* 1. **StackOperations**

#include <stdio.h>

#include <stdlib.h>

struct Stack {

    int top; // stack을 다루기 위한 top

    int size; // scanf로 받아준 int n의 값을 저장

    int\* array;

};

// 지정된 크기의 stack 생성

struct Stack\* createStack(int size) {

    struct Stack\* stack = (struct Stack\*)malloc(sizeof(struct Stack));

    stack->size = size;

    stack->top = -1;

    stack->array = (int\*)malloc(stack->size \* sizeof(int));

    return stack;

}

// stack이 가득 차 있는 경우를 top = size-1으로 구별, overflow 방지

int isFull(struct Stack\* stack) {

    return stack->top == stack->size - 1;

}

// stack이 비어있는 경우를 top = -1으로 구별, underflow 방지

int isEmpty(struct Stack\* stack) {

    return stack->top == -1;

}

// push operation new element를 넣어줌

void push(struct Stack\* stack, int item) {

    if (isFull(stack))

        return;

    stack->array[++stack->top] = item;

}

// pop operation 가장 위의 element 제거

int pop(struct Stack\* stack) {

    if (isEmpty(stack))

        return -1;

    return stack->array[stack->top--];

}

// 순차적인 operation을 보여주기 위한 출력 함수

void printStack(struct Stack\* stack) {

    if (isEmpty(stack)) {

        printf("Stack is empty.\n");

        return;

    }

    printf("Stack contents: ");

    for (int i = stack->top; i >= 0; i--)

        printf("%d ", stack->array[i]);

    printf("\n");

}

int main() {

    int n; // 원하는 stack 크기 이 문제에서는 3

    printf("Enter stack size: ");

    scanf("%d", &n); // int input을 받아서 n에 저장

    struct Stack\* stack = createStack(n); // stack 생성

    printf("Stack created with size %d.\n", n);

    push(stack, 6);

    printStack(stack);

    push(stack, 4);

    printStack(stack);

    push(stack, 2);

    printStack(stack);

    pop(stack);

    printStack(stack);

    push(stack, 10);

    printStack(stack);

    return 0;

}

**결과**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

Step1. Stack을 만드는 함수

Empty Queue 생성 with given size

Step2. Push Pop operation 함수

Push로 삽입, 이 문제에서는 Top을 제거하는 Pop operation만으로 2를 제거 가능

Step3. 각 단계별 operation을 보여주기 위한 출력 함수

Feedback

1. Malloc() 함수가 memory allocation 실패하는 경우 고려한 디버깅

Ex) exit로 allocation failure 고려

Stack\* createStack(int size){

If (stack==NULL) {

Printf()

Exit(1);

…

return stack;

1. Typedef를 사용하는 것이 조금 더 코드를 깔끔하게 연습하기!

Structure에 비해서 가독성이 좋다.

1. Switch statement같은 기능으로 input의 validation 검증

Ex)

Switch (n) {

case 0: // size can not be 0

case 1:// size must be greater than 1

default:

break;

}

* 1. **CircularQueueOperations**

#include <stdio.h>

#include <stdlib.h>

// front,rear,size를 통한 queue의 선언

struct CircularQueue {

    int front, rear;

    int size;

    char\* array;

};

// scanf로 받아준 크기의 circular queue

struct CircularQueue\* createQueue(int size) {

    struct CircularQueue\* queue = (struct CircularQueue\*) malloc(sizeof(struct CircularQueue));

    queue->front = -1;

    queue->rear = -1;

    queue->size = size;

    queue->array = (char\*) malloc(queue->size \* sizeof(char));

    return queue;

}

// queue가 가득 찬 경우를 구별, overflow 방지

int isFull(struct CircularQueue\* queue) {

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

}

// queue가 비어있는 경우를 구별, underflow 방지

int isEmpty(struct CircularQueue\* queue) {

    return queue->front == -1;

}

// enqueue operation 함수 char datatype element 삽입

void enqueue(struct CircularQueue\* queue, char item) {

    if (isFull(queue))

        return;

    if (isEmpty(queue))

        queue->front = 0;

    queue->rear = (queue->rear + 1) % queue->size;

    queue->array[queue->rear] = item;

}

// dequeue operation 함수 char datatype element 삭제

char dequeue(struct CircularQueue\* queue) {

    if (isEmpty(queue))

        return -1;

    char item = queue->array[queue->front];

    if (queue->front == queue->rear)

        queue->front = queue->rear = -1;

    else

        queue->front = (queue->front + 1) % queue->size;

    return item;

}

// operation의 확인을 위한 출력 함수

void printQueue(struct CircularQueue\* queue) {

    if (isEmpty(queue)) {

        printf("Queue is empty.\n");

        return;

    }

    printf("Queue contents: ");

    int i;

    for (i = queue->front; i != queue->rear; i = (i + 1) % queue->size)

        printf("%c ", queue->array[i]);

    printf("%c\n", queue->array[i]);

}

// 특정 element를 찾고, 이를 다른 element로 교체하는 함수

void replaceElement(struct CircularQueue\* queue, int removeElement, int newElement) {

    int i;

    for (i = queue->front; i != queue->rear; i = (i + 1) % queue->size) {

        if (queue->array[i] == removeElement) {

            queue->array[i] = newElement;

            return;

        }

    }

    if (queue->array[i] == removeElement)

        queue->array[i] = newElement;

}

int main() {

    int size;

    printf("Enter queue size: ");

    scanf("%d", &size);

    struct CircularQueue\* queue = createQueue(size);

    printf("Queue created with size %d.\n", size);

    enqueue(queue, 'A');

    enqueue(queue, 'B');

    enqueue(queue, 'C');

    enqueue(queue, 'D');

    printQueue(queue);

    replaceElement(queue, 'A', 'E');

    printQueue(queue);

    return 0;

}

**결과**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Step1. Circular Queue를 만드는 함수

Empty Queue 생성

Enqueue를 통한 A B C D 삽입

Step2. Input을 통해서 특정 element를 교체하는 replaceElement 함수

A를 E로 교체

Step3. 수행한 operation을 보여주기 위한 출력 함수

Feedback

1. Replace function에서 첫 element만 교체 즉 A 가 여러 개인 경우에 첫 번째 A만 교체 함수의 수정 필요 이 문제에서는 문제가 없는 함수
2. Input 검증 코드 char int 등 invalid input에 대한 오류 검증
3. Enqueue 함수와 dequeue 함수의 error 검증 코드