Test 1
Weeks 1-4

Modify the initial code (attached) to finish the following questions. Student must not change the function prototype.

Question 1.
Write a function to insert a new node into the linked list after the first N nodes of the linked list. There are some note for this questions:

- If N = 0 then insert the new node as head of the linked list.
- If N is greater than the number of nodes in the linked list, insert the new node at the end of it.

For example, given the linked list which a is the head node. Calling insertList(a, 4, 3) would add a new node with data = 3 after the 4th node of the linked list:

```
PrintLinkedList(a); // Output: 9 12 4 5 7
insertList(a, 4, 3);
PrintLinkedList(a); // Output: 9 12 4 5 3 7
```

Question 2.
Assume that we have two ordered list a and b arranged in descending order. Write function `void ConcatenateOrderedList(Node<T> *&a, Node<T> *&b)` concatenating two lists to create a new ordered list. After executing this function, a will point to this new list and b will point to NULL. Note: Student must not use sorting algorthims.
For example,

```
PrintLinkedList(a); // Output: 7 5
PrintLinkedList(b); // Output: 9 6 1
ConcatenateOrderedList(a, b);

PrintLinkedList(a); // Output: 9 7 6 5 1

if (b != NULL) {
  cout << "This function does not match the requirement that b
      will point to NULL";
}
```

Question 3.
Develop function `Node<T>* DivisionPoly(Node<T> *PolyList1, Node<T> *PolyList2)` implementing division of two polymonials.

For convenience, you may assume that PolyList2 is a factor of PolyList1. Moreover, there would be no rounding required when performing the division among the coefficients (i.e the coefficients are always divisible in the division).
For example,

```
PrintLinkedList(PolyList1); // Output: 1 2 1
PrintLinkedList(PolyList2); // Output: 1 1

Node<int> *dividedPoly = DivisionPoly(PolyList1, PolyList2);
PrintLinkedList(dividedPoly); // Output: 1 1
```

Question 4.

Write function `int EvaluatePrefix(char* s)` that evaluates a prefix expression (assume it's valid) using stacks. There are some assumptions for this questions:

- Operands are integer number.
- Operators include '+', '-', '*', '/'.

For example,

```
cout << EvaluatePrefix("+ 3 4"); // Output: 7
cout << EvaluatePrefix("* + 4 2 5"); // Output: 30
cout << EvaluatePrefix("/ 4 2"); // Output: 2
cout << EvaluatePrefix("- 4 2"); // Output: 2
cout << EvaluatePrefix("- * 5 6 + 4 2"); // Output: 24
```