Q.1 Use generic Dictionary collection class to do following:

1. Add a new contact
2. Search for contact based on contactid and contactname
3. List all contacts
4. Update a specific contact

Hint:

* Define Contact entity class with Id, Name, City and CellNo properties
* Populate generic dictionary collection with contact id as key and contact instance as value

Q.2 Redo the assignment above with generic List collection. Difference there will not be any key

Q.3 Create a class ListOfOrders which encapsulate List<> collection. It will have function to add CustomerOrder entity class instances . Create 2 subscribers which will be notified whenever new order is added, the first subscriber should print no of orders till now received and the second one should print total order. CustomerOrder class should have orderid, orderdate, custname, orderamount, disapatchcity properties

Hint: Use Multicast Delegate Concept

Q.4 Add Search functions to the above ListOfOrders class in assignment 8 to search for order(s) based on orderid, custname , dispatchcity and return them. Use LINQ to do this.

Q.5 Create a CustOrder class with OrderId, CustId, OrderAmt,DispatchCity properties.

Accept multiple customer order details using console till end user wants to continue in a generic collection.

Once details are accepted print the report as to how many customer orders and total order value per dispatch city

DispatchCity No Of Order Total Order Value

============= ============ ===============

Mumbai 999 999

Pune 999 999

…...

**TASK PARALLEL LIBRARY / async & await**

1. Create a function named **GetSomeData()** which returns a **Task<string>.** The task must take 5 seconds to create and return some string.
2. Invoke the function from the **Main()** function and display the string returned from the task.
3. Create 2 functions named **Total()** & **Average()** respectively. The **Total()** function takes 3 numbers as arguments and computes the total. The **Average()** function takes 2 arguments and computes the average based on the total.

Create a task which invokes the **Total()** method. The task must return the total.

From the **Main()** method invoke this task and when the results are available, invoke another task which calls the **Average()** method passing the total obtained from the previous task. *(use the concept of continuation tasks for this example)*

1. Create the following methods where each method returns a Task<T>:

Task<string> GetVideo() 🡪 task returns a string after 10 seconds

Task<string> AddIntro(string video) 🡪 task returns a string after 20 seconds, takes the

string returned from previous task

Task<string> AddSummary(string video) 🡪 task returns a string after 30 seconds, takes the

string returned from previous task

Invoke the **GetVideo()** method from the **Main()** method and display the final video from it. *(use*

*async and await pattern for this example)*