LW: Debugging Craps Game

Objectives

- Gain experience working through the debugging process!
- Review variables line by line
- Discuss the debugging process with your group as you debug together.
    - What are different ways to find this information?
    - What are the ways to fix it?
        - Would restructuring help?

Completion

- Attend the lab or have an excused absence.
- 25 of 30 points on the autograder.

Submission

- craps.cpp
    - Make sure you remove or comment out all debugging statements before submitting.

Overview

This is a chance to practice debugging a program. Sometimes when you get stuck, using the debugger to move line-by-line can shed light on the source of the problem.

The program that you will be debugging plays a game of craps, but it has a couple of bugs. You will debug the program such that it works correctly... there are two bugs that you're to find.

Craps Overview

In craps, a "Pass Line" bet proceeds as follows. Using two six-sided dice, the first roll of the dice in a craps round is called the "Come Out Roll." Pass Line bets immediately win when the shooter's come out roll is 7 or 11, and lose when the come out roll is 2 (snake eyes), 3 (cross eyes), or 12 (box cars). If 4, 5, 6, 8, 9, or 10 is rolled on the come out roll, that number becomes "the point." The shooter keeps rolling the dice until he or she rolls the point or 7 to end the round. If the shooter rolls the point first, the pass line bet wins. If, however, the shooter rolls a 7 first (i.e. "sevens out"), a pass line bet loses.

Debugging Overview

You can use whatever environment you want for debugging. However, if you want to use the VS Code Debugger, there are some optional instructions specifically for that.

Many IDEs provide the same options as the VS Code Debugger so you can also explore debugging with your preferred IDE. So feel free to search for how to debug in your IDE.

Regardless, you can just use the command line for compiling and add print statements to standard output (i.e. cout) to see what happens as the program executes.

What to do

1. Get the Start Code
2. Compile the program.
    a. There may be syntax errors, these can be fixed easily.
    b. There will be warnings. Avoid fixing for now.
3. Run the program.
    a. Try to get an idea of the path of execution.
        i. You can use the VS Code debugger to execute the program line-by-line.
        ii. You can add output statements to see variable values.
            1. We highly recommend outputting both the variable name and the variable value since it can be hard to decipher what value printed corresponds to which variable as you increase the number of values output.
    b. As you do this try to trigger all lines of code. E.g. both when an if is true and when it is not true.
    c. At some point end the game, or forcefully exit if it gets stuck.
        i. In VS Code you can click on the red square.
        ii. On the command line you can do ctrl+c.
4. Fix all errors, warnings and logical errors.

VS Code Debugger on next page

VS Code Debugger

1. Make sure the starter code is in its own folder.
2. Open that folder and not just the file.
3. Compile, run, and play the game.
4. Set an inline breakpoint on the line 'int wallet = 1000;' You will see a red dot to the left of the line number after it is set.
    a. Click to the left of the line number.As you hover, you will see a faded red dot. If you select a line that can't support a breakpoint, it will add the breakpoint to the next line of code that can have a breakpoint.
    b. Right click on the line and select "Add inline breakpoint"
5. Run your code with the debugger (Run-> Start Debugging); it should stop at the breakpoint.
6. Step through your code using "step over", "step into", and "step out".
7. Input and output is through the terminal below the code.
    a. If you are having problems with cin causing the debugger to not work, replace 'cin >> bet;' with 'bet = 12;' or some other value for the bet.
8. While stepping through, look at variable values to the left. You can also see the call stack.
    a. There are a lot more things that you can learn about the debugger, but this is sufficient for most of what you'll do in this class.
9. Find and fix bugs