할 수 있는 boolean 자료형이 생성되는 것입니다.

이때 FALSE는 0, TRUE는 1의 상수가 대입됩니다.

스택 설명

문제에서 주어진 코드로 미루어보아, 본 알고리즘은 스택을 사용해야 하는데, Arraystack이라는 명칭을 사용하였으므로 스택을 배열로 구현하여 데이터 구조와 동작을 구성하였습니다.

구성은 다음과 같습니다...

1. 데이터 구조의 정의(구조체)
    1. 스택의 정의(ArrayStack)
       스택 구조체는 내용을 저장할 공간 data 배열과

       데이터의 입출구인 top을 가리키게 하는 인덱스 번호를 저장할 int형 변수 top으로 구성됩니다.
2. 동작의 정의
    1. 스택의 초기화(stack_init)
       인덱스를 가리키는 스택의 top이 -1일때를 스택이 빈 상태로 가정합니다.

       초기화는 top의 수치를 -1로 해 주는 동작을 받은 포인터를 통해 직접 조작해줍니다.
    2. push(push_stack)
       넣어주려는 스택의 주소와 넣어주려는 데이터를 매개변수로 받아준 다음 작동합니다.

       우선 기본적으로 스택의 크기가 배열의 크기를 넘어가는지 확인하고 넘어가지 않는다면

       기존 데이터를 갈아끼우면 안되므로 전위 연산자로 top의 값을 미리 증가시키고 data배열의 인덱스 요소로 넣어주고 그 데이터의 top에 위치한 요소를 매개변수로 받은 데이터로 교체해 줍니다.
       그리고 성공적으로 삽입한 것의 의미로 1을 리턴해 줍니다.

       만약 top의 크기가 배열의 최대 사이즈 이상이면 스택 오버플로우이므로
       0을 리턴해줍니다.
    3. pop(pop_stack)
       데이터를 빼내려는 스택의 주소를 매개변수로 받습니다.

       우선 기본적으로 스택이 비어있을때를 제외하고는 pop이 가능하므로...

입출력 부분에 관하여....

메인 구문에서는
`scanf("%[^\n]*c", array);`
를 통해 scanf 값을 받고 있는데

원래라면 %c %d %s같은 형식 지정자로 값을 받아 주어야겠지만,
그냥 이런 식으로 입력을 받아버리면 띄어쓰기를 입력하는 순간 scanf가 입력을 받지 않기 때문에 두가지 방법중 하나로 받아 주어야 하는데

1. fgets 함수를 통해 입력받기(fgets함수는 개행 문자 전까지의 모든 내용을 입력받는다.)
2. 후술할 형식 지정자 옵션을 통해서 입력받기

필자는 후자를 택하였습니다.

%[문자이름]과 같이 입력하면 그 문자에 해당하는 내용을 계속 받습니다.

%[\n]이면 개행 문자를 계속 받고 개행 문자가 아닌 문자가 입력되는 순간, 그 이전까지만 받게 됩니다.
개행 문자만 계속 받다가 개행 문자가 아닌 문자를 넣고 개행을 하면 입력을 그만 받는 것입니다.

하지만 앞에 ^를 붙이게 되면 놀랍게도 그 문자의 부정이 됩니다.

즉 %[^\n]은 개행문자가 '아닌'문자를 계속 받다가 개행문자 '이면' 입력을 그만 받는 것이고
*c 옵션을 통해 개행문자 전까지만 입력을 받겠다는 뜻이 되게 됩니다.

이런 식으로 입력받으면 우리가 원하는 한 줄을 그대로 받을 수 있게 됩니다.

그 값을 array에 넣어주었고 그 array의 시작 주소를 palindrome함수에 파라미터로 넘겨 주었습니다.

그리고 이 함수에서 회문인지 아닌지 판단하여 회문이면 TRUE, 아니면 FALSE을 반환하게 하고 그 값에 따라 작동하도록 조건문의 조건에 넣어주어 true와 false를 출력하게 만들었습니다.

메인 회문 알고리즘

---

palindrome함수의 핵심은 다음과 같습니다. 회문은 앞으로 읽어도 거꾸로 읽어도 같은 단어인데 입력받은 문자열을 거꾸로 나열한 다른 문자열을 생성하면 되는데, 스택은 순서대로 넣고 pop하면 역순으로 뱉어내는 성질을 가지고 있으므로 스택의 자료구조를 이용해 이 문제를 해결하였습니다.

ArrayStack형 자료형 스택 s를 선언하였고 반복문 인자 변수 i, 원본 문자열의 요소를 순서대로 지정할 ch, 스택에서 뽑아낸 문자열 요소를 순서대로 지정할 chs 까지 선언한 후 len 변수를 선언하였습니다.

또한 매개변수로 받아들인 문자열의 길이를 미리 저장해 두었는데, 미리 값을 저장해 두고 중복 연산을 막기 위해 이런 행동을 해 주었습니다.

문자열의 길이를 n이라고 가정했을때 strlen함수는 Θ(n)의 시간 복잡도를 가지기 때문입니다.

그리고 그 len만큼 1개씩 순차적으로 배열 요소를 돌아보면서 위의 분기 조건과 같이 공백이거나, 구두점인 조건들을 전부 if문으로 걸러낸 후 나머지 문자 요소만 스택에 집어넣은 뒤

이후의 for문에서도 똑같이 len만큼 1개씩 순차적으로 순회하면서 공백과 구두점인 부분은 전부 무시하면서 pop한 데이터와 그 위치의 데이터를 비교하고 전부 같으면 회문이 맞으므로 true 아니라면 false를 리턴하였습니다.

이로서 우리가 원하는 알고리즘이 생성된 것을 확인할 수 있겠습니다.

문제 2. 괄호의 짝

코드 및 주석

```
#define _CRT_SECURE_NO_WARNINGS // scanf_s를 사용하지 않음으로 발생하는 오류 해결

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

#define Datatype char // [],{},()를 받아들일 것이기에 스택이 받을 데이터를 문자 데이터로 설정한다

typedef struct Node { // 스택 노드의 정의
        Datatype data; // 문자 1개를 받아들일 data
        struct Node* link; // 다음 노드를 가리킬 link
}Node; // Node로 부를것을 선언

typedef struct LinkedStack { // 연결 스택의 정의
        Node* top; // 가장 위쪽 노드를 가리킬 Node 포인터형 top 선언
}LinkedStack; // LinkedStack으로 부를 것을 선언

void init_stack(LinkedStack* LS){ // 스택의 초기화
        LS->top = NULL; // 스택의 top을 NULL로 초기화함으로서 스택이 비었음을 표현
}

Node* create_node(Datatype data){ // 노드의 생성
        Node* newnode = (Node*)malloc(sizeof(Node)); // Node형 포인터 newnode를 선언하고 그 포인터가 동적 생성한 Node를 가리킬수 있게 함
        if(newnode == NULL){ // 동적 할당 실패시
                return NULL; // NULL 반환
        } else { // 동적 할당 성공시
                newnode->data = data; // 파라미터로 받아준 데이터를 새로 동적 할당한 노드의 데이터에 삽입
                newnode->link = NULL; // 새로 동적 할당한 노드의 link가 아무것도 가리키지 않음을 의미하는 NULL로 초기화
        }
```

```
        return newnode; // 새로 동적 할당한 노드 반환
}

void push_stack(LinkedStack* LS, Datatype data){ // 스택의 push
        Node* newnode = create_node(data); // 새로운 노드 생성
        if(LS->top == NULL){ // 스택이 비어있다면
                LS->top = newnode; // 스택의 top이 newnode를 가리키게 한다
        } else { // 스택이 비어있지 않으면
                newnode->link = LS->top; // newnode의 link가 top이 가리키는 노드를 가리키게 하고
                LS->top = newnode; // newnode를 새로운 top으로 한다
        }
}

Datatype pop_stack(LinkedStack* LS){ // 스택의 pop
        Node* delNode; // 지울 노드를 가리킬 포인터
        Datatype data; // 반환할 데이터

        if(LS->top == NULL){ // 스택이 비어있다면
                return '0'; // 0 반환
        } else { // 스택이 비어있지 않으면
                delNode = LS->top; // 현재의 top이 pop할 노드이므로 스택의 top을 delNode에 저장
                data = delNode->data; // pop할 노드의 데이터를 data에 저장
                LS->top = LS->top->link; // 링크드 리스트의 top을 링크드 리스트의 top이 가리키는 노드가 가리키는 다음 노드를 가리키게 한다
                free(delNode); // 지울 노드의 동적 할당 해제
                return data; // 저장한 데이터 반환
        }
}

int isleft(char str_i){ // 입력받은 문자가 ( { [ 중 하나인지를 판단해주는 함수
        if(str_i == '(' || str_i == '{' || str_i == '['){ // 이중에 하나라도 해당된다면
                return 1; // 1 반환
        } else { // 거짓이면
                return 0; // 0반환
        }
}

int isRight(char str_i){ // 입력받은 문자가 ) } ] 중 하나인지를 판단해주는 함수
        if(str_i == ')' || str_i == '}' || str_i == ']'){ // 이중에 하나라도 해당된다면
                return 1; // 1 반환
        } else { // 거짓이면
                return 0; // 0반환
        }
}

int isCouple(char left, char right){
        if(left == '(' && right == ')'){ // 소괄호의 짝이 맞다면
                return 1;
        } else if(left == '{' && right == '}'){ // 중괄호의 짝이 맞다면
                return 1;
        } else if(left == '[' && right == ']'){ // 대괄호의 짝이 맞다면
                return 1;
        } else{ // 짝이 안맞거나 이상한값 들어갔으면
                return 0;
        }
}

int isTrue(char input[]){
        int len = strlen(input); // 문자열의 길이 저장

        LinkedStack stack; // 새로운 스택 선언
        init_stack(&stack); // 스택 초기화

        for(int i = 0; i < len; i++){ // 문자열의 길이만큼 반복
                if(isleft(input[i])){ // 왼쪽 기호라면
                        push_stack(&stack, input[i]); // 스택에 집어넣기
                        printf("push successful\n");
                } else if(stack.top && isRight(input[i]) == 1 && isCouple(stack.top->data, input[i]) == 1){ // 스택이 비어있지 않고 입력 받
은값이 오른쪽 닫는 기호이면서 스택의 top이 가리키는 노드의 데이터와 짝이 맞을때
                        pop_stack(&stack); // 스택에서 빼내기
                        printf("pop successful\n");
                } else { // 조건에 부합하지 않으면
                        printf("is not true\n");
                        return 0; // 0리턴
                }
        }
        if (stack.top == NULL) { // 모든 문자열 요소가 통과하고 스택이 비어 있으면
```

```
                return 1; // 1 반환
        } else {
                return 0; // 스택이 비어있지 않으면 0 반환
        }
}

int main(){
        char input[101]; // 100개까지 받을 수 있는 문자열 선언

        scanf("%s", input); // 문자열 받아들이기

        if(isTrue(input)){ // 만약 조건이 참이면
                printf("true"); // true 출력하기
        } else { // 조건이 거짓이면
                printf("false"); // false 출력하기
        }
        return 0; // 프로그램 정상 종료
}
```

출력 화면 및 설명

```
zwonyoung@jang-won-yeong-ui-MacBookAir programing % cd "/Users/zwonyoung/Desktop/programing/datastr
uture/gwnu_DataStruture/데이터구조과제/과제3/괄호의 짝/" && gcc bracket.c -o bracket && "/Users/zwo
nyoung/Desktop/programing/datastruture/gwnu_DataStruture/데이터구조과제/과제3/괄호의 짝/"bracket
([[]])]
push successful
push successful
push successful
pop successful
is not true
false%
zwonyoung@jang-won-yeong-ui-MacBookAir 괄호의 짝 % cd "/Users/zwonyoung/Desktop/programing/datastru
ture/gwnu_DataStruture/데이터구조과제/과제3/괄호의 짝/" && gcc bracket.c -o bracket && "/Users/zwon
young/Desktop/programing/datastruture/gwnu_DataStruture/데이터구조과제/과제3/괄호의 짝/"bracket
([][]{[()]})
push successful
push successful
pop successful
push successful
pop successful
push successful
push successful
push successful
pop successful
pop successful
pop successful
pop successful
true%
zwonyoung@jang-won-yeong-ui-MacBookAir 괄호의 짝 %
```

우선 문제의 조건 및 설계 방향부터 조금 짚고 넘어갈 필요가 있습니다.

교수님께 직접 문의한 결과 세부적인 문제의 조건은 다음과 같습니다

1. {[]} 와 같이 소괄호 대괄호의 대소까지 비교할 필요는 없다.
2. {[}]와 같이 양쪽의 개수가 같으나 엮여서 불균형하게 닫히는 것은 허용되지 않는다.
3. 괄호를 제외한 다른 글자를 입력받는 전제로 짜지 않아도 괜찮다.

따라서 이를 반영하여 문제 풀이에 접근하였습니다.

성질 1

일단 괄호는 왼쪽 여는 기호와 오른쪽 닫는 기호로 이루어집니다.

그 말을 다시 풀어보자면 같은 짝의 왼쪽 기호와 오른쪽 기호는 항상 일치해야 한다는 것입니다.

(() 도 성립하지 않고 {(()){} 도 참이 아닙니다. 짝이 맞지 않는 기호가 존재하기 때문입니다.

성질 2

괄호는 반드시 내부에서든 외부에서든 순서에 맞게 짝이 맞아야 합니다.

아까 2번에서 보았듯
{[}] 과 같은 기호는 절대로 성립할 수 없습니다. {[][]}는 성립하는데,

이것을 판단하기에 가장 좋은 자료 구조가 존재합니다. 바로 스택입니다.

여는 기호는 스택에 push, 닫는 기호를 만나면 스택의 top과 비교 후 다음 두 가지를 비교해야 하는데,

1. 둘의 기호가 짝이 맞는지 (둘의 기호가 짝이 맞지 않으면 {[}]는 거짓)
2. 스택이 비어있지 않은지 (닫는 기호와 여는 기호의 균형)

같지 않으면 위 조건에 성립하지 않으므로 바로 false를 반환하게 됩니다.

그리고 마지막으로 성질 1에 따라 반드시 균형이 맞아야 하므로
위의 판단 로직에 따라 분류를 완료한 후 스택에 여는 괄호가 남아있다면 이는 짝이 맞지 않는 것이므로 false를 반환하게 됩니다.

따라서 위 로직을 판단하는 isTrue함수와 조건 판단식이 너무 길어져 가독성을 위해 만들어둔 isleft, isRight, isCouple 함수로 조건 판단을 하게 하여 구현해 주었습니다.

따라서 위의 코드와 같이 중간 과정과 결과값이 같이 출력되도록 구현하였고 성공적으로 동작하는 것을 확인했습니다.

문제 3. 미팅 중개 프로그램

일단 아래 2개의 문제 다 푸는 접근 방식과 함수는 놀라울 정도로 비슷하니 문제 접근법만 여기서 서술하고 넘어가겠습니다.

남학생과 여학생의 매칭 서비스를 구현하기 위해서는 다음과 같은 내용을 구현해야 합니다.

1. 남학생 큐와 여학생 큐 생성 이때의 큐는 초기화하여 비어있음을 표시
2. 사용자의 정보를 입력받는데 이 정보는 이름과 성별로 구성되어 있다
3. 이름과 성별 순으로 입력받는데, 이름을 stop으로 입력받으면 프로그램 종료 아니면 성별로 분기문을 나눈다.
4. 성별이 남자면 여자 큐를 확인하고 비어있으면 지금까지 입력한 데이터(이름, 성별)의 정보를 남자 큐에 집어넣고 누군가 큐에 있다면 pop시키고 그 이름들을 출력
5. 성별이 여자면 남자 큐를 확인하고 비어잇으면 지금까지 입력한 데이터(이름, 성별)의 정보를 여자 큐에 집어넣고 누군가 큐에 있다면 pop시키고 그 이름들을 출력

즉 문제를 각각의 경우로 분할해서 생각하면 그리 어려운 문제가 아닌 것을 알 수 있습니다.

3.1 원형 큐로 구현

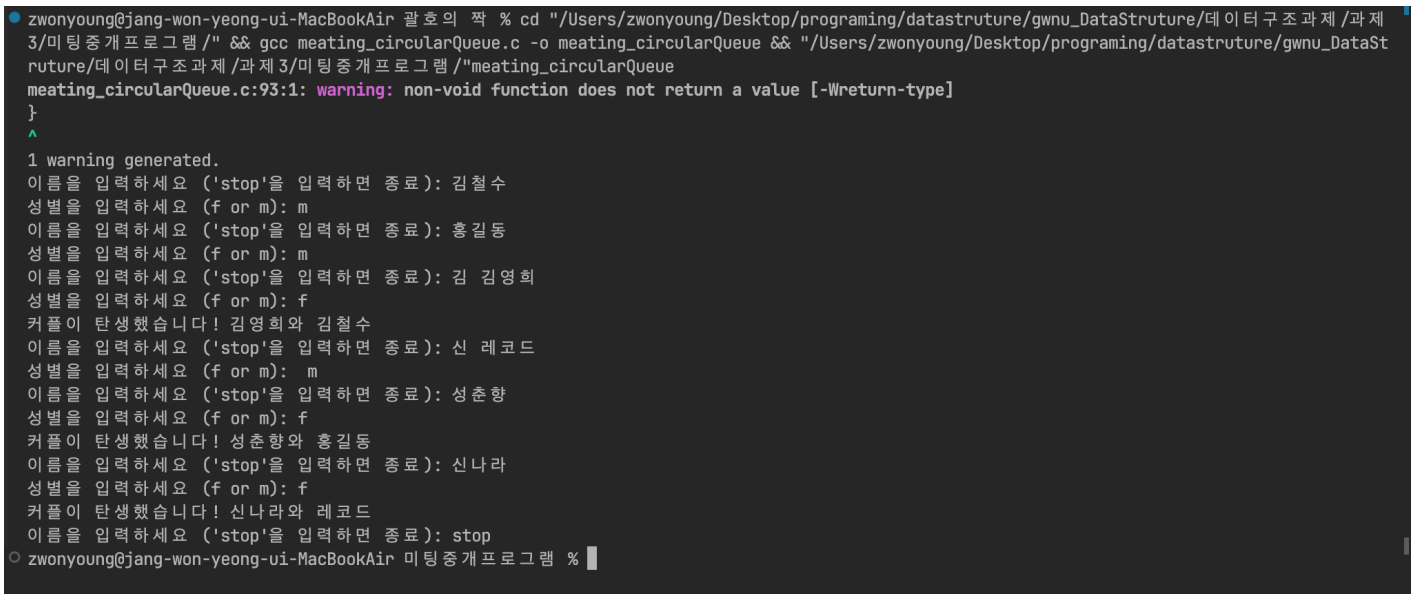
코드 및 주석

```c
#define _CRT_SECURE_NO_WARNINGS // scanf_s를 사용하지 않음으로 발생하는 오류 해결

#include <stdio.h> // 기본 입출력을 위한 헤더파일
#include <stdlib.h> // 노드의 동적 할당을 위한 malloc을 사용하기 위한 헤더파일
#include <stdbool.h> // is_queue_empty 함수의 반환값을 bool형으로 하기 위한 헤더파일
#include <string.h> // 입력한 문자열이 stop인지 확인하기 위해 비교를 위한 strcmp를 사용하기 위한 헤더파일

#define Datatype People // Datatype이라는 글자를 모두 People로 바꿔서 컴파일 할 것을 선언하는 전처리기
#define MAX_SIZE 100

typedef struct People { // 입력받을 사람에 대한 구조체 선언
        char name[31]; // 그 사람의 이름
        char gender; // 그 사람의 성별
} People; // People형 자료형으로 부를 것을 정의

typedef struct{ // 원형 큐의 정의
        People data[MAX_SIZE]; // peaple의 데이터의 포인터를 저장할 포인터 배열을 가진다 최대 큐의 크기는 MAX_SIZE를 따라간다
        int front; // 데이터의 출구를 표시할 인덱스 번호 저장
        int rear; // 데이터의 입구를 표시할 인덱스 번호 저장
}Queue; // 이름을 Queue로 할 것을 정의

void init_queue(Queue* queue){ // 원형 큐의 초기화
        queue->front = queue->rear = 0; // 큐의 front와 rear를 모두 0으로 초기화
}
```

```c
bool is_queue_full(Queue* queue){ // 큐가 가득 차있는지 검사하는 로직
        return ((queue->rear + 1) % MAX_SIZE) == queue->front; // 가장끝쪽인 입구에 +1을 한 값이 front와 동일하다면 +1을 해 준 값에 % 연산을 해 준 이유는 최대값 초과 방지이다.
}

bool is_queue_empty(Queue* queue){ // 큐가 비었는지 검사하는 로직
        return queue->front == queue->rear; // 큐의 시작과 끝이 같다면
}

void push_queue(Queue* queue, Datatype data){ // queue의 push
        if(is_queue_full(queue)){ // 만약 큐가 가득 차 있다면
                printf("포화 큐입니다! 삽입이 불가능합니다."); // 삽입 불가 메세지 출력
                exit(1); // 프로그램 종료
        } else { // 만약 큐가 가득 차 있지 않다면
                queue->rear = (queue->rear + 1) % MAX_SIZE; // 큐의 rear(출구)의 인덱스 값을 1늘리고 최대값 초과 방지를 위한 MAX_SIZE 모듈러 연산 진행
                queue->data[queue->rear] = data; // 매개변수 데이터를 queue의 데이터에 삽입
        }
}

Datatype pop_queue(Queue* queue){ // queue의 pop
        if(is_queue_empty(queue)){ // 만약 큐가 비어있다면
                printf("공백 큐입니다! 삭제가 불가능합니다."); // 삭제 불가 메세지 출력
                exit(1); // 프로그램 종료
        } else { // 큐가 비어있지 않다면
                queue->front = (queue->front + 1) % MAX_SIZE; // 큐의 front(입구)의 인덱스 값을 1늘리고 최대값 초과 방지를 위한 MAX_SIZE 모듈러 연산 진행
                return queue->data[queue->front]; // 해당 front(출구) 위치 인덱스의 값을 리턴
        }
}

int meeting(){
        Queue queueMale; // 남자 큐 생성
        init_queue(&queueMale); // 해당 큐 초기화

        Queue queueFemale; // 여자 큐 생성
        init_queue(&queueFemale); // 해당 큐 초기화

        People people; // 입력받을 사람의 데이터

        while (1) { // 무한 반복
                printf("이름을 입력하세요 ('stop'을 입력하면 종료): "); // 안내 멘트
                scanf("%s", people.name); // 이름 입력받기

                if (strcmp(people.name, "stop") == 0) { // 입력받은 이름이 stop이라면
                        break; // 반복문 탈출
                }
                getchar(); // 개행 문자 제거

                printf("성별을 입력하세요 (f or m): "); // 안내 멘트
                scanf("%c", &people.gender); // 성별 입력받기
                getchar(); // 개행 문자 제거

                // 성별 매칭 로직
                if(people.gender == 'f'){ // 입력받은 사람의 성별이 여자라면
                        // 남자 큐 확인
                        if(is_queue_empty(&queueMale)){ // 남자 대기 큐가 비었다면
                                push_queue(&queueFemale, people); // 여자 대기 큐에 해당 사람 집어넣기
                        } else { // 남자 대기 큐가 비지 않았다면
                                People male = pop_queue(&queueMale); // 남자 대기 큐에서 사람을 pop해서 저장하기
                                printf("커플이 탄생했습니다! %s와 %s\n", people.name, male.name); // 그 사람과 매칭되었음을 출력
                        }
                } else if(people.gender == 'm') { // 입력받은 사람의 성별이 남자라면
                        // 여자 큐 확인
                        if(is_queue_empty(&queueFemale)){ // 여자 대기 큐가 비었다면
                                push_queue(&queueMale, people); // 남자 대기 큐에 해당 사람 집어넣기
                        } else { // 여자 대기 큐가 비지 않았다면
                                People female = pop_queue(&queueFemale); // 여자 대기 큐에서 사람을 pop해서 저장하기
                                printf("커플이 탄생했습니다! %s와 %s\n", people.name, female.name); // 그 사람과 매칭되었음을 출력
                        }
                }
        }
}

int main(){
        meeting(); // meeting 함수 실행
```

```
        return 0; // 프로그램 종료
}
```

출력 화면 및 설명



~~글자가 하나씩 더 입력된 부분은 한글 불완전 입력 후 자우면 발생하는 표시 오류이므로... 무시해주시면 감사하겠습니다...~~

일단 회전 큐에 대해서 정의해보겠습니다..
큐라는 자료 구조를 그냥 배열로 구현하면 rear에서 삭제, front에서 삽입과 같은 행동을 여러번 반복하면 배열 자체가 배열의 크기에서 밀려나거나 배열 안의 범위에 채우기 위해서 전체 데이터를 n번 옮기는 등의 비효율적인 일이 발생합니다.
이러면 큐를 쓰는 이유가 없어지는데 이 단점을 보완하기 위해서 회전 큐라는 개념을 사용합니다.

간단하게 개념만 살펴보면 회전 큐는 배열로 만든 큐이나 배열의 끝과 시작을 연결한,

위 코드에서는 100의 크기의 배열을 선언하고 front와 rear가 100을 넘어가면 다시 0번으로 이동하면서 움직이는 데이터의 양이 배열의 크기만 넘어가지 않는다면 삽입과 삭제가 O(1)의 상큼한 시간 복잡도로 큐의 규칙을 지키면서 작동하는 큐가 완성됩니다.

실제 구현도 이름과 성별을 저장할 구조체 people형 구조체를 선언하고,

queue형 구조체를 선언하고, 데이터의 출구(front), 데이터의 입구(rear), 의 위치의 인덱스를 저장할 변수들을 선언하고 people형 자료를 저장할 수 있는 배열을 선언하였습니다.

이후 초기화 함수 init_queue는 queue형 포인터를 직접 받아 그 queue의 front와 rear의 0으로 조작하는 모습을 볼 수 있으며,

is_queue_full, is_queue_empty에서 같은 곳에서 큐의 빈 상황과 큐가 가득 찬 모습을 어떤 식으로 판단하는지 보면 알 수 있습니다.
아래 코드는 is_queue_full 의 코드입니다.

```
return ((queue->rear + 1) % MAX_SIZE) == queue->front;
```

이처럼 위와 같은 코드를 보았을 때 원형 큐는 데이터의 수가 배열의 크기만 넘어가지 않는다면 인덱스의 번호가 어떻게 되든 작동해야 합니다.

맨 처음 시작과 끝이 이어져야 하는 것인데 바로 이 점에서 원형 큐의 특징이 드러납니다.

예시를 하나 들어보겠습니다.
큐가 가득 차 있을 때이고, 큐의 조건은 위의 소스코드와 같으며,
큐의 삽입과 삭제를 계속 반복해서 실제 데이터는 100개가 존재하며 인덱스 넘버는 각각 front == 86, rear == 85인 상황을 가정해 봅시다.

이때 is_queue_full의 공식에 따라 rear에 +1을 더하고 모듈러 연산을 취해도 여전히 front와 같은 86이 됩니다. 100을 넘지 않으므로 모듈러 연산을 해도 그대로인 것입니다.

그러나 이런 상황을 가정해 봅시다.

front == 0, rear는 99인 상황인 것입니다. 이때 모듈러 연산 없이 rear에 1을 더하게 되면 100이 되고 이는 front와 같지 않은데다가 배열의 범위도 초과해 버립니다.

즉 원형 큐는 절대로 100을 넘지 않게 조작하면서 배열을 일종의 원형 연결 구조로 생각하고 사용하는 것입니다.
이것이 큐가 비어있는지 여부, 가득 차 있는지 여부, 큐의 요소 삽입, 큐의 요소 삭제 등에서 모듈러 연산을 활용하는 이유이고 이것이 원형 큐를 구현할 수 있게 하는 핵심 역할인 것입니다.

3.1 원형 큐로 구현

코드 및 주석

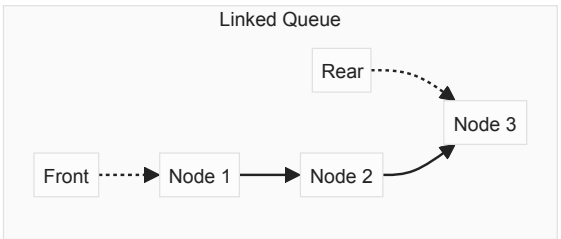
```c
#define _CRT_SECURE_NO_WARNINGS // scanf_s를 사용하지 않음으로 발생하는 오류 해결

#include <stdio.h> // 기본 입출력을 위한 헤더파일
#include <stdlib.h> // 노드의 동적 할당을 위한 malloc을 사용하기 위한 헤더파일
#include <stdbool.h> // is_queue_empty 함수의 반환값을 bool형으로 하기 위한 헤더파일
#include <string.h> // 입력한 문자열이 stop인지 확인하기 위해 비교를 위한 strcmp를 사용하기 위한 헤더파일

#define Datatype People // Datatype이라는 글자를 모두 People로 바꿔서 컴파일 할 것을 선언하는 전처리기

typedef struct People { // 입력받을 사람에 대한 구조체 선언
	char name[31]; // 그 사람의 이름
	char gender; // 그 사람의 성별
} People; // People형 자료형으로 부를 것을 정의

typedef struct Node { // 노드의 정의
	Datatype data; // 데이터
	struct Node* link; // 다음 노드
} Node; // Node형 자료형으로 부를 것을 정의

typedef struct { // 큐의 정의
	Node* front; // 출구
	Node* rear; // 입구
	int count; // 큐의 개수
} Queue; // Queue형 자료형으로 부를 것을 정의

void init_queue(Queue* queue) { // 큐의 초기화 함수 큐의 주소를 매개변수로 받는다
	queue->front = queue->rear = NULL; // 해당 큐의 front(데이터의 출구)와 rear(데이터의 입구)가 될 노드를 가리키는 포인터를 초기화
	queue->count = 0; // 큐가 비어있음을 의미하는 0으로 카운트 초기화
}

bool is_queue_empty(Queue* queue) { // 해당 큐가 차있으면 true 차있지 않으면 false를 리턴하는 함수
	return queue->count == 0; // count가 0이면 true 그렇지 않으면 false를 리턴
}
// 직접 조건문에 쓰는게 효율적입니다만 보기 쉽게 하기 위해 다음과 같은 함수를 정의하였습니다.

Node* create_node(Datatype data) { // 새로운 노드 생성
	Node* newnode = (Node*)malloc(sizeof(Node)); // Node형 newnode에 Node형 데이터를 동적 할당
	if (!newnode) { // 동적 할당 실패시
		return NULL; // 널 포인터 반환
	}
	newnode->data = data; // 파라미터로 받은 데이터를 newnode의 데이터에 집어넣기
	newnode->link = NULL; // newnode의 link를 널포인터로 초기화
	return newnode; // 새로 동적 할당한 노드 리턴
}

void push_queue(Queue* queue, Datatype data) { // 큐의 주소와 데이터(peaple형 데이터)를 매개변수로 받습니다.
	Node* newnode = create_node(data); // create_node 함수로 노드를 동적 생성하고 파라미터로 data를 넘겨주어 data를 포함하게 하고 그 동적 생성된 노드
의 주소를 newnode에 넘겨 주었습니다.
	if (newnode == NULL) { // 동적
		printf("Memory allocation failed\n"); // 할당 실패 메세지를 출력하고
		return; // 함수 종료
	}
	if (is_queue_empty(queue)) { // 만약 큐가 비어있다면(is_queue_empty 로직으로 큐 비었는지 판단)
		queue->front = newnode; // 큐의 front(출구)를 newnode로
	} else { // 큐가 비어있지 않다면
		queue->rear->link = newnode; // 큐의 rear가 가리크는 노드의 링크가 newnode를 가리키게 함
	}
	queue->rear = newnode; // 큐의 rear가 newnode를 가리키게 함
	queue->count++; // 큐의 카운트 1 증가
}

Datatype pop_queue(Queue* queue) { // 큐의 주소를 매개변수로 받습니다.
	Node* popNode; // pop할 노드의 주소를 미리 빼두기 위한 변수 popNode 선언
	Datatype popData; // pop할 노드의 데이터를 미리 빼두기 위한 변수 popData 선언
	if (is_queue_empty(queue)) { // 만약 큐가 비어있다면(is_queue_empty 로직으로 큐 비었는지 판단)
		printf("queue underflow\n"); // 큐가 비어있음을 알리고
		exit(1); // 프로그램 종료
	} else { // 큐가 비어있지 않다면
		popNode = queue->front; // queue의 front(출구)가 나갈 노드이므로 그 노드의 주소를 popNode에 저장
		popData = popNode->data; // popNode에서 데이터 빼내서 popData에 저장
		queue->front = popNode->link; // pop할 노드의 다음 링크를 front가 가리키게 하고
		free(popNode); // 동적 할당한 popNode가 가리키는 동적 메모리를 동적 할당 해제합니다.
		queue->count--; // 큐의 카운트 1 감소
		return popData; // 아까 빼둔 데이터를 리턴합니다.
	}
}
```

```c
int meeting(){
        Queue queueMale; // 남자 큐 생성
        init_queue(&queueMale); // 해당 큐 초기화

        Queue queueFemale; // 여자 큐 생성
        init_queue(&queueFemale); // 해당 큐 초기화

        People people; // 입력받을 사람의 데이터

        while (1) { // 무한 반복
                printf("이름을 입력하세요 ('stop'을 입력하면 종료): "); // 안내 멘트
                scanf("%s", people.name); // 이름 입력받기

                if (strcmp(people.name, "stop") == 0) { // 입력받은 이름이 stop이라면
                        break; // 반복문 탈출
                }
                getchar(); // 개행 문자 제거

                printf("성별을 입력하세요 (f or m): "); // 안내 멘트
                scanf("%c", &people.gender); // 성별 입력받기
                getchar(); // 개행 문자 제거

                // 성별 매칭 로직
                if(people.gender == 'f'){ // 입력받은 사람의 성별이 여자라면
                        // 남자 큐 확인
                        if(queueMale.front == NULL){ // 남자 대기 큐가 비었다면
                                push_queue(&queueFemale, people); // 여자 대기 큐에 해당 사람 집어넣기
                        } else { // 남자 대기 큐가 비지 않았다면
                                People male = pop_queue(&queueMale); // 남자 대기 큐에서 사람을 pop해서 저장하기
                                printf("커플이 탄생했습니다! %s와 %s\n", people.name, male.name); // 그 사람과 매칭되었음을 출력
                        }
                } else if(people.gender == 'm') { // 입력받은 사람의 성별이 남자라면
                        // 여자 큐 확인
                        if(queueFemale.front == NULL){ // 여자 대기 큐가 비었다면
                                push_queue(&queueMale, people); // 남자 대기 큐에 해당 사람 집어넣기
                        } else { // 여자 대기 큐가 비지 않았다면
                                People female = pop_queue(&queueFemale); // 여자 대기 큐에서 사람을 pop해서 저장하기
                                printf("커플이 탄생했습니다! %s와 %s\n", people.name, female.name); // 그 사람과 매칭되었음을 출력
                        }
                }
        }
}

int main(){
        meeting(); // meeting 함수 실행
        return 0; // 프로그램 종료
}
```

출력 화면 및 설명

여기는 조금 다릅니다. 저번 연결 리스트를 보면 아시겠지만, 데이터와 다른 노드를 가리키는 포인터를 담을 수 있는 '노드' 와 아까 문제에서 정의한 people을 정의하고 노드는 people을 담을 수 있게 정의하였습니다.

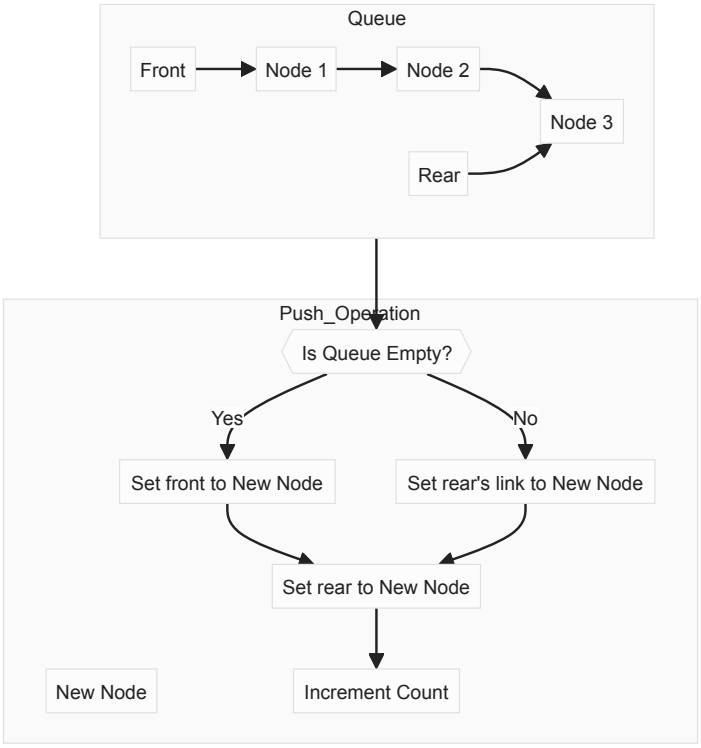
전체적인 구조는 다음과 같이 형성됩니다.



그림이 조금 이상하긴 하지만 연결 큐 구조체 요소인 front와 rear가 각각 pop push 로직 실행시 처리할 입구와 출구를 가리키고 있는 모습이고 front부터 시작해서 node를 전체적으로 순회할 수 있도록 로직이 구성되는 모습을 볼 수 있습니다.

왜 이렇게 작성했는지는 추후 설명드리겠습니다.

push



push queue는 입구인 rear를 다루는 로직입니다.

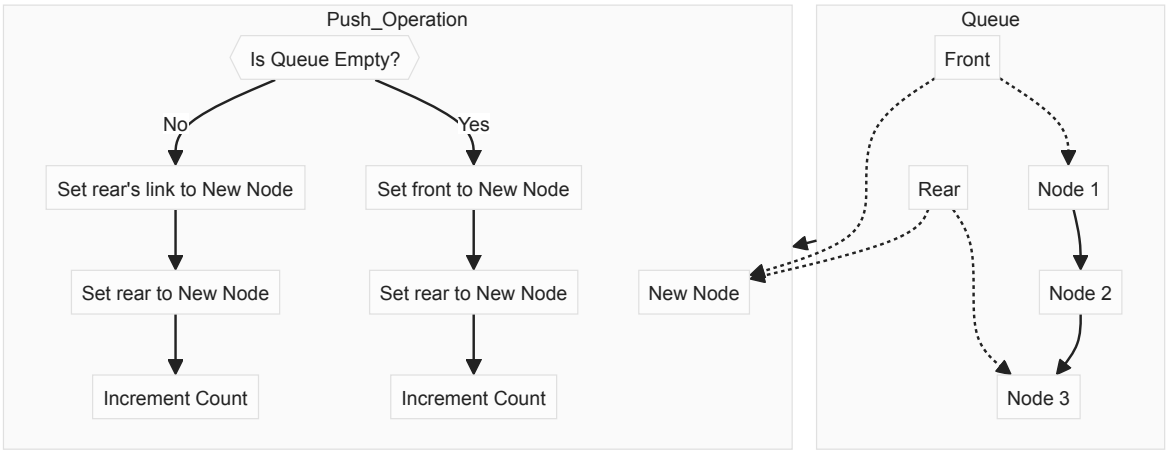
당연히 새로운 데이터를 집어넣는 로직이므로 큐의 주소와 데이터를 매개변수로 받아 주었습니다.

맨 앞쪽 노드를 생성하고 동적할당 실패시 프로그램 종료하는 로직은 뒤로 하고

문제에서 2가지 경우에 대해서 다뤄주었는데 제일 먼저 큐의 삽입입니다.

큐의 삽입은 크게 2가지 경우의 수로 나누어지는데, 큐가 비어있을때와 그렇지 않을 때입니다.

큐가 비어있을때의 로직은



큐가 비어있을때는 새로 들어온 데이터가 바로 나갈 데이터이므로 front가 newnode의 주소를 가리키게 합니다 그리고 rear 역시 비어있었으므로 새로 들어오는 노드인 newnode를 가리킵니다

큐 안에 내용물이 있을 경우에는 큐의 마지막이었던 노드가 newnode의 주소를 가리키게 하고 rear가 newnode를 가리키게 하면 작동합니다.

그러나 이 둘의 마지막 로직인 큐의 rear가 newnode를 가리키게 한다는 로직은 중복되므로 바깥쪽 순서로 빼내고 큐의 카운트를 1 증가시키는 것으로 해당 로직의 구성을 완료했습니다.

pop
pop의 로직입니다.

역시 큐가 비어있을때
큐가 비어있지 않을때를 상정하여 경우의 수를 나눕니다.

만약 큐가 비어있을때 pop을 시도하면 문제가 발생하므로 큐가 비었을때의 pop을 오류처리하여 프로그램 종료를 시켜주었고 큐가 차있을때의 로직을 순서대로

1. 큐의 front가 나갈 노드이므로 그 노드의 주소를 미리 popNode로 저장해두고
2. 아까의 popNode에서 데이터를 빼내서 popData에 미리 저장해둡니다.
3. 이후 메인 이동 로직을 실시하는데, pop할 노드의 다음 링크를 front가 가리키게 하고
4. 동적 할당한 popNode가 가리키는 동적메모리가 node이르몰 이를 동적 할당 해제합니다.
5. 이후 큐의 카운터를 줄이고
6. 아까 빼둔 데이터를 리턴합니다.
   그리고 이 로직은 큐의 데이터가 1개일때와 여러개일때의 경우 모두 정상적으로 작동하는것을 확인했습니다.

이후 메인 meeting로직은 위의 원형 큐와 소름돋을정도로 비슷합니다.(ADT 자체를 똑같은 것을 사용하므로)

다만 큐가 비었는지 확인하는 로직은 길이가 그렇게 길지 않으므로 함수로 따로 빼는것이 비효율적이라 판단하여 직접 써준것만 조금 다릅니다...