# Testing Document

Name: Blake Peck  
Campus: Melbourne

# Introductory Exercises

## Create

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Expected Output** | **Actual Output** | **Passed?** |
| LinkedList<int>\* List = new LinkedList<int>;  Assert::IsNotNull(List); | For the Test to check that List is not null. | The test comes back with the List not being null. | Passed. |

## Add

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Expected Output** | **Actual Output** | **Passed?** |
| LinkedList<int> List;  List.InsertStart(5);  List.InsertStart(2);  List.InsertStart(7);  List.InsertStart(10);  Assert::AreEqual(List.GetNode(0)->m\_Data, 10);  Assert::AreEqual(List.GetNode(1)->m\_Data, 7);  Assert::AreEqual(List.GetNode(2)->m\_Data, 2);  Assert::AreEqual(List.GetNode(3)->m\_Data, 5); | That the List.GetNode(i)->m\_Data will be equal to the value next to it.  0 = 10, 1 = 7, 2 = 2, 3 = 5. | The test come back positive with the test suggesting the Nodes are equal to the value I put into the List. | Passed. |

## CountSize

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Expected Output** | **Actual Output** | **Passed?** |
| LinkedList<int> List;  List.InsertStart(5);  List.InsertStart(2);  List.InsertStart(7);  List.InsertStart(10);  List.InsertStart(3);  List.InsertStart(14);  Assert::AreEqual(List.GetCount(), 6); | The GetCount function will be equal to the size of the List. Count = 6. | GetCount and 6 equals together will the test working. | Passed. |

## Remove

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Expected Output** | **Actual Output** | **Passed?** |
| LinkedList<int> List;  List.InsertEnd(5);  List.InsertEnd(2);  List.InsertEnd(7);  List.InsertEnd(16);  List.InsertEnd(10);  List.InsertEnd(3);  List.InsertEnd(14);  List.RemoveStart();  List.RemoveEnd();  List.Remove(16);  Assert::AreEqual(List.GetFirstNode()->m\_Data, 2);  Assert::AreEqual(List.GetLastNode()->m\_Data, 3);  Assert::AreEqual(List.GetNode(2)->m\_Data, 10); | The List will equal the test after the functions that remove the start, end, and chosen values.  FirstNode = 2.  LastNode = 3.  Node(2) = 10. | The Result come back good with the FirstNode equalled to 2, LastNode equalled to 3 and the node 2 that had a value deleted now equals 10. | Passed. |

## Sort

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Expected Output** | **Actual Output** | **Passed?** |
| LinkedList<int> List;  List.InsertEnd(25);  List.InsertEnd(14);  List.InsertEnd(7);  List.InsertEnd(29);  List.Sort();  Assert::AreEqual(List.GetNode(0)->m\_Data, 7);  Assert::AreEqual(List.GetNode(1)->m\_Data, 14);  Assert::AreEqual(List.GetNode(2)->m\_Data, 25);  Assert::AreEqual(List.GetNode(3)->m\_Data, 29); | The List will rearrange the data value in ascending order 7, 14, 25, 19. | The Test Passed with the order being 7, 14, 25, 19 after staring in random position to a ascending order | Passed. |

## Empty

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Expected Output** | **Actual Output** | **Passed?** |
| LinkedList<int> List;  Assert::IsTrue(List.Empty()); | The List function Empty will come back true when there is no data in the List. | The Test said it passed meaning the function functioned. | Passed. |