# ISIT 324 Final Exam

## Winter 2020

**50 Points Possible**

This exam will test your knowledge spanning the entire course. Make sure to look at all the questions before you begin so you can prioritize appropriately.

Submission

This exam is formulated as a quiz with two questions:

* For Question 1, submit a .docx file with the answers to questions 1 – 11 in this document.
* For Question 2, submit a .cs file with the unit test that addresses question 12.

Good luck!

**Section 1 – General Questions**

1. **(3 points)** What is the difference between a defect, an error and a failure?

An error is a human made mistake such as a typo or miscalculation that leads to a problem or defect in the code.

A defect is a fault or bug in the code that causes the product to not do what it was designed to. AKA, the defect causes the product to not meet its requirements or specifications.

A failure is the result of a defect. Failures happen when the product fails to do it’s job correctly or at all.

1. **(3 points)** Explain the four conditions needed for a failure to be observed during a test using the RIPR model.

Reachability: Location(s) of where the error is in the program being tested.

Infection: The program being tested must be in an incorrect state.

Propagation: The program must be able to produce an undesirable result.

Reveal: The undesirable result must be witnessed by a tester.

1. **(3 points)** What is the purpose of acceptance testing purpose and according to the V-model, with which software development activity is it associated?

Acceptance testing a methodology where development is done in sequential stages. Stages can only begin once all the requirements for the previous stage have been met. When performed with the V-model process this is considered an extension of the waterfall model.

1. **(3 points)** Briefly explain the pesticide paradox.

The pesticide paradox states that if you run identical tests repeatedly, they will eventually underperform. To avoid this, test cases need to be regularly reviewed and updated as the product changes.

1. **(3 points)** Name three black-box (aka specification-based) test techniques..

Equivalence partitioning

Boundary value analysis

Decision table testing

State transition testing

Use case testing

1. **(3 points)** Name and describe two types of Fakes.

Shims and Stubs are two fakes used to isolate and test components.

Shims are used to test specific methods by diverting calls to themselves to complete test cases. Shims are used to test a piece of code in a different assembly than your test project.

Stubs allow you to test specific components that call other components. The test stub takes the place of component under test in test cases.

1. **(4 points)** Consider the following code fragment**.**

1 READ A

2 READ B

3 C = A + 2 \* B

4 IF C > 50 THEN

1. PRINT ‘Large C’

6 ENDIF

If you run this test case by itself:   
 A = 20, B = 25

a. What is the statement coverage percentage? 100%

b. What is the decision coverage percentage? 50%

If you run both of these test cases:   
 A = 20, B = 25  
 A = 20, B = 10

c. What is the statement coverage percentage? 100%

d. What is the decision coverage percentage? 100%

1. **(4 points)** A salesperson selling aglets for Consolidated Shoelaces earns commissions based on their monthly sales as follows:
   1. If they sell between $1 and $1000 of aglets, they earn 10% commission.
   2. If they sell between $1001 and $5000 of aglets, they earn 20% commission.
   3. If they sell more than that, they earn a 30% commission.

List the values of the normal boundary value partitions for range 2, above.

**$1001, $1002, $3000, $4999, $5000**

**Section 2 - Right Triangle Questions**

The following questions should be reminiscent of Homework 8, your latest triangle-oriented homework. Once again with the triangles, but while this one still deals with sides, the goal is to detect whether the sides form a right triangle, an acute triangle, or an obtuse triangle. And, of course, you’ll need to determine whether the three sides can form a triangle at all and whether the values input are valid.

You’ll be designing and coding tests for a hypothetical triangle analyzer that works as follows:

* The input consists of the lengths of the three sides: a, b and c.
* To simplify the problem, side c must be the longest side. That is, neither side a nor side b can be longer than side c, although either may be the same length as side c.
* The lengths of the sides must be expressed as positive integers less than or equal to 100.
* The output will be as follows:
  + If the triangle is a right triangle, “Right triangle”
  + If the triangle is acute, “Acute triangle”
  + If the triangle is obtuse, “Obtuse triangle”
  + If side a or side b are longer than side c, “Side c not longest”
  + If side c is not shorter than the sum of the lengths of sides a and b, “Not a triangle”
  + If any of the sides is not a positive integer or is greater than 100, throw an application exception.

In case you’ve forgotten your Pythagoras, here’s the way to tell whether a triangle is acute, obtuse or right:

* if c2 = a2 + b2, the triangle is a **right** triangle (as in the famous 3, 4, 5 right triangle).
* if c2 < a2 + b2, the triangle is an **acute** triangle.
* if c2 > a2 + b2, the triangle is an **obtuse** triangle.

If you need more info you can find it here: <https://www.murrieta.k12.ca.us/cms/lib5/CA01000508/Centricity/Domain/1830/T4.5.pdf>

**Continued on next page**

1. **(4 points)** List the normal equivalence classes for this new triangle problem in table 2.1 based on its range (output). Provide one test case for each equivalence class in table 2.2.

*Table 2.1*

|  |
| --- |
| **Equivalence Class** |
| 1: Right triangle |
| 2: Acute triangle |
| 3: Obtuse triangle |
| 4: Side c not longest |
| 5: Not a triangle |

*Table 2.2*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Side a** | **Side b** | **Side c** | **Expected Output** |
| TC1 | 3 | 4 | 5 | Right triangle |
| TC2 | 49 | 36 | 60 | Acute triangle |
| TC3 | 12 | 16 | 22 | Obtuse triangle |
| TC4 | 12 | 22 | 16 | Side c not longest |
| TC5 | 4 | 4 | 16 | Not a triangle |

1. **(8 points)** Complete the decision table below. It includes a full set of valid Actions for the triangle problem and one example condition. Your task:
   * Complete the set of conditions. Do **not** include conditions for invalid values (i.e., lengths < 1 or > 100).
   * Add the cases that give rise to the actions. (A case is a combination of conditions specified in a column.) **Case 1** is specified for you.

In the **Conditions** section, use T for true, F for false and a hyphen (-) for “don’t care” conditions. Each cell must have a value.

In the **Actions** section, use X for the cells in which the corresponding action (in the row) will take place based on the precipitating conditions (in the column).

Each column has a **Case Number** in its heading. You’ll use that to ensure that each of the decision table cases are accounted for in your unit tests.

Note: For my solution, I was able to squeeze everything into a table with these rows and columns, but mine is not the only solution. If you feel the need to add or delete rows, columns, or both, feel free.

**Continued on next page**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Conditions** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| c <= a + b? |  |  |  | - | F | - | - | - | - | - |
| c2 = a2 + b2? | T |  |  | - | - | - | - | T | - | T |
| c2 < a2 + b2? |  | T |  | - | - | - | - | T | T | T |
| c2 > a2 + b2? |  |  | T | - | - | - | - | - | T | T |
| a > c? |  |  |  | T | - | - | T | - | - | - |
| b > c? |  |  |  | - | *-* | T | *T* | - | - | - |
| **Actions** |  |  |  |  |  |  |  |  |  |  |
| Not a triangle |  |  |  |  | X |  |  |  |  |  |
| Side c not longest |  |  |  | X |  | X | X |  |  |  |
| Right triangle | X |  |  |  |  |  |  |  |  |  |
| Acute triangle |  | X |  |  |  |  |  |  |  |  |
| Obtuse triangle |  |  | X |  |  |  |  |  |  |  |
| Impossible |  |  |  |  |  |  |  | X | X | X |

1. (**7 points)** Write a set of test cases that will allow you to test each of the cases you identified in your decision table in the table below. Use the decision table’s column label as the Case ID for each row. Each cell in the columns *a, b,* and *c* should be the length of a side. Each cell in column *Expected Output* should correspond to an action from the decision table.

For impossible rules, put an “x” in the values of a, b and c.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Case ID | a | b | c | Expected Ouput |
| 1 | 3 | 4 | 5 | Right triangle |
| 2 | 49 | 36 | 60 | Acute triangle |
| 3 | 12 | 16 | 22 | Obtuse triangle |
| 4 | 22 | 12 | 16 | Side c not longest |
| 5 | 4 | 4 | 16 | Not a triangle |
| 6 | 12 | 22 | 16 | Side c not longest |
| 7 | 22 | 22 | 16 | Side c not longest |
| 8 | x | x | x | Impossible |
| 9 | x | x | x | Impossible |
| 10 | x | x | x | Impossible |

**Continued on next page**

1. (**5 points)** Write a unit test program to run against the “FinalExamRightTriangle.cs” program linked to from the exam.
   * To run your tests against the provided SUT, you'll need to create a new solution with a **Class - .NET Framework** project and replace the code therein with the code you just downloaded (FinalExamRightTriangle.cs). Then create your unit test as usual.
   * If you're unable to get your unit tests to appear or to be recognized by the test runner, send me your code as is and enter a submission comment to let me know and I'll take that into consideration when I grade.

For full credit:

* + Ensure that the SUT properly handles invalid values for each side.
  + Ensure that all test cases in your decision table are accounted for.
  + *You can earn extra credit by using fluent assertions.*

Requirements

* Use MSTest V2 parameterized tests (i.e., use the DataTestMethod and DataRow attributes.)
* For tests based on the decision table, name each row with a display name that holds the case number from the decision table. Here’s an example from the NextDate exercise:

[DataRow(4, 16, 2001, 4, 15, 2001, DisplayName = "Cases 1 - 3")]

* Standard stuff:
  + Make sure to submit the correct file. If you submit the SUT or a .csproj or anything else other than the unit test .cs file, you will receive zero for this question.
  + When I attempt to run the programs you submit, the tests should run.
  + Use our standard name formulation for the test class and test method. That is:
    - Test Class: *MethodName*\_Should
    - Test Method: Return*ExpectedValue*\_When*SomethingHappens*
  + Do not use the ExpectedException attribute.
  + Arrange your test methods using the “arrange, act, assert” pattern (including comments) and make sure that your code appears in the right place.
  + Use the SUT alias to identify the software under test.

**End of Exam**