Lab 6 [ECE 2620 (C++, Data Structures & Algorithms)]

Doubly Linked Lists, Queues and Stacks

Problem Description and Design

In this lab exercise, you will write code for a **doubly linked list (DLL)** that stores an integer value in each list node. We will then use this code to implement four specific abstract data structures: a **simple list** (unsorted), a **sorted list**, a First In First Out (**FIFO**) **queue** and a Last In First Out (**LIFO**) **queue**, also called a '**stack**'. To keep things simple, our data structures will hold only **int** type data values (in addition to our pointers, of course). Templates *are not* required. The specific goals are listed below.

Your job is to do the following:

1. Start a new project and write the class code for the classes `intDLLNode` and `intDLList`. These classes are similar to the classes `intSLLNode` and `intSLList` (*refer to your lecture slides*), with similarly named member functions. However, since we are constructing a DLL instead of an SLL, the code for the member functions will of course, be different. Thus, your DLL will have the same member functions as in `intSLLNode` and `intSLList` but you will be writing your own code for these member functions!

   a. In the definition of class `intDLLNode`, when writing the constructor, use a single default constructor of type ***C*** (*refer to your lecture slide set 2, pages 18, 24-25*) . Also, make sure that you `throw` exceptions in the member functions `deleteFromHead()` and `deleteFromTail()`, if an attempt is made to delete a node from an empty list. And **do not forget** to write the corresponding `try-catch` clause(s) as well, or your program will crash when an exception gets thrown. In addition, if any `new` call fails, then **display an appropriate error message and terminate the program**.

   b. Whenever setting or checking for a null pointer, use the keyword `nullptr`. For instance:
   `if (p != nullptr) { .. };` or `q = nullptr;`

   c. Write the following *additional member functions:*
   c.1 '`addSorted(int i)`' in class `intDLLList`. This member function should insert an integer element in *sorted order* within the list. (You can assume that this member function will be used exclusively for adding items into a list, when we require a sorted list. Thus, a sorted list, if created, will always be kept in sorted order after its creation.

   c.2 '`printlist()`' in class `intDLLList`. This member function should be able to print all the integer elements from the list, in order.

   Make sure to test these member functions thoroughly so that your program does not crash due to misdirected pointers.

2. Write a C++ driver program `main()` that does the following:

   a. When the program starts, it provides the user with 5 main options:
   (1) Create Simple (Unsorted) List
   (2) Create Sorted List
   (3) Create Queue (FIFO)
   (4) Create Stack (LIFO)
   (5) Exit program

b. If option "1" is selected in step **2.a** above by the user, then the user is given the following 7 choices repeatedly:

(1) Enter integer for insertion at head of list
(2) Enter integer for insertion at tail of list
(3) Display and delete integer from head of list
(4) Display and delete integer from tail of list
(5) Search for integer in list, and delete that node
(6) Display contents of list from head to tail, in order
(7) Exit program

c. If option "2" is selected in step **2.a** above by the user, then the user is given the following 4 choices repeatedly:

(1) Enter integer for sorted insertion (increasing order) into list
(2) Search and delete integer, if present in list
(3) Display contents of sorted list of integers, in increasing order
(4) Exit program

d. If option "3" is selected in step **2.a** above by the user, then the user is given the following 4 choices repeatedly:

(1) ENQUEUE (Enter integer for insertion into queue)
(2) DEQUEUE (Display and delete integer from queue)
(3) PRINT QUEUE (Display queue contents without deleting anything, first element first)
(4) Exit program

e. If option "4" is selected by the user, then the user is given the following 4 choices repeatedly:

(1) PUSH (Enter integer for insertion into stack)
(2) POP (Display and delete integer from stack)
(3) PRINT STACK (Display stack contents without deleting anything, last element first)
(4) Exit program

f. The "Exit Program" option allows a user to terminate the program from any menu screen as shown above.

Your driver program `main()` should have **absolutely no pointer manipulations**. All pointer code should be kept in a separate class file (or files) similar to the suggestions made in Figure 3.2 for an SLL in your textbook. However, be sure to name the files appropriately. For example, the header file for the `intDLList` class should be named `intDLList.h` and the one for the `intDLLNode` class should be named `intDLLNode.h`; the source code for the corresponding member functions must be placed in `intDLList.cc` and `intDLLnode.cc`, respectively.

NOTES:

1. Document your code well. Use indentation wherever appropriate to enhance readability.
   Points will be deducted for missing or shoddy documentation.
2. Keep your source program neat and modular.
   Points will be deducted if your code is not modular and if you make unnecessary use of global variables.
3. Start every program with

```
//Your name here
//Course + section number
//  Anything else you may wish to tell us
```

   These comments at the very beginning of your program, help us to identify you, and see if you

have anything to say to me such as instructions on how to run the program, anything that does not work in your program, anything that does work(!), etc.