Assignment 24

Linq Assignment

1.

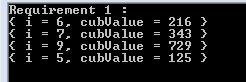
Given an array of numbers. Find the cube of the numbers that are greater than 100 but less than

1000 using LINQ.

Change some of the array elements and execute the same query again.

Hint : use the logical operators of C# to combine the conditions

Output:



Code Snippet

#region "Requirement1"

/\* Given an array of numbers. Find the cube of the numbers that are greater than 100

\* but less than 1000 using LINQ.Change some of the array elements and execute the

\* same query again.

\*

\* Hint : use the logical operators of C# to combine the conditions

\*/

int[] values = { 2, 4, 6, 7, 9, 1, 5 };

System.Console.WriteLine("Requirement 1 : ");

var CubeValues = from int i in values

let cubValue = i \* i \* i

where cubValue > 100 && cubValue < 1000

select new { i, cubValue };

foreach (var val in CubeValues)

Console.WriteLine(val);

#endregion

2.

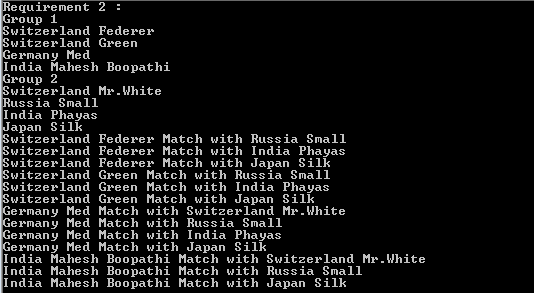
Given a list of participants for a tennis match. Split the list into 2 equal halves and display all the

possible combination of matches possible between the participants in the two lists. A condition is

that no player should have an opponent who is from h

is own his own country.

Output



Code Snippet

public static List<List<T>> Split<T>(IList<T> source)

{

return source

.Select((x, i) => new { Index = i, Value = x })

.GroupBy(x => x.Index % 2)

.Select(x => x.Select(v => v.Value).ToList()).ToList();

}

#region "Requirement2"

/\*

\* Given a list of participants for a tennis match.

\* Split the list into 2 equal halves

\* and display all the possible combination of matches

\* possible between the participants in the two lists.

\* A condition is that no player should have an opponent who is from his own his own country

\* \*/

var list = new[]

{

new Tennis( "Switzerland", "Federer" ),

new Tennis( "Switzerland", "Mr.White" ),

new Tennis( "Switzerland", "Green" ),

new Tennis( "Russia", "Small" ),

new Tennis( "Germany", "Med" ),

new Tennis( "India", "Phayas" ),

new Tennis( "India", "Mahesh Boopathi" ),

new Tennis( "Japan", "Silk" ),

};

List<List<Tennis>> splittedList = Split(list).ToList();

List<Tennis> List1 = splittedList.FirstOrDefault();

List<Tennis> List2 = splittedList.LastOrDefault();

var combination = from i in List1

from j in List2

where i.Country != j.Country

select new { i, j };

System.Console.WriteLine("Requirement 2 : ");

Console.WriteLine("Group 1 ");

foreach(var v in List1)

{

Console.WriteLine(v.Country + " "+v.Player);

}

Console.WriteLine("Group 2");

foreach (var v in List2)

{

Console.WriteLine(v.Country + " " + v.Player);

}

foreach (var v in combination)

{

Console.WriteLine(v.i.Country + " " + v.i.Player + " Match with " + v.j.Country + " " + v.j.Player);

}

#endregion "Requirement2"

3.

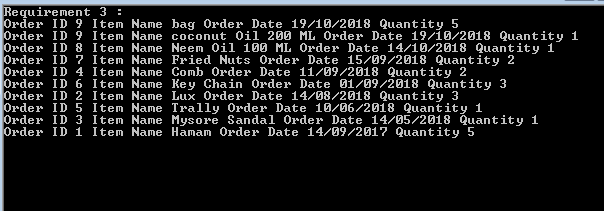
Create an Order class that has order id, item name, order date and quantity. Create a collection of

Order objects. Display the data day wise from most recently ordered to least recently ordered and

by quantity from highest to low

est.

Output:



Code Snippet:

#region "Requirement 3"

/\*Create an Order class that has order id, item name, order date and quantity.

\* Create a collection of Order objects.

\* Display the data day wise from most recently ordered to least recently ordered and

\* by quantity from highest to lowest

\* \*/

System.Console.WriteLine("Requirement 3 : ");

List<Order> orders = new List<Order>();

orders.Add(new Order(1, "Hamam", "2017-09-14 14:40:52,531", 5));

orders.Add(new Order(2, "Lux", "2018-08-14 10:40:52,531", 3));

orders.Add(new Order(3, "Mysore Sandal", "2018-05-14 12:40:52,531", 1));

orders.Add(new Order(4, "Comb", "2018-09-11 04:40:52,531", 2));

orders.Add(new Order(5, "Trally", "2018-06-10 03:40:52,531", 1));

orders.Add(new Order(6, "Key Chain", "2018-09-01 11:40:52,531", 3));

orders.Add(new Order(7, "Fried Nuts", "2018-09-15 14:40:52,531", 2));

orders.Add(new Order(8, "Neem Oil 100 ML", "2018-10-14 12:40:52,531", 1));

orders.Add(new Order(9, "coconut Oil 200 ML", "2018-10-19 19:40:52,531", 1));

orders.Add(new Order(9, "bag", "2018-10-19 22:40:52,531", 5));

var req3 = from i in orders

orderby i.OrderDate.Date descending, i.Quantity descending

select i;

foreach (var i in req3)

{

Console.WriteLine(i.ToString());

}

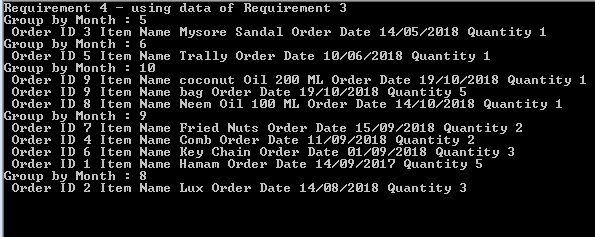
#endregion "Requirement 3"

4.

For the previous exercise, write a LINQ query that displays the details grouped by the month in

the descending order of the order date.

Output:



Code Snippet

#region "Requirement 4"

/\*For the previous exercise,

\* write a LINQ query that displays the details

\* grouped by the month in the descending

\* order of the order date \*/

Console.WriteLine("Requirement 4 - using data of Requirement 3");

var req4 = orders.GroupBy(order => order.OrderDate.Date.Month)

.Select(group =>

new

{

Month = group.Key,

ord = group.OrderByDescending(x => x.OrderDate.Date)

})

.OrderBy(group => group.ord.First().Quantity);

foreach (var group in req4)

{

Console.WriteLine("Group by Month : {0}", group.Month);

foreach (var value in group.ord)

{

Console.WriteLine(" {0}", value.ToString());

}

}

#endregion "Requirement 4"

5.

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

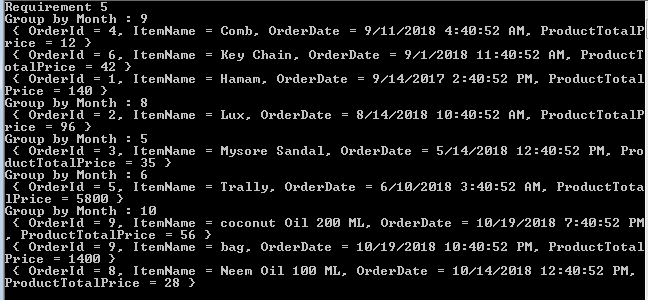
date and quant

ity . 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.

Output:



Code Snippet

#region "Requirement 5"

/\*

\* 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

\* \*/

List<Item> Items = new List<Item>();

Items.Add(new Item("Hamam", 28.00));

Items.Add(new Item("Lux", 32.00));

Items.Add(new Item("Mysore Sandal", 35.00));

Items.Add(new Item("Comb", 6.00));

Items.Add(new Item("Trally", 5800.00));

Items.Add(new Item("Key Chain", 14.00));

Items.Add(new Item("Neem Oil 100 ML", 28.00));

Items.Add(new Item("coconut Oil 200 ML", 56.00));

Items.Add(new Item("bag", 280.00));

var query5 =

from t in

(from order in orders

join item in Items on order.ItemName equals item.ItemName

select new { order.OrderId, order.ItemName, order.OrderDate, ProductTotalPrice = String.Format("{0}", order.Quantity \* item.Price) }

)

group t by new { t.OrderDate.Date.Month } into g

select new

{

Month = g.Key.Month,

ord = g.OrderByDescending(x => x.OrderDate.Date)

} ;

Console.WriteLine("Requirement 5");

foreach (var group in query5)

{

Console.WriteLine("Group by Month : {0}", group.Month);

foreach (var value in group.ord)

{

Console.WriteLine(" {0}", value.ToString());

}

}

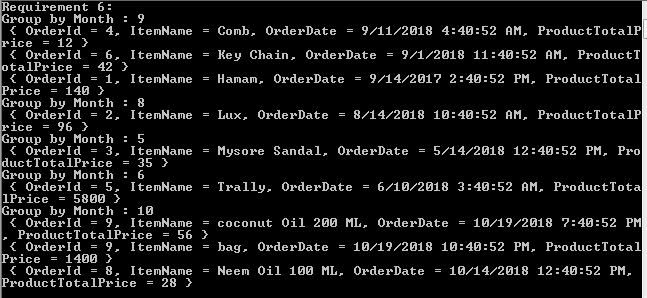
#endregion "Requirement 5"

6.

Do the

previous exercise using anonymous types.

Output:



Code Snippet

#region "Requirement 6"

/\* Do the previous exercise using anonymous types

\* \*/

Console.WriteLine("Requirement 6:");

var query6 =

orders.Join(Items,

order => order.ItemName,

item => item.ItemName,

(order, item) => new

{

order.OrderId,

order.ItemName,

order.OrderDate,

ProductTotalPrice = String.Format("{0}", order.Quantity \* item.Price)

}

).Select(computedorders => computedorders).GroupBy(data => data.OrderDate.Date.Month).Select(group =>

new

{

Month = group.Key,

ord = group.OrderByDescending(x => x.OrderDate.Date)

});

foreach (var group in query6)

{

Console.WriteLine("Group by Month : {0}", group.Month);

foreach (var value in group.ord)

{

Console.WriteLine(" {0}", value.ToString());

}

}

#endregion "Requirement 6"

7.

Check if all the quantities in the Order collection is >0.

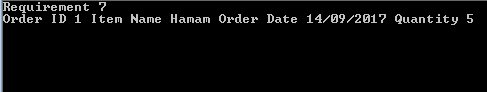
Get the name of the item that was ordered in largest quantity in a single order. (Hint: use LINQ

methods to sort)

Find if there are any orders placed bef

ore Jan of this year.

Output:



Code Snippet

#region "Requirement 7"

/\*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)

\*

\* Find if there are any orders placed before Jan of this year

\* \*/

Console.WriteLine("Requirement 7");

var Query7 = orders.Where(x => x.Quantity > 0 && x.OrderDate.Date.Month > 1 && x.OrderDate.Date.Year < 2018).OrderByDescending(x => x.Quantity).Select(x => x);

foreach (var item in Query7)

{

Console.WriteLine(item.ToString());

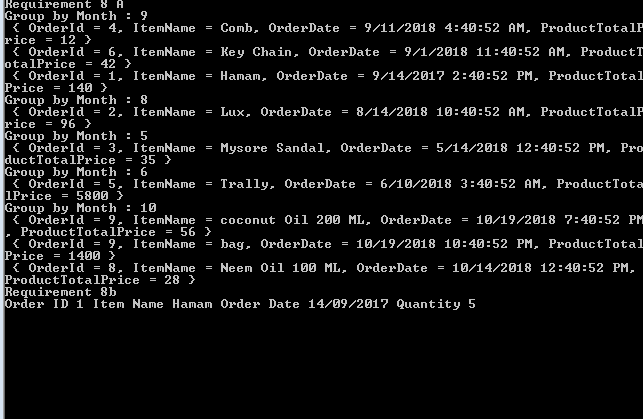
}

#endregion "Requirement 7"

8.

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

Output:



Code Snippet

#region "Requirment 8"

/\* Rewrite the last two example of that used

\* Count using LINQ query methods entirely

\* \*/

Console.WriteLine("Requirement 8 A");

var query8a =

orders.Join(Items,

order => order.ItemName,

item => item.ItemName,

(order, item) => new

{

order.OrderId,

order.ItemName,

order.OrderDate,

ProductTotalPrice = String.Format("{0}", order.Quantity \* item.Price)

}

).Select(computedorders => computedorders).GroupBy(data => data.OrderDate.Date.Month).Select(group =>

new

{

Month = group.Key,

ord = group.OrderByDescending(x => x.OrderDate.Date)

});

foreach (var group in query8a)

{

Console.WriteLine("Group by Month : {0}", group.Month);

foreach (var value in group.ord)

{

Console.WriteLine(" {0}", value.ToString());

}

}

Console.WriteLine("Requirement 8b");

var Query8b = orders.Where(x => x.Quantity > 0 && x.OrderDate.Date.Month > 1 && x.OrderDate.Date.Year < 2018)

.OrderByDescending(x => x.Quantity).Select(x => x);

foreach (var item in Query8b)

{

Console.WriteLine(item.ToString());

}

#endregion "Requirement 8"

9.

Given the array of numbers. Count

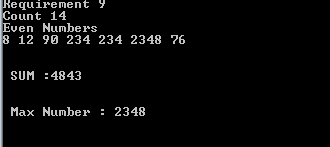
and display even numbers.

Write LINQ to get the sum of quantities for each item and also find out and display t

he item that has

overall maximum orders.

Output:



Code Snippet

#region "Requirement 9"

/\* Given the array of numbers.

\* Count and display even numbers.

\* Write LINQ to get the sum of quantities

\* for each item and also find out and display the item that has overall maximum orders

\*

\* \*/

Console.WriteLine("Requirement 9");

int[] Req9 = { 5, 8, 12, 1, 3, 90, 234, 234, 2348, 76, 77, 87, 1323, 345 };

Console.WriteLine("Count " + Req9.Count());

var val = from i in Req9

where i % 2 == 0

select i;

Console.WriteLine("Even Numbers ");

foreach(var i in val)

Console.Write (i+ " ");

Console.WriteLine("\n \n \n SUM :" + Req9.Sum(x => x) + "\n\n\n Max Number : "+Req9.Max());

#endregion "Requirment 9"