

Postal Rate Calculator Application Development

Name: Zeyu Chen 260736792

Name: Chen He 260743776

The application is a command line program written in C#.

The application takes no arguments but read inputs from stdin.

\$ from to length width height weight type

Where

from is the postcode where you send the package (length of six, with the format of Character|number| Character|number| Character|number)

to is the postcode of the destination (length of six, with the format of Character|number| Character|number| Character|number)

length is the package length in centimeters (cm)

width is the package width in centimeters (cm)

height is the package height in centimeters (cm)

weight is the package weight in kilograms (kg)

type is the type of post (Regular, Xpress, Priority)

The rules for postal rate calculation are as following:

A standard package has a minimum length of 14 cm and a maximum length of 24.5 cm. A standard package has a minimum width of 9 cm and a maximum width of 15.6 cm. A standard package has a minimum height of 15 cm and a maximum height of 35 cm. A standard package has a minimum weight of 0.3 kg. A standard package has a maximum weight of 3 kg.

Non-standard packages cannot exceed a length of 38 cm. Non-standard packages cannot exceed a width of 27 cm. Non-standard packages cannot exceed a width of 50 cm. Non-standard packages cannot exceed a weight of 5 kg.

In regular type the postal rate is as following:

The postal rate for standard packages up to 3 kg is \$0.49.

The postal rate for standard packages over 3 kg up to 5 kg is \$0.80.

The postal rate for non-standard packages up to 1 kg is \$0.98.

The postal rate for non-standard packages over 1 kg up to 5 kg is \$2.4.

In Xpress type the postal rate is 1.5 times the postal rate in regular type in each situation.
In Priority type the postal rate is 2 times the postal rate in regular type in each situation.

For calculation between two postcode:

Since the post rate is based on postal area and we do not know the exact formula that is used by CanadaPost, we assume that the postal rate is computed by taking the difference of the first letter of the postal code (which indicate the region) starting at 9\$. The final price would be the product of $(1 + \text{posttype}/2)$ and the sum of the package cost and the postal rate.

Steps:

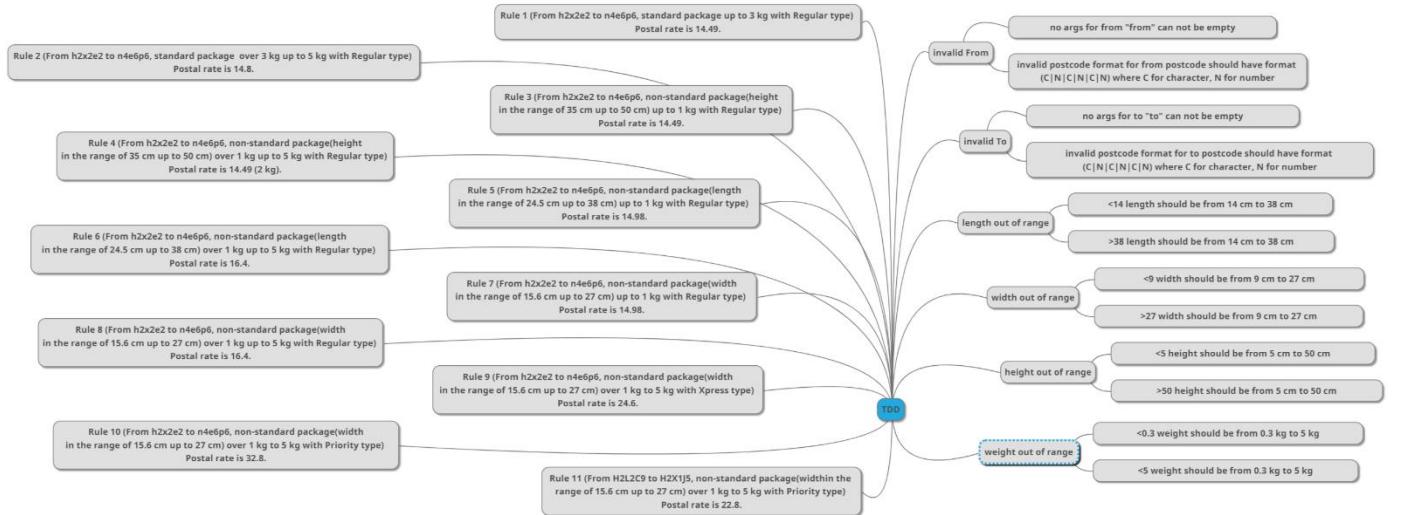
1. Write all unit-tests and declare not implemented methods
2. Write the documentation for unit-tests, then screenshot failed tests
3. Implement the minimum to pass all tests then screenshot.

Command line program:

**Calculates the postal rate of the package with from, to, length, width, height, weight, type
as input value**
Implemented with Test Driven Development.

[**Unit Test Mind Map**](#)

A mind map was created based on reviewing the requirements to guide development of the unit tests required to implement the postal rate calculator.



Initialization

Initialize the data lookup table.

Valid package: from:h2l2c9, to:h2x1j5, length:15, width:10, height:30, weight:2, PostType:Regular

Test 1

1. Test name: no_args for from

2. Call Setup: create a new package from valid package.

Let package.from equal null.

Calculator.getPrice (package).

3. Expected Result: "invalid from"

4. This test checks if the "to" section is empty and prevent null pointer exceptions

Screenshot of Visual Studio showing a failed test named `testNullFrom`. The code in `calculator.cs` contains a logic error where it fails to handle null input correctly.

```
calculator.cs
46 //throw new NotImplementedException();
47 if (validate(package) != null)
48 {
49     return validate(package);
50 }
51
52 float sum = DataSheet.getPrice(package.length, package.width, package.height);
53 int distance = postalheadlist.IndexOf(package.from[0]);
54 distance = Math.Abs(postalheadlist.IndexOf(package.to[0]) - distance);
55 sum += distance + 8;
56 sum *= (1+(int)package.type/2.0f);
57
58     return sum.ToString();
59 }
60 private static string validate(package package)
61 {
62     return null;
63 }
```

The `Output` window shows the test results:

```
[2018/3/3 20:08:19 Informational] ----- Run test started -----
[2018/3/3 20:08:20 Informational] ===== Run test finished: 1 run (0:00:00.8522667) =====
```

Screenshot of Visual Studio showing a passed test named `testNullFrom`. The code in `calculator.cs` has been modified to handle null input correctly by checking if `string.IsNullOrEmpty(package.from)`.

```
calculator.cs
51
52 float sum = DataSheet.getPrice(package.length, package.width, package.height);
53 int distance = postalheadlist.IndexOf(package.from[0]);
54 distance = Math.Abs(postalheadlist.IndexOf(package.to[0]) - distance);
55 sum += distance + 8;
56 sum *= (1+(int)package.type/2.0f);
57
58     return sum.ToString();
59 }
60 private static string validate(package package)
61 {
62     if (string.IsNullOrEmpty(package.from))
63     {
64         return "invalid from";
65     }
66     return null;
67 }
```

The `Output` window shows the test results:

```
[2018/3/3 20:13:06 Informational] ----- Run test started -----
[2018/3/3 20:13:06 Informational] ===== Run test finished: 1 run (0:00:00.2506662) =====
```

Test 2

1. Test name: no_args for to
2. Call Setup: create a new package from valid package.

Let package.to equal null.

Calculator.getPrice (package).

3. Expected Result: "invalid to"
4. This test checks if the "to" section is empty and prevent null pointer exceptions

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Test, Analyze, Window, and Help. The status bar on the right displays "Zeyu Chen".

The left sidebar contains icons for Server Explorer, Toolbox, and Data Sources. The Test Explorer window is open, showing a summary of test runs. It lists one failed test, "testNullTo", which took 76 ms, and one passed test, "testNullFrom", which took 45 ms. The Failed Tests section is expanded. The Output window at the bottom shows the command-line output of the test run, indicating it started at 2018/3/3 20:20:25 and finished at 2018/3/3 20:20:25, with a total duration of 0:00:00.4561206.

The main code editor window displays two files: "calculator.cs" and "CalculatorTests.cs". The "calculator.cs" file contains a method named "validate" that calculates a sum based on postal headlists and returns it as a string. The "CalculatorTests.cs" file contains unit tests for the "validate" method, including the failing "testNullTo" test which checks for a null "from" parameter and returns "invalid from".

```
calculator.cs
private static string validate(package package)
{
    if (string.IsNullOrEmpty(package.from))
    {
        return "invalid from";
    }
    return null;
}

CalculatorTests.cs
[TestMethod]
public void testNullTo()
{
    package = null;
    Assert.AreEqual("invalid to", calculator.validate(package));
}
```

The screenshot shows the Microsoft Visual Studio interface. The top menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Test, Analyze, Window, and Help. The status bar at the bottom right shows "Zeyu Chen".

The left sidebar contains links for Server Explorer, Toolbox, Data Sources, and a search bar.

The Test Explorer window on the left displays "Passed Tests (2)" with entries: "testNullFrom" (12 ms) and "testNullTo" (< 1 ms). It also lists "Not Run Tests (10)". A tooltip for "testNullTo" indicates it is from "CalculatorTests.cs line 33", passed, and took 0:00:00.0004976.

The main code editor window shows the file "calculator.cs" with the following code:

```
calculator.cs
428-assignment2
private static string validate(package package)
{
    if (string.IsNullOrEmpty(package.from))
    {
        return "invalid from";
    }
    if (string.IsNullOrEmpty(package.to))
    {
        return "invalid to";
    }
    return null;
}

sum *= (1+(int)package.type/2.0f);

return sum.ToString();
```

The Output window at the bottom shows test results:

```
Output
Show output from: Tests
[2018/3/3 20:23:24 Informational] ----- Run test started -----
[2018/3/3 20:23:24 Informational] ----- Run test finished: 2 run (0:00:00.19552) -----
```

Test3

1. Test name: out_of_range_length_low
2. Call Setup: create a new package from valid package.

Let package.length=13.9.

Calculator.getPrice (package).

3. Expected Result: "invalid length"
4. It was decided that the range of valid length would be extended down to 14cm.

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Test, Analyze, Window, and Help. The toolbar below has icons for New, Open, Save, Print, and others. The status bar at the bottom right shows "Zeyu Chen".

The main window displays the "calculator.cs" file under the "428-assignment2" project. The code defines a static method `validate` that checks if `package.from` and `package.to` are non-empty strings. If either is empty, it returns "invalid from" or "invalid to" respectively. If the package length is less than 14, it returns "invalid length". Otherwise, it returns the sum's string representation.

The "Test Explorer" window on the left shows the following test results:

- Run All
- Passed Tests (2):
 - testLengthLowBound (43 ms)
 - testNullFrom (1 ms)
- Not Run Tests (9)

Summary
Last Test Run Failed (Total Run Time 0:00:00.237632)
1 Test Failed
2 Tests Passed

The "Output" window at the bottom shows the following log entries:

```
[2018/3/3 20:24:55 Informational] ----- Run test started -----  
[2018/3/3 20:24:56 Informational] ===== Run test finished: 3 run (0:00:00.237632) =====
```

This screenshot is identical to the one above, showing the Microsoft Visual Studio IDE interface and the "calculator.cs" file. The "Test Explorer" window highlights the "testLengthLowBound" test, which passed in 16 ms. The tooltip for this test provides details: "Source: CalculatorTests.cs line 40", "Test Passed - testLengthLowBound", and "Elapsed time: 0:00:00.0168184".

The "Output" window shows the same log entries as the previous screenshot.

Test4

1. Test name: out_of_range_length_high

2. Call Setup: create a new package from valid package.

Let package.length=38.1.

Calculator.getPrice (package).

3. Expected Result: "invalid length"

4. It was decided that the range of valid length would be extended up to 38cm.

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Test, Analyze, Window, and Help. The status bar on the right shows the name "Zeyu Chen".

The left sidebar contains icons for Server Explorer, Toolbox, and Data Sources. The Test Explorer window is open, showing a list of tests. It has sections for Failed Tests (1) and Passed Tests (3). The Failed Test is "testLengthHighBound" with a duration of 42 ms. The Passed Tests are "testLengthLowBound" (17 ms), "testNullFrom" (< 1 ms), and "testNullTo" (< 1 ms). Below the Failed Tests, there is a "Summary" section indicating "Last Test Run Failed (Total Run Time 0:00:00.258564)" with "1 Test Failed" and "3 Tests Passed".

The Solution Explorer window shows files: calculator.cs and CalculatorTests.cs. The calculator.cs file contains the following code:

```
calculator.cs
private static string validate(package package)
{
    if (string.IsNullOrEmpty(package.from))
    {
        return "invalid from";
    }
    if (string.IsNullOrEmpty(package.to))
    {
        return "invalid to";
    }
    if (package.length < 14)
    {
        return "invalid length";
    }
    return null;
}
```

The CalculatorTests.cs file contains test methods for the validate function.

The Output window at the bottom shows the following log entries:

```
Output
Show output from: Tests
[2018/3/3 20:32:50 Informational] ----- Run test started -----
[2018/3/3 20:32:50 Informational] ----- Run test finished: 4 run (0:00:00.2585641) -----
```

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Test, Analyze, Window, and Help. The toolbar below has icons for New, Open, Save, Print, and others. The status bar at the bottom right shows "Zeyu Chen".

Test Explorer: Displays "Passed Tests (4)" including testLengthHighBound, testLengthLowBound, testNullFrom, and testNullTo. It also lists "Not Run Tests (8)". A tooltip for testLengthHighBound shows the source code line 49 and a test summary.

Solution Explorer: Shows the project structure with files like calculator.cs and CalculatorTests.cs.

CalculatorTests.cs: The code for the validate method is shown:

```

62     if (string.IsNullOrEmpty(package.from))
63     {
64         return "invalid from";
65     }
66     if (string.IsNullOrEmpty(package.to))
67     {
68         return "invalid to";
69     }
70     if (package.length < 14)
71     {
72         return "invalid length";
73     }
74     if (package.length >= 38)
75     {
76         return "invalid length";
77     }
78 }
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121

```

Output: The output window shows informational logs from the test run:

```

[2018/3/3 20:32:50 Informational] ----- Run test started -----
[2018/3/3 20:32:50 Informational] ----- Run test finished: 4 run (0:00:00.2585641) -----
[2018/3/3 20:34:06 Informational] ----- Discover test started -----
[2018/3/3 20:34:07 Informational] ----- Discover test finished: 12 found (0:00:00.544485) -----
[2018/3/3 20:34:07 Informational] ----- Run test started -----
[2018/3/3 20:34:07 Informational] ----- Run test finished: 4 run (0:00:00.2135688) -----

```

Test5

1. Test name: out_of_range_width_low

2. Call Setup: create a new package from valid package.

Let package.width=8.9.

Calculator.getPrice (package).

3. Expected Result: "invalid width"

4. It was decided that the range of valid width would be extended down to 9cm.

Screenshot of Visual Studio showing test results for `calculator.cs`. The Test Explorer shows 1 Failed Test (`testWidthLowBound`) and 4 Passed Tests (`testLengthHighBound`, `testLengthLowBound`, `testNullFrom`, `testNullTo`). The code editor shows the implementation of the `validate` method.

```
calculator.cs (calculatorTests.cs) -> _428_assignment2.calculator -> validate(package package)

64     return "invalid from";
65 }
66 if (string.IsNullOrEmpty(package.to))
67 {
68     return "invalid to";
69 }
70 if (package.length < 14)
71 {
72     return "invalid length";
73 }
74 if (package.length >= 38)
75 {
76     return "invalid length";
77 }
78
79     return null;
80 }
81 }
```

Screenshot of Visual Studio showing test results for `calculator.cs`. The Test Explorer shows 5 Passed Tests (`testLengthHighBound`, `testLengthLowBound`, `testNullFrom`, `testNullTo`, `testWidthLowBound`) and 7 Not Run Tests. The code editor shows the implementation of the `validate` method.

```
calculator.cs (calculatorTests.cs) -> _428_assignment2.calculator -> validate(package package)

66     return "invalid to";
67 }
68 if (package.length < 14)
69 {
70     return "invalid length";
71 }
72 if (package.length >= 38)
73 {
74     return "invalid length";
75 }
76 if (package.width < 9)
77 {
78     return "invalid width";
79 }
80
81     return null;
82 }
83 }
```

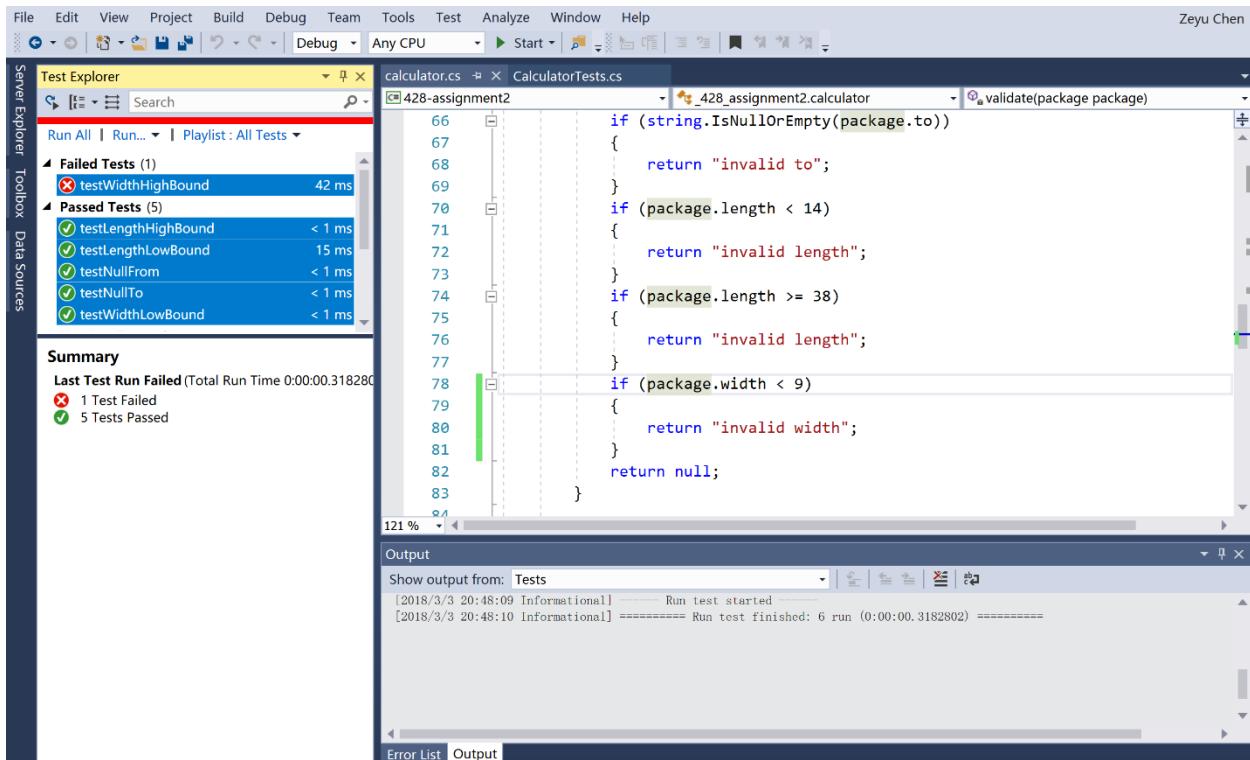
Test6

1. Test name: out_of_range_width_high
 2. Call Setup: create a new package from valid package.

Let package.width=27.1.

Calculator.getPrice (package).

3. Expected Result: Usage: "invalid width"
 4. It was decided that the range of valid width would be extended up to 27cm.



The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Test, Analyze, Window, and Help. The status bar at the bottom right shows "Zeyu Chen".

Test Explorer: Displays "Passed Tests (6)" including testLengthHighBound, testLengthLowBound, testNullFrom, testNullTo, testWidthHighBound, and testWidthLowBound, all completed in less than 1 ms.

Solution Explorer: Shows the project structure with files like calculator.cs, CalculatorTests.cs, and _428_assignment2.calculator.

CalculatorTests.cs: The code contains several if statements to validate package dimensions. The highlighted line is "if (package.width >= 27)".

```

calculator.cs 428-assignment2.calculator validate(package package)
calculator.cs
70     if (package.length < 14)
71     {
72         return "invalid length";
73     }
74     if (package.length >= 38)
75     {
76         return "invalid length";
77     }
78     if (package.width < 9)
79     {
80         return "invalid width";
81     }
82     if (package.width >= 27)
83     {
84         return "invalid width";
85     }
86 }
87

```

Output: The output window shows the test run results:

```

[2018/3/3 20:49:20 Informational] Run test started
[2018/3/3 20:49:20 Informational] ===== Run test finished: 6 run (0:00:00.4040755) =====

```

Test7

1. Test name: out_of_range_height_low

2. Call Setup: create a new package from valid package.

Let package.height=4.9.

Calculator.getPrice (package).

3. Expected Result: "invalid height"

4. It was decided that the range of valid height would be extended down to 5cm.

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Test, Analyze, Window, and Help. The left sidebar has icons for Server Explorer, Toolbox, and Data Sources. The main window has tabs for 'Test Explorer' and 'calculator.cs'. The 'Test Explorer' tab is active, displaying a list of tests: 'Failed Tests (1)' containing 'testHeightLowBound' (33 ms) with a red 'X' icon, and 'Passed Tests (6)' containing 'testLengthHighBound', 'testLengthLowBound', 'testNullFrom', 'testNullTo', 'testWidthHighBound', and 'testWidthLowBound', each with a green checkmark icon and a time less than 1 ms. Below this is a 'Summary' section stating 'Last Test Run Failed (Total Run Time 0:00:00.241643)', '1 Test Failed' with a red 'X' icon, and '6 Tests Passed' with a green checkmark icon. The 'calculator.cs' tab is active in the code editor, showing the following C# code:

```
if (package.length < 14)
{
    return "invalid length";
}
if (package.length >= 38)
{
    return "invalid length";
}
if (package.width < 9)
{
    return "invalid width";
}
if (package.width >= 27)
{
    return "invalid width";
}
return null;
```

The code editor shows line numbers 70 through 87. A vertical green bar highlights lines 78 and 85. A yellow warning lightbulb icon is on line 82. The bottom status bar shows the zoom level as 121%.

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Test, Analyze, Window, and Help. The left sidebar contains the Server Explorer, Toolbox, and Data Sources. The main area displays the Test Explorer, Solution Explorer, and Code Editor.

Test Explorer: Shows 7 Passed Tests and 5 Not Run Tests. The first test, `testHeightLowBound`, is selected and expanded, showing its source code at line 86 of `CalculatorTests.cs`. The test passed with an elapsed time of 0:00:00.0002863.

Solution Explorer: Shows the project structure with files like `calculator.cs` and `CalculatorTests.cs`.

Code Editor: Displays the `calculator.cs` file, specifically the `validate` method. The code checks package dimensions against bounds (length, width, height) and returns an error message or null.

```
calculator.cs  x  CalculatorTests.cs
428-assignment2  428_assignment2.calculator  validate(package package)
73 } if (package.length >= 38) { return "invalid length"; }
74 } if (package.width < 9) { return "invalid width"; }
75 } if (package.width >= 27) { return "invalid width"; }
76 } if (package.height < 15) { return "invalid height"; }
77 } return null;
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
```

Output: Shows the test results from the command line, indicating the test started at 2018/3/3 20:52:42 and finished at 2018/3/3 20:52:42 with 7 runs in 0:00:00.3539225.

Test8

- ## 1. Test name: out_of_range_height_high

2. Call Setup: create a new package from valid package.

Let package.height=50.1.

Calculator.getPrice (package).

3. Expected Result: "invalid height"

4. It was decided that the range of valid height would be extended up to 50cm.

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Test, Analyze, Window, and Help. The status bar on the right shows "Zeyu Chen".

The Test Explorer window on the left displays test results:

- Failed Tests (1): testHeightHighBound (30 ms)
- Passed Tests (7): testHeightLowBound, testLengthHighBound, testLengthLowBound, testNullFrom, testNullTo, testWidthHighBound, testWidthLowBound

The Solution Explorer window shows files calculator.cs and CalculatorTests.cs.

The code editor window displays calculator.cs with the following content:

```
calculator.cs
428-assignment2.calculator
validate(package package)
if (package.length >= 38)
{
    return "invalid length";
}
if (package.width < 9)
{
    return "invalid width";
}
if (package.width >= 27)
{
    return "invalid width";
}
if (package.height < 15)
{
    return "invalid height";
}
return null;
```

The Output window at the bottom shows:

```
Output
Show output from: Tests
[2018/3/3 20:53:26 Informational] ----- Run test started -----
[2018/3/3 20:53:26 Informational] ===== Run test finished: 8 run (0:00:00.2187496) =====
```

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Test, Analyze, Window, and Help. The toolbar below has icons for file operations like Open, Save, and Print. The status bar at the bottom right shows "Zeyu Chen".

Test Explorer: Displays a list of passed tests under "Passed Tests (8)". The selected test is "testHeightHighBound" which passed in 1 ms. Other tests listed include "testHeightLowBound", "testLengthHighBound", "testLengthLowBound", "testNullFrom", "testNullTo", "testWidthHighBound", and "testWidthLowBound".

Solution Explorer: Shows the project structure with files like "calculator.cs", "CalculatorTests.cs", and "428_assignment2.calculator".

Code Editor: The main window displays the code for "calculator.cs". The code checks package dimensions against bounds. It returns "invalid length" if width is less than 9, "invalid width" if width is greater than or equal to 27, "invalid height" if height is less than 15, and "invalid weight" if weight is greater than or equal to 50.

Output: The output window shows the results of the test run: "Run test started" and "Run test finished: 8 run (0:00:00.4361604)".

Test9

1. Test name: out_of_range_weight_low
2. Call Setup: create a new package from valid package.

Let package.weight=0.2.

Calculator.getPrice (package).

3. Expected Result: "invalid weight"
4. It was decided that the range of valid weight would be extended down to 0.2kg.

Screenshot of Visual Studio showing the Test Explorer and code editor.

Test Explorer:

- Run All | Run... | Playlist : All Tests
- Failed Tests (1): testWeightLowBound (56 ms)
- Passed Tests (8):
 - testHeightHighBound < 1 ms
 - testHeightLowBound < 1 ms
 - testLengthHighBound < 1 ms
 - testLengthLowBound 1 ms
 - testNullFrom < 1 ms
 - testNullTo < 1 ms
 - testWidthHighBound 1 ms
 - testWidthLowBound < 1 ms

Summary:

Last Test Run Failed (Total Run Time 0:00:00.231675)
 ✘ 1 Test Failed
 ✓ 8 Tests Passed

Code Editor:

```

    77 } if (package.width < 9)
    78 { return "invalid width";
    79 } if (package.width >= 27)
    80 { return "invalid width";
    81 } if (package.height < 15)
    82 { return "invalid height";
    83 } if (package.height >= 50)
    84 { return "invalid height";
    85 }
    86 } return null;
    87
    88
    89
    90
    91
    92
    93
    94
  
```

Output:

```

[2018/3/3 20:55:43 Informational] ----- Run test started -----
[2018/3/3 20:55:43 Informational] ===== Run test finished: 9 run (0:00:00.2316759) =====
  
```

Screenshot of Visual Studio showing the Test Explorer and code editor.

Test Explorer:

- Run All | Run... | Playlist : All Tests
- Passed Tests (9):
 - testHeightHighBound < 1 ms
 - testHeightLowBound < 1 ms
 - testLengthHighBound < 1 ms
 - testLengthLowBound < 1 ms
 - testNullFrom < 1 ms
 - testNullTo < 1 ms
 - ✓ testWeightLowBound 14 ms
 - ✓ testWidthHighBound < 1 ms
 - ✓ testWidthLowBound < 1 ms
- Not Run Tests (3)

Selected Test: testWeightLowBound

Source: CalculatorTests.cs line 79

Test Passed - testWeightLowBound
 Elapsed time: 0:00:00.0140635

Code Editor:

```

    86 if (package.height < 15)
    87 { return "invalid height";
    88 } if (package.height >= 50)
    89 { return "invalid height";
    90 } if (package.weight < 0.3)
    91 { return "invalid weight";
    92 }
    93 } return null;
    94
    95
    96
    97
    98
    99
    100
    101
    102
    103
    104
  
```

Output:

```

[2018/3/3 20:57:02 Informational] ----- Run test started -----
[2018/3/3 20:57:02 Informational] ===== Run test finished: 9 run (0:00:00.3008628) =====
  
```

Test10

1. Test name: out_of_range_weight_high

2. Call Setup: create a new package from valid package.

Let package.weight=5.1.

Calculator.getPrice (package).

3. Expected Result: "invalid weight"

4. It was decided that the range of valid weight would be extended up to 5kg.

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Test, Analyze, Window, and Help. The status bar on the right shows the user's name, Zeyu Chen. The left sidebar contains icons for Server Explorer, Toolbox, and Data Sources. The main workspace is divided into several windows:

- Test Explorer**: Shows a list of tests. Under Failed Tests, one test is listed: "testWeightHighBound" (37 ms). Under Passed Tests, nine tests are listed: "testHeightHighBound", "testHeightLowBound", "testLengthHighBound", "testLengthLowBound", "testNullFrom", "testNullTo", "testWeightLowBound", "testWidthHighBound", and "testWidthLowBound", all with execution times less than 1 ms.
- Solution Explorer**: Shows the project structure with files like calculator.cs, CalculatorTests.cs, and validate(package package).
- calculator.cs**: The code for the calculator class. It includes methods for validating package dimensions and weight. The relevant part of the code is:

```
83     {
84         return "invalid width";
85     }
86     if (package.height < 15)
87     {
88         return "invalid height";
89     }
90     if (package.height >= 50)
91     {
92         return "invalid height";
93     }
94     if (package.weight < 0.3)
95     {
96         return "invalid weight";
97     }
98     return null;
99 }
100 }
```
- Output**: Shows log messages indicating the test run started and finished successfully.

```
[2018/3/3 20:58:24 Informational] ----- Run test started -----
[2018/3/3 20:58:24 Informational] ===== Run test finished: 10 run (0:00:00.279744) =====
```

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Test, Analyze, Window, and Help. The toolbar below has icons for New, Open, Save, Print, and various project management tools. The title bar displays "calculator.cs" and "CalculatorTests.cs".

Test Explorer: Shows 10 passed tests under "Passed Tests" and 2 not run tests under "Not Run Tests". The test "testWeightHighBound" is selected, showing its details: Source: CalculatorTests.cs line 72, Test Passed - testWeightHighBound, Elapsed time: 0:00:00.0001842.

Solution Explorer: Shows the project structure with files like calculator.cs, CalculatorTests.cs, and validate(package package).

Output Window: Displays log messages from the test run: [2018/3/3 20:59:26 Informational] ----- Run test started ----- and [2018/3/3 20:59:27 Informational] ===== Run test finished: 10 run (0:00:00.2025752) =====.

```

calculator.cs
if (package.height >= 50)
{
    return "invalid height";
}
if (package.weight < 0.3)
{
    return "invalid weight";
}
if (package.weight >= 5)
{
    return "invalid weight";
}
return null;

```

Test11

1. Test name: invalid postcode_format_from
2. Call Setup: create a new package from valid package.

Let package.from="hhhhh"

Calculator.getPrice (package).

Let package.from="h2x2e2e"

Calculator.getPrice (package).

Let package.from="111111"

Calculator.getPrice (package).

3. Expected Result for all 3 situations: "invalid from"

4. This test checks whether the postcode in "from" section is right format.

Screenshot of Visual Studio showing test results for `calculator.cs`. The Test Explorer shows 1 failed test (`testPostalCodeFormatFrom`) and 10 passed tests. The code editor shows the implementation of the `calculator` class with logic for height, weight, and from validation.

```

calculator.cs  x CalculatorTests.cs
428-assignment2   _428_assignment2.calculator validate(package package)

86     if (package.height < 15)
87     {
88         return "invalid height";
89     }
90     if (package.height >= 50)
91     {
92         return "invalid height";
93     }
94     if (package.weight < 0.3)
95     {
96         return "invalid weight";
97     }
98     if (package.weight >= 5)
99     {
100        return "invalid weight";
101    }
102    return null;
103 }

121 %

```

Test Explorer

- Failed Tests (1)
 - `testPostalCodeFormatFrom` 65 ms
- Passed Tests (10)
 - `testHeightHighBound` < 1 ms
 - `testHeightLowBound` 1 ms
 - `testLengthHighBound` < 1 ms
 - `testLengthLowBound` < 1 ms
 - `testNullFrom` < 1 ms
 - `testNullTo` < 1 ms
 - `testWeightHighBound` < 1 ms
 - `testWeightLowBound` 20 ms
 - `testWidthHighBound` < 1 ms
 - `testWidthLowBound` < 1 ms

Summary

Last Test Run Failed (Total Run Time 0:00:00.334892)

1 Test Failed
10 Tests Passed

Output

```

Show output from: Tests
[2018/3/3 21:04:49 Informational] ----- Run test started -----
[2018/3/3 21:04:49 Informational] ----- Run test finished: 11 run (0:00:00.334892) -----

```

Screenshot of Visual Studio showing test results for `calculator.cs`. The Test Explorer shows 11 passed tests. The code editor shows the implementation of the `calculator` class with additional logic for validating the `from` field.

```

calculator.cs  x CalculatorTests.cs
428-assignment2   _428_assignment2.calculator validate(package package)

92     return "invalid height";
93
94     if (package.weight < 0.3)
95     {
96         return "invalid weight";
97     }
98     if (package.weight >= 5)
99     {
100        return "invalid weight";
101    }
102    if (package.from.Length != 6 || !(char.IsLetter(package.from[0]) &&
103        package.from[1] == ' ' && package.from[2] == ' ' &&
104        package.from[3] == ' ' && package.from[4] == ' ' &&
105        package.from[5] == ' ')
106    {
107        return "invalid from";
108    }
109
110    return null;
111 }

121 %

```

Test Explorer

- Passed Tests (11)
 - `testHeightHighBound` < 1 ms
 - `testHeightLowBound` < 1 ms
 - `testLengthHighBound` < 1 ms
 - `testLengthLowBound` < 1 ms
 - `testNullFrom` < 1 ms
 - `testNullTo` < 1 ms
 - `testPostalCodeFormatFrom` < 1 ms
 - `testWeightHighBound` < 1 ms
 - `testWeightLowBound` 58 ms
 - `testWidthHighBound` < 1 ms
 - `testWidthLowBound` < 1 ms

testPostalCodeFormatFrom Copy All

Source: `CalculatorTests.cs` line 101

Test Passed - `testPostalCodeFormatFrom`
Elapsed time: 0:00:00.0006371

Output

```

Show output from: Tests
[2018/3/3 21:05:47 Informational] ----- Run test started -----
[2018/3/3 21:05:47 Informational] ----- Run test finished: 11 run (0:00:00.3709859) -----

```

Test12

1. Test name: invalid_postcode_format_to

2. Call Setup: create a new package from valid package.

Let package.to="hhhhh"

Calculator.getPrice (package).

Let package.to="h2x2e2e"

Calculator.getPrice (package).

Let package.to="111111"

Calculator.getPrice (package).

3. Expected Result for all 3 situations: "invalid to"

4. This test checks whether the postcode in "from" section is right format.

The screenshot shows the Microsoft Visual Studio interface with the following windows visible:

- Test Explorer**: Shows 1 Failed Test (testPostalCodeFormatFromTo) and 11 Passed Tests (testHeightHighBound, testHeightLowBound, testLengthHighBound, testLengthLowBound, testNullFrom, testNullTo, testPostalCodeFormatFrom, testWeightHighBound, testWeightLowBound). The failed test took 25 ms.
- Solution Explorer**: Shows the project structure with files like calculator.cs and CalculatorTests.cs.
- Code Editor**: Displays the code for calculator.cs, specifically the validate method which handles invalid input cases based on height, weight, and from/postal code format.
- Output**: Shows the command-line output of the test run, indicating the test started at 2018/3/3 23:31:46 and finished at 2018/3/3 23:31:46.

```
calculator.cs
428-assignment2.calculator
validate(package package)

91     {
92         return "invalid height";
93     }
94     if (package.weight < 0.3)
95     {
96         return "invalid weight";
97     }
98     if (package.weight >= 5)
99     {
100        return "invalid weight";
101    }
102    if (package.from.Length != 6 || !(char.IsLetter(package.from[0]) &&
103        package.from[0] != 'A' && package.from[0] != 'B' &&
104        package.from[0] != 'C' && package.from[0] != 'D' &&
105        package.from[0] != 'E' && package.from[0] != 'F' &&
106        package.from[0] != 'G' && package.from[0] != 'H' &&
107        package.from[0] != 'I' && package.from[0] != 'J' &&
108        package.from[0] != 'K' && package.from[0] != 'L' &&
109        package.from[0] != 'M' && package.from[0] != 'N' &&
110        package.from[0] != 'O' && package.from[0] != 'P' &&
111        package.from[0] != 'Q' && package.from[0] != 'R' &&
112        package.from[0] != 'S' && package.from[0] != 'T' &&
113        package.from[0] != 'U' && package.from[0] != 'V' &&
114        package.from[0] != 'W' && package.from[0] != 'X' &&
115        package.from[0] != 'Y' && package.from[0] != 'Z'))
116    {
117        return "invalid from";
118    }
119    return null;
120 }
```

The screenshot shows the Microsoft Visual Studio interface. The top menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Test, Analyze, Window, and Help. The status bar on the right shows "Zeyu Chen".

- Test Explorer:** Shows a list of test cases under the "calculator.cs" project. One test, "testPostalCodeFormatFromTo", is currently selected and highlighted in blue. It has a green checkmark and a duration of < 1 ms.
- Solution Explorer:** Shows the project structure with files like "calculator.cs", "CalculatorTests.cs", and "428_assignment2.cs".
- Code Editor:** Displays the code for "calculator.cs". The selected method is "validate(package package)". The code checks various parameters like weight, length, width, and postal codes for validity.
- Output Window:** Shows the command-line output of the test run. It includes the start time (2018/3/3 23:33:21), the start of the run, and the end time (2018/3/3 23:33:25) when the run finished with 12 runs in 0:00:00.3440218.

Test13

1. Test name: valid_Rule1

2. Call Setup: create a new package from valid package.

Let package=new package ("h2x2e2", "n4e6p6", 15, 10, 20, 1, PostType.Regular)

Calculator.getPrice (package).

3. Expected Result: "14.49"

4. This test checks if the answer of the postal rate for standard packages up to 3 kg in regular type from one fixed place to a fixed destination is right.

Ecs428_AssignmentB - Microsoft Visual Studio (Administrator)

Test Explorer

- Run All | Run ~ | Playlist: All Tests
- Failed Tests (12)**
 - testPostCapital
 - testValid1
 - testValid10
 - testValid11
 - testValid2
 - testValid3
 - testValid4
 - testValid5
 - testValid6
 - testValid7
 - testValid8
 - testValid9
- Passed Tests (13)**
 - testLengthHighbound
 - testLengthLowbound
 - testLengthHighbound
 - testLengthLowbound
 - testLengthHighbound
 - testLengthLowbound
 - testWidthHighbound
 - testWidthLowbound
 - testWidthHighbound
 - testWidthLowbound
 - testWidthHighbound
 - testWidthLowbound

Solution Explorer

- Solution Ecs428_AssignmentB [2 projects]**
 - Ecs428_AssignmentB**
 - Properties
 - Calcs.cs
 - DataSheet.cs
 - Program.cs
 - UnitTestProject**
 - Dependencies
 - UnitTest.cs

Properties

Ecs428_AssignmentB - Microsoft Visual Studio (Administrator)

Test Explorer

- Run All | Run ~ | Playlist: All Tests
- Failed Tests (3)**
 - testFailed
 - testFailed
 - testFailed
- Passed Tests (12)**
 - testLengthHighbound
 - testLengthLowbound
 - testLengthHighbound
 - testLengthLowbound
 - testLengthHighbound
 - testLengthLowbound
 - testWidthHighbound
 - testWidthLowbound
 - testWidthHighbound
 - testWidthLowbound
 - testWidthHighbound
 - testWidthLowbound

Solution Explorer

- Solution Ecs428_AssignmentB [2 projects]**
 - Ecs428_AssignmentB**
 - Properties
 - Calcs.cs
 - DataSheet.cs
 - Program.cs
 - UnitTestProject**
 - Dependencies
 - UnitTest.cs

Properties

Test14

1. Test name: valid_Rule2

2. Call Setup: create a new package from valid package.

Let package=new package ("h2x2e2", "n4e6p6" , 15, 10, 20, 4, PostType.Regular)

Calculator.getPrice (package).

3. Expected Result: "14.8"

4. This test checks if the answer of the postal rate for standard packages over 3 kg up to 5 kg in regular type from one fixed place to a fixed destination is right.

The screenshot displays two instances of Microsoft Visual Studio running side-by-side. Both instances have the following windows open:

- Solution Explorer**: Shows the project structure for "Ece428 AssignmentB" (2 projects). The top instance shows "Ece428 AssignmentB" and the bottom instance shows "Ece428 AssignmentB (2)".
- Test Explorer**: Shows the test results for both instances. Failed tests include:
 - testPostCapital (1 ms)
 - testValid1 (50 ms)
 - testValid10 (1 ms)
 - testValid11 (1 ms)
 - testValid2 (1 ms)
 - testValid3 (1 ms)
 - testValid4 (1 ms)
 - testValid5 (1 ms)
 - testValid6 (1 ms)
 - testValid7 (1 ms)
 - testValid8 (1 ms)
 - testValid9 (1 ms)
- UnitTest.cs**: Contains the following code:

```
51     this.to = model.to;
52     this.length = model.length;
53     this.width = model.width;
54     this.height = model.height;
55     this.weight = model.weight;
56     this.type = model.type;
57   }
58   public class Calc
59   {
60     public static string getPrice(package package)
61     {
62       if (validate(package) != null)
63       {
64         return validate(package);
65       }
66       float sum = DataSheet.getPrice(package.length, package.width, package.weight);
67       return sum.ToString();
68     }
69     private static string validate(package package)
70     {
71       //data validation
72       if (package == null)
73       {
74         return "invalid package";
75       }
76       if (string.IsNullOrEmpty(package.from))
77       {
78         return "invalid from";
79       }
80       if (package.from.Length != 6 || !(char.IsLetter(package.from[0]) & char.IsLetter(package.from[2]) & char.IsLetter(package.from[4])))
81       {
82         return "invalid from";
83       }
84       if (package.to.Length != 6 || !(char.IsLetter(package.to[0]) & char.IsLetter(package.to[2]) & char.IsLetter(package.to[4])))
85       {
86         return "invalid to";
87       }
88       if (package.length < 0)
89       {
90         return "invalid length";
91       }
92       if (package.width < 0)
93       {
94         return "invalid width";
95       }
96       if (package.height < 0)
97       {
98         return "invalid height";
99       }
100      if (package.weight < 0)
101      {
102        return "invalid weight";
103      }
104    }
105  }
```
- Output**: Shows the command-line output for the tests, indicating successful runs and failures.
- Ready**: Status bar at the bottom.

The bottom instance of Visual Studio also shows a successful build status in the status bar.

Test15

1. Test name: valid_Rule3
 2. Call Setup: create a new package from valid package.

```
Let package=new package ("h2x2e2", "n4e6p6", 15, 10, 38, 0.8, PostType.Regular )
```

Calculator.getPrice (package).

3. Expected Result: "14.49"

4. This test checks if the answer of the postal rate for non-standard packages (height in the range of 35cm up to 50cm) up to 1 kg in regular type from one fixed place to a fixed destination is right.

The screenshot shows the Microsoft Visual Studio interface with the following details:

- Solution Explorer:** Shows the solution structure for "Ecsel428_AssignmentB" with projects "Ecsel428_AssignmentB" and "UnitTestProject".
- Test Explorer:** Displays test results:
 - Failed Tests (12):** testPostalCapital, testWidth0, testWidth10, testWidth11, testWidth12, testWidth13, testWidth14, testWidth15, testWidth16, testWidth17, testWidth18, testWidth19.
 - Passed Tests (13):** testLengthHighBound, testLengthLowBound, testLengthHighBound, testLengthLowBound, testWidthHighBound, testWidthLowBound, testWidthHighBound, testWidthLowBound, testWidthHighBound, testWidthLowBound, testWidthHighBound, testWidthLowBound, testWidthHighBound.
- Code Editor:** Shows the implementation of the `Calc` class with methods `getPrice` and `validate`.
- Output Window:** Shows the command-line output of the tests, including log entries and test results.

The screenshot shows the Microsoft Visual Studio interface with the following details:

- Test Explorer:** Shows 3 Failed Tests (testValid9, testLengthHighBound, testWidthHighBound) and 22 Passed Tests.
- Solution Explorer:** Displays the project structure for "Ece428.AssignmentB" including files like App.config, Calc.cs, DataSheet.cs, Program.cs, and UnitTest.cs.
- Output Window:** Shows the following log entries from the test run:


```

2018-03-04 11:00:44 PM Informational Discover test finished: 25 found (0:00:00.1432446)
2018-03-04 11:00:45 PM Informational Run test started - 
2018-03-04 11:00:45 PM Informational Run test finished: 25 run (0:00:00.7410014)
2018-03-04 11:00:45 PM Informational Discover test started - 
2018-03-04 11:00:45 PM Informational Run test started - 
2018-03-04 11:00:45 PM Informational Run test finished: 25 found (0:00:01.1679979)
2018-03-04 11:00:45 PM Informational Run test started - 
2018-03-04 11:00:45 PM Informational Run test finished: 25 run (0:00:00.9061336)
      
```

Test16

1. Test name: valid_Rule4

2. Call Setup: create a new package from valid package.

Let package=new package ("h2x2e2", "n4e6p6", 15, 10, 38, 2, PostType.Regular)

Calculator.getPrice (package).

3. Expected Result: "14.49"

4. This test checks if the answer of the postal rate for non-standard packages (height in the range of 35cm up to 50cm) over 1 kg up to 5 kg in regular type from one fixed place to a fixed destination is right.

Ecs428_AssignmentB - Microsoft Visual Studio (Administrator)

Test Explorer

- Run All | Run... | Playlist: All Tests**
- Failed Tests (12)**
 - testPostalCapital
 - testValid1
 - testValid10
 - testValid11
 - testValid2
 - testValid3
 - testValid4
 - testValid5
 - testValid6
 - testValid7
 - testValid8
 - testValid9
- Passed Tests (13)**
 - testLengthHighbound
 - testLengthLowbound
 - testLengthHighbound
 - testLengthLowbound
 - testLengthHighbound
 - testLengthLowbound
 - testLengthHighbound
 - testLengthLowbound
 - testWidthHighbound
 - testWidthLowbound
 - testWidthHighbound
 - testWidthLowbound

Solution Explorer

- Solution Ecs428_AssignmentB [2 projects]**
 - Ecs428_AssignmentB**
 - Properties
 - App.config
 - Calcs.cs
 - DataSheet.cs
 - Program.cs
 - UnitTestProject**
 - Dependencies
 - UnitTest.cs

Properties

Ecs428_AssignmentB - Microsoft Visual Studio (Administrator)

Test Explorer

- Run All | Run... | Playlist: All Tests**
- Failed Tests (9)**
 - testFailed - testValid9
 - Message.Assert.AreEqual failed. Expected:<24.6>, Actual:<24.5>
- Passed Tests (22)**
 - testLengthHighbound
 - testLengthLowbound
 - testWidthHighbound
 - testWidthLowbound
 - testWidthHighbound
 - testWidthLowbound

Solution Explorer

- Solution Ecs428_AssignmentB [2 projects]**
 - Ecs428_AssignmentB**
 - Properties
 - App.config
 - Calcs.cs
 - DataSheet.cs
 - Program.cs
 - UnitTestProject**
 - Dependencies
 - UnitTest.cs

Properties

Test17

1. Test name: valid_Rule5

2. Call Setup: create a new package from valid package.

Let package=new package ("h2x2e2", "n4e6p6" , 28, 10, 20, 0.8, PostType.Regular)

Calculator.getPrice (package).

3. Expected Result: "14.98"

4. This test checks if the answer of the postal rate for non-standard packages (length in the range of 24.5cm up to 38cm) up to 1 kg in regular type from one fixed place to a fixed destination is right.

The screenshot shows the Microsoft Visual Studio interface with the following details:

- Solution Explorer:** Shows the solution structure for "Ecsel428_AssignmentB" with projects "Ecsel428_AssignmentB" and "UnitTestProject".
- Test Explorer:** Displays test results for "CalcTests.cs".
 - Failed Tests (12):**
 - testPostalCapital (1 ms)
 - testWidth0 (50 ms)
 - testWidth10 (50 ms)
 - testWidth11 (1 ms)
 - testWidth12 (1 ms)
 - testWidth13 (1 ms)
 - testWidth14 (1 ms)
 - testWidth15 (1 ms)
 - testWidth16 (1 ms)
 - testWidth17 (1 ms)
 - testWidth18 (1 ms)
 - testWidth19 (1 ms)
 - Passed Tests (13):**
 - testLengthHighBound (< 1 ms)
 - testLengthLowBound (< 1 ms)
 - testLengthHighBound (< 1 ms)
 - testLengthLowBound (< 1 ms)
 - testNullFrom (< 1 ms)
 - testNullTo (< 1 ms)
 - testNullFromOrTo (< 1 ms)
 - testNullFromOrTo (< 1 ms)
 - testNullToOrFrom (< 1 ms)
 - testPostalCodeFormatIs (< 1 ms)
 - testWeightHighBound (< 1 ms)
 - testWeightLowBound (< 1 ms)
 - testWidthHighBound (< 1 ms)
 - testWidthLowBound (< 1 ms)
- Output:** Shows the command-line output of the tests, including log entries and error messages. One test failed: "Test Failed - testValid9".

The screenshot shows the Microsoft Visual Studio interface with the following details:

- Test Explorer:** Shows 3 Failed Tests (testValid9, testLengthHighbound, testWidthHighbound) and 22 Passed Tests.
- Solution Explorer:** Displays the project structure for "Ece428.AssignmentB" including files like App.config, Calc.cs, DataSheet.cs, Program.cs, and UnitTest1.cs.
- Code Editor:** Shows the code for the Calc class, specifically the getPrice method which calculates postal rates based on package dimensions and weight.
- Output Window:** Shows the command-line output of the test run, including log messages from the test runner and the results of the failed test.

```

public class Calc
{
    static string PostalHead = "abcghijklmnpqrstuvwxyz";
    static List<char> postalheadlist = PostalHead.ToString().ToList();
    public static string getPrice(package package)
    {
        if (validate(package) != null)
        {
            return validate(package);
        }

        float sum = DataSheet.getPrice(package.length, package.width, package.weight);
        int distance = postalheadlist.IndexOf(char.ToLower(package.from[0]));
        distance = Math.Abs(postalheadlist.IndexOf(char.ToLower(package.to[0])) - distance);
        sum += distance * 9;

        return sum.ToString();
    }

    private static string validate(package package)
    {
        //data validation
        if (package == null)
        {
            return "invalid package";
        }
        if (string.IsNullOrEmpty(package.from))
        {
            return "invalid from";
        }

        if (package.from.length != 6 || !(char.IsLetter(package.from[0]) && char.IsLetter(package.from[2]) && char.IsLetter(package.from[4])))
        {
            return "invalid from";
        }
    }
}

```

```

testValid9
Source: UnitTest1.cs line 63
Actual: 16.4
Expected: <16.4>
Failed. Expected:<16.4>.
Actual:<16.4>.
Elapsed time: 00:00:00.0247113
Stack Trace:

```

Test18

1. Test name: valid_Rule6

2. Call Setup: create a new package from valid package.

Let package=new package ("h2x2e2", "n4e6p6", 28, 10, 20, 2, PostType.Regular)

Calculator.getPrice (package).

3. Expected Result: "16.4"

4. This test checks if the answer of the postal rate for non-standard packages (width in the range of 24.5cm up to 38cm) over 1 kg up to 5 kg in regular type from one fixed place to a fixed

destination is right.

The screenshot shows the Microsoft Visual Studio interface with the following details:

- Solution Explorer:** Shows the solution "Ece428_AssignmentB" with projects "Ece428_AssignmentB" and "UnitTestProject".
- Test Explorer:** Displays 12 failed tests and 13 passed tests. Failed tests include: testPostalCapital, testValid1, testValid10, testValid11, testValid12, testValid13, testValid14, testValid15, testValid16, testValid17, testValid18, testValid19, and testWidthLowBound. Passed tests include: testHeightHighBound, testHeightLowBound, testLengthHighBound, testLengthLowBound, testNullFrom, testNullPreference, testNullPreference, testNullTo, testNullToEmailFormat, testTotalCodeFormat5, testTotalCodeFormat10, testWeightHighBound, testWeightLowBound, testWidthHighBound, and testWidthLowBound.
- Code Editor:** The file "DataSheet.cs" is open, showing the implementation of the "Calc" class and its static method "getPrice".
- Output Window:** Shows the command-line output of the unit tests, indicating failures for the postal capital and width low bound tests.

The screenshot shows the Microsoft Visual Studio interface with the following details:

- Solution Explorer:** Shows the solution structure for "Ecsel428_AssignmentB" with projects like "Ecsel428_AssignmentB" and "UnitTest1".
- Test Explorer:** Displays a list of tests. Failed tests include:
 - testValid10: 1 ms
 - testValid11: 1 ms
 - testValid9: 24 msPassed tests include:
 - testLengthLbound
 - testLengthUbound
 - testWidthLbound
 - testWidthUbound
 - testPostalCodeFormatFrom
 - testPostalCodeFormatFrom9
 - testValid1
 - testValid2
 - testValid3
 - testValid4
 - testValid5
 - testValid6
 - testValid7
 - testValid8
 - testWeightLbound
 - testWeightUbound
 - testWidthLbound
 - testWidthUbound
- Code Editor:** The "Calc.cs" file is open, showing the implementation of the "Calc" class. The failing test "testValid10" is highlighted.
- Output Window:** Shows the results of the test run, indicating a failure for "testValid10".
- Status Bar:** Shows the elapsed time as "Elapsed time: 00:00:00.024713".

Test19

1. Test name: valid_Rule7
 2. Call Setup: create a new package from valid package.

```
Let package=new package ("h2x2e2", "n4e6p6", 15, 19, 20, 0.8, PostType.Regular )
```

Calculator.getPrice (package).

3. Expected Result: "14.98"

4. This test checks if the answer of the postal rate for non-standard packages (width in the range of 15.6cm up to 27cm) up to 1 kg in regular type from one fixed place to a fixed destination is right.

The screenshot shows the Microsoft Visual Studio interface with the following details:

- Solution Explorer:** Shows the solution Ecs428_AssignmentB with two projects: Ecs428_AssignmentB and UnitTestProject.
- Code Editor:** Displays the `Calc.cs` file containing the `getPrice` method and the `validate` helper method.
- Test Explorer:** Shows 12 failed tests for `testPostalCapital`, `testPostal0`, `testValid0`, `testValid1`, `testValid2`, `testValid3`, `testValid4`, `testValid5`, `testValid6`, `testValid7`, `testValid8`, and `testValid9`.
- Output Window:** Shows the command-line output of the tests, including timestamps and log messages.

```

    public static string getPrice(package package)
    {
        if (validate(package) != null)
        {
            return validate(package);
        }

        float sum = DataSheet.getPrice(package.length, package.width, package.weight);

        return sum.ToString();
    }

    private static string validate(package package)
    {
        //data validation
        if (package == null)
        {
            return "invalid package";
        }

        if (string.IsNullOrEmpty(package.from))
        {
            return "invalid from";
        }

        if (package.from.Length != 6 || !(char.IsLetter(package.from[0]) & char.IsLetter(package.from[2]) & char.IsLetter(package.from[4])))
        {
            return "invalid from";
        }
    }

```

5. With implementing code for calculating non-standard packages.

The screenshot shows the Microsoft Visual Studio interface with the following details:

- Solution Explorer:** Shows the solution Ecs428_AssignmentB with two projects: Ecs428_AssignmentB and UnitTestProject.
- Code Editor:** Displays the `Calc.cs` file with additional logic for non-standard packages.
- Test Explorer:** Shows 3 failed tests for `testValid10`, `testValid11`, and `testValid9`.
- Output Window:** Shows the command-line output of the tests, including timestamps and log messages.

```

    static string PostalHead = "abcdefghijklmnopqrstuvwxyz";
    static List<char> postalheadlist = PostalHead.ToString().ToList();
    public static string getPrice(package package)
    {
        if (validate(package) != null)
        {
            return validate(package);
        }

        float sum = DataSheet.getPrice(package.length, package.width, package.weight);
        int distance = postalheadlist.IndexOf(char.ToLower(package.from[0]));
        distance = Math.Abs(postalheadlist.IndexOf(char.ToLower(package.to[0])) - distance);
        sum += distance * 9;

        return sum.ToString();
    }

    private static string validate(package package)
    {
        //data validation
        if (package == null)
        {
            return "invalid package";
        }

        if (string.IsNullOrEmpty(package.from))
        {
            return "invalid from";
        }

        if (package.from.Length != 6 || !(char.IsLetter(package.from[0]) & char.IsLetter(package.from[2]) & char.IsLetter(package.from[4])))
        {
            return "invalid from";
        }
    }

```

Test20

1. Test name: valid_Rule8

2. Call Setup: create a new package from valid package.

```
Let package=new package ("h2x2e2", "n4e6p6", 15, 19, 20, 2, PostType.Regular )
```

Calculator.getPrice (package).

3. Expected Result: "16.4"

4. This test checks if the answer of the postal rate for non-standard packages (width in the range of 15.6cm up to 27cm) over 1 kg up to 5 kg in regular type from one fixed place to a fixed destination is right.

The screenshot shows the Microsoft Visual Studio interface during a unit test execution. The title bar indicates the project is 'Excel42 Assignment'.

Solution Explorer: Shows the project structure with files like 'Assignment.cs', 'Program.cs', and 'Properties.cs'.

Test Explorer: Displays the results of a test run. It lists 10 successful tests under 'Passed Tests' and 1 failed test under 'Failed Tests'. The failed test is 'testShouldCalculatePrice'.

Output Window: Shows the command-line output of the test run, including the test results and some internal logs.

Code Editor: The main window displays the 'Assignment.cs' file. The 'testShouldCalculatePrice' method is highlighted in red, indicating it failed. The code implements a static method 'getprice' that calculates the price based on length, width, height, and weight, and a private static method 'validate' that checks if the package is valid.

5. With implementing the code

The screenshot shows the Microsoft Visual Studio interface with the following details:

- Test Explorer:** Displays test results for "Ece428.AssignmentB". It lists 3 Failed Tests (testValid10, testValid11, testValid9) and 22 Passed Tests (e.g., testLengthHighBound, testHeightLowBound). The failed tests are highlighted in red.
- Solution Explorer:** Shows the project structure for "Ece428.AssignmentB" with files like Calc.cs, DataSheet.cs, Program.cs, and UnitTest.cs.
- Code Editor:** The main editor window displays the code for "Calc.cs". The method "getPrice" is shown, which calculates a postal rate based on package dimensions and weight.
- Output Window:** Shows the command-line output of the test run, indicating successful discovery and execution of tests.

```

public class Calc
{
    static string PostalHead = "abcgehjkklmnpqrstuvwxyz";
    static List<char> postalheadlist = PostalHead.ToString().ToList();
    public static string getPrice(package package)
    {
        if (validate(package) != null)
        {
            return validate(package);
        }

        float sum = DataSheet.getPrice(package.length, package.width, package.weight);
        int distance = postalheadlist.IndexOf(char.ToLower(package.from[0]));
        distance = Math.Abs(postalheadlist.IndexOf(char.ToLower(package.to[0])) - distance);
        sum += distance * 9;

        return sum.ToString();
    }

    private static string validate(package package)
    {
        //data validation
        if (package == null)
        {
            return "invalid package";
        }
        if (string.IsNullOrEmpty(package.from))
        {
            return "invalid from";
        }

        if (package.from.length != 6 || !(char.IsLetter(package.from[0]) && char.IsLetter(package.from[2]) && char.IsLetter(package.from[4])))
        {
            return "invalid from";
        }
    }
}

```

Test21

1. Test name: valid_Rule9

2. Call Setup: create a new package from valid package.

Let package=new package ("h2x2e2", "n4e6p6", 15, 19, 20, 2, PostType.Xpress)

Calculator.getPrice (package).

3. Expected Result: "24.6"

4. This test checks if the answer of the postal rate for non-standard packages (width in the range of 15.6cm up to 27cm) over 1 kg up to 5 kg in Xpress type from one fixed place to a fixed destination is right.

The screenshot shows a Microsoft Visual Studio interface with the following details:

- MenuBar:** File, Edit, View, Project, Build, Debug, Tools, Test, Analyze, Window, Help.
- Toolbars:** Standard, Status Bar.
- Toolbox:** Standard.
- Code Editor:** Displays C# code for a class named `Calc`. The code includes static methods for calculating postage based on package dimensions and weight. A specific test case for `testValidate9` is highlighted in red, indicating a failure. The error message in the status bar states: "Test Failed: Invalid package length, failed Expected<4>, Actual<1>".
- Solution Explorer:** Shows the project structure for "Excel02 Assignment" with files like App.config, Properties.cs, and UnitTest1.cs.
- Properties:** Shows build configurations (Debug, Release).
- Task List:** Lists the failing test: "testValidate9" at line 102.
- Output Window:** Shows the error message: "Test Failed: Invalid package length, failed Expected<4>, Actual<1>".
- Help:** Microsoft Visual Studio Help.

5. With code to calculate the Xpress type

The screenshot shows the Microsoft Visual Studio interface with the following details:

- File Explorer:** Shows the project structure for "Ece428_AssignmentB".
- Solution Explorer:** Shows the solution and project files.
- Task List:** Displays a list of tasks and their descriptions.
- Code Editor:** The main window displays the source code for "Calc.cs". The code implements a class "Calc" with a static method "getPrice" that calculates the price of a package based on its dimensions and weight. It includes validation logic for package parameters.
- Output Window:** Shows the results of the unit tests, indicating 25 passed tests and one failed test ("testWeightHighBound").

Test22

1. Test name: valid_Rule10
 2. Call Setup: create a new package from valid package.
Let package=new package ("h2x2e2", "n4e6p6" , 15, 19)
Calculator.getPrice (package).
 3. Expected Result: "32.8"

4. This test checks if the answer of the postal rate for non-standard packages (width in the range of 15.6cm up to 27cm) over 1 kg up to 5 kg in Priority type from one fixed place to a fixed destination is right.

5. With implement the code to calculate the Priority type.

The screenshot shows the Microsoft Visual Studio interface with the following details:

- Solution Explorer:** Shows the solution "Ecs428_AssignmentB" with its projects and files.
- Test Explorer:** Displays 25 passed unit tests, including validation tests for height, weight, and width.
- Code Editor:** The main window displays the code for the class `Ecs428_AssignmentB.Calc`. The code implements a `getPrice` method that calculates a price based on package dimensions and type. It includes validation logic for `from` and `to` addresses.
- Status Bar:** Shows the status "Ready" and the file name "Ecs428_AssignmentB.Calc".

Test23

1. Test name: valid Rule11

2. Call Setup: create a new package from valid package.

Let package=new package ("H2L2C9", "H2X1J5", 15, 19, 20, 2, PostType.Priority)

Calculator.getPrice (package).

3. Expected Result: "22.8"

4. This test checks if the answer of the postal rate for non-standard packages (width in the range of 15.6cm up to 27cm) over 1 kg up to 5 kg in Priority type with different "from" and "to" postcodes is right.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Ecsel428.AssignmentB
{
    public class Calc
    {
        static string PostalHead = "abceghijklmnpqrstuvwxyz";
        static List<char> postalheadlist = PostalHead.ToString().ToList();
        public static string getPrice(package package)
        {
            if (validate(package) != null)
            {
                return validate(package);
            }
            float sum = DataSheet.getPrice(package.length, package.width, package.weight);
            int distance = postalheadlist.IndexOf(char.ToLower(package.from[0]));
            distance = Math.Abs(postalheadlist.IndexOf(char.ToLower(package.to[0])) - distance);
            sum += distance * 9;
            return sum.ToString();
        }
        private static string validate(package package)
        {
            //data validation
            if (package == null)
            {
                return "invalid package";
            }
            if (string.IsNullOrEmpty(package.from))
            {
                return "invalid from";
            }
        }
    }
}

```

Test Runs

Source: UnitTest.cs line 63

testValid9

Message: Assert.AreEqual failed. Expected:
Actual: 16.4
Elapsed time: 00:00:00.023932

StackTrace:

5. With implement the code that calculate the distance between two postcode.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Ecsel428.AssignmentB
{
    public class Calc
    {
        static string PostalHead = "abceghijklmnpqrstuvwxyz";
        static List<char> postalheadlist = PostalHead.ToString().ToList();
        public static string getPrice(package package)
        {
            if (validate(package) != null)
            {
                return validate(package);
            }
            float sum = DataSheet.getPrice(package.length, package.width, package.weight);
            int distance = postalheadlist.IndexOf(char.ToLower(package.from[0]));
            distance = Math.Abs(postalheadlist.IndexOf(char.ToLower(package.to[0])) - distance);
            sum += (int)(package.type / 2.0);
            return sum.ToString();
        }
        private static string validate(package package)
        {
            //data validation
            if (package == null)
            {
                return "invalid package";
            }
            if (string.IsNullOrEmpty(package.from))
            {
                return "invalid from";
            }
        }
    }
}

```

Test Runs

Source: UnitTest.cs line 63

testValid9

Message: Assert.AreEqual failed. Expected:
Actual: 16.4
Elapsed time: 00:00:00.000541

StackTrace: