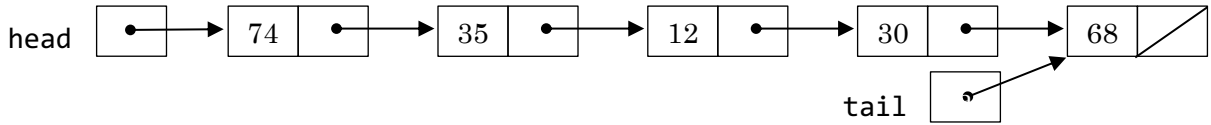


1. 下記のプログラムは、第8回の3.と同様のキューを単方向連結リストに末尾を指すフィールド tail を加えて実現したものである。適切に空欄を埋めてプログラムを完成させ、動作を確認せよ。



```

/* Node.java */
public class Node { // リスト要素 (ノード)

    // 同一パッケージからのみアクセス可能
    int data; // 格納データ
    Node next; // 次のノードへのリンク

    Node(int data) {

    }

}

```

```

/* Queue.java */
public class Queue {

    // 連結リストの先頭と末尾のノードを指す
    private Node head, tail;

    public Queue() {
        head = tail = null;
    }

    // リストの末尾にノードを加える
    public void enqueue(int data) {
        Node n = new Node(data);

        // もしリストが空だった場合
        if (head == null) {
            head = tail =
        }
        else { // そうでない場合
            = n;

            tail =
        }
    }
}

```

```

// リストの先頭からノードを取り出す
public int dequeue() throws Exception {
    // データがない場合をチェック
    if (head == ) {

        throw new Exception("empty!");
    }
    // data を取り出し、新しい head を設定
    int data = head.data;
    head =

    // リストが空になった場合
    if (head == null)
        tail =

    return data;
}

public void printAll() {
    System.out.print("[ ");

    // リストの中のノードを1つずつたどる
    for (Node n = head;
        n != null; n = n.next) {

    }
    System.out.println("]");
}
}

```

```

/* Program.java */
/* メインプログラムは第8回の3.のものを
そのま利用する。ただし new Queue(8)
は new Queue() に書き換える。
*/

```

2. 1.のプログラムにおいて、enqueue と dequeue のリンクのつなぎ替えの処理を図示せよ。さらに、キュー内のデータ数 n に対する enqueue と dequeue の計算量 (オーダー) および tail の役割を考察せよ。

3. 下記は、ジェネリクスを用いて第9回の2.の連結リストを定義し直したものである。クラス定義の最初の `class Node<E>` は、`E` に任意のクラス名が当てはめられてくることを示し、フィールドおよびメソッドの定義で `E` を使うことができる。ジェネリッククラスの利用方法については、第6回で学んだ通りである。

```

/* Node.java */
public class Node<E> {
    public E data;
    public Node next;

    public Node(E data) {
        this.data = data;
        next = null;
    }
}

/* List.java */
public class List<E> {
    public Node<E> head;

    public List() {
        head = null;
    }

    public void printAll() {
        // 第9回の2.を参考に作成せよ
    }
}

public Node<E> search(E data) {
    // 第9回の5.を参考に作成せよ
}

/* Program.java */
public class Program {
    public static void main(String[] args) {

        // new の<>の中は明確なら省略できる
        List<String> list = new List<>();
        list.head = new Node<String>("ABC");

        // 適当にリストを構築して表示させてみよう

        list.printAll();
        Node<String> n = list.search("DEF");
        if (n != null)
            System.out.println(n.data);
    }
}

```

4. 3.の `List` クラスに、引数で指定されたノード `p` の次に新しいノードを挿入するメソッド `insertNext` と、同様に `p` の次のノードを削除するメソッド `removeNext` を追加し、`main` も変更して動作を確認せよ。

```

// ノード p の“次”に新ノード n を挿入
public void insertNext(Node<E> p, Node<E> n) {
    // p が null のときは先頭に挿入
    if (p == null) {
        n.next = head;
        head = n;
    } else {
        n.next =

        p.next =
    }
}

// ノード p の“次”のノードを削除
public void removeNext(Node<E> p) {
    // p が null のときは先頭を削除
    if (p == null) {
        if (head != null)
            head = head.next;
    } else if (p.next != null) {

        p.next =
    }
}

```

5. Java では `java.util` パッケージにおいて、**双方向連結リスト**のクラス `LinkedList<E>` が提供されている。このクラスでは、例えば下の表のような操作が可能である（詳しくは `Java API` のリファレンスを参照）。これを参考に、`LinkedList<String>` のインスタンスにいくつかの文字列を登録（追加）し、辞書順（アルファベット順・あいうえお順）に並べ替えてから、全要素を表示するプログラムを作成せよ。

先頭への追加	<code>list.addFirst(e)</code> <code>list.push(e)</code>	末尾への追加	<code>list.addLast(e)</code> <code>list.add(e)</code>
先頭からの取り出し	<code>e = list.removeFirst()</code> <code>e = list.pop()</code>	末尾からの取り出し	<code>e = list.removeLast()</code> <code>e = list.poll()</code>
先頭からの線形探索	<code>i = list.indexOf(key)</code>	末尾からの線形探索	<code>i = list.lastIndexOf(key)</code>
マージソート	<code>Collections.sort(list)</code>	全要素へのアクセス	<code>for (E e : list)</code>