

6장 큐





- 큐: 먼저 들어온 데이터가 먼저 나가는 자료구조
- 선입선출(**FIFO: First-In First-Out**)
- (예)매표소의 대기열



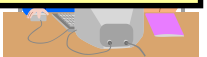


ADT

객체: 0개 이상의 요소들로 구성된 선형 리스트

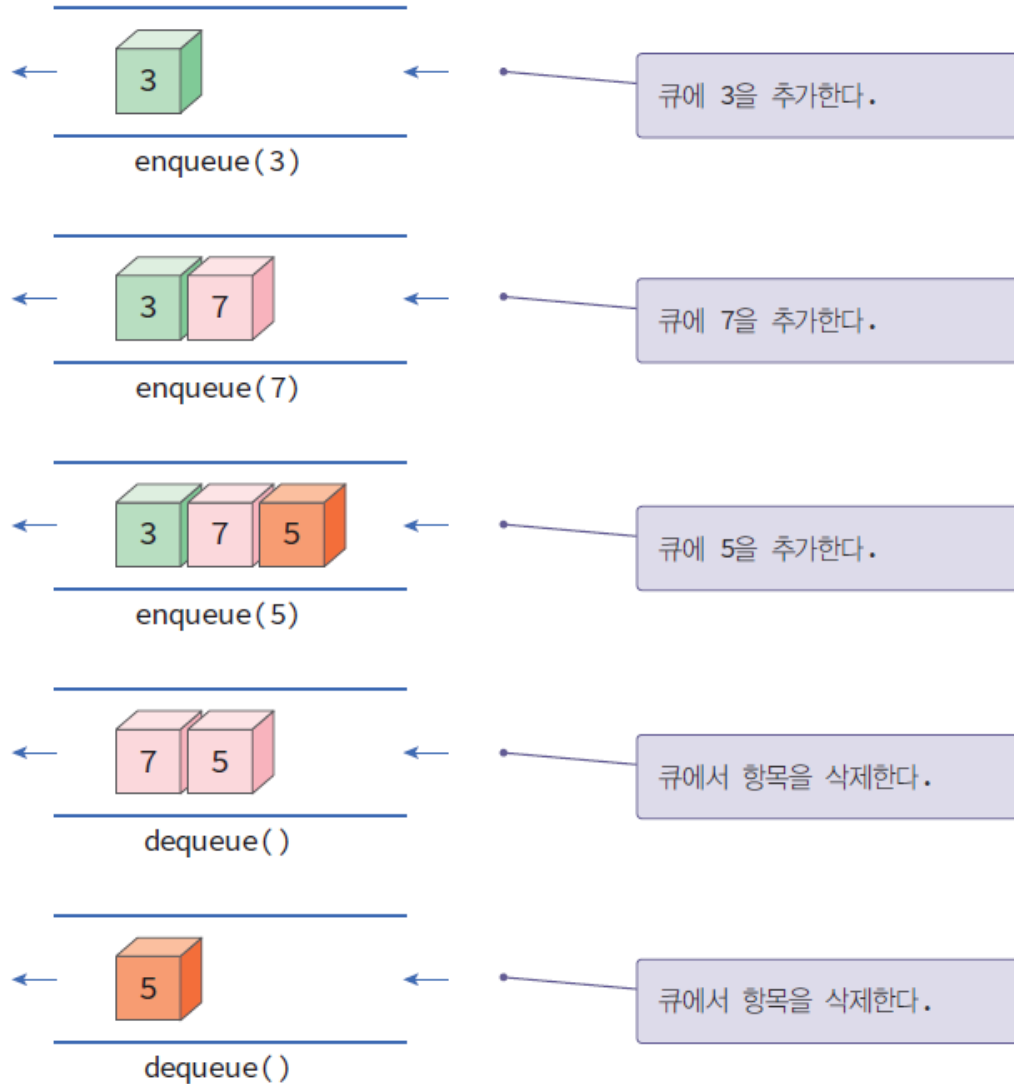
.연산:

- `create(max_size) ::=`
최대 크기가 `max_size`인 공백큐를 생성한다.
- `init(q) ::=`
큐를 초기화한다.
- `is_empty(q) ::=`
`if(size == 0) return TRUE;`
`else return FALSE;`
- `is_full(q) ::=`
`if(size == max_size) return TRUE;`
`else return FALSE;`
- `enqueue(q, e) ::=`
`if( is_full(q) ) queue_full 오류;`
`else q의 끝에 e를 추가한다.`
- `dequeue(q) ::=`
`if( is_empty(q) ) queue_empty 오류;`
`else q의 맨 앞에 있는 e를 제거하여 반환한다.`
- `peek(q) ::=`
`if( is_empty(q) ) queue_empty 오류;`
`else q의 맨 앞에 있는 e를 읽어서 반환한다.`



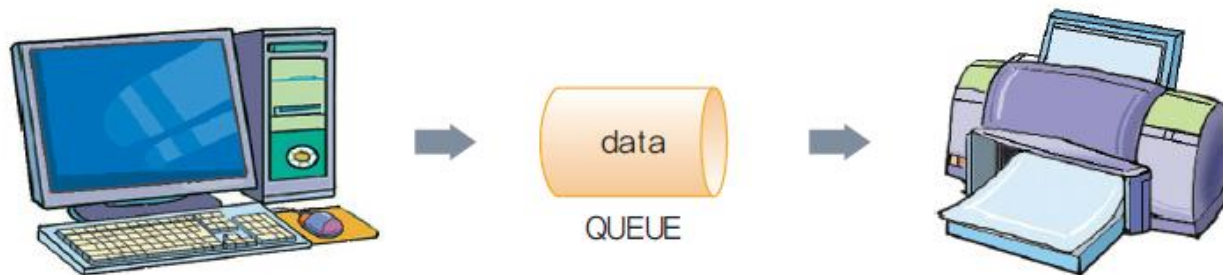


큐의 삽입, 삭제 연산



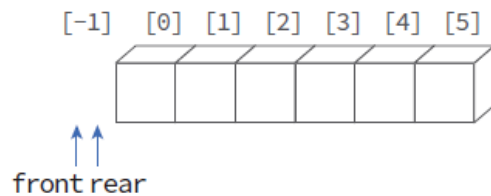


- 직접적인 응용
 - ▣ 시뮬레이션의 대기열(공항에서의 비행기들, 은행에서의 대기열)
 - ▣ 통신에서의 데이터 패킷들의 모델링에 이용
 - ▣ 프린터와 컴퓨터 사이의 버퍼링
- 간접적인 응용
 - ▣ 스택과 마찬가지로 프로그래머의 도구
 - ▣ 많은 알고리즘에서 사용됨

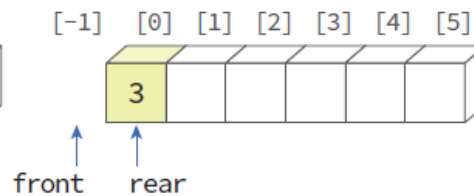




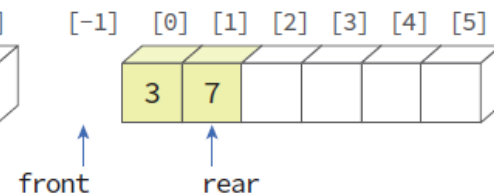
- 배열을 선형으로 사용하여 큐를 구현
 - 삽입을 계속하기 위해서는 요소들을 이동시켜야 함



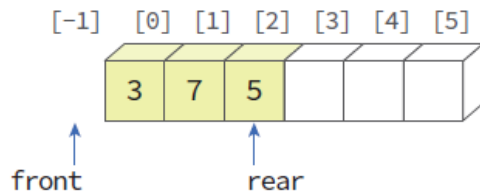
(a) 초기상태



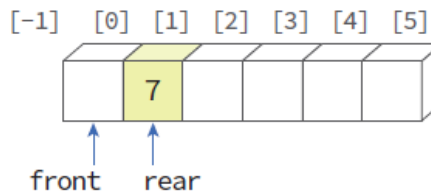
(b) enqueue(3)



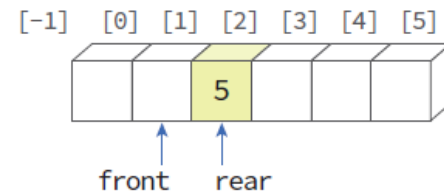
(c) enqueue(7)



(d) enqueue(5)

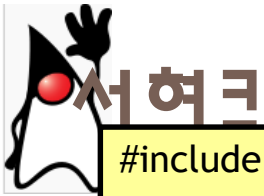


(e) dequeue()



(f) dequeue()





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

typedef int element;
typedef struct {                                // 큐 타입
    int front;
    int rear;
    element data[MAX_QUEUE_SIZE];
} QueueType;

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

void init_queue(QueueType *q)
{
    q->rear = -1;
    q->front = -1;
}
```





```
void queue_print(QueueType *q)
```

```
{  
    for (int i = 0; i < MAX_QUEUE_SIZE; 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_QUEUE_SIZE - 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 포화상태입니다.");
        return;
    }
    q->data[++(q->rear)] = item;
}

int dequeue(QueueType *q)
{
    if (is_empty(q)) {
        error("큐가 공백상태입니다.");
        return -1;
    }
    int item = q->data[++(q->front)];
    return item;
}
```





```
int main(void)
{
    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);

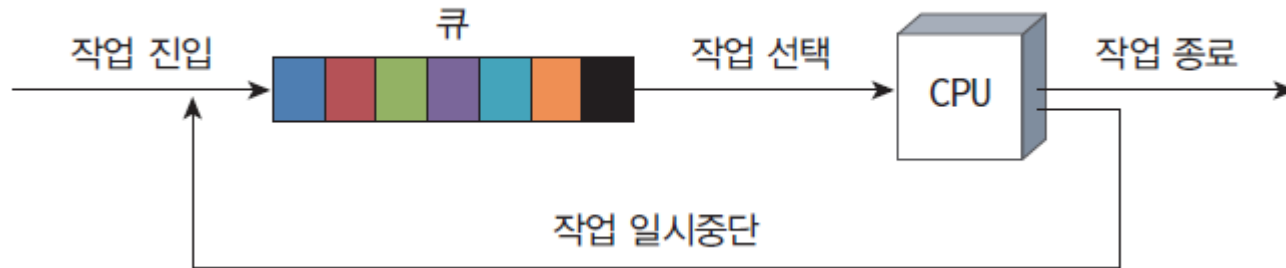
    item = dequeue(&q); queue_print(&q);
    item = dequeue(&q); queue_print(&q);
    item = dequeue(&q); queue_print(&q);
    return 0;
}
```

```
10 |  |  |  |  |
10 | 20 |  |  |  |
10 | 20 | 30 |  |  |
   | 20 | 30 |  |  |
   |  | 30 |  |  |
   |  |  |  |  |
```



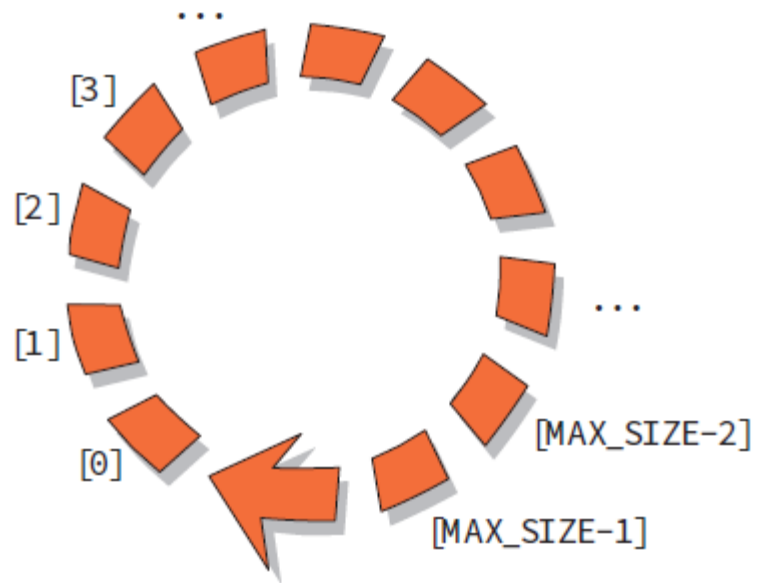


선형 큐의 응용: 작업 스케줄링

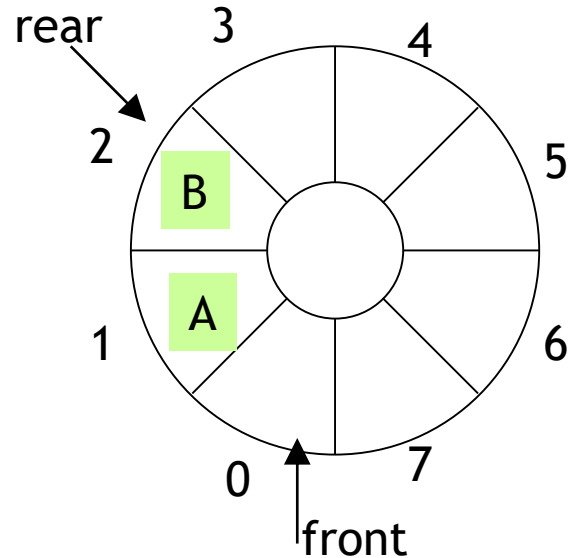


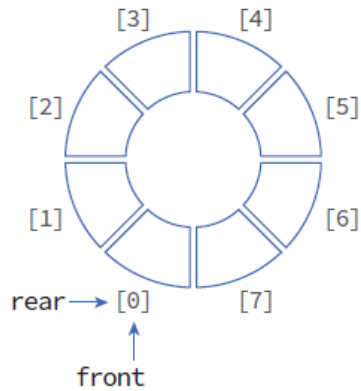
Q[0]	Q[1]	Q[2]	Q[3]	Q[4]	front	rear	설명
					-1	-1	공백 큐
Job#1					-1	0	Job#1이 추가
Job#1	Job#2				-1	1	Job#2이 추가
Job#1	Job#2	Job#3			-1	2	Job#3이 추가
	Job#2	Job#3			0	2	Job#1이 삭제
		Job#3			1	2	Job#2이 삭제



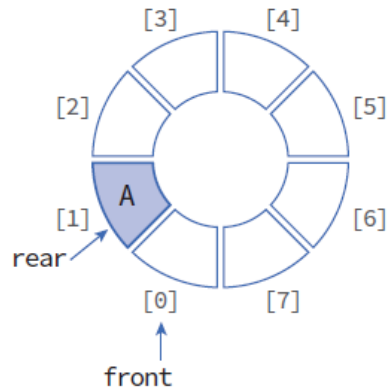


- 큐의 전단과 후단을 관리하기 위한 2개의 변수 필요
 - ▣ front: 첫번째 요소 하나 앞의 인덱스
 - ▣ rear: 마지막 요소의 인덱스

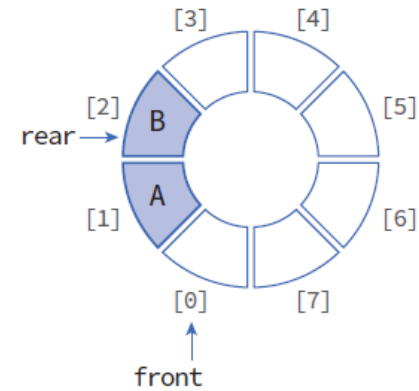




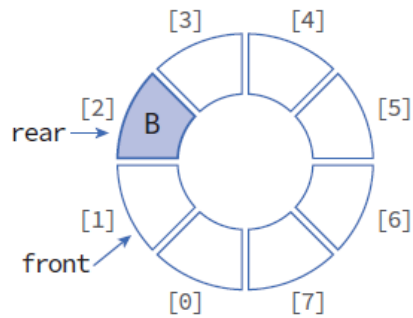
(a) 초기상태



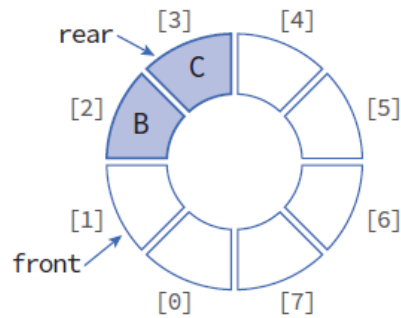
(b) A 삽입



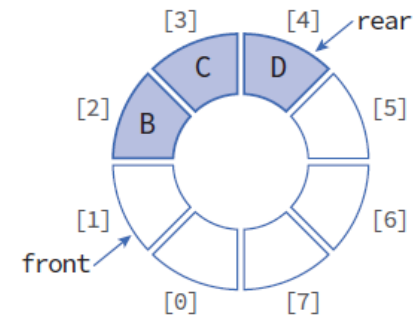
(c) B 삽입



(d) 삭제



(e) C 삽입

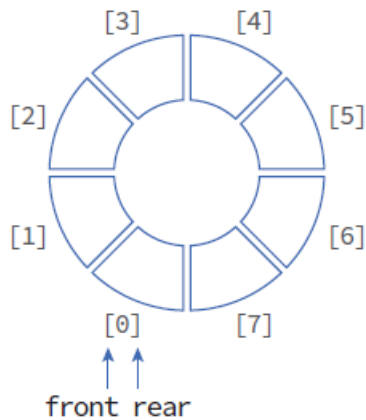


(f) D 삽입

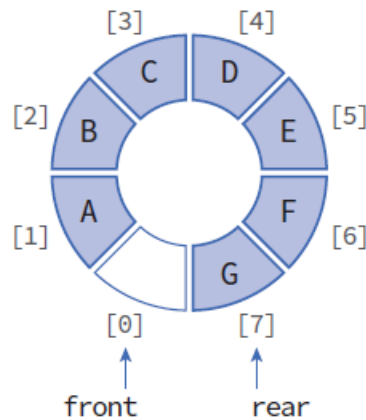


공백상태, 포화상태

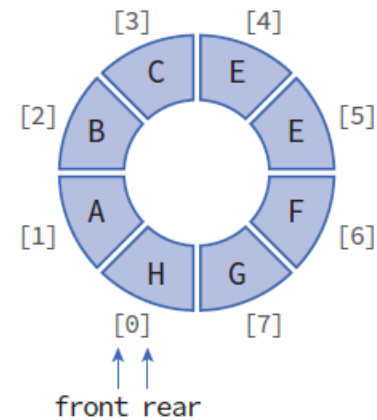
- 공백상태: $front == rear$
- 포화상태: $front \% M == (rear + 1) \% M$
- 공백상태와 포화상태를 구별하기 위하여 하나의 공간은 항상 비워둔다.



(a) 공백 상태



(b) 포화 상태



(c) 오류 상태





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

// ===== 원형큐 코드 시작 =====
#define MAX_QUEUE_SIZE 5
typedef int element;
typedef struct { // 큐 타입
    element data[MAX_QUEUE_SIZE];
    int front, rear;
} QueueType;

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





```
// 공백 상태 검출 함수
void init_queue(QueueType *q)
{
    q->front = q->rear = 0;
}

// 공백 상태 검출 함수
int is_empty(QueueType *q)
{
    return (q->front == q->rear);
}

// 포화 상태 검출 함수
int is_full(QueueType *q)
{
    return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front);
}
```





// 원형큐 출력 함수

```
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_QUEUE_SIZE);
            printf("%d | ", q->data[i]);
            if (i == q->rear)
                break;
        } while (i != q->front);
        printf("\n");
    }
}
```





```
// 삽입 함수
void enqueue(QueueType *q, element item)
{
    if (is_full(q))
        error("큐가 포화상태입니다");
    q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;
    q->data[q->rear] = item;
}

// 삭제 함수
element dequeue(QueueType *q)
{
    if (is_empty(q))
        error("큐가 공백상태입니다");
    q->front = (q->front + 1) % MAX_QUEUE_SIZE;
    return q->data[q->front];
}
```





```
int main(void)
```

```
{
```

```
    QueueType queue;
```

```
    int element;
```

```
    init_queue(&queue);
```

```
    printf("--데이터 추가 단계--\n");
```

```
    while (!is_full(&queue))
```

```
    {
```

```
        printf("정수를 입력하시오: ");
```

```
        scanf("%d", &element);
```

```
        enqueue(&queue, element);
```

```
        queue_print(&queue);
```

```
    }
```

```
    printf("큐는 포화상태입니다.\n\n");
```

```
    printf("--데이터 삭제 단계--\n");
```

```
    while (!is_empty(&queue))
```

```
    {
```

```
        element = dequeue(&queue);
```

```
        printf("꺼내진 정수: %d \n", element);
```

```
        queue_print(&queue);
```

```
    }
```

```
    printf("큐는 공백상태입니다.\n");
```

```
    return 0;
```

```
}
```





--데이터 추가 단계--

정수를 입력하시오: 10

QUEUE(front=0 rear=1) = 10 |

정수를 입력하시오: 20

QUEUE(front=0 rear=2) = 10 | 20 |

정수를 입력하시오: 30

QUEUE(front=0 rear=3) = 10 | 20 | 30 |

정수를 입력하시오: 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 공백상태입니다.





큐의 중요: 버퍼



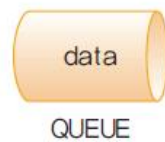
생산자



버퍼



소비자





프로그래밍

```
int main(void)
{
    QueueType queue;
    int element;

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

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

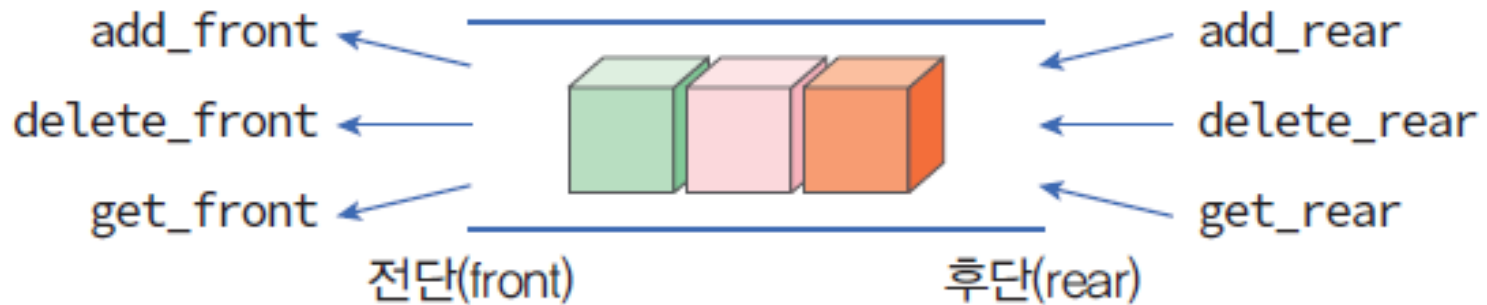


```
...  
QUEUE(front=0 rear=1) = 53 |  
QUEUE(front=1 rear=1) =  
QUEUE(front=1 rear=2) = 73 |  
QUEUE(front=1 rear=2) = 73 |  
QUEUE(front=1 rear=2) = 73 |  
QUEUE(front=1 rear=3) = 73 | 96 |  
QUEUE(front=1 rear=3) = 73 | 96 |  
QUEUE(front=1 rear=3) = 73 | 96 |  
...
```



덱(deque)

- **덱(deque)**은 **double-ended queue**의 줄임말로서 큐의 전단(front)와 후단(rear)에서 모두 삽입과 삭제가 가능한 큐



.객체: n개의 **element**형으로 구성된 요소들의 순서있는 모임

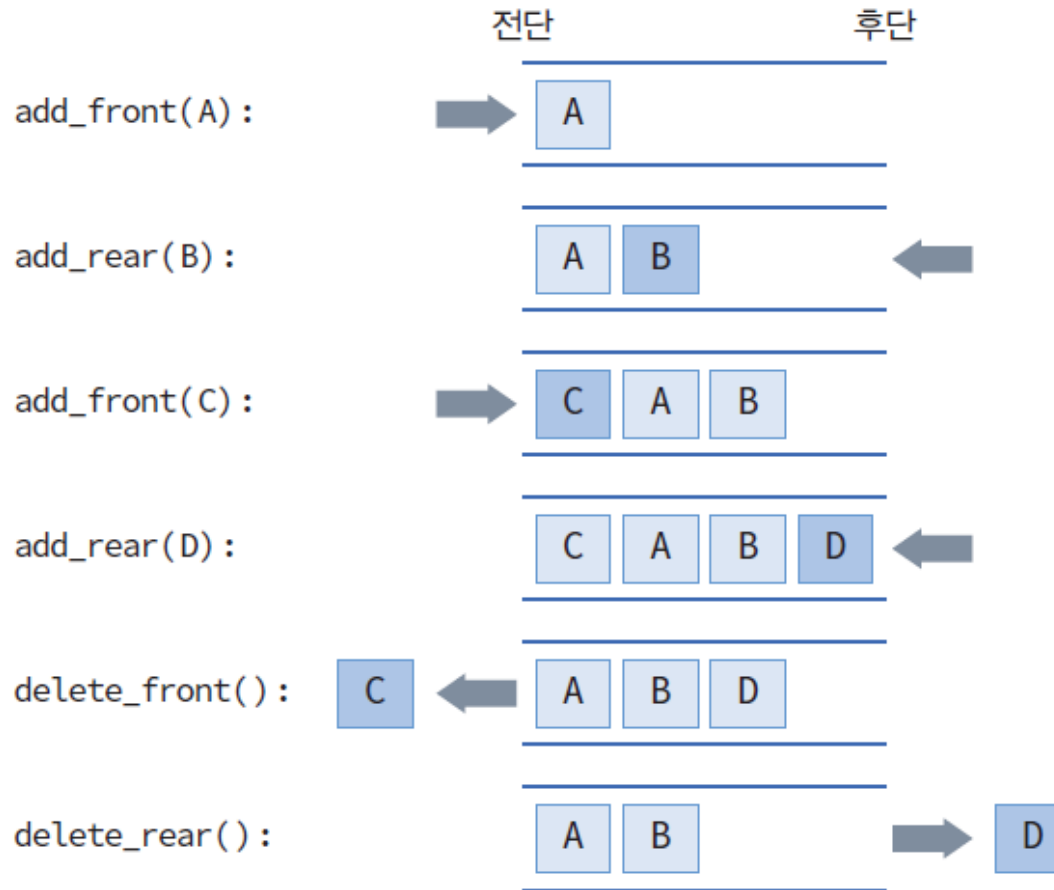
.연산:

- **create() ::=** 덱을 생성한다.
- **init(dq) ::=** 덱을 초기화한다.
- **is_empty(dq) ::=** 덱이 공백상태인지를 검사한다.
- **is_full(dq) ::=** 덱이 포화상태인지를 검사한다.
- **add_front(dq, e) ::=** 덱의 앞에 요소를 추가한다.
- **add_rear(dq, e) ::=** 덱의 뒤에 요소를 추가한다.
- **delete_front(dq) ::=** 덱의 앞에 있는 요소를 반환한 다음 삭제한다
- **delete_rear(dq) ::=** 덱의 뒤에 있는 요소를 반환한 다음 삭제한다.
- **get_front(q) ::=** 덱의 앞에서 삭제하지 않고 앞에 있는 요소를 반환한다.
- **get_rear(q) ::=** 덱의 뒤에서 삭제하지 않고 뒤에 있는 요소를 반환한다.





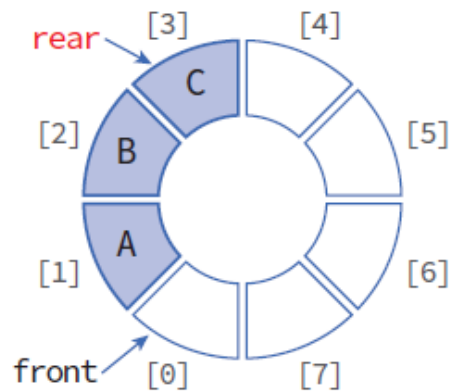
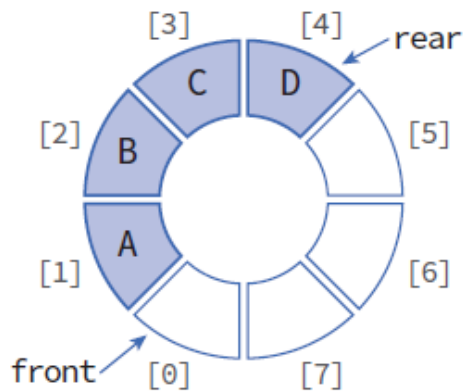
데크의 역사



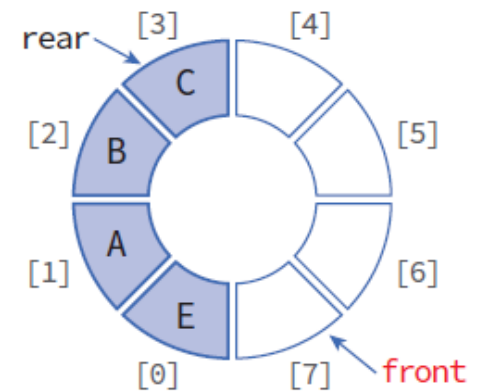


배열을 이용한 데크의 구현

```
front ← (front-1 + MAX_QUEUE_SIZE) % MAX_QUEUE_SIZE;  
rear ← (rear-1 + MAX_QUEUE_SIZE) % MAX_QUEUE_SIZE;
```



delete_rear()



add_front(E)





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

#define MAX_QUEUE_SIZE 5
typedef int element;
typedef struct { // 큐 타입
    element data[MAX_QUEUE_SIZE];
    int front, rear;
} DequeType;

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

// 초기화
void init_deque(DequeType *q)
{
    q->front = q->rear = 0;
}
```





// 공백 상태 검출 함수

```
int is_empty(DequeType *q)
{
    return (q->front == q->rear);
}
```

// 포화 상태 검출 함수

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

// 원형큐 출력 함수

```
void deque_print(DequeType *q)
{
    printf("DEQUEUE(front=%d rear=%d) = ", q->front, q->rear);
    if (!is_empty(q)) {
        int i = q->front;
        do {
            i = (i + 1) % (MAX_QUEUE_SIZE);
            printf("%d | ", q->data[i]);
            if (i == q->rear)
                break;
        } while (i != q->front);
    }
    printf("\n");
}
```





```
// 삽입 함수
void add_rear(DequeType *q, element item)
{
    if (is_full(q))
        error("큐가 포화상태입니다");
    q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;
    q->data[q->rear] = item;
}

// 삭제 함수
element delete_front(DequeType *q)
{
    if (is_empty(q))
        error("큐가 공백상태입니다");
    q->front = (q->front + 1) % MAX_QUEUE_SIZE;
    return q->data[q->front];
}
```





```
// 삭제 함수
element get_front(DequeType *q)
{
    if (is_empty(q))
        error("큐가 공백상태입니다");
    return q->data[(q->front + 1) % MAX_QUEUE_SIZE];
}

void add_front(DequeType *q, element val)
{
    if (is_full(q))
        error("큐가 포화상태입니다");
    q->data[q->front] = val;
    q->front = (q->front - 1 + MAX_QUEUE_SIZE) % MAX_QUEUE_SIZE;
}
```





```
element delete_rear(DequeType *q)
{
    int prev = q->rear;
    if (is_empty(q))
        error("큐가 공백상태입니다");
    q->rear = (q->rear - 1 + MAX_QUEUE_SIZE) % MAX_QUEUE_SIZE;
    return q->data[prev];
}

element get_rear(DequeType *q)
{
    if (is_empty(q))
        error("큐가 공백상태입니다");
    return q->data[q->rear];
}
```





```
int main(void)
{
    DequeType queue;

    init_deque(&queue);
    for (int i = 0; i < 3; i++) {
        add_front(&queue, i);
        deque_print(&queue);
    }
    for (int i = 0; i < 3; i++) {
        delete_rear(&queue);
        deque_print(&queue);
    }
    return 0;
}
```





```
DEQUE(front=4 rear=0) = 0 |  
DEQUE(front=3 rear=0) = 1 | 0 |  
DEQUE(front=2 rear=0) = 2 | 1 | 0 |  
DEQUE(front=2 rear=4) = 2 | 1 |  
DEQUE(front=2 rear=3) = 2 |  
DEQUE(front=2 rear=2) =
```





큐의 응용: 시뮬레이션

- 큐잉모델은 고객에 대한 서비스를 수행하는 서버와 서비스를 받는 고객들로 이루어진다
- 은행에서 고객이 들어와서 서비스를 받고 나가는 과정을 시뮬레이션
 - ▣ 고객들이 기다리는 평균시간을 계산





큐의 응용: 시뮬레이션

- 시뮬레이션은 하나의 반복 루프
- 현재시각을 나타내는 **clock**이라는 변수를 하나 증가
- **is_customer_arrived** 함수를 호출한다.
is_customer_arrived 함수는 랜덤 숫자를 생성하여 시뮬레이션 파라미터 변수인 **arrival_prov**와 비교하여 작으면 새로운 고객이 들어왔다고 판단
- 고객의 아이디, 도착시간, 서비스 시간 등의 정보를 만들어 구조체에 복사하고 이 구조체를 파라미터로 하여 큐의 삽입 함수 **enqueue()**를 호출한다.





큐의 응용: 시뮬레이션

- 고객이 필요로 하는 서비스 시간은 역시 랜덤숫자를 이용하여 생성된다.
- 지금 서비스하고 있는 고객이 끝났는지를 검사: 만약 `service_time`이 0이 아니면 어떤 고객이 지금 서비스를 받고 있는 중임을 의미한다.
- `clock`이 하나 증가했으므로 `service_time`을 하나 감소시킨다.
- 만약 `service_time`이 0이면 현재 서비스받는 고객이 없다는 것을 의미한다. 따라서 큐에서 고객 구조체를 하나 꺼내어 서비스를 시작한다..





프로그래

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

// 프로그램 5.2에서 다음과 같은 부분을 복사한다.
// ===== 원형큐 코드 시작 =====
typedef struct { // 요소 타입
    int id;
    int arrival_time;
    int service_time;
} element;          // 교체!
// ...
// ===== 원형큐 코드 종료 =====

int main(void)
{
    int minutes = 60;
    int total_wait = 0;
    int total_customers = 0;
    int service_time = 0;
    int service_customer;
    QueueType queue;
    init_queue(&queue);
```





```
    srand(time(NULL));  
    for (int clock = 0; clock < minutes; clock++) {  
        printf("현재시각=%d\n", clock);  
        if ((rand()%10) < 3) {  
            element customer;  
            customer.id = total_customers++;  
            customer.arrival_time = clock;  
            customer.service_time = rand() % 3+1;  
            enqueue(&queue, customer);  
            printf("고객 %d이 %d분에 들어옵니다. 업무처리시간= %d분  
            \n",  
                customer.id, customer.arrival_time,  
                customer.service_time);  
        }  
    }
```





프로그래밍

```
        if (service_time > 0) {
            printf("고객 %d 업무처리중입니다. \n", service_customer);
            service_time--;
        }
        else {
            if (!is_empty(&queue)) {
                element customer = dequeue(&queue);
                service_customer = customer.id;
                service_time = customer.service_time;
                printf("고객 %d이 %d분에 업무를 시작합니다. 대
기시간은 %d분이었습니다.\n",
                    customer.id, clock, clock -
                    customer.arrival_time);
                total_wait += clock - customer.arrival_time;
            }
        }
    }
    printf("전체 대기 시간=%d분 \n", total_wait);
    return 0;
}
```





현재시각=0

현재시각=1

고객 0이 1분에 들어옵니다. 업무처리시간= 2분

고객 0이 1분에 업무를 시작합니다. 대기시간은 0분이었습니다.

현재시각=2

고객 0 업무처리중입니다.

현재시각=3

고객 1이 3분에 들어옵니다. 업무처리시간= 1분

고객 0 업무처리중입니다.

현재시각=4

고객 1이 4분에 업무를 시작합니다. 대기시간은 1분이었습니다.

현재시각=5

고객 1 업무처리중입니다.

현재시각=6

고객 2이 6분에 들어옵니다. 업무처리시간= 2분

