TRINITY COLLEGE
DEPARTMENT OF COMPUTER SCIENCE

CPSC 215: Data Structures and Algorithms
Instructor: Dr. Chandranil Chakraborttii
Spring 2025

Assignment 4

Academic Honesty Policy

In working on programming assignments, you may discuss broad issues of interpretation and understanding and general approaches to a solution. However, conversion to a specific solution or to program code must be your own work. Programming assignments are expected to be the work of the individual student, designed, and coded by him or her alone. Violations are easy to identify and will be dealt with promptly according to the Academic Integrity and Intellectual Dishonesty outlined in the Student Handbook.

- Copying another person's programs or encouraging or assisting another person to commit plagiarism is cheating. In particular, the following activities are strictly prohibited:
- Giving and receiving help in the actual development of code or writing of an assignment.
- Looking at another person's code or showing your code to another person.
- Sharing a copy of all or part of your code regardless of whether that copy is on paper or in a computer file.
- Turning in the work of any other person(s) (former students, friends, textbook authors, people on the Internet, etc.) and representing it as your own work.
- Fabricating compilation or execution results.
- Use of AI ChatBots for any help regarding this assignment is **strictly prohibited.**

The penalty for cheating is a failing grade (F) for the course, and the student is asked to appear at an Academic Dishonesty Hearing. In addition, the College also places a record of the incident in the student's permanent record. It is your responsibility to protect your work from unauthorized access. Do not discard copies of your programs in public places. Do not leave computers unattended and copies of output lying around.

Objective:

- To learn how to work with Dequeues.
- To learn how to make a Deque to behave as a Stack or a Queue.
- To gain experience working with Tree data structure.

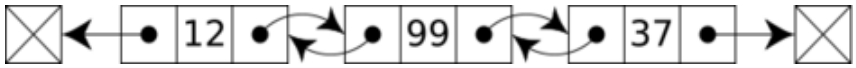
Resources:

Getting Started

- Open Eclipse and create a Java Project (File -> New -> Java Project)
- Give a project name (For example: assignment_4_CPSC_215)
- Go to Project -> Right Click -> New Package
- Give a package name (For example: assignment_4_CPSC215)
- Now to create the classes, go to package -> Right Click -> New Java class.
- Paste contents of the given files. Keep the first line in place (package “package_name”;)
- Understand the code structure before starting to write code.

Part I: DQueue implementation based on Double Linked Lists

A **DQueue**, is a **linear** data structure that permits to insert and to remove elements on both sides. To build it, we will be based on the use of a **Double Linked List (DLL)** on which we will implement the corresponding methods to achieve such behavior.



You will need the following files to get started:

- *DLink.java* *[No need to change]*
- *DQueue.java* *[No need to change]*
- *DLDQueue.java* ***[Need to complete]***
- *DLDTest.java* *[Update as required]*

To implement a Deque by using a DLL, we will perform the following tasks:

1. **Review** the **DLink** class (**DLink.java)** which represents the node of a Double Linked List (DLL). Such node will contain both **prev** and **next** references to the previous and next node of the current one, as well as an Object reference to the data to be stored.
2. **Review** the **DLDQueue** class that will implement a **DQueue (DQueue.java)** based on a DLL. In order to do this, it must implement the **DQueue** interface (Dqueue.java) that has been previously mentioned.
3. **Add** the **head** and **tail** attributes that represents the nodes at both ends of the DLL and the integer attribute size that permits to store the size of it.
4. **Program** the constructor of the **DLDQueue** class that initializes the head and tail references to an empty DLL node and set the size to 0.

Steps 1-4 is already completed for you. Please start from Step 5

5. A **DQueue's** behavior can be achieved by the implementation of the following interface that shows its main methods. Implement the methods of the DQueue's interface:

    - **public void insertFirst(E obj):** This method inserts an object at the beginning of the Double Linked List (DLL).
    - **public void insertLast(E obj):** This method inserts an object at the end of the DLL.
    - **public E removeFirst():** This method extracts an object from the beginning of the DLL and removes it from the list. If such object does not exist, the method returns null.
    - **public E removeLast():** This method extracts an object from the end of the DLL and removes it from the list. If such object does not exist, the method returns null.
    - **public int size():** This method returns the size of the DLL list.
    - **void truncate_first(int num):** This method deletes n elements from the beginning of the DLL list.
    - **void truncate_first(int num):** This method deletes n elements from the beginning of the DLL list.

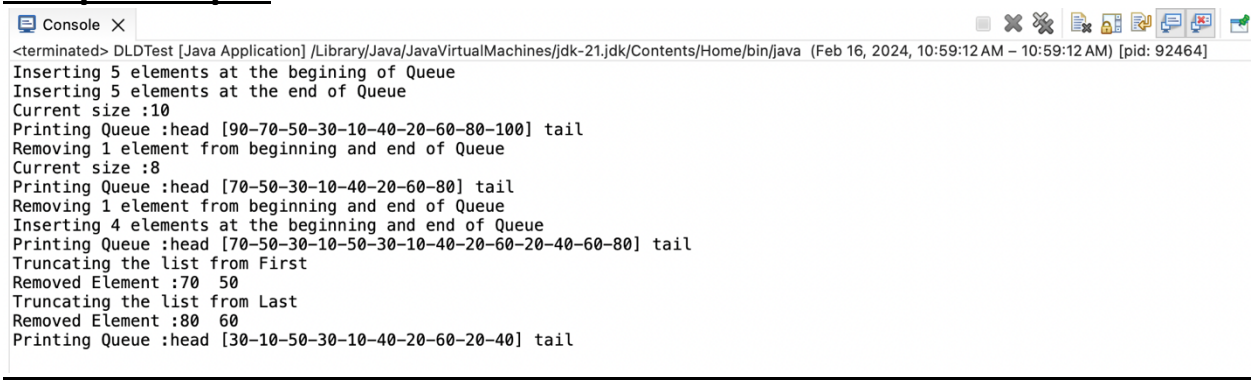
6. Finally, implement the **public String toString()** method that permits to print the content of the Deque on console according the next format (the shown values are examples):

    head [1-2-3-4-5] tail

To do:

- Review the **DLink** class and complete the above methods of the **DLDQueue**
- Use the test class (**DLDTest.java**) to test your program for correctness
- In your test file, you need create objects of **DLDQueue** class and make calls to all the methods described above for testing.
- Add JavaDoc in all the classes.

Sample Output



Part II: Stack and Queue implementations with DQueues

In this exercise, we will take advantage of the flexibility of a DQueue to make it behave as if it was a Stack or a Queue. To build it, we will be based on the use of a Double Linked List (DLL) on which we will implement the corresponding methods to achieve such behavior. For this, we will force the class using the Deque to implement the corresponding interface (Stacks, Queues and Dequeues).

First start by implementing the methods in **DLDQueue.java (to complete)** defined in the Dequeue.java interface. You may take help from Section II of this exercise for completion.

- *DLink.java* *(No need to update)*
- *Stack.java* *(No need to update)*
- *Queue.java* *(No need to update)*
- *DQueue.java* *(No need to update)*
- *DLDQueue.java* *(**Need to complete**)*
- *DQStack.java* *(**Need to complete**)*
- *DQQueue.java* *(**Need to complete**)*
- *Test_Stack.java* *(No need to update)*

First start by implementing the methods in **DLDQueue.java (to complete)** defined in the Dequeue.java interface. You may take help from Part I of this exercise for completion.

Part A: Implement a Stack using a Deque:

- Complete the **DQStack** class that should behave like a Stack. It should implement the Stack interface that has been previously mentioned.
- Add a data attribute of **DLDqueue** type which constitutes the list where stack's data will be stored.
- Program the **DQStack** class constructor that should initialize the data attribute creating an instance of a **DLDQueue** class.
- Program the following methods of the Stack's interface:

    - **public void push(Object obj):** This method inserts an object at the beginning of the DLDqueue.
    - **public E pop():** This method extracts an object from the beginning of the DLDqueue and it removes it from the list. If such object does not exist, the method returns null.
    - **public int size():** This method returns the size of the DLDqueue.
    - **public String toString():** Returns a string representation of the stack

Part B: To implement a Queue using a DLDqueue:

- Declare the **DQQueue** class that should behave like a Queue. In order to do this, it should implement the Queue interface that has been previously mentioned.
- Add a data attribute of **DLDQueue** type which constitutes the list where queue's data will be stored.
- Program the **DQQueue** class constructor that should initialize the data attribute creating an instance of a **DLDQueue** class.
- Program the following methods of the Queue's inteface:

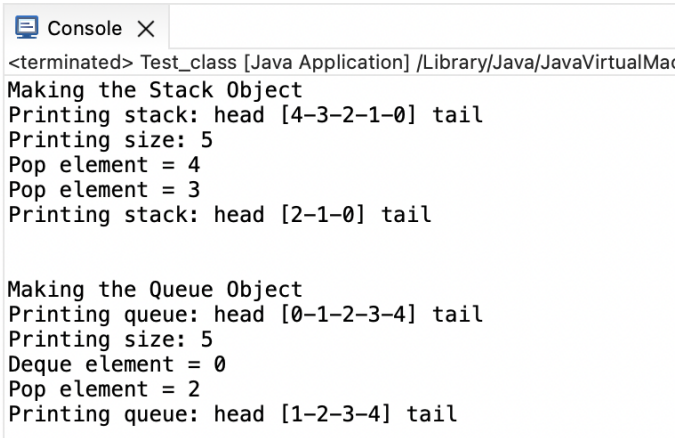
    - **public void enqueue(Object obj):** This method inserts an object at the beginning of the DLDqueue.
    - **public E dequeue():** This method extracts an object from the end of the DLDqueue and it removes it from the list. If object does not exist, method returns null.
    - **public int size():** This method returns the size of the DLDqueue.

Review the main method (in Test_class) that instantiates a **DQStack** object, pushing and popping several data while it prints on console the content and the size of the stack. The main method also instantiates a **DQQueue** object, queueing and enqueueing several data while it prints on console the content and the size of the queue.

To do:

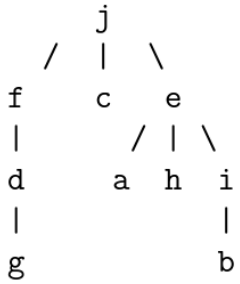
- Review the **DLink** class and complete the above methods of the **DLDQueue**
- Implement the methods in DQStack and DQQueue classes
- Please use the Test file (Test_class.java) to check the output. Verify and test your program for correctness.
- Add JavaDoc in all the classes.

Sample Output



Part III: Working with Trees

Problem 1: Print the visited the sequence of nodes for the tree below for a post-order and pre-order traversal. **[Show Steps]**



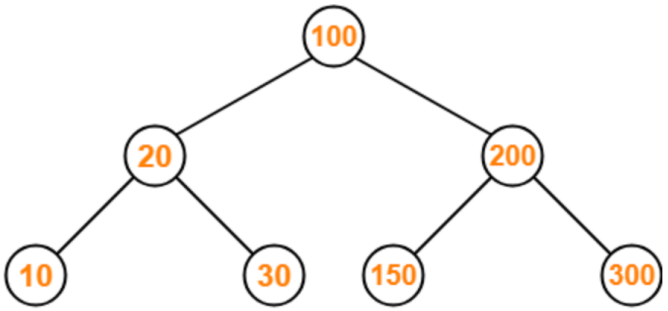
Problem 2: Consider the polynomial $5y^2 - 3y + 2$.

a) Write the polynomial as an expression tree that obeys the usual ordering of operations. ***Hint****: To clarify the ordering for yourself, first write the expression with brackets indicating the order of operations.* **[prefix expression]**
b) Write the polynomial as **postfix expression.**

Problem 3: Draw a binary tree T that simultaneously satisfies the following. Justify your answer.

a) Each internal node of T stores a single character.
b) A preorder traversal of T yields EXAMFUN.
c) An in-order traversal of T yields MAFXUEN.

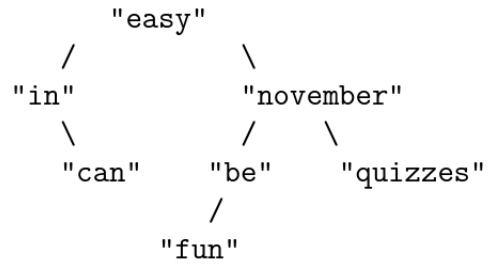
Problem 4: Consider the following binary search tree.



Binary Search Tree

Write the order of nodes visited in a pre-, in- and post-order traversal.

Problem 5: Consider the binary tree (with null child references not shown):



a) What is the order of nodes visited in a pre-order traversal?
b) What is the order of nodes visited in a post-order traversal?
c) Transform this binary tree into a binary search tree of height 2, defined by the natural ordering on strings.

Problem 6: Consider a binary tree:

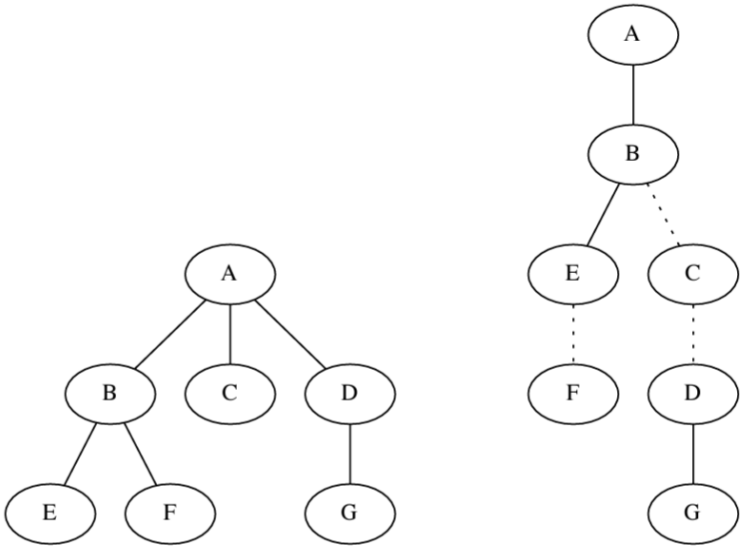
a) Draw the binary tree given the in-order traversal is DBEAFC and the pre-order traversal is ABDECF.
b) Write down the post-order traversal of this tree?
c) Draw all binary search trees of height 2 that can be made from all the letters ABCDEF, assuming the natural ordering.

Problem 7: Let T be a tree with more than one node. Is it possible that the preorder traversal of T visits the nodes in the same order as the post-order traversal of T? If so, give an example; otherwise, argue why this cannot occur. Likewise, is it possible that the preorder traversal of T visits the nodes in the reverse order of the post-order traversal of T? If so, give an example; otherwise, argue why this cannot occur?

Problem 8: We can define a binary tree representation T' for an ordered general tree T as follows:

- For each node u of T, there is an internal node u' of T' associated with u.
- If u is an external node of T and does not have a sibling immediately following it, then the children of u' in T' are external nodes.
- If u is an internal node of T and v is the first child of u in T, then v' is the left child of u' in T.
- If node v has a sibling w immediately following it, then w' is the right child of v' in T'.

Note: *In a binary tree, an external node is a node that does not have any child nodes. External nodes are also known as leaf nodes.*



(a) T (b) T'

Figure: Representation of a tree with a binary tree:

a) tree T;
b) binary tree T' for T. The dashed edges connect nodes of T' that are siblings in T.

Given such a representation T' of a general ordered tree T, answer each of the following questions and justify your answer.

a) Is a preorder traversal of T' equivalent to a preorder traversal of T?
b) Is a postorder traversal of T' equivalent to a postorder traversal of T?
c) Is an inorder traversal of T' equivalent to one of the standard traversals of T? If so, which one?

Submission Guidelines:

- Complete methods as instructed. Add JavaDoc in your code (Part I and II).
- Upon the completion of your lab, arrange each section into individual folders (PI, PII, PIII), then compress these folders into a single Zip file. Upload the consolidated submission to the Moodle course website.
