**Assignment 8 – Containers**

Submit to MUOnline as a compressed (.zip) file containing your code project.

|  |  |  |
| --- | --- | --- |
| 1. | Using the LinkedList we’ve made in class, implement and demonstrate these LinkedList functions:  **Insert**  **Purpose:** Add an element anywhere in the linked list.  **Input(s):** A value to insert, and a value to search for.  **Process:** Walk through the list looking for the specified value. If the value to search for is found, insert the new value to its left; if the value to search for is not found, push the new value to the back of the list.  **Remove**  **Purpose:** Remove an element from anywhere in the linked list.  **Input(s):** A value to search for and remove.  **Process:** Walk through the list looking for the specified value. If found, remove it from the list so that the node before it points to the node after it, and the node is deallocated properly. If the value isn’t found it does nothing.  Handle all cases appropriately, making sure that head and tail and each node’s ‘next’ pointer are updated. For example, remove needs to do different things if the value is at the front or back of the list, or if it’s the only value in the list. Using other linked list functions in any combination before or after them shouldn’t break. | 50% |
|  |  |  |
| 2. | Write and demonstrate a function that takes as input a string containing an expression such as “(2 \* 2) + 2”. The function should return whether or not (true or false) the expression contains matching grouping symbols. If you use user input and include spaces in your expression, remember to use getline().  ‘Matching grouping symbols’ just means whether every opening parenthesis or bracket has a corresponding closing one. It’s the exact same technique Visual Studio uses to tell you when you are missing a parenthesis somewhere in your code.  For example:  7 + 8 \* 2 => true  {25 + (3-6) \* 8} => true  5 + {(13+7) / 8 – 2 \* 9 => false (missing ‘}’)  (2 + {3 \* 4) + 1} => false (right number, wrong order) | 50% |
|  | To do this, use a stack<char> (stack of characters). Loop through the string, and as you encounter the opening symbols (, {, and [, push them on the stack. The most recent one encountered will be on top of the stack, so as you encounter the closing symbols ), }, or ], check if the top of the stack is the corresponding opening symbol. Remember you can’t use stack.top() or stack.pop() if the stack is empty (it will throw an exception). |  |