# CS 185
# Programming Assignment 9

## Details

This assignment gives you more practice with classes, constructors, pointers, and dynamic memory allocation/deallocation and of course linked lists. 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/removing items to and from either end, inserting new node, 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. This is the interface to the class:

```
class List
{
public:
        // Default Constructor
        // Copy Constructor
        // Non Default Constructor
        // Destructor

        // 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
        // pop_back, removes the last item in the list
        // remove_node_by_value, removes the first node it finds with the user defined value
        // insert_node_at, inserts a new node in the list. The node will be inserted at the user
           defined location and will have the user defined value.

        // list_size , returns the number of items in the list
        // empty, returns true if empty, else false
        // clear, clears the list from all nodes

        // display, used for printing the list's content

        static int created_list_count(); // returns the number of Lists that have been created
        static int alive_node_count();  // returns the number of Nodes that are still alive

        private:
                // used to build a linked list of integers
                struct Node
                {
                        Node(int);              // constructor
                        ~Node();                // destructor
                        Node *next;             // pointer to the next Node
                        int data;               // the actual data in the node
                        static int nodes_alive; // count of nodes still allocated
                };
```

```
            Node *head; // pointer to the head of the list
            Node *tail; // pointer to the last node
            int size;   // number of items in the list

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

The class contains 18 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**, **pop_front** and **pop_back**. 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 get 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** should 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**, **push_back** and **insert_node_at** methods should call **new_node** (to create nodes).
3. You must deallocate the nodes using **delete**. Make sure you are deallocating the memory anytime a node is removed. **Memory leaks will cost you a lot of points, make sure you check with Dr. Memory before submitting the assignment.**
4. The **pop_front** and **pop_back** methods return the value of the node that got popped. If no nodes were popped, the methods will return the value **-1**.
5. The **insert_node_at** method takes the location index as its first parameter. If a negative value is provided, the node should be inserted in the beginning of the list. If an index greater than the list size is provided, the node is added at the end of the list.
6. Make sure that you use **const** where appropriate. Double-check your code. (And then check it again.)
7. You are given the implementation for the **display** method in the .cpp file.
8. **THINK CODE REUSE.** In other words, can you call one of your functions to do something rather than write more of the same code?

## Comments

In this and future assignments, you are required to include:

- **A file header comment in every piece of source file. The format is shown in the "Comments.cpp" file given to you in the beginning of the semester and should be present at the very top of all your code.**
- **Function header for each function you create. The format is shown in the "Comments.cpp" file given to you in the beginning of the semester and should be present at the top of every function.**
- **Inline commenting for your code.**

## What to submit

You must submit the implementation file ***(List.cpp)*** in a single .zip file (go to the class page on moodle and you will find the assignment submit link). ***Do not submit any other files than the ones listed.***

**If you've forgotten how to submit files, the details are posted in the syllabus and in the assignment guidelines document. Failure to follow the instructions will result in a poor score on the assignment (and possibly a zero).**

## Usual stuff

Your code must compile cleanly (no errors and no warnings) to receive full credit. You should do a "test run" by extracting the files into a folder and verifying that you can compile and execute what you have submitted (because that's what I'm going to do.)

## Special note:

The due date/time posted is the positively latest you are allowed to submit your code. Since the assignments can easily be completed well before the deadline, you should strive to turn it in as early as possible. If you wait until the deadline, and you encounter unforeseen circumstances (like being sick, or your car breaking down, or something else), you may not have any way to submit the assignment on time. Moral: **Don't wait until the last day to do your homework.**