CSDS 233 Assignment #2

Due October 6, 2024, before 11:59 pm. 100 points

Submission Instructions

- The submissions will be evaluated on completeness, correctness, and clarity.
- Please provide sufficient comments in your source code to help the TAs read it.
- Please generate a single zip file containing all your *.java files needed for this assignment (not .class) and optionally a README.txt file with an explanation about added classes and extra changes you may have done.
- Name your file P1_YourCaseID_YourLastName.zip for your **coding exercises**. Submit your zip file electronically to Canvas.
- Please submit a **PDF** for your answers to the written exercise **separate from the zip file**.

Office Hours for This Assignment

- October 2 ,from 10:00 am - 11:00 am (only for Written Exercise):
  Zoom:https://cwru.zoom.us/j/2336305061?pwd=yN4eRRA0xbpFdnBy7kqxNxQ5oklo45.1
  Passcode: 313723
- September 26th from 4:00 pm - 5:00 pm:
  Zoom:https://cwru.zoom.us/j/8153265328?pwd=S0JUSWVyQWF2aEZ4MDY4UnM3L3E4QT09
  Passcode 666661

Contact Information

If you have questions outside of the scheduled office hours:

- For written questions, email: yxw2533@case.edu
- For coding problems, email: qat3@case.edu

Written Exercise [50 points]

Answer the following questions.

P.1)Answer the following questions: (10 points, 2 points each)

a) Each element in an array occupies 4 storage units. If the storage address of the 20th element is 500, what is the starting address of the array? (Give the calculation process or explanation.)

b) What do stacks and queues have in common?

c) For a binary tree with a depth of 8, what are the maximum and minimum possible numbers of nodes it can have? (Give the calculation process or explanation.)

d) A complete binary tree contains 256 nodes. What is its depth, and how many leaf nodes does it have? (Give the calculation process or explanation.)

e) If the input sequence of a stack is 1, 2, 3, 4, 5, 6, provide three possible output sequences.

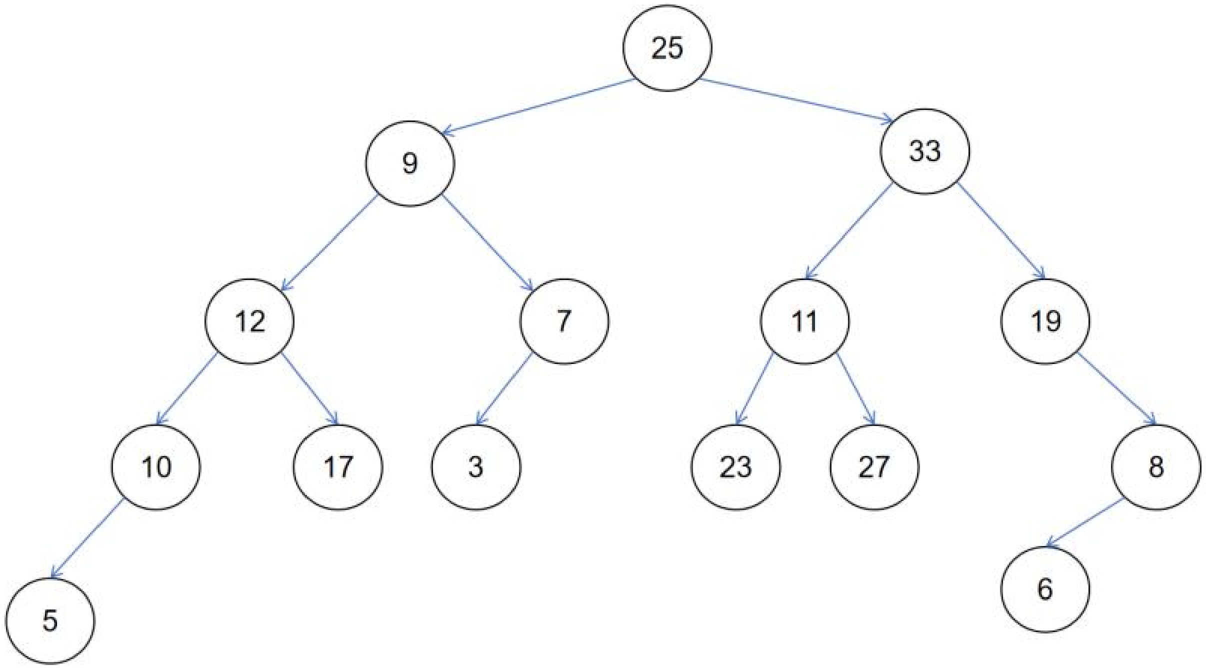
P.2)(20 points)

Consider the following binary tree with 15 nodes. Describe the order of nodes visited in each of the following traversals of the tree:

a) In-Order (6 points)

b) Pre-Order (6 points)

c) Post-Order (7 points)



P.3)(20 points)

Convert the following expressions from infix into postfix or from postfix into infix. (5 pts each)

a) $\boldsymbol{7/5 * G \,\hat{}\, 6 * C + A \,\hat{}\, 3 * B + Z - T}$

b) $X * Y^2 * \frac{7}{\sqrt{1-\frac{A^2}{B^2}}} - Z$

c) $X\ Y\ Z\ -\ /\ A\ 8\ O\ P\ Q\ /\ -\ *\ G\ 2\ \hat{}\ *\ /\ -$

d) $b\ 2\ -\ 4\ a\ c\ *\ -\ 2\ a\ *\ /\ +\ 7\ c\ *\ +$

Programming Exercise [50 points]

1. Implementing and Operating on a Linked List [20 points]

In this assignment, you will implement a singly linked list from scratch. You are not allowed to use the built-in class in Java (java.util.LinkedList). The purpose of this programming task is to implement a linked list manually and understand the fundamentals of node traversal.

Task

- Implement a class named IntegerNode (similar to StringNode discussed in class) to represent a node in a singly linked list. This class should have only two attributes:
    - element (int): The element stored in the node.
    - nextNode (IntegerNode): A reference to the next node in the list.
    - Provide appropriate getter and setter methods for these fields.
- Implement a class named NumLinkedList to represent the linked list using IntegerNode as its nodes. This class should provide the following functionalities:
  Required public methods:
    - *int size()*: Returns the number of elements in the list.
    - *void add(int)*: Adds an element to the end of the list.
    - *boolean isSorted()*: Returns true if the list is in ascending order, false otherwise.
    - *void reverse()*: Reverses the elements of the list in place, using constant space and linear time.
    - *static NumLinkedList merge(list1, list2)*: Combines list1 and list2 into a new NumLinkedList containing all elements from both lists. If both input lists are sorted, the resulting list should maintain sorted order.
    - *static NumLinkedList duplicate(list)*: Returns a copy of the input list with a different address from the original object and different node addresses.

Instructions: Write the IntegerNode and NumLinkedList classes with the specified attributes and methods. Create a test class to demonstrate the various list operations. **Encapsulation** should be taken into account for all methods and fields in both classes.

2. Implementing a QueuedStack<T> Class [15 points]

- Task: Create a class named **QueuedStack<T>** which has a **single** Queue(java.util.Queue<T>) object in its constructor.
- Include the following public methods for stack operations:
  - *T push(T element)*: Method to push an element onto the stack. Returns the element that was pushed.
  - *T pop()*: Method to remove and return the element at the top of the stack.
  - *T peek()*: Method to look at the element at the top of the stack without removing it.
  - *boolean empty()*: Method to check if the stack is empty.

Instructions:
Write a QueuedStack<T> class with the specified constructor and methods, mimicking the behavior of a stack using a **single** queue. Ensure to throw EmptyStackException by importing it from the java.util library where appropriate, referring to the Java Stack class behavior. Create a test class to demonstrate creating a QueuedStack object with different data types, adding some elements, and performing stack operations.

Reference:
For the official Java Stack API, refer to Java Stack Documentation. Note: This reference is for understanding the expected behavior; do not use java.util.Stack in your implementation.

3. Implementing a StackifiedQueue<T> Class [15 points]

- Task: Create a class named **StackifiedQueue<T>** which has **two** Stack (java.util.Stack<T>) objects in its constructor.
- Include the following public methods for queue operations:
  - *boolean add(T element)*: Method to insert an element into the queue. Returns true if the element is successfully added.
  - *T poll()*: Method to remove and return the element at the head of the queue. Returns null if the queue is empty.
  - *T peek()*: Method to look at the element at the front of the queue without removing it. Returns null if the queue is empty.
  - *boolean isEmpty()*: Method to check if the queue is empty.

Instructions:
Write a StackifiedQueue<T> class with the specified constructor and methods, mimicking the

behavior of a queue using stacks. Create a test class to demonstrate creating a StackifiedQueue object with different data types, adding some elements, and performing queue operations.

Reference:
For the official Java Queue API, refer to the Java Queue Documentation. Note: This reference is for understanding the expected behavior; do not use java.util.Queue in your implementation.