HW3: Testcases

Your assignment is to design a test-suite for **Project 2**. A test-suite is a set of tests to verify that your program works correctly. You will turn in a document specifying the tests that you would need to run in order to verify that ALL subparts of your program function correctly and that the program as a whole functions correctly. You will specify the Purpose of the test using the following table

|  |  |
| --- | --- |
| **Purpose of the Test** |  |
| Pre-Conditions |  |
| Post-Conditions |  |

Then you will specify the specific test steps / sequences used to verify the program results. Note – the Pass/Fail and the Comments columns would be left blank when turning in HW 3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Step** | **Action** | **Expected System Response** | **Pass/ Fail** | **Comments** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Your test suite will be graded based on its completeness. Don’t forget there are 3 basic kinds of testcases:

* Common Cases: normal input / data
* Boundary Cases: extreme but valid input / data
* Error Cases: bad input / data

# Example Test Scenario

|  |  |
| --- | --- |
| **Purpose of the Test** | Test whether the code handles valid guesses correctly as well as invalid guesses properly |
| Pre-Conditions | None |
| Post-Conditions | Program complete successfully and returns to Windows |

For example, if you were writing test cases for a guessing game application, it might look like the following. In this game, the computer would generate a random number between say 1-100. Then the program would ask the user to guess that number. If the guess was higher than the number, then it would print a message and ask to guess again.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Step** | **Action** | **Expected System Response** | **Pass/ Fail** | **Comments** |
| 1 | Enter a guess that is too high | Prints correct message and proceeds to next guess | Pass |  |
| 2 | Enter a guess that is too low | Prints correct message and proceeds to next guess | Pass |  |
| 3 | Enter correct answer | Prints correct message and skips the next guess | Pass |  |
| 4 | Enter a negative number | Code correctly re-prompts the user until valid input is entered | Pass |  |
| 5 | Enter a character | Code correctly re-prompts the user until valid input is entered | Fail | Code crashes because it is expecting an integer. |

NOTE – Do NOT use the above example for your HW. Your HW should have test cases related to Project 2.