Practical Task 1.1

(Pass Task)

Submission deadline: 11:59 pm Sunday, July 26
Discussion deadline: 11:59 pm Sunday, August 16

Task Objective

In this task you will use the programming skills acquired during prior units (namely SIT232 Object-Oriented Development) to complete an already partially implemented generic data structure called a Vector<T>.

Background

A Vector is a simple data structure used to store any number of elements in a single dimension. Essentially, Vectors are like arrays except with a few differences. Firstly, they are not a built-in language construct and instead need to be directly implemented. Secondly, unlike array's fixed size, a Vector can be resized during run time. The Vector class can perform several basic operations such as accessing, recording, and deleting elements from the data collection.

In this task, you are going to complete an implementation of a Vector class. The class being created will use a generic implementation. This means that it will be parameterised over types, which allows the Vector to store elements with a variety of types rather than a single one. Students that have completed SIT232 Object-Oriented Development should be familiar with Generics.

As revision a generic class is created using a generic type parameter list within angle brackets (< >). For example, the following code defines a class that stores a single element of any type:

```
public class MyGenericClass<T>
{
        private T data;
        public MyGenericClass(T d)
        {
                data = d;
        }
        public T MyData()
        {
                return data;
        }
}
```

As can be seen, this class is defined as using the generic type T. This allows the client code to create an instance of the class to use the type of the client's choosing without any risk of a runtime casting exception. Now wherever the type parameter T is used can be regarded as the type to be identified by the client. For example. If my client code declares the class using an int then they would use the following code.

```
public class Tester
{
        static void Main(string[] args)
        {
                MyGenericClass<int> myclass = new MyGenericClass<int>(44);
                Console.WriteLine("Contains: " + myclass.MyData());
        }
}
```

For more information on Generics see the Week 1 lecture in SIT221 Data Structures and Algorithms.

Task Details

The Vector we will be implementing is similar to the collection class List<T> offered within the Microsoft .NET Framework[1]. If you have trouble understanding what the Vector class should be doing then review the List<T> class (https://msdn.microsoft.com/en-us/library/6sh2ey19(v=vs.110).aspx). The following steps indicate what you need to do:

1. Download the two C# source code files attached to this task. These files serve as a template for the program that you need to develop. Create a new Microsoft Visual Studio project and import the files. Your newly built project should compile and work without errors although it will initially fail the test cases provided.

   The given template of the Vector<T> class should help you with development of its remaining methods. Therefore, explore the existing code as other methods are to be very similar in terms of implementation. For more information see Appendix A of this task.

2. You must complete the Vector<T> implementation by providing the following functionality:

   - **void Insert(int index, T item)**

     Inserts a new element into the data structure at the specified index. This will involve four parts (Note a part may be more than a single line of code):

     - If Count already equals Capacity (eg the currently allocated space is full), then the capacity of the Vector<T> is increased by reallocating the internal array, and the existing elements are copied to the new larger array ready for the new element to be added in the following steps.
     - If index is equal to Count, item add the item to the end of the Vector<T>.
     - If index is somewhere internal to the vector then starting at the end, shuffle each element along one until you reach the index location indicated and place the item in the vector at that point.
     - If index is outside the range of the current Vector then this method should throw an IndexOutOfRangeException.

   - **void Clear()**

     Remove all elements from the Vector<T> and set the Count back to 0.

   - **bool Contains(T item)**

     Determines whether a specified item is in the data collection. It returns true when the item is presented, and false otherwise.

     Hint: you can use the IndexOf method already provided to determine if the item is already in the data collection.

   - **bool Contains(T item)**

     Determines whether a specified item is in the data collection. It returns true when the item is presented, and false otherwise.

   - **bool Remove(T item)**

     Removes the first occurrence of the specified item from the data collection. It returns true if the item is successfully removed. This method returns false if item was not found in the Vector<T>.

     Hint 1: to find the element you can use the IndexOf method already provided.

     Hint 2: to remove the element you can use the RemoveAt method you will write in the next dot point.

   - **void RemoveAt(int index)**

     Removes the element at a specified index of the Vector<T>. When you call RemoveAt to remove an item, the remaining items in the list are renumbered to replace the removed item.

[1] https://msdn.microsoft.com/en-us/library/6sh2ey19(v=vs.110).aspx

For example, if you remove the item at index 3, the item at index 4 is moved to the 3rd position. In addition, the number of items in the data collection (as represented by the Count property) is reduced by 1. This method throws IndexOutOfRangeException when the index's value is invalid.

- **string ToString()**

  ToString() is a is the major formatting method in the .NET Framework. The Object class, which is the base class for all C# classes provides a default implementation of this method. However, this simply generates a fully qualified type name and does not provide information about the Vectors contents. You need to override this by providing your own implementation of the ToString() method.

  Your method will return a string that represents the current Vector's contents. Your implementation should construct a string that starts and ends with square brackets ([ ]). Between these bracket you should provide a comma separated list of each of the elements in your Vector. This will require you to step through your Vector element-by-element appending each to your string. After each element, if there are more elements add a comma (,) before you add the next element.

  Hint: StringBuilder is a C# class that will help you construct a String from your vector. See the following link for details: https://docs.microsoft.com/en-us/dotnet/api/system.text.stringbuilder?view=netcore-3.1

Assumption: for simplicity, you may assume that the value of Capacity can only be increased.

Hint: As you progress through the implementation of the Vector<T> class, you should start using the Tester class to thoroughly test the Vector<T> aiming on the coverage of all potential logical issues and runtime errors. This (testing) part of the task is as important as writing the vector class. The given version of the testing class covers only some basic cases. Therefore, you should extend it with extra cases to make sure that your vector class is checked against other potential mistakes.

Expected Printout

Appendix B provides an example printout for you program. If you are getting a different printout then your implementation is not read for submission. Please ensure your program prints out Success for all tests before submission.

Further Information

1. Read Chapter 1 of SIT221 Workbook available in CloudDeakin in Resources → Course materials → SIT221 Workbook. It will give you general understanding of the task and issues related to the use of arrays in practice.
2. If you still struggle with such OOP concepts as Generics and their application, you may wish to read Chapter 11 of SIT232 Workbook available in Resources → Course materials → Materials Object-Oriented Programming. You may also have to read Chapter 6 of SIT232 Workbook about Polymorphism and Interfaces as you need excellent understanding of these topics to progress well through the practical tasks of the unit. Make sure that you are proficient with them as they form a basis to design and develop programming modules in this and all the subsequent tasks. You may find other important topics required to complete the task, like exceptions handling, in other chapters of the workbook.
3. We will test your code in Microsoft Visual Studio 2017. Find and install the community version of Microsoft Visual Studio 2017 available on the SIT221 unit web-page in CloudDeakin at Resources → Course materials → Visual Studio Community 2017 Installation Package. You are free to use another code editor if you prefer that, e.g. Visual Studio Code. But we recommend you to take a chance to learn this environment.

Marking Process and Discussion

To get your task completed, you must finish the following steps strictly on time.

1. Work on your task either during your allocated lab time or during your own study time.
2. Once the task is complete you should make sure that your program implements all the required functionality, is compliable, and has no runtime errors. Programs causing compilation or runtime errors will not be accepted as a solution. You need to test your program thoroughly before submission. Think about potential errors where your program might fail. Note we can sometime use test cases that are different to those provided so verify you have checked it more thoroughly than just using the test program provided.
3. Submit your solution as an answer to the task via the OnTrack submission system. This first submission must be prior to the submission "S" deadline indicated in the unit guide and in OnTrack.
4. If your task has been assessed as requiring a "Redo" or "Resubmit" then you should prepare a new submission. You will have 1 (7 day) calendar week from the day you receive the assessment from the tutor. This usually will mean you should revise the lecture, the readings indicated, and read the unit discussion list for suggestions. After your submission has been corrected and providing it is still before the due deadline you can resubmit.
5. If your task has been assessed as correct, either after step 3 or 4, you can "discuss" with your tutor. This first discussion must occur prior to the discussion "D".
6. Meet with your tutor or answer question via the intelligent discussion facility to demonstrate/discuss your submission. Be on time with respect to the specified discussion deadline.
7. The tutor will ask you both theoretical and practical questions. Questions are likely to cover lecture notes, so attending (or watching) lectures should help you with this compulsory interview part. The tutor will tick off the task as complete, only if you provide a satisfactory answer to these questions.
8. If you cannot answer the questions satisfactorily your task will remain on discussion and you will need to study the topic during the week and have a second discussion the following week.
9. Please note, due to the number of students and time constraints tutors will only be expected to mark and/or discuss your task twice. After this it will be marked as a "Exceeded Feedback".
10. If your task has been marked as "Exceeded Feedback" you are eligible to do the redemption quiz for this task. Go to this unit's site on Deakin Sync and find the redemption quiz associated with this task. You get three tries at this quiz. Ensure you record your attempt.
    I. Login to Zoom and join a meeting by yourself.
    II. Ensure you have both a camera and microphone working
    III. Start a recording.
    IV. Select Share screen then select "Screen". This will share your whole desktop. Ensure Zoom is including your camera view of you in the corner.
    V. Bring your browser up and do the quiz.
    VI. Once finished select stop recording.
    VII. After five to ten minutes you should get an email from Zoom providing you with a link to your video. Using the share link, copy this and paste in your chat for this task in OnTrack for your tutor to verify the recording.
11. Note that we will not check your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces the chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work through the unit.
12. Final note, A "Fail" or "Exceeded Feedback" grade on a task does not mean you have failed the unit. It simply means that you have not demonstrated your understanding of that task through OnTrack. Similarly failing the redemption quiz also does not mean you have failed the unit. You can replace a task with a task from a higher grade.

Appendix A: Files Provided

Two files are proved and detailed below.

Tester.cs

This files already contains the *Main* method and is the starting point for your program. It also contains a series of tests that will verify if your Vector class works correctly. When first running your program these tests will fail. Once you have completed the task all these tests will report success. See Appendix B for an example of what the program will output when working correctly.

Vector.cs

This file contains a template of the Vector<T>, which you will be completing for your task. Below is a list of the methods already implemented for you:

Vector()

Constructor. Initializes a new instance of the Vector<T> class that is empty and has a default initial capacity, say 10 elements.

Vector(int capacity)

Constructor. Initializes a new instance of the Vector<T> class that is empty and has a specified initial capacity. As an input parameter, this constructor accepts a number of elements that the data structure can initially store. If the size of the collection can be estimated beforehand, specifying the initial capacity eliminates the need to perform a number of resizing operations while adding new elements to it.

Count

Property. Gets the number of elements physically contained in the Vector<T>.

Capacity

Property. Gets the total number of elements that the internal array can potentially hold without resizing. Capacity represents the number of elements that the Vector<T> can store before resizing is required, whereas Count is the number of elements that are actually in the Vector<T>. Capacity is always greater than or equal to Count. If Count exceeds Capacity while adding elements, the capacity is increased by automatically reallocating the internal array to one of a larger size before adding the new elements.

void Add(T item)

Adds a new item of type T to the end of the Vector<T>. If Count already equals Capacity, the capacity of the Vector<T> is increased by automatically reallocating the internal array, and the existing elements are copied to the new larger array before the new element is added.

int IndexOf(T item)

Searches for the specified item and returns the zero-based index of the first occurrence within the entire data structure. It returns the zero-based index of the item, if found; otherwise, –1.

T this[int index]

Returns the element at a specified index of the sequence. As an argument, it accepts a zero-based index of the element to retrieve. This method throws IndexOutOfRangeException when the index's value is invalid.

Appendix B: Expected Output

The following provides an example of the output generated from the Tester function once you have correctly implemented all the Vector<T> class correctly

```
Test A: Create a new vector by calling 'Vector<int> vector = new Vector<int>(50);'
 :: SUCCESS

Test B: Add a sequence of numbers 2, 6, 8, 5, 5, 1, 8, 5, 3, 5
 :: SUCCESS

Test C: Remove number 3, 7, and then 6
 :: SUCCESS

Test D: Insert number 50 at index 6, then number 0 at index 0, then number 60 at
index 'vector.Count-1', then number 70 at index 'vector.Count'
 :: SUCCESS

Test E: Insert number -1 at index 'vector.Count+1'
 :: SUCCESS

Test F: Remove number at index 4, then number index 0, and then number at index
'vector.Count-1'
 :: SUCCESS

Test G: Remove number at index 'vector.Count'
 :: SUCCESS

Test H: Run a sequence of operations:
vector.Contains(1);
 :: SUCCESS
vector.Contains(2);
 :: SUCCESS
vector.Contains(4);
 :: SUCCESS
vector.Add(4); vector.Contains(4);
 :: SUCCESS

Test I: Print the content of the vector via calling vector.ToString();
[2,8,5,1,8,50,5,60,5,4]
 :: SUCCESS

Test J: Clear the content of the vector via calling vector.Clear();
 :: SUCCESS

 ------------------- SUMMARY -------------------
Tests passed: ABCDEFGHIJ
```