Name: _____ Net-ID: _____

Testing is when we determine whether a program executes correctly. I.E. Does it perform as described in the program specification? Note that debugging is trying to find the causes of failed tests and how to correct the issues. Debugging is different from testing! In this lab, we will focus on testing.

What are Test Cases and What Kinds of Testing Cases Exist?

Test cases are composed of two things:

1. A description of the actions to be given (often a set of input or parameter values) and
2. The expected results for that given input

When running a test case, the result is whether the test passed or failed. This is done by executing the set of input and seeing if the expected results were generated. If so, the test case passes. Otherwise, it fails.

What input values should we use for the test cases? Consider the following Specification:

Program Specification for a Simple Tax Calculation:

Create a web form that will allow the user to calculate the amount of taxes owed to the government. The user should enter the income amount in an input field, then press/click a button and the amount of taxes owed is to be displayed. If the user enters a non-numeric income amount or a negative income amount, display an error message telling the user that a positive numeric value must be entered. The amount of taxes owed is determined by:

- *If the income amount is $5,000 or less, the tax amount is 10% of the income amount.*
- *If the income amount is more than $5,000 and is $50,000 or less, the tax amount is 15% of the income amount.*
- *If the income amount is more than $50,000, the tax amount is 20% of the income amount.*

Equivalence Class Testing: Test cases should include inputs from all possible ranges of inputs (including error inputs). Each of these possible ranges of input is often called an **Equivalence Class Testing**. For the above specification, there are five Equivalence Classes:

- EC1 - the values starting from $0 up to and including $5,000.
- EC2 - the values greater than $5,000 up to and including $50,000.
- EC3 - the values greater than $50,000.
- EC4 - the values less than $0 (negative input).
- EC5 - the non-numeric values.

Each Equivalence Class should produce (at least) one test case. A good input value for an Equivalence Class is near the median (middle) of the range of possible inputs. For Equivalence Classes that have an infinite range (i.e. …values more than $50,000…), select an input value that is toward the middle of the typically "expected input range" (for this example one might assume a maximum input near $500,000).

1. Which of the following is the best Equivalence Class Test Case for EC2?

   $2,500 $5,000 $10,000 $27,500 $50,000 $275,000

2. Which Equivalence Class(es) have a finite range of inputs? (Indicate/circle all that apply.)

   EC1 EC2 EC3 EC4 EC5

Name: _____ Net-ID: _____

3. Which Equivalence Class is being testing when the input of "Apple" is given?

E1 EC2 EC3 EC4 EC5

Boundary Case Testing: Test cases should also include inputs that verify the proper functioning of the program for values at the boundary between two Equivalence Classes. This are often called **Boundary Case Testing**. Each boundary should generate multiple test cases: a test for the boundary value, and a test for each value on either side of the boundary value. For the above specification,

- one Boundary Value is the value of $0. The boundary between EC1 and EC4
- another Boundary Value is the value of $5,000. The boundary between EC1 and EC2
- a third Boundary Value it the value of $50,000. The boundary between EC2 and EC3

4. Which of the following values is NOT an input for a Boundary Case Test for this example?

-1 0 4999 22499 50000 50001

5. Which of the following values ARE input for Boundary Test Cases between EC2 and EC3? (Indicate all that apply.)

-1 0 4999 22499 50000 50001

6. Are they any additions values for Boundary Test Cases between EC2 and EC3 that are not listed by Q. 5? If so, list those additional values below.

Automated Unit Testing

Testing is often done using a technique called **Unit Testing**, where a method/function/unit in the program being tested is called with the desired input and a result is produced. This actual result is compared with the expected result. If the two are the same, the test passes.

Often this can result in a huge number of test cases. Fortunately, there exist many **automated** ways to run test cases. In C/C++, one such way of doing this is using the **TinyTest Tool Kit** which can easily be used with any program. (JUnit is a similar tool for Java programming.)

Each **Automated Unit Test** for TinyTest is written as a function were the input, expected result and the actual result are specified in the code. This test function might create variables or an instance of the class that contains the starting method of the code being testing. The test may call other methods/functions to initialize the variables used for the test. The Unit Test then calls the ASSERT_EQUALS( ) macro with the expect result value and the function call being tested as parameters to determine if the expected result matches the actual result.

Name: _____ Net-ID: _____

If all tests pass, you get a message such as:

```
PASSED [Point2dTest.cpp] (total:4)
```

or as follows if a test does not pass:

```
failure: Point2dTest.cpp:60: In test testDistanceFrom():
    p1.distanceFrom(p2) ((75) == (p1.distanceFrom(p2)))
FAILED [Point2dTest.cpp] (passed:3, failed:1, total:4)
```

In the code files of Point2dTest.cpp is a program that tests the various methods from the Point2d class. The method distanceFrom( ) returns the distance between 2 points. An example to test this method using TinyTest is as follows (and see the code Point2dTest.cpp). The following test function contains 3 different unit tests.

```
void testDistanceFrom() {                        // the test function
  Point2d p1 ( 0, 0 );
  Point2d p2 ( 8, 0 );
  ASSERT_EQUALS ( 8.0 , p1.distanceFrom(p2) );   // unit test #1
  ASSERT_EQUALS ( 8.0 , p2.distanceFrom(p1) );   // unit test #2
  p2.setXY ( 3, 4 );
  ASSERT_EQUALS ( 5.0 , p1.distanceFrom(p2) );   // unit test #3
}
```

For this lab, compile and run the **Point2dTest** executable for the following unit tests in Point2dTest.zip. This is done by using the makefile using the target of Point2dTest, which is done with the command:

make Point2dTest

The code portion of the testGetQuadrant( ) function to test/verify the point (3, 4) is in Quadrant 1 is:

```
p1.setXY ( 3, 4 );
ASSERT_EQUALS ( 1 , p1.getQuadrant() );
```

7. Does the above test (verifying the point (3, 4) is in quadrant 1) pass or fail?

The code portion of the testGetQuadrant( ) function to test/verify the Origin, the point at (0, 0) is in Quadrant 0 is:

```
p1.setXY ( 0, 0 );
ASSERT_EQUALS ( 0 , p1.getQuadrant() );
```

(The code specification has any point that lies on either the X axis or the Y Axis exists in Quadrant 0.)

8. Does the above test (verifying the Origin is in quadrant 0) pass or fail?

Name: _____ Net-ID: _____

9. What is the portion of code that will test/verify the point at (-3, 5), is in quadrant 2?

10. Does the above test (verifying the point (-3, 5) is in quadrant 2) pass or fail?

11. What is the portion of code that will test/verify the point at (8, -2), is in quadrant 4?

12. Does the above test (verifying the point (8, -2) is in quadrant 4) pass or fail?

13. What is the portion of code that will test/verify the point at (-7, 0), is in quadrant 0?

14. Does the above test (verifying the point (-7, 0) is in quadrant 0) pass or fail?