

# Computer Science II – Lab 3

## Testing and Debugging

### Introduction

Testing and debugging are important steps in programming. Loosely, you can think of testing as verifying that your program works and debugging as finding and fixing errors once you've discovered it does not. Writing test code is an important (and sometimes tedious) step. Many software libraries have “regression tests” that run automatically to verify that code is behaving the way it should.

Here are four strategies for testing and debugging:

1. When you write a class, write a separate “driver” main function that calls each member function, providing input that produces a known, correct result. Output of the actual result or, better yet, automatic comparison between actual and correct result allows for verifying the correctness of a class and its member functions.
2. Carefully reading the code. In doing so, you must strive to read what the code actually says and does rather than what you think and hope it will do. Although developing this skill isn't necessarily easy, it is important.
3. Judicious use of `cout` statements to see what the program is actually doing. This is especially useful for printing the contents of a large data structure or class. It is often hard to visualize large objects using the debugger (see next item) alone.
4. Using the debugger to (a) step through your program, (b) check the contents of various variables, and (c) locate floating point exceptions and segmentation violations that cause your program to crash.

### Points and Rectangles

The programming context for this lab is the problem of determining what 2D points are in what 2D rectangles. For rectangles, we will assume they are aligned with the coordinate axes, as shown in Figure 1. This makes it easy to represent and to test if a point is inside. Our code will store points in rectangles and determine which points are in which rectangles. This is a toy example of problems that must be addressed in graphics and robotics.

Please download the following 3 files needed for this lab and then turn off your internet connection:

<http://www.cs.rpi.edu/academics/courses/spring08/cs2/labs/lab03/Point2D.h>

<http://www.cs.rpi.edu/academics/courses/spring08/cs2/labs/lab03/Rectangle.h>

<http://www.cs.rpi.edu/academics/courses/spring08/cs2/labs/lab03/Rectangle.cpp>

### Checkpoint 1

Start by creating a project and adding the files `Point2D.h`, `Rectangle.h`, and `Rectangle.cpp`. Examine these files briefly. `Point2D.h` has a simple, self-contained class for representing point

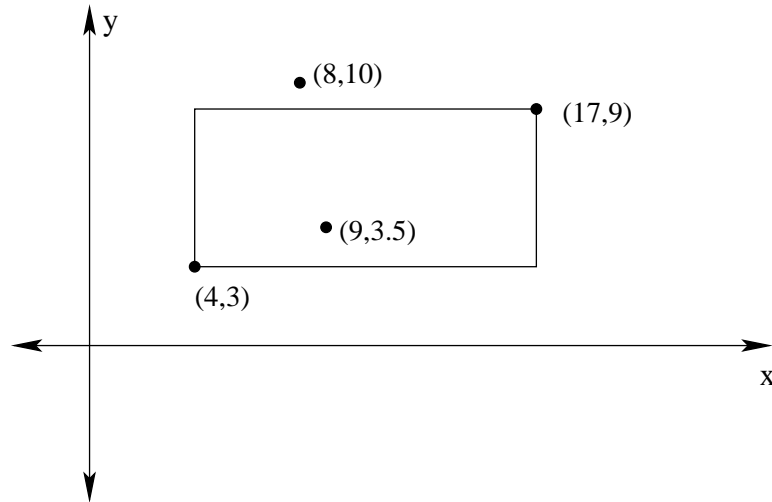


Figure 1: Example of a rectangle aligned with the coordinate axes — the only type of rectangle considered here. The rectangle is specified by its upper right corner point,  $(17, 9)$ , and its lower left corner point,  $(4, 3)$ . The point  $(9, 3.5)$  is inside the rectangle, whereas the point  $(8, 10)$  is outside.

coordinates in two-dimensions. No associated `.cpp` file is needed because all member functions are defined in the class declaration. `Rectangle.h` and `Rectangle.cpp` contain the start to the Rectangle class. They also contain a bug. Please read the code now to see if you can find it. Do not worry if you can not, but do not fix it in the code if you do!

**Your job in this checkpoint** is to complete the implementation of the `Rectangle.cpp` class. Look through `Rectangle.h` and `Rectangle.cpp` to determine what functions need to be added. Then, compile these files and remove any compilation errors.

## Checkpoint 2

Create a new file (within Visual Studio put this within the current project) and call it `test_rectangle.cpp`. Create a `main` function within this file. In the main function, write code to test *each of the member functions*. For example, write code to create several rectangles, and print their contents right after they are created. Write code that should produce both true and false in the function `is_point_within`. (In fact, if there is non-trivial logic in a function, the test code should call the function several (or even many) times with varying inputs to test all the possible conditions.) Write code to add points (or not) to a rectangle. Write code to find what points are contained in both rectangles.

**To complete this checkpoint**, show a TA your test cases and the error(s) that those test cases reveal in the provided code. After doing this you should be able to spot that there is an error in the code that I provided you (as well as, perhaps, errors in your own code). Even if you know where the bug or bugs occur, please do not fix them yet.