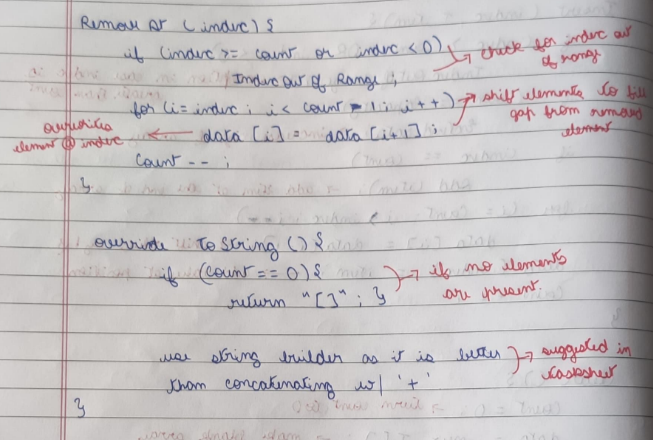
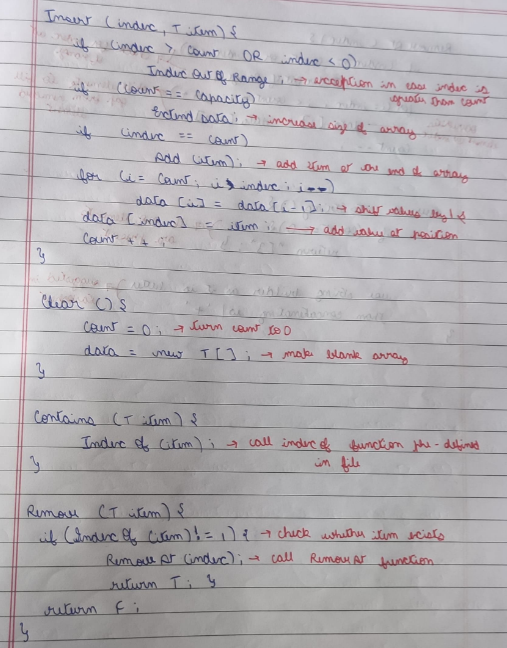
|  |  |
| --- | --- |
|  |  |
| SIT221 |  |
|  |  |
|  | Submitted By:Arnav Sharma, 2410994769 |
|  | Task – 1.1P |

## Solution of Task

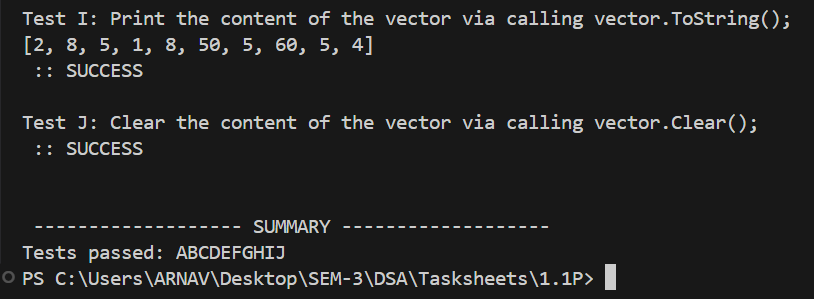
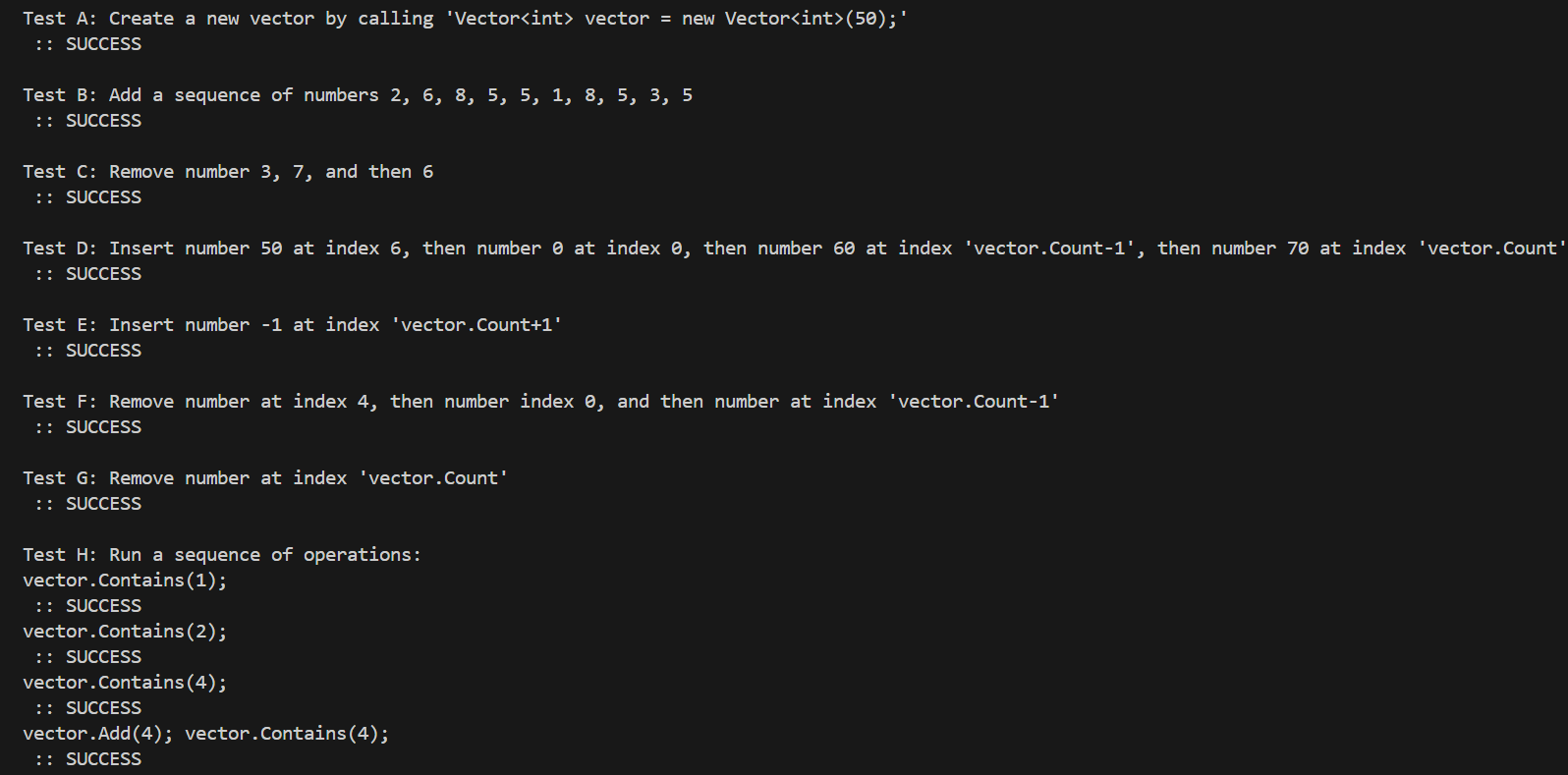
The attached code has comments which explain each line’s functionality, and after the code, there are images attached which showcase the pseudo-code that I had written prior to writing the code.

1. using System;
2. using System.Collections.Generic;
3. using System.Text;
4. namespace Vector
5. {
6. public class Vector<T>
7. {
8. // This constant determines the default number of elements in a newly created vector.
9. // It is also used to extended the capacity of the existing vector
10. private const int DEFAULT\_CAPACITY = 10;
11. // This array represents the internal data structure wrapped by the vector class.
12. // In fact, all the elements are to be stored in this private  array.
13. // You will just write extra functionality (methods) to make the work with the array more convenient for the user.
14. private T[] data;
15. // This property represents the number of elements in the vector
16. public int Count { get; private set; } = 0;
17. // This property represents the maximum number of elements (capacity) in the vector
18. public int Capacity { get; private set; } = 0;
19. // This is an overloaded constructor
20. public Vector(int capacity)
21. {
22. data = new T[capacity];
23. Capacity = capacity;
24. }
25. // This is the implementation of the default constructor
26. public Vector() : this(DEFAULT\_CAPACITY) { }
27. // An Indexer is a special type of property that allows a class or structure to be accessed the same way as array for its internal collection.
28. // For example, introducing the following indexer you may address an element of the vector as vector[i] or vector[0] or ...
29. public T this[int index]
30. {
31. get
32. {
33. if (index >= Count || index < 0) throw new IndexOutOfRangeException();
34. return data[index];
35. }
36. set
37. {
38. if (index >= Count || index < 0) throw new IndexOutOfRangeException();
39. data[index] = value;
40. }
41. }
42. // This private method allows extension of the existing capacity of the vector by another 'extraCapacity' elements.
43. // The new capacity is equal to the existing one plus 'extraCapacity'.
44. // It copies the elements of 'data' (the existing array) to 'newData' (the new array), and then makes data pointing to 'newData'.
45. private void ExtendData(int extraCapacity)
46. {
47. T[] newData = new T[data.Length + extraCapacity];
48. for (int i = 0; i < Count; i++) newData[i] = data[i];
49. data = newData;
50. Capacity = data.Length;
51. }
52. // This method adds a new element to the existing array.
53. // If the internal array is out of capacity, its capacity is first extended to fit the new element.
54. public void Add(T element)
55. {
56. if (Count == data.Length) ExtendData(DEFAULT\_CAPACITY);
57. data[Count++] = element;
58. }
59. // This method searches for the specified object and returns the zero‐based index of the first occurrence within the entire data structure.
60. // This method performs a linear search; therefore, this method is an O(n) runtime complexity operation.
61. // If occurrence is not found, then the method returns –1.
62. // Note that Equals is the proper method to compare two objects for equality, you must not use operator '=' for this purpose.
63. public int IndexOf(T element)
64. {
65. for (var i = 0; i < Count; i++)
66. {
67. if (data[i].Equals(element)) return i;
68. }
69. return -1;
70. }
71. // TODO:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
72. // TODO: Your task is to implement all the remaining methods.
73. // Read the instruction carefully, study the code examples from above as they should help you to write the rest of the code.
74. public void Insert(int index, T item)
75. {
76. //To check whether the index could be present in array or not
77. if (index > Count || index < 0)
78. {
79. throw new IndexOutOfRangeException();
80. }
81. //To check if the array is full
82. if (Count == Capacity)
83. {
84. ExtendData(DEFAULT\_CAPACITY);
85. }
86. //To check if the array still has 1 more value left ONLY
87. if (index == Count)
88. {
89. Add(item);
90. return;
91. }
92. //Shifting the elements to the left to allow addition of the element at index
93. for (int i = Count; i > index; i--)
94. {
95. data[i] = data[i - 1];
96. }
97. //Add value at index and increase count
98. data[index] = item;
99. Count++;
100. }
101. public void Clear()
102. {
103. //Set count to 0
104. Count = 0;
105. //Create a new array to replace the old one
106. data = new T[Capacity];
107. }
108. public bool Contains(T item)
109. {
110. //Calling the pre-existing function
111. return IndexOf(item) != -1;
112. }
113. public bool Remove(T item)
114. {
115. //Figure out the index of the item to reduce the number of functions created
116. int index = IndexOf(item);
117. //Check whether item exists
118. if (index != -1)
119. {
120. //Call RemoveAt to, again, reduce the number of functions
121. RemoveAt(index);
122. return true;
123. }
124. //In case the value doesn't exist
125. return false;
126. }
127. public void RemoveAt(int index)
128. {
129. //Check for exception
130. if (index >= Count || index < 0)
131. {
132. throw new IndexOutOfRangeException();
133. }
134. //Use a loop to overwrite the value at index, and set the other values over it
135. for (int i = index; i < Count - 1; i++)
136. {
137. data[i] = data[i + 1];
138. }
139. //Decrement count as 1 value was reduced
140. Count--;
141. }
142. public override string ToString()
143. {
144. //Check whether array has no elements in it
145. if (Count == 0)
146. {
147. return "[]";
148. }
149. //Using StringBuilder as said in tasksheet
150. StringBuilder sb = new StringBuilder();
151. sb.Append("[");
152. //Run a loop to go through each element one by one
153. for (int i = 0; i < Count; i++)
154. {
155. //Add the values
156. sb.Append(data[i]);
157. if (i < Count - 1)
158. {
159. sb.Append(", ");
160. }
161. }
162. sb.Append("]");
163. //Return final value
164. return sb.ToString();
165. }
166. }
167. }

Pseudo-code:



Output of Tester file:



GitHub link:

<https://github.com/ArnavSharma2007/SIT221/tree/main/1.1P>