CS 170 | Spring 2017
Programming Assignment #5

submit due
- Monday, June 5, 2017
- 01:00am

Information

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.. This is the definition of the Node that will be used by the List class:

```
struct Node
{
      Node* pNext; // pointer to the next node in the list
      int data; // the data stored in the node
};
```

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:

```
#ifndef LIST_H
#define LIST_H
#include <iostream> // ostream

//! The namespace for the CS170 class
namespace CS170
{

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

        // Six methods:
        //   PushFront, adds the item to the front of the list
        //   PushBack, adds the item to the end of the list
        //   PopFront, removes the first item in the list
        //   size, returns the number of items in the list
        //   IsEmpty, returns true if IsEmpty, else false
        //   Clear, clears the list (removes all of the nodes)



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

        // Output operator for printing lists (<<)
      friend std::ostream & operator<<(std::ostream & os, const List& list);

        // Returns the number of Lists that have been created
      static int ObjectCount(void);

    private:
        // used to build a linked list of integers
      struct Node
      {
        Node *pNext;  // pointer to the next Node
```

```
        int   data;   // the actual data in the node
      };

      Node* m_pHead; // pointer to the head of the list
      Node* m_pTail; // pointer to the last node
      int   m_size;   // number of items on the list

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

} // namespace CS170

#endif
//////////////////////////////////////////////////////////////////////////////
```

The class contains 17 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 `PushBack`, `PushFront`, and `PopFront`. 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.

Sample command lines for compiling:

GNU:

```
g++ -o GNU.exe Driver.cpp List.cpp -Wall -Wextra -Werror -ansi -pedantic
-O
```

Microsoft

```
cl -W4 -EHa -WX -Za -FeMs.exe Driver.cpp List.cpp
```

Borland

```
bcc32 -v -vG -w -eBor.exe Driver.cpp List.cpp
```

Other criteria

1. You must allocate the nodes using `new`. The only function that should use `new` is `MakeNode`. This means that the keyword `new` should be used exactly once in your program. (If it is used more than once, you will lose points.)

2. Only the `PushFront` and `PushBack` methods should call `MakeNode` (to create nodes).

3. You must deallocate the nodes using `delete`. The only function that should use `delete` is PopBack. This means that the keyword `delete` should be used exactly once in your program. (If it is used more than once, 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?

Deliverables

You must submit the header file (List.h), the implementation file (List.cpp). These file must be zipped up and submitted to the appropriateMoodle 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.

Here are some popular Questions and Answers for the List assignment.

1. **Question: What should the `PopFront()` method return when the list is empty?**

Answer:

It should return -1.

2. **Question: What should we do if the index provided to the overloaded subscript operators is invalid?**

Answer:
Just keep "walking" the list until you get to the "index" that you are searching for. Yes, if the user provides an index that is too large you will get undefined results. Serves the user right. (When we talk about exceptions, we'll see a better way to handle this "exceptional" case.)

3. **Question: How does the clear method work? The sample driver isn't calling it.**

Answer:
The handout describes what the clear method should do. You should be able to figure out what the prototype of the function will be. (Note, it doesn't return anything.) Even though the provided *sample* driver doesn't call it, your code should. (Hint: The destructor should call it.)

4. **Question:**
   When I compile and run the program with all 3 compilers, each program gives me a different value for the total number of lists created. Am I doing something wrong?

Answer:
No. You will get different answers. Bonus to those students that know why. (This is an advanced topic covered in the advanced C++ course.)

5. **Question:**
   I'm getting a linker error regarding the `static` members. How do I fix that?

Answer:
The answer is in the notes. You can also find the answer in the required reading from the textbook.

6. **Question:**
   Why do we need two subscript operators (`operator[]`)?

Answer:
Because the driver won't compile with just one. You can find the answers in the required reading and in the notes.