# Homework: Test Techniques

## Equivalence Partitioning / Boundary Value Analysis – Income Checker

Now that you are familiar with the Equivalence Partitioning / Boundary Value Analysis Techniques, let's recall [**The Income Checker App**](http://softuni-qa-loadbalancer-2137572849.eu-north-1.elb.amazonaws.com/income-checker/) from the QA Basics course, that categorizes the given **monthly income** into one of the following categories: "**low**", "**mid**", "**high"**. It works as follows:

* If the income is less than 1000, returns **"low"**
* If the income between 1000 (inclusively) and 3000 (exclusively), returns **"mid"**
* If the income is equal or bigger than 3000, returns **"high"**
* If the income is negative, returns **"error"**

**Your task is:**

**Equivalence Partitioning:** Divide the possible input values of the "**income**" into different equivalence classes or partitions. Remember to include both valid and invalid partitions.

**Boundary Value Analysis:** Identify the boundary values of the defined partitions and come up with test cases that include these boundary values. Ensure you consider "**edge cases**" - values just outside of valid ranges.

**Note:** Keep in mind that testing should cover not only expected or valid inputs but also unexpected or invalid ones. Consider all possible scenarios that might be encountered in a real-world situation.

**Equivalence Partitioning Test Cases including invalid cases:**

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **Input** | **Expected Output** |
| TC01 | 500 | "low" |
| TC01 | 2000 | “mid” |
| TC02 | 5000 | “high” |
| TC03 | 555.55 | "low" |
| TC04 | 2635.63 | “mid” |
| TC05 | 5267.80 | “high” |
| TC06 | -0 | “error” |
| TC07 | 0 | "low" |
| TC08 | -300 | “error” |
| TC09 | -333.33 | “error” |
| TC10 | string | “error” |
| TC11 | 100000000000000000000000000 | “high” |
| TC12 | -100000000000000000000000000 | “error” |
| TC13 | 7 | “low” |
| TC14 | @ | “error” |
| TC15 | 0.25 | “low” |
| TC16 | -0.25 | “error” |

**Boundary Value Analysis Test Cases including invalid cases:**

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **Input** | **Expected Output** |
| TC07 | 999 | "low" |
| TC17 | 999.99 | "low" |
| TC18 | 1000.01 | “mid” |
| TC19 | 2999.99 | “mid” |
| TC20 | 3000.01 | “high” |
| TC21 | -3000.01 | “error” |
| TC22 | 300- | “error” |
| TC23 | -0.01 | “error” |
| TC24 | 30..01 | “error” |
| TC25 | 30+5 | “error” |

## 2. Pairwise Testing - eCommerce Checkout Function

Assume you have a checkout function of an eCommerce application for testing. The function contains the following fields with their input values:

**Drop-down menu that contains 5 different shipping methods (input values – 1, 2, 3, 4, 5);**

**Radio button for gift wrapping (input values – Yes or No);**

**Checkbox for agreeing to terms and conditions (input values - Checked or Unchecked);**

**Place Order button (input values - Does not accept any value, only finalizes the order).**

**Your task is:**

1. Calculate how many would be the positive test cases, if you have to cover every single possibility?

|  |
| --- |
| **Your Answer:** 20 test cases |

Using Pairwise testing, reduce the number of necessary test cases.

|  |
| --- |
| **Add a screenshot of the reduced test cases here** |

We have only considered positive test cases so far. What about negative ones? Write at least 3 negative test cases.

|  |
| --- |
| Example: Attempt to place an order with no shipping method selected. |
| Attempt to place an order without agreeing to terms and conditions. |
| Try to select more than one shipping method. |
| Try to finalize without changing anything in the previous fields. |