# CS 311 Fall 2017
# Assignment 1

Assignment 1 is due at **5 pm Thursday, September 14**. It is worth 25 points.

## Procedures

This assignment is to be done individually.

Turn in answers to the exercises below on the UA Blackboard Learn site, under Assignment 1 for this class.

- Your answers should consist of the source code for Exercise A (files `product.h`, `product.cpp`). These should be **attached** to your homework submission.
- Send only the above! I do not want project files or executables.
- I may not look at your homework submission immediately. If you have questions, e-mail me.

## Exercises (25 pts total)

### Exercise A — "Product" Class

#### Purpose

In this exercise, you will write a simple class of the kind that might have been done as an exercise in operator overloading in CS 202. However, a number of the concepts covered in CS 311 will need to be applied, including documentation comments based on the idea of an operation contract. Further, your code will need to work with a thorough test program. Most importantly, quality standards will be quite high.

#### Instructions

This assignment is to be done individually.

Implement a C++ class that holds information about the sales of a product made by some company: the name of the product, and the number of items sold. Be sure to follow the coding standards. *You do not need to follow the standards in part 3 ("Additional Standards"); these come later.*

- Name your class "`Product`", and implement it in files `product.h` and `product.cpp`.
- An object of type `Product` should act as if it maintains two data items:
  - A string holding a product's name.
  - A nonnegative integer, giving the sales of the product (the number of items that have been sold).
- Include the following functions, and no others, in the public interface of your package:
  - Default ctor, copy ctor, move ctor, copy assignment, move assignment, dctor.
    - The default ctor sets the product name to an empty string (`""`) and the number of items sold to zero.
  - 2-parameter ctor: parameters are a string (name) and an integer (sales).
  - "Get" member functions for the data stored. Each of these takes no parameters and return either a string or a number, as appropriate. Call these `getName` and `getSales`.
  - "Set" member functions for the data stored. Each of these takes a single parameter and return nothing. Call these `setName`, `setSales`.
  - Member function `toString`. This takes no parameters and return a string representation of the stored information: the product name, followed by a blank and a left parenthesis ("`(`"), the word "`sales`", a colon ("`:`"), a blank, the number of items sold as a base-ten integer, and a right parenthesis ("`)`").
    - For example, if the product name is "`Super Widget`", and 42 items have been sold, then `toString` returns "`Super Widget (sales: 42)`".
  - Equality ("==") and inequality ("!=") operators for comparing two `Product` objects. They are equal if *both* the product names and the sales are equal.
  - Pre- and postincrement operators ("++") and pre- and postdecrement operators ("--"). These increment or decrement the number of items sold, as appropriate—except that, if the number of items is zero, then the

    decrement operators do *not* reduce the number to minus-one, but keep it at zero. Each operator returns the appropriate `Product` object.
  - A stream insertion ("<<") operator. This outputs to the given stream the same string as that generated by `toString`. It does *not* add a newline at the end.
- Externally, all strings are handled using the C++ standard library class `std::string`, and all integers are handled using type `int`.

Other Requirements

- Remember that the number of items sold cannot be negative. However it is passed in as an `int`, which can be negative. What if the client code gives you a negative value? You must deal with this somehow. The way you do it is up to you, but you must **document** how you did it.
- Avoid unnecessary duplication of code (for example, in pre `operator++` and post `operator++`).

## Test Program

A test program is available. This is a single program comprising two source files: `product_test_main.cpp` and `product_test_suites.cpp` If you compile and run the test program (unmodified!) with your code, then it will test whether your code works properly.

The test program requires `catch.hpp`, the single-header version of the "Catch" unit-testing framework. This file may be downloaded from the Catch GitHub site.

Do not turn in the test program or the Catch framework.

Note: The test program does **not** check the following completely, but you still need to do them correctly.

- Handling of out-of-range values.
- Whether extra public functions, other than those in the instructions, are defined.
- Whether any function performs any actions it does not need to do (like debugging printout).
- Whether the most appropriate parameter-passing and return methods are used.

## Thoughts

- This is to be **high-quality** code. That takes time, even for experienced programmers.
- When grading, I plan to go through and check whether you have met each of the requirements above, as well as each of the coding standards. I suggest that, before you turn in your work, you do the same.