# Lab 09: Stacks and Queues

COSC 102 - Spring '24

**Goal:** In this week's lab, you will utilize provided `Stack` and `Queue` implementations to gain familiarity with the workings and advantages of these data structures.

## 1 Overview

You will be utilizing `Stack` and `Queue` implementations, provided in **`LabStack.java`** and **`LabQueue.java`**, to complete the tasks below. Also pay attention to each task's argument and time/space complexity constraints.

Review the methods and constructors available in the `LabStack` and `LabQueue` classes – comments are provided to understand their utility. You **may not** modify these classes.

## 2 Your Task

Implement the four methods defined below in your **`Lab09.java`**. For each method, test it with a variety of cases in your `Lab09Client.java`. You will submit your test cases (which will make up part of this lab's grade!)

### 2.1 Smallest to Top

In your `Lab09.java`, create a `static` function **`smallestToTop`**, which accepts a `LabStack` of `int`s and modifies the `labStack` so that the **smallest** number is now at the top. All other values in the `LabStack` stay in their original positions. If multiple values are tied for the smallest, only the value *closest to the bottom* is moved up.

For example, given the following `LabStack` (left-most number being bottom, right-most number being top):

```
{3, 1, 6, 2, 4, 7, 3, 9}
```

After calling `smallestToTop(...)`, the contents of the `LabStack` would be:

```
{3, 6, 2, 4, 7, 3, 9, 1}
```

A few additional notes/requirements:

- This function **does not return anything**, but rather modifies the argument `LabStack` in place.
- Your solution must run in **`O(n)`**, where `n` is the size of the argument `LabStack`.
- Your solution may use additional **primitive** and **`LabStack`** type variables, but **no other data structures** (*i.e.* no array, `ArrayList`, etc). In other words, you're limited to **constant** extra space, excluding any `LabStacks`.
- if a `null` or empty `Stack` argument is provided, this function does nothing.

## 2.2 Bring to Front

In your `Lab09.java`, create a `static` function **`bringToFront`** which accepts two arguments **in this order:** a `LabQueue` of `String`s and an `int` index. The function moves the `String` at the specified index of the argument `LabQueue` to the *front* of the argument `LabQueue`.

For example, given an argument `LabQueue` containing (front to back):

```
[Cecil -> Golbez -> Rydia -> Cid -> Tellah]
```

and an index argument of **`2`**, the provided `LabQueue` would be changed to:

```
[Rydia -> Cecil -> Golbez -> Cid -> Tellah]
```

Some additional requirements for this function are listed on the next page:

- This function **does not return anything**, but rather modifies the argument `LabQueue` in place.
- Your solution must run in **`O(n)`**, where `n` is the size of the argument `LabQueue`.
- Your solution may only create additional primitive and `String` variables – no other data structures **including no additional `LabQueue`s**!
- If a `null` `LabQueue` or invalid index is passed, the function does nothing.

## 2.3 Mirrored Positives

In your `Lab09.java`, create a `static` function **`mirroredPositives`**, which accepts an `LabQueue` of ints and returns a `boolean` indicating whether the *positive* numbers in the argument `LabQueue` are mirrored – meaning the `LabQueue` contains the same positive values front-to-back as back-to-front.

The values are considered mirrored if the first positive value in the `LabQueue` matches the last positive value; the second positive matches the second-to-last positive, and so on. An empty `LabQueue` is also considered mirrored. If a `LabQueue` contains an odd number of positives, the median positive value is considered mirrored with itself.

Any **non positive-values (`<= 0`) are ignored**, meaning they aren't factored in when determining if the argument `LabQueue` is mirrored. For example:

`[17 -> 4 -> 8 -> 9 -> 8 -> 4 -> -12 -> 17]` would return **`true`**

`[12 -> 2 -> 6 -> 12]` would return **`false`**

`[0 -> -5 -> 13 -> 8 -> 13]` would return **`true`**

A few additional notes/requirements:

- The argument `LabQueue` must be **in its original state** after the function is done running – meaning it should have the same data in the same order before and after the function is ran.
- Your solution must run in **`O(n)`**, where `n` is the size of the argument `LabQueue`.
- Your solution may use additional **primitive** and **`LabStack`** type variables, but **no other data structures** (*i.e.* no array, `ArrayList`, etc). In other words, you're limited to **constant** extra space, excluding any `LabStacks`.
- If a `null` argument is provided, this function returns **`false`**.

### 2.4 Get Common Nums

In your `Lab09.java`, create a `static` function named **`getCommonNums`** which accepts two `LabQueue`s of **sorted, unique** `int`s. This function returns a new `LabQueue` containing only the `int`s appearing in **both** `LabQueue`s, in sorted order.

For example, given `LabQueue`s containing:

```
[1 -> 4 -> 8 -> 11]  and  [1 -> 4 -> 5 -> 11 -> 21]
```

`getCommonNums`'s returned `LabQueue` would contain:

```
[1 -> 4 -> 11]
```

A few final notes/requirements:

- Your solution must run in **`O(n)`**, where `n` is the combined size of the argument `LabQueue`s.
- Both argument `LabQueue`s must be **in their original state** after the function is done running.
- Your solution may only create additional primitive variables, as well as **one additional `LabQueue`** (which you will return). No other data structures.
- You **may assume** both argument `LabQueue`s contain **sorted, unique** values and do no `null`s. If either argument is `null`, the function returns `null`. You do **not** need to account for arguments with unsorted/duplicate values.

## 3 Submission

Upload your **`Lab09.java`** and **`Lab09Client.java`** to the submission link on your lab's Moodle.

This lab is due:

- **Tuesday, April 23rd** by **5:00 PM** for lab sections **A**, **B**, and **C** (which meet on Wednesday)
- **Wednesday, April 24th** by **5:00 PM** for lab sections **D**, **E**, and **F** (which meet on Thursday)