String Objects and Introduction to Test Driven Development

CSE 2341 - Data Structures

Introduction

One fundamental type that you'll work with as a programmer is the **string type**. In Programming Assignment 01, you will implement a String class in C++, and you will use that class in your implementation of the first project. Further, you'll be limited to using your custom string class for the remainder of the semester (no STL string objects).

There are many schools of thought on the best way to write code and test it. One of these methodologies is called **Test Driven Development (TDD)**. In TDD, the basic idea is you write the testing code for whatever things you're working on first, and then write the thing to pass the tests. For example, if you're implementing a String class, you might initially decide that String objects need to be able to be copied and printed to the screen. So, before writing any part of the string class implementation, you write some code to test copy functionality and printing functionality. Clearly, if string class hasn't been written yet, these test will fail - that's exactly what you want to happen. Initially, all your tests will fail. But as you begin writing the code to implement string, the tests will begin to pass.

Quite a few frameworks exists for TDD (and unit testing, for which TDD is a particular type). For Data Structures, we will use the CATCH Framework. While CATCH supports multiple paradigms of testing, we will use it in the TDD mindset for 2341. Please read the CATCH Framework Tutorial.

In this programming assignment, you are provided with a file called tests.cpp which shows an example of how to use the CATCH framework for testing your classes. It would be a good idea to use, at the very least, the provided CATCH example file to test your own String class. In lab this week, more about CATCH will be covered including how to incorporate it into your projects.

The String Class

The String class shall represent a sequence of characters of unbounded length (within the limits of the system). Minimally, the string class should expose an interface of functionality that is consistent with typical string libraries. An example minimal string interface can be found in string.h associated with this handout.

The manner in which you implement the underlying functionality is up to your discretion. One option would be to store null-terminated c-strings and effectively create a wrapper for the c-string functions. Another option would be to simply store an array of characters and data

members holding size as well as capacity.

Dynamic Memory Management

Any class which manages dynamic memory must manage that memory following best practices (applying Rule-of-3, etc). Not following those best practices could lead to segmentation faults, memory leaks, and shallow copy-related problems.

Testing Your Classes

It is expected that you will follow the TDD mindset when developing the string class. The TAs have been directed to give guidance under that paradigm. Therefore, you'll need to create your test case source files along with your project files. This will be covered in more detail in Lab this week. You are only doing yourself a disservice by not fully investing your time and energy in TDD. Every bug you tease out now is one less you'll need to worry about later.

Running Your Code

Using the CATCH testing interface, providing a TEST_CASE in tests.cpp is equivalent to providing a main function. Inside your Sprint1 folder, you can use g++ *.cpp -std=c++11 to compile your code and then ./a.out to execute it and run your test cases. Again, please read CATCH Framework Tutorial for more details.