6장 큐(QUEUE)

큐(QUEUE)

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

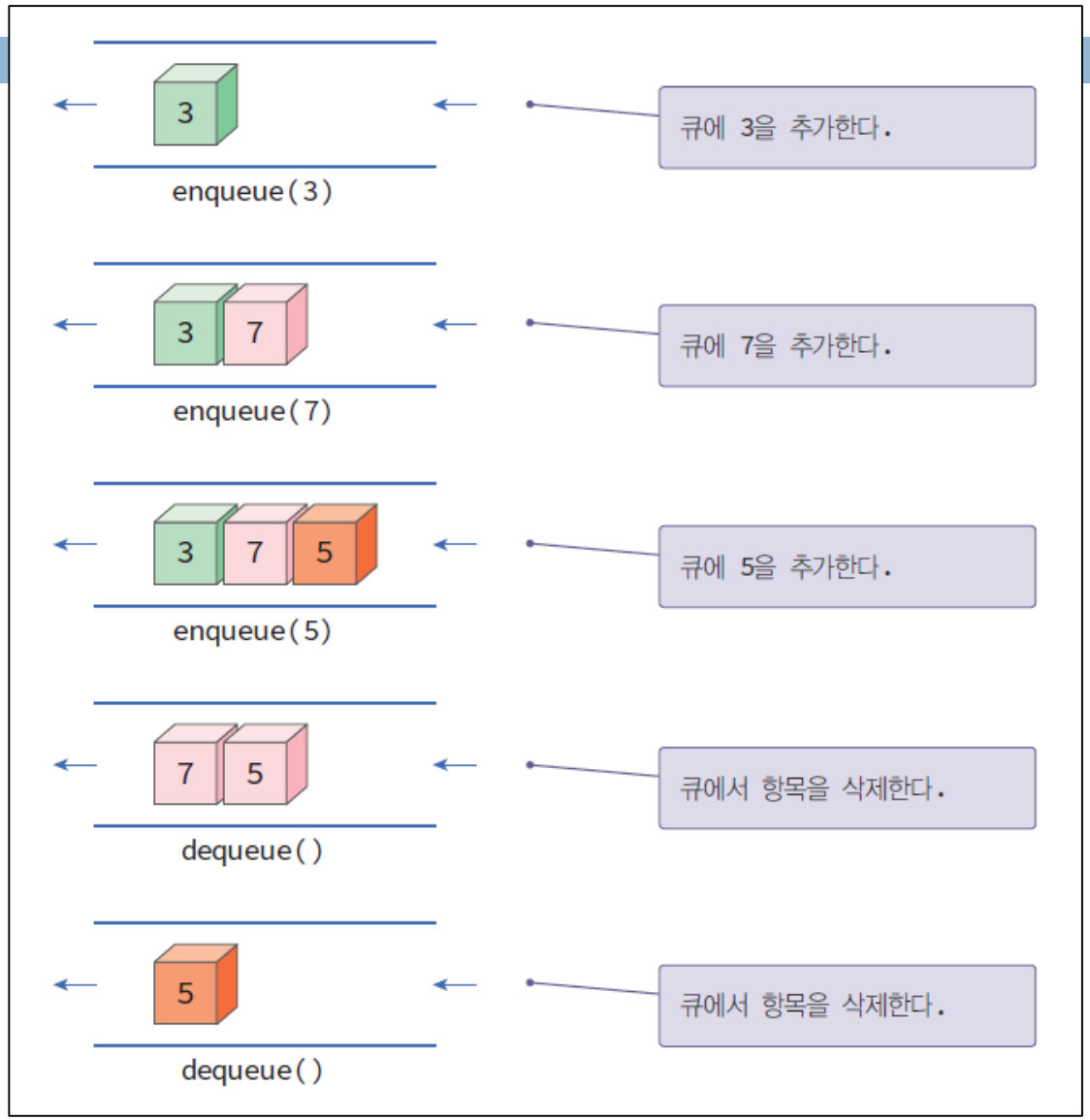
ADT

객체: 0개 이상의 요소들로 구성된 선형 리스트

·연산:

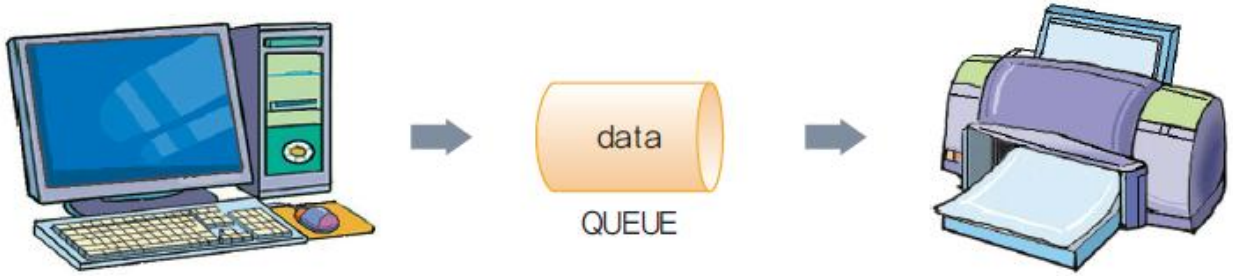
- create(max_size) ::=최대 크기가 max_size인 공백큐를 생성한다.
- init(q) ::= 큐를 초기화한다.
- is_empty(q) ::= if(size == 0) return TRUE;
  else return FALSE;
- is_full(q) ::= if(size == max_size) return TRUE;
  else return FALSE;
- enqueue(q, e) ::= if(is_full(q)) queue_full 오류;
  else q의 끝에 e를 추가한다.
- dequeue(q) ::= 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


3
enqueue(3)
큐에 3을 추가한다.
3
7
enqueue(7)
큐에 7을 추가한다.
3
7
5
enqueue(5)
큐에 5을 추가한다.
7
5
dequeue()
큐에서 항목을 삭제한다.
5
dequeue()
큐에서 항목을 삭제한다.

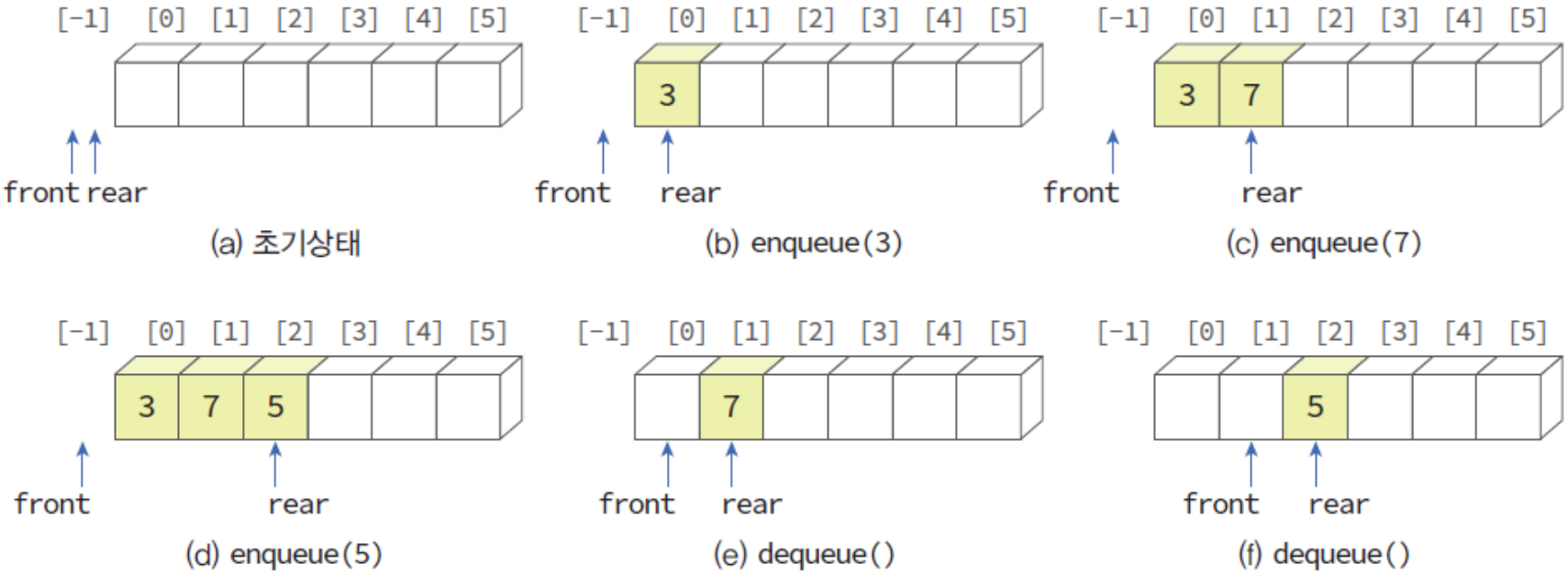
큐의 응용

- 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



선형큐

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



선형큐: linear_queue.c (1/3)


```c
#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(const 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");
}
```

선형큐: linear_queue.c (2/3)

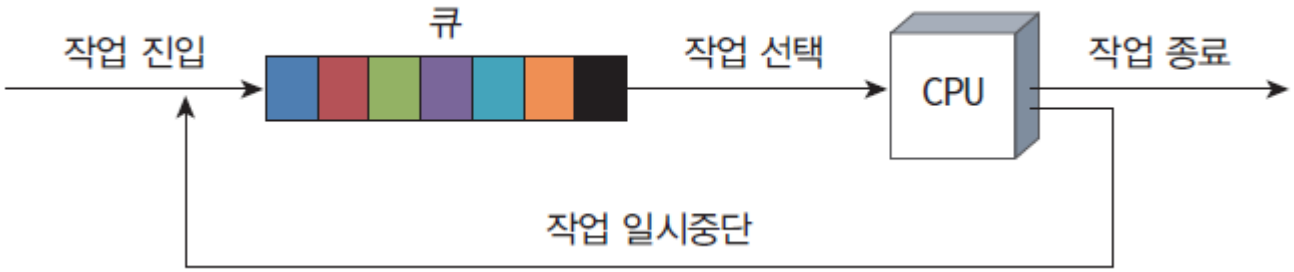

```c
int is_full(QueueType *q)
{
    if (q->rear == MAX_QUEUE_SIZE - 1)
        return 1;
    else
        return 0;
}

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;
}
```

선형큐: linear_queue.c (3/3)


```c
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 |    |    |
   |    |    |    |    |


--------------------------------
Process exited after 0.01576 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

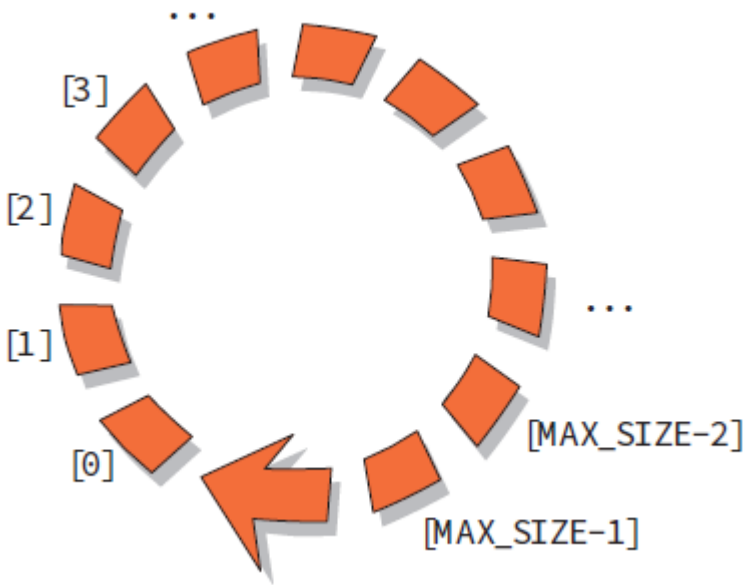
선형 큐의 응용: 작업 스케줄링



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

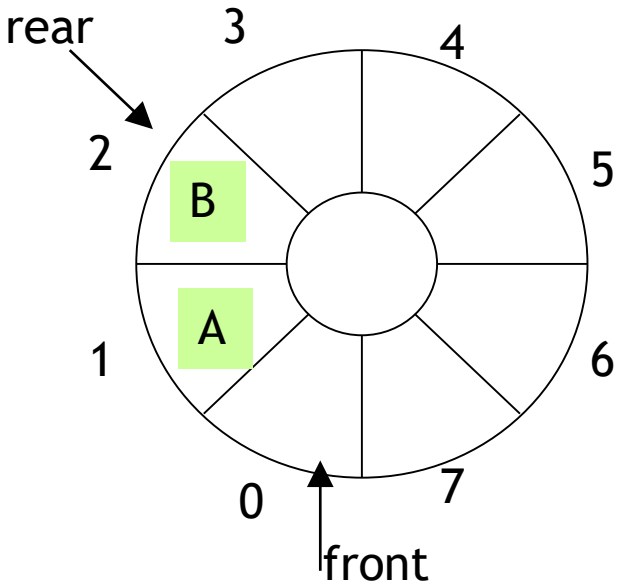
원형큐

- 선형큐의 문제점: front, rear이 모두 증가하여 배열의 앞부분이 비어 있더라도 사용하지 못함
  - 앞부분이 비어 있으면, 주기적으로 배열 데이터를 이동시키는 작업이 필요
- 이런 문제점 해결을 위해 원형큐를 도입

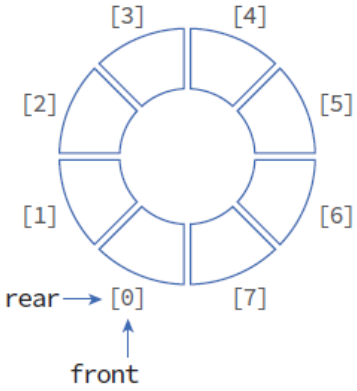


원형큐의 구조

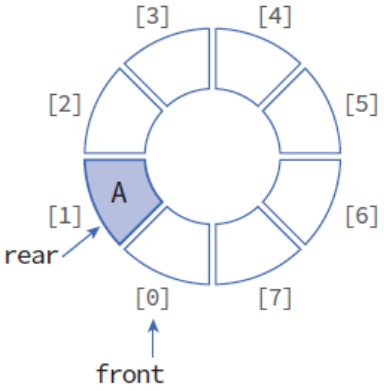
- 큐의 전단과 후단을 관리하기 위한 2개의 변수 필요
  - front: 첫번째 요소 하나 앞의 인덱스
  - rear: 마지막 요소의 인덱스



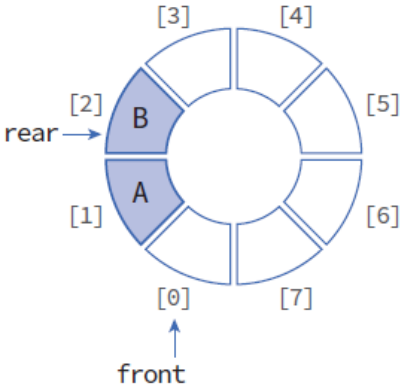
원형큐의 동작



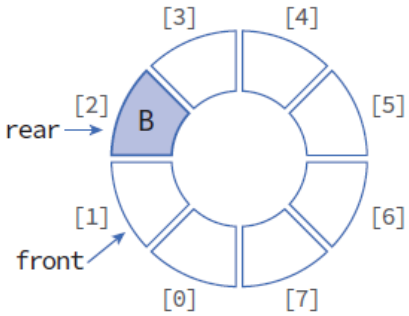
(a) 초기상태



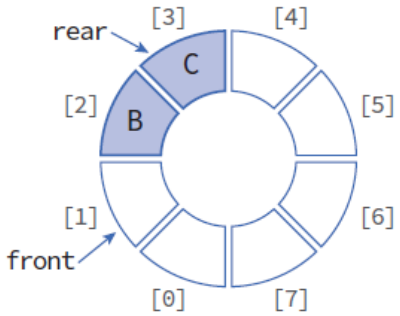
(b) A 삽입



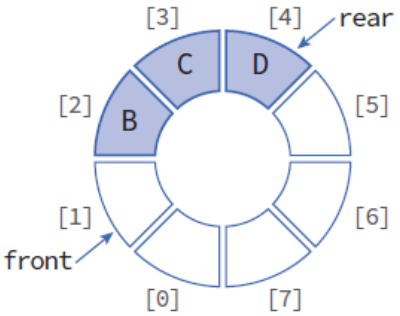
(c) B 삽입



(d) 삭제



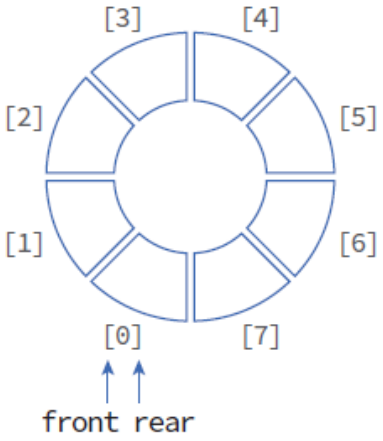
(e) C 삽입



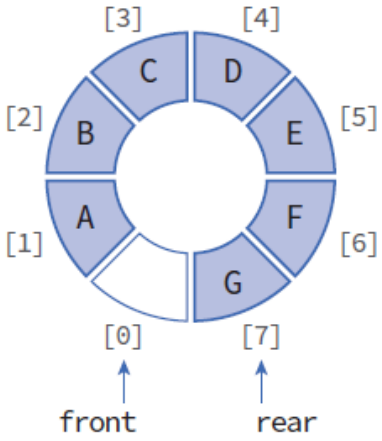
(f) D 삽입

공백상태, 포화상태

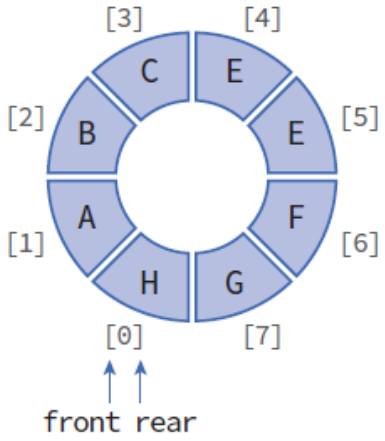
- 공백상태: front == rear
- 포화상태: front % M==(rear+1) % M
- 공백상태와 포화상태를 구별하기 위하여 하나의 공간은 항상 비워둔다.



(a) 공백 상태



(b) 포화 상태



(c) 오류 상태

원형 큐 프로그램: cir_queue.c (1/3)


```
6   // ===== 원형 큐 코드 시작 ======
7   #define MAX_QUEUE_SIZE 5
8   typedef int element;
9   typedef struct { // 큐 타입
10      element  data[MAX_QUEUE_SIZE];
11      int  front, rear;
12  } QueueType;
13
14  // 오류 함수:
15  void error(char *message)
16  {
17      fprintf(stderr, "%s\n", message);
18      exit(1);
19  }
20
21  // 공백 상태 검출 함수
22  void init_queue(QueueType *q)
23  {
24      q->front = q->rear = 0;
25  }
26
27  // 공백 상태 검출 함수:
28  int is_empty(QueueType *q)
29  {
30      return (q->front == q->rear);
31  }
32
33  // 포화 상태 검출 함수
34  int is_full(QueueType *q)
35  {
36      return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front);
37  }
```

원형 큐 프로그램: cir_queue.c (2/3)

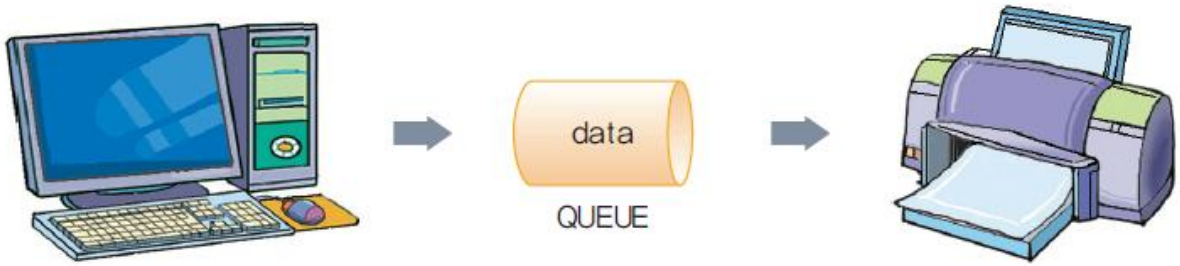

```c
// 원형 큐 출력 함수
void queue_print(QueueType *q)
{
    printf("QUEUE(front=%d rear=%d) = ", q->front, q->rear);
    if (!is_empty(q)) {
        int i = q->front;
        do {
            i = (i + 1) % (MAX_QUEUE_SIZE);
            printf("%d | ", q->data[i]);
            if (i == q->rear)
                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];
}
```

원형 큐 프로그램: cir_queue.c (3/3)

```
73  // 삭제 함수.
74  element peek(QueueType *q)
75  {
76      if (is_empty(q))
77          error("큐가 공백상태입니다");
78      return q->data[(q->front + 1) % MAX_QUEUE_SIZE];
79  }
80  // ===== 원형큐 코드 끝 ======
81
82  int main(void)
83  {
84      QueueType queue;
85      int element;
86
87      init_queue(&queue);
88      printf("--데이터 추가 단계--\n");
89      while (!is_full(&queue))
90      {
91          printf("정수를 입력하시오: ");
92          scanf("%d", &element);
93          enqueue(&queue, element);
94          queue_print(&queue);
95      }
96      printf("큐는 포화상태입니다.\n\n");
97
98      printf("--데이터 삭제 단계--\n");
99      while (!is_empty(&queue))
100     {
101         element = dequeue(&queue);
102         printf("꺼내진 정수: %d \n", element);
103         queue_print(&queue);
104     }
105     printf("큐는 공백상태입니다.\n");
106     return 0;
107 }
```

```
--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 응용: 버퍼



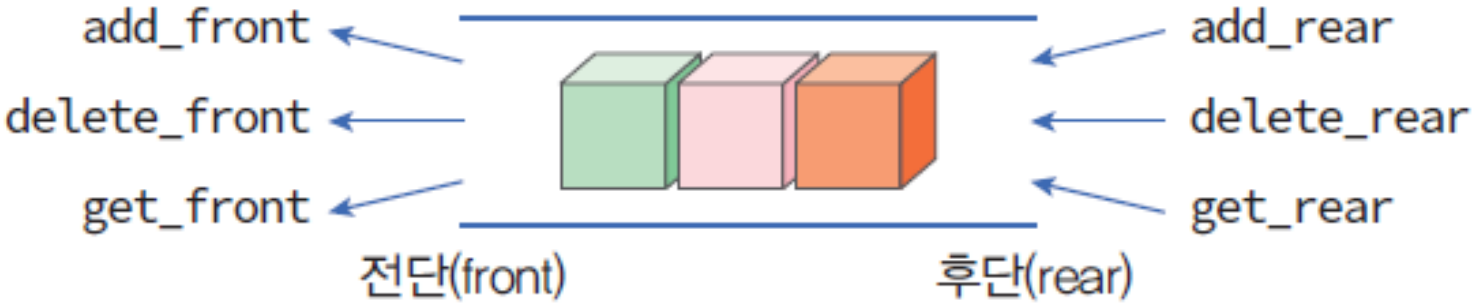
버퍼 응용 프로그램: queue_buffer.c


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

    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) = 62 |
QUEUE(front=0 rear=1) = 62 |
QUEUE(front=0 rear=1) = 62 |
QUEUE(front=0 rear=1) = 62 |
QUEUE(front=0 rear=2) = 62 | 48 |
QUEUE(front=0 rear=2) = 62 | 48 |
QUEUE(front=0 rear=2) = 62 | 48 |
QUEUE(front=0 rear=2) = 62 | 48 |
QUEUE(front=0 rear=3) = 62 | 48 | 68 |
QUEUE(front=0 rear=3) = 62 | 48 | 68 |
QUEUE(front=0 rear=3) = 62 | 48 | 68 |
QUEUE(front=0 rear=3) = 62 | 48 | 68 |
QUEUE(front=0 rear=4) = 62 | 48 | 68 | 2 |
QUEUE(front=0 rear=4) = 62 | 48 | 68 | 2 |
QUEUE(front=0 rear=4) = 62 | 48 | 68 | 2 |
QUEUE(front=0 rear=4) = 62 | 48 | 68 | 2 |
큐가 포화상태입니다
```

덱(deque)

- **덱(deque)**은 **double-ended queue의 줄임말**로서 큐의 전단(front)와 후단(rear)에서 모두 삽입과 삭제가 가능한 큐



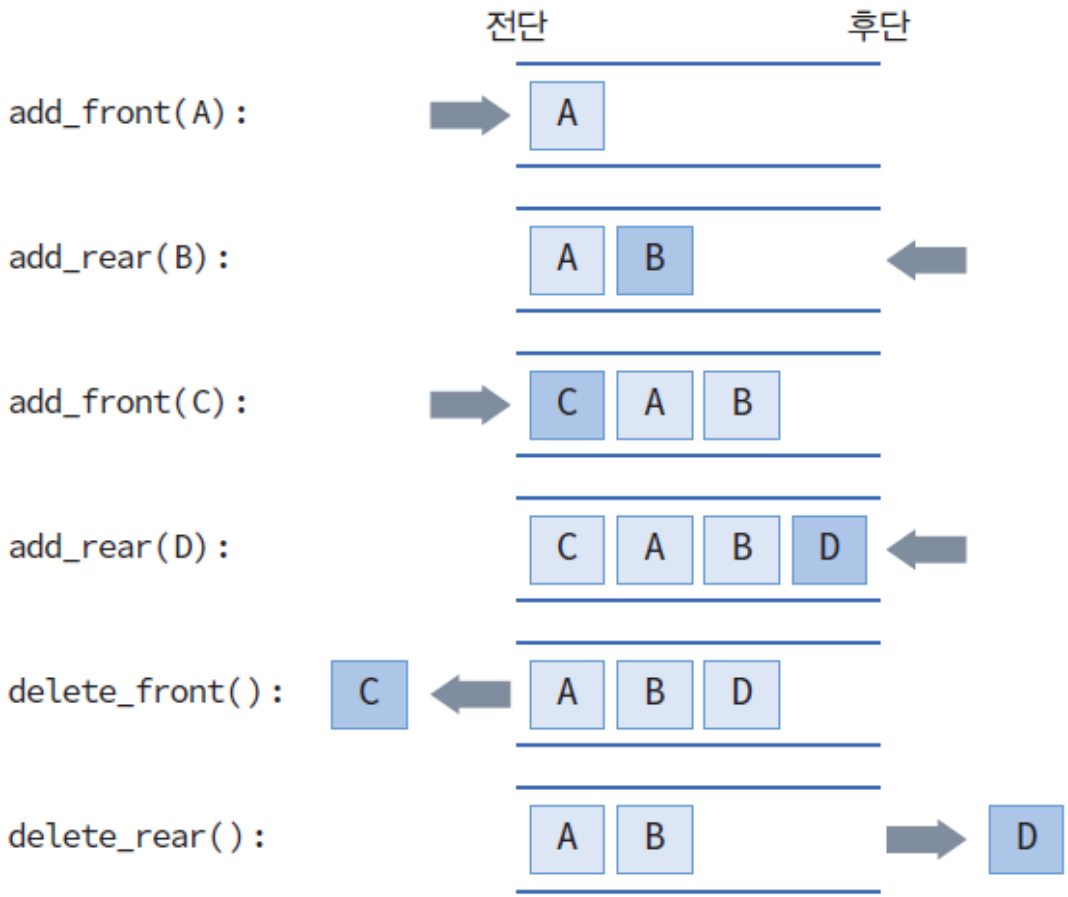
덱 ADT

·객체: n개의 element형으로 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.

덱의 연산

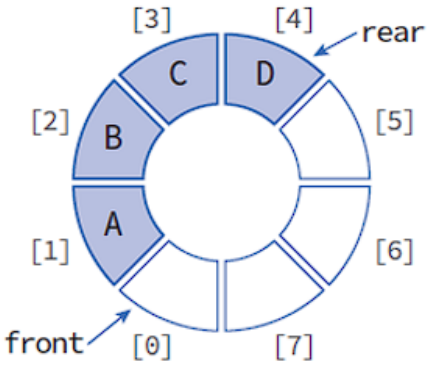

전단
후단
add_front(A):
A
add_rear(B):
A
B
add_front(C):
C
A
B
add_rear(D):
C
A
B
D
delete_front():
C
A
B
D
delete_rear():
A
B
D

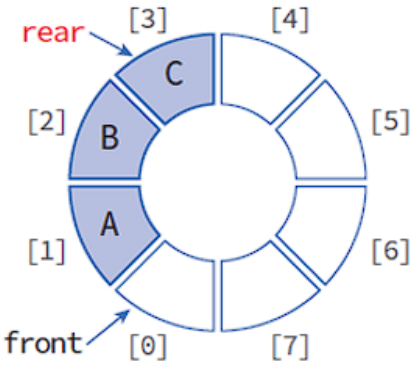
배열을 이용한 덱의 구현

add_front(): 현재 front == 0 일 때를 감안

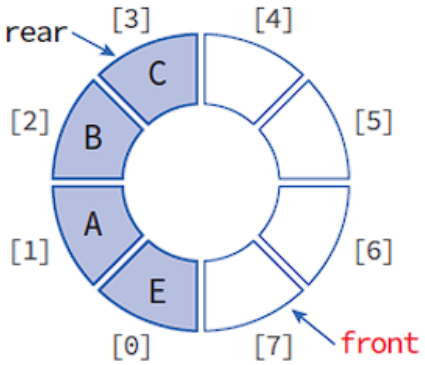
```
front ← (front-1 + MAX_QUEUE_SIZE) % MAX_QUEUE_SIZE;
rear ← (rear-1 + MAX_QUEUE_SIZE) % MAX_QUEUE_SIZE;
```

delete_rear(): 현재 rear == 0 일 때를 감안





delete_rear()



add_front(E)

Deque 프로그램: deque.c (1/4)


```c
#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(const 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);
}
```

Deque 프로그램: deque.c (2/4)


```c
// 원형 큐 출력 함수
void deque_print(DequeType *q)
{
    printf("DEQUE(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];
}
```

Deque 프로그램: deque.c (3/4)


```
70  // 삭제 함수
71  element get_front(DequeType *q)
72  {
73      if (is_empty(q))
74          error("큐가 공백상태입니다");
75      return q->data[(q->front + 1) % MAX_QUEUE_SIZE];
76  }
77
78  void add_front(DequeType *q, element val)
79  {
80      if (is_full(q))
81          error("큐가 포화상태입니다");
82      q->data[q->front] = val;
83      q->front = (q->front - 1 + MAX_QUEUE_SIZE) % MAX_QUEUE_SIZE;
84  }
85
86  element delete_rear(DequeType *q)
87  {
88      int prev = q->rear;
89      if (is_empty(q))
90          error("큐가 공백상태입니다");
91      q->rear = (q->rear - 1 + MAX_QUEUE_SIZE) % MAX_QUEUE_SIZE;
92      return q->data[prev];
93  }
94
95  element get_rear(DequeType *q)
96  {
97      if (is_empty(q))
98          error("큐가 공백상태입니다");
99      return q->data[q->rear];
100 }
```

Deque 프로그램: deque.c (4/4)


```
102  int main(void)
103  {
104      DequeType queue;
105
106      init_deque(&queue);
107      for (int i = 0; i < 3; i++) {
108          add_front(&queue, i);
109          deque_print(&queue);
110      }
111      for (int i = 0; i < 3; i++) {
112          delete_rear(&queue);
113          deque_print(&queue);
114      }
115      return 0;
116  }
```

```
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) =

--------------------------------
Process exited after 0.04956 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

큐의 응용: 시뮬레이션

- 큐를 이용한 큐잉 이론 기반 시뮬레이션
  - 은행의 고객 서비스
  - 네트워크의 패킷 처리 서비스

[그림 5-14] 은행에서의 서비스 대기큐

은행 서비스 시뮬레이션

- 시뮬레이션 파라미터
  - 서비스 창구 수 = 1
  - 총 60분 시뮬레이션
  - 각 1분 당 고객이 도착할 확률은 0.3
  - 고객 당, 서비스 시간: 1/2/3 분(랜덤)

  - 고객 도착 시에 원형 큐를 이용하여 고객 처리 업무를 enqueue
  - 큐가 empty가 아니면 서비스 창구는 dequeue 하여 업무를 처리

- 시뮬레이션에 따라 고객의 평균 대기시간을 계산하여, 서비스 창구를 늘일지를 결정

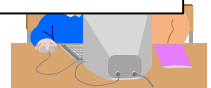
은행 서비스 시뮬레이션: bank_simul.c (1/3)


```
 6  // 프로그램 5.2에서 다음과 같은 부분을 복사한다.
 7  // ================ 원형큐 코드 시작 ================
 8  typedef struct { // 요소 타입
 9      int id;
10      int arrival_time;
11      int service_time;
12  } element;          // 교체!
13  // ================ 원형큐 코드 종료 ================
```

```
91  int main(void)
92  {
93      int minutes = 60;
94      int total_wait = 0, total_customers = 0, service_time = 0;
95      int service_customer;
96      QueueType queue;
97
98      init_queue(&queue);
99      srand(time(NULL));
```

은행 서비스 시뮬레이션: bank_simul.c (2/3)


```
101     for (int clock = 0; clock < minutes; clock++) {
102         printf("현재시각=%d\n", clock);
103 
104         // 고객 도착 시뮬레이션 (0.3의 확률)
105         if ((rand()%10) < 3) {
106             element customer;
107             customer.id = total_customers++;
108             customer.arrival_time = clock;
109             customer.service_time = rand() % 3 + 1;      // 해당 고객의 처리 시간 시뮬레이션
110             enqueue(&queue, customer);
111             printf("고객 %d이 %d분에 들어옵니다. 업무처리시간 = %d분 \n",
112                 customer.id, customer.arrival_time, customer.service_time);
113         }
114         if (service_time > 0) {
115             printf("고객 %d 업무처리중입니다. \n", service_customer);
116             service_time--;
117         }
118         else {
119             if (!is_empty(&queue)) {
120                 element customer = dequeue(&queue);
121                 service_customer = customer.id;
122                 service_time = customer.service_time;    // 해당 고객의 업무를 처리
123                 printf("고객 %d이 %d분에 업무를 시작합니다. 대기시간은 %d분이었습니다.\n",
124                     customer.id, clock, clock - customer.arrival_time);
125                 total_wait += clock - customer.arrival_time;    // 현재 고객의 대기시간 합산
126             }
127         }
128     }
129     printf("전체 대기 시간=%d분 \n", total_wait);
130     return 0;
131 }
```

은행 서비스 시뮬레이션: bank_simul.c (3/3)


```
고객 120이 49분에 들어옵니다. 업무처리시간= 3분
고객 10 업무처리중입니다.
현재시각=50
고객 130이 50분에 들어옵니다. 업무처리시간= 1분
고객 110이 50분에 업무를 시작합니다. 대기시간은 2분이었습니다.
현재시각=51
고객 11 업무처리중입니다.
현재시각=52
고객 120이 52분에 업무를 시작합니다. 대기시간은 3분이었습니다.
현재시각=53
고객 12 업무처리중입니다.
현재시각=54
고객 140이 54분에 들어옵니다. 업무처리시간= 3분
고객 12 업무처리중입니다.
현재시각=55
고객 12 업무처리중입니다.
현재시각=56
고객 150이 56분에 들어옵니다. 업무처리시간= 1분
고객 130이 56분에 업무를 시작합니다. 대기시간은 6분이었습니다.
현재시각=57
고객 13 업무처리중입니다.
현재시각=58
고객 140이 58분에 업무를 시작합니다. 대기시간은 4분이었습니다.
현재시각=59
고객 14 업무처리중입니다.
전체 대기 시간=32분
```