**CODE:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Linq\_Assignment

{

class Program

{

static void Main(string[] args)

{

#region Members

linq l = new linq();

List<Order> orders = new List<Order>() { new Order { OrderID = "O001",ItemName = "Redmi 5A" ,OrderDate = Convert.ToDateTime("2019-01-01"),Quantity = 5},

new Order { OrderID = "O002",ItemName = "Redmi 6" ,OrderDate = Convert.ToDateTime("2019-01-05"),Quantity = 2},

new Order { OrderID = "O003",ItemName = "Redmi 6A" ,OrderDate = Convert.ToDateTime("2018-12-31"),Quantity = 3},

new Order { OrderID = "O004",ItemName = "Redmi 5" ,OrderDate = Convert.ToDateTime("2018-12-02"),Quantity = 4},

new Order { OrderID = "O005",ItemName = "Redmi note 5" ,OrderDate = Convert.ToDateTime("2019-01-11"),Quantity = 6},

new Order { OrderID = "O006",ItemName = "Redmi note 6" ,OrderDate = Convert.ToDateTime("2018-10-23"),Quantity = 7}};

List<Items> items = new List<Items>() { new Items { itemName = "Redmi 5A" ,price = 6000},

new Items { itemName = "Redmi 6" ,price = 8000},

new Items { itemName = "Redmi 6A" ,price = 10000},

new Items { itemName = "Redmi 5" ,price = 5000},

new Items { itemName = "Redmi note 5" ,price = 14000},

new Items { itemName = "Redmi note 6" ,price = 16000}};

List<Players> team1 = new List<Players>(){new Players { playerName = "Suba", Country = "India", Game = "Tennis" },

new Players { playerName = "Raji", Country = "US", Game = "Tennis" },

new Players { playerName = "Raja", Country = "China", Game = "Tennis" },

new Players { playerName = "Jegan", Country = "Japan", Game = "Tennis" } };

List<Players> team2 = new List<Players>(){new Players { playerName = "Rohan", Country = "India", Game = "Tennis" },

new Players { playerName = "Rishon", Country = "US", Game = "Tennis" },

new Players { playerName = "Gayu", Country = "China", Game = "Tennis" },

new Players { playerName = "Sindhu", Country = "Japan", Game = "Tennis" } };

#endregion Members

#region Q.1

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

Console.WriteLine("=========================================================");

int[] a = { 1, 2, 3, 4, 5, 10 };

int[] b = l.GEtCube\_values(a);

Console.WriteLine("==================1.Cube Value start=======================");

foreach (var item in b)

{

Console.WriteLine("Cube value : {0}", item);

}

Console.WriteLine("==================1.Cube Value END=======================");

int[] a1 = { 45, 67, 12, 3, 6, 10 };

int[] b1 = l.GEtCube\_values(a1);

Console.WriteLine("==================1.Cube Value start=======================");

foreach (var item in b1)

{

Console.WriteLine("Cube value : {0}", item);

}

Console.WriteLine("==================1.Cube Value END=======================");

int[] a2 = { 12, 54, 23, 14, 15, 26, 7 };

int[] b2 = l.GEtCube\_values(a2);

Console.WriteLine("==================1.Cube Value start=======================");

foreach (var item in b2)

{

Console.WriteLine("Cube value : {0}", item);

}

Console.WriteLine("==================1.Cube Value END=======================");

#endregion Q.1

#region Q.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 his own his own country\*/

Console.WriteLine("=========================================================");

List<string> match = l.Get\_TennisMatch\_Players(team1, team2);

Console.WriteLine("==================2.TennisMatch start=======================");

foreach (var item in match)

{

Console.WriteLine("Match between : {0}", item);

}

Console.WriteLine("==================3.TennisMatch END=======================");

#endregion Q.2

#region Q.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.\*/

Console.WriteLine("=========================================================");

List<Order> order = l.Get\_Order\_Details(orders);

Console.WriteLine("==================3.Order start=======================");

foreach (var item in order)

{

Console.WriteLine("Order Details : {0} {1} {2} {3}", item.OrderID, item.ItemName, item.OrderDate, item.Quantity);

}

Console.WriteLine("==================3.Order END=======================");

#endregion Q.3

#region Q.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("=========================================================");

List<Order> order1 = l.Get\_Order\_GroupedbyMonth(orders);

Console.WriteLine("==================4.Order by decending Month start=======================");

foreach (Order item in order1)

{

Console.WriteLine("Order Month {4} Order Details : {0} {1} {2} {3}", item.OrderID, item.ItemName, item.OrderDate, item.Quantity, item.Month);

}

Console.WriteLine("==================4.Order by decending Month END=======================");

#endregion Q.4

#region Q.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\*/

Console.WriteLine("=========================================================");

List<Order> order2 = l.Get\_Order\_Price\_GroupedbyMonth(orders, items);

Console.WriteLine("==================5.Order and item by decending Month start=======================");

foreach (Order item in order2)

{

Console.WriteLine("Order Month : '{4}' Order Details : {0} {1} {2} {3} and Order total Price {5}", item.OrderID, item.ItemName, item.OrderDate, item.Quantity, item.Month, item.totalPrice);

}

Console.WriteLine("==================5.Order and item by decending Month END=======================");

#endregion Q.5

#region Q.6

/\* Do the previous exercise using anonymous types\*/

Console.WriteLine("=========================================================");

List<Order> order3 = l.Get\_Order\_Price\_GroupedbyMonth\_anonymous\_type(orders, items);

Console.WriteLine("==================5.Order and item from anonymous types start=======================");

foreach (Order item in order3)

{

Console.WriteLine("Order Month : '{4}' Order Details : {0} {1} {2} {3} and Order total Price {5}", item.OrderID, item.ItemName, item.OrderDate, item.Quantity, item.Month, item.totalPrice);

}

Console.WriteLine("==================5.Order and item from anonymous types Month END=======================");

#endregion Q.6

#region Q.7

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

Console.WriteLine("=========================================================");

List<Order> order4 = l.Get\_Order\_OnConditions\_Quantity(orders);

Console.WriteLine("==================6.Order on conditions start=======================");

foreach (var item in order4)

{

Console.WriteLine("Order Details : {0} {1} {2} {3}", item.OrderID, item.ItemName, item.OrderDate, item.Quantity);

}

Console.WriteLine("==================6.Order on conditions END=======================");

#endregion Q.7

#region Q.8

/\* Get the name of the item that was ordered in largest quantity in a single order. (Hint: use LINQ methods to sort)\*/

Console.WriteLine("=========================================================");

Order Largest\_order = l.Get\_Order\_OnConditions\_Lagrest\_Quantity(orders);

Console.WriteLine("==================8.Order on conditions start=======================");

Console.WriteLine("Order Details : {0} {1} {2} {3}", Largest\_order.OrderID, Largest\_order.ItemName, Largest\_order.OrderDate, Largest\_order.Quantity);

Console.WriteLine("==================8.Order on conditions END=======================");

#endregion Q.8

#region Q.9

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

Console.WriteLine("=========================================================");

List<Order> Jan\_Order = l.Get\_Order\_OnConditions\_Year(orders);

Console.WriteLine("==================9.Order on conditions start=======================");

foreach (var item in Jan\_Order)

{

Console.WriteLine("Order Details : {0} {1} {2} {3}", item.OrderID, item.ItemName, item.OrderDate, item.Quantity);

}

Console.WriteLine("==================9.Order on conditions END=======================");

#endregion Q.9

#region Q.10

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

Console.WriteLine("=========================================================");

List<Order> order5 = l.Get\_Order\_QueryMethod(orders);

Console.WriteLine("==================10.Order on Query Method start=======================");

foreach (var item in order5)

{

Console.WriteLine("Order Details : {0} {1} {2} {3}", item.OrderID, item.ItemName, item.OrderDate, item.Quantity);

}

Console.WriteLine("==================10.Order on Query Method END=======================");

#endregion Q.10

#region Q.11

/\* Given the array of numbers. Count and display even numbers.\*/

Console.WriteLine("=========================================================");

int[] num = { 1, 2, 3, 4, 5, 10, 11, 12, 23, 34, 46 };

int[] output = l.Get\_even\_number(a);

Console.WriteLine("==================1.Count and display even numbers=======================");

Console.WriteLine("Count List: {0}", output.Count());

foreach (var item in output)

{

Console.WriteLine("Even numbers: {0}", item);

}

Console.WriteLine("==================1.Count and display even numbers=======================");

#endregion Q.11

#region Q.12

/\*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("=========================================================");

List<Order> order6 = l.Get\_Order\_Quantity\_Sum(orders);

Console.WriteLine("==================12.sum of quantities for each item start=======================");

foreach (var item in order5)

{

Console.WriteLine("Order Name : {0} ,Order Sum for Each Product : {1}", item.ItemName,item.Quantity);

}

Console.WriteLine("==================12.sum of quantities for each item END=======================");

Order order7 = l.Get\_Order\_Quantity\_Maximum(orders);

Console.WriteLine("==================12.find out and display the item that has overall maximum orders start=======================");

Console.WriteLine("Order Name : {0} ,Order Count : {1}", order7.ItemName, order7.Quantity);

Console.WriteLine("==================12.find out and display the item that has overall maximum orders END=======================");

#endregion Q.12

Console.ReadLine();

}

}

}

--------------------------------------------------------------------------------------------------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Linq;

namespace Linq\_Assignment

{

public class linq

{

public int[] GEtCube\_values(int[] a)

{

int[] Cube = (from l in a

let k = l \* l \* l

where k > 100 && k < 1000

select l).ToArray();

return Cube;

}

public List<string> Get\_TennisMatch\_Players(List<Players> Team1,List<Players> Team2)

{

List<string> match = (from l in Team1

from l1 in Team2

where l.Country != l1.Country

select l.playerName + "vs" + l1.playerName).ToList();

return match;

}

public List<Order> Get\_Order\_Details(List<Order> orders)

{

List<Order> order = (from l in orders

orderby l.OrderDate,l.Quantity descending

select new Order

{

ItemName = l.ItemName,

OrderID = l.OrderID,

OrderDate = l.OrderDate,

Quantity = l.Quantity

}).ToList();

return order;

}

public List<Order> Get\_Order\_GroupedbyMonth(List<Order> orders)

{

var o = (from l in orders

orderby l.OrderDate.Month descending

group l by l.OrderDate.Month

);

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

foreach (var item in o)

{

Order OR = new Order();

OR.Month = item.Key;

foreach (var l in item)

{

OR.OrderID = l.OrderID;

OR.ItemName = l.ItemName;

OR.OrderDate = l.OrderDate;

OR.Quantity = l.Quantity;

order.Add(OR);

}

}

return order;

}

public List<Order> Get\_Order\_Price\_GroupedbyMonth(List<Order> orders,List<Items> items)

{

var o = (from l in orders

from l1 in items

where l.ItemName == l1.itemName

orderby l.OrderDate.Month descending

group new { l.OrderID,l.ItemName,l.OrderDate, l.Quantity , l1.price} by l.OrderDate.Month

);

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

foreach (var item in o)

{

Order OR = new Order();

OR.Month = item.Key;

foreach (var l in item)

{

OR.OrderID = l.OrderID;

OR.ItemName = l.ItemName;

OR.OrderDate = l.OrderDate;

OR.totalPrice = l.Quantity \* l.price;

order.Add(OR);

}

}

return order;

}

public List<Order> Get\_Order\_Price\_GroupedbyMonth\_anonymous\_type(List<Order> orders, List<Items> items)

{

var o = (from l in orders

from l1 in items

where l.ItemName == l1.itemName

orderby l.OrderDate.Month descending

group new { l.OrderID, l.ItemName, l.OrderDate, l.Quantity, l1.price } by l.OrderDate.Month into t

select t.ToList()

);

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

foreach (var item in o)

{

foreach (var l in item)

{

Order OR = new Order();

OR.Month = l.OrderDate.Month;

OR.OrderID = l.OrderID;

OR.ItemName = l.ItemName;

OR.OrderDate = l.OrderDate;

OR.totalPrice = l.Quantity \* l.price;

order.Add(OR);

}

}

return order;

}

public List<Order> Get\_Order\_OnConditions\_Quantity(List<Order> orders)

{

List<Order> order = (from l in orders

where l.Quantity > 0

select new Order

{

ItemName = l.ItemName,

OrderID = l.OrderID,

OrderDate = l.OrderDate,

Quantity = l.Quantity

}).ToList();

return order;

}

public List<Order> Get\_Order\_OnConditions\_Year(List<Order> orders)

{

List<Order> order = (from l in orders

where l.OrderDate.Year < DateTime.Now.Year

select new Order

{

ItemName = l.ItemName,

OrderID = l.OrderID,

OrderDate = l.OrderDate,

Quantity = l.Quantity

}).ToList();

return order;

}

public Order Get\_Order\_OnConditions\_Lagrest\_Quantity(List<Order> orders)

{

Order order = (from l in orders

orderby l.Quantity descending

select new Order

{

ItemName = l.ItemName,

OrderID = l.OrderID,

OrderDate = l.OrderDate,

Quantity = l.Quantity

}).Take(1).SingleOrDefault();

return order;

}

public List<Order> Get\_Order\_QueryMethod(List<Order> orders)

{

List<Order> order = orders.Where(o => o.Quantity > 0 && o.OrderDate.Year < DateTime.Now.Year).OrderByDescending(o => o.Quantity).Select(o => o).ToList();

return order;

}

public int[] Get\_even\_number(int[] num)

{

int[] output = num.Where(n => n % 2 == 0).Select(n => n).ToArray();

return output;

}

public List<Order> Get\_Order\_Quantity\_Sum(List<Order> orders)

{

var o = (from l in orders

group new { l.ItemName, l.Quantity } by l.ItemName into s

select new

{

sum = s.Sum(t=>t.Quantity).ToString(),

Name = s.Select(n => n.ItemName).ToString()

}

);

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

foreach (var item in o)

{

Order OR = new Order();

OR.ItemName = item.Name;

OR.Quantity = Convert.ToInt32(item.sum);

order.Add(OR);

}

return order;

}

public Order Get\_Order\_Quantity\_Maximum(List<Order> orders)

{

var o = (from l in orders

group l by l.ItemName into s

select new

{

sum = s.Sum(t => t.Quantity).ToString(),

Name = s.Key.ToString()

}

);

var od = (from l in orders

group l by l.ItemName into s

select new

{

quantity = s.Sum(t => t.Quantity).ToString(),

Name = s.Select(n => n.ItemName).ToString()

}

).Max(a=>a.quantity);

var Maximum\_order = o.Select(s => s).Where(s => s.sum.ToString() == od.ToString() );

Order OR = new Order();

foreach (var item in Maximum\_order)

{

OR.ItemName = item.Name;

OR.Quantity = Convert.ToInt32(item.sum);

}

return OR;

}

}

}

--------------------------------------------------------------------------------------------------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Linq\_Assignment

{

public class Players

{

public string playerName { get; set; }

public string Country { get; set; }

public string Game { get; set; }

}

}

--------------------------------------------------------------------------------------------------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Linq\_Assignment

{

public class Order

{

public string OrderID { get; set; }

public string ItemName { get; set; }

public DateTime OrderDate { get; set; }

public int Quantity { get; set; }

public double totalPrice { get; set; }

public int Month { get; set; }

}

}

--------------------------------------------------------------------------------------------------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Linq\_Assignment

{

public class Items

{

public string itemName { get; set; }

public double price { get; set; }

}

}

--------------------------------------------------------------------------------------------------------------------------

**OutPUT**:



