Linked Stack Push Operation

Assume that we have the following lines of code:

```cpp
Stack<int> stack1;  // Line 1
stack1.push(5);    // Line 2
stack1.push(8);    // Line 3
stack1.push(3);    // Line 4
```

The following sequence of diagrams shows how the Stack object and its associated dynamic storage changes as these lines are executed.

**Figure 1:** The new, empty Stack object `stack1` created in Line 1 of the code above. The stkTop pointer is nullptr while stkSize is 0.

```
stkTop   X
stkSize  0
```

**Figure 2a:** The call to `push()` in Line 2 causes a new list Node to be allocated using the temporary pointer `newNode`. The node's data field is initialized to the value passed to `push()`, while its next field is initialized to the current value of stkTop.

```
stkTop   X
stkSize  0
```
```
newNode
```
```
5     X
```

**Figure 2b:** The pointer stkTop is set to point at newNode and the stkSize is incremented to 1.

```
stkTop   •
stkSize  1
```
```
newNode
```
```
5     X
```
Figure 2c: When the `push()` method ends, the local variable `newNode` ceases to exist.

Figure 3a: The call to `push()` in Line 3 causes a new list `Node` to be allocated using the temporary pointer `newNode`. The node’s `data` field is initialized to the value passed to `push()`, while its `next` field is initialized to the current value of `stkTop`.

Figure 3b: The pointer `stkTop` is set to point at `newNode` and the `stkSize` is incremented to 2.

Figure 3c: When the `push()` method ends, the local variable `newNode` ceases to exist.
Figure 4: Linked stack following the call to `push()` in Line 4.

```
stkTop
stkSize 3
```

```
3 → 8 → 5 X
```

**Linked Stack Pop Operation**

Assume that we then add the following lines of code after the code listed above:

```c
stack1.pop();       // Line 5
stack1.pop();       // Line 6
stack1.pop();       // Line 7
```

The following sequence of diagrams shows how the `Stack` object and its associated dynamic storage changes as these lines are executed.

**Figure 5a:** The call to `pop()` in Line 5 creates the temporary pointer `delNode` and sets it to the value of `stkTop`.

```
stkTop
stkSize 3
```

```
3 → 8 → 5 X
```

```
stkTop
stkSize 3
```

```
3 → delNode
```

**Figure 5b:** The pointer `stkTop` is set to `stkTop->next`. It now points to the 2nd node in the list.

```
stkTop
stkSize 3
```

```
8 → 5 X
```

```
stkTop
stkSize 3
```

```
3 → delNode
```
Figure 5c: The node pointed to by delNode is deleted and stkSize is decremented to 2.

![Diagram showing deletion of a node and decrement of stkSize]

Figure 5d: When the pop() method ends, the local variable delNode ceases to exist.

![Diagram showing pop() method end]

Figure 6a: The call to pop() in Line 6 creates the temporary pointer delNode and sets it to the value of stkTop.

![Diagram showing creation of delNode]

Figure 6b: The pointer stkTop is set to stkTop->next. It now points to the 2nd node in the list.

![Diagram showing stkTop set to the 2nd node]
Figure 6c: The node pointed to by `delNode` is deleted and `stkSize` is decremented to 1.

Figure 6d: When the `pop()` method ends, the local variable `delNode` ceases to exist.

Figure 7a: The call to `pop()` in Line 7 creates the temporary pointer `delNode` and sets it to the value of `stkTop`.

Figure 7b: The pointer `stkTop` is set to `stkTop->next`. It is now `nullptr`. 
Figure 7c: The node pointed to by delNode is deleted and stkSize is decremented to 1.

Figure 7d: When the pop() method ends, the local variable delNode ceases to exist. The stack is now empty.