**Assignment Description:**

Sometimes you will be given a program that someone else has written, and you will be asked to fix, update and enhance that program. In this assignment you will start with an existing implementation of the classify triangle program that will be given to you. You will also be given a starter test program that tests the classify triangle program, but those tests are not complete.

In order to determine if the program is correctly implemented, you will need to update the set of test cases in the test program.  You will need to update the test program until you feel that your tests adequately test all of the conditions.   Then you should run the complete set of tests against the original triangle program to see how correct the triangle program is.    Capture and then report on those results in a formal test report described below.   For this first part you should not make any changes to the classify triangle program.  You should only change the test program.

Based on the results of your initial tests, you will then update the classify triangle program to fix all defects.  Continue to run the test cases as you fix defects until all of the defects have been fixed.   Run one final execution of the test program and capture and then report on those results in a formal test report described below.

Note that you should NOT simply replace the logic with your logic from Assignment 1.  Test teams typically don't have the luxury of rewriting code from scratch and instead must fix what's delivered to the test team.

**Author:**

Samantha Inneo

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Input | Expected Results | Actual Results | Pass or Fail |
| testRightTriangleA | 3, 4, 5 | Right | InvalidInput | Fail |
| testRightTriangleB | 6, 8, 10 | Right | InvalidInput | Fail |
| testRightTriangleC | 5, 3, 4 | Right | InvalidInput | Fail |
| testRightTriangleD | 10, 8, 6 | Right | InvalidInput | Fail |
| testEquilateralTriangleA | 1, 1, 1 | Equilateral | InvalidInput | Fail |
| testEquilalateralTriangleB | 200, 200, 200 | Equilateral | InvalidInput | Fail |
| testNotATriangleA | 10, 10, 20 | NotATrianlge | InvalidInput | Fail |
| testNotATriangleB | 10, 10, 100 | NotATriangle | InvalidInput | Fail |
| testNotATriangleC | 10, 15, 30 | NotATriangle | InvalidInput | Fail |
| testIsoscelesTriangleA | 10, 10, 18 | Isosceles | InvalidInput | Fail |
| testIsoscelesTriangleB | 18, 10, 10 | Isosceles | InvalidInput | Fail |
| testScaleneTriangleA | 2, 5, 6 | Scalene | InvalidInput | Fail |
| testScaleneTriangleB | 6, 5, 2 | Scalene | InvalidInput | Fail |
| testInvalidInputA | -1, -1, -1 | InvalidInput | InvalidInput | Pass |
| testInvalidTriangleB | 201,201,201 | InvalidInput | InvalidInput | Pass |
| testInvalidTriangleB | 6.5, 5.5, 2.5 | InvalidInput | InvalidInput | Pass |
|  |  |  |  |  |
|  |  |  |  |  |

**Summary:**

After running the test file for the first time, it was clear there was at least one major issue with the provided code. Every single input resulted in ‘InvalidInput’ as the output, which caused nearly every test in the file to fail. I knew it was likely that all these failures would be caused by one issue, so I looked for the areas of code that return ‘InvalidInput’. I found that on line 34, the if statement checking that the number was at least 0 was checking if b >= b, which was always true, so every single input was being categorized as invalid. I changed it to instead check that b>=0, and instantly fixed that issue. The next major issue I set out to fix was that many triangles were incorrectly being labeled ‘NotATriangle’ when they were in fact valid triangles. (Ex: 1,1,1) To fix this, I went to line 46, where the logic on what defines a triangle was incorrect. It was checking that the sum of any 2 sides is shorter than the third, when it should have been checking for it to be greater. Changing this made 5 more tests pass. Looking at the remaining tests, I realized something might be wrong with the equilateral code, since it was incorrectly defining some isosceles to be equilateral. I found that the code only checked that the first two sides were equal, not all three. The remaining failures were all on right triangles, so that is the next place I began checking. The first reason this was wrong was that they were multiplying by 2 rather than squaring the numbers. The last fix I had to make was that they never checked which side would be a, b, and c. For Pythagorean theorem to work, c must by the largest of the 3 sides. I added ors to the clause to check if a or b are the largest. At this point, all of the tests passed.

**Honor Pledge:**

I pledge my honor that I have abided by the Stevens Honor System.

**Detailed Results:**

Write