LINQ

1. Determinati cuvintele dintr-un text care sunt scrise cu litere mari. (`String.Equals`)
2. Afisati numerele si frecventele lor de apartitie dintr-un sir de numere.
3. Determinati toate numerele distincte dintr-un sir si patratele lor, daca patratele sunt mai mari decat 20.
4. Join
5. Query pe un XML file.

```
       static void Main(string[] args)
        {
            //1
            String text = "ACESTA Este UN Text MARE";
            var rez = from v in text.Split(' ')
                      where v.ToUpper().Equals(v)
                      select v;

            //foreach (string cuv in rez)
            //    Console.WriteLine(cuv);

            //rez.ToList().ForEach(cuv => Console.WriteLine(cuv));
            rez.ToList().ForEach(Console.WriteLine);

            text.Split(' ').Where(v => v.ToUpper().Equals(v)).ToList().ForEach(Console.WriteLine);

            int[] numbers = { 1, 2, 3, 5, 2, 1, 2, 3, 6, 2, 2, 4, 1, 2, 1, 4, 6, 2, 4, 1, 2, 5, 7 };
            //var result = from nr in numbers
            //             group nr by nr into g
            //             select g;

            //result.ToList().ForEach((nr) => Console.WriteLine(nr.Key + " apare de: " + nr.Count()
+ " ori."));

            numbers.GroupBy(nr => nr).ToList().ForEach(nr => Console.WriteLine(nr.Key + " apare de:
" + nr.Count() + " ori."));

            numbers.Distinct().Where(nr => nr * nr > 20).Select(nr => new
            {
                Numar = nr,
                Patrat = nr * nr
            }).ToList().ForEach(nr => Console.WriteLine("Numar: " + nr.Numar + ", Patrat: " +
nr.Patrat));
        }
    }
```

LINQ ofera doua modalitati de a scrie interogari:

1. Folosind sintaxa asemanatoare SQL- denumite *Query expressions (Queryable)*

```
int[] numbers = { 7, 53, 45, 99 };
var res = from n in numbers
          where n > 50
```

```
                          orderby n
                         select n.ToString();
```

2. Extensii de metode (IEnumerable)

```
            var res2 = numbers
                .Where(n => n > 50)
                .OrderBy(n => n)
                .Select(n => n.ToString());
```

Deferred and immediate execution

LINQ queries can execute in two different ways: deferred and immediate:

With deferred execution, the resulting sequence of a LINQ query is not generated until it is required. The following query does not actually execute until Max() is called, and a final result is required:

Exemplu 1: Deferred

```
int[] numbers = { 1, 2, 3, 4, 5 };
var result = numbers.Where(n => n >= 2 && n <= 4);
Console.WriteLine(result.Max());
```

Exemplu 2: immediate

```
string[] words = { "one", "two", "three" };
var result = words.Select((w, i) => new { Index = i, Value = w })
    .Where(w => w.Value.Length == 3).ToList();
```

Adding items to an existing query is another benefit of deferred execution. The following example shows the concept:

Exemple 3:

```
List<String> vegetables = new List<String> { "Carrot", "Selleri" };
var result = from v in vegetables select v;
Console.WriteLine("Elements in vegetables array (before add): " +
result.Count());
vegetables.Add("Broccoli");
Console.WriteLine("Elements in vegetables array (after add): " +
result.Count());
```

Deferred execution makes it useful to combine or extend queries. Have a look at this example, which creates a base query and then extends it into two new separate queries:

```
int[] numbers = { 1, 5, 10, 18, 23 };
var baseQuery = from n in numbers select n;
var oddQuery = from b in baseQuery where b % 2 == 1 select b;
```

LINQ Operatori

Aggregate: Performs a specified operation to each element in a collection, while carrying the result forward.

```
var numbers = new int[] { 1, 2, 3, 4, 5 };
var result = numbers.Aggregate((a, b) => a * b);
Console.WriteLine("Aggregated numbers by multiplication:");
Console.WriteLine(result);
```

Any: Checks if any elements in a collection satisifies a specified condition.

```
string[] names = { "Bob", "Ned", "Amy", "Bill" };
var result = names.Any(n => n.StartsWith("B"));
Debug.WriteLine("Does any of the names start with the letter 'B':");
Debug.WriteLine(result);
```

ElementAtOrDefault: Retrieves element from a collection at specified (zero-based) index, but gets default value if out-of-range.

```
string[] colors = { "Red", "Green", "Blue" };
var resultIndex1 = colors.ElementAtOrDefault(1);
var resultIndex10 = colors.ElementAtOrDefault(10);
Debug.WriteLine("Element at index 1 in the array contains:");
Debug.WriteLine(resultIndex1);
Debug.WriteLine("Element at index 10 in the array does not exist:");
Debug.WriteLine(resultIndex10 == null);
```

SelectMany: Flattens collections into a single collection (similar to cross join in SQL).

```
string[] fruits = { "Grape", "Orange", "Apple" };
int[] amounts = { 1, 2, 3 };
var result = fruits.SelectMany(f => amounts, (f, a) => new
{
    Fruit = f,
    Amount = a
});
Debug.WriteLine("Selecting all values from each array, and mixing them:");
foreach (var o in result)
Debug.WriteLine(o.Fruit + ", " + o.Amount);
```

ToDictionary: Converts collection into a Dictionary with Key and Value.

```
        class English2German
        {
            public string EnglishSalute { get; set; }
            public string GermanSalute { get; set; }
        }

English2German[] english2German =
                {
            new English2German { EnglishSalute = "Good morning", GermanSalute = "Guten Morgen" },
            new English2German { EnglishSalute = "Good day", GermanSalute = "Guten Tag" },
            new English2German { EnglishSalute = "Good evening", GermanSalute = "Guten Abend" },
            };


            var result = english2German.ToDictionary(k => k.EnglishSalute, v => v.GermanSalute);
            Console.WriteLine("Values inserted into dictionary:");

            foreach (KeyValuePair<string, string> dic in result)
                Console.WriteLine(String.Format("English salute {0} is {1} in German", dic.Key,
dic.Value));
```

Concat: Concatenates (combines) two collections.

```
            int[] numbers1 = { 1, 2, 3 };
            int[] numbers2 = { 4, 5, 6 };
            var result = numbers1.Concat(numbers2);
```

Distinct: Removes duplicate elements from a collection

```
            int[] numbers = { 1, 2, 2, 3, 5, 6, 6, 6, 8, 9 };
            var result = (from n in numbers.Distinct()
                          select n);
            Console.WriteLine("Distinct removes duplicate elements:");
            result.ToList().ForEach(x => Console.WriteLine(x));
```

Except: Removes all elements from one collection which exist in another collection.

```
            int[] numbers1 = { 1, 2, 3 };
            int[] numbers2 = { 3, 4, 5 };
            var result = (from n in numbers1.Except(numbers2) select n);
```

GroupBy: Projects elements of a collection into groups by key.

```
            int[] numbers = { 10, 15, 20, 25, 30, 35 };
            var result = from n in numbers
                         group n by (n % 10 == 0) into groups
                         select groups;
            Console.WriteLine("GroupBy has created two groups:");
            foreach (IGrouping<bool, int> group in result)
            {
                if (group.Key == true)
                    Console.WriteLine("Divisible by 10");
                else
                    Console.WriteLine("Not Divisible by 10");
                foreach (int number in group)
                Console.WriteLine(number);
            }
```

Join: Joins two collections by a common key value, and is similar to inner join in SQL

Exemplu 1

```
            string[] warmCountries = { "Turkey", "Italy", "Spain", "Saudi Arabia", "Etiopia",
"Portugal" };
            string[] europeanCountries = { "Denmark", "Germany", "Italy", "Portugal", "Spain" };
            var result = (from w in warmCountries
                          join e in europeanCountries on w equals e
                          select w); // new { warm = w, euro = e }

            Console.WriteLine("Joined countries which are both warm and European using Query
Syntax:");
            foreach (var country in result)
                Console.WriteLine(country);
```

```
Exemplu 2

    internal class Item_mast
    {
        public int ItemId { get; set; }
        public String ItemDes { get; set; }
    }

    internal class Purchase
    {
        public int InvNo { get; set; }
        public int ItemId { get; set; }
        public int PurQty { get; set; }

    }

           List<Item_mast> itemlist = new List<Item_mast>
            {
           new Item_mast { ItemId = 1, ItemDes = "Biscuit  " },
           new Item_mast { ItemId = 2, ItemDes = "Chocolate" },
           new Item_mast { ItemId = 3, ItemDes = "Butter   " },
           new Item_mast { ItemId = 4, ItemDes = "Brade    " },
           new Item_mast { ItemId = 5, ItemDes = "Honey    " }
            };

            List<Purchase> purchlist = new List<Purchase>
            {
           new Purchase { InvNo=100, ItemId = 3,  PurQty = 800 },
           new Purchase { InvNo=101, ItemId = 2,  PurQty = 650 },
           new Purchase { InvNo=102, ItemId = 3,  PurQty = 900 },
           new Purchase { InvNo=103, ItemId = 4,  PurQty = 700 },
           new Purchase { InvNo=104, ItemId = 3,  PurQty = 900 },
           new Purchase { InvNo=105, ItemId = 4,  PurQty = 650 },
           new Purchase { InvNo=106, ItemId = 1,  PurQty = 458 }
            };


            var innerJoin = from e in itemlist
                            join d in purchlist on e.ItemId equals d.ItemId
                            select new
                            {
                              itid = e.ItemId,
                              itdes = e.ItemDes,
                              prqty = d.PurQty
                          };
            Console.WriteLine("Item ID\t\tItem Name\tPurchase Quantity");
            Console.WriteLine("---------------------------------------------------------");
            foreach (var data in innerJoin)
            {
                Console.WriteLine(data.itid + "\t\t" + data.itdes + "\t\t" + data.prqty);
            }
```

OrderBy: Sorts a collection in ascending order.

```
        class Car
        {
            public string Name { get; set; }
            public int HorsePower { get; set; }
        }
      Car[] cars =
            {
                new Car { Name = "Super Car", HorsePower = 215 },
                new Car { Name = "Economy Car", HorsePower = 75 },
                new Car { Name = "Family Car", HorsePower = 145 },
            };
            var result = from c in cars
                         orderby c.HorsePower ascending
                         select c;
            Console.WriteLine("Ordered list of cars by horsepower using Query Syntax:");
            foreach (Car car in result)
                Console.WriteLine(String.Format("{0}: {1} horses", car.Name, car.HorsePower));
        }
```

ThenBy: Use after earlier sorting, to further sort a collection in ascending order.

```
           string[] capitals = { "Berlin", "Paris", "Madrid", "Tokyo", "London", "Athens",
"Beijing", "Seoul" };
            var result = (from c in capitals
                          orderby c.Length
                          select c)
            .ThenBy(c => c);
            Console.WriteLine("Ordered list of capitals, first by length and then alphabetical:");
            foreach (string capital in result)
                Console.WriteLine(capital);
```

XML Query

```
//fetching all MessageTask
           XDocument document = XDocument.Load("D:\\Messages.xml");
            // Fetch All the Messages
            var messages = from r in document.Descendants("messageTask")
                           select new
                           {
                               //Desc = 1,
                               Id = r.Attribute("id").Value,
                               Desc = r.Element("desc").Value,
                               From = r.Element("from").Value,
                               To = r.Element("to").Value,
                               Message = r.Element("message").Value
                           };
            messages.ToList().ForEach(x => Console.WriteLine(x.Id + " " +x.Desc + " " +x.From+"
"+x.Message));


           // Fetch some Messages or a particular one
            var messages2 = from r in document.Descendants("messageTask").Where
                            (x=>x.Attribute("id").Value.CompareTo("1")==0)
                           select new
                           {
                               //Desc = 1,
                               Id = r.Attribute("id").Value,
                               Desc = r.Element("desc").Value,
                               From = r.Element("from").Value,
                               To = r.Element("to").Value,
                               Message = r.Element("message").Value
                           };
            messages2.ToList().ForEach(x => Console.WriteLine(x.Id + " " + x.Desc + " " + x.From + "
" + x.Message));
```

XML – write in file

```
            XmlWriter xmlWriter = XmlWriter.Create("D:\\test.xml");

            xmlWriter.WriteStartDocument();
            xmlWriter.WriteStartElement("messageTasks");

            xmlWriter.WriteStartElement("messageTask");
            xmlWriter.WriteAttributeString("id", "3");
            xmlWriter.WriteStartElement("desc");
            xmlWriter.WriteString("Sarbatori fericite!");
            xmlWriter.WriteEndElement();

            xmlWriter.WriteStartElement("from");
            xmlWriter.WriteString("Jane");
            xmlWriter.WriteEndElement();

            xmlWriter.WriteStartElement("to");
            xmlWriter.WriteString("Michelle");
            xmlWriter.WriteEndElement();

            xmlWriter.WriteStartElement("message");
            xmlWriter.WriteString("Craciun Fericit!");
            xmlWriter.WriteEndElement();

            xmlWriter.WriteEndDocument();
            xmlWriter.Close();
```