24 LINQ Assignment DOTNET

1. Given an array of numbers. Find the cube of the numbers that are greater than 100 but less than 1000 using LINQ.

private static void CubeFinder()

{

Console.WriteLine("Int Array element : ");

int[] intarry = new int[] { 5, 6, 7, 9, 10, 11, 12 };

foreach (var item in intarry)

{

Console.Write($"{item } ");

}

Console.WriteLine("");

var filter = from i in intarry

let cube = (i \* i \* i)

where cube > 100 && cube < 1000

select new { ArrayItem = i, cube };

Console.WriteLine("Find the cube of the numbers that are greater than 100 but less than 1000 using LINQ");

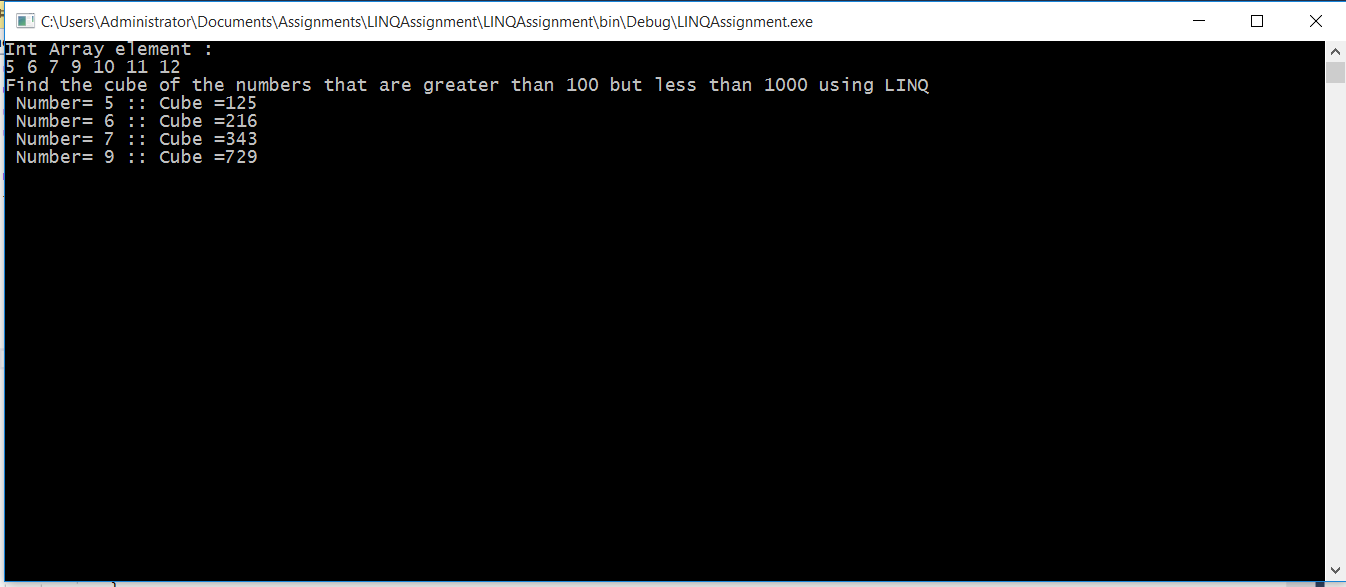
foreach (var item in filter)

{

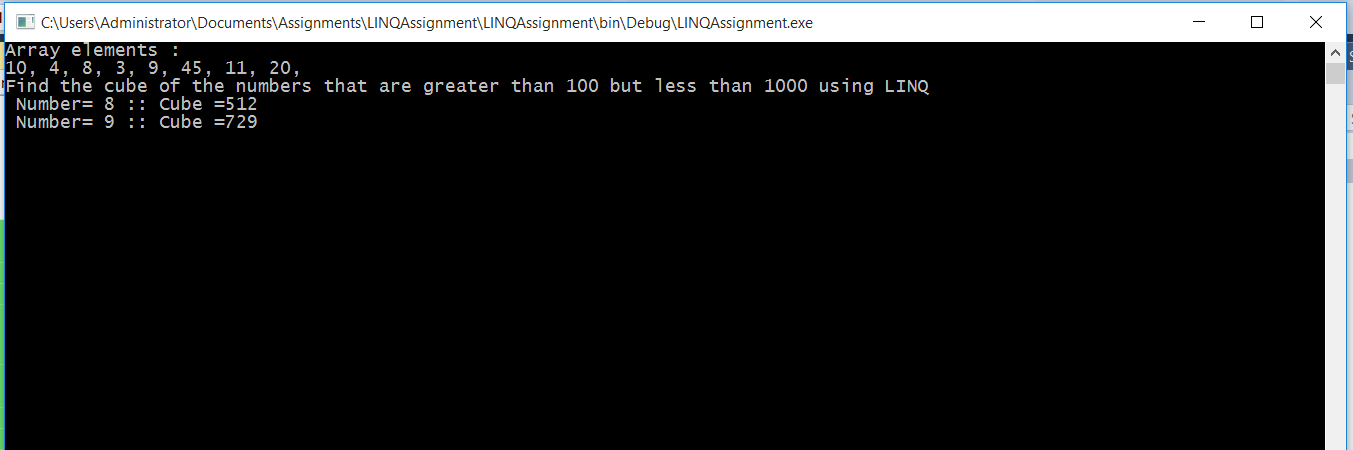
Console.WriteLine($" Number= {item.ArrayItem} :: Cube ={item.cube}");

}

}



After changing array elements.



1. Display the data day wise from most recently ordered to least recently ordered and by quantity from highest to lowest.

private static void DisplayOrdersDatenQty()

{

List<Order> orderList = new List<Order> {

new Order(){ Id=1, ItemName="Item1", OrderDate= Convert.ToDateTime("02/01/2019"), Quantity=3 },

new Order(){ Id=2, ItemName="Item5", OrderDate= Convert.ToDateTime("02/05/2019"), Quantity=10 },

new Order(){ Id=3, ItemName="Item8", OrderDate= Convert.ToDateTime("02/01/2019"), Quantity=5 },

new Order(){ Id=4, ItemName="Item3", OrderDate= Convert.ToDateTime("02/03/2019"), Quantity=7 },

new Order(){ Id=5, ItemName="Item15", OrderDate= Convert.ToDateTime("02/09/2019"), Quantity=9 },

};

var orderbyQty = orderList

.OrderByDescending(x => x.OrderDate)

.ThenByDescending(x => x.Quantity);

foreach (var item in orderbyQty)

{

Console.WriteLine($" Item ID ={ item.Id} , " +

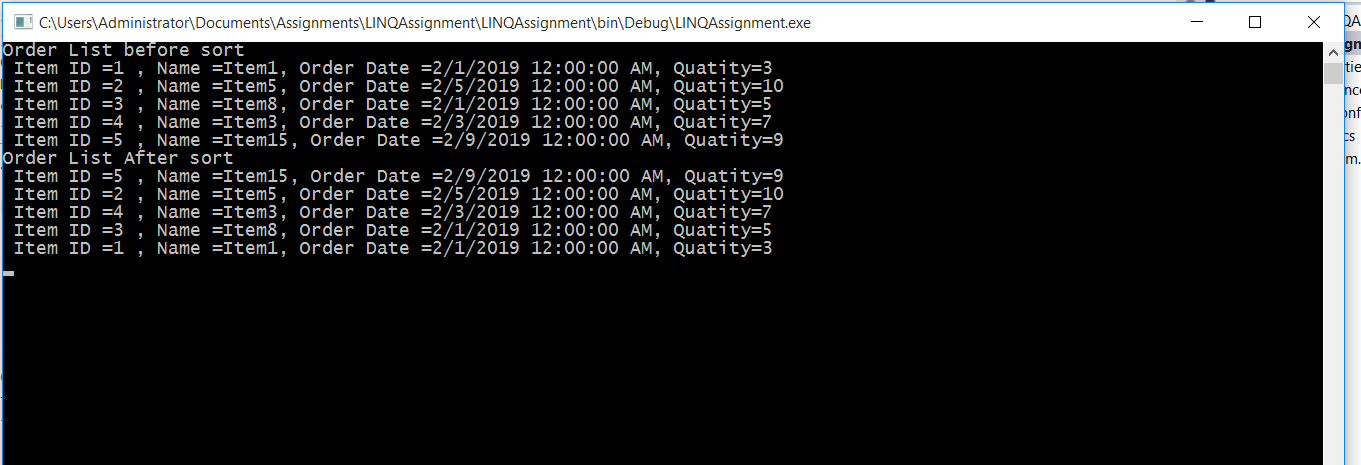
$"Name ={item.ItemName}, " +

$"Order Date ={item.OrderDate}, " +

$"Quatity={item.Quantity} ");

}

}



1. For the previous exercise, write a LINQ query that displays the details grouped by the month in the descending order of the order date.

private static void DisplayOrdersGrpByMonth()

{

List<Order> orderList = new List<Order> {

new Order(){ Id=1, ItemName="Item1", OrderDate= Convert.ToDateTime("02/01/2019"), Quantity=3 },

new Order(){ Id=2, ItemName="Item5", OrderDate= Convert.ToDateTime("05/05/2019"), Quantity=10 },

new Order(){ Id=3, ItemName="Item8", OrderDate= Convert.ToDateTime("02/01/2019"), Quantity=5 },

new Order(){ Id=4, ItemName="Item3", OrderDate= Convert.ToDateTime("04/03/2019"), Quantity=7 },

new Order(){ Id=5, ItemName="Item15", OrderDate= Convert.ToDateTime("02/09/2019"), Quantity=9 },

new Order(){ Id=6, ItemName="Item7", OrderDate= Convert.ToDateTime("04/01/2019"), Quantity=5 },

new Order(){ Id=7, ItemName="Item6", OrderDate= Convert.ToDateTime("02/03/2019"), Quantity=7 },

new Order(){ Id=8, ItemName="Item10", OrderDate= Convert.ToDateTime("05/09/2019"), Quantity=9 },

}; Console.WriteLine("Order List before Grouping and sorting");

foreach (var item in orderList)

{

Console.WriteLine($" Item ID ={ item.Id} , " +

$"Name ={item.ItemName}, " +

$"Order Date ={item.OrderDate}, " +

$"Quatity={item.Quantity} ");

}

var orderedList = (from i in orderList

group i by i.OrderDate.Month.ToString() into oList

select new { Month = oList.Key, oList }).ToList();

Console.WriteLine("");

Console.WriteLine("Order List After Grouping and sorting");

for (int i = 0; i < orderedList.Count(); i++)

{

Console.WriteLine($" Month ={ orderedList[i].Month}");

var orderByDate = orderedList[i].oList.OrderByDescending(x => x.OrderDate);

foreach (var item in orderedList[i].oList)

{

Console.WriteLine($" Item ID ={ item.Id} , " +

$"Name ={item.ItemName}, " +

$"Order Date ={item.OrderDate}, " +

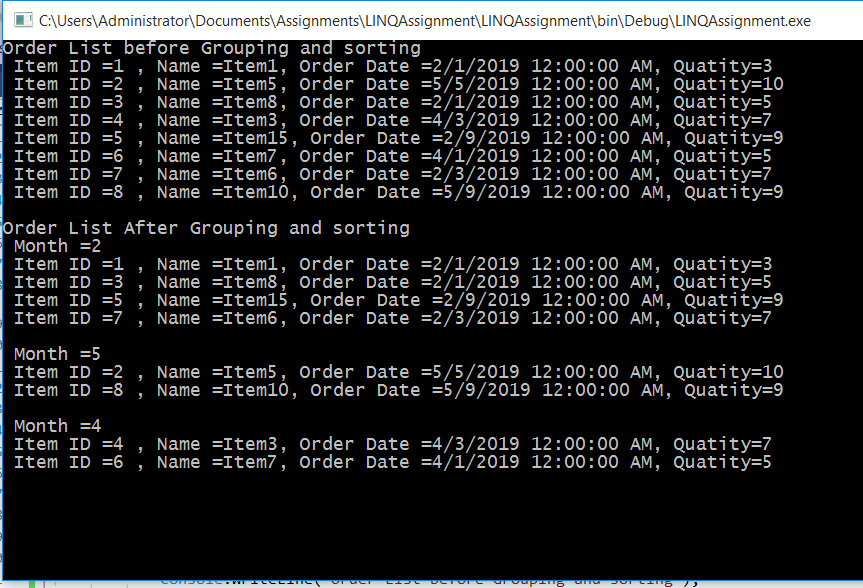
$"Quatity={item.Quantity} ");

}

Console.WriteLine("");

}

}



1. You have created Order class in the previous exercise and that has order id, item name, order date and quantity.

Create another class called Item that has item name and price.

Write a LINQ query such that it returns order id, item name, order date and the total price (price \* quantity ) grouped by the month in the descending order of the order date.

private static void DisplayOrdersPrice()

{

List<Order> orderList = new List<Order> {

new Order(){ Id=1, ItemName="Item1", OrderDate= Convert.ToDateTime("02/01/2019"), Quantity=3 },

new Order(){ Id=2, ItemName="Item5", OrderDate= Convert.ToDateTime("05/05/2019"), Quantity=10 },

new Order(){ Id=3, ItemName="Item3", OrderDate= Convert.ToDateTime("02/01/2019"), Quantity=5 },

new Order(){ Id=4, ItemName="Item3", OrderDate= Convert.ToDateTime("04/03/2019"), Quantity=7 },

new Order(){ Id=5, ItemName="Item1",OrderDate= Convert.ToDateTime("02/09/2019"), Quantity=9 },

new Order(){ Id=6, ItemName="Item1", OrderDate= Convert.ToDateTime("04/01/2019"), Quantity=5 },

new Order(){ Id=7, ItemName="Item5", OrderDate= Convert.ToDateTime("02/03/2019"), Quantity=7 },

new Order(){ Id=8, ItemName="Item3", OrderDate= Convert.ToDateTime("05/09/2019"), Quantity=9 },

};

List<Item> ItemList = new List<Item> {

new Item(){ ItemName="Item1", Price=10.5M },

new Item(){ ItemName="Item3", Price=30 },

new Item(){ ItemName="Item5", Price=50.10M},

};

Console.WriteLine("Item List");

foreach (var item in ItemList)

{

Console.WriteLine($" Item Name ={ item.ItemName} , " +

$"Price ={item.Price}, "

);

}

Console.WriteLine("");

Console.WriteLine("Order List before Grouping and sorting");

foreach (var item in orderList)

{

Console.WriteLine($" Item ID ={ item.Id} , " +

$"Name ={item.ItemName}, " +

$"Order Date ={item.OrderDate}, " +

$"Quatity={item.Quantity} ");

}

var orderedList = (from ol in orderList

group ol by ol.OrderDate.Month.ToString() into oList

select new { Month = oList.Key, oList }).ToList();

Console.WriteLine("");

Console.WriteLine("Order List After Grouping and sorting");

for (int i = 0; i < orderedList.Count(); i++)

{

Console.WriteLine($" Month ={ orderedList[i].Month}");

var orderByDate = orderedList[i].oList

.OrderByDescending(x => x.OrderDate)

.Join(ItemList, li => li.ItemName, itm => itm.ItemName,

(li, itm) => new

{

OrderId = li.Id,

ItemName = itm.ItemName,

OrderDate = li.OrderDate,

TotalPrice = li.Quantity \* itm.Price

});

;

foreach (var item in orderByDate)

{

Console.WriteLine($" Order ID ={ item.OrderId} , " +

$"Item Name ={item.ItemName}, " +

$"Order Date ={item.OrderDate}, " +

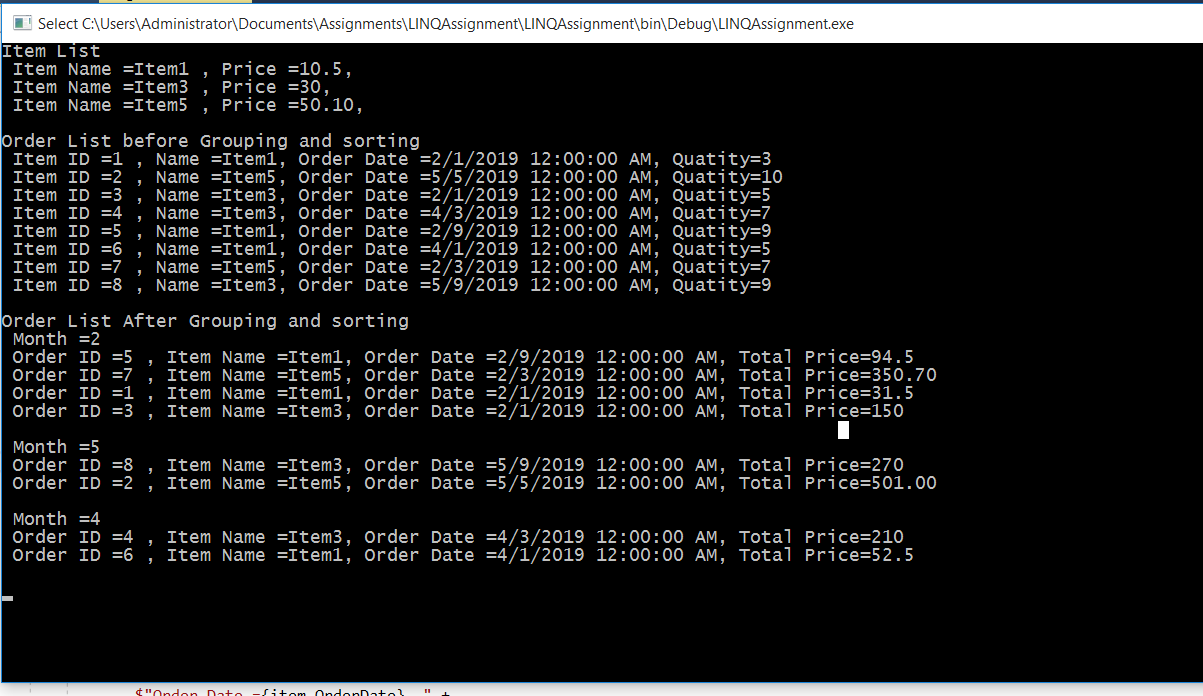
$"Total Price={item.TotalPrice} ");

}

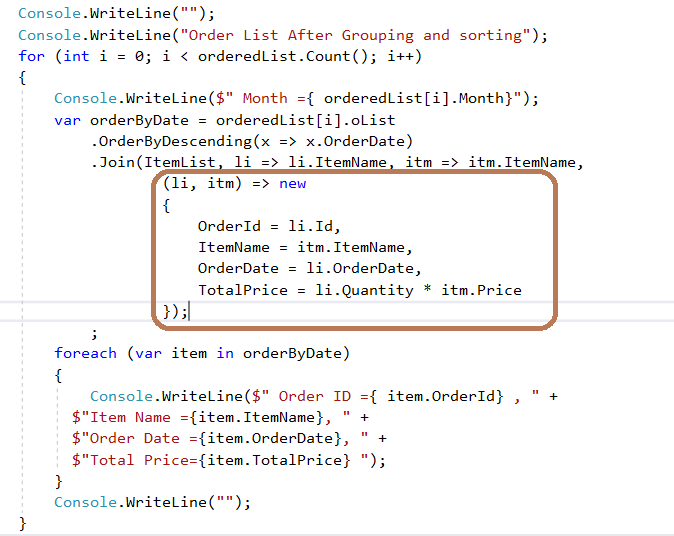
Console.WriteLine("");

}

}



1. using anonymous type



1. Check if all the quantities in the Order collection is >0.Get the name of the item that was ordered in largest quantity in a single order. (Hint: use LINQ methods to sort)

* Implemented IComparable interface.

public class Order : IComparable<Order>

{

//id, item name, order date and quantity

public int Id { get; set; }

public string ItemName { get; set; }

public DateTime OrderDate { get; set; }

public int Quantity { get; set; }

public int CompareTo(Order other)

{

return this.Quantity.CompareTo(other.Quantity);

}

}

private static void LargetsQty()

{

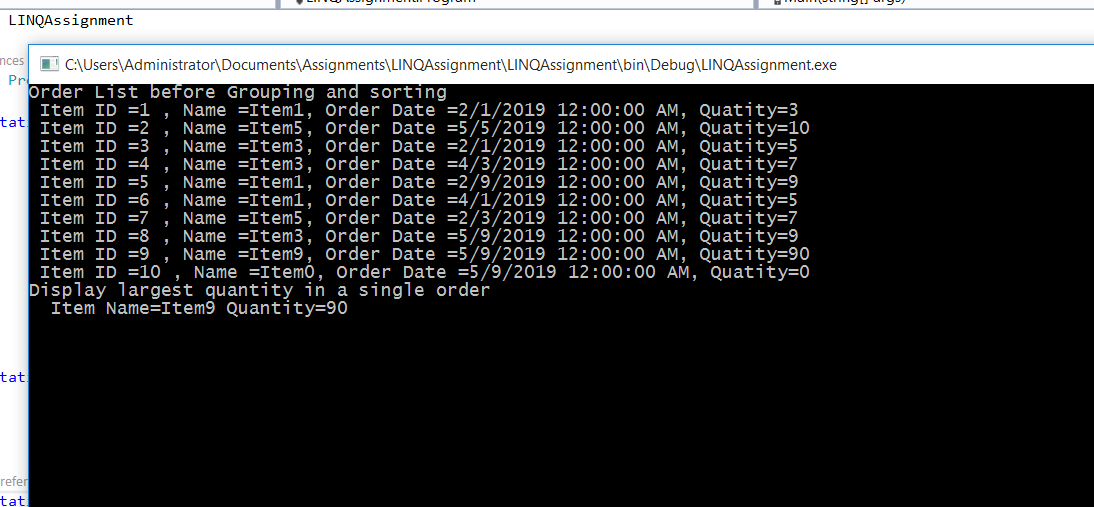
Console.WriteLine("Display largest quantity in a single order");

orderList.Reverse();

var largestQty = orderList.Where(x=>x.Quantity>0).FirstOrDefault();

Console.WriteLine( $" Item Name={largestQty.ItemName} Quantity={ largestQty.Quantity}");

}



1. Find if there are any orders placed before Jan this year

private static void OrderBeforeJan()

{

Console.WriteLine("Find if there are any orders placed before Jan of this year");

var filter = orderList.Where(x => x.OrderDate< Convert.ToDateTime("01/01/2019"));

Console.WriteLine("");

foreach (var item in filter)

{

Console.WriteLine($" Item ID ={ item.Id} , " +

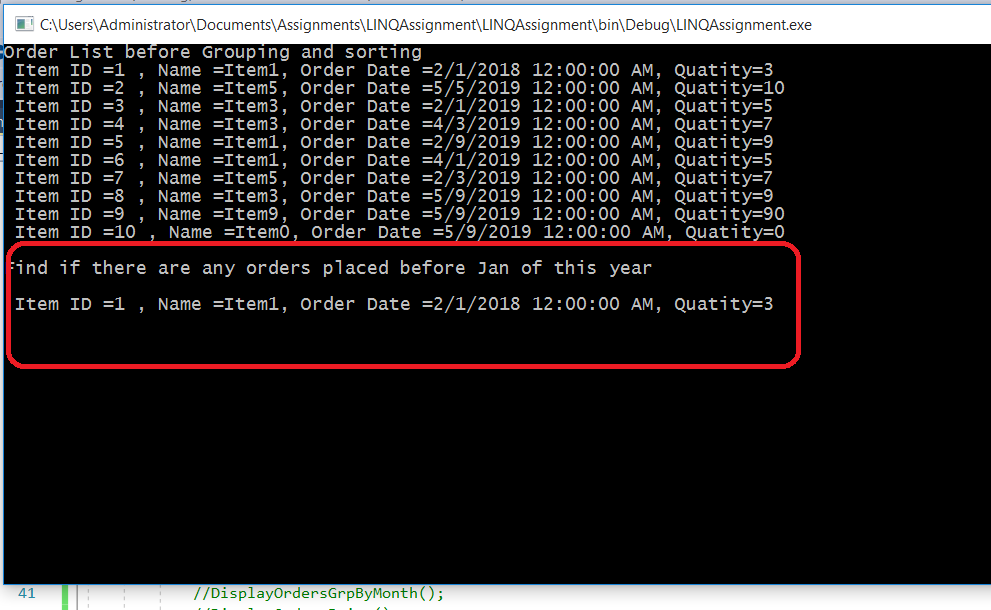
$"Name ={item.ItemName}, " +

$"Order Date ={item.OrderDate}, " +

$"Quatity={item.Quantity} ");

}

}



1. Rewrite the last example of that used Count using LINQ query methods entirely.

Sd

private static void OrderBeforeJan()

{

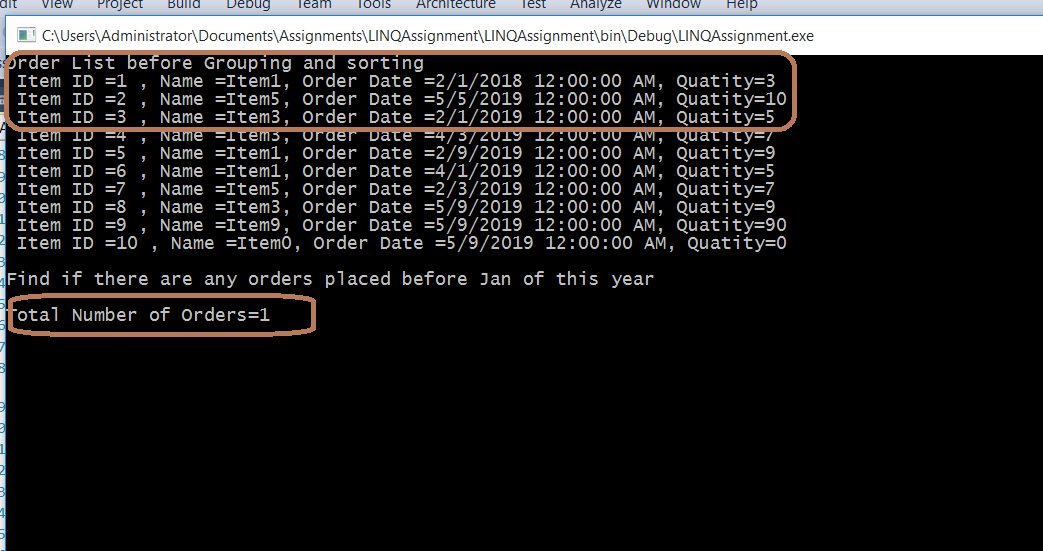
Console.WriteLine("Find if there are any orders placed before Jan of this year");

var filter = orderList.Where(x => x.OrderDate< Convert.ToDateTime("01/01/2019"));

Console.WriteLine("");

Console.WriteLine("Total Number of Orders="+ filter.Count());

}



1. Given the array of numbers. Count and display even numbers.

private static void DisplayEvenNumbers()

{

var filter = intArry.Where(x => x % 2 == 0).ToList();

Console.WriteLine($"Total Number of Even Number = { filter.Count()}");

foreach (var item in filter)

{

Console.Write($" {item},");

}

}

