Scream!!(절규)

Professor(scream time)

Nirvana(해탈)

Data Structure

Spring 2019

MW 11:00-12:15, SN 014

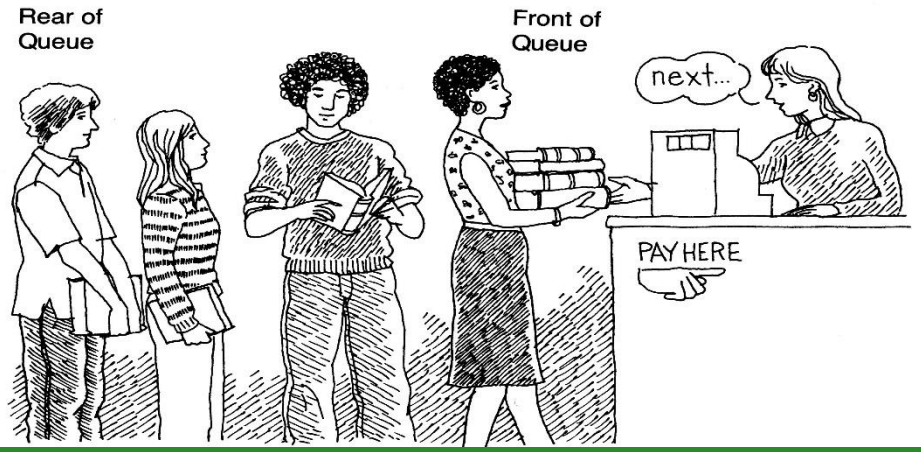
http://smart.hallym.ac.kr

Instructor: Jin Kim

010-6267-8189(033-248-2318)

jinkim@hallym.ac.kr

Office Hours: M 12:15-1, Tu 3:30-5pm M 2-3pm, W 2-3:30



Queue

스택 큐

[그림 7-1] 스택과 큐의 구조 예

Chapter Contents

Queue Example

remove

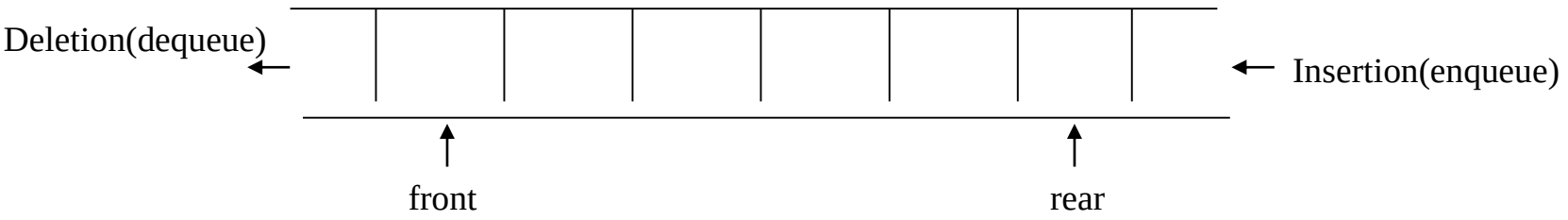
front

back

add

Definition of a Queue(큐의 정의)

- A queue is a data structure that models/enforces the **first-come first-serve** order, or equivalently the **first-in first-out** (FIFO) order. 큐는 선입선출(먼저삽입된 원소가 먼저 제거된다)의 자료구조
- That is, the element that is inserted first into the queue will be the element that will deleted first, and the element that is inserted last is deleted last.(첫번째삽입된원소는 첫번째제거되고,가장나중삽입된 원소는 가장 나중 제거된다.
- A waiting line is a good real-life example of a queue. (In fact, the Britich word for “line” is “queue”.)줄서기는 큐의 좋은예

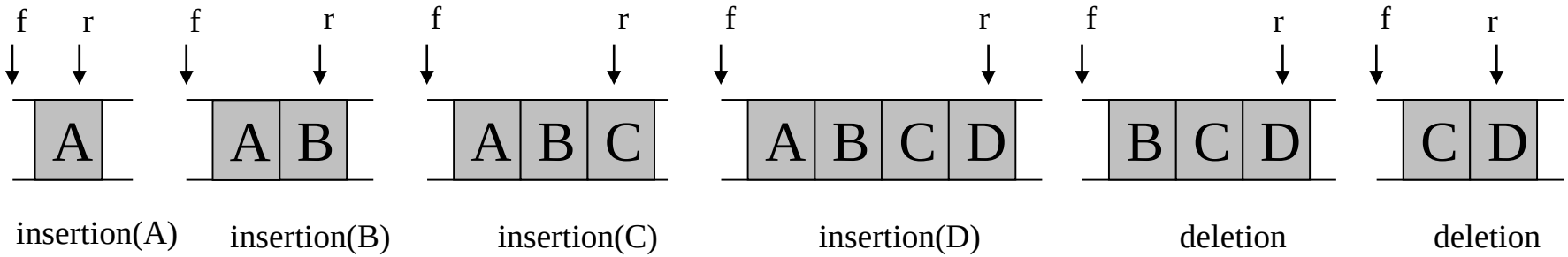


Queues

- **Queue**: a collection whose elements are added at one end (the ***rear*** or ***tail*** of the queue) and removed from the other end (the ***front*** or ***head*** of the queue) 한쪽끝 ***rear***끝에서 삽입, 다른쪽끝 ***front*** 에서 제거된다
- A queue is a ***FIFO*** (first in, first out) data structure 큐는 선입선출 자료구조

Queue(2)

- Insertion and deletion in queue(큐에서 삽입삭제)
  - ✓f : front, r : rear



- Example of queue(큐의 예)
  - ✓Operating system(OS)운영체제: Jobs in a job queue. Jobs waiting for what?(cpu time) 작업큐에서의 작업들이 cpu시간 배정을 줄서서 기다린다.

Queue ADT

- ADT Queue 큐의 추상 데이터 타입

data : finite ordered list with 0 or more elements

operations :

queue ∈ Queue; item ∈ Element;
createQ() ::= create an empty queue;
enqueue(queue, item) ::= insert item at the rear of queue;
isEmpty(queue) ::= **if** (queue is empty) **then return** true
else return false;
dequeue(queue) ::= **if** (isEmpty(queue)) **then return** error
else {delete and return the front item of queue};
delete(queue) ::= **if** (isEmpty(queue)) **then return** error
else {delete the front item of queue};
peek(queue) ::= **if** (isEmpty(queue)) **then return** error
else {return the front item of queue};

End Queue

Array implementation of queues배열로 큐의 구현

- 1-d array(1차원배열)
  - ✓ Simplest way to implement queue(큐를 구현하는간단한 방법)
  - ✓ Array Q[n](큐크기 n인 배열
  - ✓ Variable to use an array to implement a queue(큐를 구현하기 위한 변수들)
    - n : maximum number of element in a queue(큐의 최대원소)
    - Two index variables front, rear (두 인덱스 변수 front, rear)
      - initialize : front = rear = -1 (empty queue) (초기 큐)
      - Enpty queue : front = rear (큐가 비어있을때)
      - Queue full : rear = n-1(큐가 모두 채워저있을때)

Array implementation of queues배열 큐

- Operators

```
createQ()
      // 공백 큐(q[])를 생성
    q[n];
    front ← -1;      // 초기화
    rear ← -1;
end createQ()

isEmpty(q)
       // 큐(q)가 공백인지를 검사
    if (front = rear) then return true
    else return false;
end isEmpty()

enqueue(q, item)  //final exam, 기말고사문제
        // 큐(q)에 원소를 삽입
    if (rear=n-1) then queueFull()   // 큐(q)가 만원인 상태를 처리
    rear ← rear + 1;
    q[rear] ← item;
end enqueue()
```

Array implementation of queue

- Operators(2)

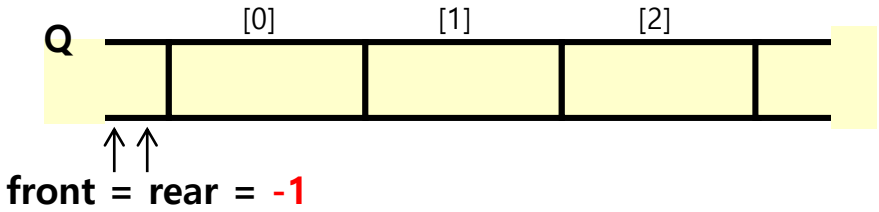
```
dequeue(q)
        // 큐(q)에서 원소를 삭제하여 반환
    if (isEmpty(q)) then queueEmpty()  // 큐(q)가 공백인 상태를 처리
    else {
       front ← front + 1;
       return q[front];
    };
end dequeue()

delete(q)
        // 큐(q)에서 원소를 삭제
    if (isEmpty(q)) then queueEmpty()  // 큐(q)가 공백인 상태를 처리
    front ← front + 1;
end delete()

peek(q)
      // 큐(q)에서 원소를 검색
   if (isEmpty(q)) then queueEmpty()   // 큐(q)가 공백인 상태를 처리
    else return q[front+1];
end peek()
```

Array implementation of queue배열 큐

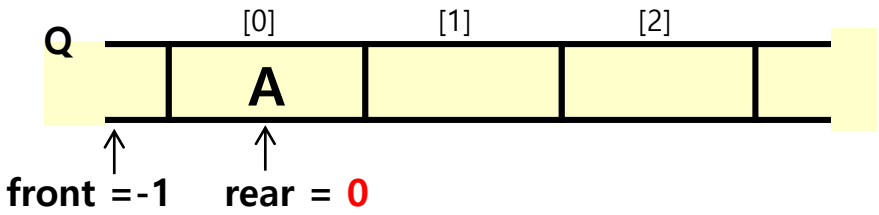
- Problem of array-based queue(배열큐의 문제)
  - ✓ Rear=n-1
    - Even though queue is not full, can not add element (큐에 빈공간이 있어도 원소를 추가할 수 없다)
    - -> relocation (additional time required) 원소의 재배치,추가시간소요
    - Really queue full 실제 큐가 만원
      - Enlarge the size of the array배열의 크기를 확대
    - 따라서 절대로 평범한 1차원배열로 큐를 구현하지 않음

Array based queue 배열 큐

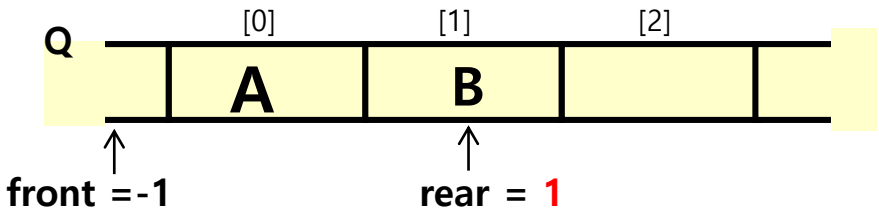
```
CreateQ();
```



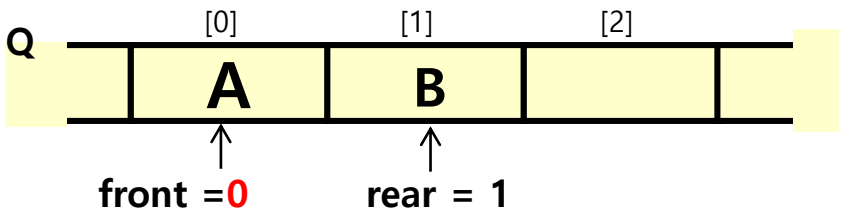
의자가 세 개 있다 가정하자.



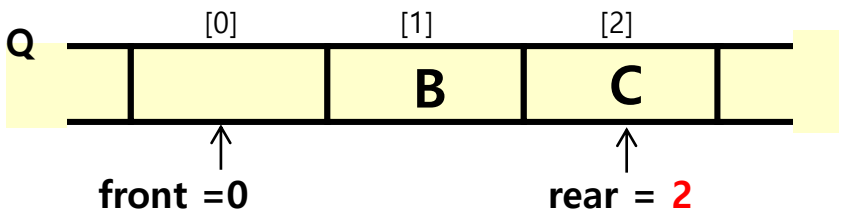

```
CreateQ();
Enqueue("A");
```



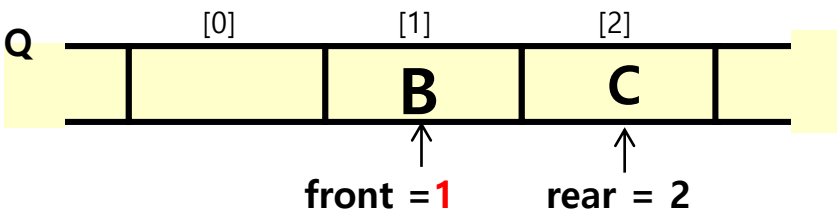

```
CreateQ();
Enqueue("A");
Enqueue("B");
```



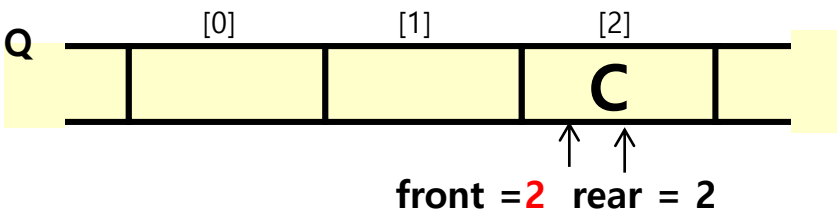

```
CreateQ();
Enqueue("A");
Enqueue("B");
dequeue();
```




```
CreateQ();
Enqueue("A");
Enqueue("B");
dequeue();
Enqueue("C");
```



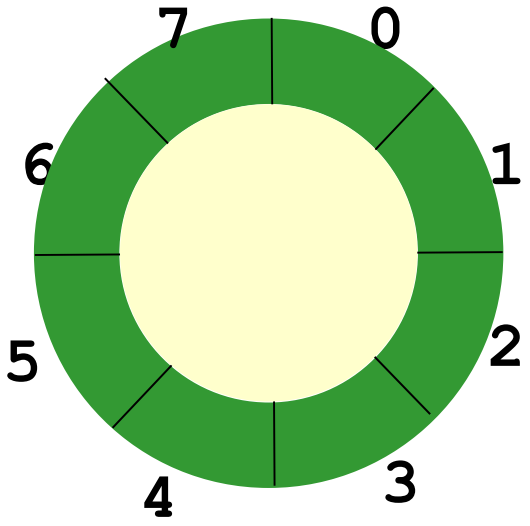

```
CreateQ();
Enqueue("A");
Enqueue("B");
dequeue();
Enqueue("C");
Dequeue();
```




```
CreateQ();
Enqueue("A");
Enqueue("B");
dequeue();
Enqueue("C");
Dequeue();
Dequeue();
```

의자가 세 개 있음에도 enqueue할 수 없음. 따라서 재배치

- **Neat trick**: use a ***circular array*** to insert and remove items from a queue in constant time 트릭: 큐에서 삽입과 삭제를 원형배열을 이용한다.
- The idea of a circular array is that the end of the array "wraps around" to the start of the array 배열의 시작과 끝을 연결한다.



Numerics for Circular Queues원형큐를 위한 식

- **front** increases by (1 modulo capacity) after each dequeue( ): 나머지 함수를 사용한다.

  front = (**front** +1) % capacity;

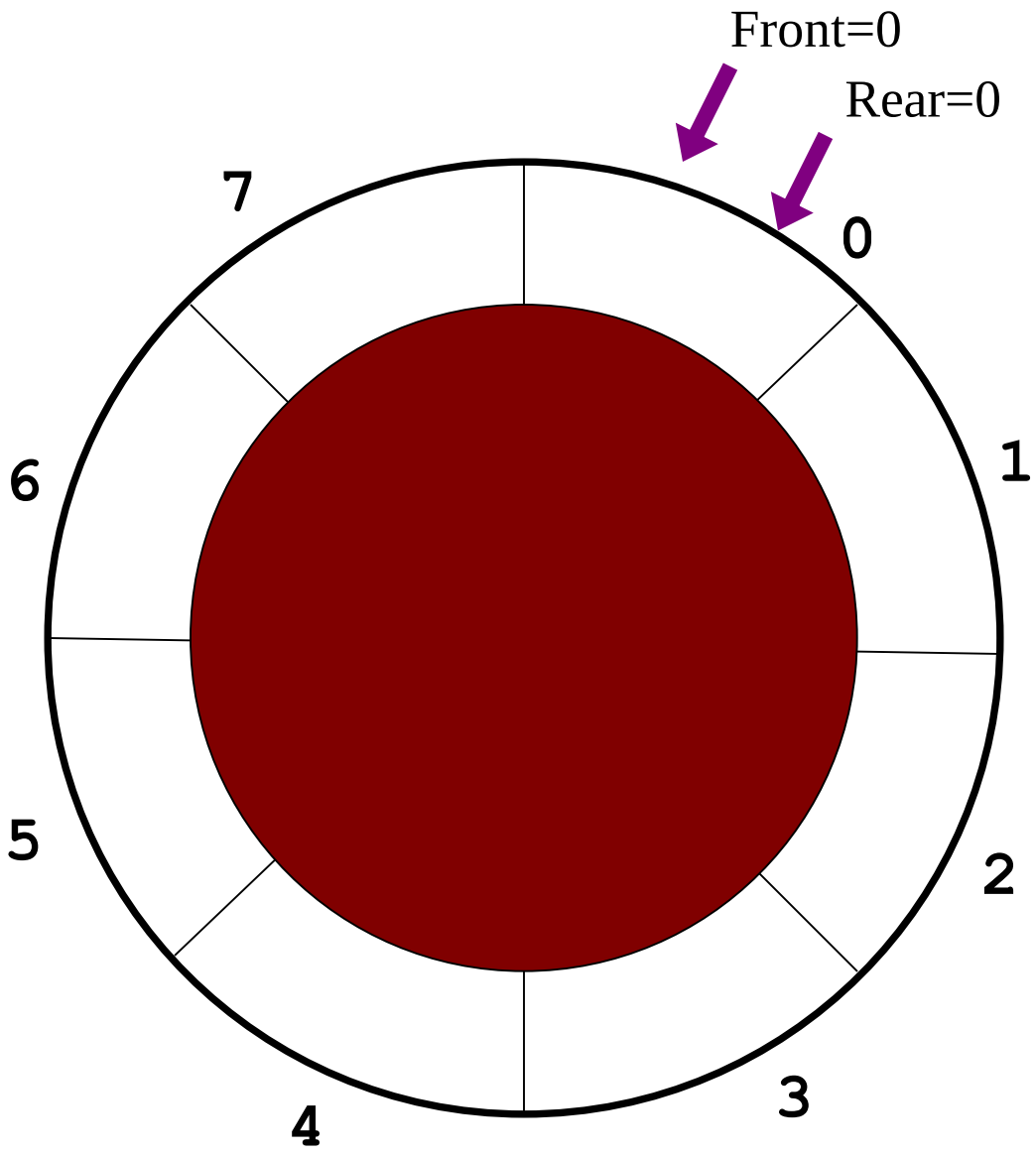
- **rear** increases by (1 modulo capacity) after each enqueue( ):나머지 함수를 사용한다.

  rear = (**rear** +1) % capacity;

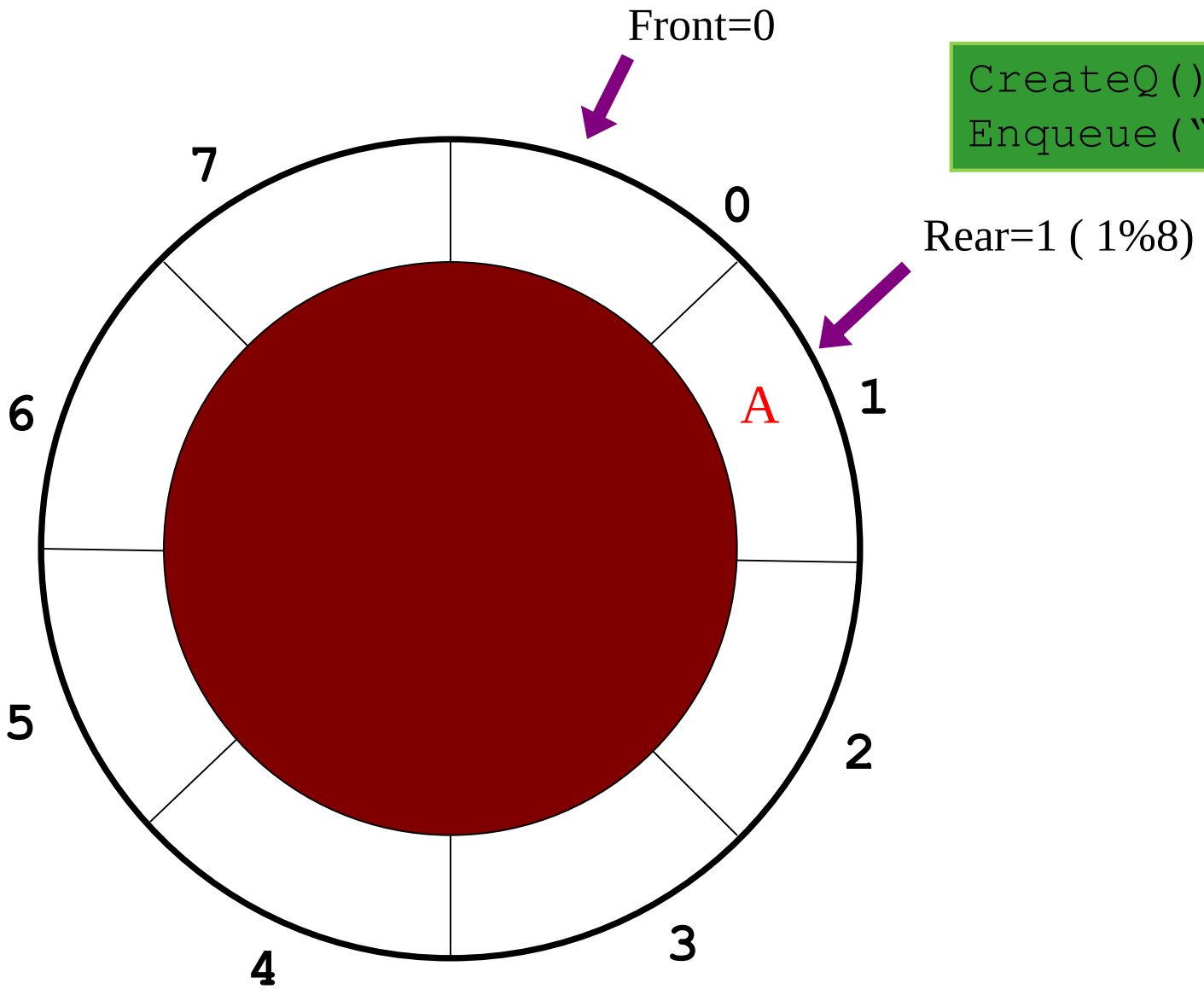
Array based circular queue(6)

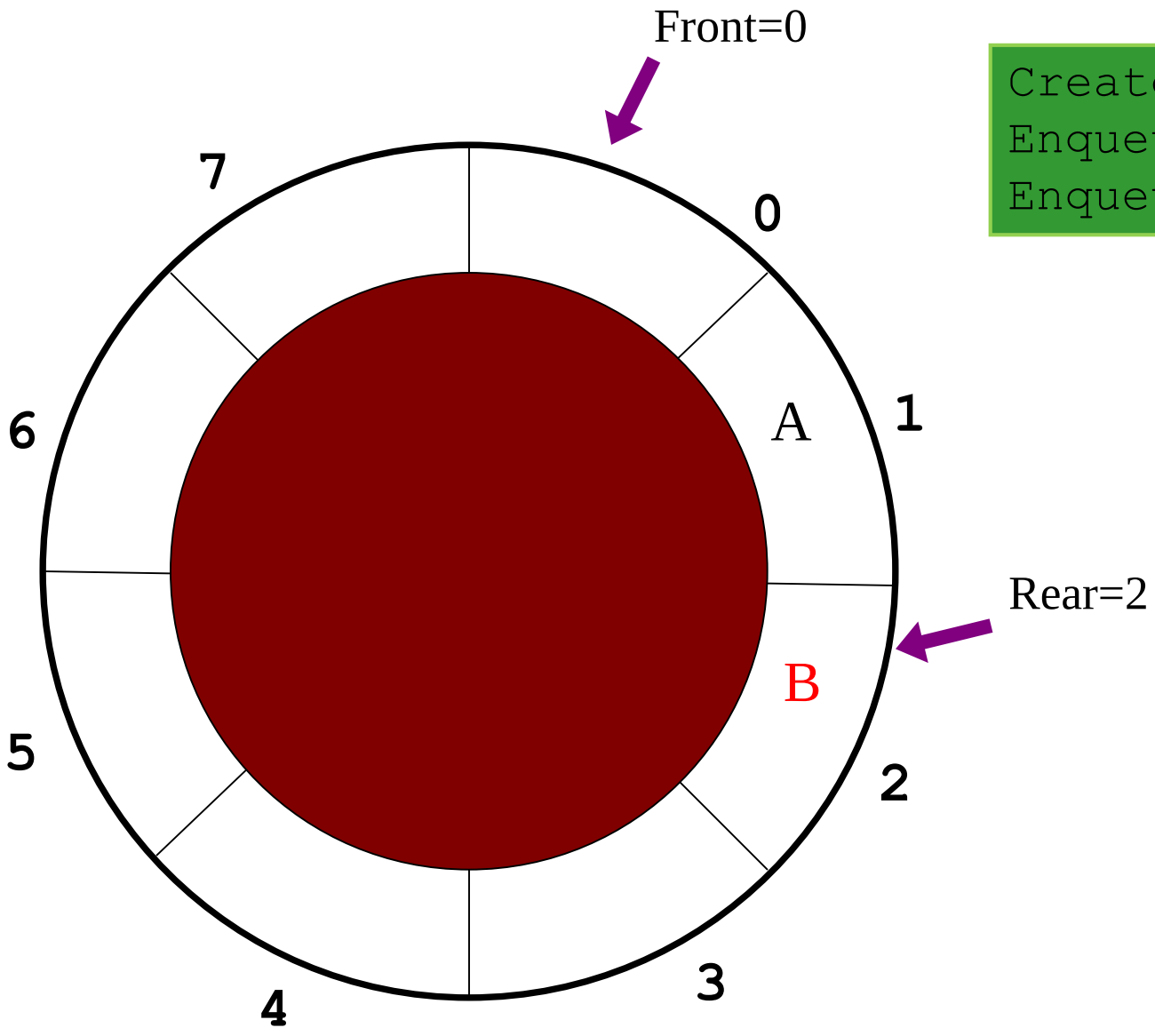
- Enqueue, dequeue operations in circular queue

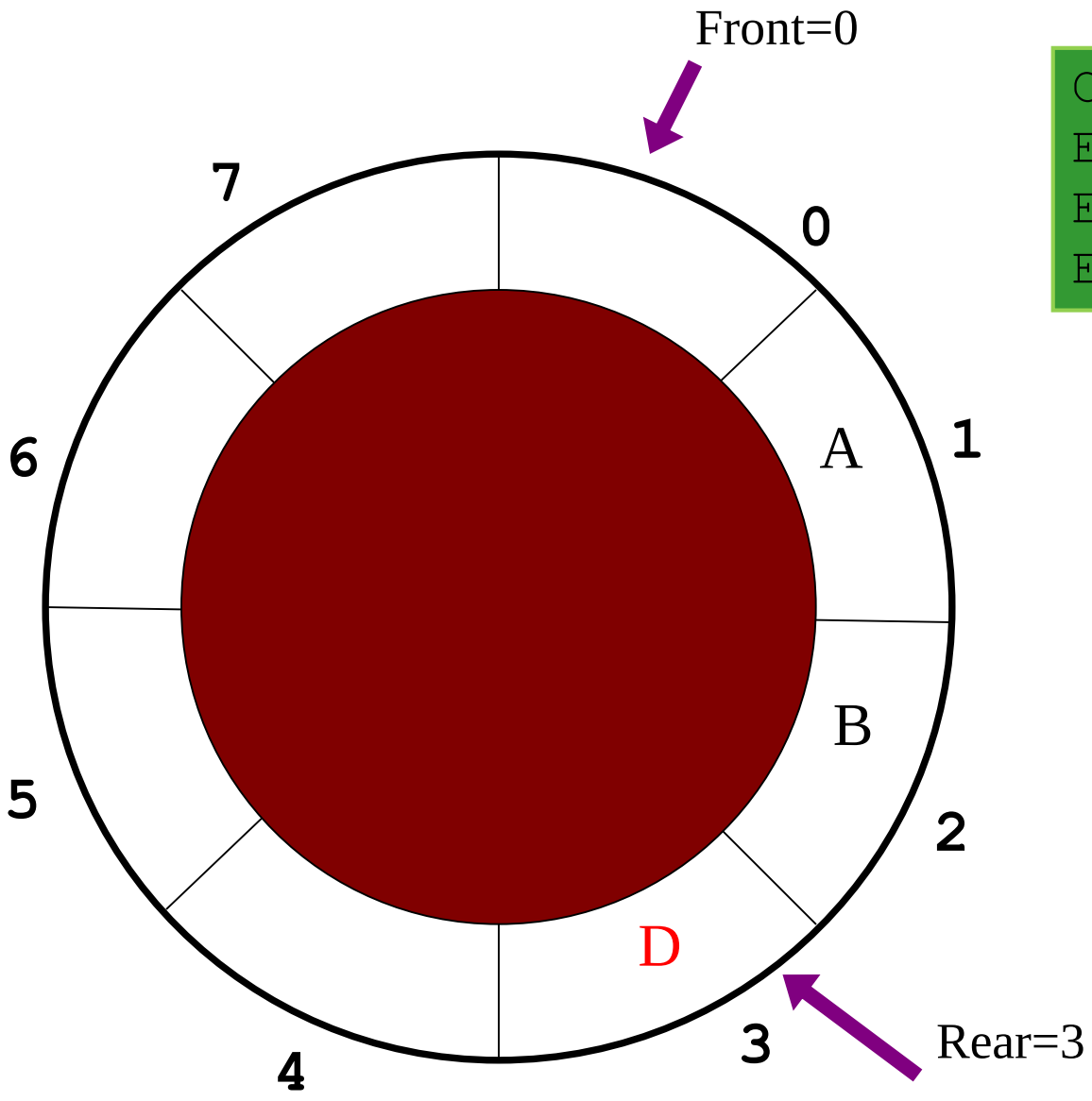
```
enqueue(q, item)
        // 원형 큐(q)에 item을 삽입
    rear ← (rear+1) mod n;              // 원형 큐(q) 인덱스
    if (front = rear) then queueFull();  // 큐(q)가
                                          만원인 상태를 처리
    q[rear] ← item;
end enqueue()

dequeue(q)
        // 원형 큐(q)에서 원소를 삭제하여 반환
    if (front = rear) then queueEmpty()    // 큐(q)가
                                            공백인 상태를 처리
    else {front ← (front+1) mod n;        // 원형 인덱스
     return q[front]
    };
end dequeue()
```




```
CreateQ();
```

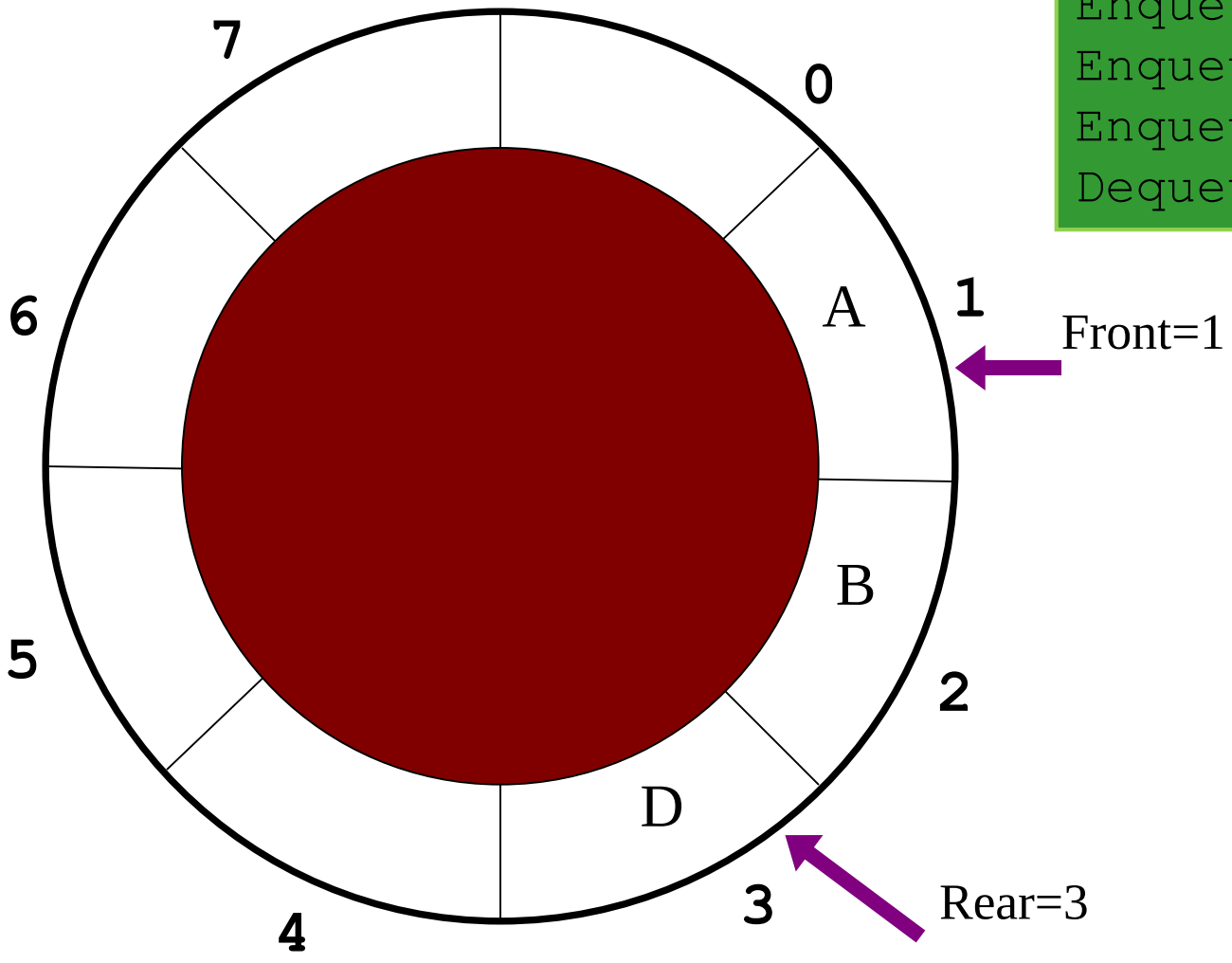

Front=0
CreateQ();
Enqueue("A")
Rear=1 (1%8)
0
1
2
3
4
5
6
7
A


Front=0
CreateQ();
Enqueue("A")
Enqueue("B")
0
1
2
3
4
5
6
7
A
B
Rear=2

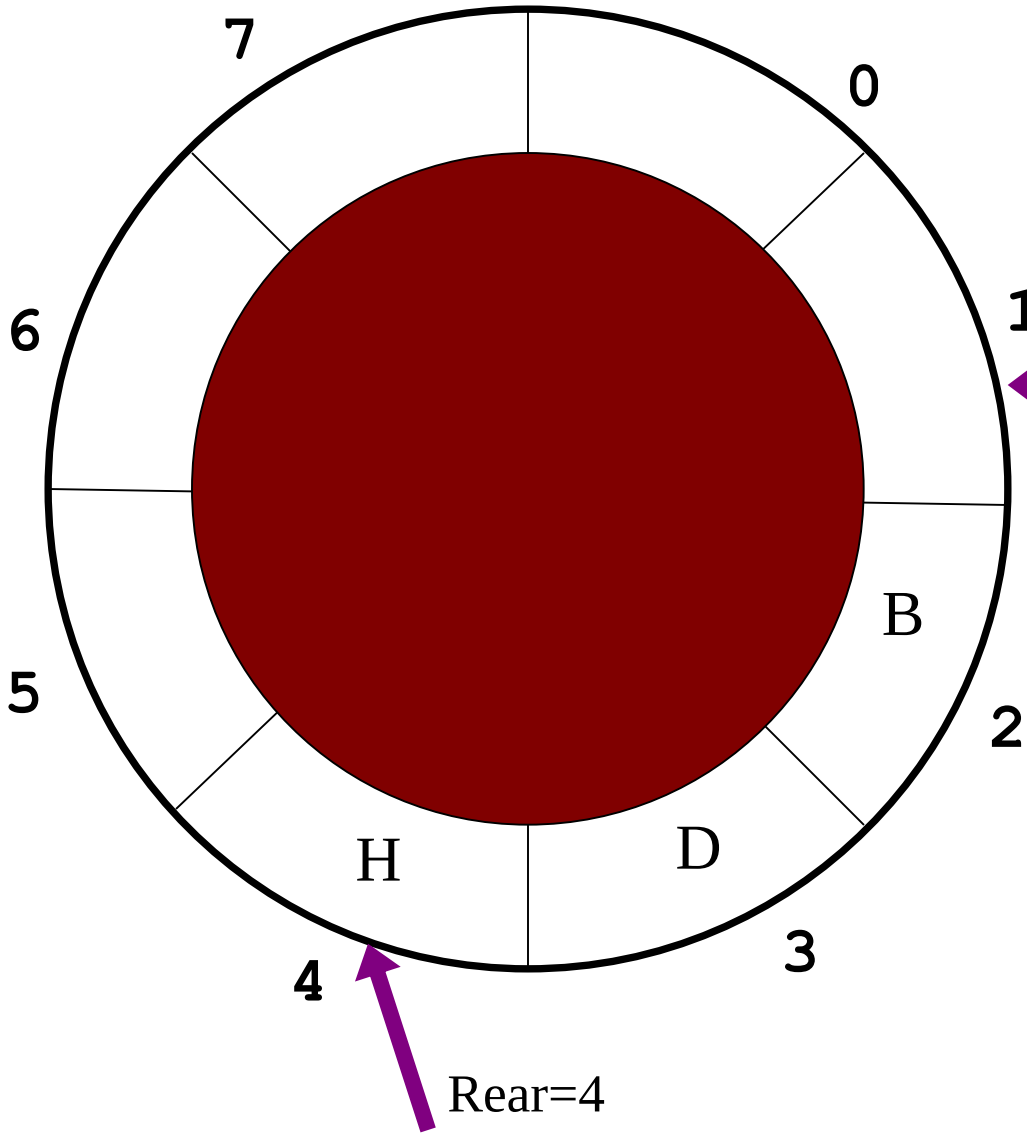



```
CreateQ();
Enqueue("A")
Enqueue("B")
Enqueue("D")
```

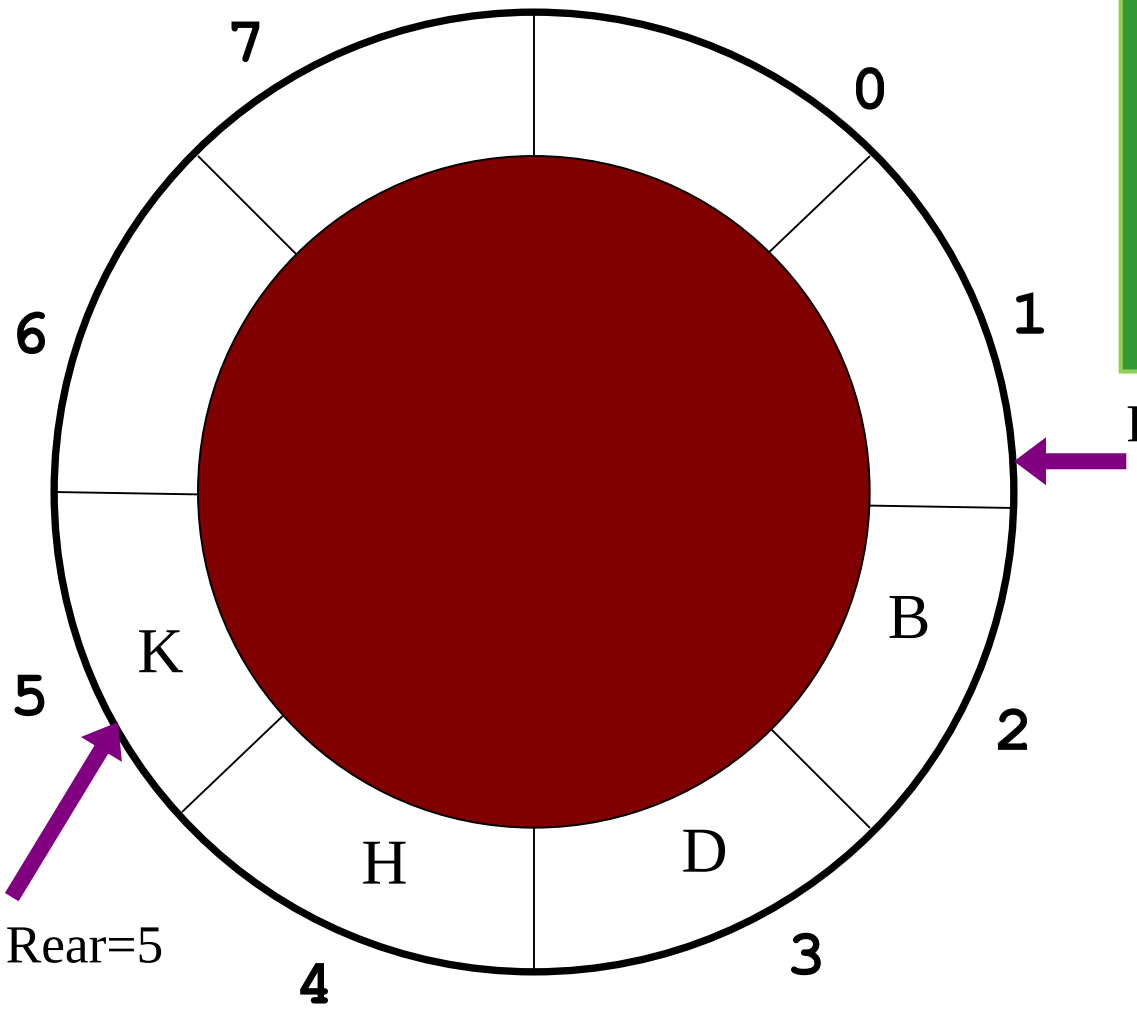
```
CreateQ();
Enqueue("A")
Enqueue("B")
Enqueue("D")
Dequeue()
```



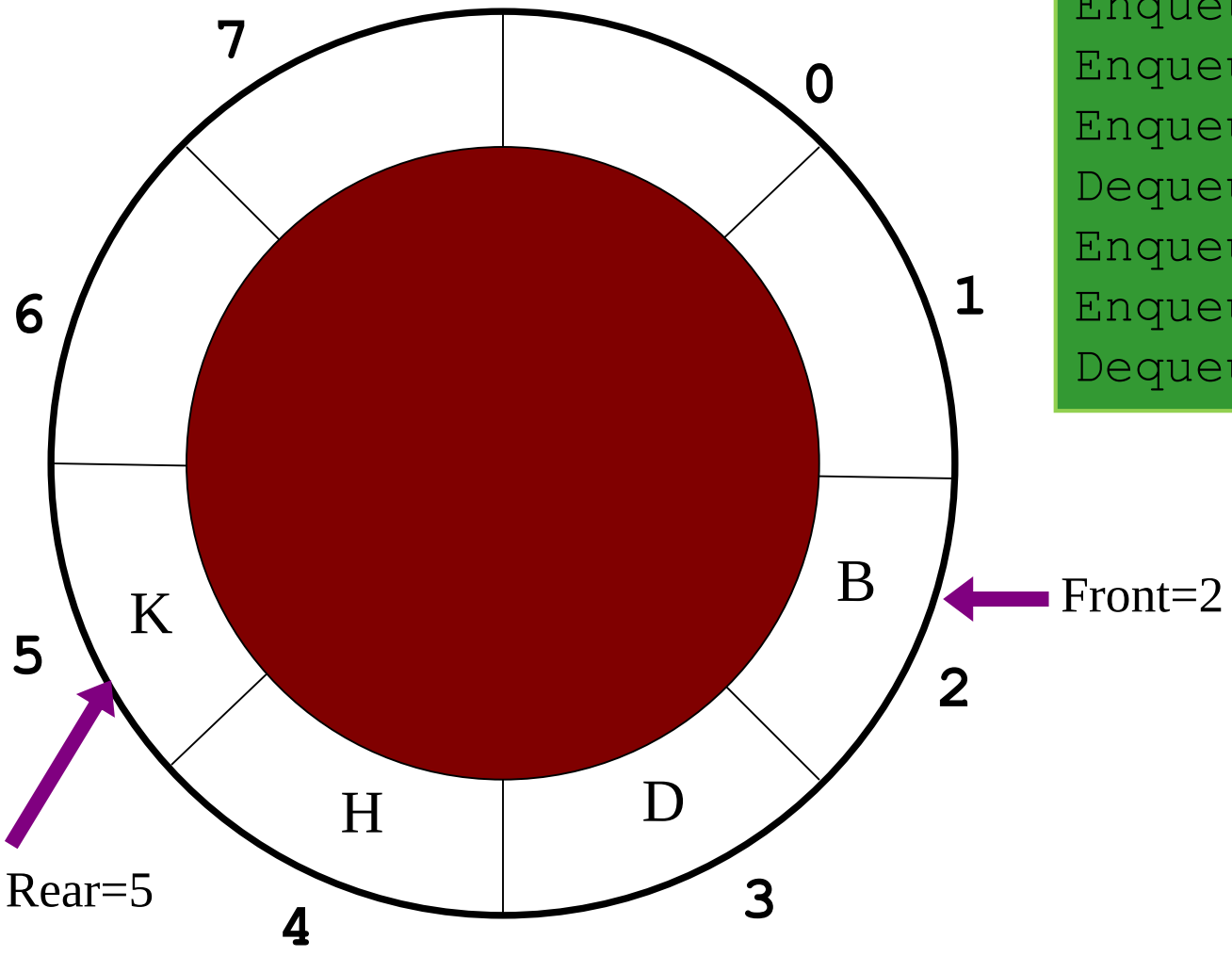
```
CreateQ();
Enqueue("A")
Enqueue("B")
Enqueue("D")
Dequeue()
Enqueue("H")
```

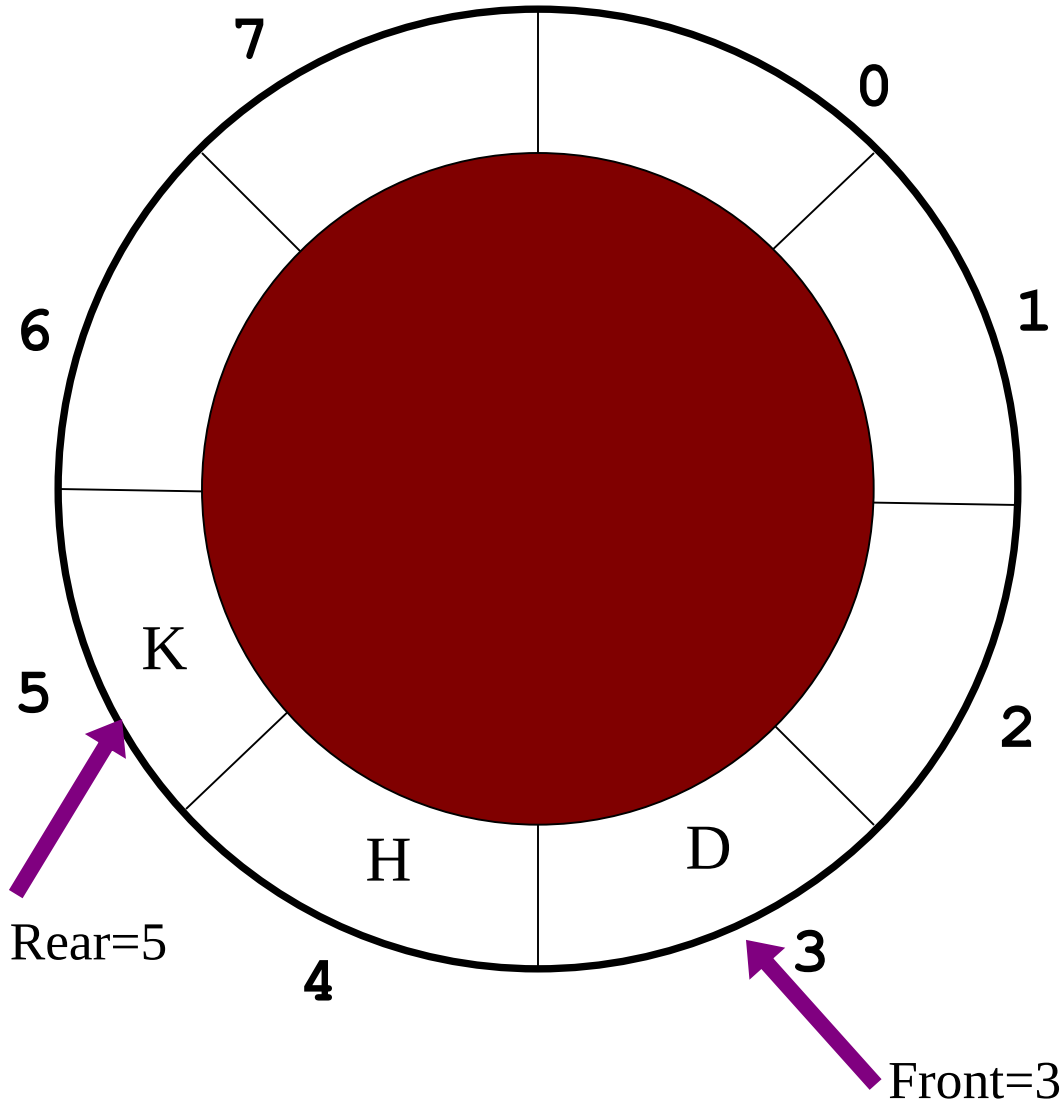


```
CreateQ();
Enqueue("A")
Enqueue("B")
Enqueue("D")
Dequeue()
Enqueue("H")
Enqueue("K")
```

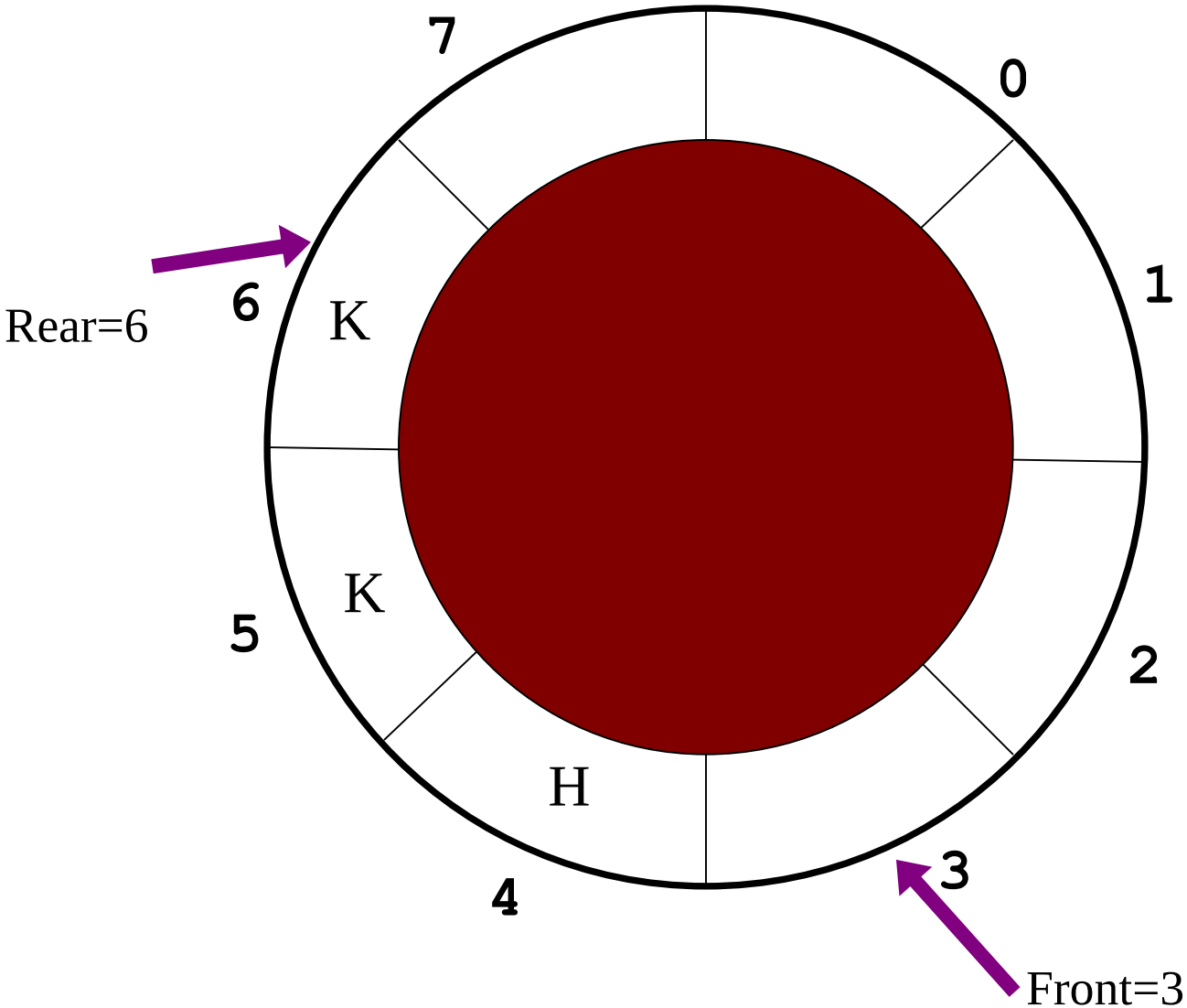


```
CreateQ();
Enqueue("A")
Enqueue("B")
Enqueue("D")
Dequeue()
Enqueue("H")
Enqueue("K")
Dequeue()
```

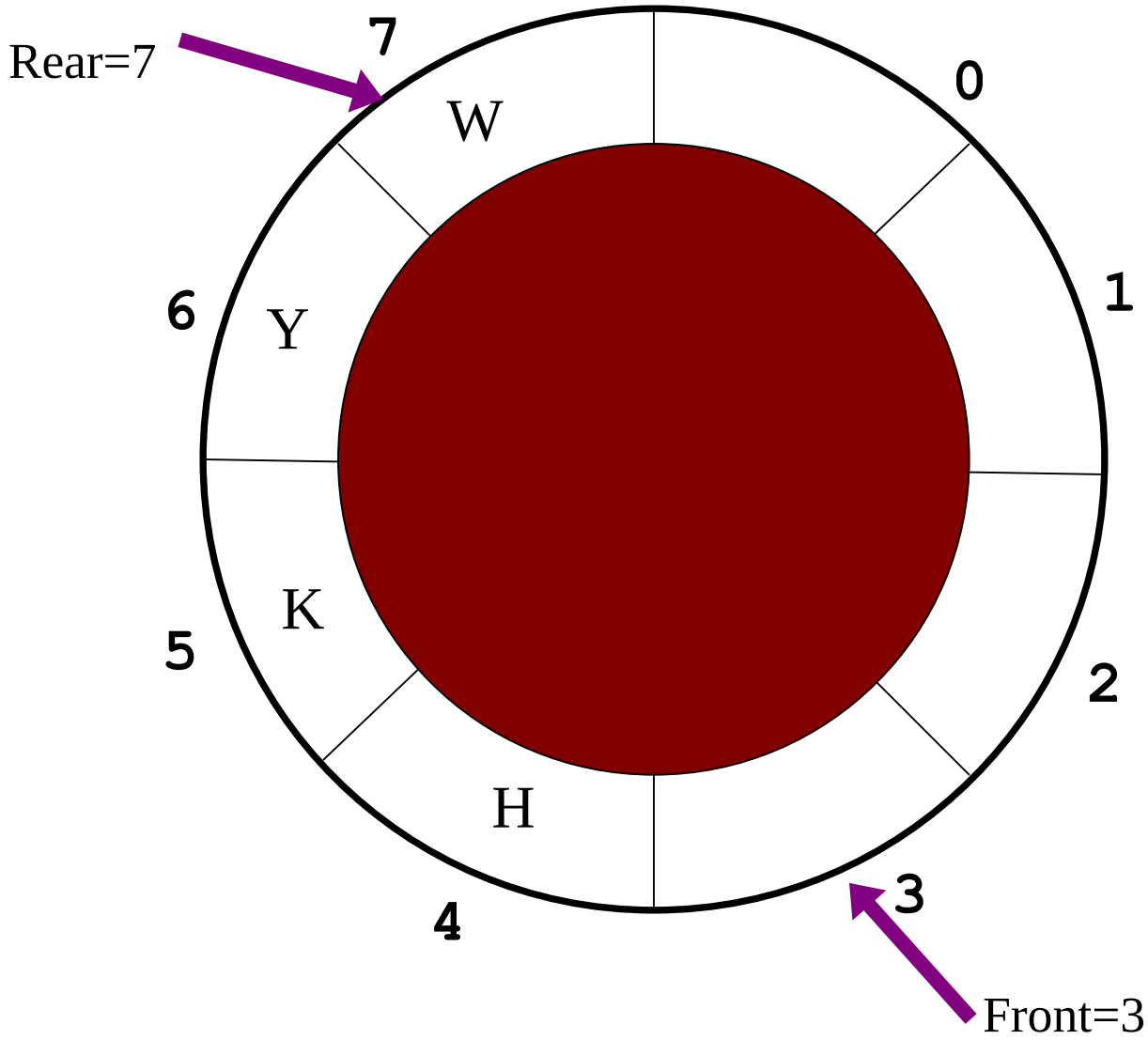





```
CreateQ();
Enqueue("A")
Enqueue("B")
Enqueue("D")
Dequeue()
Enqueue("H")
Enqueue("K")
Dequeue()
Dequeue()
```

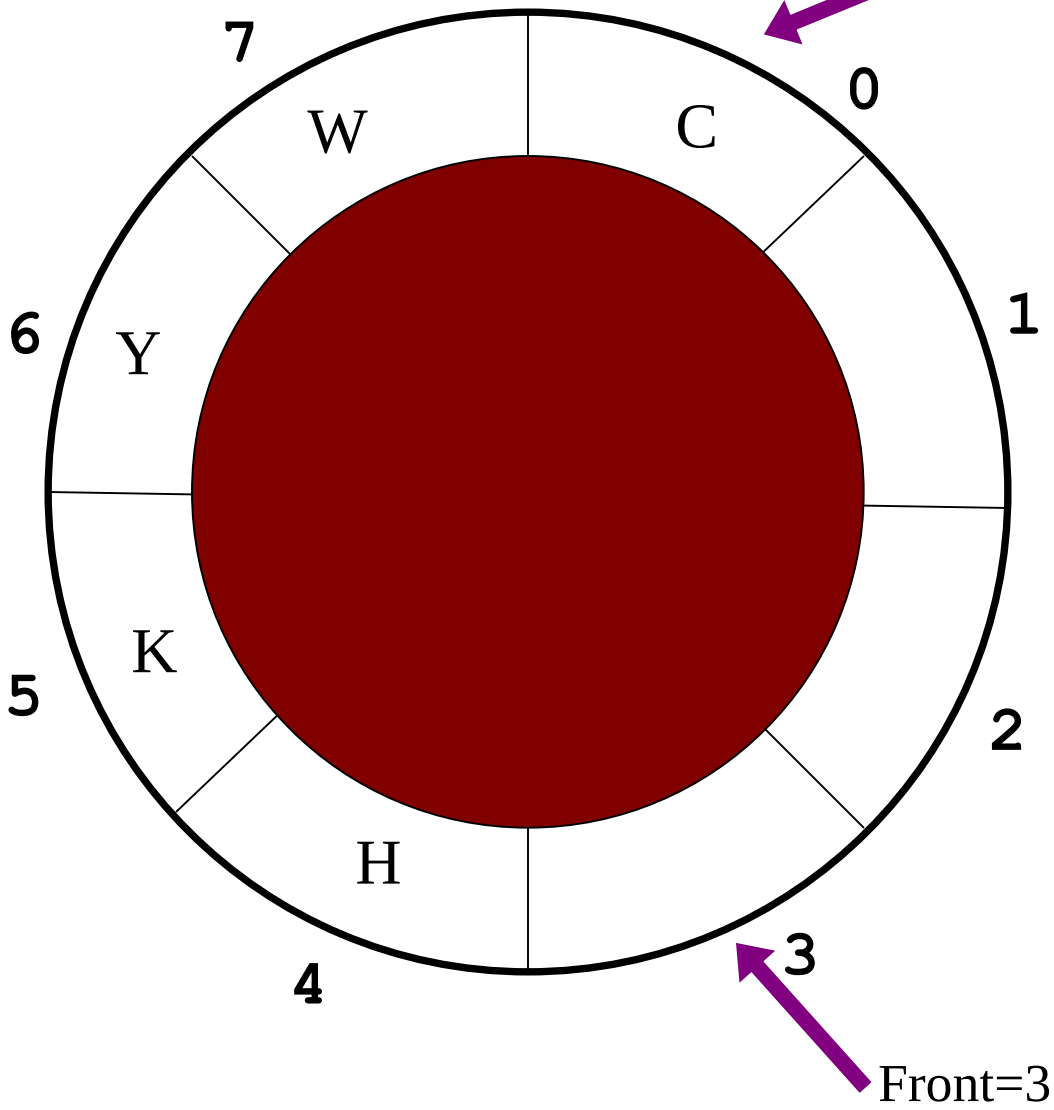



```
CreateQ();
Enqueue("A")
Enqueue("B")
Enqueue("D")
Dequeue()
Enqueue("H")
Enqueue("K")
Dequeue()
Dequeue()
Enqueue("Y")
```



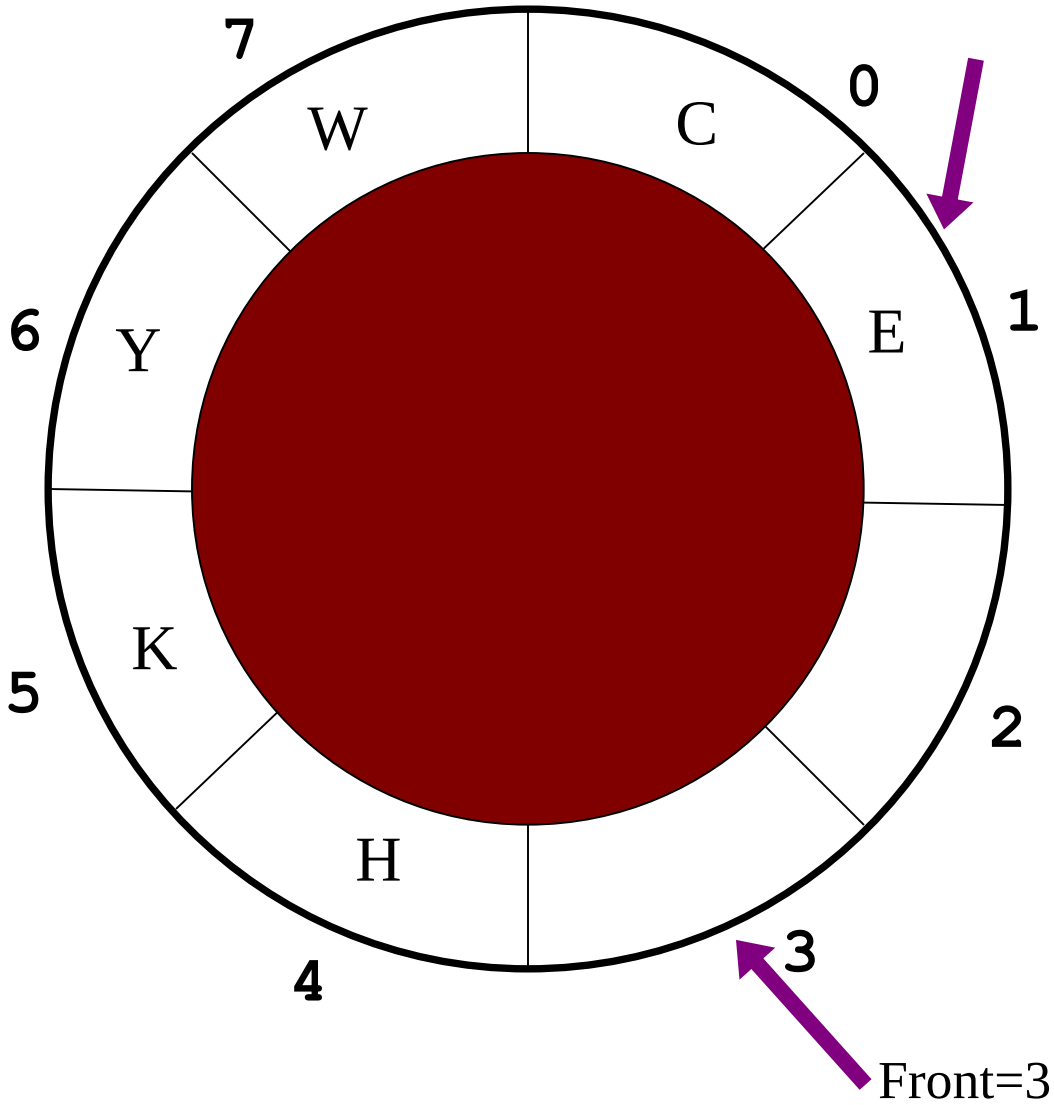

```
CreateQ();
Enqueue("A")
Enqueue("B")
Enqueue("D")
Dequeue()
Enqueue("H")
Enqueue("K")
Dequeue()
Dequeue()
Enqueue("Y")
Enqueue("W")
```

Rear=0((7+1)%8)

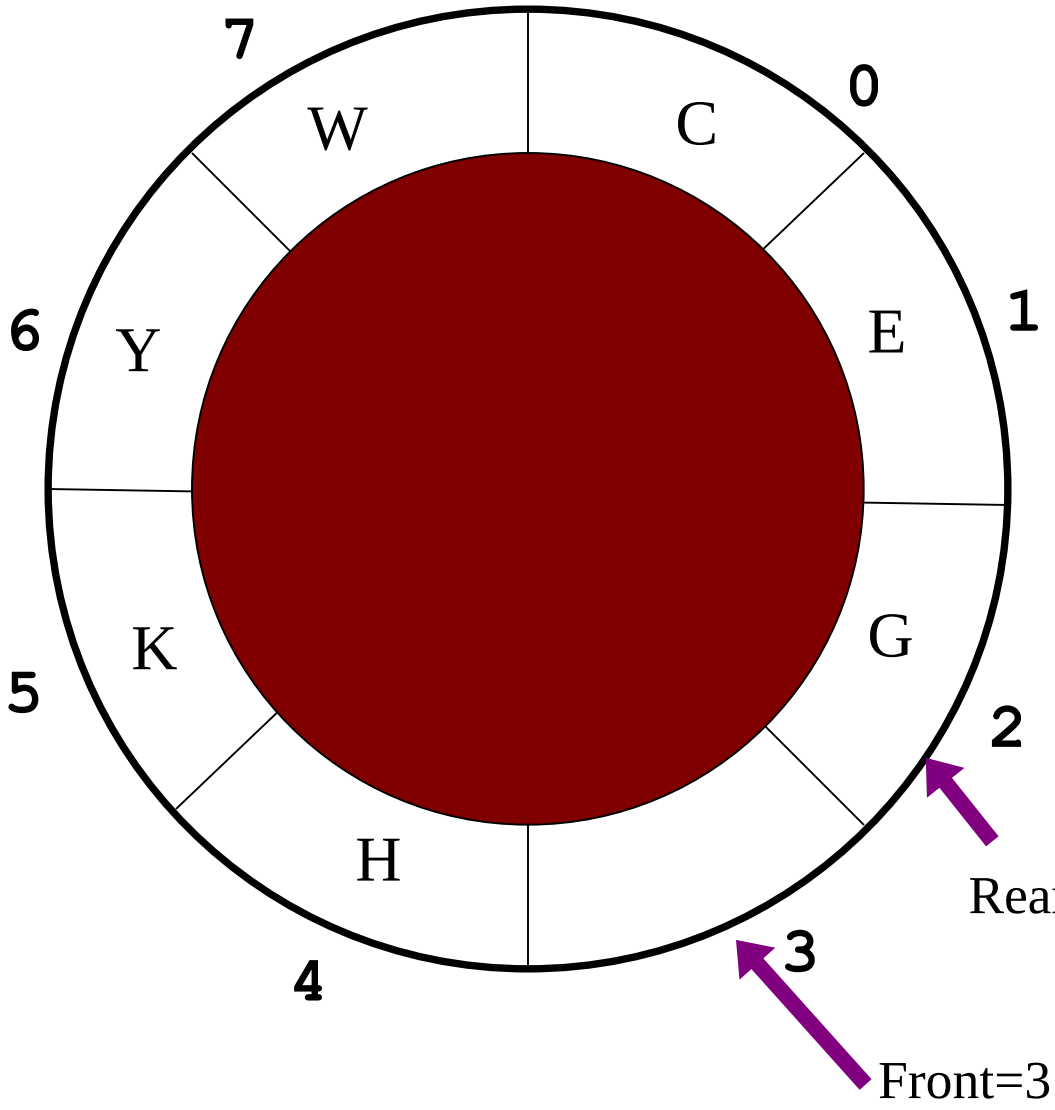



```
CreateQ();
Enqueue("A")
Enqueue("B")
Enqueue("D")
Dequeue()
Enqueue("H")
Enqueue("K")
Dequeue()
Dequeue()
Enqueue("Y")
Enqueue("W")
Enqueue("C")
```

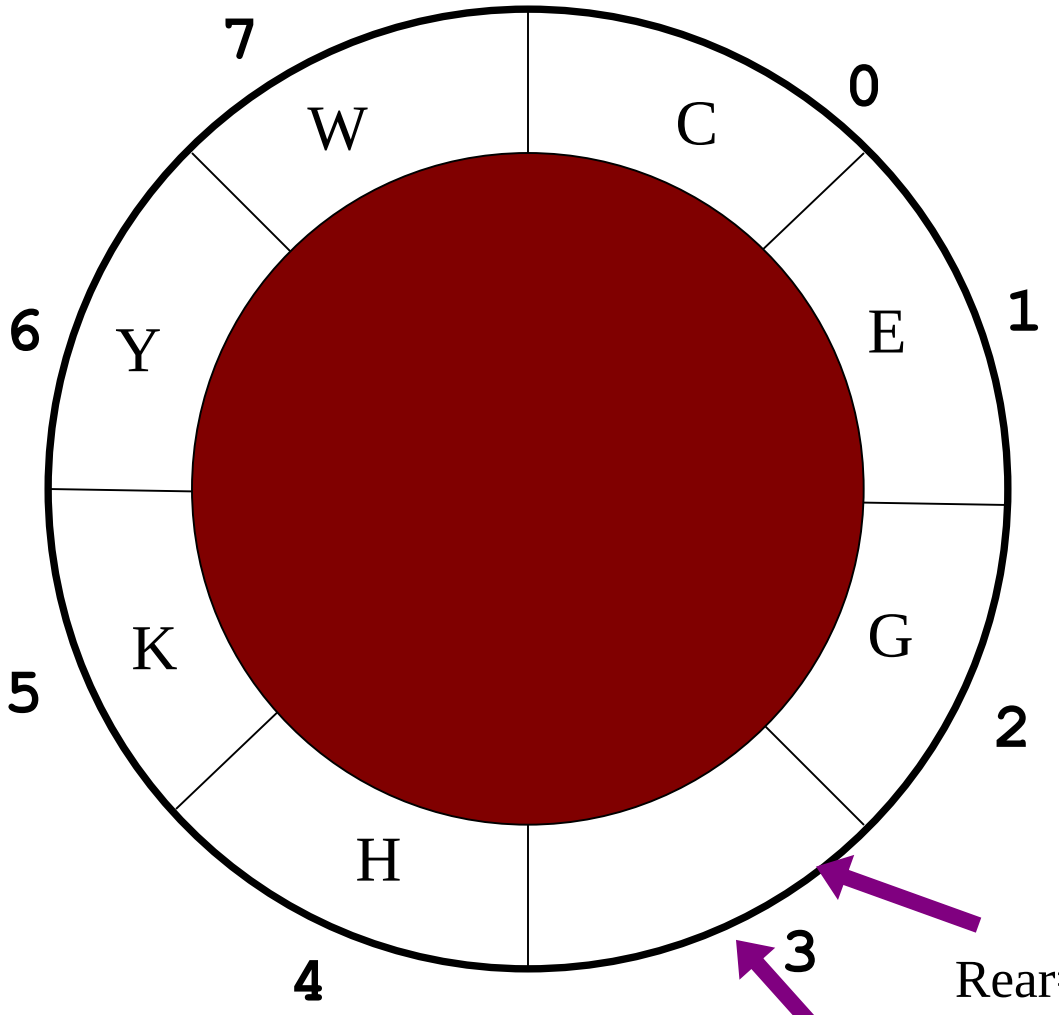
Rear=1((0+1)%8)




```
CreateQ();
Enqueue("A")
Enqueue("B")
Enqueue("D")
Dequeue()
Enqueue("H")
Enqueue("K")
Dequeue()
Dequeue()
Enqueue("Y")
Enqueue("W")
Enqueue("C")
Enqueue("E")
```




```
CreateQ();
Enqueue("A")
Enqueue("B")
Enqueue("D")
Dequeue()
Enqueue("H")
Enqueue("K")
Dequeue()
Dequeue()
Enqueue("Y")
Enqueue("W")
Enqueue("C")
Enqueue("E")
Enqueue("G")
```




```
CreateQ();
Enqueue("A")
Enqueue("B")
Enqueue("D")
Dequeue()
Enqueue("H")
Enqueue("K")
Dequeue()
Dequeue()
Enqueue("Y")
Enqueue("W")
Enqueue("C")
Enqueue("E")
Enqueue("G)
Enqueue("S")
```

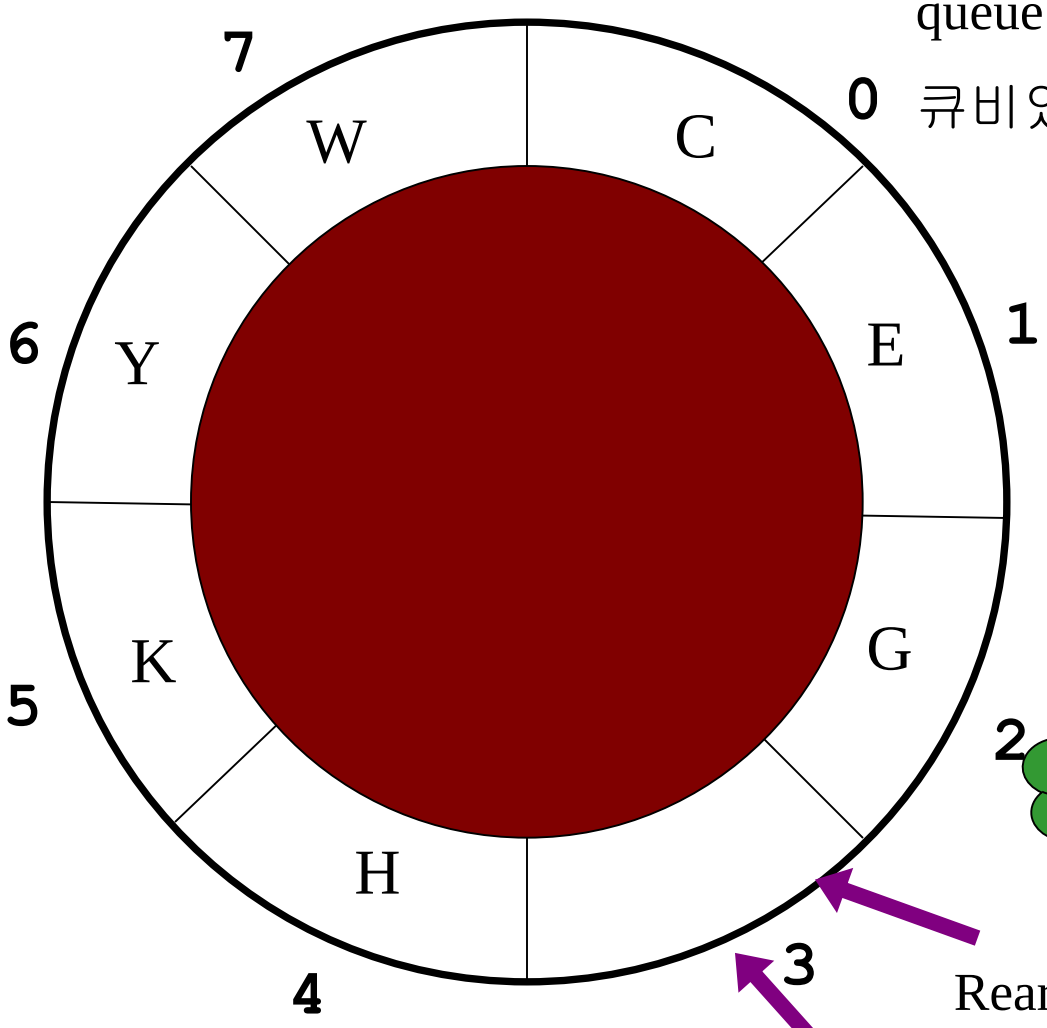
Rear=3

Front=3

Queue full

When **`front==rear`**, is queue empty or full?

큐비었음?꽉찼음?



Our circular array must be 1 cell larger than the # of values it's expected to contain at any given time.

원형큐를 꽉채우면
큐가비었을때와
구분불가능 따라서
빈방하나는 못채움.
구별하기위해

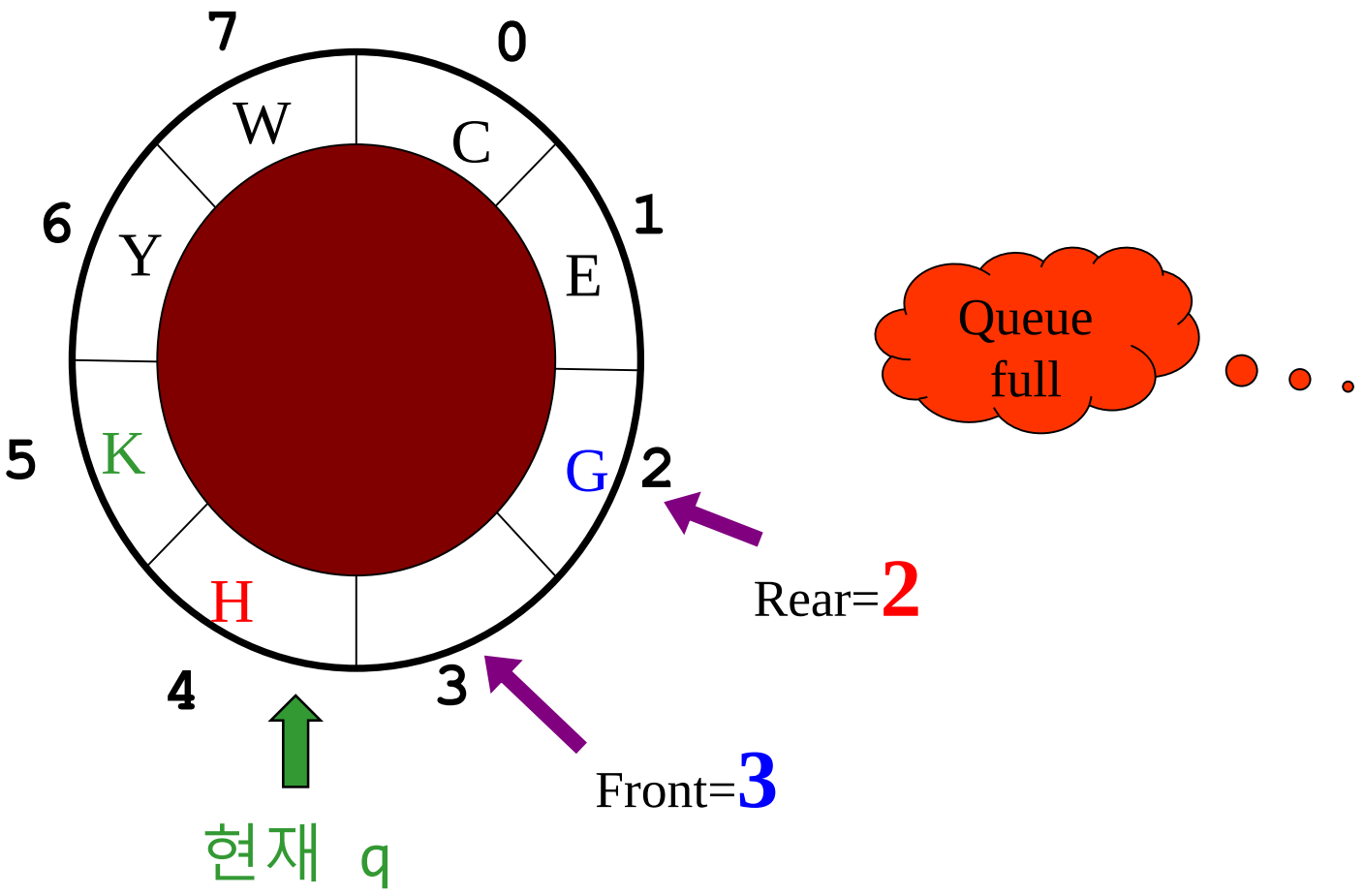
Queue Empty?

Rear=3

Front=3

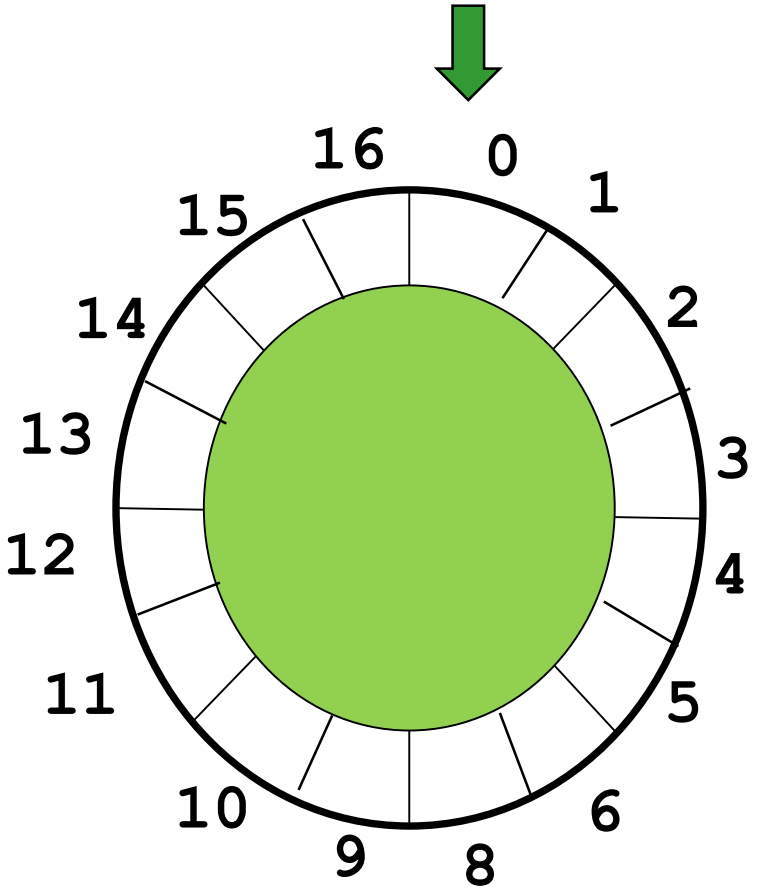
Queue full?

원형큐 오버플로우때 크기확장

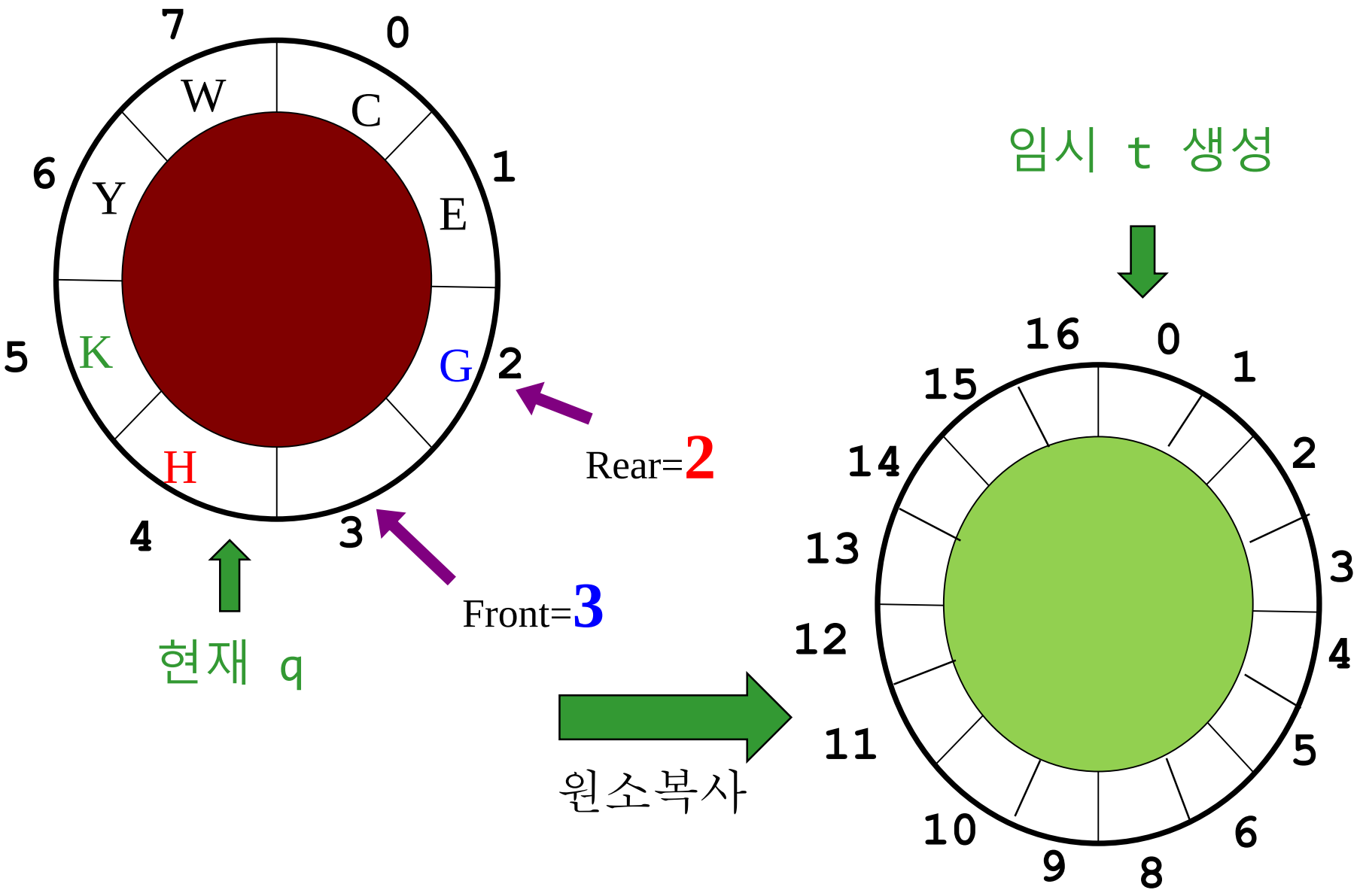


원형큐 오버플로우때 크기확장

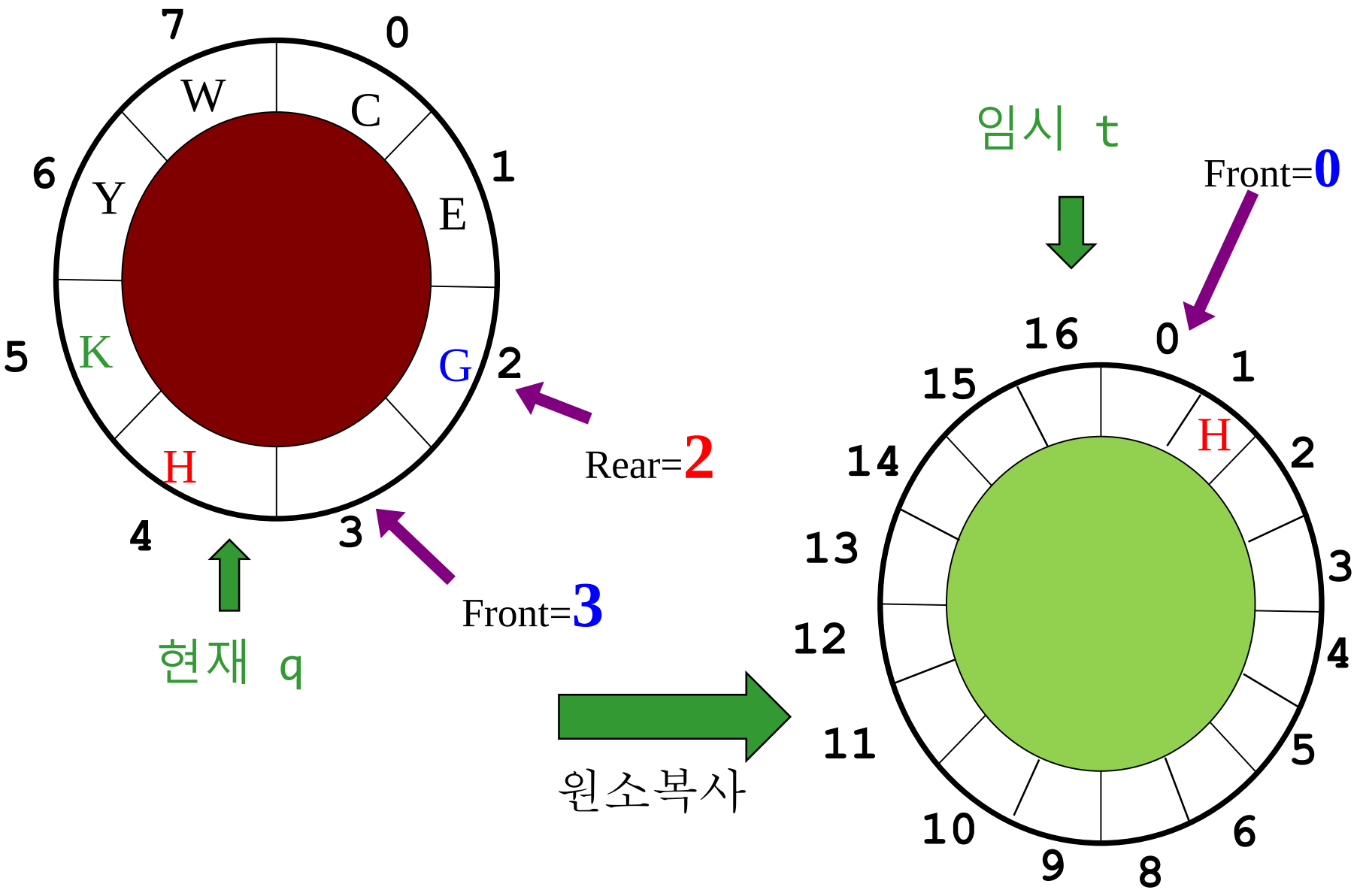
임시 원형큐 t 생성



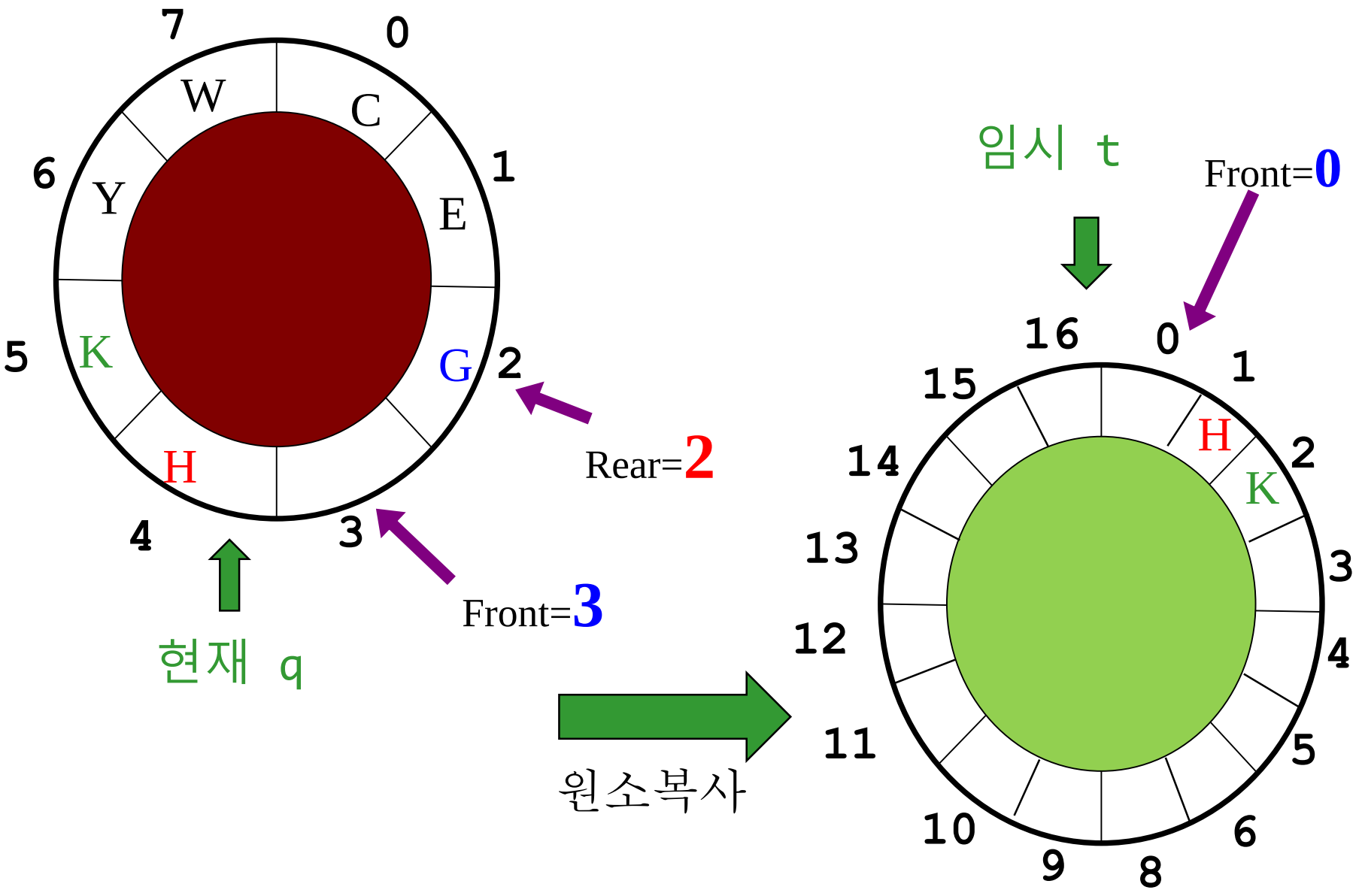
원형큐 오버플로우때 크기확장



원형큐 오버플로우때 크기확장



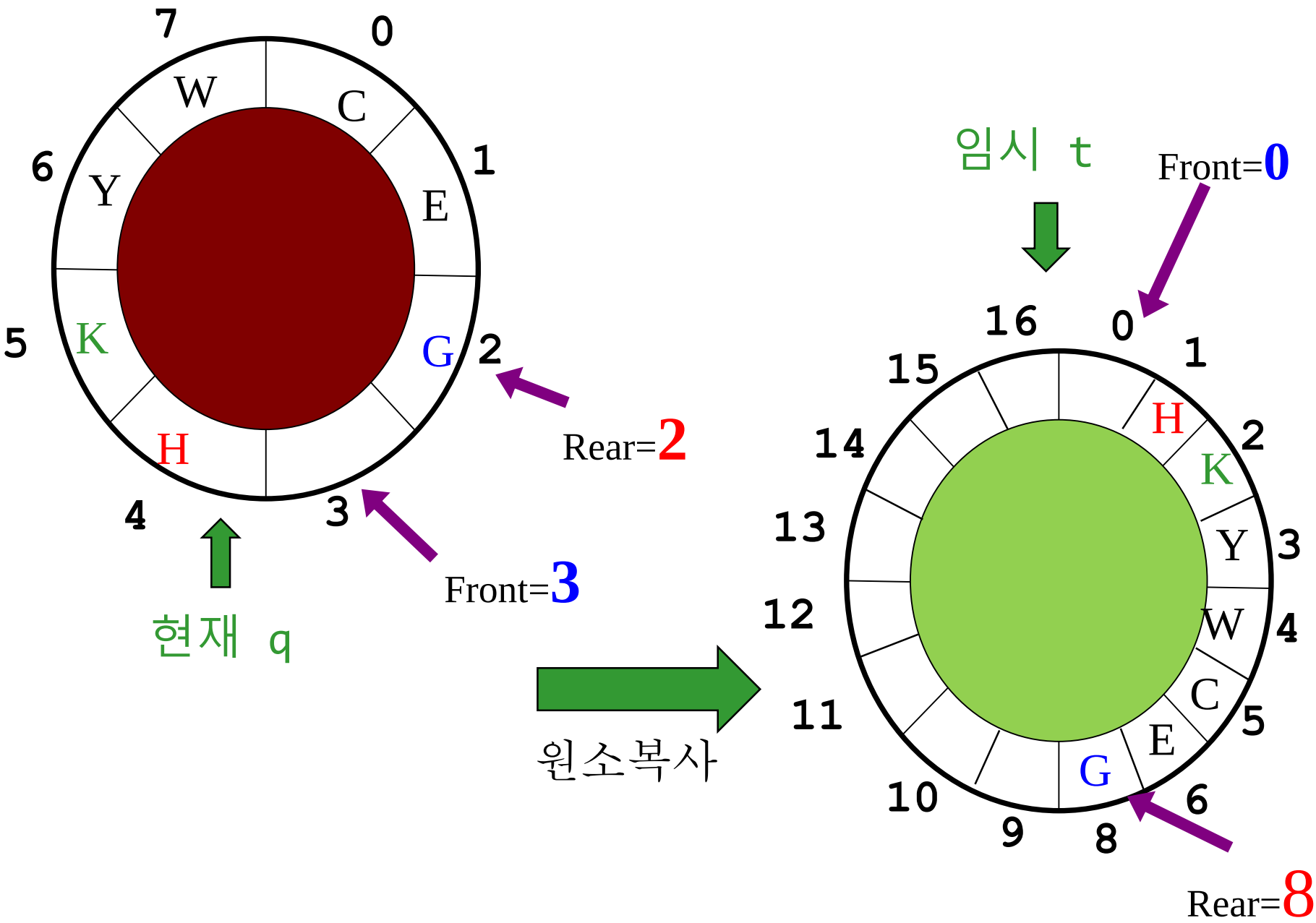
원형큐 오버플로우때 크기확장



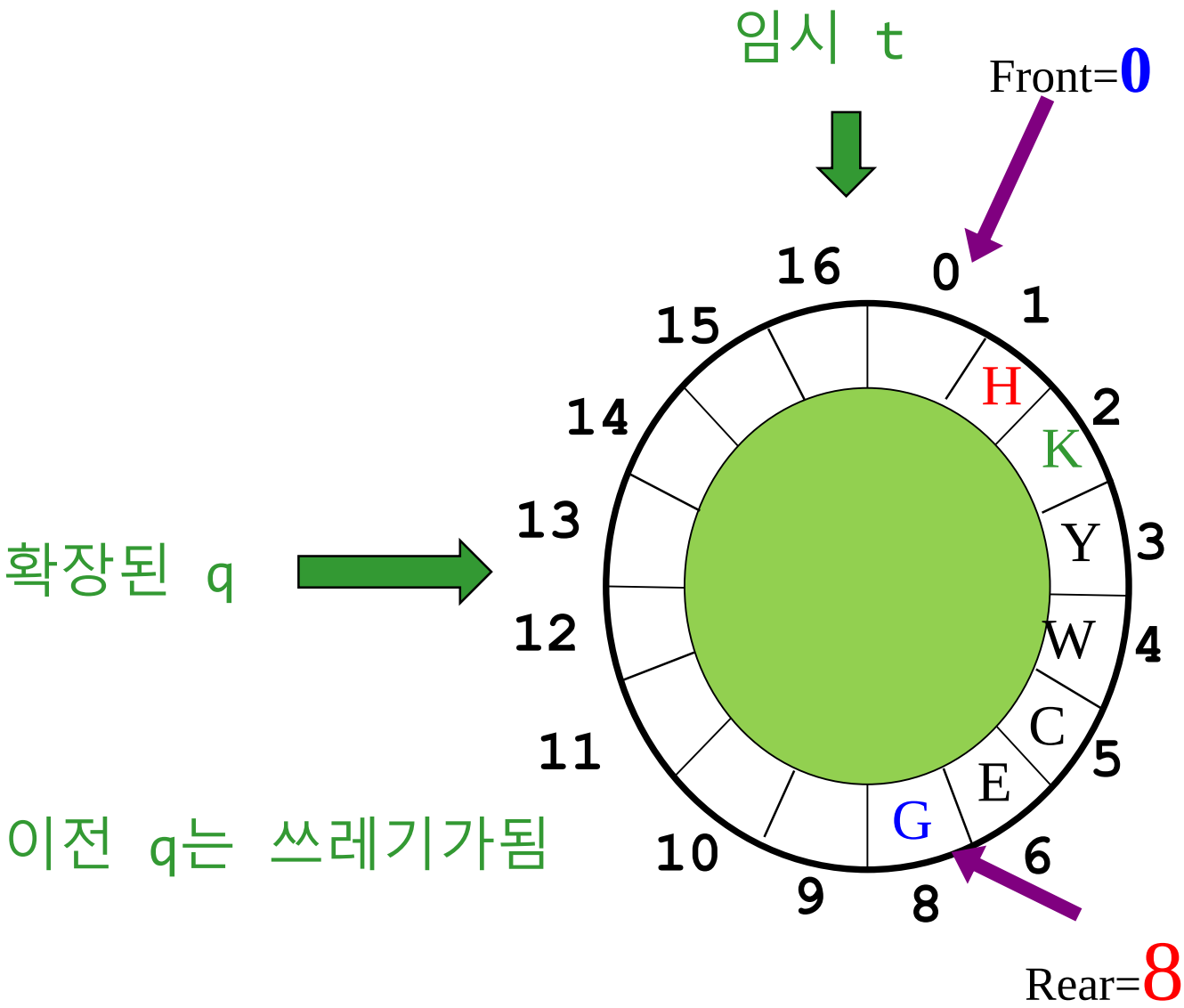
원형큐 오버플로우때 크기확장

원소들을 모두 복사

원형큐 오버플로우때 크기확장



원형큐 오버플로우때 크기확장



원형큐 오버플로우때 크기확장

배열을 사용할 시 연결리스트와 다른점

Size : 원소의 개수, 계산을 용이하게 하기 위해 추가

```
private void resize(int newSize){
    String[] t = new String[newSize];//새로운배열생성
    for (int i = 1, j = front + 1; i < size+1; i++, j++) {
        t[i] = q[j%q.length]; //하나씩 복사
    }
    front =0;
    rear = size;
    q = (String[]) t; //q가 t의 주소를 가지도록 함. 기존의 q는 가비지
}
```

Circular Arrays원형큐

교과서의 프로그램임.

참조할 것

알고리즘 구현이 약간 다르다

이 코드는 방하나가 낭비되지 않는다.

Implementation of array based circular queue 배열 원형 큐구현

- Queue ADT의 구현 방법
  - ✓Use interface in JAVA : 메소드에 대한 선언만 함
- Queue interface 정의

```
public interface Queue  {
    boolean isEmpty();     // 큐가 공백인가를 검사
    void enqueue(Object x);     // 원소 x를 삽입
    Object dequeue();     // 원소를 삭제하고 반환
    void remove();     // 원소를 삭제
    Object peek();     // 원소값만 반환
}
```

Implementation of array based circular queue(2)

- ArrayQueue 클래스 정의(1)

```
public class ArrayQueue implements Queue {
                        // 배열을 이용한 Queue interface의 구현
   private int front;              // 큐의 삭제 장소
   private int rear;               // 큐의 삽입 장소
   private int count;  // 큐의 원소 수
   private int queueSize;          // 큐(배열)의 크기
   private int increment;          // 배열의 확장 단위
   private Object[] itemArray;             // Java 객체 타입의 큐 원소를 위한 배열

   public ArrayQueue() {
                    // 무인자 큐 생성자
      front = 0;           // 초기화
      rear = 0;
      count = 0;
      queueSize = 50;  // 초기 큐 크기
      increment = 10;  // 배열의 확장 단위
      itemArray = new Object[queueSize];
   }

   public boolean isEmpty(){
      return (count == 0);
   }
```

배열을 이용한 큐의 구현(3)

- ArrayQueue 클래스 정의(2)

```
public void enqueue(Object x) {
                        // 큐에 원소 x를 삽입
      if (count == queueSize) queueFull();
      itemArray[rear] = x;  // 원소를 삽입
      rear = (rear + 1) % queueSize;
      count++;
   } // end enqueue( )

   public void queueFull() {
                       // 배열이 만원이면 increment만큼 확장
      int oldsize = queueSize;        // 현재의 배열 크기를 기록
      queueSize += increment; // 새로운 배열 크기
      Object[] tempArray = new Object[queueSize]; //확장된크기의임시배열
      for (int i = 0; i < count; i++) {
                        // 임시 배열로 원소들을 그대로 이동
           tempArray[i] = itemArray[front];
           front = (front + 1) % oldsize
      }
      itemArray = tempArray;  // 배열 참조 변수를 변경
      front = 0;
      rear = count;
   } // end queueFull()
```

배열을 이용한 큐의 구현(4)

- ArrayQueue 클래스 정의(3)

```
public Object dequeue( ){// 큐에서 원소를 삭제해서 반환
      if (isEmpty()) return null;     // 큐가 공백일 경우
           // 큐가 공백이 아닌 경우
      Object item = itemArray[front];
      front = (front + 1) % queueSize;
      count --;
      return item;
   } //end dequeue()

   public Object remove( ){// 큐에서 원소를 삭제
      if (isEmpty()) return null;     // 큐가 공백일 경우
                     // 큐가 공백이 아닌 경우
      front = (front + 1) % queueSize;
      count --;
   } //end remove()


   public Object peek( ) {     // 큐에서 원소값을 반환
      if (isEmpty()) return null;
      else return itemArray[front];
   } // end peek()

} //end ArrayQueue class
```

Circular Arrays원형큐
이 프로그램으로 해보자
이론시간에 배운 알고리즘으로 구현

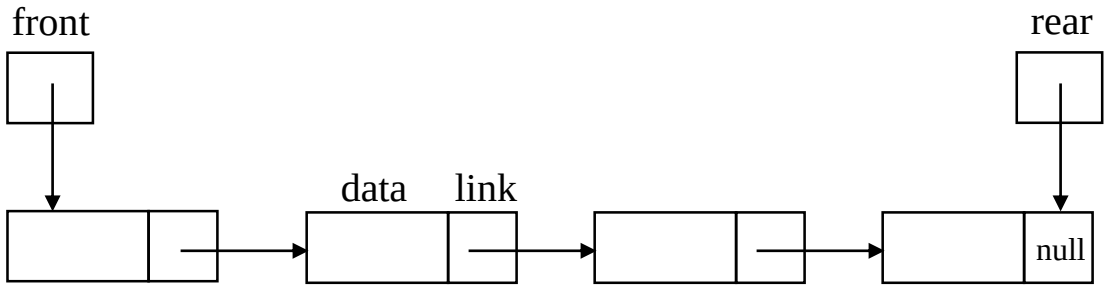
```
public class ArrayQueue {

  private String[] q;
  private int front, rear, size;

  public ArrayQueue() {
       q = (String[]) new String[2];
       front = rear = size = 0;
  }

  public int queueSize(){
      return size;
  }

  public boolean isEmpty(){
      return(size == 0);
  }

  public void add(String newItem){
        if ((rear + 1) % q.length == front) {
            // queue full
            resize(2*q.length);//크기2배확대
        }
        rear = (rear +1) % q.length;
        q[rear] = newItem;
        size++;
  }

  public String remove() {
      if (isEmpty()) {System.out.println("Underflow");
            item="Underflow";
      }
      front = (front +1) % q.length;
      String item = q[front];
      q[front] = null;
      size--;
      if (size > 0 && size == q.length/4){ //원소가 1/4면
            resize(q.length/2);//크기 반으로 축소
      }
      return item;
  }

  private void resize(int newSize){
      String[] t = new String[newSize];
      for (int i = 1, j = front + 1; i < size+1; i++, j++) {
          t[i] = q[j%q.length];
      }
      front =0;
      rear = size;
      q = (String[]) t;
  }

  public void printQueue(){
      for (int i = front + 1; i <= rear; i++){
          System.out.println(" " + q[i]);
      }
  }

} // public class ArrayQueue
```

```
public class QueueMain {

    public static void main(String[] ar){

        ArrayQueue aq1 = new ArrayQueue();
        aq1.add("banana");
        aq1.add("tomato");
        aq1.add("pineapple");
        //aq1.remove();
        //aq1.remove();
        aq1.printQueue();
    }

} //public class QueueMain
```

연결리스트로 큐 구현

Linked list queues연결리스트큐

- Linked list queue 연결리스트로 표현된 큐
  - ✓Linked list queue structure 연결리스트 큐 구조
    - Singly linked list with front, rear (후론트,리어를 가진 단일연결리스트)
    - init : front = rear = null (empty queue) (초기 빈큐)
    - Queue empty check: front or rear=null (빈큐확인)



Linked list queues연결리스트큐

- 연결 큐에서의 삽입, 삭제, 검색 연산 구현(1)

```
enqueue(q, item)// 연결 큐(q)에 item을 삽입
    newNode ← getNode();          // 새로운 노드를 생성
    newNode.data ← item;
    newNode.link ← null;          // 삽입되는 노드의 link 필드는 항상 null
    if (rear = null) then {       // 큐(q)가 공백인 경우
         rear ← newNode;
         front ← newNode;
    }
    else {
         rear.link ← newNode;
         rear ← newNode;
    }
end enqueue()
```

```
dequeue(q)
       // 큐(q)에서 원소를 삭제하고 값을 반환
    if (front = null) then queueEmpty();    // 큐(q)가 공백인 경우
    else {
        oldNode ← front;
        item ← front.data;
        front ← front.link;
        if (front = null) then rear ← null; // 삭제로 인해 큐(q)가 공백
        retNode(oldNode);
        return item;
     }
end dequeue()
```

Linked list queues(3)

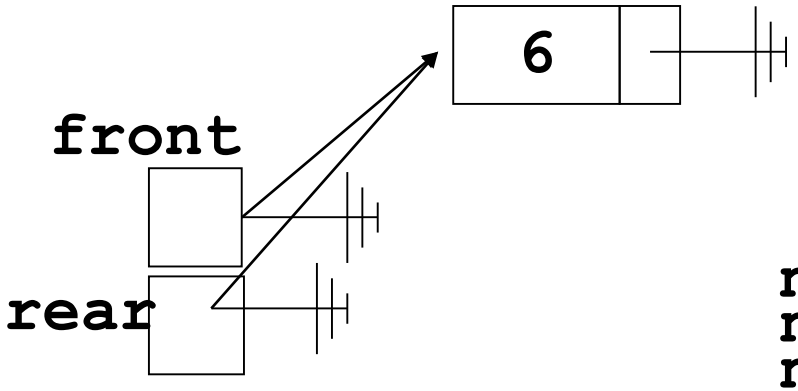
- 연결 큐에서의 삽입, 삭제, 검색 연산 구현(2)

```
delete(q)
        // 큐(q)에서 원소를 삭제
    if (front = null) then queueEmpty(); // 큐(q)가 공백인 경우
    else {
        oldNode ← front;
        front ← front.link;
        if(front = null) then rear ← null;// 삭제로 인해
큐(q)가 공백이 된 경우
        retNode(oldNode);
     }
end delete()

peek(q)
        // 큐(q)의 front 원소를 검색
    if (front = null) then queueEmpty(); // 큐(q)가 공백인 경우
    else return (front.data);
end peek()
```

Linked list queues(4)

- Characteristics of linked list queue연결큐의 특징
  - ✓Insertion,deletion don't require element movement 삽입, 삭제로 인한 다른 원소들의 이동이 필요 없음
  - ✓Operation is fast 연산이 신속하게 수행
  - ✓Simple operation for Multiple queues여러 개의 큐 운영시에도 연산이 간단
  - ✓Requires addition space for link fields 링크 필드에 할당하는 추가적인 저장 공간 필요

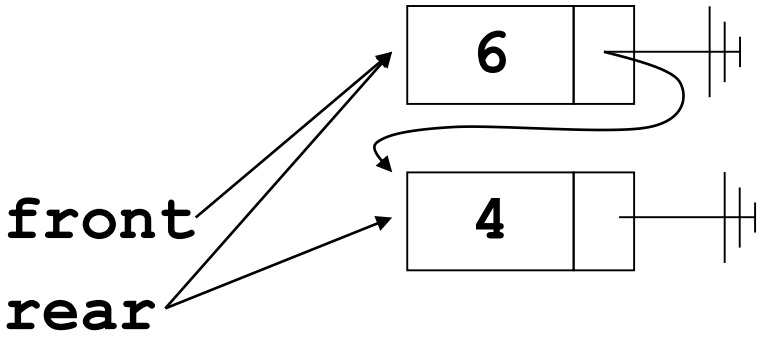
List Queue Example연결 큐 예




```
ListQueue();
enqueue(6);
```

```
newNode ← getNode(); 빵을찍고
newNode.data ← item;
newNode.link ← null;
if (rear = null) then {
        rear ← newNode;
      front ← newNode;
}
else {
      rear.link ← newNode;
      rear ← newNode;
}
```

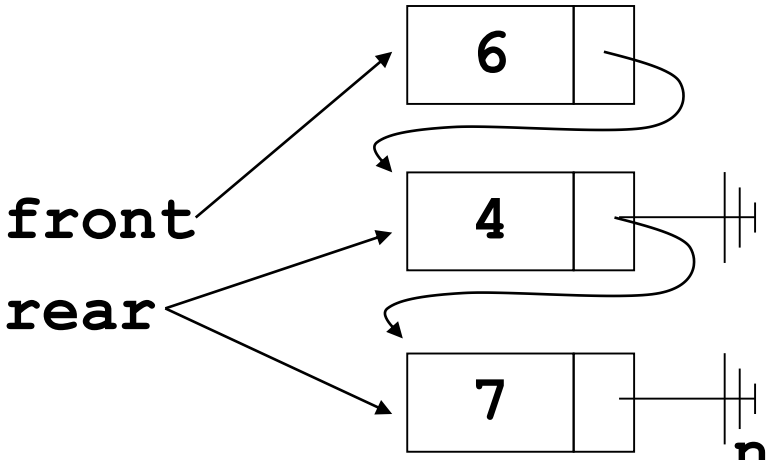
List Queue Example




```
ListQueue();
enqueue(6);
enqueue(4);
```

```
newNode ← getNode(); 빵을찍고
newNode.data ← item;
newNode.link ← null;
if (rear = null) then {
        rear ← newNode;
      front ← newNode;
}
else {
      rear.link ← newNode;
      rear ← newNode;
}
```

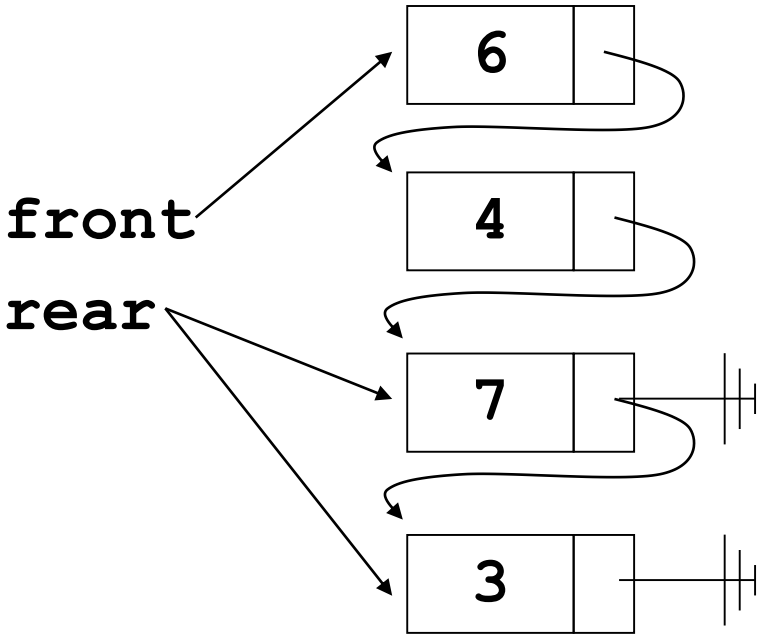
List Queue Example




```
ListQueue();
enqueue(6);
enqueue(4);
enqueue(7);
```

```
newNode ← getNode();  빵을찍고
newNode.data ← item;
newNode.link ← null;
if (rear = null) then {
        rear ← newNode;
        front ← newNode;
}
else {
        rear.link ← newNode;
        rear ← newNode;
}
```

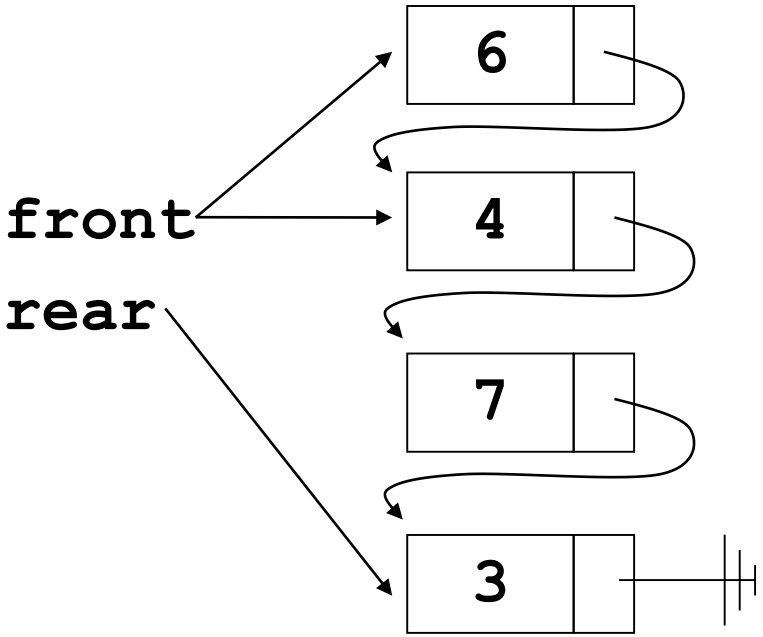
List Queue Example




```
ListQueue();
enqueue(6);
enqueue(4);
enqueue(7);
enqueue(3);
```

```
 newNode ← getNode();  빵을찍고
newNode.data ← item;
newNode.link ← null;
if (rear = null) then {
        rear ← newNode;
     front ← newNode;
}
else {
     rear.link ← newNode;
     rear ← newNode;
}
```

List Queue Example




```
ListQueue();
enqueue(6);
enqueue(4);
enqueue(7);
enqueue(3);
dequeue();
```

```
item ← front.data;
front ← front.link;
return item;
```

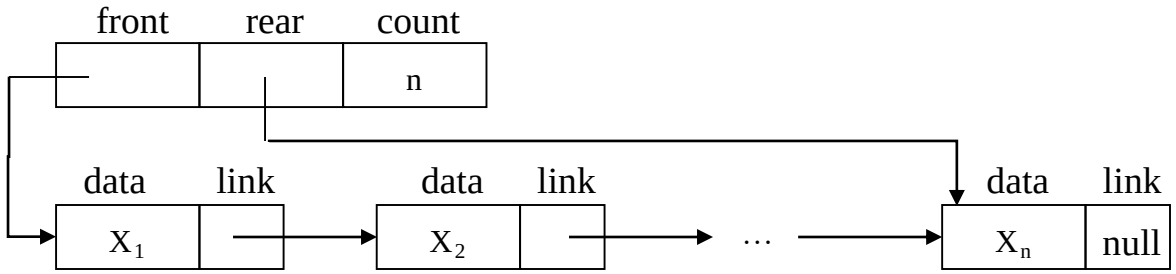
Implementation of linked list queues(1)
연결리스트큐구현

- 먼저 노드의 구조를 결정 : ListNode 클래스

```
public class ListNode {
      Object data;
      ListNode Link;
}
```

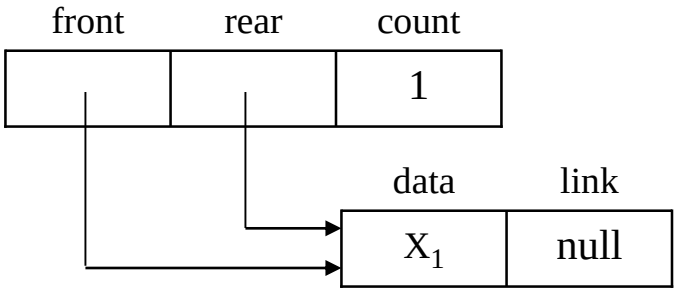
Implementation of queues(2)

- 큐를 구현한 연결 리스트 표현



공백이 아닌 큐를 구현한 연결 리스트 표현

공백 큐를 구현한 연결 리스트 표현



하나의 노드를 가진 큐의 연결 리스트 표현

Implementation of queues(3)

- 연결 리스트로 큐를 구현한 ListQueue 클래스(1)

```
public class ListQueue implements Queue {
                        // 연결 리스트를 이용한 Queue interface의
구현
   private ListNode front;              // 큐에서의 front 원소
   private ListNode rear;               // 큐에서의 rear 원소
   private int count;           // 큐의 원소 수

   public ListQueue() {
            // 공백 큐를 생성
      front = null;
      rear = null;
      count = 0;
   }
  public boolean isEmpty() {
      return (count == 0);
   }
```

Implementation of queues(4)

- ListQueue 클래스(2)

```
public void enqueue(Object x) {
                    // 큐에 원소 x를 삽입
    ListNode newNode = new ListNode();
    newNode.data = x;
    newNode.link = null;
    if (count == 0) {   // 큐(리스트)가 공백인 경우
        front = rear = newNode;
    } else {
        rear.link = newNode;
        rear = newNode;
    }
    count++;
} // end enqueue()

public Object dequeue() {
                // 큐에서 원소를 삭제하고 반환
    if (count == 0) return null;
    Object item = front.data;
    front = front.link;
    if (front == null) {      // 리스트의 노드를 삭제 후 공백이 된 경우
        rear = null;
    }
    count-- ;
    return item;
} // end dequeue()
```

Implementation of queues(5)

- ListQueue 클래스(3)

```
public void remove() {
                // 큐에서 원소를 삭제
        if (count == 0) return null;
        front = front.link;
        if (front == null) {      // 리스트의 노드를 삭제 후 공백이 된
경우
            rear = null;
        }
        count-- ;
    } // end remove()

    public Object peek() {
        if (count == 0) return null;
        else return front.data;
    } // end peek()

} // end ListQueue class
```

Application of queues-O.S

- 컴퓨터 운영체제에서 큐의 응용
  - ✓ 상이한 속도로 실행하는 두 프로세스 간의 상호 작용을 조화시키는 버퍼 역할을 담당
  - ✓ 예: CPU와 프린터 사이의 프린트 버퍼(printBufferQueue)
  - ✓ 프린트 버퍼에서의 판독과 기록 작업

```
writeLine()
     // CPU가 프린트해야할 라인을 프린트 버퍼 큐에 삽입
  if (there is a line L to print) and (printBufferQueue ≠ full)
  and (printBufferQueue ≠ busy) then
     enqueue(printBufferQueue, L);
end writeLine()

readLine()
      // 프린터가 프린트 버퍼 큐의 라인들을 프린트
  if(printBufferQueue ≠ empty) and (printBufferQueue ≠ busy) then {
     L ← dequeue(printBufferQueue);
     print L;
  }
end readLine()
```

Implement a stack using single queue

- 하나의 큐로 스택을 구현할 수 있을까?
- Push() : enqueue deque->deque
- Pop() : dequeue

Implement a stack using single queue

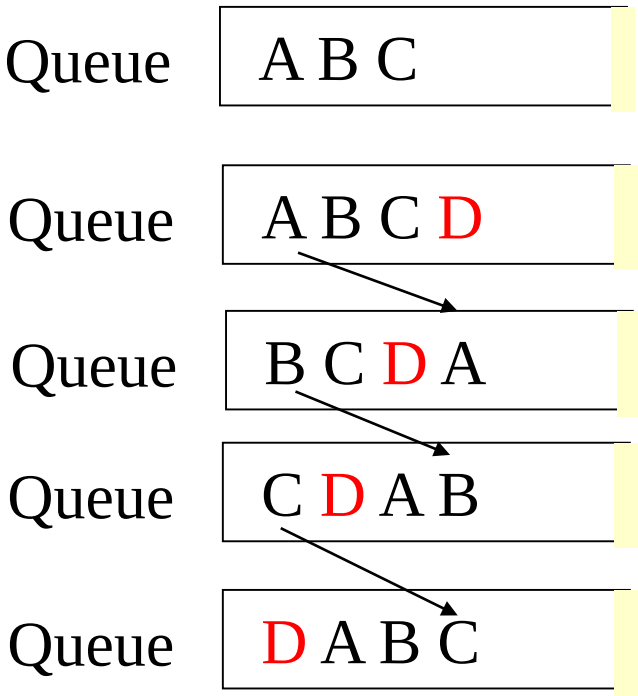
- 하나의 큐로 스택을 구현할 수 있을까?
- Push() : enqueue deque->deque
- Pop() : dequeue

Implement a stack using single queue

- 하나의 큐로 스택을 구현할 수 있을까?
- Push() : enqueue deque->deque
- Pop() : dequeue

Implement a stack using single queue

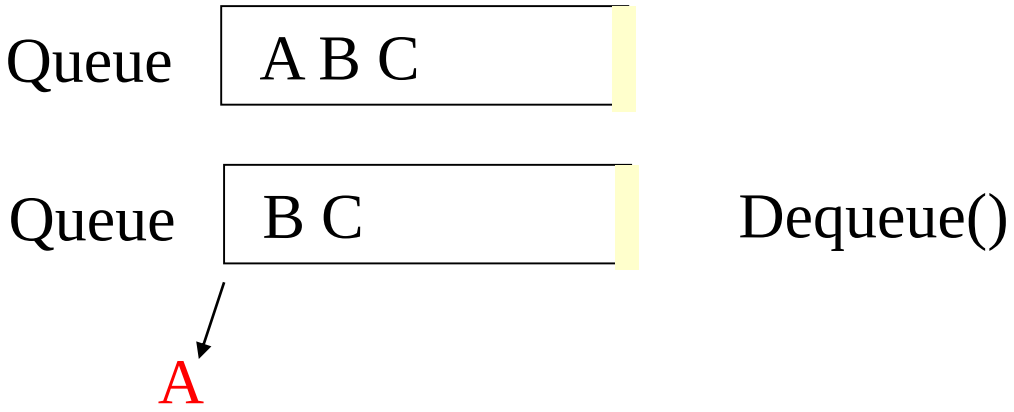
- 하나의 큐로 스택을 구현할 수 있을까?
- Push(D) : 위에 원소 추가




```
void push(int val) {
    // get previous size of queue
    int size = q.size();
    q.add(val);

    for (int i = 0; i < size; i++) {

    //int x = q.remove(); q.add(x);
    q.add(q.remove());
}
```

Implement a stack using single queue

- 하나의 큐로 스택을 구현할 수 있을까?
- Pop() : 선두 원소 제거 dequeue()와 같음



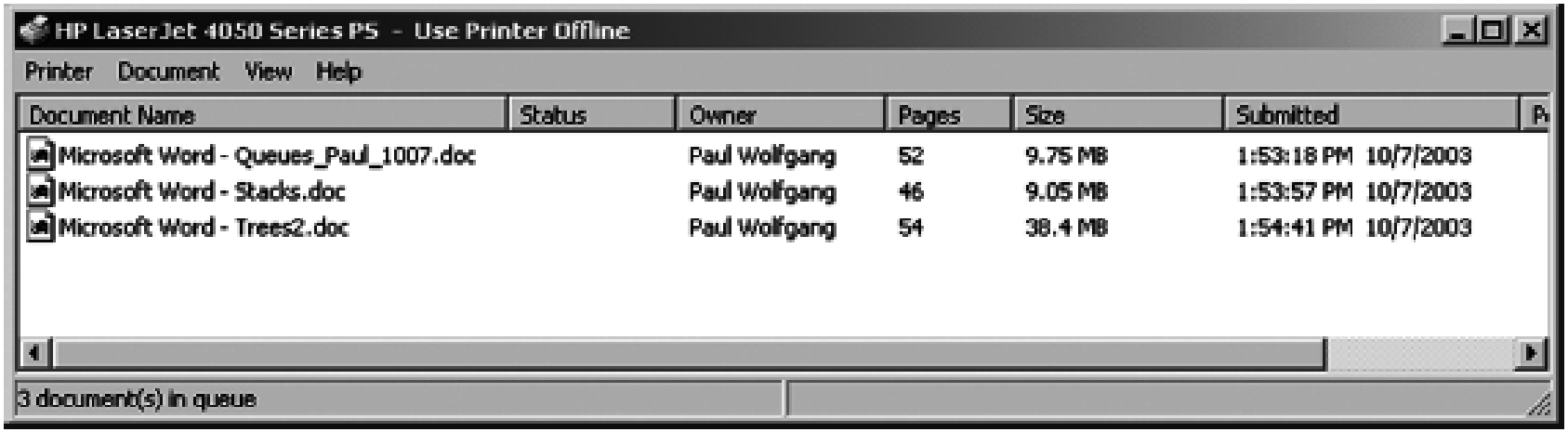

```
int pop() {
        return q.remove();
}
```

Application of Q 큐의 응용

- Print Queue 프린트 큐

FIGURE 6.2

A Print Queue in the Windows Operating System



Queue Summary큐종합

- Queue Operation Complexity for Different Implementations큐의 시간복잡도

	Array 배열 Fixed-Size	List 연결리스트 Singly-Linked
dequeue()	O(1)	O(1)
enqueue(o)	O(1)	O(1)
front()	O(1)	O(1)
Size(), isEmpty()	O(1)	O(1)

컴퓨터 시뮬레이션(1)

- 컴퓨터 시뮬레이션의 정의

✓어떤 물리적 시스템의 행태를 분석하고 예측하기 위해 컴퓨터 모델을 통해 시뮬레이트하는 것

✓물리적 시스템

- 특정 목적을 달성하기 위해 동작하고 상호 작용하는 독립적인 원소나 개체의 집합

컴퓨터 시뮬레이션(2)

- 물리적 시스템의 상태
  - ✓일련의 상태 변수(state variable)를 사용하여 표현
    - 공항의 항공기 교통 시뮬레이션 예
      - 개체: 항공기
      - 상태: 항공기의 상태 (공중, 지상에서 착륙이나 이륙)
  - ✓보통 시간대별로 측정
    - 연속적 시스템(continuous system) 시뮬레이션
      - 시스템 상태는 시간에 따라 연속적으로 변함
    - 이산 시스템(discrete system) 시뮬레이션
      - 상태 변수들은 어떤 특정 사건 발생 시점에서만 변함

컴퓨터 시뮬레이션(3)

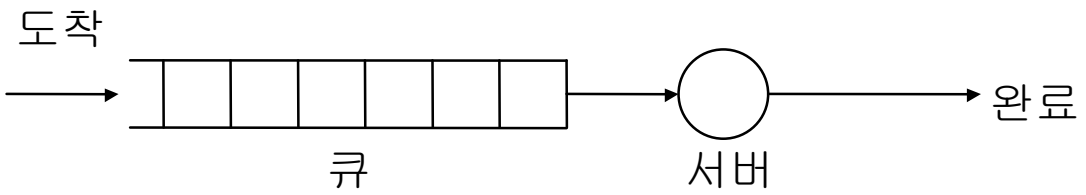
- 시뮬레이션에서의 시간
  - ✓물리적 시스템에서의 시간을 의미하는 시뮬레이트 되는 시간 (simulated time)과 시뮬레이션 프로그램에서의 연산 시간 (computation time)을 분명하게 구별 해야함
  - ✓대부분의 경우 시뮬레이트 되는 시간보다 연산 시간이 훨씬 짧음
    - 예 : 기상 예측 시스템

컴퓨터 시뮬레이션(4)

- 큐잉 시스템(queueing system)
  - ✓큐잉 시스템의 특징
    - 대부분의 물리적 시스템을 모델링
    - 서비스를 기다리는 대기선(waiting line)을 가진 시스템
    - 대기선 : 큐의 선입선출(FIFO) 성질을 만족
  - ✓큐잉 시스템의 종류
    - 단일 서버 큐잉 시스템
    - 다중 큐 다중 서버 큐잉 시스템
    - 큐잉 네트워크

컴퓨터 시뮬레이션(5)

- 단일 서버 큐잉 시스템
  - ✓단일 큐, 단일 서버로 구성
  - ✓시뮬레이트
    - 가정
      - 고객 도착 시간 간격의 분포 : 고객들의 도착시간은 어떤 확률 분포 함수 A(t)에 따라 상이하다고 가정
      - 서비스 시간 분포 : 개개 고객의 서비스 시간은 어떤 확률 분포 함수 S(t)에 따라 상이하게 제공된다고 가정
    - 위의 정보를 기초로 큐의 평균 길이, 고객의 평균 대기 시간, 서버의 작업 처리 능력 등을 분석
  - ✓단일 서버 큐잉 시스템



컴퓨터 시뮬레이션(6)

- 큐잉 시스템 시뮬레이션
  - ✓ 구성 요소
    - 고객 큐 : 대기선을 모델링
    - 서버 : 고객들을 서비스
    - 스케줄러 : 시뮬레이션하는 동안 사건이 일어날 시간을 스케줄
  - ✓ 시뮬레이션에 필요한 자료
    - 새로운 고객 도착 : 확률 분포 함수 A(t)에 따라 정해지는 시간
    - 서비스 받고 시스템 나감 : 확률 분포 함수 S(t)에 따라 결정되는 시간
  - ✓ 시뮬레이션의 종류
    - 시간 중심 시뮬레이션(time-driven simulation)
      - 시뮬레이션 클록에 일정 단위 시간을 계속적으로 증가시키면서 시뮬레이션을 수행
    - 사건 중심 시뮬레이션(event-driven simulation)
      - 시뮬레이션 클록을 사건이 발생할 때마다 경과된 시간을 증가시키고, 상태 변수를 갱신하면서 시뮬레이션을 수행

컴퓨터 시뮬레이션(7)

- 단일 서버 큐잉 시스템 시뮬레이션(1)

```
timeDrivenSimulation(startSimulation, endSimulation)
        // startSimulation: 시뮬레이션 시작 시간
        // endSimulation: 시뮬레이션 종료 시간
        // 필요한 여러 상태 변수들을 초기화
  clock ← startSimulation;   // 시뮬레이션 시계(clock)
  arrivalTime;       // 고객이 서비스를 받기 위해 시스템에 도착한 시간
  leaveTime;         // 고객이 서비스를 받고 떠나는 시간
  serverState;       // 서버의 busy(서비스 중) 또는 idle(휴식 중) 상태를 표현
  tick;    // 시뮬레이션 시계의 단위 시간
  queue;             // 고객이 서비스를 받기 위해 대기하는 큐
```

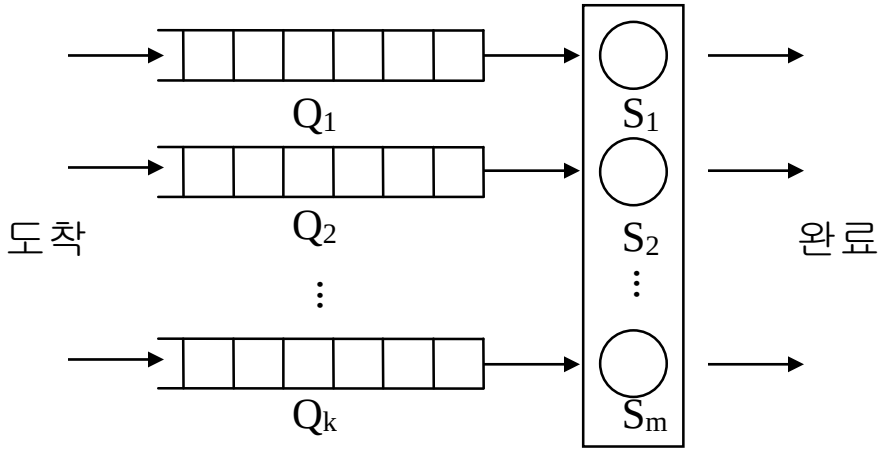
컴퓨터 시뮬레이션(8)

- 단일 서버 큐잉 시스템 시뮬레이션(2)

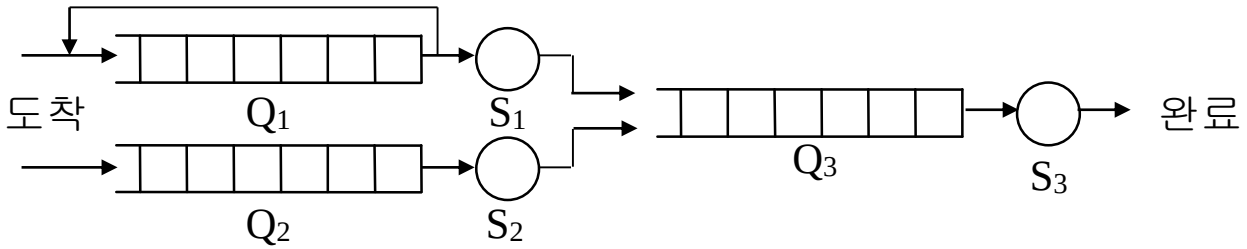
```
while (clock < endSimulation) do {// 시뮬레이션 종료 시간이 되지 않은 경우
    clock ← clock + tick;   // 클록의 단위 시간을 증가
    if (clock ≥ arrivalTime) then {
            // 새로운 고객이 도착한 경우
       enqueue(queue, customer);
            // 고객(customer), arrivalTime은 큐로 들어가 서비스를 대기
       update(statistics);  // 통계 분석 데이타(statistics) 처리
       arrivalTime ← clock + arrival(clock);
                    // 다음 고객 도착 시간을 생성
                    // arrival(): 고객의 도착 시간 간격
    }
    if (clock ≥ leaveTime) then  // 고객의 서비스가 완료된 경우
       serverState ← idle;          // 서버는 쉬는 상태로
    if (serverState = idle and not isEmpty(queue)) then {
                                    // 다음 고객을 서비스할 수 있는가를 점검
       dequeue(queue);              // 다음 차례의 고객에 서비스 시작
       update(statistics);          // 통계 분석 데이타(statistics) 처리
       serverState ← busy;          // 서버를 서비스 중인 상태로
       leaveTime ← clock + service(clock); // 고객이 서비스를 받고 떠나는 시간을 설정
                                           // service(t): 고객의 서비스 시간 생성

    }
  }  // end while
end timeDrivenSimulation()
```

컴퓨터 시뮬레이션(9)

- 복잡한 큐잉 시스템
  - ✓다중 큐 다중 서버 큐잉 시스템



  - ✓큐잉 네트워크



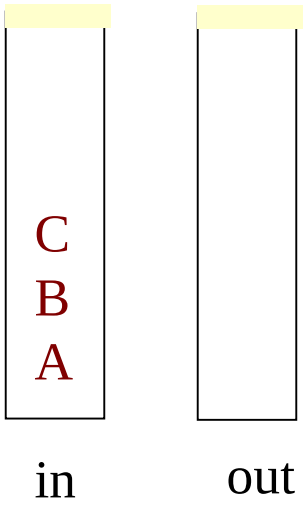
Stack 2개로 큐를 구현할 수 있을까?

Queue with two stacks

A clever and simple queue implementation using only stacks

```
class Queue<E> {
  Stack<E> in  = new Stack<E>();
  Stack<E> out = new Stack<E>();
  void enqueue(E x){ in.push(x); }
  E dequeue(){
    if(out.isEmpty()) {
      while(!in.isEmpty()) {
        out.push(in.pop());
      }
    }
    return out.pop();
  }
}
```

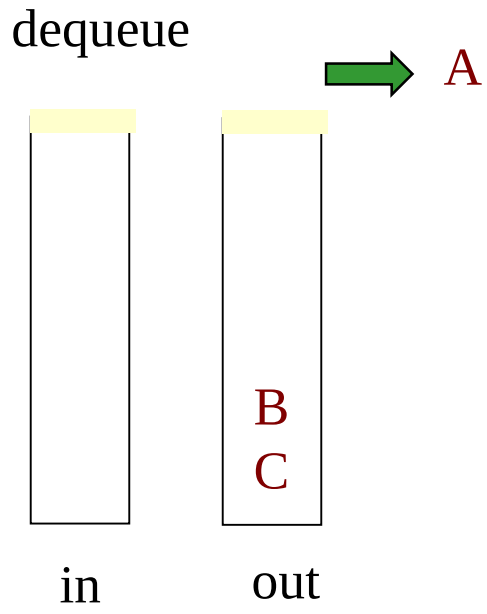
enqueue: A, B, C



Queue with two stacks

A clever and simple queue implementation using only stacks

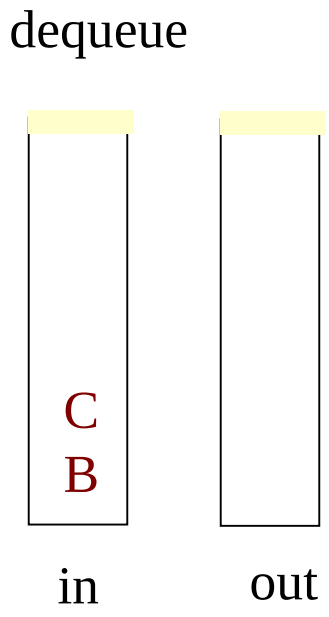
```
class Queue<E> {
  Stack<E> in  = new Stack<E>();
  Stack<E> out = new Stack<E>();
  void enqueue(E x){ in.push(x); }
  E dequeue(){
    if(out.isEmpty()) {
      while(!in.isEmpty()) {
        out.push(in.pop());
      }
    }
    return out.pop();
  }
}
```



Queue with two stacks

A clever and simple queue implementation using only stacks

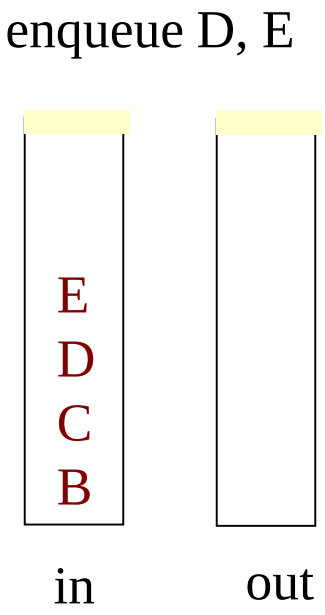
```
class Queue<E> {
  Stack<E> in  = new Stack<E>();
  Stack<E> out = new Stack<E>();
  void enqueue(E x){ in.push(x); }
  E dequeue(){
    if(out.isEmpty()) {
      while(!in.isEmpty()) {
        out.push(in.pop());
      }
    }
    return out.pop();
  }
}
```



Queue with two stacks

A clever and simple queue implementation using only stacks

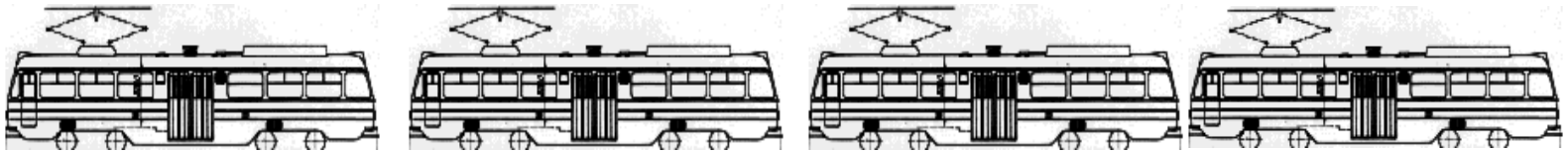
```
class Queue<E> {
  Stack<E> in  = new Stack<E>();
  Stack<E> out = new Stack<E>();
  void enqueue(E x){ in.push(x); }
  E dequeue(){
    if(out.isEmpty()) {
      while(!in.isEmpty()) {
        out.push(in.pop());
      }
    }
    return out.pop();
  }
}
```



Double-Ended Queues
데크

Front

Rear

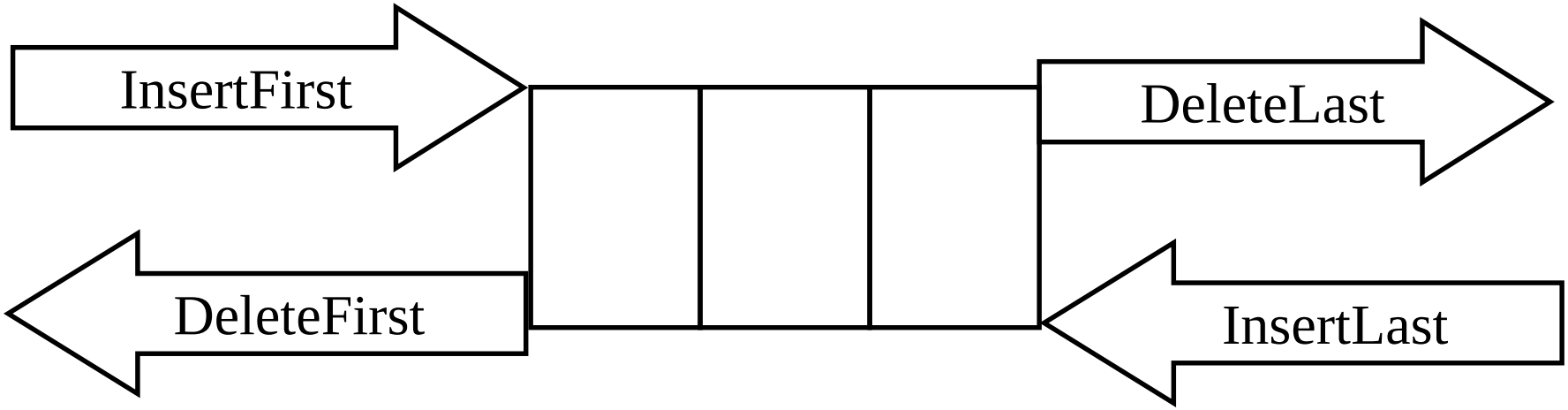


deque덱

- 덱(deque : double-ended queue)
  - ✓Ordered list with statck and queue properties스택과 큐의 성질을 종합한 순서 리스트
  - ✓Insertion and deletion occur at the both ends삽입과 삭제가 리스트의 양끝에서 임의로 수행될 수 있는 자료구조
  - ✓Operations in stack and queue are possible스택이나 큐 ADT이 지원하는 연산을 모두 지원

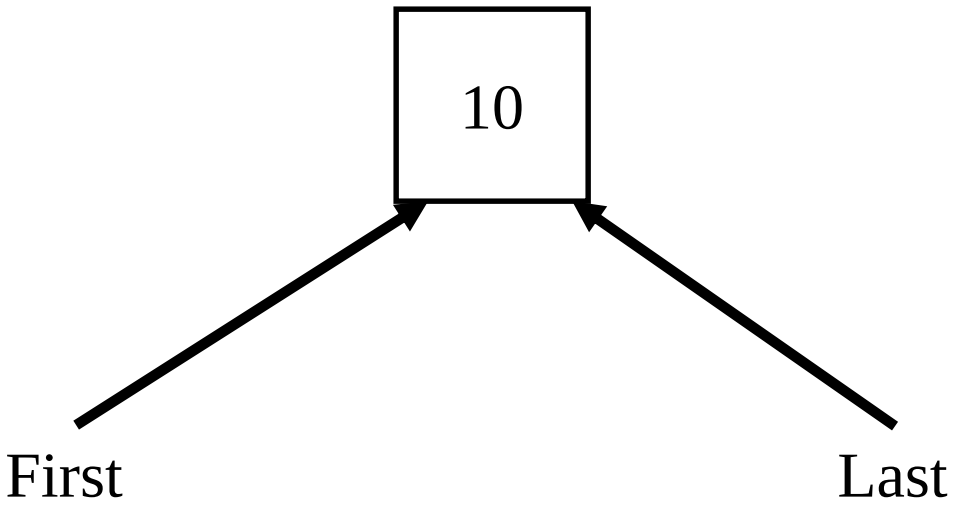
Deque 데크

- A Deque is a “Double Ended QUEue”. 데크는 양방향큐라는 의미
- A Deque is a restricted List which supports only 덱은 아래와 같은 작업만을 지원
- add to the last (last 에서 삽입)
- remove from the last(last 에서 삭제)
- add to the first(first에서 삽입)
- remove from the first(first에서 삭제)
- Stacks and Queues are often implemented using a Deque(스택과 큐를 덱으로 구현가능)

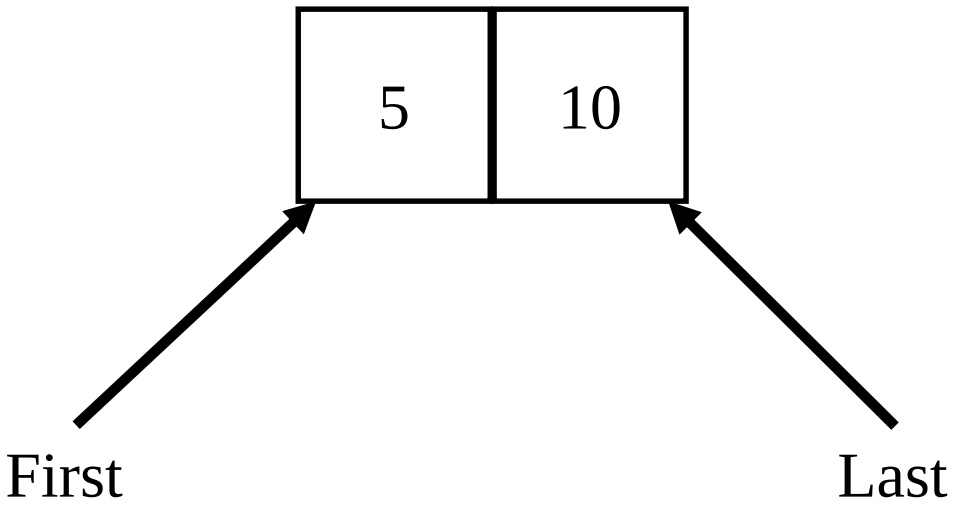

InsertFirst
DeleteLast
DeleteFirst
InsertLast


First
Last

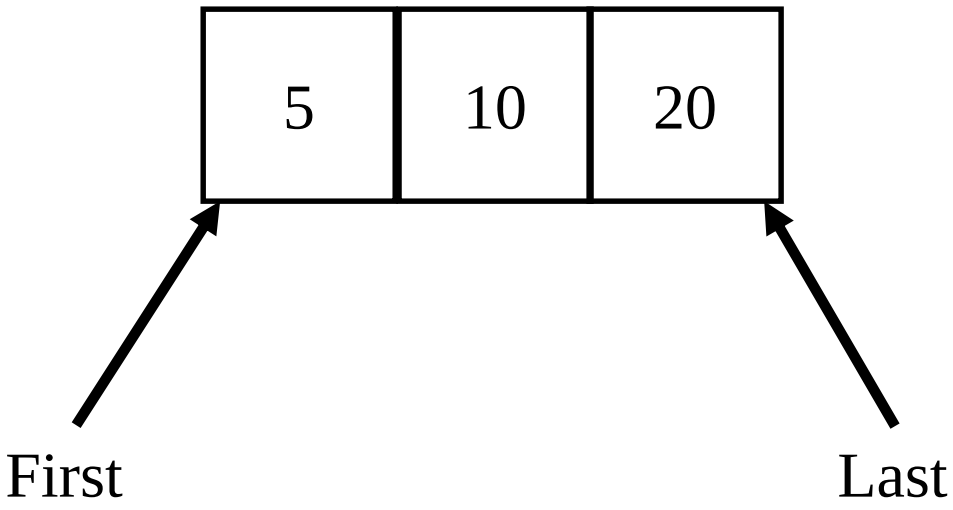
InsertFirst(10)



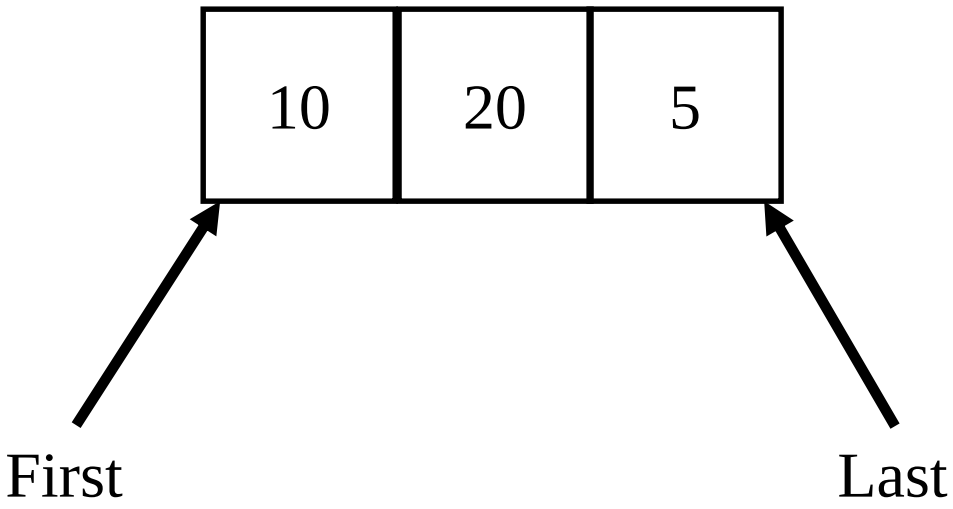
InsertFirst(5)



InsertLast(20)



InsertLast(DeleteFirst())



덱(2)

- 덱의 추상 데이타 타입(ADT)

createDeque() ::= create an empty deque;
insertFirst(Deque,e) ::= insert new element e at the beginning of Deque;
insertLast(Deque,e) ::= insert new element e at the end of Deque;
isEmpty(Deque) ::= **if** Deque is empty **then** return true
else return false;
deleteFirst(Deque) ::= **if** isEmpty(Deque) **then return** null
else remove and **return** the first element of Deque;
deleteLast(Deque) ::= **if** isEmpty(Deque) **then return** null
else remove and **return** the last element of Deque;
removeFirst(Deque) ::= **if** isEmpty(Deque) **then return** null
else remove the first element of Deque;
removeLast(Deque) ::= **if** isEmpty(Deque) **then return** null
else remove the last element of Deque;
peekLast(Deque) ::= **return** the last element of Deque;
peekFirst(Deque) ::= **return** the first element of Deque;

Deque덱(3)

- Series of operation in a deque공백 덱에 대한 일련의 연산 수행

operation연산	덱(Deque)
insertFirst(Deque,3)	(3)
insertFirst(Deque,5)	(5, 3)
deleteFirst(Deque)	(3)
insertLast(Deque,7)	(3, 7)
deleteFirst(Deque)	(7)
deleteLast(Deque)	()
insertFirst(Deque,9)	(9)
insertLast(Deque,7)	(9, 7)
insertFirst(Deque,3)	(3, 9, 7)
insertLast(Deque,5)	(3, 9, 7, 5)
deleteLast(Deque)	(3, 9, 7)
deleteFirst(Deque)	(9, 7)

Operation	Output	D
insertFirst(3)	-	(3)
insertFirst(5)	-	(5,3)
deleteFirst()	5	(3)
insertLast(7)	-	(3,7)
deleteFirst()	3	(7)
deleteLast()	7	()
deleteFirst()	“error”	()
isEmpty()	true	()

덱(4)

- Deque operations correspond to opeartions in stack and queue스택과 큐 ADT 연산에 대응하는 덱의 연산
  - ✓ Operations corresponds to stack's스택 ADT 연산에 대응하는 연산

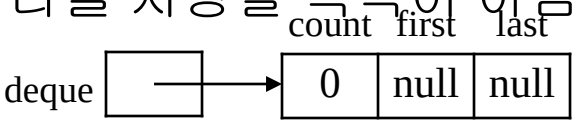
stack 스택 연산	deque덱 연산
createStack()	createDeque()
push(S,e)	insertLast(Deque,e)
isEmpty(S)	isEmpty(Deque)
pop(S)	deleteLast(Deque)

  - ✓ 큐 ADT 연산에 대응하는 연산

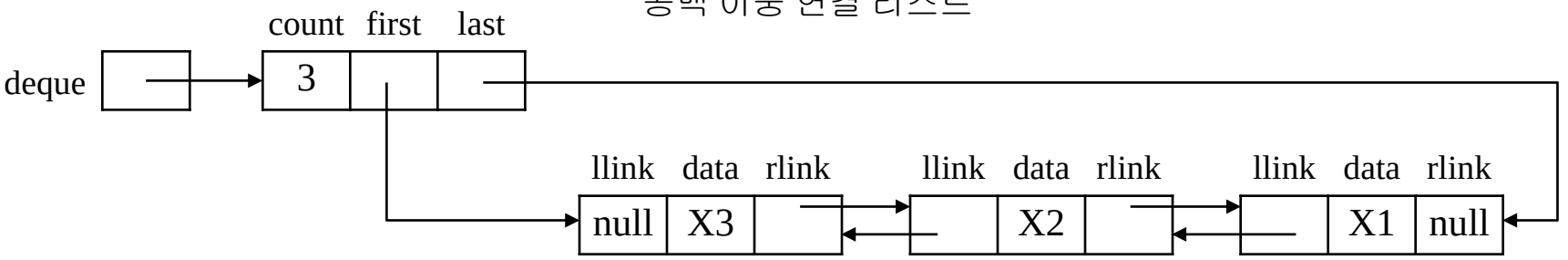
queue 큐 연산	de덱 연산
createQ()	createDeque()
enqueue(Q,e)	insertLast(Deque,e)
isEmpty(Q)	isEmpty(Deque)
dequeue(Q)	deleteFirst(Deque)
remove(Q)	removeFirst(Deque)

deque덱(5)

- Implementation of deque덱의 구현
  - ✓ Singly linked list(단일연결리스트)
    - adv : 리스트의 마지막 노드를 가리키는 포인터를 이용
    - disadv : deletion of last node requires O(n)
  - ✓ doubley linked list(이중연결리스트)
    - adv : insertion/deletion time of each end node requires O(1)
    - head 노드, tail 노드 사용
      - 리스트의 첫 번째 노드와 마지막 노드만을 가리키는 노드
      - 다른 데이타를 저장할 목적이 아님

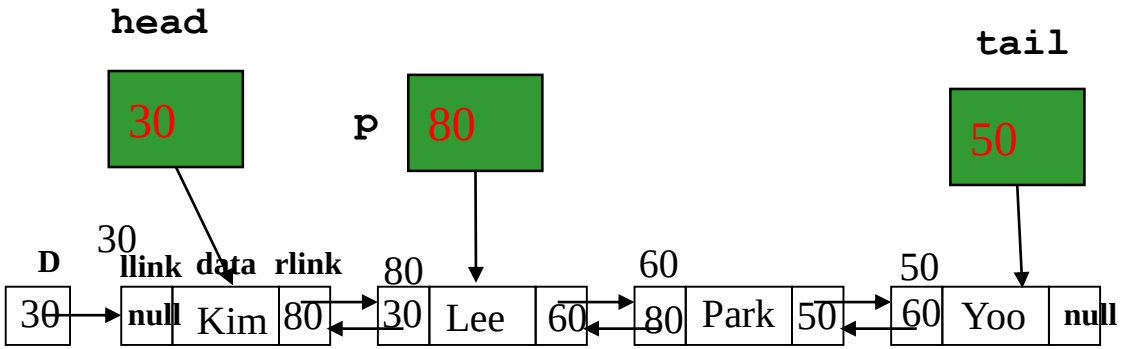


공백 이중 연결 리스트



2개의 원소를 포함한 이중 연결 리스트

조별과제



- 이중연결리스트를 이번 주 일요일까지 완성하여 업로드하라. 이때 포인터 head, tail을 사용하여 구현하라.

Deque(6)

- Implementation of Deque as doubly linked list(1)

```
public class DoubleListNode {
   Object data;
   DoubleListNode rlink, llink;
}

public class Deque {   // 이중 연결 리스트로 구현
   private DoubleListNode head;
   private DoubleListNode tail;
   private int count;

   public Deque() {
      head = null;
      tail = null;
      count = 0;
   }

   public boolean isEmpty() {
     return (count == 0);
   }

   public boolean isEmpty() {
     return (count == 0);
   }
```

덱(7)

- 이중 연결 리스트를 이용한 Deque 구현(2)

```
public void insertFirst(Object value) {
        // 연결 덱에 첫 번째 원소를 삽입
    DoubleListNode newNode;
    newNode= new DoubleListNode();
    newNode.data = value;
    if (count == 0) {  // 덱이 공백인 경우
       first = newNode;
       last = newNode;
       rlink = llink = NULL;
    }
    else {
        first.llink = newNode;
        newNode.rlink = first;
        newNode.llink = NULL;
        first = newNode;
    }
    count++;
}
```

```
public void insertLast(Object value) {
        // 연결 덱에 마지막 원소로 삽입
    DoubleListNode newNode;
     newNode = new DoubleListNode();
    newNode.data = value;
   if (count == 0) {    // 덱이 공백인 경우
       first = newNode;
       last = newNode;
       rlink = NULL;
       llink = NULL;
    }
    else {
       last.rlink = newNode;
        newNode.rlink = NULL;
        newNode.llink = last;
        last = newNode;
       count++;
    }
}
```

덱(8)

- 이중 연결 리스트를 이용한 Deque 구현(3)

```
public Object deleteFirst() {
  // 연결 덱에서 첫 번째 원소를 삭제하고 반환
    if (count == 0)
      return null;  // 연결 덱이 공백인 경우
    else {
      Object value = first.data;
      if (first.rlink = NULL) {
          // 원소가 1개인 경우
          first = NULL;
          last = NULL;
      }
      else {          // 원소가 2개 이상인 경우
          first = first.rlink;
          first.llink = NULL;
      }
      count--;
      return value;
    }
  }
```

```
public Object deleteLast() {
    // 연결 덱에서 마지막 원소를 삭제하고 반환
    if (count == 0)
      return null; // 연결 덱이 공백인 경우
    else {
      Object value = last.data;
      if (last.llink = NULL) {
          // 원소가 1개인 경우
          first = NULL;
          last = NULL;
      }
      else {            // 원소가 2개 이상인 경우
          last = last.llink;
          last.rlink = NULL;
      }
      count--;
      return value;
    }
  }
```

deq덱(9)

- 이중 연결 리스트를 이용한 Deque 구현(4)

```
    public void removeFirst() {
            ⋮
    }

    public void removeLast() {
            ⋮
    }

    public Object peekFirst() {
            ⋮
    }

    public Object peekLast() {
            ⋮
    }
} // end Deque class
```

덱(10)

- 덱을 이용한 Stack 구현(1)
  - ✓Depue 클래스를 이용해 스택을 구현한 DequeStack 클래스

```
public class DequeStack implements Stack {    // 덱을 스택을 이용해 구현
    private Deque d;    // Deque 타입의 참조 변수

    public DequeStack() {    // 생성자, 스택을 초기화
        d = new Deque();
    }

    public boolean isEmpty() {    // 스택이 공백인가를 검사
        return d.isEmpty();
    }

    public void push(Object x) {    // 스택에 원소 삽입
        d.insertLast(x);
    }
```

덱(11)

- 덱을 이용한 Stack 구현(2)

```
    public Object peek()  {     // 스택의 톱 원소를 검색
        if (isEmpty()) return null;     // 스택이 공백인 경우
        else return d.peekLast();
    }

    public Object pop()  {     // 스택의 톱 원소를 삭제하고 반환
        if (isEmpty()) return null;     // 스택이 공백인 경우
        else return d.deleteLast();
    }

    public void remove()  {     // 스택의 톱 원소를 삭제
        if (isEmpty()) return;
        else d.removeLast();
    }

}     // end DequeStack class
```