데구 5장 큐

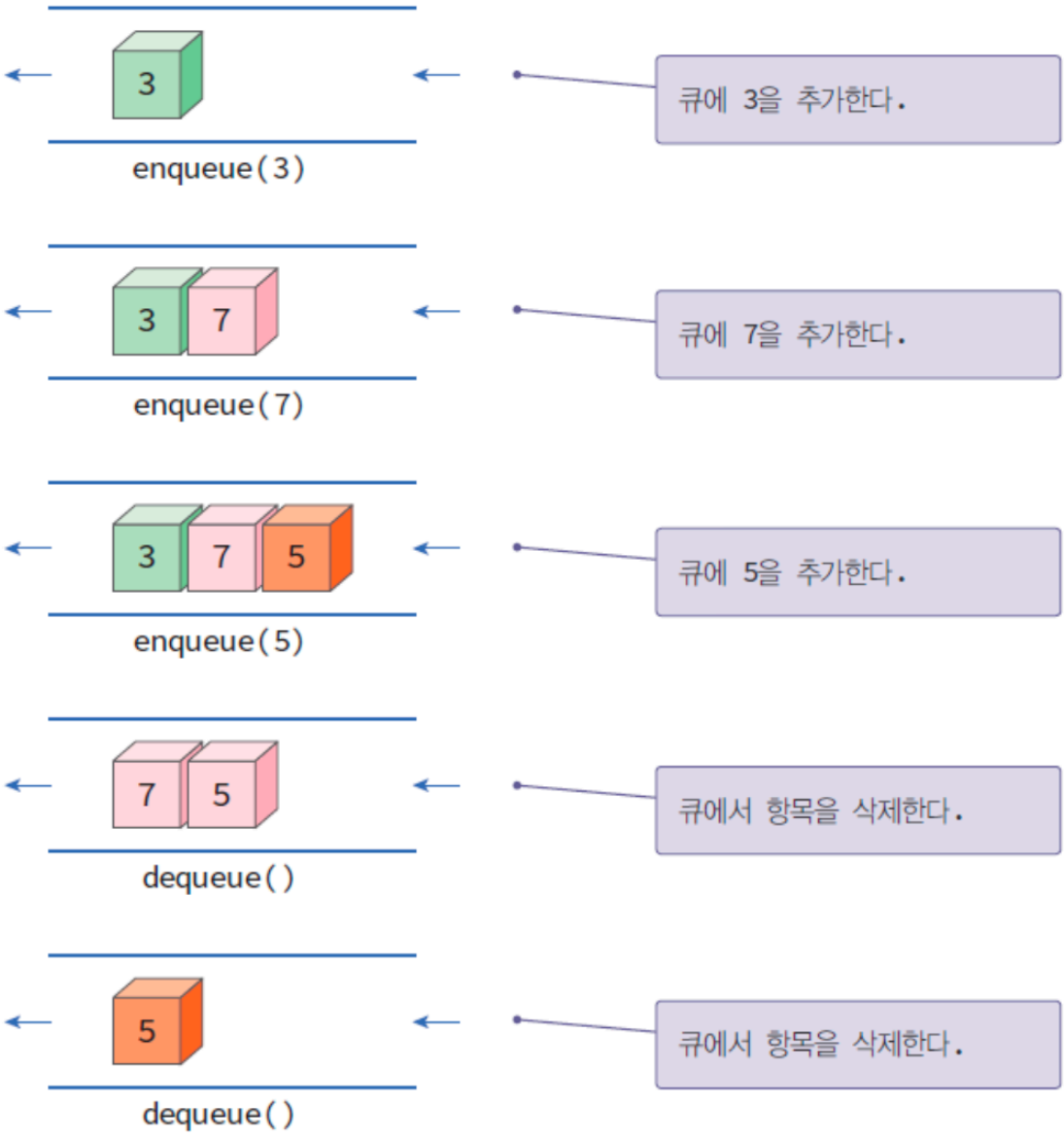
상태	완료
데드라인	@April 16, 2025
PROCESS	💚 데이터구조

▼ Queue

- 먼저 들어온 데이터가 먼저 나가는 자료구조
- First In First Out : FIFO 선입선출

▼ ADT

- create(max_size) : 최대 크기가 맥스 사이즈인 공백 큐를 생성
- init(q) : 큐를 초기화함
- is_empty(q) : `if(size == 0) return TRUE else return FALSE`
- is_full(q) : `if(size == maxx_size) return TRUE else return FALSE`
- enqueue(q, e) : `if(is_full(q)) queue_full 오류; else q의 끝에 e를 추가함`
- dequeue(q) : `if(is_empty(q)) queue_empty 오류; else q맨 앞에 있는 e를 제거하여 반환`
- peek(q) : `if(is_empty(q)) queue_empty 오류; else q맨 앞에있는 e를 읽어 반환함`



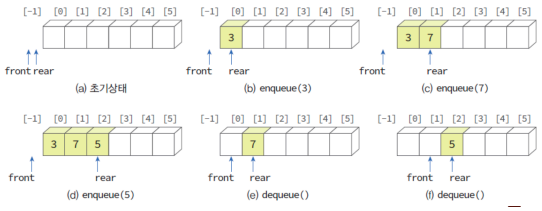
▼ 큐의 응용

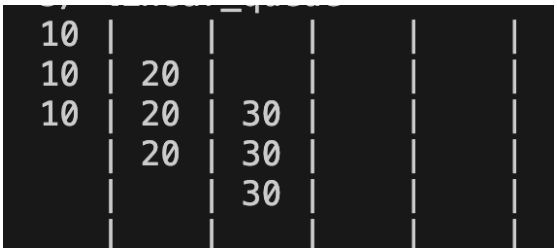
- 직접적인 응용
  - 시뮬레이션의 대기열(공항에서의 비행기들, 은행에서의 대기열)
  - 통신에서의 데이텉 패킷들의 모델링에 이용
  - 프린터와 컴퓨터 사이의 버퍼링
- 간접적인 응용
  - 스택과 마찬가지로 프로그래머의 도구

- 많은 알고리즘에서 사용됨

▼ 선형 큐

- 배열을 선형으로 사용하여 큐를 구현
- 삽입을 계속 하기 위해서는 요소들을 이동시켜야 함






```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5

typedef int element;
typedef struct{
    int front;
    int rear;
    element data[MAX];
}QueueType;

void error(const char *message){
    fprintf(stderr, "%s\n", message);
    exit(1);
}

void init_queue(QueueType *q){
    q->front = -1;
    q->rear = -1;
}
void queue_print(QueueType *q){
    for(int i = 0; i < MAX; i++){
        if(i <= q->front || i > q->rear)
            printf("   | ");
        else
            printf("%d | ", q->data[i]);
    }
```

```
    printf("\n");
}
int is_full(QueueType *q){
    if(q→rear == MAX -1)
        return 1;
    else
        return 0;
}
int is_empty(QueueType *q){
    if(q→front == q→rear)
        return 1;
    else
        return 0;
}
void enqueue(QueueType *q, int item){
    if(is_full(q)){
        error("큐가 포화 상태입니다.");
        return ;
    }
    q→data[++(q→rear)] = item;
}
int dequeue(QueueType *q){
    if(is_empty(q)){
        error("큐가 공백 상태입니다.");
        return -1;
    }
    int item = q→data[++(q→front)];
    return item;
}

int main(){
    int item = 0;
    QueueType q;;
    init_queue(&q);

    enqueue(&q, 10); queue_print(&q);
    enqueue(&q, 20); queue_print(&q);
    enqueue(&q, 30); queue_print(&q);

    item = dequeue(&q); queue_print(&q);
    item = dequeue(&q); queue_print(&q);
```

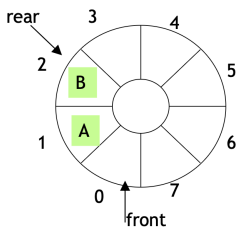
```
    item = dequeue(&q); queue_print(&q);
    return 0;
}
```

▼ 선형 큐의 응용 : 작업 스케줄링

Q[0]	Q[1]	Q[2]	Q[3]	Q[4]	front	rear	설명
					–1	–1	공백 큐
Job#1					–1	0	Job#1이 추가
Job#1	Job#2				–1	1	Job#2이 추가
Job#1	Job#2	Job#3			–1	2	Job#3이 추가
	Job#2	Job#3			0	2	Job#1이 삭제
		Job#3			1	2	Job#2이 삭제

▼ 원형 큐

- 선형 큐의 문제점 : front, rear이 모두 증가하여 배열의 앞부분이 비어 있더라도 사용하지 못함
- 앞부분이 비어 있으면 주기적으로 배열 데이터를 이동시키는 작업이 필요
- 큐의 전단과 후단을 관리하기 위한 2개의 변수 필요
  - front : 첫번째 요소 하나 앞의 인덱스
  - rear : 마지막 요소의 인덱스
- 공백 상태 : front == rear
- 포화 상태 : front % M == (rear + 1) % M
- 공백 상태와 포화 상태를 구별하기 위해 하나의 공간은 항상 비워둠




```
====데이터 추가 단계 ====
정수를 입력하세요 : 10
QUEUE(front 0 rear 1) = 10 |
정수를 입력하세요 : 20
QUEUE(front 0 rear 2) = 10 | 20 |
정수를 입력하세요 : 30
QUEUE(front 0 rear 3) = 10 | 20 | 30 |
정수를 입력하세요 : 40
QUEUE(front 0 rear 4) = 10 | 20 | 30 | 40 |
큐는 포화상태입니다.

====데이터 삭제 단계 ====
꺼내진 정수 : 10
QUEUE(front 1 rear 4) = 20 | 30 | 40 |
꺼내진 정수 : 20
QUEUE(front 2 rear 4) = 30 | 40 |
꺼내진 정수 : 30
QUEUE(front 3 rear 4) = 40 |
꺼내진 정수 : 40
QUEUE(front 4 rear 4) =
큐는 공백 상태입니다.
```

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
```

```
typedef int element;
typedef struct{
    int front;
    int rear;
    element data[MAX];
}QueueType;

void error(const char *message){
    fprintf(stderr, "%s\n", message);
    exit(1);
}
void init_queue(QueueType *q){
    q->front = q->rear = 0;
}
int is_empty(QueueType *q){
    return (q->front == q->rear);
}
int is_full(QueueType *q){
    return ((q->rear + 1) % MAX == q->front);
}
void queue_print(QueueType *q){
    printf("QUEUE(front %d rear %d) = ", q->front, q->rear);
    if(!is_empty(q)){
        int i = q->front;
        do{
            i = (i+1) % MAX;
            printf("%d | ", q->data[i]);
            if(i == q->rear)
                break;
        }while(i != q->front);
    }
    printf("\n");
}

void enqueue(QueueType *q, element item){
    if(is_full(q)){
        error("큐가 포화 상태입니다.");
    }
    q->rear = (q->rear + 1) % MAX;
    q->data[q->rear] = item;
```

```
}
element dequeue(QueueType *q){
    if(is_empty(q))
        error("큐가 공백 상태입니다.");
    q→front = (q→front +1) % MAX;
    return q→data[q→front];
}
element peek(QueueType *q){
    if(is_empty(q))
        error("큐가 공백 상태입니다.");
    return q→data[(q→front + 1) % MAX];
}

int main(){
    QueueType q;
    int element = 0;

    init_queue(&q);
    printf("====데이터 추가 단계 ====\n");
    while(!is_full(&q)){
        printf("정수를 입력하세요 : ");
        scanf("%d", &element);
        enqueue(&q, element);
        queue_print(&q);
    }
    printf("큐는 포화상태입니다. \n\n");

    printf("====데이터 삭제 단계 ====\n");
    while(!is_empty(&q)){
        element = dequeue(&q);
        printf("꺼내진 정수 : %d\n", element);
        queue_print(&q);
    }
    printf("큐는 공백 상태입니다. \n\n");
    return 0;
}
```

▼ 큐의 응용 : 버퍼

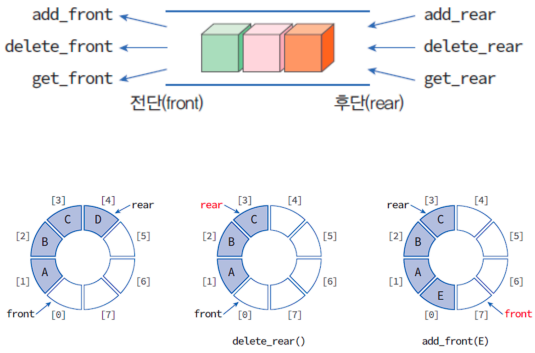
```
#include <stdio.h>
#include <stdlib.h>
```

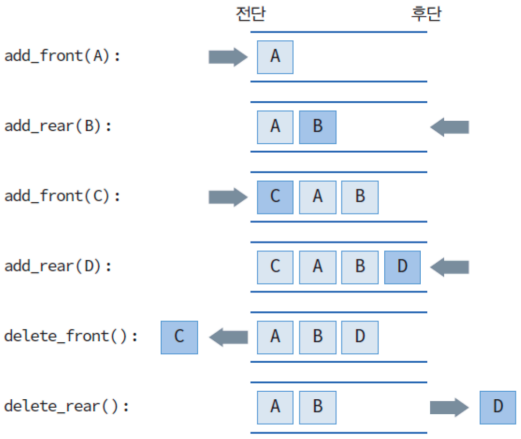
```
#include <time.h>
#define MAX 5

typedef int element;
typedef struct{
    int front;
    int rear;
    element data[MAX];
}QueueType;

void error(const char *message){
    fprintf(stderr, "%s\n", message);
    exit(1);
}
void init_queue(QueueType *q){
    q->front = q->rear = 0;
}
int is_empty(QueueType *q){
    return (q->front == q->rear);
}
int is_full(QueueType *q){
    return ((q->rear + 1) % MAX == q->front);
}
void queue_print(QueueType *q){
    printf("QUEUE(front %d rear %d) = ", q->front, q->rear);
    if(!is_empty(q)){
        int i = q->front;
        do{
            i = (i+1) % MAX;
            printf("%d | ", q->data[i]);
            if(i == q->rear)
                break;
        }while(i != q->front);
    }
    printf("\n");
}

void enqueue(QueueType *q, element item){
    if(is_full(q)){
        error("큐가 포화 상태입니다.");
    }
```

```
    q->rear = (q->rear + 1) % MAX;
    q->data[q->rear] = item;
}
element dequeue(QueueType *q){
    if(is_empty(q))
        error("큐가 공백 상태입니다.");
    q->front = (q->front +1) % MAX;
    return q->data[q->front];
}
element peek(QueueType *q){
    if(is_empty(q))
        error("큐가 공백 상태입니다.");
    return q->data[(q->front + 1) % MAX];
}

int main(){
    QueueType q;
    int data = 0;

    init_queue(&q);
    srand(time(NULL));

    for(int i = 0; i < 100; i++){
        if(rand() % 5 == 0)
            enqueue(&q, rand()%100);
        queue_print(&q);
        if(rand() % 10 == 0)
            data = dequeue(&q);
        queue_print(&q);
    }
    return 0;
}
```

▼ Deque

- double-ended queue의 줄임말로서 큐의 전단과 후단에서 모두 삽입과 삭제가 가능한 큐





front ← (front-1 + MAX) % MAX

rear ← (rear-1 + MAX) % MAX

이건 front, rear == 0일때를 감안함

▼ ADT

- create() : 덱을 생성함
- init(dq) : 덱을 초기화함
- is_empty(dq) : 덱이 공백 상태인지를 검사함
- is_full(dq) : 덱이 포화상태인지를 검사함
- add_front(dq, e) : 덱의 앞에 요소를 추가함
- add_rear(dq, e) : 덱의 뒤에 요소를 추가함
- delete_front(dq) : 덱의 앞에 있는 요소를 반환하고 삭제함
- delete_rear(dq) : 덱의 뒤에 있는 요소를 반환하고 삭제함
- get_front(q) : 덱의 앞에 삭제하지 않고 앞에 있는 요소를 반환함
- get_rear(q) : 덱의 뒤에 삭제하지 않고 뒤에있는 요소를 반환함

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

#define MAX 5
typedef int element;
typedef struct{
    int front, rear;
    element data[MAX];
}DequeueType;
```

```
void error(const char *message){
    fprintf(stderr, "%s\n", message);
    exit(1);
}

void init_deque(DequeueType *dq){
    dq->front = dq->rear = 0;
}
int is_empty(DequeueType *dq){
    return (dq->front == dq->rear);
}
int is_full(DequeueType *dq){
    return ((dq->rear + 1) % MAX == dq->front);
}
void dequeue_print(DequeueType *q){
    printf("DEQUE(front=%d rear=%d) = ", q->front, q->rear);
    if(!is_empty(q)){
        int i = q->front;
        do{
            i = (i+1) % MAX;
            printf("%d | ", q->data[i]);
            if(i == q->rear)
                break;
        }while(i != q->front);
    }
    printf("\n");
}
void add_rear(DequeueType *dq, element item){
    if(is_full(dq))
        error("덱이 포화 상태입니다.");
    dq->rear = (dq->rear + 1) % MAX;
    dq->data[dq->rear] = item;
}
element delete_front(DequeueType *dq){
    if(is_empty(dq)){
        error("덱이 공백 상태입니다.");
    }
    dq->front = (dq->front + 1) % MAX;
    return dq->data[dq->front];
}
element get_front(DequeueType *dq){
```

```
    if(is_empty(dq))
        error("덱이 공백 상태입니다.");
    return dq->data[(dq->front + 1) % MAX];
}
void add_front(DequeueType *dq, element val){
    if(is_full(dq))
        error("덱이 포화 상태입니다.");
    dq->data[dq->front] = val;
    dq->front = (dq->front -1 + MAX) % MAX;
}
element delete_rear(DequeueType *q){
    int prev = q->rear;
    if(is_empty(q))
        error("덱이 공백 상태입니다.");
    q->rear = (q->rear -1 + MAX) % MAX;
    return q->data[prev];
}

element get_rear(DequeueType *q){
    if(is_empty(q))
        error("덱이 공백 상태입니다.");
    return q->data[q->rear];
}

int main(){
    DequeueType dq;
    init_deque(&dq);
    for(int i = 0; i < 3 ; i++){
        add_front(&dq, i);
        dequeue_print(&dq);
    }
    for(int i = 0; i < 3; i++){
        delete_rear(&dq);
        dequeue_print(&dq);
    }

    return 0;
}
```

▼ 큐의 응용 : 시뮬레이션

- 은행의 고객 서비스, 네트워크 패킷 처리 서비스

▼ 은행 서비스 시뮬레이션

- 서비스 창구 수 = 1
- 총 60분 시뮬레이션
- 각 1분당 고객이 도착할 확률 0.3
- 고객 당, 서비스 시간 1/2/3분 랜덤
- 고객 도착시 원형큐를 이용해서 고객 처리 업무를 enqueue
- queue가 empty가 아니면 서비스 창구는 dequeue하여 업무를 처리
- 시뮬레이션에 따라 고객의 평균 대기 시간을 계산하여 서비스 창구를 늘릴지 결정

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAX 5
typedef struct{
    int id;
    int arrival_time;
    int service_time;
}element;

typedef struct{
    int front;
    int rear;
    element data[MAX];
}QueueType;


void error(const char *message){
    fprintf(stderr, "%s\n", message);
    exit(1);
}

void init_queue(QueueType *q){
    q->front = q->rear = 0;
}

int is_empty(QueueType *q){
```

```
    return (q->front == q->rear);
}
int is_full(QueueType *q){
    return ((q->rear + 1) % MAX == q->front);
}

void queue_print(QueueType *q){
    printf("QUEUE(front %d rear %d) = ", q->front, q->rear);
    if(!is_empty(q)){
        int i = q->front;
        do{
            i = (i+1) % MAX;
            printf("%d | ", q->data[i]);
            if(i == q->rear)
                break;
        }while(i != q->front);
    }
    printf("\n");
}

void enqueue(QueueType *q, element item){
    if(is_full(q))
        error("큐가 포화 상태입니다.");
    q->rear = (q->rear + 1) % MAX;
    q->data[q->rear] = item;
}
element dequeue(QueueType *q){
    if(is_empty(q))
        error("큐가 공백 상태입니다.");
    q->front = (q->front + 1) % MAX;
    return q->data[q->front];
}

element peek(QueueType *q){
    if(is_empty(q))
        error("큐가 공백 상태입니다.");
    return q->data[(q->front + 1) % MAX];
}

int main(){
    int minutes = 60;
```

```
    int total_wait = 0, total_customers = 0, service_time = 0;
    int service_customer;
    QueueType q;

    init_queue(&q);
    srand(time(NULL));

    for(int clock = 0; clock < minutes; clock++){
        printf("현재 시각 : %d\n", clock);
        if((rand()%10 < 3)){
            element customer;
            customer.id = total_customers++;
            customer.arrival_time = clock;
            customer.service_time = rand() % 3 + 1;
            enqueue(&q, customer);
            printf("고객 %d가 %d분에 들어옵니다. 업무 처리 시간 = %d분\n", customer.id, custo
        }
        if(service_time > 0){
            printf("고객 %d의 업무 처리중입니다. \n", service_customer);
            service_time--;
        }else{
            if(!is_empty(&q)){
                element costomer = dequeue(&q);
                service_customer = costomer.id;
                service_time = costomer.service_time;
                printf("고객 %d이 %d분에 업무를 시작합니다. 대기 시간은 %d분 이었습니다.\n", co
                total_wait += clock - costomer.arrival_time;
            }
        }
    }
    printf("전체 대기 시간 %d분\n", total_wait);
    return 0;
}
```