Programming Section

All classes and members have been declared. You are only allowed to fill in the implementation of the given methods to meet the requirements. Download the base project at:

https://github.com/felixsoum/420J13AS-MidtermA

6. Insertion Sort

a) **(10 points)** Implement a method which sorts an array in **descending** order using insertion sort.

Sample Input:

```
int[] inputA = new int[] { 3, 7, 4, 2, 8, 5, 7, 1, 6, 9, 7, 0 };
DescendingInsertionSort(inputA);
Console.WriteLine(string.Join(", ", inputA));
```

Sample Output:

```
9, 8, 7, 7, 7, 6, 5, 4, 3, 2, 1, 0
```

b) **(10 points)** Lucky sort is a sorting algorithm which uses insertion sort to sort an array of integers in **increasing** order. Additionally, it takes a lucky number as an input which is a number that must take precedence above all other numbers (the lucky number will move to the start of the array). Implement lucky sort as a variant of insertion sort and taking into account this lucky number input.

Sample Input:

```
int[] inputB = new int[] { 3, 7, 4, 2, 8, 5, 7, 1, 6, 9, 7, 0 };
LuckySort(inputB, 7);
Console.WriteLine(string.Join(", ", inputB));
LuckySort(inputB, 8);
Console.WriteLine(string.Join(", ", inputB));
```

Sample Output:

```
7, 7, 7, 0, 1, 2, 3, 4, 5, 6, 8, 9
8, 0, 1, 2, 3, 4, 5, 6, 7, 7, 7, 9
```

7. Data Structure

a) **(10 points)** Implement class **MyLinkedList<T>** which implements the List ADT. The list is **singly** linked. Implement these methods:

- *InsertFirst(x)* which inserts a new node at the head, replacing the head
- *DeleteFirst()* which deletes the head node, setting the next node as head

Sample Input:

```
myLinkedList.InsertFirst(new MyLinkedListNode<int>(1));
myLinkedList.InsertFirst(new MyLinkedListNode<int>(2));
myLinkedList.InsertFirst(new MyLinkedListNode<int>(3));
Console.WriteLine(myLinkedList);
myLinkedList.DeleteFirst();
Console.WriteLine(myLinkedList);
```

Sample Output:

```
3, 2, 1,
2, 1,
```

b) **(10 points)** Implement class **MyLinkedStack<T>** which implements the Stack ADT. For storage, you must only use the **MyLinkedList<T>** class. Implement these methods:

- *Push(x)* which inserts a value at the top of the stack
- *Pop()*, which removes and returns the value at the top of the stack
- *Peek()*, which returns without removing the value at the top of the stack
- *Count()*, which returns the count of elements in the stack

Sample Input:

```
myLinkedStack.Push("Button");
myLinkedStack.Push("Mutton");
myLinkedStack.Push("Sutton");
string output = "";
output += myLinkedStack.Count() + ", ";
output += myLinkedStack.Pop() + ", ";
output += myLinkedStack.Peek() + ", ";
output += myLinkedStack.Pop() + ", ";
output += myLinkedStack.Pop() + ", ";
output += myLinkedStack.Count() + ", ";
Console.WriteLine(output);
```

Sample Output:

```
3, Sutton, Mutton, Mutton, Button, 0
```

Bonus: Implement the method *Reverse()* which reverses the order of elements in the stack **in-place** and in **linear time**. Define your own console test for the method.