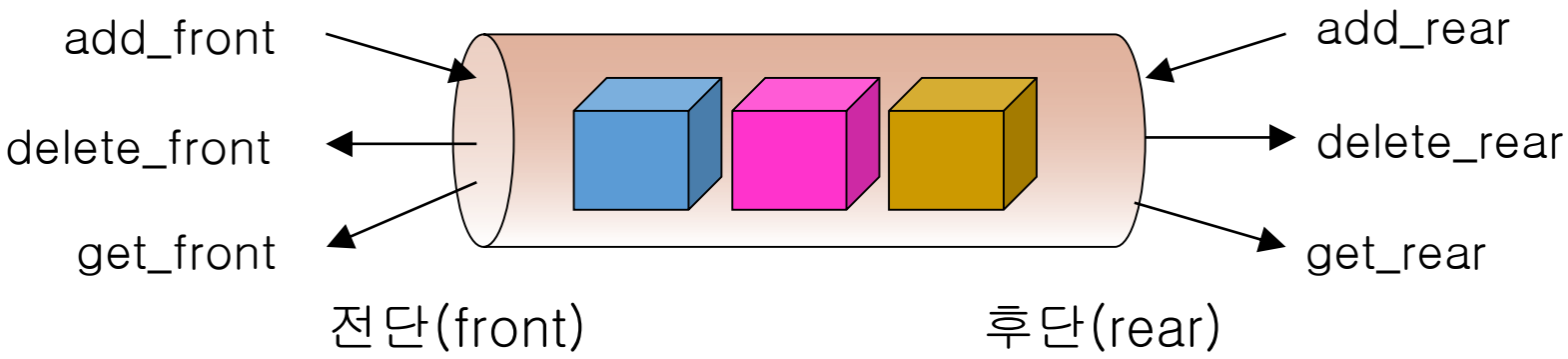
# 덱(deque)

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

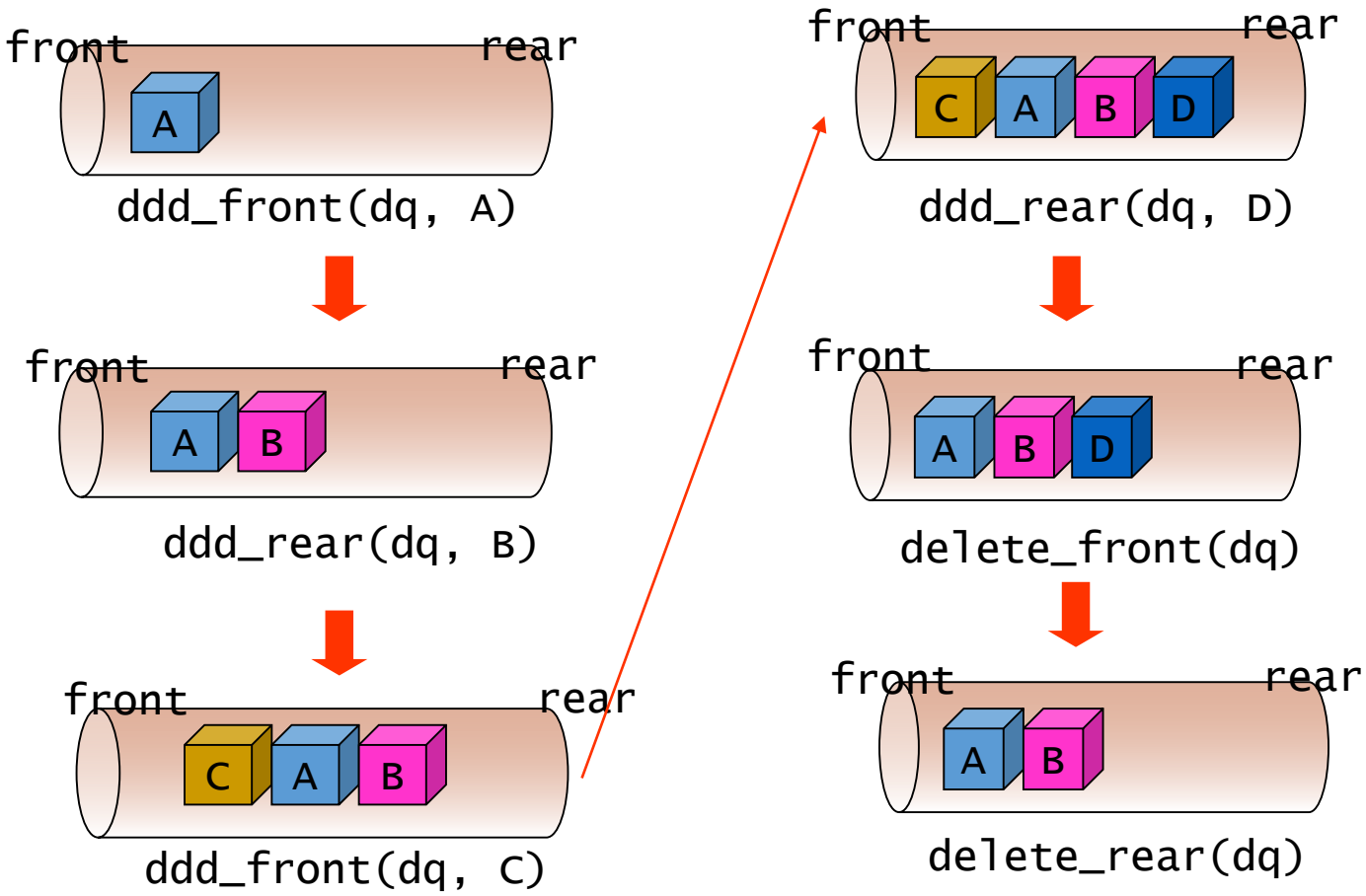


# 덱 ADT

.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


front
rear
A
ddd_front(dq, A)
front
rear
A
B
ddd_rear(dq, B)
front
rear
C
A
B
ddd_front(dq, C)
front
rear
C
A
B
D
ddd_rear(dq, D)
front
rear
A
B
D
delete_front(dq)
front
rear
A
B
delete_rear(dq)

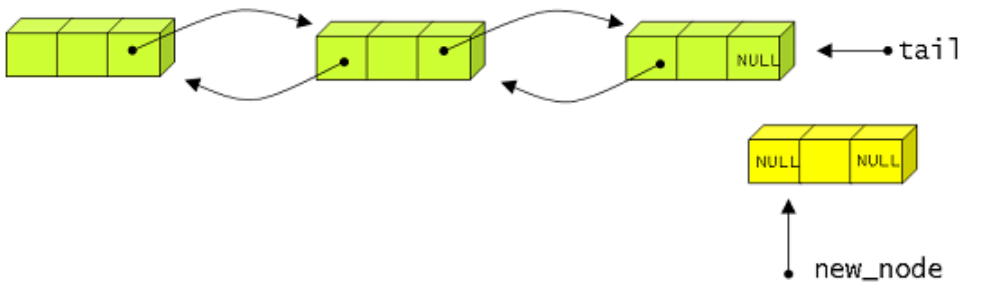
# 덱의 구현

- 양쪽에서 삽입, 삭제가 가능하여야 하므로 일반적으로 이중 연결 리스트 사용

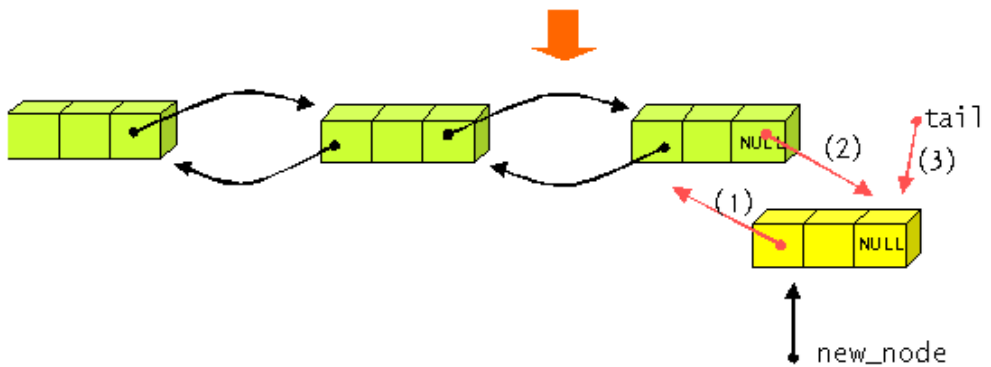
```
typedef int element;             // 요소의 타입


typedef struct DlistNode {     // 노드의 타입
     element data;
     struct DlistNode *llink;
     struct DlistNode *rlink;
} DlistNode;


typedef struct DequeType {     // 덱의 타입
     DlistNode *head;
     DlistNode *tail;
} DequeType;
```

# 덱에서의 삽입연산

- 연결리스트의 연산과 유사
- 헤드포인터 대신 head와 tail 포인터 사용




(a) 삽입하기 전




(b) 삽입한 후

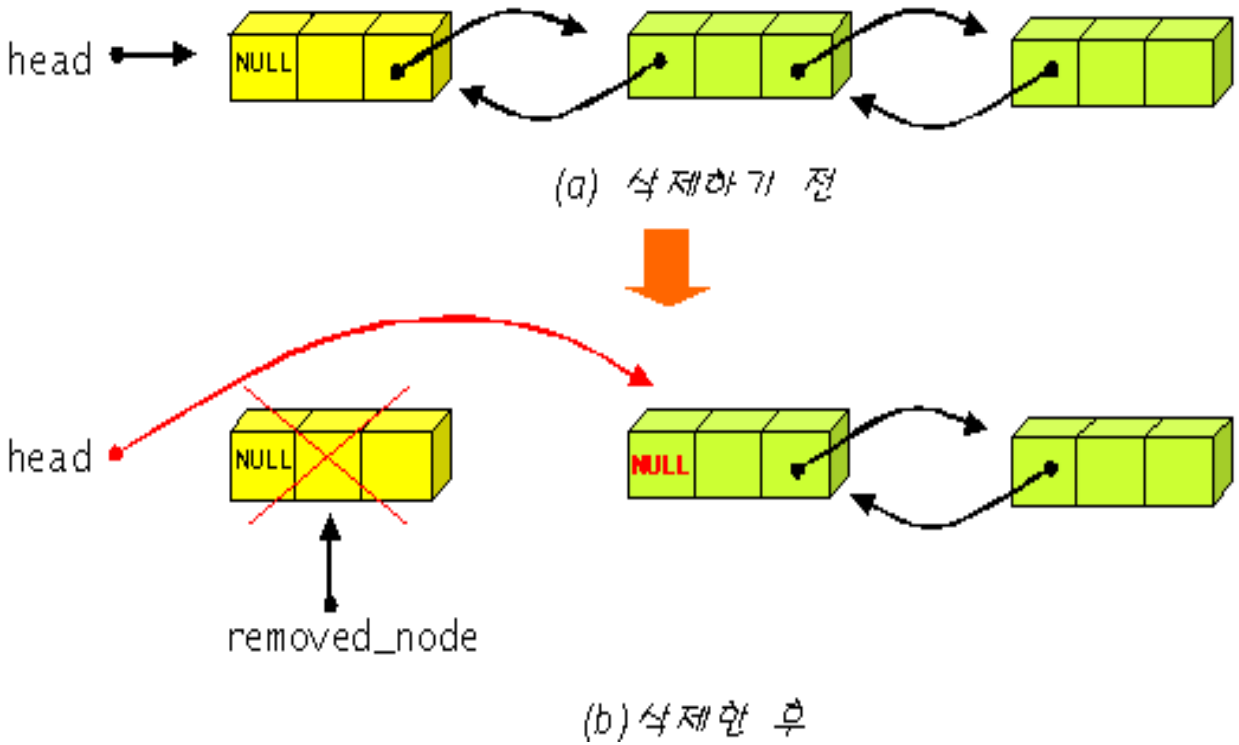
# 덱에서의 삽입연산

```
void add_rear(DequeType *dq, element item)
{
  DlistNode *new_node =  create_node(dq->tail, item, NULL);
  if( is_empty(dq))
     dq->head = new_node;
  else
     dq->tail->rlink = new_node;
  dq->tail = new_node;
}
```

# 덱에서의 삽입연산

```
//
void add_front(DequeType *dq, element item)
{
  DlistNode *new_node = create_node(NULL, item, dq->head);

  if( is_empty(dq))
         dq->tail = new_node;
  else
         dq->head->llink = new_node;
  dq->head = new_node;
}
```

# 덱에서의 삭제연산



# 덱에서의 삭제연산

```
// 전단에서의 삭제
element delete_front(DequeType *dq)
{
        element item;
        DlistNode *removed_node;

        if (is_empty(dq))  error("공백 덱에서 삭제");
        else {
                removed_node = dq->head;   // 삭제할 노드
                item = removed_node->data; // 데이터 추출
                dq->head = dq->head->rlink; // 헤드 포인터 변경
                free(removed_node);                // 메모리 공간 반납
                if (dq->head == NULL)     // 공백상태이면
                        dq->tail = NULL;
                else     // 공백상태가 아니면
                        dq->head->llink=NULL;
        }
        return item;
}
```

# 덱에서의 삭제연산

```
// 후단에서의 삭제
element delete_rear(DequeType *dq)
{
   element item;
        DlistNode *removed_node;

        if (is_empty(dq))  error("공백 덱에서의 삭제");
        else {
                removed_node = dq->tail;
                item = removed_node->data;
                dq->tail = dq->tail->llink;
                free(removed_node);

                if (dq->tail == NULL)
                        dq->head = NULL;
                else
                        dq->tail->rlink=NULL;
        }
        return item;
        }
```