3. Lab 3: Stacks & Queues

3.1. Objectives

- Know how to use the data structure Stack for solving real problems.

3.2. Problem 1: Simple stack application

Write a program to

- Convert a decimal number and convert it to octal form.
- Concatenate two stacks.
- Determine if the contents of one stack are identical to that of another.

3.3. Problem 2: Arithmetic Expression Evaluation

Given a string containing an infix-form arithmetic expression which contains:

- Single digit number (i.e., from 0 to 9)
- Operators such as +, -, *, /
- Parentheses.

Write a program to evaluate and display on screen the result of the expression.

Instruction for Problem 2: (Follow instructions step-by-step)

- Re-create, re-compile and re-run Java projects (postfix, infix).
- Extend the program by allowing multiple digit numbers in expressions such as 123+56*78-1. You need to extract the token before determining wh
- at type it belongs to. For example, you need to treat 123 as a single token but not 3 tokens (1, 2, and 3). ***Hints***: just use a loop.
- Extend the program by allowing not only constant numbers but also variables. You may ask the user to input variables' values when evaluating expression. You should start with a simple solution first, although it's not good. Don't try a highly complicated solution immediately.

3.4. QueueApp.java

- Write a method to display the queue array and the front and rear indices. Explain how wraparound works.
- Write a method to display the queue (loop from 1 to nItems and use a temporary front for wraparound).
- Display the aray, the queue, and the front and rear indices.
- Insert fewer items or remove fewer items and investigate what happens when the queue is empty or full.
- Extend the insert and remove methods to deal with a full and empty queue.

- Add processing time to the queue. Create a new remove method that removes item N after N calls to the method.
- Simulate a queue of customers each one served for a random amount of time. Investigate how simulation is affected by:
    - the size of the queue
    - the range of time for wich each customer is served
    - the rate at which customers arrive at the queue

3.5. StackApp.java

- Write methods to display the stack array and the stack itself. Use them to trace the stack operation.
- Extend the push and pop methods to deal with a full and empty stack.

3.6. PriorityQApp.java

- Write a method to display the queue and use to trace the queue operation.
- Modify the insert method to insert the new item at the rear. Compare this queue with QueueApp.java. Which one is more efficient?
- Use a priority queue instead of an ordinary one in the simulation experiments described above.

3.7. Challenge

Suppose you have a deque D containing the numbers (1,2,3,4,5,6,7,8), in this order. Suppose further that you have an initially empty queue Q. Implement a function that uses only D and Q (and no other variables) and results in D storing the elements in the order (1,2,3,**5,4**,6,7,8).

Basic Operations:

- Deque D:
    - pushFront(x), pushBack(x), popFront(), popBack().
- Queue Q:
    - enqueue(x), dequeue().

Extended Problem

Reversing the Deque

1. **Original Setup**: D contains (1,2,3,4,5,6,7,8. Q) is empty.
2. **Goal**: Reverse the order of D so that it becomes (8,7,6,5,4,3,2,1)
3. **Constraints**:
    - Only D and Q may be used.
    - No additional variables allowed.

Note: explain your algorithm in your report and show your results.