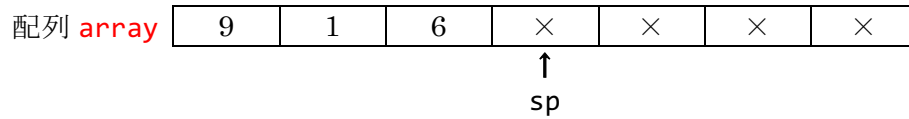


1. スタックとは後入れ先出し (LIFO, 先入れ後出し FILO) のデータ構造であり, 下記のプログラムはこれを配列で実現したものである。スタックにデータを格納する操作をプッシュ, スタックからデータを取り出す操作をポップという。適切に空欄を埋めてプログラムを完成させ, 動作を確認せよ。



```

/* Stack.java */
public class Stack {

    private int[] array; // スタック本体
    private int sp; // スタックポインタ

    public Stack(int size) {
        array = new int[size];
        sp = 0;
    }

    public void push(int data)
        throws Exception {
        // 空かない場合をチェック
        if (          ) {

            throw new Exception("full!");
        }
        // スタックにデータを「積む」

        array[sp] =

    }

    public int pop() throws Exception {
        // データがない場合をチェック
        if (          ) {

            throw new Exception("empty!");
        }
        // トップにあるデータを取り出す

        = array[sp];

        return data;
    }
}

public void printAll() {
    System.out.print("[ ");
    for (int i = 0; i < sp; i++)

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

/* Program.java */
import java.util.Scanner;

public class Program {

    public static void main(String[] args)
        throws Exception {

        // 容量 5 個のスタックの新規生成
        Stack stack = new Stack(5);
        Scanner sc = new Scanner(System.in);

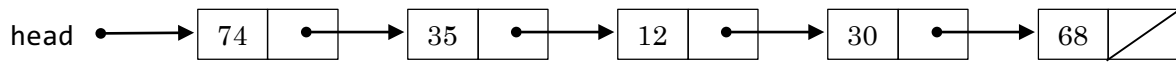
        for (int i = 0; i < 10; i++) {
            System.out.print("number? ");
            int data = sc.nextInt();

            if (data > 0) {
                // 入力为正数ならその数を格納
                stack.push(data);
            } else {
                // 入力為負数か 0 なら取り出し
                data = stack.pop();
                System.out.println(data);
            }
            stack.printAll();
        }
    }
}

```

2. 1.のプログラムのプッシュとポップの計算量は, スタック内のデータ数 n に対して $O(1)$ (一定) である。もしスタックポインタ (sp) が無いとどうなるか想定し, それが高速化に果たしている役割を考察せよ。

3. 連結リストは動的なデータ構造の一種であり、データを格納したノード（またはセル）をリンクによって連結したものである。下記のプログラムは、1.と同様のスタックを単方向リストで実現したものである。このアルゴリズムでは、リストの先頭（head）にデータを格納し、先頭から順に取り出す。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。また、このアルゴリズムでもスタック内のデータ数 n に対するプッシュとポップの計算量は $O(1)$ になることを理解せよ。



```

/* Stack.java */
public class Stack {

    // 連結リストの先頭ノードを指す
    private Node head;

    public Stack() {
        head = null;
    }

    // リストの先頭にノードを加える
    public void push(int data) {
        Node n = new Node(data);

        // nの後ろに今のリストをつなげる
        n.next = head;

        head = n;
    }

    // リストの先頭からノードを取り出す
    public int pop() throws Exception {
        // データがない場合をチェック
        if (head == null) {
            throw new Exception("empty!");
        }
        // dataを取り出し、新しいheadを設定
        int data = head.data;
        head = head.next;

        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.print(n.data + " ");
    }
    System.out.println("]");
}

/* 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) {
        this.data = data;
    }
}

/* Program.java */
// メインプログラムは1.のものをそのまま
// 利用する。ただし、new Stack(5); は
// new Stack() に書き換える。

```

4. クラス Stack に連結リストの先頭から順にデータを探索するメソッドを追加し、その計算量を考察せよ（これも線形探索の一種である）。データが見つからなかった場合は、nullを返すようにすること。

```

public Node search(int data) {

}

```