e-learning

2020 학년도 1 학기

컴퓨터 정보과

자료구조(Data Structures)

담당교수 : 김주현

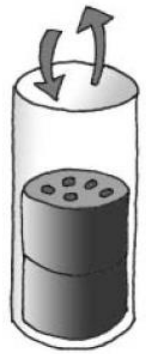
제 4 주차 / 제 3 차시

DIT 동의과학대학교
DONG-EUI INSTITUTE OF TECHNOLOGY

큐(Queue)

- 스택과 마찬가지로 삽입과 삭제의 위치가 제한된 유한 순서 리스트
- 큐의 뒤에서는 삽입만 하고, 앞에서는 삭제만 할 수 있는 구조
- 선입선출 구조 (FIFO, First-In-First-Out)
  - 삽입한 순서대로 원소가 나열되어 가장 먼저 삽입(First-In)한 원소는 맨 앞에 있다가 가장 먼저 삭제(First-Out)된다.
  - 일상생활의 관공서, 은행, 우체국, 병원 등에서 번호표를 이용한 줄서기가 대표적인 큐

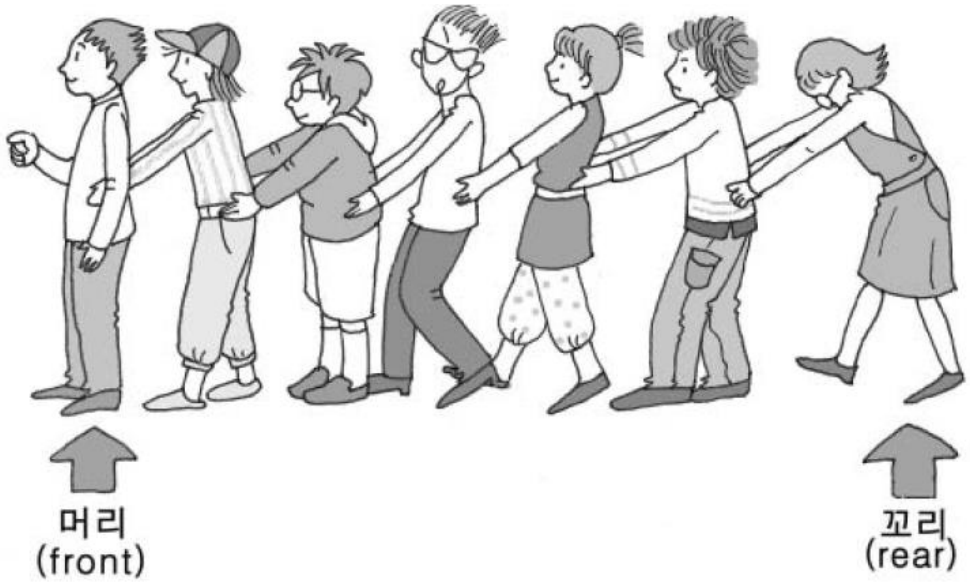
스택과 큐의 구조 비교

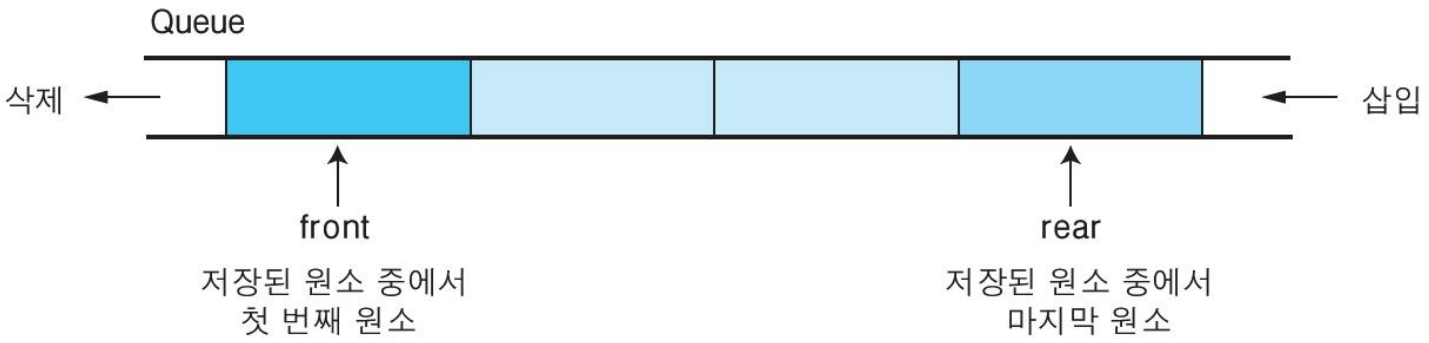


〈스택: 후입선출〉

〈큐: 선입선출〉

큐의 구조


머리
(front)
꼬리
(rear)


Queue
삭제
삽입
front
저장된 원소 중에서
첫 번째 원소
rear
저장된 원소 중에서
마지막 원소

큐의 연산

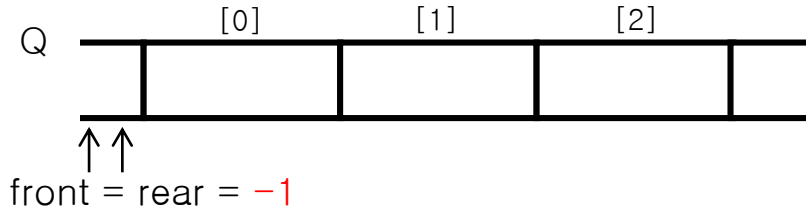
- 삽입 : enqueue
- 삭제 : deQueue

스택과 큐의 연산 비교

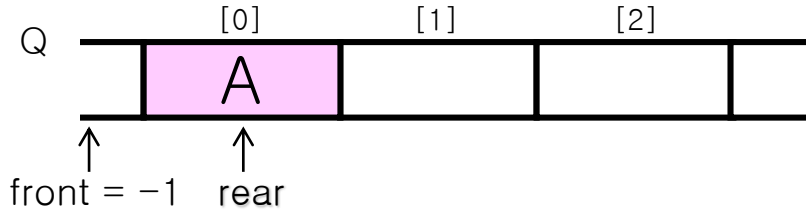
항목 / 자료구조	삽입 연산		삭제 연산	
	연산자	삽입 위치	연산자	삭제 위치
스택	push	top	pop	top
큐	enQueue	rear	deQueue	front

큐의 연산 과정

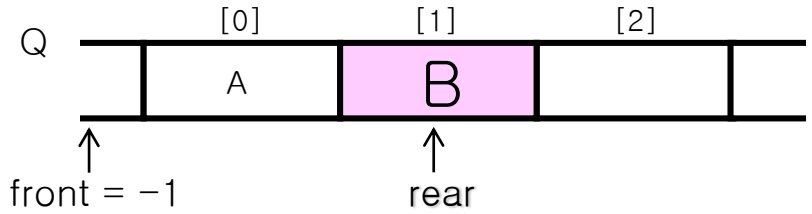
① 공백 큐 생성 : createQueue();



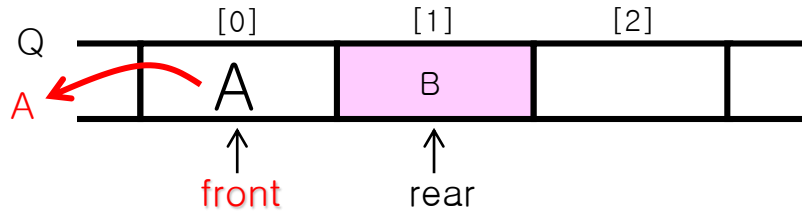
② 원소 A 삽입 : enQueue(Q, A);



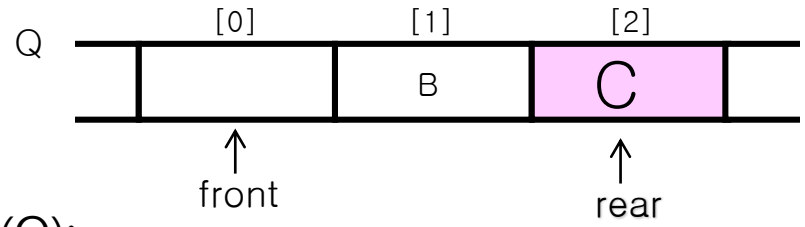
① 원소 B 삽입 : enQueue(Q, B);



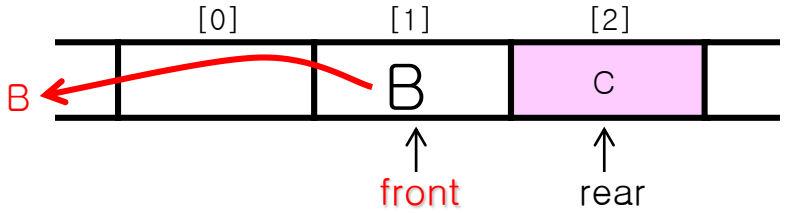
④ 원소 삭제 : deQueue(Q);



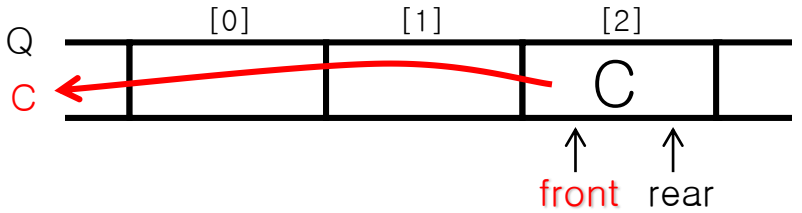
⑤ 원소 C 삽입 : enQueue(Q, C);



⑥ 원소 삭제 : deQueue(Q);



⑦ 원소 삭제 : deQueue(Q);



❖선형 큐

- 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 : 배열의 마지막 인덱스)

```
interface Queue{
        boolean isEmpty();
        void enQueue(char item);
        char deQueue();
        void delete();
        char peek();
}

class ArrayQueue implements Queue{
        private int front;
        private int rear;
        private int queueSize;
        private char itemArray[];

        public ArrayQueue(int queueSize){
                front = -1;
                rear = -1;
                this.queueSize = queueSize;
                itemArray = new char[this.queueSize];
        }
```

```
public boolean isEmpty(){
        return (front == rear);
}

public boolean isFull(){
        return (rear == this.queueSize-1);
}

public void enQueue(char item){
        if(isFull()){
                System.out.println("Inserting fail! Array Queue is full!!");
        }
        else{
                itemArray[++rear] = item;
                System.out.println("Inserted Item : " + item);
        }
}
```

```
        public char deQueue(){
                if(isEmpty()) {
                        System.out.println("Deleting fail! Array Queue is empty!!");
                        return 0;
                }
                else{
                        return itemArray[++front];
                }

        }

        public void delete(){
                if(isEmpty()){
                        System.out.println("Deleting fail! Array Queue is empty!!");
                }
                else {
                        ++front;
                }
        }
```

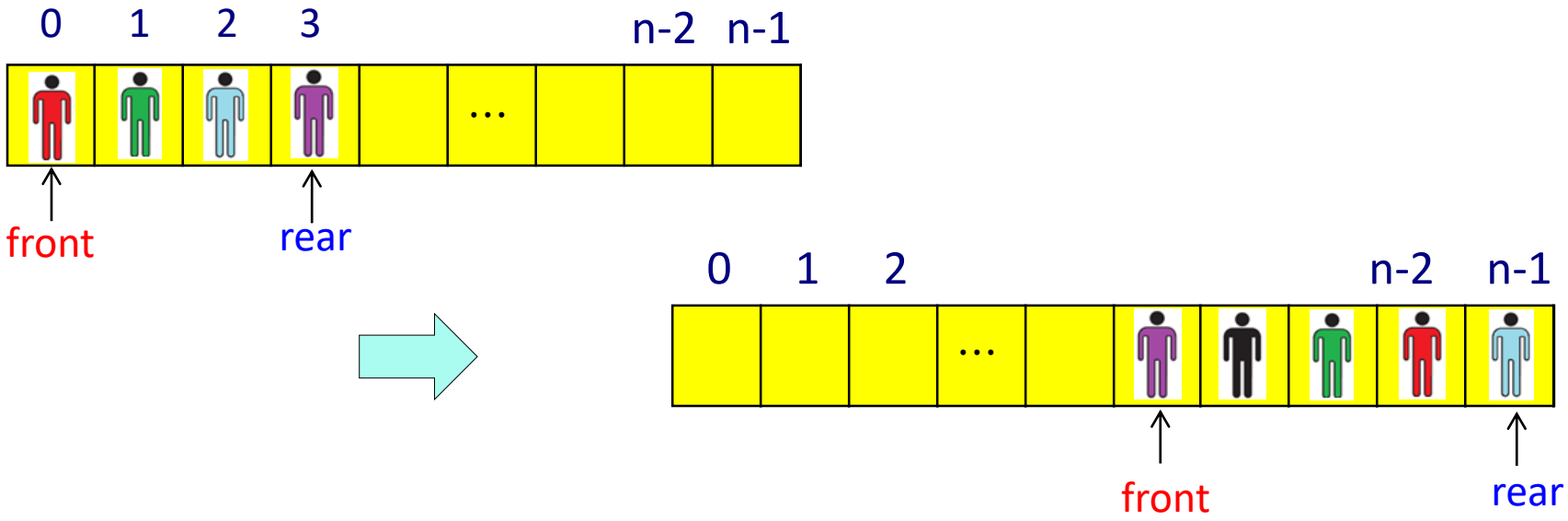
```
        public char peek(){
                if(isEmpty()){
                        System.out.println("Peeking fail! Array Queue is empty!!");
                        return 0;
                }
                else
                        return itemArray[front+1];
        }

        public void printQueue(){
                if(isEmpty())
                        System.out.printf("Array Queue is empty!! %n %n");
                else{
                        System.out.printf("Array Queue>> ");
                        for(int i=front+1; i<=rear; i++)
                                System.out.printf("%c ", itemArray[i]);
                        System.out.println();System.out.println();
                }
        }

}
```

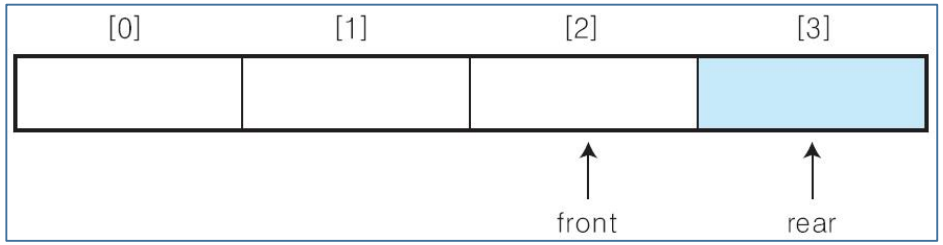
```
class Queue_Test{
        public static void main(String args[]){
                int queueSize = 3;
                char deletedItem;
                ArrayQueue Q = new ArrayQueue(queueSize);

                Q.enQueue('A');
                Q.printQueue();

                Q.enQueue('B');
                Q.printQueue();

                deletedItem = Q.deQueue();
                if(deletedItem != 0)
                        System.out.println("deleted Item : " + deletedItem);
                Q.printQueue();

                Q.enQueue('C');
                Q.printQueue();
```

```
                deletedItem = Q.deQueue();
                if(deletedItem != 0)
                        System.out.println("deleted Item : " + deletedItem);
                Q.printQueue();

                deletedItem = Q.deQueue();
                if(deletedItem != 0)
                        System.out.println("deleted Item : " + deletedItem);
                Q.printQueue();

                deletedItem = Q.deQueue();
                if(deletedItem != 0)
                        System.out.println("deleted Item : " + deletedItem);
                Q.printQueue();

                deletedItem = Q.deQueue();
                if(deletedItem != 0)
                        System.out.println("deleted Item : " + deletedItem);
                Q.printQueue();
        }
}
```

```
Inserted Item : A
Array Queue>> A

Inserted Item : B
Array Queue>> A B

deleted Item : A
Array Queue>> B

Inserted Item : C
Array Queue>> B C

deleted Item : B
Array Queue>> C

deleted Item : C
Array Queue is empty!!

Deleting fail! Array Queue is empty!!
Array Queue is empty!!
```

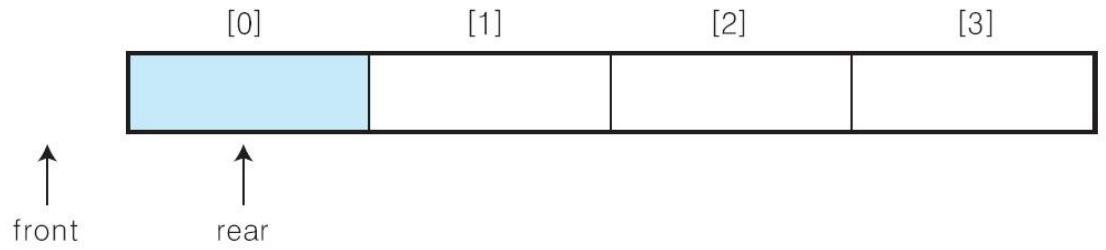
선형 큐의 잘못된 포화상태 인식





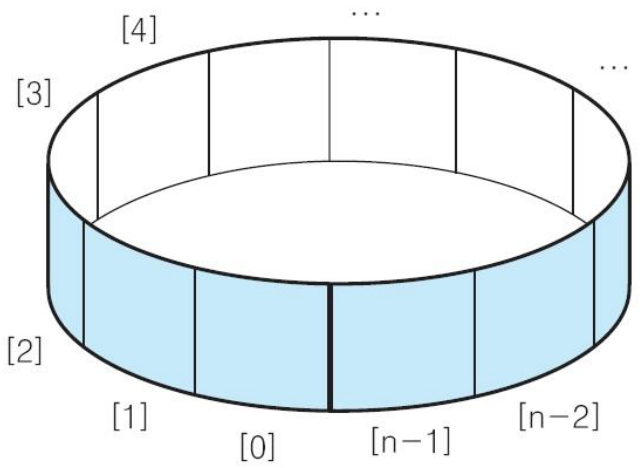
선형 큐의 잘못된 포화상태 인식의 해결 방법-1

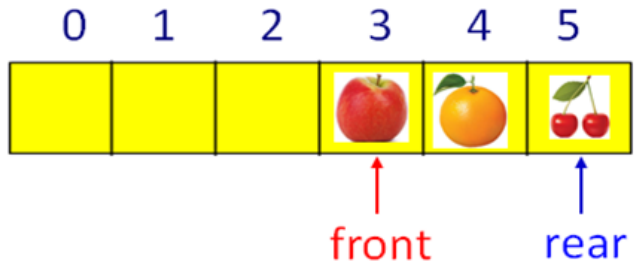
- 저장된 원소들을 배열의 앞부분으로 이동시키기
- 순차자료에서의 이동 작업은 연산이 복잡하여 효율성이 떨어짐

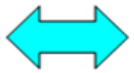


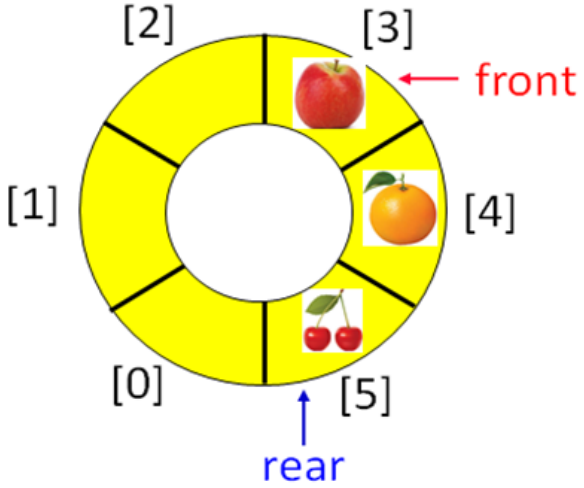
선형 큐의 잘못된 포화상태 인식의 해결 방법-2

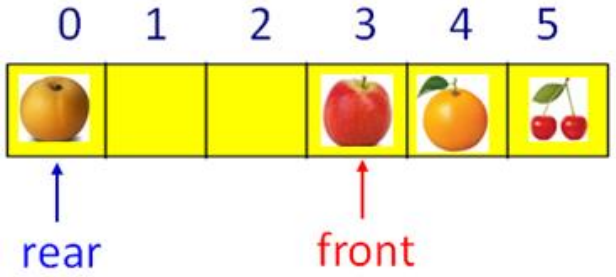
- 1차원 배열을 사용하면서 논리적으로 배열의 처음과 끝이 연결되어 있다고 가정하고 사용
- 원형 큐의 논리적 구조

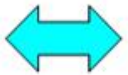


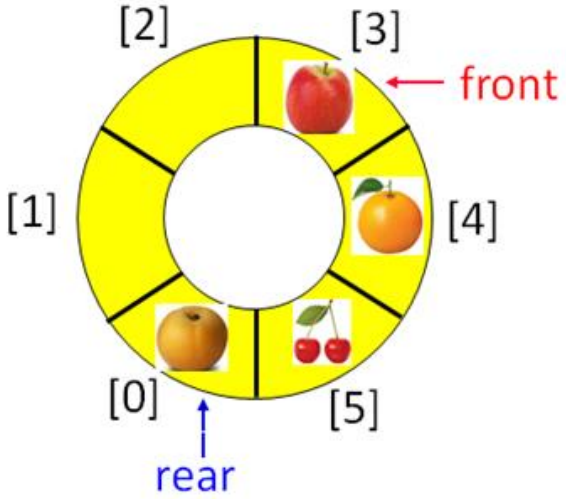

0 1 2 3 4 5
front
rear




[2]
[3]
front
[1]
[4]
[0]
[5]
rear


0 1 2 3 4 5
rear
front




[2]
[3]
front
[1]
[4]
[0]
[5]
rear

rear = (rear + 1) % N

```
interface Queue{
        boolean isEmpty();
        void enQueue(char item);
        char deQueue();
        void delete();
        char peek();
}

class ArrayCQueue implements Queue{
        private int front;
        private int rear;
        private int queueSize;
        private char itemArray[];

        public ArrayCQueue(int queueSize){
                front = 0;
                rear = 0;
                this.queueSize = queueSize;
                itemArray = new char[this.queueSize];
        }
```

```
public boolean isEmpty(){
        return (front == rear);
}

public boolean isFull(){
        return (((rear+1) % this.queueSize) == front);
}

public void enQueue(char item){
        if(isFull()){
                System.out.println("Inserting fail! Array Circular Queue is full!!");
        }
        else{
                rear = (rear+1) % this.queueSize;
                itemArray[rear] = item;
                System.out.println("Inserted Item : " + item);
        }
}
```

```
	public char deQueue(){
		if(isEmpty()) {
			System.out.println("Deleting fail! Array  Circular Queue is empty!!");
			return 0;
		}
		else{
			front = (front+1) % this.queueSize;
			return itemArray[front];
		}

	}

	public void delete(){
		if(isEmpty()){
			System.out.println("Deleting fail! Array  Circular Queue is empty!!");
		}
		else {
			front = (front+1) % this.queueSize;
		}
	}
```

```
        public char peek(){
                if(isEmpty()){
                        System.out.println("Peeking fail! Array  Circular Queue is empty!!");
                        return 0;
                }
                else
                        return itemArray[(front+1) % this.queueSize];
        }

        public void printQueue(){
                if(isEmpty())
                        System.out.println("Array Circular Queue is empty!!");
                else{
                        System.out.printf("Array Circular Queue>> ");
                        for(int i=(front+1) % this.queueSize; i!=(rear+1)% this.queueSize; i=++i % this.queueSize)
                                System.out.printf("%c ", itemArray[i]);
                        System.out.println();  System.out.println();
                }
        }

}
```

```
class CircularQ_Test{
        public static void main(String args[]){
                int queueSize = 4;
                char deletedItem;
                ArrayCQueue cQ = new ArrayCQueue(queueSize);

                cQ.enQueue('A');
                cQ.printQueue();

                cQ.enQueue('B');
                cQ.printQueue();

                deletedItem = cQ.deQueue();
                if(deletedItem != 0)
                        System.out.println("deleted Item : " + deletedItem);
                cQ.printQueue();
```

```
            cQ.enQueue('C');
            cQ.printQueue();

            cQ.enQueue('D');
            cQ.printQueue();

            cQ.enQueue('E');
            cQ.printQueue();
        }
}
```

```
Inserted Item : A
Array Circular Queue>> A

Inserted Item : B
Array Circular Queue>> A B

deleted Item : A
Array Circular Queue>> B

Inserted Item : C
Array Circular Queue>> B C

Inserted Item : D
Array Circular Queue>> B C D

Inserting fail! Array Circular Queue is full!!
Array Circular Queue>> B C D
```

```
import java.util.NoSuchElementException;
public class ArrayQueue <E>{
        private E[] q;    // 큐을 위한 배열
        private int front, rear, size;
        public ArrayQueue() {        // 큐 생성자
                q = (E[]) new Object[2]; // 초기에 크기가 2인 배열 생성
                front = rear = size = 0;
        }
        public int     size()    { return size;}        // 큐에 있는 항목의 수를 리턴
        public boolean isEmpty() { return (size == 0);} // 큐가 empty이면 true를 리턴
        public void add(E newItem) {   // 큐 삽입 연산
                if ((rear+1)%q.length == front) // 비어있는 원소가 1개뿐인 경우(즉, 큐가 full인 경우)
                resize(2*q.length);    // 큐의 크기를 2배로 확장
                rear = (rear+1) % q.length;
                q[rear] = newItem;         // 새 항목을 add
                size++;
        }
```

```
public E remove() {  //큐 삭제 연산
        if (isEmpty()) throw new NoSuchElementException(); // underflow 경우에 프로그램 정지
        front  = (front+1) % q.length;
        E item = q[front];
        q[front] = null;    // null로 만들어 가비지 컬렉션되도록
        size--;
        if (size > 0 && size == q.length/4) // 큐의 항목수가  배열 크기의 1/4가 되면
                resize(q.length/2);              // 큐를 1/2 크기로 축소
        return item;
}
private void resize(int newSize) {    // 큐릐 배열 크기 조절
        Object[] t = new Object[newSize]; // newSize 크기의 새로운 배열 t 생성
        for(int i = 1, j=front+1; i <size+1; i ++, j++){
                t[i] = q[j%q.length];          // 배열q의 항목들을 배열 t[1]로부터 복사
        }
        front = 0;
        rear  = size;
        q = (E[]) t;      // 배열 t를 배열 q로
}
```

```
        public void print() { // 큐의 항목들을 출력
                if (isEmpty())
                        System.out.print("큐가 비어있음.");
                else {
                        for (int i=0; i<q.length; i++)  System.out.print(q[i]+"\t ");
                        System.out.println();
                }
        }
}
```

```
public class main {
        public static void main(String[] args) {
                ArrayQueue<String> queue = new ArrayQueue<String>();

                queue.add("apple");             queue.add("orange");
                queue.add("cherry");    queue.add("pear");              queue.print();

                queue.remove();                 queue.print();

                queue.add("grape");             queue.print();

                queue.remove();                 queue.print();

                queue.add("lemon");             queue.print();
                queue.add("mango");         queue.print();
                queue.add("lime");              queue.print();
                queue.add("kiwi");              queue.print();

                queue.remove();                 queue.print();
        }
}
```

null	apple	orange	cherry	pear	null	null	null
null	null	orange	cherry	pear	null	null	null
null	null	orange	cherry	pear	grape	null	null
null	null	null	cherry	pear	grape	null	null
null	null	null	cherry	pear	grape	lemon	null
null	null	null	cherry	pear	grape	lemon	mango
lime	null	null	cherry	pear	grape	lemon	mango
lime	kiwi	null	cherry	pear	grape	lemon	mango
lime	kiwi	null	null	pear	grape	lemon	mango

front

rear

```
interface Queue{
        boolean isEmpty();
        void enQueue(char item);
        char deQueue();
        void delete();
        char peek();
}

class QNode{
        char data;
        QNode link;
}
```

```
class LinkedQueue implements Queue{
        QNode front;
        QNode rear;

        public LinkedQueue(){
                front = null;
                rear = null;
        }

        public boolean isEmpty(){
                return (front == null);
        }
```

```
public void enQueue(char item){
        QNode newNode = new QNode();
        newNode.data = item;
        newNode.link = null;
        if(isEmpty()){
                front = newNode;
                rear = newNode;
        }
        else {
                rear.link = newNode;
                rear = newNode;
        }
        System.out.println("Inserted Item : " + item);
}
```

```
public char deQueue(){
        if(isEmpty()) {
                System.out.println("Deleting fail! Linked Queue is empty!!");
                return 0;
        }
        else{
                char item = front.data;
                front = front.link;
                if(front == null)
                        rear = null;
                return item;
        }
}
```

```
	public void delete(){
		if(isEmpty()){
			System.out.println("Deleting fail! Linked Queue is empty!!");
		}
		else {
			front = front.link;
			if(front == null)
				rear = null;
		}
	}

	public char peek(){
		if(isEmpty()){
			System.out.println("Peeking fail! Linked Queue is empty!!");
			return 0;
		}
		else
			return front.data;
	}
```

```
        public void printQueue(){
                if(isEmpty())
                        System.out.printf("Linked Queue is empty!! %n %n");
                else{
                        QNode temp = front;
                        System.out.printf("Linked Queue>> ");
                        while(temp != null){
                                System.out.printf("%c ", temp.data);
                                temp = temp.link;
                        }
                        System.out.println();System.out.println();
                }
        }
}
```

```
class LinkedQ_Test{
        public static void main(String args[]){
                char deletedItem;
                LinkedQueue LQ = new LinkedQueue();

                LQ.enQueue('A');
                LQ.printQueue();

                LQ.enQueue('B');
                LQ.printQueue();

                deletedItem = LQ.deQueue();
                if(deletedItem != 0)
                        System.out.println("deleted Item : " + deletedItem);
                LQ.printQueue();

                LQ.enQueue('C');
                LQ.printQueue();
```

```
                deletedItem = LQ.deQueue();
                if(deletedItem != 0)
                        System.out.println("deleted Item : " + deletedItem);
                LQ.printQueue();

                deletedItem = LQ.deQueue();
                if(deletedItem != 0)
                        System.out.println("deleted Item : " + deletedItem);
                LQ.printQueue();

                deletedItem = LQ.deQueue();
                if(deletedItem != 0)
                        System.out.println("deleted Item : " + deletedItem);
                LQ.printQueue();
        }
}
```

```
public class Node <E> {
        private E       item;
        private Node<E> next;

        public Node(E newItem, Node<E> node){  // 생성자
                item = newItem;
                next = node;
        }
        // get 메소드들과 set 메소드 들
        public E       getItem()                { return item; }
        public Node<E> getNext()                { return next; }
        public void    setItem(E newItem)       { item = newItem; }
        public void    setNext(Node<E> newNext){ next = newNext; }
}
```

```
import java.util.NoSuchElementException;
public class ListQueue <E> {
        private Node<E> front, rear;
        private int size;

        public ListQueue() {   // 생성자
                front = rear = null;
                size  = 0;
        }

        public int     size()    { return size; }        // 큐의 항목의 수를 리턴
        public boolean isEmpty() { return size() == 0; } // 큐가 empty이면 true 리턴

        public void add(E newItem){
                Node newNode = new Node(newItem, null);      // 새 노드 생성
                if (isEmpty()) front = newNode; // 큐가 empty이었으면  front도 newNode를 가리키게 한다
                else rear.setNext(newNode);     // 그렇지않으면 rear의 next를 newNode를 가리키게 한다
                rear = newNode;                 // 마지막으로 rear가 newNode를 가리키게 한다
                size++;                         // 큐 항목 수 1 증가
        }
```

```
public E remove() {
        if (isEmpty()) throw new NoSuchElementException(); // underflow 경우에 프로그램 정지
        E frontItem = front.getItem();  // front가 가리키는 노드의 항목을 frontItem에 저장
        front = front.getNext();                // front가 front 다음 노드를 가리키게 한다.
        if (isEmpty()) rear = null;             // 큐가 empty이면 rear = null
        size--;                                                 // 큐 항목 수 1 감소
        return frontItem;
}

public void print() { // 큐의 항목들을 front부터 차례로 출력
        if (isEmpty()) System.out.print("큐가 empty임");
        else
                for (Node p = front; p != null; p = p.getNext())
                        System.out.print(p.getItem()+"\t ");
        System.out.println();
}
```

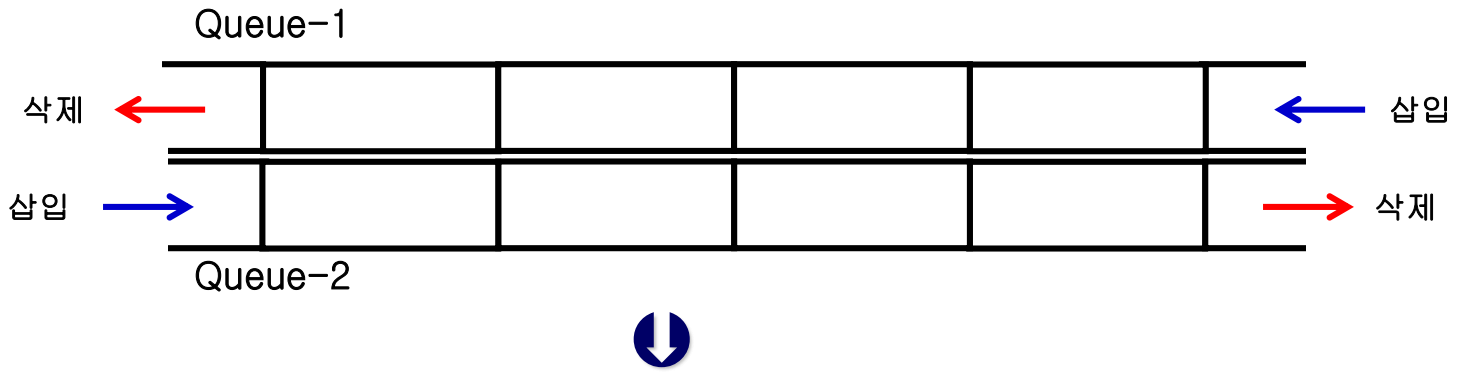
```
        public static void main(String[] args) {
                ListQueue<String> q = new ListQueue<String>();

                        q.add("apple");         q.add("orange");
                        q.add("cherry");        q.add("pear");
                        q.print();

                        q.remove(); q.print();
                        q.remove(); q.print();

                        q.add("grape");         q.print();
        }
}
```

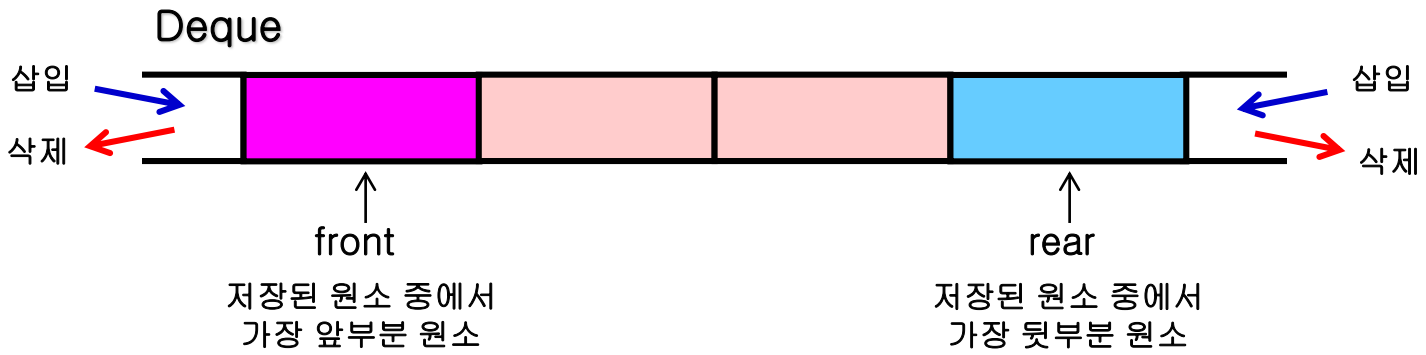
큐 자료구조의 응용

- CPU의 태스크 스케줄링(Task Scheduling)
- 네트워크 프린터
- 실시간(Real-time) 시스템의 인터럽트(Interrupt) 처리
- 다양한 이벤트 구동 방식(Event-driven) 컴퓨터 시뮬레이션
- 콜 센터의 전화 서비스 처리
- 이진트리의 레벨순서 순회(Level-order Traversal)
- 그래프에서 너비우선탐색(Breath-First Search)
- Etc

❖ 덱 혹은 데크(Deque, double-ended queue)

- 양쪽 끝에서 삽입과 삭제를 허용하는 자료구조
- 데크는 스택과 큐 자료구조를 혼합한 자료구조
- 따라서 데크는 스택과 큐를 동시에 구현하는데 사용





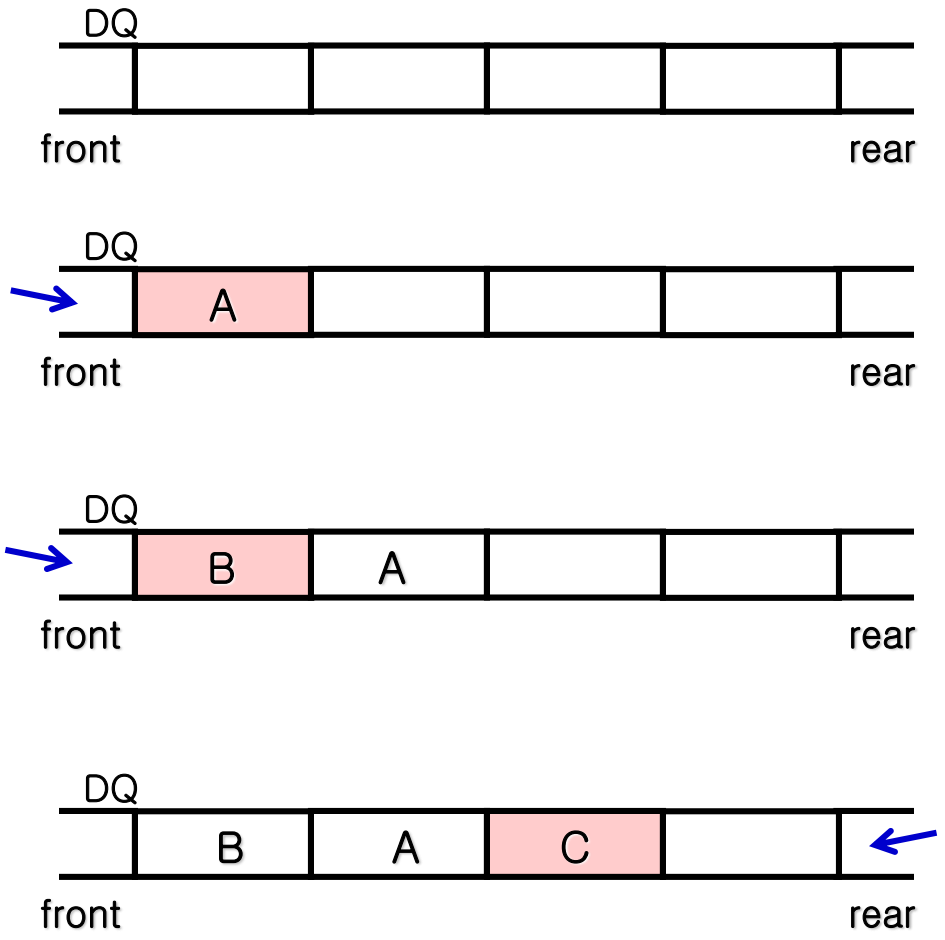
덱에서의 연산 과정

① createDeque();

② insertFront(DQ, 'A');

③ insertFront(DQ, 'B');

④ insertRear(DQ, 'C');



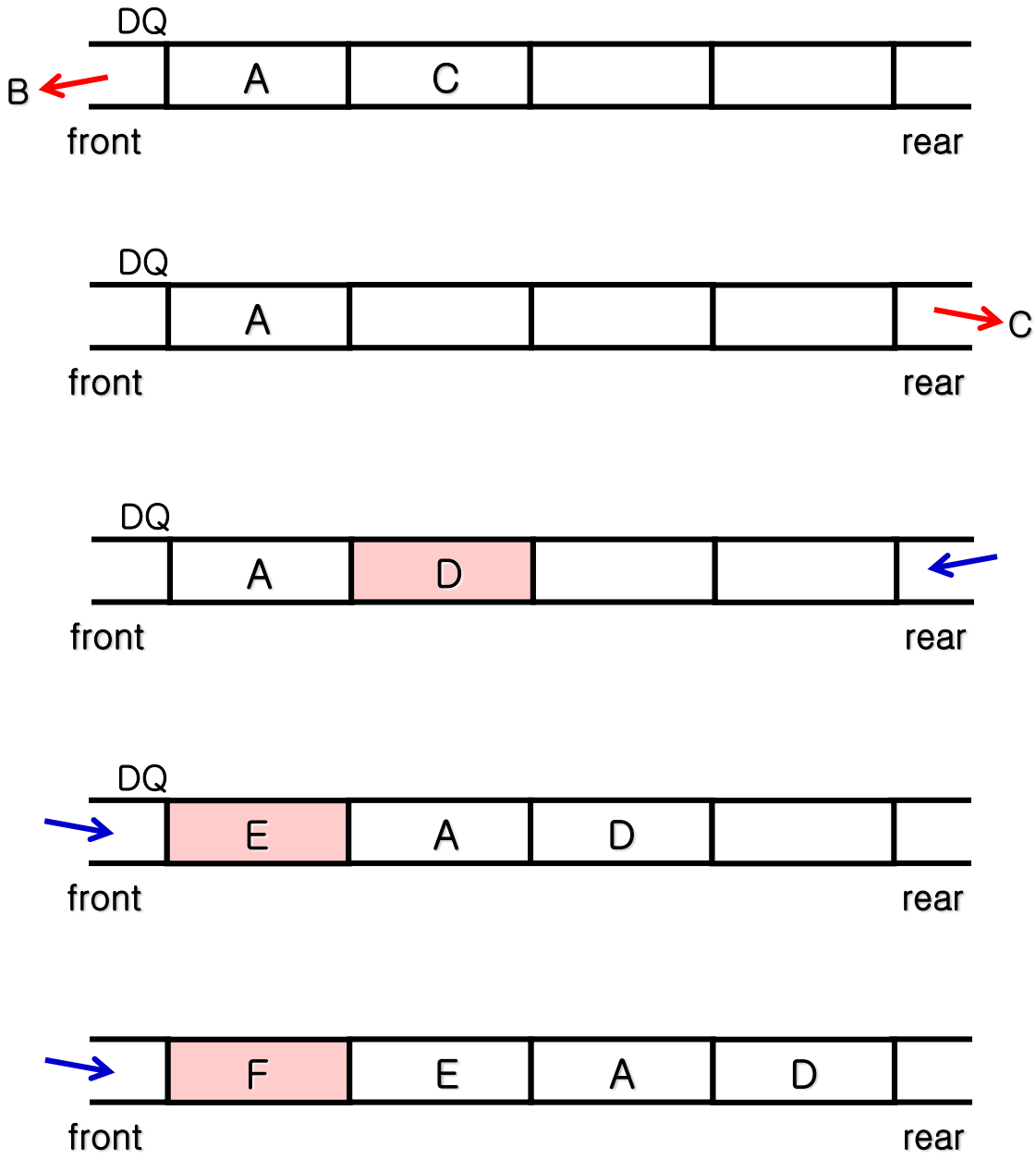
⑤ deleteFront(DQ);

⑥ deleteRear(DQ);

⑦ insertRear(DQ, 'D');

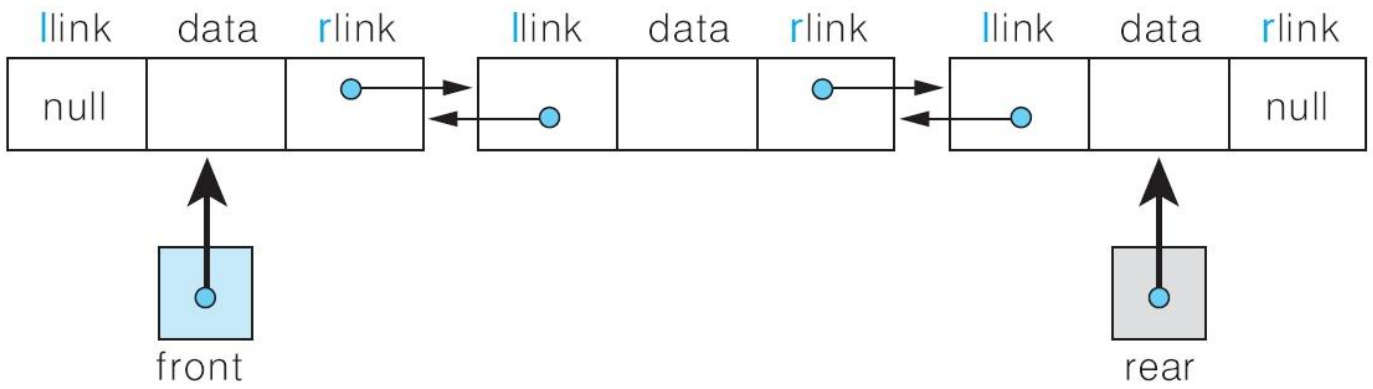
⑧ insertFront(DQ, 'E');

⑨ insertFront(DQ, 'F');



덱의 구현

- 양쪽 끝에서 삽입/삭제 연산을 수행하면서 크기 변화와 저장된 원소의 순서 변화가 많으므로 순차 자료구조는 비효율적
- 양방향으로 연산이 가능한 이중 연결 리스트를 사용



```
class DQNode{
        char data;
        DQNode rlink;
        DQNode llink;
}

class DQueue{
        DQNode front;
        DQNode rear;

        public DQueue(){
                front = null;
                rear = null;
        }

        public boolean isEmpty(){
                return (front == null);
        }
```

```
public void insertFront(char item){
        DQNode newNode = new DQNode();
        newNode.data = item;
        if(isEmpty()){
                front = newNode;
                rear = newNode;
                newNode.rlink = null;
                newNode.llink = null;
        }
        else {
                front.llink = newNode;
                newNode.rlink = front;
                newNode.llink = null;
                front = newNode;
        }
        System.out.println("Front Inserted Item : " + item);
}
```

```
public void insertRear(char item){
        DQNode newNode = new DQNode();
        newNode.data = item;
        if(isEmpty()){
                front = newNode;
                rear = newNode;
                newNode.rlink = null;
                newNode.llink = null;
        }
        else {
                rear.rlink = newNode;
                newNode.rlink = null;
                newNode.llink = rear;
                rear = newNode;
        }
        System.out.println("Rear Inserted Item : " + item);
}
```

```
public char deleteFront(){
        if(isEmpty()) {
                System.out.println("Front Deleting fail! DQueue is empty!!");
                return 0;
        }
        else{
                char item = front.data;
                if(front.rlink == null){
                        front = null;
                        rear = null;
                }
                else {
                        front = front.rlink;
                        front.llink = null;
                }
                return item;
        }
}
```

```
public char deleteRear(){
        if(isEmpty()) {
                System.out.println("Rear Deleting fail! DQueue is empty!!");
                return 0;
        }
        else{
                char item = rear.data;
                if(rear.llink == null){
                        rear = null;
                        front = null;
                }
                else {
                        rear = rear.llink;
                        rear.rlink = null;
                }
                return item;
        }
}
```

```
public void removeFront(){
        if(isEmpty()) {
                System.out.println("Front Removing fail! DQueue is empty!!");
        }
        else{
                if(front.rlink == null){
                        front = null;
                        rear = null;
                }
                else {
                        front = front.rlink;
                        front.llink = null;
                }
        }
}
```

```
public void removeRear(){
        if(isEmpty()) {
                System.out.println("Rear Removing fail! DQueue is empty!!");
        }
        else{
                if(rear.llink == null){
                        rear = null;
                        front = null;
                }
                else {
                        rear = rear.llink;
                        rear.rlink = null;
                }
        }
}
```

```
public char peekFront(){
        if(isEmpty()){
                System.out.println("Front Peeking fail! DQueue is empty!!");
                return 0;
        }
        else
                return front.data;
}

public char peekRear(){
        if(isEmpty()){
                System.out.println("Rear Peeking fail! DQueue is empty!!");
                return 0;
        }
        else
                return rear.data;
}
```

```
        public void printDQueue(){
                if(isEmpty())
                        System.out.printf("DQueue is empty!! %n %n");
                else{
                        DQNode temp = front;
                        System.out.printf("DQueue>> ");
                        while(temp != null){
                                System.out.printf("%c ", temp.data);
                                temp = temp.rlink;
                        }
                        System.out.println();   System.out.println();
                }
        }
}
```

```
class DQ_Test{
        public static void main(String args[]){
                char deletedItem;
                DQueue DQ = new DQueue();

                DQ.insertFront('A');
                DQ.printDQueue();

                DQ.insertFront('B');
                DQ.printDQueue();

                DQ.insertRear('C');
                DQ.printDQueue();
```

```
                deletedItem = DQ.deleteFront();
                if(deletedItem != 0)
                        System.out.println("Front deleted Item : " + deletedItem);
                DQ.printDQueue();

                deletedItem = DQ.deleteRear();
                if(deletedItem != 0)
                        System.out.println("Rear deleted Item : " + deletedItem);
                DQ.printDQueue();

                DQ.insertRear('D');
                DQ.printDQueue();

                DQ.insertFront('E');
                DQ.printDQueue();

                DQ.insertFront('F');
                DQ.printDQueue();
        }
}
```

데크 자료구조의 응용

- 스크롤(Scroll)
- 문서 편집기 등의 undo 연산
- 웹 브라우저의 방문 기록 등
  - 웹 브라우저 방문 기록의 경우, 최근 방문한 웹 페이지 주소는 앞에 삽입하고, 일정 수의 새 주소들이 앞쪽에서 삽입되면 뒤에서 삭제가 수행

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

public class structureClass {
        public static void main(String[] args) {
                Queue <String> que = new LinkedList <String>();
                Stack <String> stack = new Stack <String>();
                String words[] = {"장발장","jsieun73","들판위에하","level","안녕하세요"};
                char queS = 0;

                System.out.println("큐 & 스택 예제 입니다. ");

                for(int i = 0; i< 5; i++)
                {
                        for(int j= 0 ; j<words[i].length(); j++)
                        {
                                queS = words[i].charAt(j);
                                que.offer(""+queS);
                                stack.push(""+queS);
                        }
```

```
                        for(int k=0; k<words[i].length(); k++)
                        {
                                if(que.remove().equals(stack.pop()) && k == words[i].length()-1)
                                {
                                        System.out.println(words[i]+"\t : 회문");
                                }
                                else if(k== words[i].length()-1)
                                {
                                        System.out.println(words[i]+"\t : 회문아님");
                                }
                        }
                }
        }
}
```

Report

이번 회차에서 살펴본 소스를 테스트하고 업로드하세요!

Reference

- https://geoseong.tistory.com/23
- https://jsieun73.tistory.com/26
- 자바로 배우는 쉬운 자료구조, 이지영, 한빛아카데미
- 자바와 함께하는 자료구조의 이해, 양성봉, 생능출판

언제 어디서나 즐공, 열공, 진공하세요.

감사합니다

DIT 동의과학대학교
DONG-EUI INSTITUTE OF TECHNOLOGY

DIT 동의과학대학교
DONG-EUI INSTITUTE OF TECHNOLOGY