Programming Assignment #2
Incentive Date: Feb 13th 2024
Due Date: Feb 16th 2024
Deque

Description:
For this programming assignment, you will implement a **Deque** whose size can grow as elements are inserted into the deque. Specifically, you will implement a deque using a doubly linked list.

starter.zip (Starter Code)
The following is a brief description of the starter code provided to you:

- **node.h**
  - This header file declares and defines a fully functional **Node class.** The Node class contains the following three private member variables:
    - **Type data:** This will store the data that the Node is instantiated with (Type is a template).
    - **Node<Type>* next:** This will be the pointer to the next node in the doubly linked list.
    - **Node<Type>* prev:** This will be the pointer to the previous node in the doubly linked list.
  - You have been provided with implemented setters and getters in the Node class which modifies and obtains the value of these member variables respectively.
  - **No changes to this file are necessary.**
- **deque.h**
  - This header file declares the **non-functional Deque class.** The Deque class contains the following three private member variables:
    - **int s:** This keeps track of the size of the doubly linked list (and thereby the deque).
    - **Node<Type>* firstNode:** This will be the pointer to the head of the doubly linked list.
    - **Node<Type>* lastNode:** This will be the pointer to the tail of the doubly linked list.
- **main.cpp**
  - The starter main.cpp has been populated with a few sample test cases which tests the insertion to the front and the back of the deque.
    - You can use this file to instantiate Deque objects and test all Deque methods comprehensively.

Contract (TASK):
Your task is to implement the methods declared in the Deque class. You have been provided with non-functional definitions of these methods in Deque.h. **You must implement these method definitions in the given header file,** by replacing the current non-functional definitions with your functional implementations. **Do not create an additional cpp file to define these functions, make sure that they are part of the Deque.h file.**

The following are the methods that must be implemented in the Deque class:

- **Constructor**
- **Destructor**
- **Copy Constructor**
- **Copy Assignment**
- **bool isEmpty**
  - Must return true if the deque is empty and false otherwise
- **int size**
  - Returns the size of the deque
- **Type first**
  - Returns the element present at the front of the deque
- **Type last**
  - Returns the element present at the back of the deque
- **void insertFirst**
  - Pushes a new element to the front of the deque
- **void insertLast**
  - Pushes a new element to the back of the deque
- **Type removeFirst**
  - Removes the element present at the front of the deque and returns this element
- **Type removeLast**
  - Removes the element present at the back of the deque and returns this element

[Advisory]:

- You must throw a **std::out_of_range** exception when you come across invalid calls to the removeFirst and removeLast methods
  - You do not need to throw this exception on invalid calls to the first and last methods. Incase of an invalid call to the first and last methods, you can simply "**return Type()**" as it is given in the starter code.

Deliverables and Submissions:

In a zip folder named using the format **<FirstName>-<LastName>-<UIN>-<PA2>.zip** you must submit the following file(s) to Canvas:

- **deque.h** (containing the functional method definitions of the Deque class)

Do not use angular brackets in the name of the submission folder.

Grading:

This assignment is worth 80 points. Each method of the Deque will be tested using an automatic testing infrastructure. **The testing infrastructure for this assignment will not be revealed, so please make sure to test your code comprehensively before submitting.**

The grading rubric is as follows:

- Copy Constructor: 10 points
- Copy Assignment: 10 points
- Insert First & Remove First: 15 points
- Insert Last & Remove Last: 15 points
- Insert & Remove: 25 points
- Remove (First & Last) Exception Case: 5 points

All the test cases above sum up to 80 points. In case of memory leaks in your program, we will subtract 10 points.

Note: There is a possibility that one or more of your methods causes a segmentation fault which in turn causes the test script to crash when calling that particular method. In such a case, we will not be able to calculate your score for the assignment and you will be assigned a default grade of 0. After the grades are posted, you will be given 1 week to rectify your error(s) and resubmit the assignment. You will be subject to a 25% penalty on the grade obtained from the resulting resubmission of the assignment.

How to test your program natively (run your own test cases):
You can use main.cpp provided in the starter.zip folder to instantiate Deque objects. You can then test the functionality of your methods by calling these methods and checking whether what they return is what you expect.
To compile your own tests in main.cpp, **you can use the makefile provided within starter.zip to compile your program.** In your terminal:

- Run **make** on the terminal (this will create an executable named main)
- Next, run **./main** on your terminal to run your native tests
- Finally run **valgrind --leak-check=full ./main** to check whether there are memory leaks.