Programming Assignment #4

This assignment gives you more practice with classes, constructors, overloaded operators, pointers, and dynamic memory allocation/deallocation. The goal is to implement a class called List which encapsulates a single-linked list structure. There are several methods that manipulate the nodes in the linked list. These methods include adding items to either end, removing an item from the front, copying lists, concatenating lists, etc Each node (*Node*) in a list contains an integer and a pointer to another Node. This node structure is intentionally kept simple so you can focus on the list aspect of this assignment.

Much of the work required has already been completed in the lab sessions, so this should not be a challenging assignment.

This is the interface to the class:

```
class List
{
  public:
      // Three constructors
      // Destructor

      // Seven methods:
      //   push_front, adds the item to the front of the list
      //   push_back, adds the item to the end of the list
      //   pop_front, removes the first item in the list
      //   front, returns the first item in the list
      //   size, returns the number of items in the list
      //   empty, returns true if empty, else false
      //   clear, clears the list by removing (deleting) all of the nodes

      // Five operators:
      //   operator=
      //   operator+=
      //   operator+
      //   operator[] (2 of these)

      // Returns the number of Lists that have been created
    static int list_count();
      // Returns the number of nodes still alive
    static int node_count();
      // Operator for printing lists (<<)
    friend std::ostream & operator<<(std::ostream & os, const List &list);

  private:
    struct Node
    {
      Node(int);              // constructor
      ~Node();                // destructor
      Node *next;             // pointer to the next node in the list
      int data;               // the data stored in the node
      static int nodes_alive; // count of nodes still allocated
    };

    Node *head_; // pointer to the first node in the list
    Node *tail_; // pointer to the last node in the list
    int size_;   // number of items on the list

    static int list_count_;         // number of Lists created
    Node *new_node(int data) const; // allocate node, initialize data/next
};
```

The class contains 19 functions that need to be implemented. (One of them is already implemented.) Many of the functions will call other functions you've implemented (code reuse), so the amount of code is not that great. The "worker" functions are `push_back`, `push_front`, and `pop_front`. Once these functions are implemented and **thoroughly tested**, the rest of the assignment is fairly straight-forward. The sample driver shows many example function calls with the appropriate output. You should be able to glean all of the information required from the driver.

Other criteria

1. You must allocate the nodes using **`new`.** The only function that should use **`new`** is `new_node`. This means that the keyword **`new`** will be used exactly once in your program. (If it is used more than once, even in a comment, you will lose points.)
2. Only the `push_front` and `push_back` methods should call `new_node` (to create nodes).
3. You must deallocate the nodes using **`delete`**. The only function that will use **`delete`** is `pop_front`. This means that the keyword **`delete`** should be used exactly once in your program. (If it is used more than once, even in a comment, you will lose points.)
4. `operator+` works like it has in the past. It will create a new *List* object and fill it with nodes from the list (`this`) and nodes from the right-hand-side.
5. `operator+=` will add the nodes from the right-hand-side to the list.
6. You will need two version of `operator[]`, one is for **`const`** objects, the other for non-**`const`** objects.
7. Make sure that you use **`const`** where appropriate. Double-check your code. (And then check it again.)
8. You are given the implementation for **`operator<<`** in the .cpp file.
9. Do not include any other header files in the .cpp file.
10. **THINK CODE REUSE.** In other words, can you call one of your functions to do something rather than write more of the same code? The exception to this is the overloaded subscript operators. You will have duplicate code in these two functions (const and non-const). Do not attempt to do any code re-use in these functions.

Deliverables
You must submit the header file (List.h) and the implementation file (List.cpp). These file must be zipped up and submitted to the appropriate submission area.

Source files	Description
List.cpp	The implementation file. All implementation for the functions goes here. You must document the file (file header comments) and all functions (function header comments) using the appropriate Doxygen tags.
List.h	The header file. You will need to modify the one I posted. **No implementation is permitted in this file.**

Usual stuff
Your code must compile (using the compilers specified) to receive credit. The code must be formatted as per the documentation on the website.

Make sure your name and other info is on all documents.