큐

9주차-강의

남춘성

SUNG KYUN KWAN UNIVERSITY

큐(Queue)의 개념적 이해 - I

- 스택과 마찬가지로 삽입과 삭제의 위치가 제한되어있는 유한 순서 리스트
- 큐의 뒤에서는 삽입만 하고, 앞에서는 삭제만 할 수 있는 구조
  - 삽입한 순서대로 원소가 나열되어 가장 먼저 삽입(First-In)한 원소는 맨 앞에 있다가 가장 먼저 삭제(First-Out)
  ✓ **선입선출 구조** (**FIFO**, First-In-First-Out)

- 스택과 큐의 구조 비교

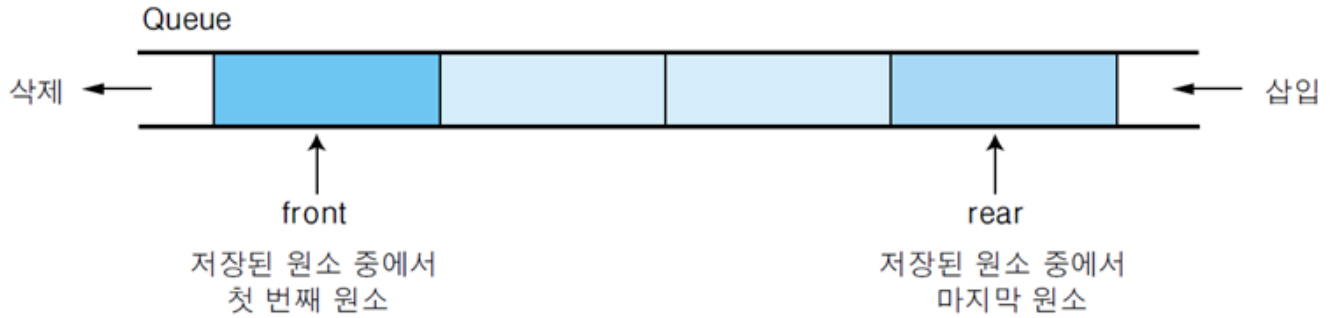
- 큐의 기본 연산
  - Enqueue : 큐에 데이터를 넣는 연산
  - Dequeue : 큐에 데이터를 꺼내는 연산

- 운영체제 관점
  - 프로세스나 쓰레드의 관리에 활용되는 자료구조.

큐(Queue)의 개념적 이해 - II

SUNG KYUN KWAN UNIVERSITY

- 큐의 구조



- 스택과 큐의 연산 비교

스택과 큐에서의 삽입과 삭제 연산 비교

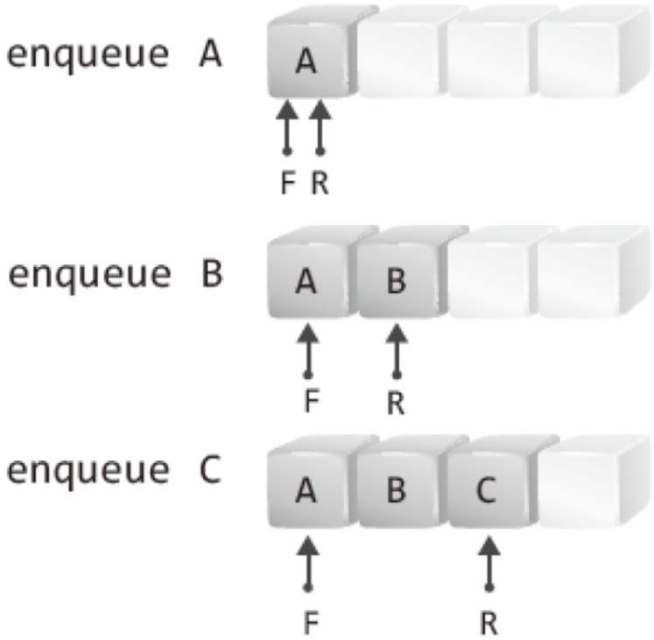
항목 / 자료구조	삽입 연산		삭제 연산	
	연산자	삽입 위치	연산자	삭제 위치
스택	push	top	pop	top
큐	enQueue	rear	deQueue	front

큐의 ADT 정의

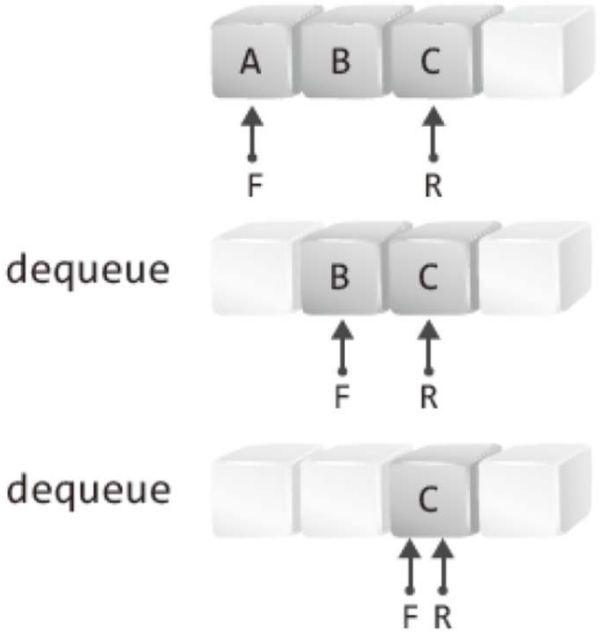
SUNG KYUN KWAN UNIVERSITY

- Void QueueInit(Queue * pq)
  - 큐 생성 후 제일 먼저 호출되는 함수 (큐의 초기화 진행)
- Int QIsEmpty(Queue *pq)
  - 큐가 빈 경우 True(1), 아닌 경우 False(0) 반환
- Void Enqueue(Queue * pq, Data data)
  - 큐에 데이터 저장(data)
- Data Dequeue(Queue * pq)
  - 저장순서가 가장 앞선 데이터 삭제
  - 삭제된 데이터 반환(데이터 하나 이상 존재해야 함)
- Data QPeek(Queue * pq)
  - 저장순서가 가장 앞선 데이터 반환(데이터 하나 이상 존재해야 함)

큐의 구현 : 배열기반 - I



Enqueue
- 큐의 꼬리 rear , R, 을 한칸 이동시키고 새 데이터 저장



Dequeue
- 큐의 머리 front, F, 를 반환하고 F를 한칸 이동

큐의 구현 : 배열기반 - II

SUNG KYUN KWAN UNIVERSITY

- 1차원 배열을 이용한 큐
  - 큐의 크기 = 배열의 크기
  - 변수 front : 저장된 첫 번째 원소의 인덱스 저장
  - 변수 rear : 저장된 마지막 원소의 인덱스 저장

- 상태 표현
  - 초기 상태 : front = rear = -1
  - 공백 상태 : front = rear
  - 포화 상태 : rear = n-1 (n : 배열의 크기, n-1 : 열의 마지막 인덱스)

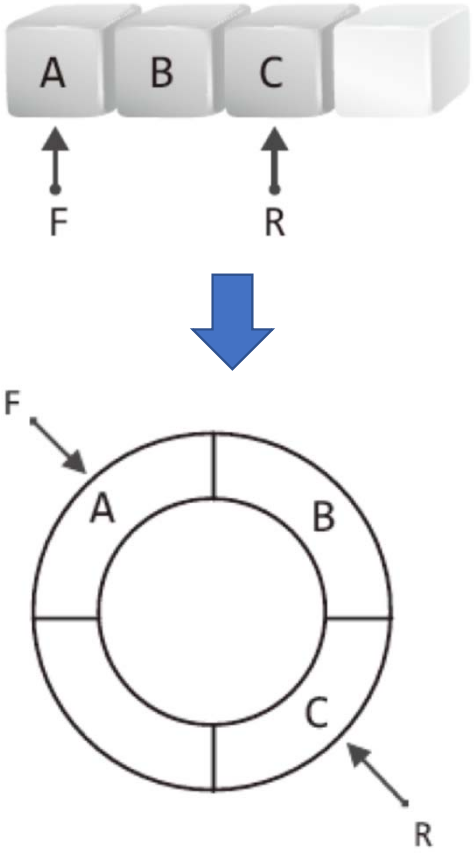
큐의 구현 : 배열기반 - 문제점 및 해결방법

- 선형 큐의 잘못된 포화상태 인식
  - 큐에서 삽입과 삭제를 반복하면서 아래와 같은 상태일 경우, 앞부분에 빈자리가 있지만 rear=n-1 상태이므로 포화상태로 인식하고 더 이상의 삽입을 수행하지 않음

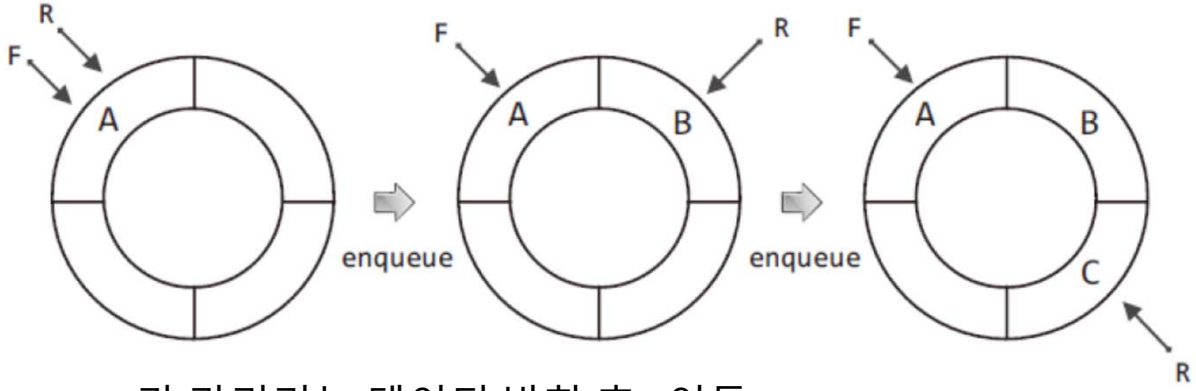
- 선형 큐의 잘못된 포화상태 인식의 해결 방법-1
  - 저장된 원소들을 배열의 앞부분으로 이동시키기
    - 순차자료에서의 이동 작업은 연산이 복잡하여 효율성이 떨어짐

- 선형 큐의 잘못된 포화상태 인식의 해결 방법-2
  - 1차원 배열을 사용하면서 논리적으로 배열의 처음과 끝이 연결되어 있다고 가정하고 사용 ⇒ **원형큐**
  - 원형 큐의 논리적 구조

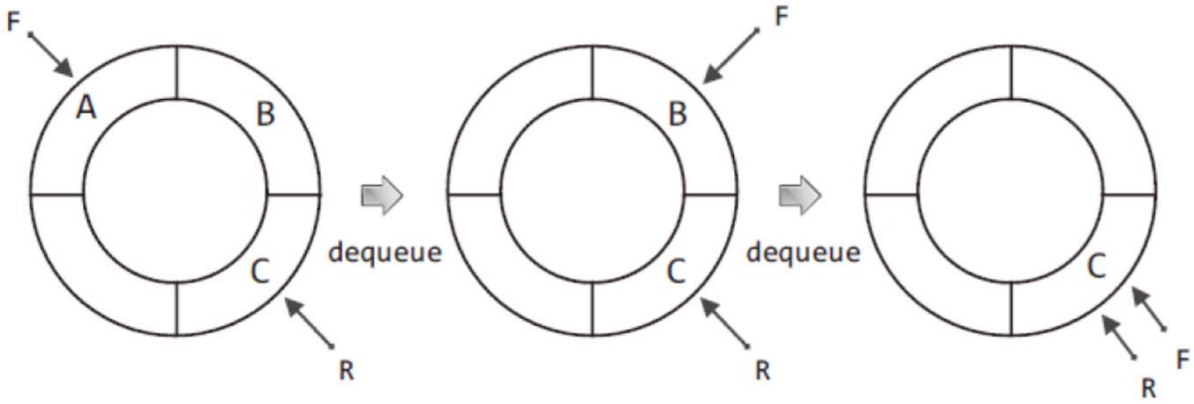
원형 큐의 개념적 이해



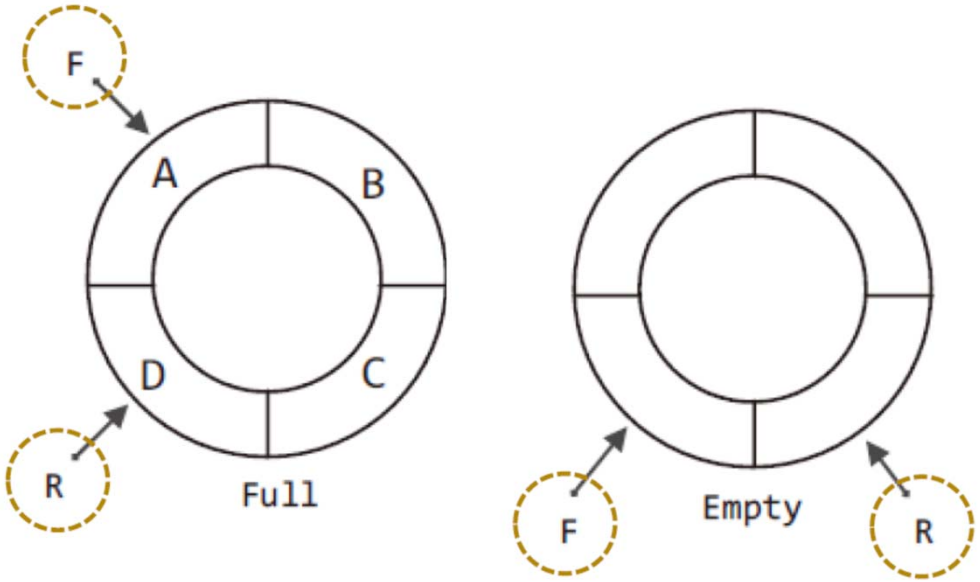
Enqueue : R이 이동한 다음 데이터 저장



Dequeue : F가 가리키는 데이터 반환 후 F이동

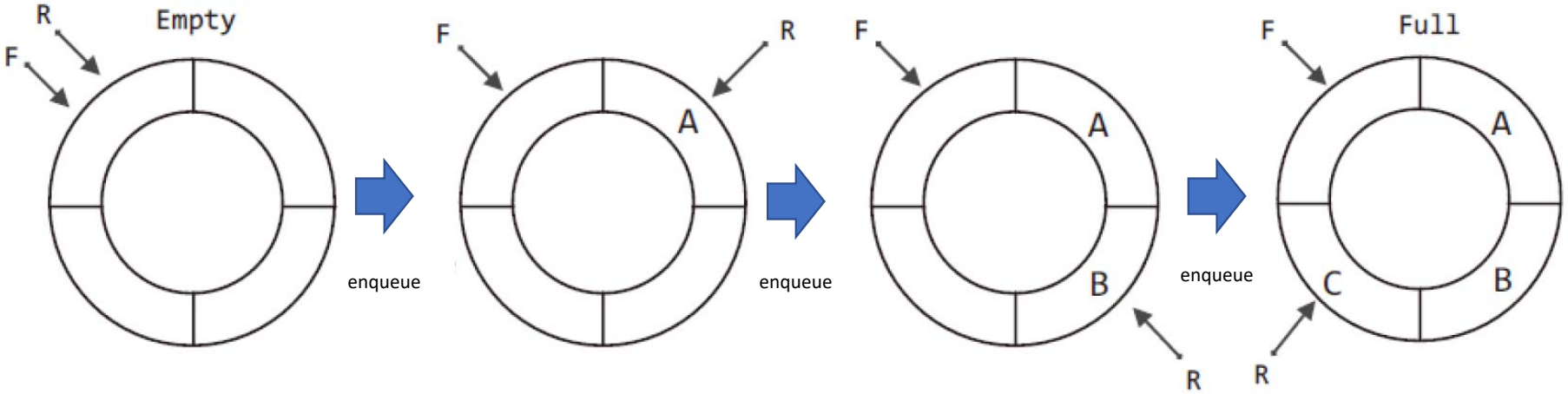


원형 큐 연산 문제점



Full 인경우와 Empty인 경우 모두 F가 R보다 한칸 앞선 위치에 가리킴을 알 수 있음

원형 큐 연산 문제 해결 방법 - I



데이터가 하나 존재하는 경우 F와 R이 다른 위치를 가리키게 함(isEmpty) 그리고 enqueue는 R을 증가해서 데이터를 넣고, dequeuer는 F를 증가해서 가리키는 값을 빼고, Full의 상태는 R의 증가한 값이 F와 같으면 Full로 인식함.

다만, 배열의 하나의 저장 공간은 잃음.

원형 큐 연산 문제 해결 방법 - II

SUNG KYUN KWAN UNIVERSITY

- 원형 큐의 구조
  - 초기 공백 상태 : front = rear = 0
  - front와 rear의 위치가 배열의 마지막 인덱스 n-1에서 논리적인 다음 자리인 인덱스 0번으로 이동하기 위해서 나머지연산자 mod를 사용
    - 3 ÷ 4 = 0 ...**3** (몫=0, 나머지=3)
    - 3 **mod** 4 = **3**

	삽입위치	삭제위치
선형큐	rear = rear + 1	front = front + 1
원형큐	rear = (rear+1) mod n	front = (front+1) mod n

  - 사용조건) 공백 상태와 포화 상태 구분을 쉽게 하기 위해서 front가 있는 자리는 사용하지 않고 항상 빈자리로 둔다.

원형 큐 구현 : 헤더파일 및 큐의 위치를 위한 함수

SUNG KYUN KWAN UNIVERSITY

```
#define QUE_LEN 100
typedef int Data;

typedef struct _cQueue{
	int front;
	int rear;
	Data queArr[QUE_LEN];
}CQueue;

typedef CQueue Queue;

void QueueInit(Queue  * pq);
int QIsEmpty(Queue * pq);
void Enqueue(Queue * pq, Data data);
Data Dequeue(Queue * pq);
Data QPeek(Queue * pq);

#endif
```

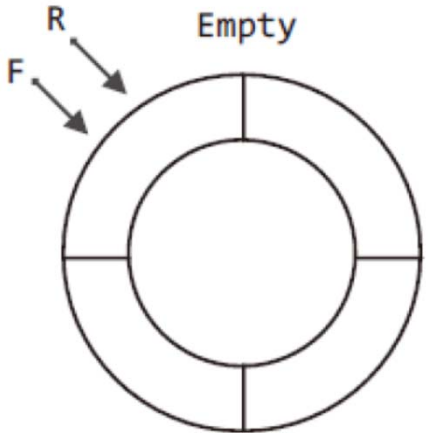
```
int NextPostIdx(int pos) {
	if (pos == QUE_LEN - 1)
		return 0;
	else
	return pos + 1;
}
```

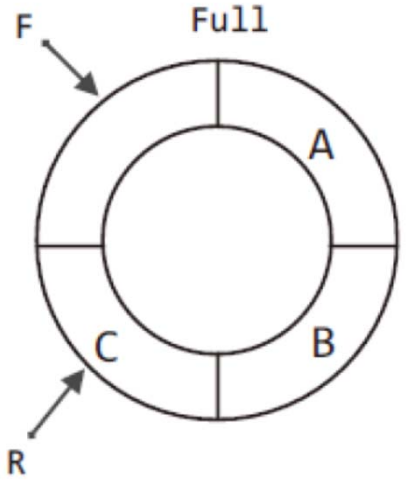
큐의 연산에 의해 F와 R이 이동할 때 위치를 알려주는 함수 구현.

원형 큐 구현 : 초기화 및 공백검사

```
void QueueInit(Queue * pq){
        pq->front = 0;
        pq->rear = 0;
}
```

```
int QIsEmpty(Queue * pq) {
        if (pq->front == pq->rear)
                return TRUE;
        else
                return FALSE;

}
```





원형 큐 구현 : enqueue , dequeue, qpeek

SUNG KYUN KWAN UNIVERSITY

```
void Enqueue(Queue * pq, Data data){
	if (NextPostIdx(pq->rear) == pq->front) {
		printf("Queue memeroy is Full!");
		exit(-1);
	}

	pq->rear = NextPostIdx(pq->rear);
	pq->queArr[pq->rear] = data;
}
```

```
Data Dequeue(Queue * pq) {
	if (QIsEmpty(pq)) {
		printf("Queue is Empty!");
		exit(-1);
	}

	pq->front = NextPostIdx(pq->front);
	return pq->queArr[pq->front];
}
```

```
Data QPeek(Queue * pq) {
	if (QIsEmpty(pq)) {
	printf("Queue is Empty!");
	exit(-1);
	}

	return pq->queArr[NextPostIdx(pq->front)];
}
```

원형 큐 구현 : main

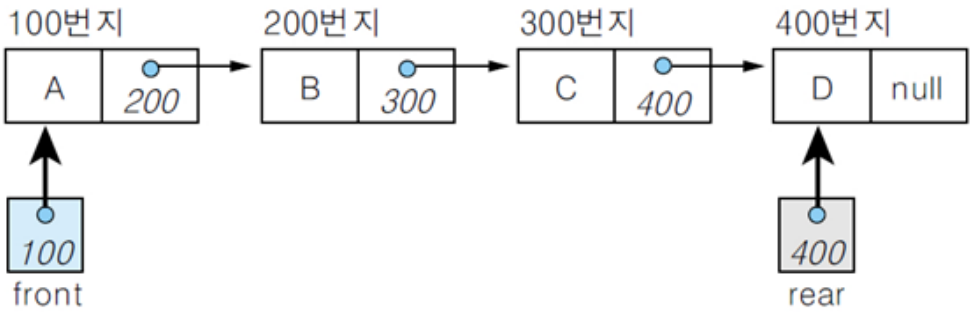
SUNG KYUN KWAN UNIVERSITY

```
int main(void){
	Queue q;//Q 생성
	QueueInit(&q);//Q 초기화

	Enqueue(&q, 1);//Q 데이터 넣기
	Enqueue(&q, 2);
	Enqueue(&q, 3);
	Enqueue(&q, 4);
	Enqueue(&q, 5);

	while (!QIsEmpty(&q))//Q 데이터 꺼내기
	printf("%d ", Dequeue(&q));

	return 0;
}
```

출력 : 1 2 3 4 5

연결 리스트 기반 큐의 개념

- 단순 연결 리스트를 이용한 큐
  - 큐의 원소 : 단순 연결 리스트의 노드
  - 큐의 원소의 순서 : 노드의 링크 포인터로 연결
  - 변수 front : 첫 번째 노드를 가리키는 포인터 변수
  - 변수 rear : 마지막 노드를 가리키는 포인터 변수
- 상태 표현
  - 초기 상태와 공백 상태 : front = rear = null
- 연결 큐의 구조



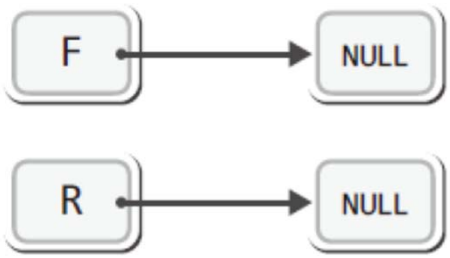
연결 리스트 기반 큐 : 헤더파일

SUNG KYUN KWAN UNIVERSITY

```
typedef int Data;

typedef struct _node {
	Data data;
	struct _node * next;
}Node;

typedef struct _LQueue{
	Node * front;
	Node * rear;
}LQueue;

typedef LQueue Queue;

void QueueInit(Queue * pq);
int QIsEmpty(Queue * pq);
void Enqueue(Queue * pq, Data data);
Data Dequeue(Queue * pq);
Data QPeek(Queue *pq);
```

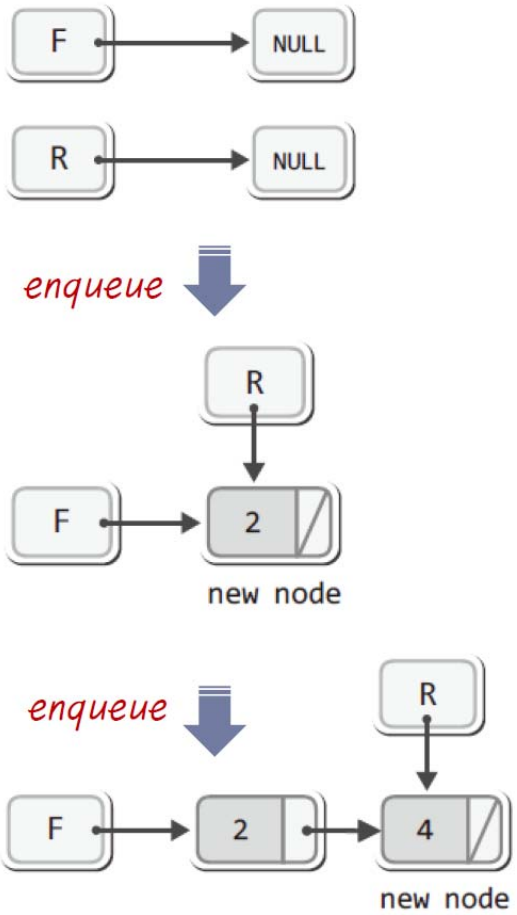
연결 리스트 기반 큐 : 초기화 및 공백검사

```
void QueueInit(Queue * pq){
	pq->front = NULL;
	pq->rear = NULL;
}
```



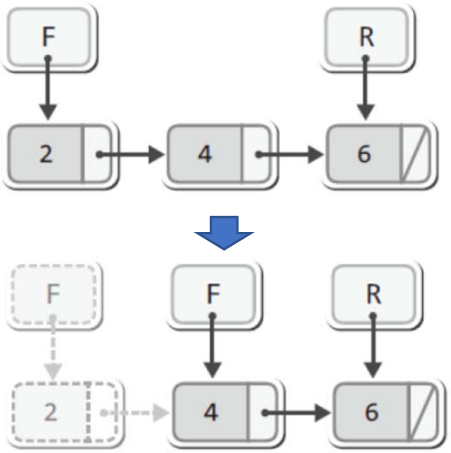

```
int QIsEmpty(Queue * pq) {
	if (pq->front == NULL)
		return TRUE;
	else
		return FALSE;
}
```

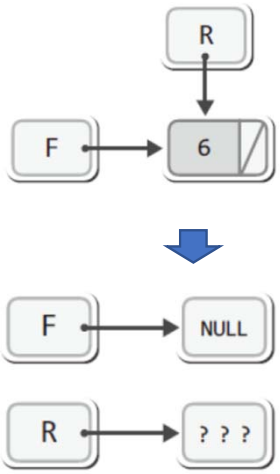
연결 리스트 기반 큐 : enqueue




```
void Enqueue(Queue * pq, Data data) {
	Node * newNode = (Node *)malloc(sizeof(Node));
	newNode->next = NULL;
	newNode->data = data;

	if (QIsEmpty(pq)){
		pq->front = newNode;
		pq->rear = newNode;
	}
	else {
		pq->rear->next = newNode;
		pq->rear = newNode;
	}
}
```

연결 리스트 기반 큐 : dequeue





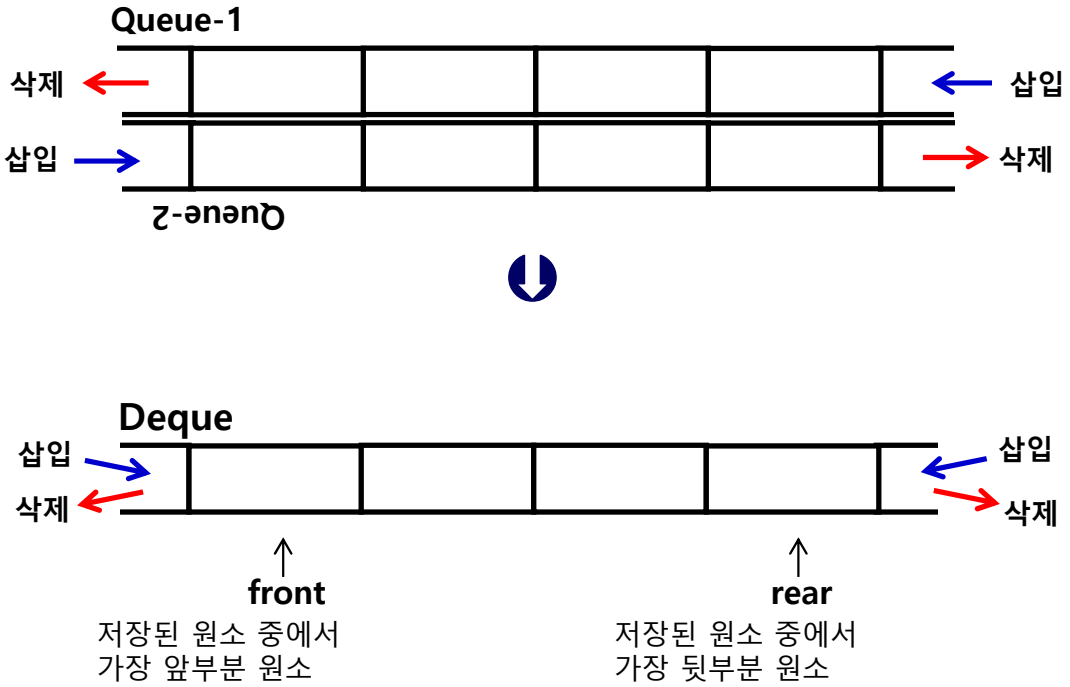

```
Data Dequeue(Queue * pq) {
        Node * delNode;
        Data retData;

        if (QIsEmpty(pq)) {
                printf("Queue is Empty!");
                exit(-1);
        }

        delNode = pq->front;
        retData = delNode->data;
        pq->front = pq->front->next;

        free(delNode);
        return retData;
}
```

R의 값을 변경해야 하는가?
아니면 그냥 내버려 두어야 하는가?

Qpeek과 main은 같음

덱(Deque, double-ended queue)의 개념적 이해

- 큐 2개를 반대로 붙여서 만든 자료구조
  - 앞으로도 뒤로도 넣을 수 있고, 앞으로도 뒤로도 꺼낼 수 있는 구조
  - 이중연결구조가 쉬움 이유는?



덱(Deque)의 ADT

SUNG KYUN KWAN UNIVERSITY

- Void DequeInit(Deque * pdeq) : 초기화, 맨먼저

- Int DQIsEmpty(Deque * pdeq) : True or False 로 큐의 상태

- Void DQAddFirst(Deque * pdeq, Data data) : 큐 앞으로 넣기
- Void DQAddLast(Deque * pdeq, Data data) : 큐 뒤로 넣기

- Data DQRemoveFirst(Deque * pdeq) : 큐 앞으로 꺼내기
- Data DQRemoveLast(Deque * pdeq) : 큐 뒤로 꺼내기

- Data DQGetFirst(Deque * pdeq) : 큐 앞의 데이터 반환
- Data DQGetLast(Deque * pdeq) : 큐 뒤의 데이터 반환

덱(Deque)의 ADT를 이용하여 Dequeue을 완성하시요.

SUNG KYUN KWAN UNIVERSITY

- 덱을 PPT의 ADT 혹은 자신만의 ADT를 완성하여 덱을 구현하시오.
  - 다음 enqueue을 실행하고
    - enqueueFirst(3), enqueueLast(4), enqueueLast(6), enqueueLast(9) enqueueFirst(1), enqueueFirst(10), enqueueLast(5), enqueueFirst(7), enqueueLast(11)
  - 다음 dequeue를 실행하여
    - dequeueFirst(), dequeueLast(), dequeueFirst(), dequeueLast(), dequeueFirst(), dequeueLast(), dequeueFirst(), dequeueLast(), dequeueFirst()
  - 출력값을 다음과 같이 나타나도록 함
    - 7,11, 10, 5, 9, 3, 6, 4