

C - 자료 구조 1

Data Structure



자료 구조

■ 자료 구조

자료구조는 프로그래밍에서 데이터를 효율적으로 저장하고, 접근하고, 관리하기 위한 구조를 의미합니다.

자료 구조	특징	C 언어 구현 방법
스택(Stack)	LIFO(후입선출)	배열 또는 구조체
큐(Queue)	FIFO(선입선출)	배열, 원형 큐, 구조체
연결 리스트	동적 크기, 삽입/삭제 용이	단순, 이중, 원형 연결 리스트
트리	계층 구조(비선형 구조)	구조체, 연결리스트
해시	해시 함수로 변환	배열 및 연결리스트



스택(Stack) 자료 구조

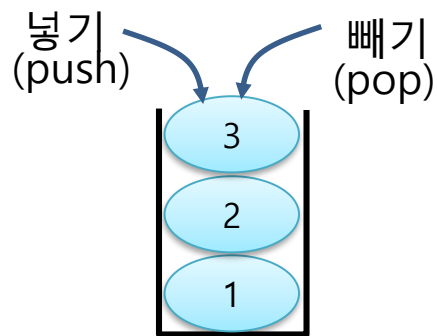
■ 스택(Stack)

- 후입선출(LIFO : Last in First Out) 구조

배열에서 나중에 들어간 자료를 먼저 꺼냄

(응용 예: 스택 메모리, 문자열 역순 처리, 게임 무르기)

메소드명	설명
push()	원소(자료)를 스택에 넣는다.
pop()	스택의 맨 위 원소를 가져온다.

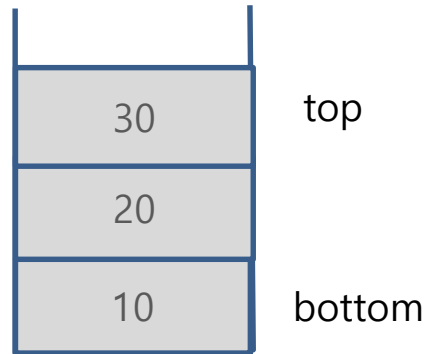
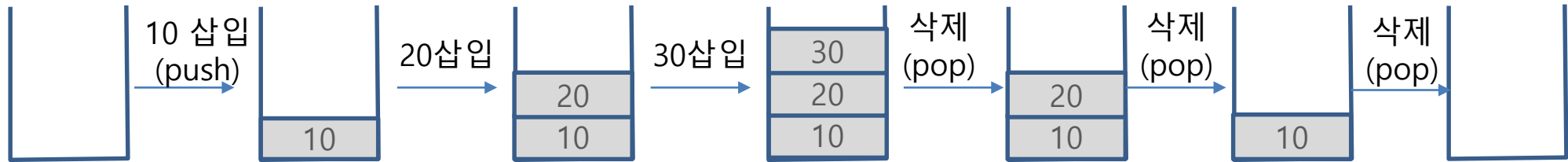


스택(Stack)



스택(Stack)

■ 스택(Stack)



스택(Stack)

- 스택(Stack) – 정수형

```
#define MAX_LEN 4

int stack[MAX_LEN]; // 스택 배열
int top = -1; // 스택의 top을 나타냄

// 요소 추가 함수 (push)
void push(int value) {
    if (top >= MAX_LEN - 1)
        printf("스택이 가득 찼습니다. PUSH 실패: %d\n", value);
    else {
        stack[++top] = value; //선증가
        printf("PUSH: %d\n", value);
    }
}
```



스택(Stack)

- 스택(Stack) - 정수형

```
int pop() { // 요소 제거 함수 (pop)
    if (top < 0) {
        printf("스택이 비어 있습니다. POP 실패\n");
        return -1;
    }
    else
        return stack[top--];
}

int main() {
    puts("=== 스택에 자료 삽입 ===");
    // 값 추가 (push) : 80 - 70 - 95 - 85
    push(80);
    push(70);
    push(95);
    push(85);
    //push(100); // 초과 입력 시 실패
}
```



스택(Stack)

- 스택(Stack) - 정수형

```
puts("\n=== 스택에서 자료 삭제 ===");  
// 값 제거 (pop) : 85 - 95 - 70 - 80  
/*printf("%d\n", stack[top]); //현재 맨 위 요소  
pop();  
printf("%d\n", stack[top]);  
pop();  
printf("%d\n", stack[top]);  
pop();  
printf("%d\n", stack[top]);  
pop();  
printf("%d\n", stack[top]);  
pop(); //빈 상태에서 pop 시도 */  
  
while (top != -1) { //안전한 삭제  
    printf("%d ", stack[top]);  
    pop();  
}
```

=== 스택에 자료 삽입 ===

PUSH: 80

PUSH: 70

PUSH: 95

PUSH: 85

=== 스택에서 자료 삭제 ===

85 95 70 80



스택(Stack)

- 스택(Stack) – 문자형

```
#define MAX_LEN 3

char stack[MAX_LEN];
int top = -1;

void push(char c) { //요소 삽입(저장)
    if (top >= MAX_LEN - 1) {
        printf("스택이 가득 찼습니다.\n");
        return; //return '\0' (널문자, 비어있음)
    }
    stack[++top] = c;
    printf("%c ", stack[top]);
}
```



스택(Stack)

- 스택(Stack) – 문자형

```
char pop() { //요소 삭제(빼기)
    if (top < 0) {
        printf("스택이 비었습니다!!\n");
        return '\0';
    }
    return stack[top--];
}

int main()
{
    //a - b - c
    printf("스택에서 자료 저장\n");
    push('a');
    push('b');
    push('c');
    push("d"); //초과했을 때 처리 확인
}
```



스택(Stack)

- 스택(Stack) – 문자형

```
//c - b - a
printf("스택에서 자료 삭제\n");
/*printf("%c\n", pop());
printf("%c\n", pop());
printf("%c\n", pop());
printf("%c\n", pop()); //비었을 때 처리 확인*/

while (top != -1) { //안전한 삭제
    printf("%c ", stack[top]);
    pop();
}
```

스택에서 자료 저장
a b c 스택이 가득 찼습니다.
스택에서 자료 삭제
c b a



단어 순서 뒤집기

- 스택 구조체 활용

```
#define MAX_LEN 4

// 스택 구조체 정의
typedef struct {
    int data[MAX_LEN]; // 스택 요소 저장
    int top;           // top 인덱스
} Stack;

// 스택 초기화 함수
void initStack(Stack* s) {
    s->top = -1;
}
```



단어 순서 뒤집기

- 스택 구조체 활용

```
void push(Stack* s, int value) {
    if (s->top < MAX_LEN - 1) {
        s->data[++(s->top)] = value;
        printf("PUSH: %d\n", value);
    }
    else {
        printf("스택이 가득 찼습니다. PUSH 실패: %d\n", value);
    }
}

int pop(Stack* s) {
    if (s->top > -1) {
        return s->data[(s->top)--];
    }
    else {
        printf("스택이 비어 있습니다. POP 실패\n");
        return -1;
    }
}
```



단어 순서 뒤집기

- 스택 구조체 활용

```
void printStack(Stack* s) {  
    printf("\n현재 스택 상태:\n");  
    if (s->top == -1) {  
        printf("(비어 있음)\n");  
    }  
    else {  
        printf("남은 요소 수: %d\n", s->top + 1);  
        for (int i = 0; i <= s->top; i++) {  
            printf("%d\n", s->data[i]);  
        }  
    }  
}  
  
int main() {  
    Stack stack;    // 스택 변수 생성  
    initStack(&stack); // 초기화
```

단어 순서 뒤집기

■ 스택 구조체 활용

```
puts("=== 스택에 자료 삽입 ===");  
// 값 추가 (push) : 80 - 70 - 95 - 85  
push(&stack, 80);  
push(&stack, 70);  
push(&stack, 95);  
push(&stack, 85);  
//push(&stack, 100); // 초과 입력 시 실패  
  
puts("\n=== 스택에서 자료 삭제 ===");  
// 값 제거 (pop) : 85 - 95 - 70 - 80  
/*pop(&stack);  
pop(&stack);  
pop(&stack);*/  
//pop(&stack); // 빈 상태에서 pop 시도  
  
while (stack.top != -1) {  
    printf("%d\n", pop(&stack));  
}  
printStack(&stack); // 최종 상태 출력
```

```
=== 스택에 자료 삽입 ===  
PUSH: 80  
PUSH: 70  
PUSH: 95  
PUSH: 85  
  
=== 스택에서 자료 삭제 ===  
85  
95  
70  
80  
  
현재 스택 상태 :  
(비어 있음)
```



단어 순서 뒤집기

- 문자열 뒤집기

```
#define MAX_LEN 128 // 문자열 최대 길이

// 스택 구조체 정의
typedef struct {
    char data[MAX_LEN];
    int top;
} Stack;

// 스택 초기화
void initStack(Stack* s) {
    s->top = -1;
}
```



단어 순서 뒤집기

- 문자열 뒤집기

```
void push(Stack* s, char ch) {  
    if (s->top >= MAX_LEN - 1) {  
        printf("스택이 가득 찼습니다.\n");  
        return;  
    }  
    else  
        s->data[++(s->top)] = ch; //문자 저장  
}  
  
int pop(Stack* s) {  
    if (s->top < 0) {  
        printf("스택이 비어 있습니다.\n");  
        return -1;  
    }  
    else  
        return s->data[(s->top)--]; //문자 반환  
}
```



단어 순서 뒤집기

■ 문자열 뒤집기

```
Stack stack; //스택 구조체 변수 생성
char str[MAX_LEN]; //문자열 배열

initStack(&stack); //초기화 함수 호출

//사용자 입력
printf("문자열 입력: ");
fgets(str, MAX_LEN, stdin); //공백 포함 문자 입력

// 문자열을 한 글자씩 push
for (int i = 0; str[i] != '\0'; i++) {
    push(&stack, str[i]);
}

// pop 하면서 뒤집어 출력
printf("뒤집은 문자열: ");
while (stack.top != -1) {
    printf("%c", pop(&stack));
}
```

```
문자열 입력: hello world
뒤집은 문자열:
dlrow olleh
```

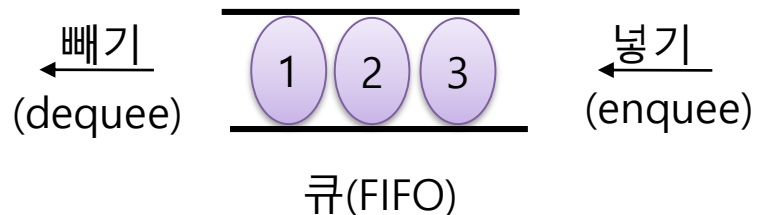


큐(Queue) 자료 구조

- 큐(Queue)

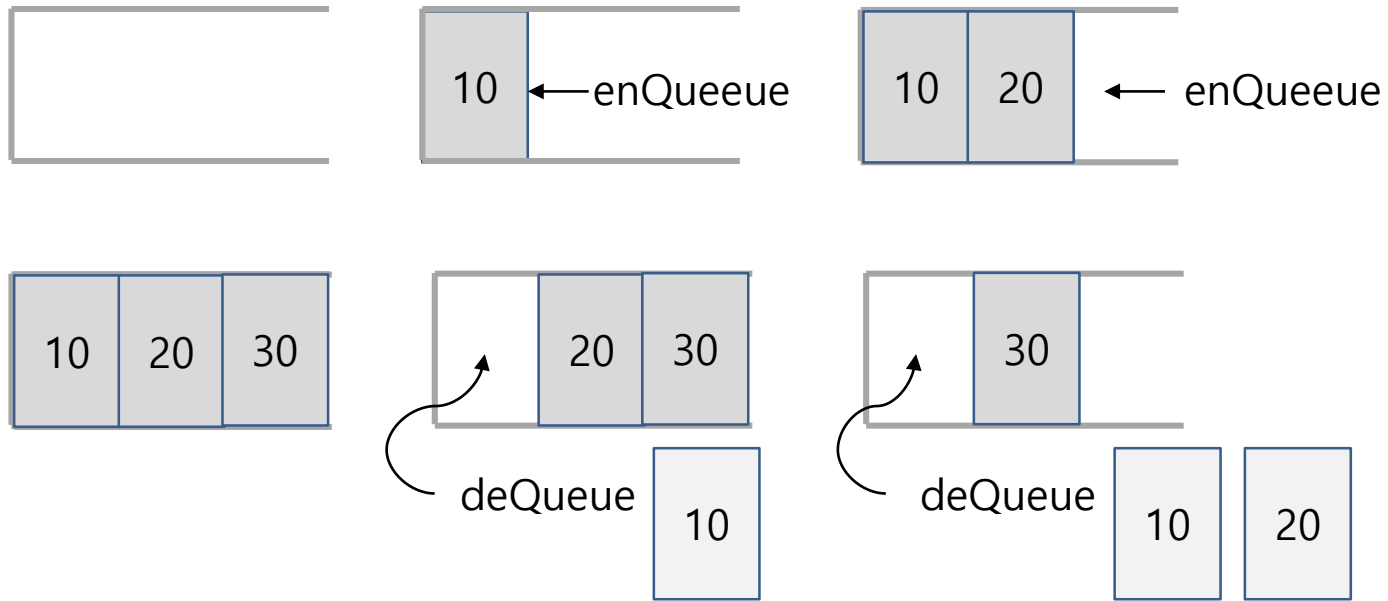
- 선입선출(FIFO : First in First Out) 구조
배열에서 먼저 들어간 자료를 먼저 꺼냄
(응용 예: 프린터 대기열, CPU 스케줄링)

메소드명	설명
enqueue()	주어진 객체를 큐에 넣는다.
dequeue()	객체 하나를 가져온다. 객체를 큐에서 제거한다.



큐(Queue) 자료 구조

- 큐(Queue)



큐(Queue) 자료 구조

- 원형 큐 상태 변화

1. 초기 상태(비어 있음)

[][][][] front=0 rear=0
^front
^rear

2. enqueue("A")

[A][][][] front=0 rear=1
^front
^rear

3. enqueue("B")

[A][B][][] front=0 rear=2
^front
^rear

MAX_QUEUE = 4

front: 데이터 꺼낼 위치

rear: 데이터 넣을 위치

비어있음: front == rear

가득 참: (rear + 1) % MAX_QUEUE == front

rear는 (rear+1) % MAX_QUEUE,

front는 (front+1) % MAX_QUEUE로 이동하면서
배열을 원처럼 사용하는 것이 원형 큐의 핵심이다.
front는 배열 끝에 도달하면 다시 0으로 돌아오게
연산함



큐(Queue) 자료 구조

- 원형 큐 상태 변화

4. enqueue("C")
[A][B][C][] front=0 rear=3
^front
 ^rear
5. dequeue() → "A" 꺼냄
[][B][C][] front=1 rear=3
 ^front
 ^rear
6. enqueue("D")
[][B][C][D] front=1 rear=0(회전)
 ^front
 ^rear

7. enqueue("E") → 불가능 (가득 참)
큐가 가득 찼습니다!

8. dequeue() → "B" 꺼냄
[][C][D] front=2 rear=0
 ^front
 ^rear

원형 큐에서 MAX_QUEUE가 4이라면, 실제로 저장할 수 있는 원소 개수는 3개이다.
(왜냐하면 front==rear일 때 "빈 상태"와 구분해야 하므로 한 칸은 항상 비워둔다.)



원형 큐(Queue)

- 배열 기반 원형 큐(Queue)

```
#define MAX_QUEUE 4 //큐의 최대 크기

//전역 변수 선언
int queue[MAX_QUEUE];
int front = 0; //데이터 꺼낼 위치
int rear = 0;  //데이터 넣을 위치

void enqueue(int x) {
    if ((rear + 1) % MAX_QUEUE == front) {
        printf("큐가 가득 찼습니다.\n");
        return;
    }
    queue[rear] = x; //뒤에서 데이터 넣기
    rear = (rear + 1) % MAX_QUEUE; //rear를 다음 위치로 이동
    //printf("front=%d, rear=%d, x=%d\n", queue[front], queue[rear], x);
    printf("%d ", x);
}
```



원형 큐(Queue)

- 배열 기반 원형 큐(Queue)

```
int deQueue() {
    if (front == rear) {
        printf("큐가 비었습니다.\n");
        return -1;
    }
    int data = queue[front]; //앞(front) 데이터를 저장
    front = (front + 1) % MAX_QUEUE; //front 다음 위치로 이동
    return data; //데이터를 내보냄
}

int main()
{
    puts("=== 큐에 데이터 넣기 ===");
    enqueue(10);
    enqueue(20);
    enqueue(30);
    //enqueue(40); //오류, 큐가 가득 참
}
```



원형 큐(Queue)

- 배열 기반 원형 큐(Queue)

```
puts("\n=== 큐에서 데이터 꺼내기 ===");
int val; //큐에서 꺼낸 값(데이터)

/*val = dequeue();
printf("%d ", val);
|
val = dequeue();
printf("%d ", val);
|
val = dequeue();
printf("%d ", val);
|
val = dequeue();
printf("%d ", val); */ // 비었을 때 처리 확인

while (front != rear) {
    val = dequeue();
    printf("%d ", val);
}
```

```
=== 큐에 데이터 넣기 ===
10 20 30
=== 큐에서 데이터 꺼내기 ===
10 20 30
```



원형 큐(Queue)

- 고객 대기열 큐(Queue)

```
#define MAX_QUEUE 4
#define NAME_LEN 20

/*
queue[MAX_QUEUE][NAME_LEN]; //이차원 배열
queue[4][20] = {"고객A", "고객B", "고객C", ""};
...
queue[0] → {'고','객','A','\0', ... } // 고객A
queue[1] → {'고','객','B','\0', ... } // 고객B
queue[2] → {'고','객','C','\0', ... } // 고객C
queue[3] → { ?, ?, ?, ... }         // 비어있음
...
*/

// 큐 정의
char queue[MAX_QUEUE][NAME_LEN];
int front = 0;
int rear = 0;
```



원형 큐(Queue)

- 고객 대기열 큐(Queue)

```
// 큐가 비었는지 확인
int isEmpty() {
    return front == rear;
}

// 큐가 가득 찼는지 확인
int isFull() {
    return (rear + 1) % MAX_QUEUE == front;
}

void enqueue(const char* name) { //고객 추가
    if (isFull()) {
        printf("큐가 가득 찼습니다!\n");
        return;
    }
    strcpy(queue[rear], name); //뒤에 고객 이름 저장(복사)
    rear = (rear + 1) % MAX_QUEUE;
}
```



원형 큐(Queue)

- 고객 대기열 큐(Queue)

```
int deQueue(char* name) { // 고객 꺼내기
    if (isEmpty()) {
        printf("큐가 비었습니다!\n");
        name[0] = '\0'; //name을 빈 문자로 초기화 {'고' '객' 'A' '\0'}
        return -1; //실패
    }
    strcpy(name, queue[front]); //앞의 이름을 복사
    front = (front + 1) % MAX_QUEUE;
    return 0; //성공
}
```



원형 큐(Queue)

- 고객 대기열 큐(Queue)

```
char name[NAME_LEN];

// 고객 대기열 추가
enqueue("고객A");
enqueue("고객B");
enqueue("고객C");

// 대기열 처리
while (!isEmpty()) {
    dequeue(name);
    printf("%s님 업무 처리 중...\n", name);
}

printf("모든 고객의 업무가 완료되었습니다.\n");
```

고객A님 업무 처리 중...
고객B님 업무 처리 중...
고객C님 업무 처리 중...
모든 고객의 업무가 완료되었습니다.



원형 큐(Queue)

- 고객 대기열 큐(Queue) – 포인터 배열로 정의

```
#include <stdbool.h>

#define MAX_QUEUE 4
#define NAME_LEN 20

// 큐 정의 (포인터 배열)
char* queue[MAX_QUEUE];
int front = 0;
int rear = 0;

// 큐가 비었는지 확인
bool isEmpty() {
    return front == rear;
}

// 큐가 가득 찼는지 확인
bool isFull() {
    return (rear + 1) % MAX_QUEUE == front;
}
```



원형 큐(Queue)

- 고객 대기열 큐(Queue) – 포인터 배열로 정의

```
// 고객 추가
void enqueue(const char* name) {
    if (isFull()) {
        printf("큐가 가득 찼습니다!\n");
        return;
    }
    // 문자열 길이만큼 메모리 동적 할당
    queue[rear] = (char*)malloc(strlen(name) + 1); //'\\0' 포함
    if (queue[rear] == NULL) {
        printf("메모리 할당 실패!\n");
        exit(1);
    }
    strcpy(queue[rear], name); // 문자열 복사
    rear = (rear + 1) % MAX_QUEUE;
}
```



원형 큐(Queue)

- 고객 대기열 큐(Queue) – 포인터 배열로 정의

```
// 고객 꺼내기
int dequeue(char* name) {
    if (isEmpty()) {
        printf("큐가 비었습니다!\n");
        name[0] = '\0';
        return -1;
    }
    strcpy(name, queue[front]); // 문자열 복사
    free(queue[front]);        // 메모리 해제
    queue[front] = NULL;       // 안전하게 초기화
    front = (front + 1) % MAX_QUEUE;
    return 0;
}
```



원형 큐(Queue)

- 구조체 기반 원형 큐(Queue)

```
#define QUEUE_SIZE 10

// 구조체 정의
typedef struct {
    int data[QUEUE_SIZE];
    int front;
    int rear;
} CircularQueue;

// 큐 초기화
void initQueue(CircularQueue* q) {
    q->front = 0;
    q->rear = 0;
}
```



원형 큐(Queue)

- 구조체 기반 원형 큐(Queue)

```
// 큐가 비었는지 확인
bool isEmpty(CircularQueue* q) {
    return q->front == q->rear;
}

// 큐가 가득 찼는지 확인
bool isFull(CircularQueue* q) {
    return (q->rear + 1) % QUEUE_SIZE == q->front;
}

// 데이터 추가
void enqueue(CircularQueue* q, int value) {
    if (isFull(q)) {
        printf("큐가 가득 찼습니다!!\n");
        return;
    }
    q->data[q->rear] = value;
    q->rear = (q->rear + 1) % QUEUE_SIZE;
}
```



원형 큐(Queue)

- 구조체 기반 원형 큐(Queue)

```
int dequeue(CircularQueue* q) { // 데이터 삭제
    if (isEmpty(q)) {
        printf("큐가 비었습니다!!\n");
        return -1;
    }
    int value = q->data[q->front];
    q->front = (q->front + 1) % QUEUE_SIZE;
    return value;
}

void printQueue(CircularQueue* q) { // 큐 상태 출력
    printf("큐 상태: ");
    if (isEmpty(q)) {
        printf("(비어 있음)\n");
        return;
    }
    int i = q->front;
    while (i != q->rear) {
        printf("%d ", q->data[i]);
        i = (i + 1) % QUEUE_SIZE;
    }
    printf("\n");
}
```



원형 큐(Queue)

- 구조체 기반 원형 큐(Queue)

```
CircularQueue q1, q2;

// 큐 초기화
initQueue(&q1);
initQueue(&q2);

printf("=== 큐 1에 데이터 넣기 ===\n");
enqueue(&q1, 10);
enqueue(&q1, 20);
enqueue(&q1, 30);
printQueue(&q1);

printf("\n=== 큐 1에서 데이터 빼기 ===\n");
printf("%d ", dequeue(&q1));
printf("%d ", dequeue(&q1));
printQueue(&q1);

printf("\n=== 큐 2 테스트 ===\n");
enqueue(&q2, 100);
enqueue(&q2, 200);
printQueue(&q2);
```

```
=== 큐 1에 데이터 넣기 ===
큐 상태 : 10 20 30

=== 큐 1에서 데이터 빼기 ===
10 20 큐 상태 : 30

=== 큐 2 테스트 ===
큐 상태 : 100 200
```



원형 큐(Queue)

- 헤더 파일 – CircularQueue.h

```
#include <stdbool.h>
#define QUEUE_SIZE 10

//구조체 정의
typedef struct {
    int data[QUEUE_SIZE];
    int front;
    int rear;
}CircularQueue;

void initQueue(CircularQueue* q); //큐 초기화

bool isEmpty(CircularQueue* q); //큐가 비었는지 여부

bool isFull(CircularQueue* q); //큐가 가득 차있는지 여부

void enqueue(CircularQueue* q, int value); //큐에 데이터 넣기

int dequeue(CircularQueue* q); //큐에서 데이터 꺼내기

void printQueue(CircularQueue* q); //큐상태 출력
#endif
```



원형 큐(Queue)

- 구현 파일 – CircularQueue.c

```
#include "CircularQueue.h"
#include <stdio.h>

void initQueue(CircularQueue* q) {
    q->front = 0;
    q->rear = 0;
}

bool isEmpty(CircularQueue* q) {
    return q->front == q->rear;
}

bool isFull(CircularQueue* q) {
    return (q->rear + 1) % QUEUE_SIZE == q->front;
}
```



원형 큐(Queue)

- 구현 파일 – CircularQueue.c

```
void enqueue(CircularQueue* q, int value) {
    if (isFull(q)) {
        printf("큐가 가득 찼습니다.\n");
        return;
    }
    q->data[q->rear] = value;
    q->rear = (q->rear + 1) % QUEUE_SIZE;
}

int dequeue(CircularQueue* q) {
    if (isEmpty(q)) {
        printf("큐가 비었습니다.\n");
        return -1;
    }
    int value = q->data[q->front];
    q->front = (q->front + 1) % QUEUE_SIZE;
    return value;
}
```



원형 큐(Queue)

- 구현 파일 – CircularQueue.c

```
void printQueue(CircularQueue* q) {  
    printf("큐 상태: ");  
    if (isEmpty(q)) {  
        puts("(비어 있음)");  
        return;  
    }  
    int i = q->front; //i = 0  
    while (i != q->rear) {  
        printf("%d ", q->data[i]);  
        i = (i + 1) % QUEUE_SIZE;  
    }  
    printf("\n");  
}
```



원형 큐(Queue)

- 실행 파일 – CircularQueueMain.c

```
#include "CircularQueue.h"

int main()
{
    CircularQueue q1;
    CircularQueue q2;

    initQueue(&q1);
    initQueue(&q2);

    puts("=== 큐1에 데이터 넣기 ===");
    enqueue(&q1, 10);
    enqueue(&q1, 20);
    enqueue(&q1, 30);
    printQueue(&q1);
}
```



원형 큐(Queue)

- 실행 파일 – CircularQueueMain.c

```
puts("=== 큐1에서 데이터 꺼내기 ===");  
printf("%d ", deQueue(&q1));  
printf("%d ", deQueue(&q1));  
printQueue(&q1);  
  
puts("=== 큐2에 데이터 넣기 ===");  
enqueue(&q2, 100);  
enqueue(&q2, 200);  
printQueue(&q2);  
  
return 0;  
}
```

