Lab 7: Review Lab 5 (Doubly-linked Lists)

You have been assigned another student's lab to review. This is what you will need to do:

1. Go to Canvas to find which project you need to review. You will write your comments in the appropriate space.

2. Follow the **rubric** below, **along with the solution posted**, to verify that the implementation is correct.

3. **Create a new project** and add the **given files** (**DoublyList.h**, **DoublyList.cpp** and **Main.cpp**) along with the **Functions.cpp** file to test the program as follows:

    - If the program does **NOT** run:
        - Check all **function headers**:
            - If missing **const** and/or **&**, **correct the header** and run the program.
        - File not named **Functions.cpp**
            - Re-name the file.
        - For other errors, no need to correct the code; simply proceed on reading the code.

        - *For ANY of the above*, make sure you **write the appropriate comment**.

    - **Output:**
        - For each output that is not as expected ➔ Clearly explain what the error is.

Name header	There is no name header. Name header is not at the top of the file. Incorrect name header. *Explain what is missing/incorrect.*
Function **print**	`void DoublyList::print() const {…}` (The function header **must** be as indicated above.) Checking if the list is empty is inefficient, because there is the assumption that the calling object is non-empty. There should be a **WHILE** loop, rather than a FOR loop, with a pointer that stops when it becomes a nullptr.
Function **reversePrint**	`void DoublyList::reversePrint() const {…}` (The function header **must** be as indicated above.) Checking if the list is empty is inefficient, because there is the assumption that the calling object is non-empty. There should be a **WHILE** loop, rather than a FOR loop, with a pointer that stops when it

	becomes a nullptr.
Function **front**	`int DoublyList::front() const {…}` (The function header must be as indicated above.) Checking if the list is empty is inefficient, because there is the assumption that the calling object is non-empty. This should be a one-statement function.
Function **back**	`int DoublyList::back() const {…}` (The function header must be as indicated above.) Checking if the list is empty is inefficient, because there is the assumption that the calling object is non-empty. This should be a one-statement function.
Function **copyToList**	`void DoublyList::copyToList(DoublyList& identifier) const {…}` (The function header must be as indicated above.) The identifier is not descriptive. The identifier does not follow the camelCase convention. It is inefficient to re-write a lot of code instead of using the function insertFront. There should be a **WHILE** loop, rather than a FOR loop. Function **insertFront** is not used. Since the instructions ask to use the function **insertFront**, the best way to copy the list is by starting from the last node of the calling object and keeping inserting to the front of the parameter object. *(Check my solution to compare the implementation.)* **NOTE:** The instructions assume that the parameter object was empty, but it asked to test also a non-empty parameter object. Omit the latter. The function is correct as long as it works for these two scenarios: • Calling object is empty and parameter object is empty. • Calling object has elements and parameter object is empty.
Function **insertInOrder**	`void DoublyList::insertInOrder(int identifier) {…}` (The function header must be as indicated above.) There should be a sequence of IF/ELSE statements, rather than a sequence of IF statements. The case when the list is empty was not considered.

	The case when the new node should be inserted to the left of the first node was not considered. The case when the new node should be inserted somewhere in the middle or at the end of the list was not considered. Variable count is not incremented.
Other errors	Other items include whether or not the implementation is efficient, readable, follows the standards presented in class. Variables are not initialized ➔ *Specify which ones.* Variables are not declared right before using them ➔ *Specify which ones.* Poor indentation. Unnecessary white space. Statements are too long and force viewer to scroll horizontally. Use of more than one statement when it can be a single statement. There are no spaces to the left and the right of operators.
Other	For errors that are not listed in the rubric provide a **clear explanation** of **why** it is incorrect. You may also leave "**suggestions**" to improve the implementation; do **not** detract points for given suggestions.