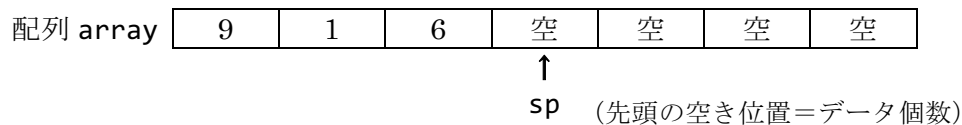


1. スタックとは後入れ先出し (LIFO, 先入れ後出し FILO) のデータ構造であり, 下記のプログラムはこれを配列で実現したものである。スタックにデータを格納する操作を**プッシュ**, スタックからデータを取り出す操作を**ポップ**という。適切に空欄を埋めてプログラムを完成させ, 動作を確認せよ。



```

/* 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 {
        // 空きが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();

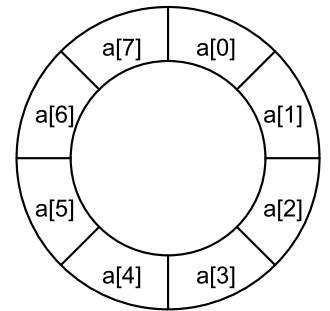
            if (data > 0) {
                // 入力为正数ならその数を格納
                stack.push(data);
            } else {
                // 入力為負数か0なら取り出し
                data = stack.pop();
                System.out.println(data);
            }
            stack.printAll();
        }
    }
}
    
```

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) がなくなる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. キュー（待ち行列）とは、先入れ先出し方式（FIFO、先着順処理）のデータ構造である。これを配列を使って素朴な方法で実現すると、先頭のデータを取り出すアルゴリズムは以下ようになる。この方法の計算量を O 記法で表せ。取り出し処理を高速にするためにはどうすればいいか考えてみよう。

```
data = a[0]; n--;
for (i = 0; i < n; i++)
    a[i] = a[i + 1];
```

4. リングバッファは右図のように配列の末尾と先頭がつながっているものとして扱うデータ構造である。下記はリングバッファによってキューを実現したプログラムである。キューの末尾にデータを追加する操作を**エンキュー**、キューの先頭からデータを取り出す操作を**デキュー**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。また、計算量の改善を考察せよ。



```
/* Queue.java */
public class Queue {

    private int[] array; // リングバッファ
    private int head; // キューの先頭 (添字)
    private int n; // 格納しているデータ数

    public Queue(int size) {
        array = new int[size];
        head = 0;
        n = 0;
    }

    public void enqueue(int data)
        throws Exception {
        // 空きがない場合をチェック
        if (n
            ) {

            throw new Exception("full!");
        }
        // キューの末尾の位置 (添字) を求める
        int tail = (head + n) %

        array[tail] = data;
    }

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

            throw new Exception("empty!");
        }
        int data = array[head];

        // 先頭位置を 1 つ進める
        head = (head + 1) %

        return data;
    }
}
```

```
public void printAll() {
    System.out.print("[ ");
    for (int i = 0; i < n; i++) {
        int k = (head + i) % array.length;
        System.out.print(array[k] + " ");
    }
    System.out.println("]");
}

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

public class Program {

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

        // 容量 8 個のキューの新規生成
        Queue queue = new Queue(8);
        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
                queue.enqueue(data);
            } else {
                // 入力為負 or 0 なら取り出し
                data = queue.dequeue();
                System.out.println(data);
            }
            queue.printAll();
        }
    }
}
```

5. 1.と 4.のプログラムを元にして文字列を要素とするスタックとキューを作成し、それぞれ動作を確認せよ。