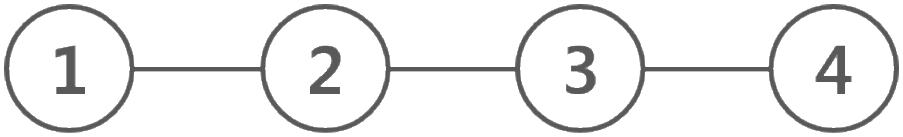
선형 자료구조

선형 자료구조

❖ 선형 자료구조

✓ 자료를 구성하는 데이터를 순차적으로 1:1로 나열시킨 자료구조



✓ 종류

- 정적 배열
- Vector(Array List)
- Linked List
- Stack
- Queue
- Deque

선형 자료구조

- List
  - ✓ 동일한 자료형의 데이터를 순서대로 저장하는 자료구조
  - ✓ 순서대로 = 차례대로 = 한 줄로 = 선형 구조
  - ✓ 다른 자료구조를 구현하는 수단으로도 사용
    - 트리(tree), 그래프(graph) 등을 구현하는데 사용
  - ✓ 종류
    - 순차 자료구조: Array(배열) & Vector(Array List), Stack, Queue, Deque
    - 연결 자료구조: Linked List

구분	순차 자료구조	연결 자료구조
메모리 저장 방식	메모리의 저장 시작 위치부터 빈자리 없이 자료를 순서대로 연속하여 저장한다. 논리적인 순서와 물리적인 순서가 일치하는 구현 방식이다.	메모리에 저장된 물리적 위치나 물리적 순서와 상관없이, 링크에 의해 논리적인 순서를 표현하는 구현 방식이다.
연산 특징	삽입·삭제 연산을 해도 빈자리 없이 자료가 순서대로 연속하여 저장된다. 변경된 논리적인 순서와 저장된 물리적인 순서가 일치한다.	삽입·삭제 연산을 하여 논리적인 순서가 변경되어도, 링크 정보만 변경되고 물리적 순서는 변경되지 않는다.

Array

- 배열
  - ✓ 특징
    - 크기가 고정 – 한 번 생성하면 크기를 변경할 수 없음
    - 빈 공간없이 데이터를 저장하기 때문에 메모리 효율이 좋음
    - 데이터를 중간에 삽입하거나 삭제할 수 없음
    - 삽입이나 삭제 작업을 해야 하는 경우에는 다른 배열을 생성하고 데이터를 복사하는 작업을 통해서 수행
  - ✓ 언어별 특징
    - C 에서는 배열이 정적 자료형으로 스택에 생성됨
    - Java 에서는 배열이 동적 자료형으로 Heap 에 생성됨
    - Python에서는 배열이 없고 list가 제공되는데 numpy 라이브러리 안에는 ndarray 라는 배열이 있음

Array

- 배열
  - ✓ 생성
    - C
      - 자료형 배열이름[데이터개수]
      - 자료형 배열이름 [] = {데이터 나열}
    - Java
      - 자료형 [] 배열명 = {초기 데이터}
      - 자료형 [] 배열명 = new 자료형[크기]

Array

- 배열
  - ✓ 데이터 개수 파악
    - C: sizeof(배열이름) / sizeof(배열이름[0])
    - Java: 배열이름.length
  - ✓ 데이터 접근
    - 배열명[인덱스]: 인덱스는 0 부터 (데이터 개수 – 1) 까지
    - 전체 순회 – Java는 빠른 열거 사용 가능

Array

❖ 배열의 생성과 접근 - Java

```
public class ArrayCreate {

    public static void main(String[] args) {
      //배열의 생성
      int [] intAr = new int[3];
      intAr[0] = 100;
      intAr[1] = 200;
      intAr[2] = 300;

      String [] strAr = {"컴퓨터개론", "운영체제", "프로그래밍언어", "자료구조", "알고리즘"};
```

Array

❖ 배열의 생성과 접근 - Java

```
        System.out.println("intAr 의 데이터 개수:" + intAr.length);

        //인덱스를 이용한 모든 데이터 접근
        int len = strAr.length;
        for(int i=0; i<len; i++) {
                System.out.print(strAr[i] + "₩t");
        }
        System.out.println();

        //빠른 열거를 이용한 접근
        for(int temp : intAr) {
                System.out.print(temp + "₩t");
        }

        System.out.println();
    }
}
```

Array

- 배열의 생성과 접근 - Python

```
n_ar = []
n_ar.append(100)
n_ar.append(300)
n_ar.append(200)

#초기 데이터를 가지고 생성
str_ar = ["컴퓨터개론", "운영체제 및 네트워크", "프로그래밍언어", "자료구조", "알고리즘"]

#데이터 개수
print("n_ar 의 데이터 개수:", len(n_ar))
print()

#배열의 데이터 순회
i = 0
while i < 3:
    print(n_ar[i])
    i = i + 1
print()

for temp in str_ar:
    print(temp)
```

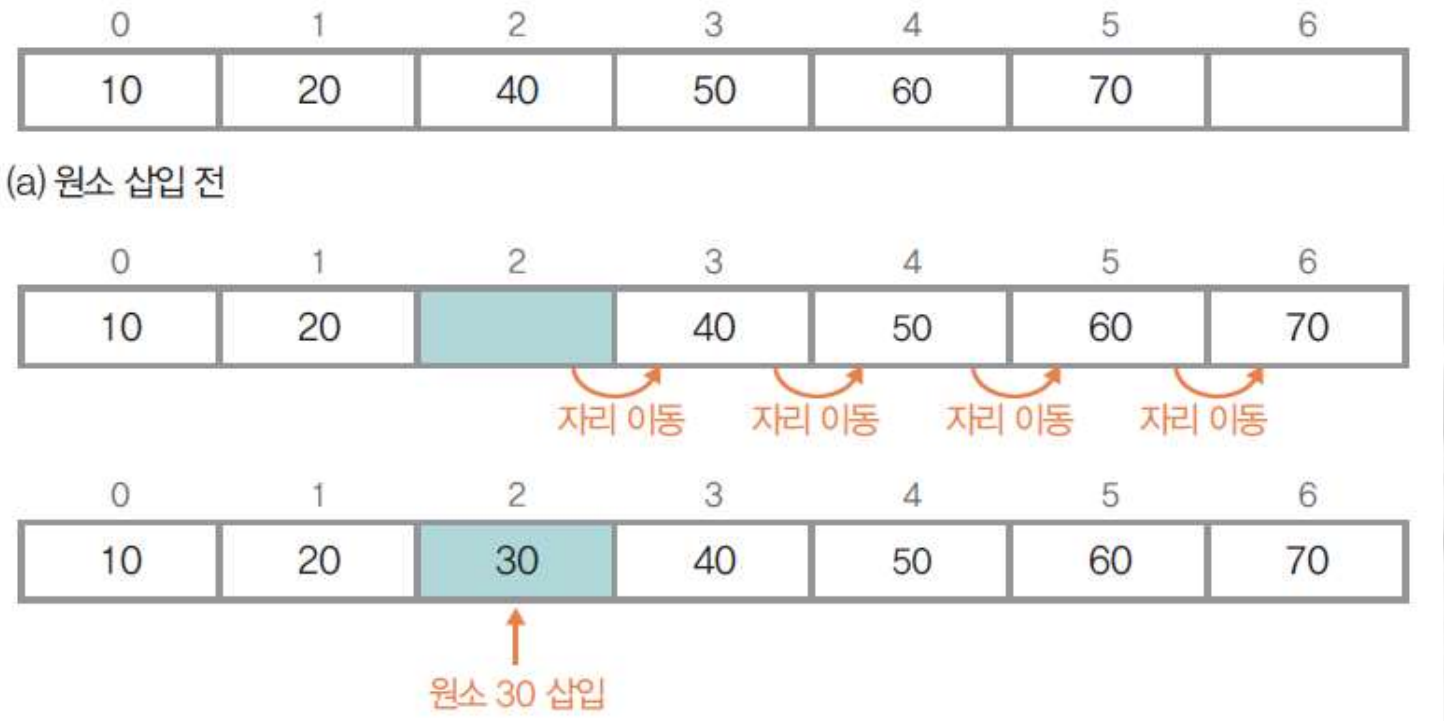
Array

- ❖ 배열의 데이터 변경 작업
  - ✓ 배열의 마지막에 데이터 추가
    - 배열은 저장할 수 있는 데이터의 크기를 늘릴 수 없음
    - 데이터를 마지막에 추가하고자 하는 경우에는 기존 배열보다 데이터를 1개 더 저장할 수 있는 배열을 생성하고 데이터를 복사한 후 마지막 위치에 새로운 데이터를 추가
    - 10,20,30,40 이 저장된 배열의 마지막에 50 추가하기

	10	**20**	**30**	**40**	
1번 – 새로운 배열 생성					
2번 – 기존 데이터 복사	**10**	**20**	**30**	**40**	
3번 – 데이터를 추가	**10**	**20**	**30**	**40**	**50**

Array

- 배열의 데이터 변경 작업
  - ✓ 배열의 중간에 데이터 추가
    - 데이터를 중간에 추가하고자 하는 경우에는 기존 배열보다 데이터를 1개 더 저장할 수 있는 배열을 생성하고 삽입하고자 하는 위치 전까지의 데이터를 복사한 후 삽입할 데이터를 삽입한 후 나머지 데이터를 복사
    - 10,20,40,50,60,70 이 저장된 배열의 20 다음에 30을 삽입하기



Array

- ❖ 배열의 데이터 변경 작업
  - ✓ 배열의 데이터 삭제
    - 중간에서 원소가 삭제되면, 그 이후의 원소들은 한 자리씩 자리를 앞으로 이동하여 물리적 순서를 논리적 순서와 일치시킴
    - 10, 20, 30, 40, 50, 60, 70 배열에서 데이터 삭제

Array

- 배열의 데이터 변경 작업
  - ✓ 배열 작업 클래스

```
public class Array {
    //데이터를 마지막에 삽입하는 Method
    public static Object[] append(Object [] ar, Object data) {
            Object [] result = new Object[ar.length+1];

            int i = 0;
            for(Object obj : ar) {
                    result[i] = obj;
                    i++;
            }
            result[result.length-1] = data;

            return result;
    }
```

Array

- 배열의 데이터 삽입과 삭제
  - ✓ 배열 작업 클래스

```
//데이터를 중간에 삽입하는 Method
public static Object[] insert(Object[] ar, int idx,  Object data) {
        if(idx < 0 || idx > ar.length) {
                System.out.println("데이터를 삽입할 수 없는 위치입니다.");
                return ar;
        }
        Object [] result = new Object[ar.length+1];
        int i = 0;
        for(i=0; i<idx; i++) { result[i] = ar[i];
        }
        result[idx] = data;
        for(i=idx; i<ar.length; i++) { result[i+1] = ar[i];
        }
        return result;
}
```

Array

- 배열의 데이터 삽입과 삭제
  - ✓ 배열 작업 클래스

```
//데이터를 삭제하는 Method
public static Object[] delete(Object [] ar, int idx) {
        if(idx < 0 || idx > ar.length - 1) {
                System.out.println("데이터를 삭제할 수 없는 위치입니다.");
                return ar;
        }
        Object [] result = new Object[ar.length-1];
        int i = 0;
        for(i=0; i<idx; i++) { result[i] = ar[i];
        }
        for(i=idx+1; i<ar.length; i++) { result[i-1] = ar[i];
        }
        return result;
    }
}
```

Array

- 배열의 데이터 삽입과 삭제
  - ✓ 배열 실행 클래스

```
public class ArrayMain {

    public static void main(String[] args) {
        Object[] intAr = { 100, 200, 400 };
        for (Object temp : intAr) {
            System.out.print(temp + "₩t");
        }
        System.out.println();

        // 200 다음의 위치에 300을 추가
        intAr = Array.insert(intAr, 2, 300);
        for (Object temp : intAr) {
            System.out.print(temp + "₩t");
        }
        System.out.println();
```

Array

- 배열의 데이터 삽입과 삭제
  - ✓ 배열 실행 클래스

```
                // 마지막 위치에 500을 추가
                intAr = Array.append(intAr, 500);
                for (Object temp : intAr) {
                        System.out.print(temp + "₩t");
                }
                System.out.println();

                // 200을 제거
                intAr = Array.delete(intAr, 1);
                for (Object temp : intAr) {
                        System.out.print(temp + "₩t");
                }
                System.out.println();
        }
    }
```

ArrayList

- ArrayList(Vector)
  - ✓ 자바에는 배열과 유사한 형태로 동작하지만 삽입과 삭제를 고려해서 여분의 공간을 확보하고 있는 ava.util.ArrayList라는 자료구조 클래스를 제공
  - ✓ 이전에는 java.util.Vector 라는 클래스를 사용
  - ✓ 생성

    ArrayList<요소의 자료형> 변수명 = new ArrayList<요소의 자료형>
  - ✓ Method
    - boolean add(E e): 마지막에 데이터를 추가
    - void add(int index, E element): index 위치에 데이터를 추가
    - E get(int index): index 위치의 데이터 가져오기
    - E remove(int index): index 위치의 데이터 삭제
    - int size(): 데이터 개수 리턴
    - 빠른 열거를 이용한 접근 가능

ArrayList

❖ ArrayList의 데이터 삽입과 삭제

```
import java.util.ArrayList;

public class ArrayListMain {

    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<String>();
        //데이터를 마지막에 추가
        list.add("서울특별시");
        list.add("인천광역시");

        //데이터를 중간에 추가
        list.add(1, "부산광역시");

        //데이터 개수 확인
        System.out.println(list.size());
        //데이터 확인
        System.out.println(list);
```

ArrayList

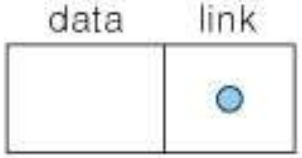
❖ ArrayList의 데이터 삽입과 삭제

```
        //데이터 제거
        list.remove(2);

        //전체 데이터 접근
        for(String imsi : list) {
                System.out.println(imsi);
        }
    }
}
```

```
3
[서울특별시, 부산광역시, 인천광역시]
서울특별시
부산광역시
```

LinkedList

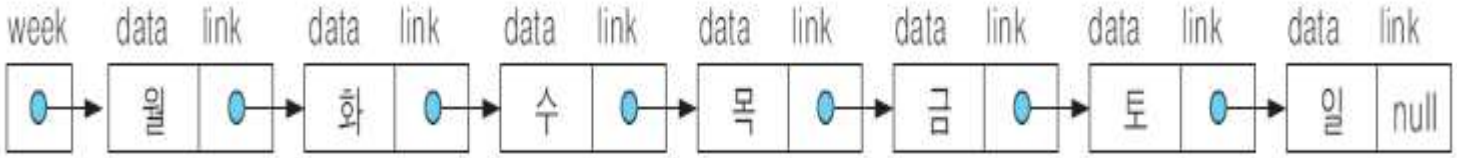
- Linked List
  - ✓ Data 와 Link로 구성된 노드들로 구성된 List
  - ✓ 배열이나 ArrayList는 Data 만으로 구성된 List
  - ✓ Linked List는 Data에 실제 자료를 저장하고 Link에 다음 데이터의 위치를 저장하는 구조로 데이터들이 물리적으로 연속된 공간에 저장될 필요가 없는 자료구조
  - ✓ 노드
    - 연결 자료구조에서 하나의 원소를 표현하기 위한 단위 구조



    - 데이터 필드(data field)
      - 원소의 값을 저장
      - 저장할 원소의 형태에 따라서 하나 이상의 필드로 구성
    - 링크 필드(link field)
      - 다음 노드의 주소를 저장
      - 포인터 변수를 사용하여 주소 값을 저장
      - 마지막 노드는 다음 데이터가 더 이상 없으므로 NULL

LinkedList

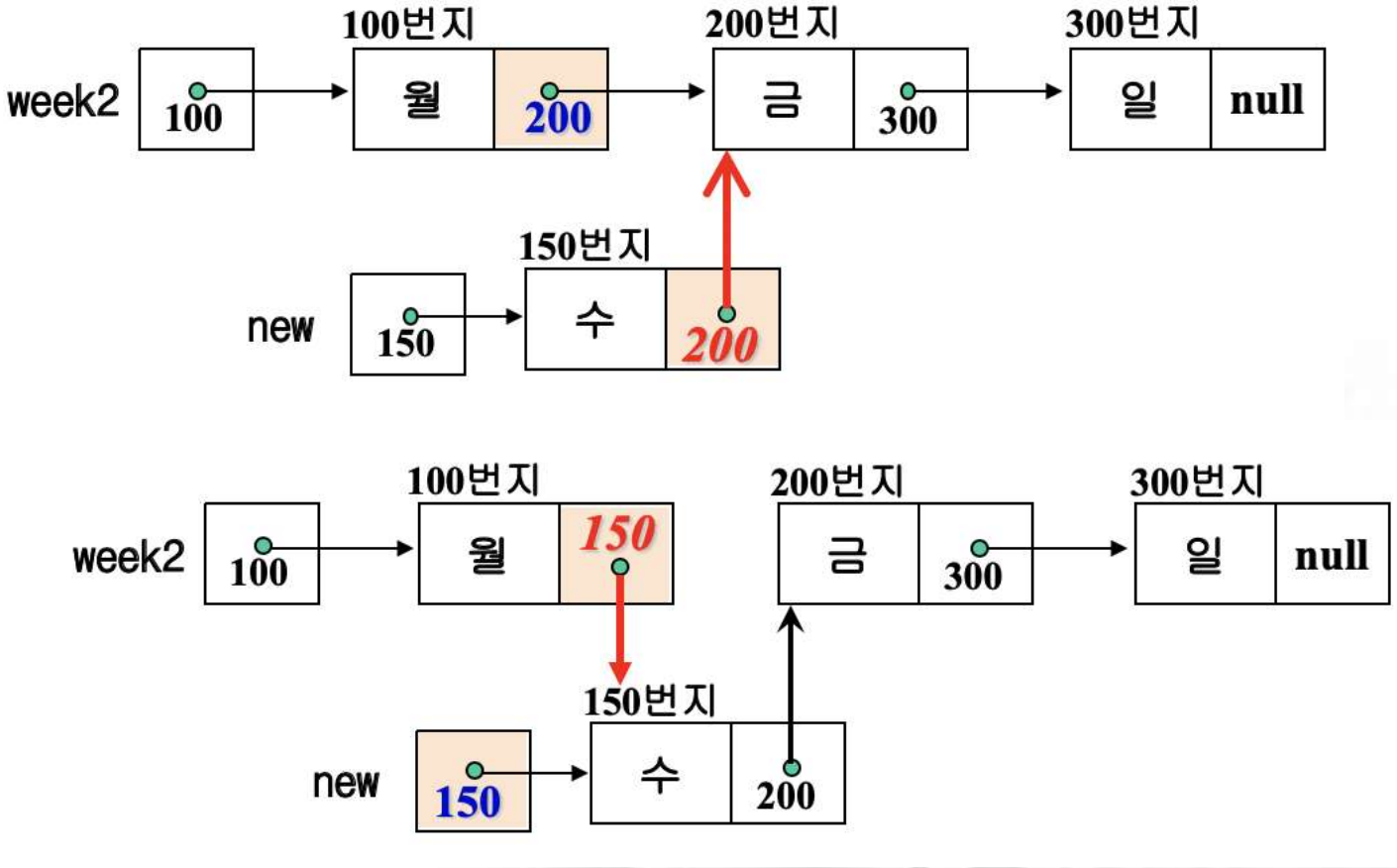
- Linked List
  - ✓ 저장 형태



- 순서 상으로는 연결된 것처럼 보임
- 물리적으로는 떨어져 있을 수 있음
- Link에 의해 논리적으로 연결되어 있음
- 물리적으로 연속될 필요가 없기 때문에 새로운 원소를 추가할 경우 동적으로 원소를 생성하고 Link를 이용해서 연결만 해주면 됨

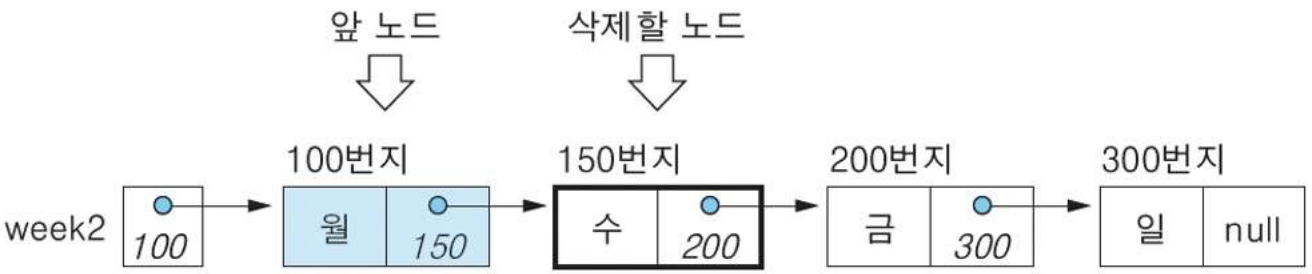
LinkedList

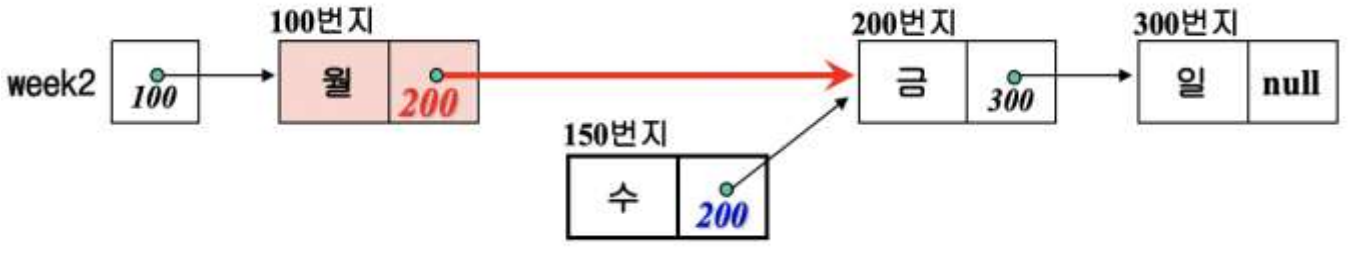
- Linked List
  - ✓ 데이터의 삽입: 새로운 노드를 만들어서 Link만 수정



LinkedList

- Linked List
  - ✓ 데이터의 삭제: 삭제할 데이터의 앞 요소를 찾아서 링크를 수정





LinkedList

- Linked List
  - ✓ 배열에 비교하여 가지는 장점
    - 새로운 원소를 중간에 삽입하거나 삭제하는 경우 원소의 이동 연산이 불필요하기 때문에 원소의 추가/삭제가 빈번히 발생한 경우라면 배열보다는 연결 리스트로 구현하는 것이 바람직
    - 배열은 연속적인 메모리 공간이 필요하므로 생성할 때 원소의 개수를 지정해야 하지만 연결 리스트는 원소의 개수 지정이 불필요
  - ✓ 배열과 비교하여 가지는 단점
    - 구현의 어려움 – 동적인 메모리 할당을 해야하기 때문에 구현의 비용이 높음
    - 메모리 관리와 관련하여 메모리 누수(Leak) 발생 가능성이 높음
    - 데이터를 탐색하는 비용이 높음 – 배열은 데이터를 연속적으로 저장했기 때문에 위치 인덱스를 이용해서 원하는 원소에 대한 접근이 한 번에 가능하지만 연결 리스트는 노드 사이에 연결된 Link를 통해 첫번째 원소부터 시작해서 위치-1 만큼 이동해야 하기 때문에 탐색 속도가 느림
  - ✓ 종류
    - Single Linked List: 하나의 link 만을 가지고 다음 데이터를 가리키는 형태
    - Circular Linked List: 마지막 노드의 link에 첫번째 데이터의 위치를 저장시키는 형태
    - Double Linked List: 2개의 link를 가지고 다음 데이터와 이전 데이터의 위치를 저장시키는 형태

LinkedList

- Single Linked List

```
//하나의 노드를 나타내기 위한 클래스
class Node {
    int data;
    Node next;
}
```

LinkedList

- Single Linked List

```
public class SingleLinkedList {
    //출발점을 만들기 위한 노드
    private Node head;

    //생성자 - head에 노드 할당
    public SingleLinkedList()
    {
       head = new Node();
    }
```

LinkedList

- Single Linked List

```
//데이터를 마지막에 추가하는 Method
public void addElement(int data)
{
   Node temp = head;
   while(true) {
                if(temp.next == null) {
                        break;
                }
                temp = temp.next;
   }
   Node node = new Node();
   node.data = data;
   temp.next = node;
}
```

LinkedList

- Single Linked List

```
//pos에 해당하는 데이터를 찾아오는 Method
public Node getElement(int pos) {
   if(size() == 0) {
                System.out.println("데이터가 없어서 데이터를 가져올 수 없습니다.");
                return null;
   }
   if(pos < 0 || pos >= size()) {
                System.out.println("데이터를 가져올 수 없는 위치입니다.");
                return null;
   }
   int i=0;
   Node node = head.next;
   while(i < pos) {
                i=i+1;
                node = node.next;
   }
   return node;
}
```

LinkedList

- Single Linked List

```
//list 전체를 순회하는 Method
public void traversal() {
   Node temp = head.next;
   if(temp == null) {
                System.out.println("리스트에 데이터가 없습니다.");
   }else {
                while(true) {
                             System.out.print(temp.data);
                             temp = temp.next;
                             if(temp == null) {
                                          break;
                             }
                             System.out.print("->");
                }
   }
}
```

LinkedList

- Single Linked List

```
//list의 데이터 개수를 리턴하는 Method
public int size()
{
   Node temp = head;
   int cnt = 0;
   while(true) {
                if(temp.next == null) {
                        break;
                }
                temp = temp.next;
                cnt = cnt + 1;
   }
   return cnt;
}
```

LinkedList

- Single Linked List

```
//위치와 데이터를 받아서 데이터를 중간에도 삽입할 수 있는 Method
  // 위치와 데이터를 받아서 데이터를 중간에도 삽입할 수 있는 Method
public void insertElement(int pos, int data) {

   if (pos < 0 || pos > size()) {
                System.out.println("삽입할 수 없는 위치입니다.");
                return;
   }

   Node node = new Node();
   node.data = data;

   Node temp = getElement(pos - 1);
   node.next = temp.next;
   temp.next = node;
}
```

LinkedList

- Single Linked List

```
    // pos 위치에 해당하는 데이터를 삭제하는 Method
public void removeElement(int pos) {
    if (size() == 0) {
                System.out.println("데이터가 없어서 데이터를 삭제할 수 없습니다.");
                return;
    }
    if (pos < 0 || pos >= size()) {
                System.out.println("삭제할 수 없는 위치입니다.");
                return;
    }
    Node node = null;
    if (pos == 0) {
                node = head;
    } else {
                node = getElement(pos - 1);
    }

    Node temp = node.next;
    node.next = temp.next;
    temp = null;
}
```

LinkedList

- Single Linked List

```
// pos 위치에 해당하는 데이터를 수정하는 Method
public void updateElement(int pos, int data) {
   if (size() == 0) {
                System.out.println("데이터가 없어서 데이터를 수정할 수 없습니다.");
                return;
   }
   if (pos < 0 || pos >= size()) {
                System.out.println("수정할 수 없는 위치입니다.");
                return;
   }

   Node node = getElement(pos);
   node.data = data;

}
```

LinkedList

- Single Linked List

```
        //list 전체를 초기화하는 Method
        public void clearElement()
        {
           if(size() == 0) {
                        System.out.println("데이터가 없어서 데이터를 삭제할 필요가 없습니다.");
                        return;
           }
           Node temp = head;
           while(temp.next != null) {
                        Node node = temp.next;
                        temp.next = temp.next.next;
                        node = null;
           }
        }
    }
```

LinkedList

- Single Linked List

```
public class SingleLinkedListMain {

    public static void main(String[] args) {
        SingleLinkedList list = new SingleLinkedList();
        list.addElement(200);
        list.addElement(300);
        // 두번째 데이터 가져오기
        System.out.println(list.getElement(1).data);

        // 전체 순회 : 200->300
        list.traversal();
        System.out.println();

        // 1번째 위치에 데이터 추가
        list.insertElement(1, 250);
        // 200->250->300
        list.traversal();
        System.out.println();
```

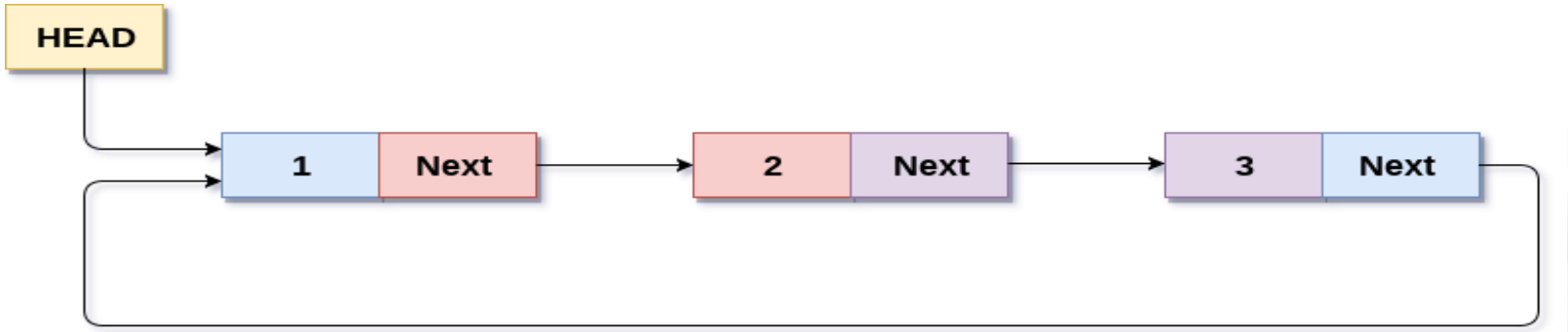
LinkedList

- Single Linked List

```
            // 1번째 데이터 삭제
            list.removeElement(1);
            list.traversal();
            System.out.println();

            // 1번째 데이터 수정
            list.updateElement(1, 230);
            list.traversal();
            System.out.println();

            // 전체 데이터 삭제
            list.clearElement();
            list.traversal();
        }
    }
```

LinkedList

- Circular Linked List(환형 연결 리스트)
  - ✓ 리스트의 마지막 노드가 리스트의 첫번째 노드와 연결된 단순 연결 리스트
  - ✓ 단순 연결 리스트와 동일한 구조를 지니지만 리스트의 마지막 노드가 첫번째 노드와 연결되어 순환 구조를 지님
  - ✓ 단순 연결 리스트는 이전 노드에 접근하려면 HEAD로 이동한 후 접근해야 하지만 환형 연결 리스트는 계속 이동하다 보면 이전 노드에 접근이 가능
  - ✓ 이동하다가 다음 노드의 위치가 현재 노드이면 이전 노드가 됨



LinkedList

- Circular Linked List(환형 연결 리스트)
  - ✓ 데이터 개수 세기
    - head의 next 가 null 이면 데이터 없음
    - head 의 next 가 null 이 아니면 데이터 개수를 1씩 증가시키면서 다음 노드를 가져오는데 가져오는 노드의 다음 노드가 head의 next이면 중단
    - 자신의 다음 노드가 head 의 next(시작 노드)이면 마지막 노드
  - ✓ 마지막 위치에 노드 추가
    - 데이터가 없다면 삽입할 노드를 head 의 next로 추가하고 삽입할 노드의 next 도 자신이 되도록 설정
    - 데이터가 있다면 노드의 next가 head의 next 가 될 때 까지 이동한 후(마지막 노드) 이 노드의 next가 새로 삽입할 노드가 되도록 하고 삽입할 노드의 next가 head의 next가 되도록 설정

LinkedList

- Circular Linked List(환형 연결 리스트)
  - ✓ 데이터 가져오기: 가져올 위치만큼 노드를 이동시켜서 리턴
  - ✓ 전체 순회: 데이터를 이동시키면서 다음 노드가 시작 노드(head.next)가 될 때 까지 가져오기
  - ✓ 이전 노드 찾기: 현재 위치에서 size()-1 만큼 다음 노드로 진행해서 찾아옴
  - ✓ 중간에 추가하기
    - 삽입할 위치가 데이터의 개수와 같다면 마지막 위치에 삽입하는 Method 호출
    - 첫번째 위치에 추가하는 경우 기존 head.next를 임시변수에 보관하고 head.next 에 추가할 노드를 설정하고 마지막 노드로 이동한 후 마지막 노드의 next를 추가할 노드로 설정
    - 나머지 경우에는 삽입할 위치의 이전 데이터까지 이동한 후 삽입할 노드의 next로 찾은 데이터의 next를 설정한 후 찾은 데이터의 next를 삽입할 노드로 설정
  - ✓ 수정하기: 수정할 위치의 데이터를 찾아서 data를 수정

LinkedList

- Circular Linked List(환형 연결 리스트)
  - ✓ 데이터 삭제하기
    - 데이터가 개수가 1개이고 0번째 데이터를 삭제하는 경우에는 첫번째 데이터를 삭제하고 head.next를 null로 설정
    - 그 이외 0번째 데이터를 삭제하는 경우에는 마지막 데이터를 찾아서 마지막의 next로 설정하고 head.next도 그 데이터로 설정한 후 삭제
    - 마지막 데이터를 삭제하는 경우에는 마지막 이전 데이터를 찾아서 첫번째 데이터를 가리키도록 하고 삭제
    - 첫번째나 마지막 데이터가 아닌 경우의 삭제는 삭제하기 이전 데이터를 찾고 삭제하기 이전 데이터의 next를 삭제할 데이터의 next로 설정하고 데이터 삭제
  - ✓ 전체 삭제 - 초기화
    - 데이터의 개수가 0개가 될 때 까지 첫번째 데이터를 삭제

LinkedList

- Circular Linked List(환형 연결 리스트)

```
//하나의 노드를 나타내기 위한 클래스
class Node {
    int data;
    Node next;
}
```

LinkedList

- Circular Linked List(환형 연결 리스트)

```
class CircularLinkedList {
    // 출발점을 만들기 위한 노드
    private Node head;

    // 생성자 - head에 노드 할당
    public CircularLinkedList() {
       head = new Node();
    }
```

LinkedList

- Circular Linked List(환형 연결 리스트)

```
// list의 데이터 개수를 리턴하는 Method
public int size() {
   Node temp = head;
   int cnt = 0;
   if (temp.next == null) {
                return cnt;
   }
   while (true) {
                temp = temp.next;
                cnt = cnt + 1;
                if (temp.next == head.next) {
                             break;
                }
   }
   return cnt;
}
```

LinkedList

- Circular Linked List(환형 연결 리스트)

```
// 데이터를 마지막에 추가하는 Method
public void addElement(int data) {
   Node temp = head;
   Node node = new Node();
   node.data = data;

   // 데이터가 없는 경우
   if (temp.next == null) {
                head.next = node;
                node.next = node;
   }
   // 데이터가 존재하는 경우에는 데이터의 마지막까지 이동시킨 후 마지막 노드 다음이
새로운 노드가 되고
   // 새로운 노드 의 다음은 첫번째 노드
   else {
                do {
                        temp = temp.next;
                } while (temp.next != head.next);
                temp.next = node;
                node.next = head.next;
   }
}
```

LinkedList

- Circular Linked List(환형 연결 리스트)

```
// pos에 해당하는 데이터를 찾아오는 Method
public Node getElement(int pos) {
   if (size() == 0) {
                System.out.println("데이터가 없어서 데이터를 가져올 수 없습니다.");
                return null;
   }
   if (pos < 0 || pos >= size()) {
                System.out.println("데이터를 가져올 수 없는 위치입니다.");
                return null;
   }
   int i = 0;
   Node node = head.next;
   while (i < pos) {
                i = i + 1;
                node = node.next;
   }
   return node;
}
```

LinkedList

- Circular Linked List(환형 연결 리스트)

```
// list 전체를 순회하는 Method
public void traversal() {
   Node temp = head.next;
   if (temp == null) {
                System.out.println("리스트에 데이터가 없습니다.");
   } else {
                while (true) {
                             System.out.print(temp.data);
                             temp = temp.next;
                             if (temp == head.next) {
                                          break;
                             }
                             System.out.print("->");
                }
                System.out.println();
   }
}
```

LinkedList

- Circular Linked List(환형 연결 리스트)

```
//현재 노드의 이전 위치에 해당하는 데이터를 리턴하는 Method
public Node prevElement(Node node) {
        if (size() == 0) {
                System.out.println("데이터가 없어서 데이터를 가져올 수
없습니다.");
                return null;
        }
        if (size() == 1) {
                System.out.println("데이터가 1개라서 이전 노드가 자기
자신입니다.");
                return node;
        }

        Node temp = node;
        int i=0;
        while(i < size()-1) {
                i = i + 1;
                temp = temp.next;
        }
        return temp;
}
```

LinkedList

- Circular Linked List(환형 연결 리스트)

```
// 위치와 데이터를 받아서 데이터를 중간에도 삽입할 수 있는 Method
public void insertElement(int pos, int data) {

  if (pos < 0 || pos > size()) {
            System.out.println("삽입할 수 없는 위치입니다.");
            return;
  }
  // 마지막에 삽입하고자 하는 경우에는 addElement() 호출
  else if (pos == size()) {
            addElement(data);
            return;
  }
```

LinkedList

❖ Circular Linked List(환형 연결 리스트)

```
Node temp = head;
Node node = new Node();
node.data = data;
int i = 0;
// 첫번째에 노드를 추가하고자 하는 경우 마지막 노드가 추가하는 노드를 가리키는
작업을 수행하고 head의 다음에 추가
if (pos == 0) {
        temp = getElement(size() - 1);
        temp.next = node;
        node.next = head.next;
        head.next = node;
        return;
}
// 첫번째가 아닌 경우 추가
i = 0;
while (i < pos) {
        temp = temp.next;
        i = i + 1;
}
node.next = temp.next;
temp.next = node;
}
```

LinkedList

❖ Circular Linked List(환형 연결 리스트)

```
// pos 위치에 해당하는 데이터를 수정하는 Method
public void updateElement(int pos, int data) {
   if (size() == 0) {
                System.out.println("데이터가 없어서 데이터를 수정할 수 없습니다.");
                return;
   }
   if (pos < 0 || pos >= size()) {
                System.out.println("수정할 수 없는 위치입니다.");
                return;
   }

   Node node = getElement(pos);
   node.data = data;
}
```

LinkedList

- Circular Linked List(환형 연결 리스트)

```
//pos 위치에 해당하는 데이터를 삭제하는 Method
public void removeElement(int pos) {
   if (size() == 0) {
                System.out.println("데이터가 없어서 데이터를 삭제할 수 없습니다.");
                return;
   }
   if (pos < 0 || pos >= size()) {
                System.out.println("삭제할 수 없는 위치입니다.");
                return;
   }

   if(size() == 1 && pos == 0) {
                Node temp = head.next;
                temp = null;
                head.next = null;
   }
```

LinkedList

❖ Circular Linked List(환형 연결 리스트)

```
        else if(pos == 0) {
                Node last = getElement(size()-1);
                Node node = head.next;
                last.next = node.next;
                head.next = node.next;
                node = null;
        }else if(pos == size()-1) {
                Node temp = getElement(size()-2);
                temp.next = null;
                temp.next = head.next;
        }else {
                Node temp = getElement(pos-1);
                Node node = temp.next;
                temp.next = node.next;
                node = null;
        }
    }
```

LinkedList

- Circular Linked List(환형 연결 리스트)

```
        // list 전체를 초기화하는 Method
        public void clearElement() {
           if (size() == 0) {
                        System.out.println("데이터가 없어서 데이터를 삭제할 필요가 없습니다.");
                        return;
           }
           while (size() != 0) {
                        removeElement(0);
           }
        }
    }
```

LinkedList

❖ Circular Linked List(환형 연결 리스트)

```
public class CircularLinkedListMain {

    public static void main(String[] args) {
        CircularLinkedList list = new CircularLinkedList();
        System.out.println(list.size());
        // 마지막에 데이터 삽입
        list.addElement(100);
        System.out.println(list.size());
        list.addElement(200);
        System.out.println(list.size());
        list.addElement(300);
        System.out.println(list.size());
        System.out.println();

        // 노드를 1개씩 가져오기
        System.out.println("0번째 노드:" + list.getElement(0).data);
        System.out.println("1번째 노드:" + list.getElement(1).data);
        System.out.println("2번째 노드:" + list.getElement(2).data);
        System.out.println();
```

LinkedList

- Circular Linked List(환형 연결 리스트)

```
// 전체 노드 순회
list.traversal();
System.out.println();


//1번째 노드의 이전 노드 찾기
Node node = list.getElement(1);
Node prevNode = list.prevElement(node);
System.out.println("1번의 이전 노드:" + prevNode.data);

// 중간에 노드 추가
list.insertElement(2, 270);
list.traversal();
System.out.println();

// 0번째와 데이터 수정
list.updateElement(0, 50);
list.traversal();
System.out.println();
```

LinkedList

❖ Circular Linked List(환형 연결 리스트)

```
            // 0번째와 1번째 데이터 삭제
            list.removeElement(0);
            list.traversal();
            System.out.println();

            list.removeElement(1);
            list.traversal();
            System.out.println();

            //전체 삭제
            list.clearElement();
            list.traversal();

        }
    }
```

LinkedList

- Double Linked List(이중 연결 리스트)

```
class Node {
    int data;
    Node next;
    Node prev;
}
```

LinkedList

- Double Linked List(이중 연결 리스트)
  - ✓ 하나의 노드가 이전 노드와 다음 노드를 가리키는 포인터를 2개 가지는 연결 리스트
  - ✓ 단순 연결 리스트와 환형 연결 리스트는 이전 노드를 접근하는 것이 어렵고 노드의 포인터가 소멸되면 리스트가 분할되는 현상이 발생
  - ✓ 양방향으로 접근이 가능해서 데이터의 접근은 편리하지만 추가적인 메모리 공간을 필요로 하는 단점이 있음
  - ✓ 데이터 개수 세기
    - head.next가 tail이면 데이터 없음
    - head.next가 tail이 아니면 next로 이동하면서 tail을 만날때까지 1씩 증가
  - ✓ 마지막에 데이터 추가
    - 데이터가 없는 경우에는 노드를 생성하고 head의 next를 노드로 설정하고 tail의 prev를 node로 설정하며 노드의 prev는 head 로 설정하고 노드의 next는 tail로 설정
    - 데이터가 존재하는 경우에는 노드를 생성하고 마지막 데이터를 찾은 후 마지막의 next를 노드로 설정하고 tail의 prev를 노드로 설정하고 노드의 prev를 마지막 데이터로 설정하고 노드의 next를 tail로 설정

LinkedList

- Double Linked List(이중 연결 리스트)
  - ✓ 이전 노드 가져오기: 현재 노드가 head가 아니라면 prev
  - ✓ 다음 노드 가져오기: 현재 노드가 tail이 아니라면 next
  - ✓ 데이터를 가져오기 – 양수는 앞에서부터 이동해서 가져오고 음수는 뒤에서부터 이동해서 가져오기
    - 양수인 경우는 head에서부터 next를 따라서 위치만큼 이동해서 가져오기
    - 음수인 경우는 tail에서부터 prev를 따라서 위치만큼 이동해서 가져오기
  - ✓ 데이터 순회
    - traversal은 head에서부터 next를 따라서 tail을 만날 때까지 하나씩 가져오기
    - reversetraversal은 tail에서부터 prev를 따라서 head를 만날 때까지 하나씩 가져오기

LinkedList

- Double Linked List(이중 연결 리스트)
  - ✓ 노드 중간에 삽입하기
    - 데이터가 없거나 마지막인 경우는 addElement()
    - 양수 위치에 삽입할 때는 삽입할 위치의 이전 노드까지 next를 이용해서 이동한 후 노드를 생성하고 이전 노드의 next에 노드를 설정하고 삽입할 위치의 이전 노드의 다음 노드의 prev에 삽입할 노드를 설정하고 삽입할 노드의 prev로 이전 노드를 설정하고 노드의 next로 이전 노드의 다음 노드를 설정
    - 음수 위치를 이용할 때는 삽입할 위치의 노드까지 prev를 이용해서 이동한 후 노드를 생성하고 찾은 노드의 next에 노드를 설정하고 삽입할 위치의 이전 노드의 다음 노드의 prev에 삽입할 노드를 설정하고 삽입할 노드의 prev로 이전 노드를 설정하고 노드의 next로 이전 노드의 다음 노드를 설정
  - ✓ 데이터 수정하기
    - 수정할 위치의 노드를 찾아서 data를 수정

LinkedList

- Double Linked List(이중 연결 리스트)
  - ✓ 노드 삭제하기
    - 삭제할 위치의 이전 노드와 삭제할 노드를 찾고 이전 노드의 next에 삭제할 노드의 next를 설정하고 삭제할 노드의 다음 노드의 prev에 삭제할 노드의 이전 노드를 설정하고 삭제할 노드에 null을 대입
  - ✓ 리스트 초기화: 데이터의 개수가 0이 될 때 까지 첫번째 데이터를 삭제

LinkedList

- Double Linked List(이중 연결 리스트)

```
class DoubleLinkedList {
  // 출발점을 만들기 위한 노드
  private Node head;
  // 종료점을 만들기 위한 노드
  private Node tail;

  // 생성자 - head에 노드 할당
  public DoubleLinkedList() {
            head = new Node();
            tail = new Node();
            head.next = tail;
            tail.prev = head;

  }
```

LinkedList

- Double Linked List(이중 연결 리스트)

```
// list의 데이터 개수를 리턴하는 Method
public int size() {
        Node temp = head;
        int cnt = 0;
        while (true) {
                temp = temp.next;
                if (temp == tail) {
                        break;
                }
                cnt = cnt + 1;
        }
        return cnt;
}
```

LinkedList

- Double Linked List(이중 연결 리스트)

```
// 데이터를 마지막에 추가하는 Method
public void addElement(int data) {
        Node temp = head;
        Node node = new Node();
        node.data = data;

        // 데이터가 없는 경우
        if (temp.next == tail) {
                head.next = node;
                node.prev = head;
        } else {
                Node last = tail.prev;
                last.next = node;
                node.prev = last;

        }
        tail.prev = node;
        node.next = tail;
}
```

LinkedList

- Double Linked List(이중 연결 리스트)

```
// pos에 해당하는 데이터를 찾아오는 Method
public Node getElement(int pos) {
        if (size() == 0) {
                System.out.println("데이터가 없어서 데이터를 가져올 수
없습니다.");
                return null;
        }
        Node temp = null;
        if (pos >= 0) {
                if (pos >= size()) {
                        System.out.println("데이터를 가져올 수 없는
위치입니다.");
                        return null;
                }
                int i = 0;
                temp = head;
                while (i <= pos) {
                        temp = temp.next;
                        i = i + 1;
                }
        }
```

LinkedList

❖ Double Linked List(이중 연결 리스트)

```
                else {
                        pos = pos * -1;
                        if (pos > size()) {
                                System.out.println("데이터를 가져올 수 없는
위치입니다.");
                                return null;
                        }
                        int i = 1;
                        temp = tail;
                        while (i <= pos) {
                                temp = temp.prev;
                                i = i + 1;
                        }
                }
                return temp;
        }
```

LinkedList

- Double Linked List(이중 연결 리스트)

```
// 이전 데이터를 찾아오는 Method
public Node prevElement(Node node) {
        Node temp = null;
        if (size() == 0) {
                System.out.println("데이터가 없어서 데이터를 가져올 수
없습니다.");
        } else if (node.prev == head) {
                System.out.println("첫번째 노드라서 다음 데이터가 없습니다.");
        } else {
                temp = node.prev;
        }
        return temp;
}
```

LinkedList

- Double Linked List(이중 연결 리스트)

```
// 다음 데이터를 찾아오는 Method
public Node nextElement(Node node) {
        Node temp = null;
        if (size() == 0) {
                System.out.println("데이터가 없어서 데이터를 가져올 수
없습니다.");
        } else if (node.next == tail) {
                System.out.println("마지막 노드라서 다음 데이터가 없습니다.");
        } else {
                temp = node.next;
        }
        return temp;
}
```

LinkedList

- Double Linked List(이중 연결 리스트)

```
// list 전체를 순회하는 Method
public void traversal() {
        Node temp = head;
        if (head.next == tail) {
                System.out.println("리스트에 데이터가 없습니다.");
        } else {
                while (true) {
                        temp = temp.next;
                        System.out.print(temp.data);
                        if (temp.next == tail) {
                                break;
                        }
                        System.out.print("->");
                }
                System.out.println();
        }
}
```

LinkedList

- Double Linked List(이중 연결 리스트)

```
// list 전체를 뒤에서 부터 순회하는 Method
public void reversetraversal() {
        Node temp = tail;
        if (head.next == tail) {
                System.out.println("리스트에 데이터가 없습니다.");
        } else {
                while (true) {
                        temp = temp.prev;
                        System.out.print(temp.data);
                        if (temp.prev == head) {
                                break;
                        }
                        System.out.print("->");
                }
                System.out.println();
        }
}
```

LinkedList

- Double Linked List(이중 연결 리스트)

```
// 위치와 데이터를 받아서 데이터를 중간에도 삽입할 수 있는 Method
public void insertElement(int pos, int data) {
        if(pos >= 0) {
                if(pos > size()) {
                        System.out.println("삽입할 수 없는 위치 입니다.");
                }
                else if(pos == size()) {
                        addElement(data);
                }else {
                        Node node = new Node();
                        node.data = data;
                        int i=0;
                        Node temp = head;
                        while(i <= pos-1) {
                                temp = temp.next;
                                i = i + 1;
                        }
                        node.next = temp.next;
                        temp.next.prev = node;
                        temp.next = node;
                        node.prev = temp;
                }
        }
```

LinkedList

❖ Double Linked List(이중 연결 리스트)

```
                else {
                        pos = pos * -1;
                        if(pos > size()+1) {
                                System.out.println("삽입할 수 없는 위치 입니다.");
                        }else if(pos == -1) {
                                addElement(data);
                        }else {
                                Node node = new Node();
                                node.data = data;
                                int i=0;
                                Node temp = tail;
                                while(i < pos) {
                                        temp = temp.prev;
                                        i = i + 1;
                                }
                                node.next = temp.next;
                                temp.next.prev = node;
                                temp.next = node;
                                node.prev = temp;
                        }
                }
        }
```

LinkedList

- Double Linked List(이중 연결 리스트)

```
// pos 위치에 해당하는 데이터를 수정하는 Method
public void updateElement(int pos, int data) {
        if (size() == 0) {
                System.out.println("데이터가 없어서 데이터를 수정할 수
없습니다.");
                return;
        }
        if (pos >= size() || pos < -size()) {
                System.out.println("수정할 수 없는 위치입니다.");
                return;
        }

        Node node = getElement(pos);
        node.data = data;
}
```

LinkedList

- Double Linked List(이중 연결 리스트)

```
// pos 위치에 해당하는 데이터를 삭제하는 Method
public void removeElement(int pos) {
        if (size() == 0) {
                System.out.println("데이터가 없어서 삭제할 수 없습니다.");
                return;
        }
        if (pos >= size() || pos < -size()) {
                System.out.println("삭제할 수 없는 위치입니다.");
                return;
        }
        if(pos >= 0) {
                Node temp = head;
                int i=0;
                while(i<pos) {
                        temp = temp.next;
                        i=i+1;
                }
                Node node = temp.next;
                temp.next = node.next;
                node.next.prev = temp;
                node = null;
        }
```

LinkedList

❖ Double Linked List(이중 연결 리스트)

```
                else {
                        pos = -pos;
                        Node temp = tail;
                        int i=0;
                        while(i < pos) {
                                temp = temp.prev;
                                i=i+1;
                        }
                        Node prev = temp.prev;
                        Node next = temp.next;
                        prev.next = next;
                        next.prev = prev;
                        temp = null;
                }
        }
```

LinkedList

- Double Linked List(이중 연결 리스트)

```
// list 전체를 초기화하는 Method
public void clearElement() {
        if (size() == 0) {
                System.out.println("데이터가 없어서 데이터를 삭제할 필요가
없습니다.");
                return;
        }
        while (size() != 0) {
                removeElement(0);
        }
  }
}
```

LinkedList

❖ Double Linked List(이중 연결 리스트)

```
public class DoubleLinkedListMain {
   public static void main(String[] args) {
            DoubleLinkedList list = new DoubleLinkedList();
            list.addElement(100);
            list.addElement(200);
            list.addElement(300);

            System.out.println(list.size());
            System.out.println(list.getElement(0).data);
            System.out.println(list.getElement(1).data);
            System.out.println();

            System.out.println(list.getElement(-1).data);
            System.out.println(list.getElement(-2).data);
            System.out.println();
```

LinkedList

❖ Double Linked List(이중 연결 리스트)

```
Node node = list.getElement(1);
System.out.println(list.prevElement(node).data);
System.out.println(list.nextElement(node).data);

list.traversal();
System.out.println();
list.reversetraversal();
System.out.println();

list.insertElement(0, 80);
list.traversal();
list.insertElement(2, 150);
list.traversal();
list.insertElement(5, 400);
list.traversal();
System.out.println();
```

LinkedList

- Double Linked List(이중 연결 리스트)

```
list.insertElement(-1, 500);
list.traversal();
list.insertElement(-8, 10);
list.traversal();
System.out.println();

list.updateElement(1, 90);
list.traversal();
System.out.println();

list.updateElement(-1, 600);
list.traversal();
System.out.println();

list.removeElement(1);
list.traversal();
System.out.println();
```

LinkedList

- Double Linked List(이중 연결 리스트)

```
                list.removeElement(-2);
                list.traversal();
                System.out.println();

                list.clearElement();
                list.traversal();
        }
    }
```

LinkedList

- 2개의 정렬된 배열의 병합
  - ✓ 2개의 정렬된 배열을 병합해서 하나의 배열 만들기
  - ✓ 입력: 1->2->4, 1->3->4
  - ✓ 출력: 1->1->2->3->4->4

LinkedList

- 2개의 정렬된 배열의 병합
  - ✓ Python

```
# 두 개의 정렬된 목록 `X`와 `Y`를 병합하는 기능
def solution(X, Y):
    k = i = j = 0
    aux = [0] * (len(X) + len(Y))

    # 첫 번째 및 두 번째 배열에 요소가 있는 동안
    while i < len(X) and j < len(Y):
        if X[i] <= Y[j]:
            aux[k] = X[i]
            i = i + 1
        else:
            aux[k] = Y[j]
            j = j + 1
        k = k + 1
```

LinkedList

- 2개의 정렬된 배열의 병합
  - ✓ Python

```
# 나머지 요소 복사
while i < len(X):
    aux[k] = X[i]
    k = k + 1
    i = i + 1

while j < len(Y):
    aux[k] = Y[j]
    k = k + 1
    j = j + 1

return aux
```

LinkedList

- 2개의 정렬된 배열의 병합
  - ✓ Python

```
X = [1, 4, 7, 8, 10]
Y = [2, 3, 9]

aux = solution(X, Y)

print('First Array :', X)
print('Second Array:', Y)
print('After Merge :', aux)
```

LinkedList

- 2개의 정렬된 배열의 병합
  - ✓ Java

```
class Main
{
    // 두 개의 어레이된 어레이을 병합하는 함수 X[] 및 Y[]
    public static int[] solution(int[] X, int[] Y)
    {
        int k = 0, i = 0, j = 0;
        int[] aux = new int[X.length + Y.length];

        // 첫 번째와 두 번째 배열에 요소가 있는 동안
        while (i < X.length && j < Y.length)
        {
            if (X[i] <= Y[j]) {
                aux[k++] = X[i++];
            }
            else {
                aux[k++] = Y[j++];
            }
        }
```

LinkedList

- 2개의 정렬된 배열의 병합
  - ✓ Java

```
        // 나머지 요소 복사
        while (i < X.length) {
            aux[k++] = X[i++];
        }

        while (j < Y.length) {
            aux[k++] = Y[j++];
        }

        return aux;
    }
```

LinkedList

- 2개의 정렬된 배열의 병합
  - ✓ Java

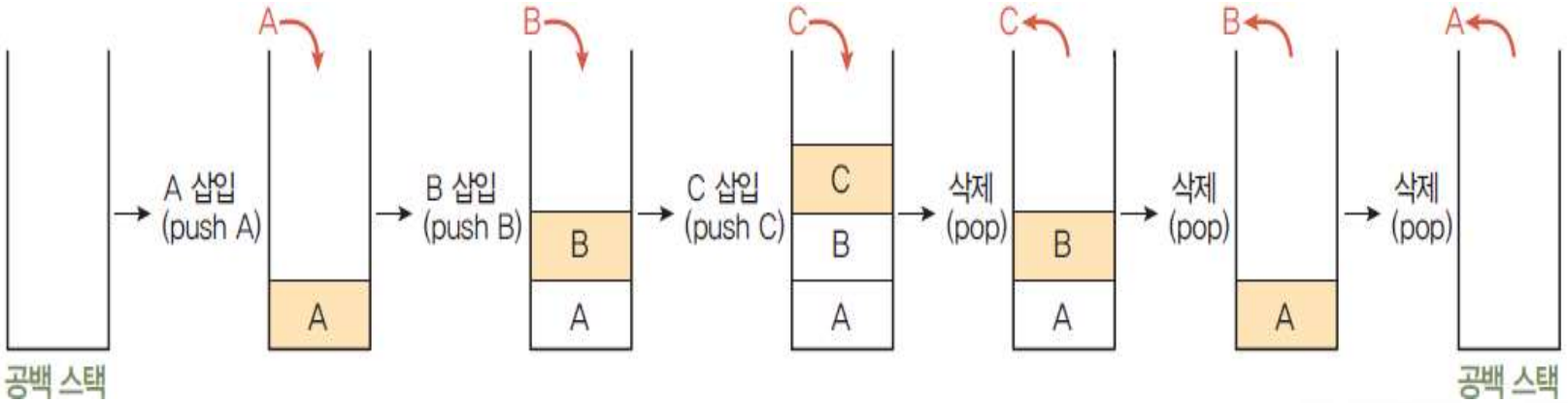
```
        public static void main (String[] args)
        {
            int[] X = { 1, 4, 7, 8, 10 };
            int[] Y = { 2, 3, 9 };

            int[] aux = solution(X, Y);

            System.out.println("First Array : " + Arrays.toString(X));
            System.out.println("Second Array: " + Arrays.toString(Y));
            System.out.println("After Merge : " + Arrays.toString(aux));
        }
    }
```

Stack

- Stack
  - ✓ 접시를 쌓듯이 자료를 차곡차곡 쌓아 올린 형태의 자료구조
  - ✓ 스택에 저장된 원소는 top으로 정한 곳에서만 접근 가능
  - ✓ top의 위치에서만 원소를 삽입하므로, 먼저 삽입한 원소는 밑에 쌓이고 나중에 삽입한 원소는 위에 쌓이는 구조
  - ✓ 마지막에 삽입(Last-In)한 원소는 맨 위에 쌓여 있다가 가장 먼저 삭제(First-Out)됨
  - ✓ 후입선출 구조 (LIFO, Last-In-First-Out)

Stack

- Stack
  - ✓ 스택에서의 원소 삽입(push)/삭제(pop) 과정 - 스택에 원소 A, B, C를 순서대로 삽입하고 한번 삭제하는 연산 과정 동안의 스택 변화



  - ✓ peek: 마지막 데이터를 삭제하지 않고 리턴하는 동작
  - ✓ Stack의 용도
    - 운영체제(OS: Operating System): 프로그램에서 사용되는 함수들을 스택 자료형에 저장하여 사용
    - 컴파일러(Compiler): 수학 기호들을 기계어로 변환 시 사용(괄호 등을 매칭하는 경우)
    - 자바 가상 머신(JVM: Java Virtual Machine): JVM 내에서 메서드가 실행, 종료될 때 스택 프레임을 이용하여 관리

Stack

- 배열을 이용한 Stack
  - ✓ 생성자 – 초기값을 받아서 배열을 초기값 크기로 생성
  - ✓ boolean isEmpty – 스택이 비어있는지 확인하는 Method
  - ✓ boolean isFull – 스택에 데이터가 전부 저장되었는지 확인하는 Method
  - ✓ void push(char item) – 데이터를 삽입하는 Method
  - ✓ char pop() – 마지막 데이터를 제거하고 리턴하는 Method
  - ✓ char peek() – 마지막 데이터를 리턴하는 Method
  - ✓ void clear() – 스택을 초기화하는 Method
  - ✓ void printStack – 스택의 모든 데이터를 출력해주는 Method

Stack

- 배열을 이용한 Stack
  - ✓ ArrayStack

```
class ArrayStack  {
    private int top;
    private int stackSize;
    private char stackArr[];

    // 스택을 생성하는 생성자
    public ArrayStack(int stackSize) {
        top = -1;    // 스택 포인터 초기화
        this.stackSize = stackSize;    // 스택 사이즈 설정
        stackArr = new char[this.stackSize];    // 스택 배열 생성
    }

    // 스택이 비어있는 상태인지 확인
    public boolean isEmpty() {
        // 스택 포인터가 -1인 경우 데이터가 없는 상태이므로 true 아닌 경우 false를
     return
        return (top == -1);
    }
```

Stack

- 배열을 이용한 Stack
  - ✓ ArrayStack

```
// 스택이 가득찬 상태인지 확인
public boolean isFull() {
    // 스택 포인터가 스택의 마지막 인덱스와 동일한 경우 true 아닌 경우 false를
 return
    return (top == this.stackSize-1);
}

// 스택에 데이터를 추가
public void push(char item) {
    if(isFull()) {
        System.out.println("Stack is full! - Overflow");
    } else {
        stackArr[++top] = item;    // 다음 스택 포인터가 가리키는 인덱스에 데이터
 추가
        System.out.println("Inserted Item : " + item);
    }
}
```

Stack

- 배열을 이용한 Stack
  - ✓ ArrayStack

```
// 스택의 최상위(마지막) 데이터 추출 후 삭제
public char pop() {
    if(isEmpty()) {
        System.out.println("마지막 데이터 삭제하면서 가져오기 실패 - 데이터 없음");
        return 0;
    } else {
        System.out.println("Deleted Item : " + stackArr[top]);
        return stackArr[--top];
    }
}

// 스택의 최상위(마지막) 데이터 추출
public char peek() {
    if(isEmpty()) {
        System.out.println("마지막 데이터 가져오기 실패 - 데이터 없음");
        return 0;
    } else {
        System.out.println("Peeked Item : " + stackArr[top]);
        return stackArr[top];
    }
}
```

Stack

- 배열을 이용한 Stack
  - ✓ ArrayStack

```
// 스택 초기화
    public void clear() {
        if(isEmpty()) {
            System.out.println("스택은 이미 초기화 되어 있음!");
        } else {
            top = -1;    // 스택 포인터 초기화
            stackArr = new char[this.stackSize];    // 새로운 스택 배열 생성
            System.out.println("스택 초기화 성공");
        }
    }
```

Stack

- 배열을 이용한 Stack
  - ✓ ArrayStack

```
// 스택에 저장된 모든 데이터를 출력
public void printStack() {
    if(isEmpty()) {
        System.out.println("스택에 데이터가 없음!");
    } else {
        System.out.print("스택의 데이터 : ");
        for(int i=0; i<=top; i++) {
            System.out.print(stackArr[i] + "    ");
        }
        System.out.println();
    }
}
}
```

Stack

- 배열을 이용한 Stack
  - ✓ ArrayStack

```
public class ArrayStackMain {
    public static void main(String args[]) {
        int stackSize = 5;
        ArrayStack arrStack = new ArrayStack(stackSize);

        arrStack.push('A');
        arrStack.printStack();

        arrStack.push('B');
        arrStack.printStack();

        arrStack.push('C');
        arrStack.printStack();

        arrStack.pop();
        arrStack.printStack();

        arrStack.pop();
        arrStack.printStack();
```

Stack

- 배열을 이용한 Stack
  - ✓ ArrayStack

```
            arrStack.peek();
            arrStack.printStack();

            arrStack.clear();
            arrStack.printStack();
        }
    }
```

Stack

- LinkedList를 이용한 Stack

```
class Node {
    char data;
    Node next;
}
```

Stack

- LinkedList를 이용한 Stack

```
class LinkedListStack {
    // 출발점을 만들기 위한 노드
    private Node head;

    // 생성자 - head에 노드 할당
    public LinkedListStack() {
       head = new Node();
    }

    // 데이터를 마지막에 추가하는 Method
    public void push(char data) {
       Node temp = head;
       while (true) {
                    if (temp.next == null) {
                                 break;
                    }
                    temp = temp.next;
       }
       Node node = new Node();
       node.data = data;
       temp.next = node;
    }
```

Stack

❖ LinkedList를 이용한 Stack

```
//데이터 개수를 리턴하는 Method
public int size() {
   Node temp = head;
   int cnt = 0;
   while (true) {
                if (temp.next == null) {
                        break;
                }
                temp = temp.next;
                cnt = cnt + 1;
   }
   return cnt;
}
```

Stack

- LinkedList를 이용한 Stack

```
//마지막 데이터를 삭제하고 리턴하는 Method
public char pop() {
   if (size() == 0) {
                return 0;
   }
   Node node = head;
   int i = 0;
   while (i < size() - 1) {
                node = node.next;
                i = i + 1;
   }
   Node lastNode = node.next;
   char ch = lastNode.data;
   node.next = null;
   lastNode = null;
   return ch;
}
```

Stack

❖ LinkedList를 이용한 Stack

```
//Stack 전체를 순회하는 Method
public void printStack() {
   Node temp = head.next;
   if (temp == null) {
                System.out.println("리스트에 데이터가 없습니다.");
   } else {
                while (true) {
                             System.out.print(temp.data);
                             temp = temp.next;
                             if (temp == null) {
                                          break;
                             }
                }
                System.out.println();
   }
}
```

Stack

❖ LinkedList를 이용한 Stack

```
        //Stack을 초기화하는 Method
        public void clear() {
          if (size() == 0) {
                        return;
          }
          while (true) {
                        char ch = pop();
                        if (ch == 0) {
                                      break;
                        }
          }
        }
    }
```

Stack

- Stack을 사용해서 수식의 괄호 검사
  - ✓ 입출력 예

s	answer
"()()"	true
"(())()"	true
")()("	false
"(()("	false

Stack

- 수식의 괄호 검사
  - ✓ Java – 스택 활용

```
public class Main{
    public static boolean solution(String s) {
        Stack<Character> stack = new Stack<>();

        for (char c : s.toCharArray()) {
            switch (c) {
                case '(' -> stack.push(c);
                case ')' -> {
                    if (stack.isEmpty()) return false;
                    stack.pop();
                }
            }
        }

        return stack.isEmpty();
    }
```

Stack

- 수식의 괄호 검사
  - ✓ Java – 스택 활용

```
public static void main(String [] args) {
        System.out.println(solution("()()"));
        System.out.println(solution("(())()"));
        System.out.println(solution(")()("));
        System.out.println(solution("(()("));
    }
}
```

Stack

- 수식의 괄호 검사
  - ✓ Java – 카운터 활용

```
public class Main{
    public static boolean solution(String s) {
      int counter = 0;

        for (char c : s.toCharArray()) {
            switch (c) {
                case '(' -> counter++;
                case ')' -> {
                    if (counter-- == 0) return false;
                }
            }
        }

        return counter == 0;
    }
```

Stack

- 수식의 괄호 검사
  - ✓ Java – 카운터 활용

```
public static void main(String [] args) {
        System.out.println(solution("()()"));
        System.out.println(solution("(())()"));
        System.out.println(solution(")()("));
        System.out.println(solution("(()("));
    }
}
```

Stack

- Stack을 사용해서 수식의 괄호 검사
  - ✓ (또는) 로만 이루어진 문자열 s가 주어졌을 때, 문자열 s가 올바른 괄호이면 true를 return하고 올바르지 않은 괄호이면 false를 return하는 solution 함수를 완성해주세요
  - ✓ 알고리즘
    - 여는 괄호를 만났을 때 스택에 괄호를 저장
    - 닫는 괄호를 만났을 때 스택에서 저장한 괄호를 pop 해서 쌍이 맞지 않으면 괄호의 순서가 잘못된 것임
    - 닫는 괄호를 만났을 때 스택에서 더 이상 꺼낼 괄호가 없다면 여는 괄호가 부족한 것임
    - 닫는 괄호를 전부 만났는데 아직 스택에 데이터가 남아있다면 닫는 괄호가 부족한 것임
  - ✓ 제한사항
    - 문자열 S의 길이: 100,000 이하의 자연수
    - 문자열 s는 (또는)로만 이루어져 있습니다

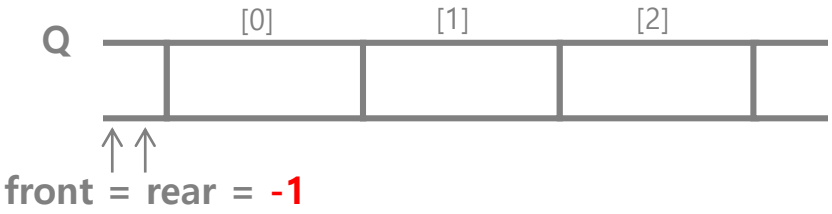
Queue

- Queue
  - ✓ 큐는 뒤(rear)에서는 삽입만 하고 앞(front)에서는 삭제만 할 수 있는 구조
  - ✓ 삽입한 순서대로 원소가 나열되어 가장 먼저 삽입(First-In)한 원소는 맨 앞에 있다가 가장 먼저 삭제(First-Out)

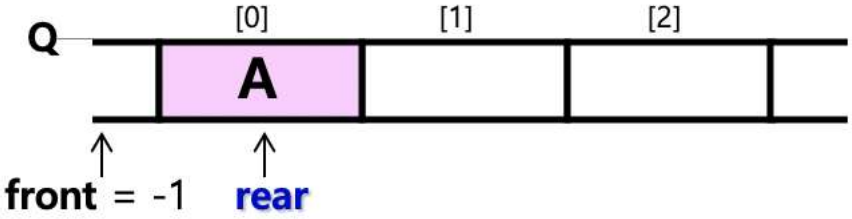
  - ✓ 큐의 연산
    - 삽입 : En-Queue
    - 삭제 : De-Queue

Queue

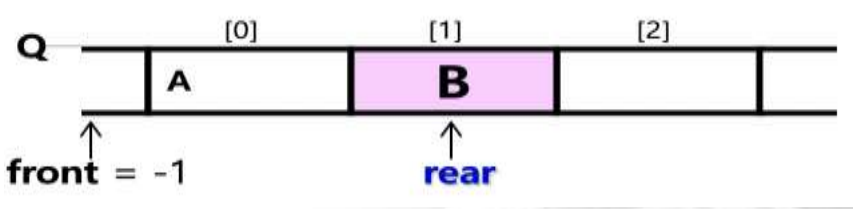
- Queue
  - ✓ 공백 큐 생성 : createQueue()



  - ✓ En-Queue(Q, A)

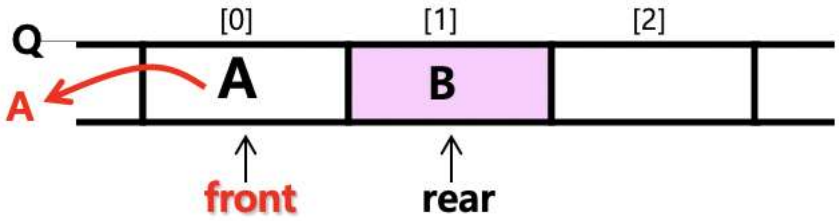


  - ✓ En-Queue (Q, B)

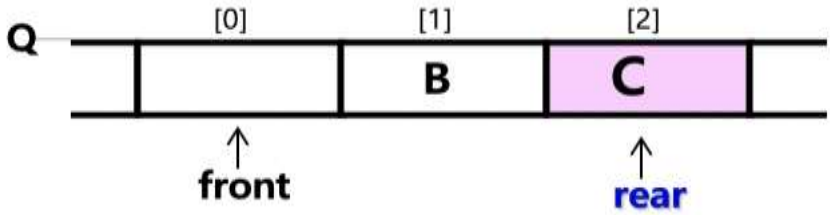


Queue

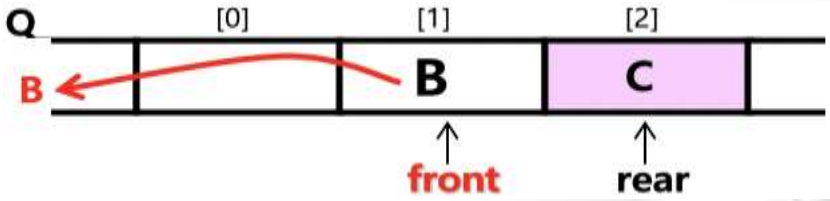
- Queue
  - ✓ De-Queue(Q)



  - ✓ En-Queue(Q, C);



  - ✓ De-Queue(Q)



Queue

❖ 배열로 구현된 문자 LinearQueue

```
//배열로 구현한 문자 큐
public class ArrayQueue{
    private int max;                    // 큐의 용량
    private int front;                  // 첫 번째 요소 커서
    private int rear;                   // 마지막 요소 커서
    private int num;                    // 현재 데이터 수
    private char[] que;                 // 큐 본체

    // 생성자
    public ArrayQueue(int capacity) {
       num = front = rear = 0;
       max = capacity;
       que = new char[max];                          // 큐 본체용 배열을  생성
    }
```

Queue

- 배열로 구현된 문자 LinearQueue

```
// 큐에 데이터를 인큐
    public void enQueue(char ch) {
      if (num >= max) {
                System.out.println("큐가 가득 참");
      }
      que[rear++] = ch;
      num++;
      System.out.println("삽입한 데이터:" + ch);

    }
}
```

Queue

❖ 배열로 구현된 문자 LinearQueue

```
// 큐에서 데이터를 디큐
public int deQuque(){
   if (num <= 0) {
                System.out.println("큐가 비어 있음");
                return 0;
   }
   char ch = que[front++];
   System.out.println("삭제된 데이터:" + ch);
   num--;
   return ch;
}
```

Queue

- 배열로 구현된 문자 LinearQueue

```
// 큐에서 데이터를 피크 (프런트 데이터를 들여다봄)
public int peek() {
   if (num <= 0) {
                System.out.println("큐가 비어 있음");
                return 0;
   }
   return que[front];
}

// 큐를 비움
public void clear() {
   num = front = rear = 0;
}

// 큐의 용량을 반환
public int capacity() {
   return max;
}
```

Queue

- 배열로 구현된 문자 LinearQueue

```
// 큐에 쌓여 있는 데이터 수를 반환
public int size() {
   return num;
}

// 큐가 비어 있나요?
public boolean isEmpty() {
   return num <= 0;
}

// 큐가 가득 찼나요?
public boolean isFull() {
   return num >= max;
}
```

Queue

- 배열로 구현된 문자 LinearQueue

```
// 큐 안의 모든 데이터를 프런트 → 리어 순으로 출력
public void printQueue() {
   if (num <= 0){
                System.out.println("큐가 비어 있습니다.");
   }else {
                for (int i = 0; i < num; i++){
                             System.out.print(que[(i + front)] + " ");
                }
                System.out.println();

   }
}
```

Queue

❖ 배열로 구현된 문자 LinearQueue

```
public class ArrayQueueMain {
        public static void main(String args[]) {
                int queueSize = 3;
                ArrayQueue arrQueue = new ArrayQueue(queueSize);

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

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

                arrQueue.enQueue('C');
                arrQueue.printQueue();

                arrQueue.deQuque();
                arrQueue.printQueue();
```

Queue

- 배열로 구현된 문자 LinearQueue

```
arrQueue.deQuque();
arrQueue.printQueue();

arrQueue.deQuque();

arrQueue.clear();
arrQueue.printQueue();

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

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

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

Queue

❖ 배열로 구현된 문자 LinearQueue

```
        arrQueue.deQuque();
            arrQueue.printQueue();
            //선형 큐의 문제점 - 2개의 데이터만 저장된 상태 인데도 데이터가 저장이 안됨
            arrQueue.enQueue('D');
            arrQueue.printQueue();

        }
    }
```

Queue

- 배열로 구현된 문자 CircularQueue
  - ✓ ArrayQueue 클래스에서 3개의 Method 수정

```
// 큐에 데이터를 인큐
    public void enQueue(char ch) {
        if (num >= max) {
            System.out.println("큐가 가득 참");
        }
        que[rear++] = ch;
        num++;
        if (rear == max) {
            rear = 0;
        }
        System.out.println("삽입한 데이터:" + ch);
    }
```

Queue

- 배열로 구현된 문자 CircularQueue
  - ✓ ArrayQueue 클래스에서 3개의 Method 수정

```
// 큐에서 데이터를 디큐
public int deQuque(){
        if (num <= 0) {
                System.out.println("큐가 비어 있음");
                return 0;
        }
        char ch = que[front++];
        System.out.println("삭제된 데이터:" + ch);
        num--;
        if (front == max) {
                front = 0;
        }
        return ch;
}
```

Queue

- 배열로 구현된 문자 CircularQueue
  - ✓ ArrayQueue 클래스에서 3개의 Method 수정

```
// 큐 안의 모든 데이터를 프런트 → 리어 순으로 출력
public void printQueue() {
        if (num <= 0)
                System.out.println("큐가 비어 있습니다.");
        else {
                for (int i = 0; i < num; i++) {
                        System.out.print(que[(i + front) % max] + " ");
                }
                System.out.println();
        }
}
```

Queue

- Queue를 이용한 기능 개발
  - ✓ 문제
    - 프로그래머스 팀에서는 기능 개선 작업을 수행 중입니다.
    - 각 기능은 진도가 100%일 때 서비스에 반영할 수 있습니다.
    - 각 기능의 개발 속도는 모두 다르기 때문에 뒤에 있는 기능이 앞에 있는 기능보다 먼저 개발될 수 있고 이때 뒤에 있는 기능은 앞에 있는 기능이 배포될 때 함께 배포됩니다.
    - 먼저 배포되어야 하는 순서대로 작업의 진도가 적힌 정수 배열 progresses와 각 작업의 개발 속도가 적힌 정수 배열 speeds가 주어질 때 각 배포마다 몇 개의 기능이 배포되는지를 return 하도록 solution 함수를 완성하세요.
  - ✓ 제한 사항
    - 작업의 개수(progresses, speeds배열의 길이)는 100개 이하입니다.
    - 작업 진도는 100 미만의 자연수입니다.
    - 작업 속도는 100 이하의 자연수입니다.
    - 배포는 하루에 한 번만 할 수 있으며 하루의 끝에 이루어진다고 가정합니다. 예를 들어 진도율이 95%인 작업의 개발 속도가 하루에 4%라면 배포는 2일 뒤에 이루어집니다.

Queue

- Queue를 이용한 기능 개발
  - ✓ 입출력 예

progresses	speeds	return
[93, 30, 55]	[1, 30, 5]	[2, 1]
[95, 90, 99, 99, 80, 99]	[1, 1, 1, 1, 1, 1]	[1, 3, 2]

Queue

- Queue를 이용한 기능 개발
  - ✓ 입출력 예 설명
    - 입출력 예 #1
      - 첫 번째 기능은 93% 완료되어 있고 하루에 1%씩 작업이 가능하므로 7일간 작업 후 배포가 가능합니다.
      - 두 번째 기능은 30%가 완료되어 있고 하루에 30%씩 작업이 가능하므로 3일간 작업 후 배포가 가능합니다. 하지만 이전 첫 번째 기능이 아직 완성된 상태가 아니기 때문에 첫 번째 기능이 배포되는 7일째 배포됩니다.
      - 세 번째 기능은 55%가 완료되어 있고 하루에 5%씩 작업이 가능하므로 9일간 작업 후 배포가 가능합니다.
      - 따라서 7일째에 2개의 기능, 9일째에 1개의 기능이 배포됩니다.
    - 입출력 예 #2
      - 모든 기능이 하루에 1%씩 작업이 가능하므로, 작업이 끝나기까지 남은 일수는 각각 5일, 10일, 1일, 1일, 20일, 1일입니다. 어떤 기능이 먼저 완성되었더라도 앞에 있는 모든 기능이 완성되지 않으면 배포가 불가능합니다.
      - 따라서 5일째에 1개의 기능, 10일째에 3개의 기능, 20일째에 2개의 기능이 배포됩니다.

Queue

- Queue를 이용한 기능 개발
  - ✓ 자바

```
public class Main{
    public static int[] solution(int[] progresses, int[] speeds) {
        Queue<Integer> q = new LinkedList<>();
        for (int i = 0; i < progresses.length; i++) {
            q.add(i);
        }
```

Queue

- Queue를 이용한 기능 개발
  - ✓ 자바

```
        List<Integer> result = new ArrayList<>();
        int days = 0;
        int count = 0;
        while (!q.isEmpty()) {
            int index = q.poll();
            int expiration = (int) Math.ceil(
                    (double) (100 - progresses[index]) / speeds[index]);
            if (expiration > days) {
                if (days != 0) {
                    result.add(count);
                    count = 0;
                }
                days = expiration;
            }
            count++;
        }

        result.add(count);
        return result.stream().mapToInt(Integer::intValue).toArray();
    }
```

Queue

- Queue를 이용한 기능 개발
  - ✓ 자바

```
        public static void main(String [] args) {
                int [] progresses1 = {93, 30, 55};
                int [] speeds1 = {1, 30, 5};
                System.out.println(Arrays.toString(solution(progresses1, speeds1)));

                int [] progresses2 = {95, 90, 99, 99, 80, 99};
                int [] speeds2 = {1, 1, 1, 1, 1, 1};
                System.out.println(Arrays.toString(solution(progresses2, speeds2)));
        }
    }
```

Queue

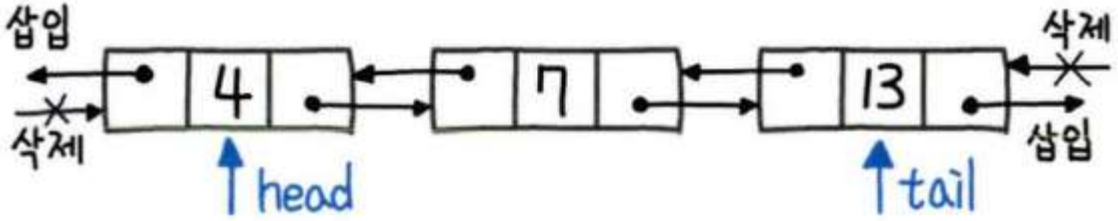
❖ Queue를 이용한 기능 개발

✓ 파이썬

```
def solution(progresses, speeds):
    answer = []

    while progresses:
        for i in range(len(progresses)):
            progresses[i] += speeds[i]

        cnt = 0
        while progresses and progresses[0] >= 100:
            progresses.pop(0)
            speeds.pop(0)
            cnt += 1

        if cnt > 0: answer.append(cnt)

    return answer


print(solution([93, 30, 55], [1, 30, 5]))
print(solution([95, 90, 99, 99, 80, 99], [1, 1, 1, 1, 1, 1]))
```

Deque

- Deque(Double-Ended Queue)
  - ✓ 개요
    - 양쪽에서 삭제와 삽입을 모두 처리할 수 있으며 스택과 큐의 특징을 모두 갖음
    - 구현은 배열이나 연결 리스트 모두 가능하지만 이중 연결 리스트(Doubly Linked List)로 구현하는 경우가 많음



    - 파이썬은 데크 자료형을 collections 모듈에서 deque 라는 이름으로 지원하고 자바에서는 java.util.ArrayDeque 라는 클래스로 지원