Lab 12 – Overloading
CPSC 1021- Fall 2017

Introduction

In today's lab, we will work on operator overloading, init lists, friend functions, and demystifying the std::string class.

Lab Objectives

- Write a CPP class that manages its own memory
- Overload operators using friend and member functions
- Use **delete** and **new**

Prior to Lab

Brush up on operator overloading and some of the standard string/memory functions from the ancient times of plain old C:

http://en.cppreference.com/w/cpp/language/operators

https://www.tutorialspoint.com/c_standard_library/string_h.htm

(yes, I'm linking to tutorialspoint…)

Lab Instructions

Download the provided **lab12.cpp** and **MyString.hpp** from Canvas. This is a complete driver and a complete header file for the **MyString** class that you will implement.

Create a **MyString.cpp** file.

MyString

The MyString class is a stripped-down version of the std::string class you've come to know and love. We want to be able to append MyStrings together, both with other MyStrings and string literals. We also want all normal operations like assignment and construction to work as expected. All of this **without any memory leaks!**

This means you need to implement all the functions specified in the header file such that the driver executes correctly. Note that you **can comment out some of the driver code while you work**. This way you can focus on just the constructors to start, then complete assignment, then addition, etc.

I’m going to replace the driver when we grade this, so feel free to change the driver code around if it will help you debug.

Your code should:

- Use initializer lists (including initializing the char* data pointer!)
- Null terminate your MyString data with the ‘\0’ character
- Use your destructor to eliminate memory leaks

There’s not much else to say for this one, just make it work! This is about becoming familiar with the concept of operator overloading, and using more complex C++ code in general. You can do it!

Friend Functions

Friend functions are like static functions in that they are not members of a class. This means they **are not bound to the context of a class instance**. However, you do not have to invoke the class scope (MyString::) to call them. The benefit of using a friend function is that they have access to the member variables of the class they are friends with. How nice.

Every ostream operator is declared as a friend function, because if we did not then the order of operations would be the reverse of what we are used to. Consider this overloaded operator declared as a member function instead of as a friend:

```
ostream& MyString::operator<< (ostream& out) {
  return out << this->some_data;
}
```

To invoke this function, we must call it like this:

```
MyString s = “Hello World”;
s << cout;
```

You should be rubbing your eyes because that code looks bizarre. But it is 100% correct, since it compiles to the following:

```
MyString s = “Hello World”;
s.operator<<(cout);
```

By declaring the output operator (and others, such as addition) as friend functions, we can ensure that the order of invocation makes more sense. As a friend function (the way we have it defined in MyString.hpp) the code would be written and compile to the following:

```
MyString s = "Hello World";
cout << s;
...
operator<<(cout, s);
```

Notice how the operator<< function is free floating and not invoked on an instance of any class. This is the power of friendly and static methods.

Hints

1. You can use the **new** keyword in init lists. For example… **data(new char[])**
2. Member and friend functions can access member variables without calling getters
3. ***this** return the member object executing a function, and can be returned as a reference
4. Don't be afraid to use simple for loops
5. You'll probably want to use **strlen()**
6. Use **delete [] ptr** to delete an array/pointer you allocated using **new**

What to turn in

- **All CPP files, all header files (lab12.cpp, MyString.hpp, MyString.cpp)**

Compile and Execute

Use G++ to compile your code as follows **and include the C++11 standard!**

```
g++ -std=c++11 -Wall -g *.cpp -o strings
```

FORMATTING:

1. Your program should be well documented
2. Each file should include a header:
3. Your program should consist of proper and consistent indention
4. No lines of code should be more than 80 characters

```
// Sample Header
/******************
 your name
 username
 Lab X
 Lab Section:
 Name of TA
******************/
```

5 – 10 points will be deducted for each of the above formatting infractions.

Submission Instructions

- Test your program on the School of Computing server prior to submitting.
- Use the tar utility to tar.gz all source files. **Do not tar an entire directory! When I untar your archive, I should see all the files you included, not a top-level directory! Failure to correctly tar may result in up to a 25-point penalty!**
- Name your tarred file **<username>-lab<#>.tar.gz** (ex. nglyder-lab12.tar.gz)
- Use handin (http://handin.cs.clemson.edu) to submit your archive