SIT221 – Data Structures and Algorithms

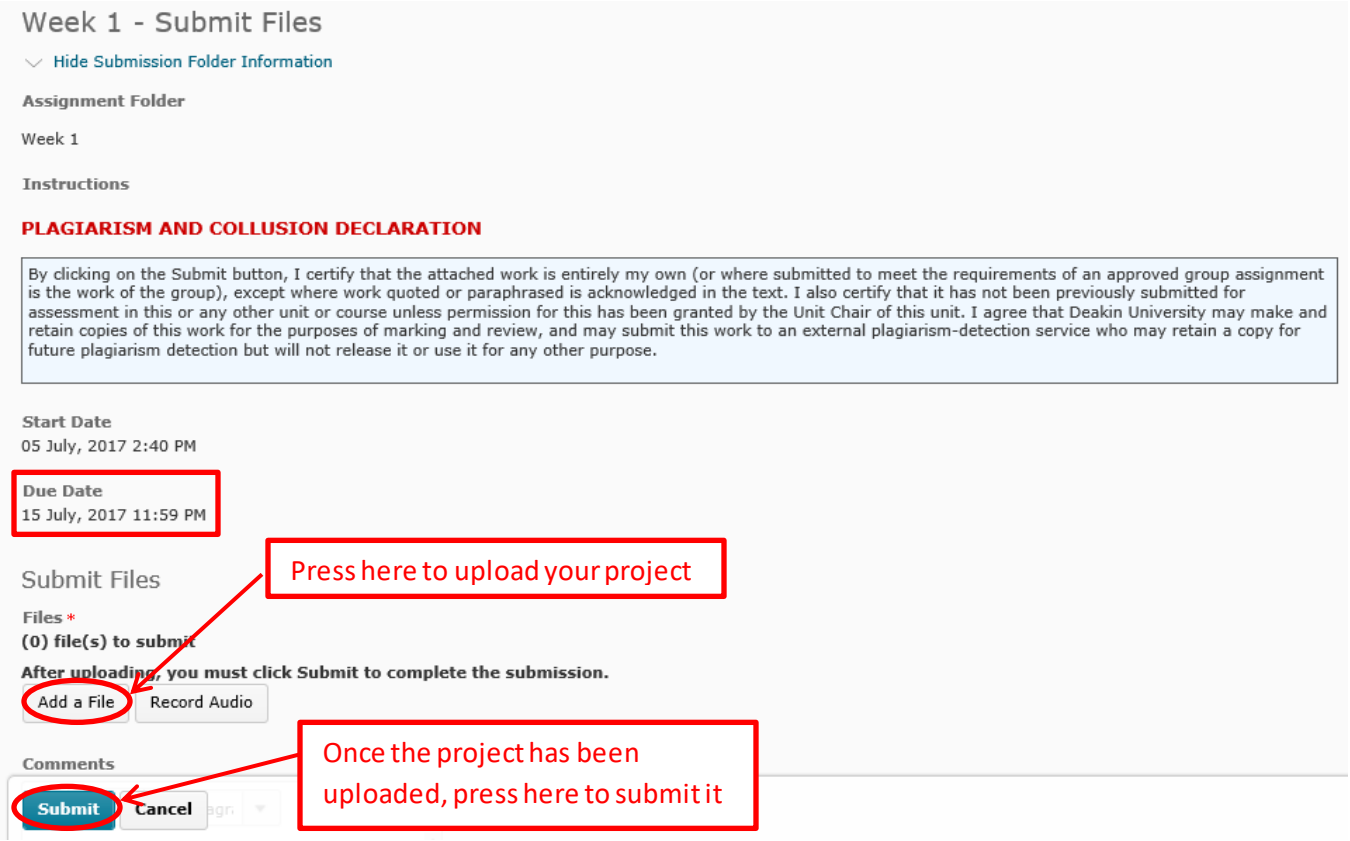
Lab 1: Get started

Objectives

1. Review key object-oriented programming concepts
2. Practice complexity analysis
3. Review basic file I/O operations

Submission instructions

1. Submit **Task 1 & 4** via **CloudDeakin** by **11:59pm 15 July 2017**. The figure below shows the submission folder.
2. You should zip your code into **one file only** and submit this file.



Preparation

1. Make sure that Microsoft Visual Studio 2015/2017 is installed on your machines. Community version can be found at: https://www.visualstudio.com/en-us/downloads/download-visual-studiovs.aspx

2. If you use Mac, you can still download Visual Studio Code from the URL above, or use Xamarin Studio.
3. Download the resources from the weekly Prac folder. There are two projects: **DataStructures_Algorithms** & **Runner**.
4. In the Runner project, there is a Data folder for Week01 – Note that the dataset has three files (1H.txt, 1T.txt, 1M.txt) [1H = 100 points, 1T = 1000 points, 10T = 10,000 points]

Lab tasks

1. **Vector class**

   In this class, we need to implement a new generic Vector class that can maintain any number of data elements. The class should provide the following capabilities,

   a. **Vector()**: This is a class constructor that initializes your Vector capacity to a default value – Assume the default capacity is 10.
   b. **Vector(int capacity)**: This is a class constructor that takes as a parameter the initial number of elements (capacity) that we can maintain in Vector object.
   c. **Capacity**: A property that gets/sets the maximum number of elements we can store in a Vector object.
   d. **Count**: A property that gets (read-only) number of current elements stored in a Vector object.
   e. **Add(T element)**: This method should add an item at the end of the list – e.g. if you have three items in your Vector object, it will add the new item in the fourth position. If no more space for a new element (max capacity reached), then you need to extend your array capacity.
   f. **Insert(T element, int index)**: This method should insert an item at a specified index – This means that you will need to move the other items on the right hand side in your Vector by one element.
   g. **Clear()**: This method should remove all the items in the Vector object.
   h. **IndexOf(T element)**: This method should return the index of a given element in the list maintained by a Vector object.
   i. **Contains(T element)**: This method should return true if the element exists in the Vector list.
   j. **Remove(T element)**: This method should delete the first occurrence of an element from a Vector object.
   k. **RemoveAll(T element)**: This method should delete all occurrences of an element from a Vector object.
   l. **RemoveAt(int index)**: This method should delete an element at a given index, and adjust array elements accordingly.

m. **Max()**: This method should return the maximum data point in the Vector object.
n. **Min()**: This method should return the minimum data point in the Vector list.
o. **ToString()**: This method should return a string that contains all array elements.
p. **This[int index]**: This indexer should get/set a Vector element at a given index – Please make sure that the index value is within range (0 to current number of elements).

Vector		
-	capacity	int
-	count	int
-	elements	T
+	Vector()	
+	Vecotr(int capacity)	
+	Capacity {get;set;}	int
+	Count {get;}	int
+	Add(T element)	
+	Insert(T element, int index)	
+	Contains(T element)	bool
+	Clear()	
+	IndexOf(T element)	int
+	Remove(T element)	bool
+	RemoveAll(T element)	bool
+	RemoveAt(int index)	bool
+	ToString()	string
+	Max()	T
+	Min()	T

2. Complexity analysis

In this task, we want to practice algorithm complexity analysis of the methods we have implemented:

a. What is the best and worst case running time (i.e. the number of operators involved in) for **IndexOf(…)**, **Max()**, **Min()** methods?
b. How the run time changes between the three datasets 1H.txt, 1T.txt, and 10T.txt?

3. Disk I/O

In folder **DataStructures_Algorithms/Utils** there is a Data Serializer class that we use to load data from text file, save data to text file, serialize and deserialize your Vector class. Review the class, and make sure you understand it – We will revise this class in the following weeks.

4. **Extension of task 1**

   In this task we want to extend the **Runner01** class to introduce a new **CleanData** method. This method should receive a vector of type int and remove all data points less than LowerBound or greater than UpperBound.