STACK & QUEUE EXERCISES

In this mini-workbook, we'll get you up and running with Stacks and Queues.

In the guide, we will quickly highlight the key concepts that you need to know for your interview. If you're unfamiliar with any of these, I highly recommend taking some time to review them.

Then, I've provided you with a series of exercises to help you get up and running quickly. I recommend setting aside 2-3 days to work through these exercises. Simply start from the top and work through them as much as you have time.

Make sure to track the time you spend here in your Interview Prep Tracker. You can categorize this as "Study" time.

For these exercises, we have complete code samples and templates available here.

TABLE OF CONTENTS

KEY STACK & QUEUE TERMINOLOGY

- **Abstract Data Type**
  Stacks and Queues are both Abstract Data Types. This means that they define the behavior of a data structure rather than it's specific implementation. Stacks and Queues can both be implemented in many different ways as long as they adhere to the proper interface

- **LIFO (Last-In-First-Out)**
  LIFO describes the behavior of a Stack. In a stack, the last (or most recent) element added to the stack is the first to be removed from the stack

- **FIFO (First-In-First-Out)**
  FIFO describes the behavior of a Queue. This is the opposite of a stack, where the first element added is the first to be removed

KEY STACK & QUEUE PATTERNS

- **Accessing and Modifying Values**
  As always, it is important to know how to implement standard implementations of Stacks and Queues. These are fairly straightforward to implement using Linked Lists

- **Implementing Recursive Algorithms Iteratively**
  Oftentimes the easiest way to translate recursive code into iterative code is to use a Stack to explicitly implement the recursive stack. Recursion stores data onto an implicit "recursive stack," which we can explicitly implement using a Stack data structure.

  For example exercises, see Tree Traversals in the Trees Workbook

- **Processing Data In Reverse**
  Stacks are a very easy way to reverse the order of data

- **Inner Grouping Problems**
  In Exercise Set #2, we will go through several examples of this. These are problems where data is nested in some way (for example, Valid Parentheses). Stacks are useful because they allow us to handle the nested section of our data first before returning back to outer data

- **Breadth-first Search**
  This is covered more in the Trees and Graphs Exercises. It is commonly implemented using Queues

STACK & QUEUE EXERCISES

Exercise Set #1 - Accessing and Modifying Values

1. Implement a Stack of integers using a Linked List. Your stack should implement the following methods:
    a. `push(int x)` - Push an int onto the top of the stack
    b. `pop()` - Remove the top item from the stack
    c. `size()` - Get the size of the stack

2. Implement a Queue of integers using a Linked List. Your queue should implement the following methods:
    a. `enqueue(int x)` - Enqueue an int onto the front of the queue
    b. `dequeue()` - Remove the last item from the queue
    c. `size()` - Get the size of the queue

3. Given a stack, write a function that returns the nth element in the stack

    eg.
    ```
    stack = [1,2,3,4,5]
    nthElement(stack, 2) = 2
    ```

4. Implement a Stack using Queues (Full Problem Definiton)

Exercise Set #2 - Inner Grouping Problems

1. Valid Parentheses (Full Problem Definition)

2. Basic Calculator (Full Problem Definition)