**CST8130 – Data Structures**

**Test Plan < Lab #1>**

**Prepared By - <Sean Bradbury>**

<add more tables if you have more features to test. Add more rows if you have more test cases> Cover all possible test cases with sufficient detail.

**Feature: addValue()**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test #** | **Description** | **Input** | **Expected output/result** | **Actual Output** | **Status (Pass/Fail)** |
| T1 | Tests 1 value | 3 | "Numbers are: \n3.0\n" | "Numbers are: \n3.0\n" | Pass |
| T2 | Tests a second value | 2.5 | "Numbers are: \n3.0\n2.5\n" | "Numbers are: \n3.0\n2.5\n" | Pass |
| T3 | Test third value | -1.2 | "Numbers are: \n3.0\n2.5\n-1.2\n" | "Numbers are: \n3.0\n2.5\n-1.2\n" | Pass |
| T4 | Tests 4th value | 4.65767 | Numbers are: \n3.0\n2.5\n-1.2\n4.7\n | Numbers are: \n3.0\n2.5\n-1.2\n4.7\n | Pass, toString() method rounds value |
| T5 | Tests 5th value | 0 | Numbers are: \n3.0\n2.5\n-1.2\n4.7\n0.0\n | Numbers are: \n3.0\n2.5\n-1.2\n4.7\n0.0\n | Pass |
| T6 | Tests 6th value, too large for size | 2 | Array full  Numbers are: \n3.0\n2.5\n-1.2\n4.7\n0.0\n | Array full  Numbers are: \n3.0\n2.5\n-1.2\n4.7\n0.0\n | Pass |

**Feature: calcAverage()**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test #** | **Description** | **Input** | **Expected output/result** | **Actual Output** | **Status (Pass/Fail)** |
| T1 | Test empty array | Empty Float array of size 5 | 0.0 | 0.0 | Pass |
| T2 | Test array of small decimal numbers | [2.2, 1.1, 3.3, 4] | 2.65 | 2.65 | Pass |
| T3 | Test array of one element | [7] | 7.0 | 7.0 | Pass |
| T4 | Test array with negative numbers, 0, and decimals | [-5.6453, 1000, 0, 2, 6.59999] | 200.570938 | 200.570938 | Pass, decimal number is within EPSILON |

**Feature: findMinMax() - minimum**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test #** | **Description** | **Input** | **Expected output/result** | **Actual Output** | **Status (Pass/Fail)** |
| T1 | Test empty array | Empty Float array of size 5 | 0.0 | 0.0 | Pass |
| T2 | Test array of small decimal numbers | [2.2, 1.1, 3.3, 4] | 1.1 | 1.1 | Pass |
| T3 | Test array of one element | [7] | 7.0 | 7.0 | Pass |
| T4 | Test array with negative numbers, 0, and decimals | [-5.6453, 1000, 0, 2, 6.59999] | -5.6453 | -5.6453 | Pass, decimal number is within EPSILON |

**Feature: findMinMax() - maximum**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test #** | **Description** | **Input** | **Expected output/result** | **Actual Output** | **Status (Pass/Fail)** |
| T1 | Test empty array | Empty Float array of size 5 | 0.0 | 0.0 | Pass |
| T2 | Test array of small decimal numbers | [2.2, 1.1, 3.3, 4] | 4.0 | 4.0 | Pass |
| T3 | Test array of one element | [7] | 7.0 | 7.0 | Pass |
| T4 | Test array with negative numbers, 0, and decimals | [-5.6453, 1000, 0, 2, 6.59999] | 1000.0 | 1000.0 | Pass |

**Feature: findMinMax() – max mod min**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test #** | **Description** | **Input** | **Expected output/result** | **Actual Output** | **Status (Pass/Fail)** |
| T1 | Test empty array | Empty Float array of size 5 | NaN | NaN | Pass, min is 0 so divide by 0 |
| T2 | Test array of small decimal numbers | [2.2, 1.1, 3.3, 4] | 0.7 | 0.7 | Pass, decimal number is within EPSILON |
| T3 | Test array of one element | [7] | 0.0 | 0.0 | Pass |
| T4 | Test array with negative numbers, 0, and decimals | [-5.6453, 1000, 0, 2, 6.59999] | -5.64533.402823 | 3.402823 | Pass, decimal number is within EPSILON |

**Feature: getFactorialMax()**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test #** | **Description** | **Input** | **Expected output/result** | **Actual Output** | **Status (Pass/Fail)** |
| T1 | Test empty array | Empty Float array of size 5 | 1 | 1 | Pass, factorial of 0! is 1 |
| T2 | Test array of small decimal numbers | [2.2, 1.1, 3.3, 4] | 24 | 24 | Pass, 4! = 24 |
| T3 | Test array of one element | [7] | 5040 | 5040 | Pass, 7! = 5040 |
| T4 | Test array with negative numbers, 0, and decimals | [-5.6453, 1000, 0, 2, 6.59999] | ~4024E+2567 | 0 | Fail, number is too large for int type. 1000! ~ 4024E+2567 |

**Feature: addValues()**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test #** | **Description** | **Input** | **Expected output/result** | **Actual Output** | **Status (Pass/Fail)** |
| T1 | Standard case | 3  2.2  1.1  3.3  4 | "Numbers are: \n2.2\n1.1\n3.3\n4\n" | "Numbers are: \n2.2\n1.1\n3.3\n4\n" | Pass |
| T2 | Not enough room in array | 3  2.2  1.1  3.3  4 | "Numbers are: \n2.2\n1.1\n3.3\n4\n" | "Numbers are: \n2.2\n1.1\n3.3\n4\n" | Pass, nothing is added since array is full |
| T3 | Add on to end of array | 3  2.2  1.1  3.3  4 | "Numbers are: \n-5.6\n1000.0\n0.0\n2.0\  n6.5\n2.2\n1.1\n3.3\n4.0\n" | "Numbers are: \n-5.6\n1000.0\n0.0\n2.0\  n6.5\n2.2\n1.1\n3.3\n4.0\n" | Pass |

**Feature: readValues()**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test #** | **Description** | **Input** | **Expected output/result** | **Actual Output** | **Status (Pass/Fail)** |
| T1 | File exists | testSaveValue.txt | "Numbers are: \n3.0\n2.0\n1.0\n" | "Numbers are: \n3.0\n2.0\n1.0\n" | Pass |
| T2 | File does not exist | “thisisnotafile.oops” | fileRead == false | fileRead == false | Pass |
|  |  |  |  |  |  |

**Feature: saveValues()**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test #** | **Description** | **Input** | **Expected output/result** | **Actual Output** | **Status (Pass/Fail)** |
| T1 | File is created | testSaveValues.txt | file.isFile() == true | file.isFile() == true | Pass |
| T2 | File has correct values | numItems = 4;  numbers = {2.2,1.1,3.3,4.0} | "Numbers are: \n2.2\n1.1\n3.3\n4.0\n" | "Numbers are: \n2.2\n1.1\n3.3\n4.0\n"0 | Pass |