Birla Institute of Technology and Science, Pilani Hyd Campus
BITS F232: Foundations of Data Structures and Algorithms
1st Semester 2023-24 Lab No: 8
Iterators, Sequences and Tree Traversals

General Instructions

- You should use STL only for the program where it is mentioned explicitly. For other programs you should solve the given task without using it.
- In addition to the methods already specified in the given code, you may also come up with your own methods to execute the tasks.

Program 1: (Prog1.cpp attached)

Iterators are very handy tool to iterate over the given container sequence whenever needed. The main advantage of iterators is "flexibility". Each time new items are added or removed from the Container; we need not maintain any special 'size' variable to loop over. Iterators help us avoid such overhead with clean and readable code.

Given a **list** of marks of a student in **N** subjects. Prog1.cpp given stores these marks in a list. It then uses Iterators to traverse each element and prints those (3rd line in the below output). Then it traverses the list and checks if each element is odd/ even and prints those. Finally, updates the input marks by adding each element with the number 10, and prints those out. Write down the missing code (A to E) in Prog1.cpp and run it to get similar output as shown below:

Output:

```
Enter the number of subjects: 4
Enter the marks in 4 subjects: 20 17 30 13
Input Marks: 20 17 30 13
20 = Even
17 = Odd
30 = Even
13 = Odd
Updated Marks: 30 27 40 23
```

Program 2: (Prog2.cpp attached)

In the lecture classes, you were taught about Bubble Sort in Sequences along with its' time complexity. Each sequence container will have its own time complexity for sorting. Given a **doubly linked list**, Prog2.cpp implements swap() method to swap two adjacent nodes in a doubly linked list. It also contains the **bubbleSort()** method to implement bubble sort algorithm over a doubly linked list. This should be done in O(n^2) time as this is a doubly linked list. Try to use the swap () method wherever needed. Try to follow along the comments for writing the missing code to get the below output. If you want to write your own logic, you are welcome to do so. The driver code is already available in the code itself. You should get the output as shown below and also run your code on other test cases:

Output:

```
Before Sorting:
NULL <- 20 ->  <- 30 ->  <- 15 -> NULL
After Sorting:
NULL <- 15 ->  <- 20 ->  <- 30 -> NULL
Before Sorting:
NULL <- 15 ->  <- 20 ->  <- 30 ->  <- 10 ->  <- 25 ->  <- 35 -> NULL
After Sorting:
NULL <- 10 ->  <- 15 ->  <- 20 ->  <- 25 ->  <- 30 ->  <- 35 -> NULL


...Program finished with exit code 0
Press ENTER to exit console.
```

Program 3: (Prog3.cpp attached)

This program is about a **Non-Linear Data Structure, i.e. Tree.** Here is a Binary Tree is given for you. Go through the attached code and understand how recursive methods are created to do basic operations on binary tree. For Non-Linear data structures, recursive codes are a lot cleaner and easily understandable. Here are your tasks:

- **Task 1**: Implement recursive preorder traversal.
- **Task 2**: Implement recursive postorder traversal.

In-order traversal code is given in the program which will be discussed in the next theory class.The driver code is already available in the code itself. You should get the output as shown below and run it with different test cases and draw the trees:

Output: