LAB 2: Doubly-linked list

Doubly-linked list Lab Specification

Implement a doubly-linked list with dummy head and tail nodes to store integer values. Unlike singly-linked list which only allows forward navigation, a doubly-linked list allows you to navigate the list in both forward and backward directions. A node in a doubly-linked list stores two pointers, prev and next. prev points to the previous node in the list and next points to the next node in the list. The header file (IntList.h) should declare and implement the IntNode struct (just copy it exactly as it is below) as well as the IntList Class interface only. You are also required to come up with a separate implementation file (IntList.cpp) that implements the member functions of the IntList class.

IntNode struct

I am providing the IntNode class you are required to use. Place this class definition within the IntList.h file exactly as is. Make sure you place it above the definition of your IntList class. Notice that you will not code an implementation file for the IntNode class. The IntNode constructor has been defined inline (within the class declaration). Do not write any other functions for the IntNode class. Use as is.

```
struct IntNode {
    int data;
    IntNode *prev;
    IntNode *next;
    IntNode(int data) : data(data), prev(0), next(0) {}
};
```

IntList class

To avoid dealing with special cases, it is usually more convenient to add two dummy nodes at both end of a doubly-linked list. Dummy node does not contain any data. A dummy head node precedes the first node in the list and dummy tail follows the last node in the list.

Encapsulated (Private) Data Fields

- dummyHead: IntNode *
- dummyTail: IntNode *

Public Interface (Public Member Functions)

- **IntList()**: Initializes an empty list with dummy head and dummy tail.
- **~IntList()**: Deallocates all remaining dynamically allocated memory (all remaining IntNodes).
- **void push_front(int value)**: Inserts a data value (within a new node) at the front end of the list. This must be an O(1) operation.
- **void pop_front()**: Removes the node at the front end of the list (the node after the dummy head). Does nothing if the list is already empty. This must be an O(1) operation.
- **void push_back(int value)**: Inserts a data value (within a new node) at the back end of the list. This must be an O(1) operation.
- **void pop_back()**: Removes the node at the back end of the list (the node before the dummy tail). Does nothing if the list is already empty. This must be an O(1) operation.
- **bool empty() const**: Returns true if the list does not store any data values (it only has dummy head and dummy tail), otherwise returns false.
- **friend ostream & operator<<(ostream &out, const IntList &rhs)**: A global friend function that outputs to the stream all of the integer values within the list on a single line, each separated by a space. This function does **NOT** send to the stream a newline or space at the end.
- **void printReverse() const**: prints to a single line all of the int values stored in the list in **REVERSE** order, each separated by a space. This function does **NOT** output a newline or space at the end.