

The logo features the word "LINQ" in a bold, black, sans-serif font centered on a light gray background. The text is flanked by two large, black, L-shaped corner brackets. The top-left bracket is positioned in the upper-left quadrant, and the bottom-right bracket is in the lower-right quadrant, creating a frame around the central text.

LINQ

Задача

Создать метод для фильтрации массивов целых чисел, но с возможностью указания алгоритма, применяемого для фильтрации.

Использование делегатов

```
public class Common
{
    public delegate bool IntFilter(int i);
    public static int[] FilterArrayOfInts(int[] ints,
IntFilter filter)
    {
        List<int> aList = new List<int>();
        foreach (int i in ints)
        {
            if (filter(i))
            {
                aList.Add(i);
            }
        }
        return (aList.ToArray());
    }
}
```

Использование делегатов

```
class Program
{
    public static bool IsOdd(int i)
    {
        return ((i & 1) == 1);
    }
    static void Main(string[] args)
    {
        int[] nums = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        int[] oddNums = Common.FilterArrayOfInts(nums,
IsOdd);

        foreach (int i in oddNums)
            Console.WriteLine(i);
    }
}
```

Использование анонимных методов

```
int[] nums = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  
    int[] oddNums = Common.FilterArrayOfInts (nums,  
delegate (int i)  
    { return ((i & 1) == 1); });  
foreach (int i in oddNums)  
    Console.WriteLine(i);
```

Использование лямбда-выражений

`(параметр1, параметр2, параметр3) => выражение`

`(параметр1, параметр2, параметр3) =>`

`{`

`оператор1;`

`оператор2;`

`оператор3;`

`return (тип_возврата_лямбда_выражения);`

`}`

Использование лямбда-выражений

1. `x => x`

2. `x => x.Length > 0`

3. `s => s.Length`

4. `(x, y) => x == y`

5. `(x, y) =>`

```
{ if (x > y)
```

```
    return (x) ;
```

```
else
```

```
    return (y) ;
```

```
}
```

Использование лямбда-выражений

```
int[] nums = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  
    int[] oddNums =  
Common.FilterArrayOfInts(nums, i => ((i & 1) == 1));  
foreach (int i in oddNums)  
    Console.WriteLine(i);
```


LINQ (*Language Integrated Query*)

По большей части LINQ ориентирован на **запросы** — будь то запросы, возвращающие набор подходящих объектов, единственный объект или подмножество полей из объекта либо набора объектов.

В LINQ этот возвращенный набор называется *последовательностью* (*sequence*). Большинство последовательностей LINQ имеют тип [IEnumerable<T>](#), где T — тип данных объектов, находящихся в последовательности.

LINQ

- **LINQ to Objects:** применяется для работы с массивами и коллекциями
- **LINQ to Entities:** используется при обращении к базам данных через технологию Entity Framework
- **LINQ to Sql:** технология доступа к данным в MS SQL Server
- **LINQ to XML:** применяется при работе с файлами XML
- **LINQ to DataSet:** применяется при работе с объектом DataSet
- **Parallel LINQ (PLINQ):** используется для выполнения параллельных запросов

LINQ (*Language Integrated Query*)

```
string[] numbers = { "40", "2012", "176", "5" };
```

```
// Преобразуем массив строк в массив типа int  
используя LINQ
```

```
int[] nums = numbers.Select(s =>  
    Int32.Parse(s)).ToArray();
```

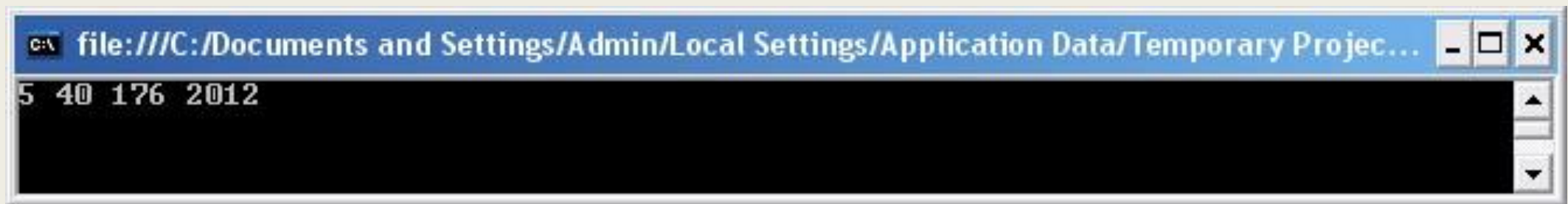
```
foreach (int n in nums)  
    Console.WriteLine(n + " ");
```



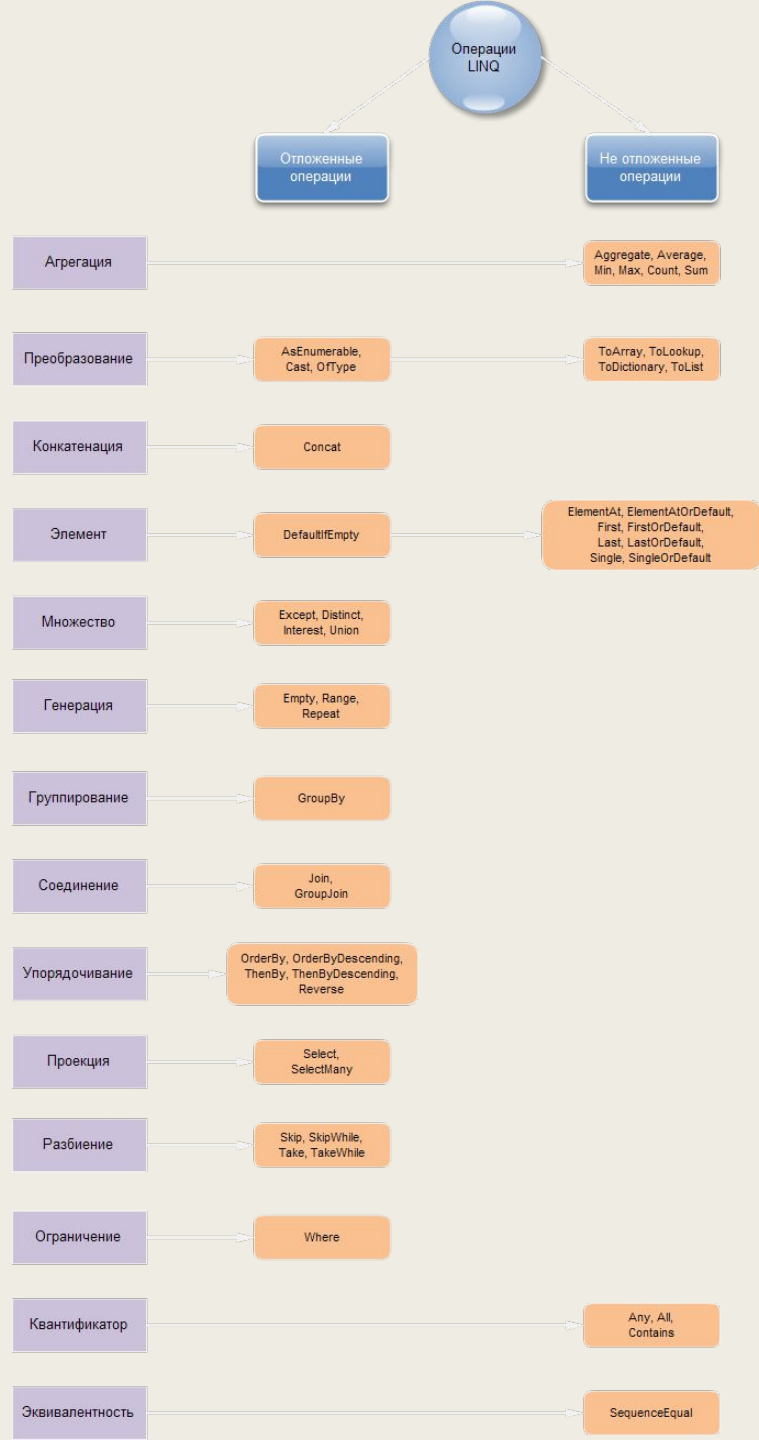
A screenshot of a Windows command prompt window. The title bar shows the file path: "C:\ file:///C:/Documents and Settings/Admin/Local Settings/Application Data/Temporary Projec...". The command prompt displays the output of the LINQ code: "40 2012 176 5".

LINQ (*Language Integrated Query*)

```
string[] numbers = { "40", "2012", "176", "5" };  
  
// Преобразуем массив строк в массив типа int и  
// сортируем по возрастанию используя LINQ  
  
int[] nums = numbers.Select(s =>  
    Int32.Parse(s)).OrderBy(s => s).ToArray();
```



The screenshot shows a Windows command prompt window with a blue title bar. The title bar text is "file:///C:/Documents and Settings/Admin/Local Settings/Application Data/Temporary Projec...". The command prompt area is black with white text. The output of the program is "5 40 176 2012", which are the integers from the original string array sorted in ascending order.



Ленивые (отложенные) вычисления

Ленивые вычисления (англ. *lazy evaluation*, также *отложенные вычисления*) — применяемая в некоторых языках программирования стратегия вычисления, согласно которой вычисления следует откладывать до тех пор, пока не понадобится их.

Метод Enumerable.Range(Int32, Int32)

```
public static  
System.Collections.Generic.IEnumerable<int> Range  
(int start, int count);
```

```
var enumerable = Enumerable.Range(2, 10);  
foreach (int num in mas)  
    Console.WriteLine(num);
```

Enumerable.Repeat<TResult>(TResult, Int32)

```
public static
System.Collections.Generic.IEnumerable<TResult>
Repeat<TResult> (TResult element, int count);

IEnumerable<string> strings =
    Enumerable.Repeat("I like programming.", 15);
foreach (String str in strings)
{
    Console.WriteLine(str);
}
```


Enumerable.Concat<TSource> (IEnumerable<TSource>, IEnumerable<TSource>)

```
public static  
System.Collections.Generic.IEnumerable<TSource>  
Concat<TSource> (this  
System.Collections.Generic.IEnumerable<TSource> first,  
System.Collections.Generic.IEnumerable<TSource>  
second) ;
```

Enumerable.Concat<TSource> (IEnumerable<TSource>, IEnumerable<TSource>)

```
var mas1 = Enumerable.Range(1, 10);  
var mas2 = Enumerable.Repeat(1, 10);  
var mas_union = Enumerable.Concat(mas1, mas2);  
var mas_union2 = mas1.Concat(mas2);
```

Метод Enumerable.Where

```
public static  
System.Collections.Generic.IEnumerable<TSource>  
Where<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, Func<TSource, bool> predicate) ;
```

```
var mas1 = Enumerable.Range(1, 10) ;  
var mas2 = mas1.Where(x => x % 2 == 0) ;
```

Метод Enumerable.Where

```
public static  
System.Collections.Generic.IEnumerable<TSource>  
Where<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, Func<TSource,int,bool> predicate);
```

```
int[] numbers = { 0, 30, 20, 15, 90, 85, 40, 75 };  
IEnumerable<int> query = numbers.Where((number, index)  
=> number <= index * 10);
```

Метод Enumerable.Count

```
public static int Count<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source) ;
```

```
string[] fruits = { "apple", "banana", "mango", "orange",  
"passionfruit", "grape" };  
int numberOfFruits = fruits.Count();
```

Метод Enumerable.Count

```
public static int Count<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, Func<TSource, bool> predicate);
```

```
string[] fruits = { "apple", "banana", "mango", "orange",  
"passionfruit", "grape" };  
int numberOfFruits = fruits.Count(s => s.First() == 'a');
```

Метод Enumerable.First

```
public static TSource First<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source) ;
```

```
int[] numbers = { 9, 34, 65, 92, 87, 435, 3, 54,  
                 83, 23, 87, 435, 67, 12, 19 } ;
```

```
int first = numbers.First() ;
```

```
Console.WriteLine(first) ;
```

Метод Enumerable.First

```
public static TSource First<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, Func<TSource,bool> predicate);
```

```
int[] numbers = { 9, 34, 65, 92, 87, 435, 3, 54,  
                 83, 23, 87, 435, 67, 12, 19 };
```

```
int first = numbers.First(number => number > 80);
```

```
Console.WriteLine(first);
```


Метод

Enumerable.FirstOrDefault

```
public static TSource FirstOrDefault<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, Func<TSource, bool> predicate);
```

```
string[] names = { "Hartono, Tommy", "Adams, Terry", "Andersen,  
Henriette Thaulow", "Hedlund, Magnus", "Ito, Shu" };
```

```
string firstLongName = names.FirstOrDefault(name => name.Length  
> 20);
```

```
Console.WriteLine("The first long name is '{0}'.",  
firstLongName);
```

Метод Enumerable.Last

```
int[] numbers = { 9, 34, 65, 92, 87, 435, 3, 54,  
                 83, 23, 87, 67, 12, 19 };
```

```
int last = numbers.Last();
```

```
int[] numbers = { 9, 34, 65, 92, 87, 435, 3, 54,  
                 83, 23, 87, 67, 12, 19 };
```

```
int last = numbers.Last(num => num > 80);
```

Метод Enumerable.LastOrDefault

```
string[] fruits = { };  
string last = fruits.LastOrDefault();  
Console.WriteLine(  
    String.IsNullOrEmpty(last) ? "<string is null or  
empty>" : last);
```

```
double[] numbers = { 49.6, 52.3, 51.0, 49.4, 50.2, 48.3  
};  
double last50 = numbers.LastOrDefault(n => Math.Round(n)  
== 50.0);
```

Метод Enumerable.LastOrDefault

```
double last40 = numbers.LastOrDefault(n => Math.Round(n)
== 40.0);
Console.WriteLine( "The last number that rounds to 40 is
{0}.", last40 == 0.0 ? "<DOES NOT EXIST>" :
last40.ToString());
```

Метод Enumerable.Sum

```
public static float Sum (this  
System.Collections.Generic.IEnumerable<float> source);
```

```
List<float> numbers = new List<float> { 43.68F, 1.25F,  
583.7F, 6.5F };
```

```
float sum = numbers.Sum();
```

```
Console.WriteLine("The sum of the numbers is {0}.",  
sum);
```

Метод Enumerable.Sum

```
public static float Sum<TSource> (this  
System.Collections.Generic.IEnumerable<TSource> source,  
Func<TSource, float> selector);
```

Метод Enumerable.Sum

```
List<Package> packages = new List<Package> {  
    new Package { Company = "Vineyard", Weight = 25.2 },  
    new Package { Company = "Lucerne", Weight = 18.7},  
    new Package { Company = "Wingtip Toys", Weight = 6.0 },  
    new Package { Company = "Adventure", Weight = 33.8 } };  
  
double totalWeight = packages.Sum(pkg => pkg.Weight);  
Console.WriteLine("The total weight of the packages is: {0}",  
totalWeight);  
}
```

Метод Enumerable.Average

```
public static int Average (this  
System.Collections.Generic.IEnumerable<int> source) ;
```

```
List<int> grades = new List<int> { 78, 92, 100, 37, 81  
};
```

```
double average = grades.Average() ;
```

```
Console.WriteLine ("The average grade is {0}.",  
average) ;
```


Метод Enumerable.Average

```
public static double Average<TSource> (this  
System.Collections.Generic.IEnumerable<TSource> source,  
Func<TSource, long> selector);
```

```
string[] numbers = { "10007", "37", "299846234235" };
```

```
double average = numbers.Average(num =>  
Convert.ToInt64(num));
```

```
Console.WriteLine("The average is {0}.", average);
```

Метод Enumerable.Min

```
public static decimal Min (this  
System.Collections.Generic.IEnumerable<decimal> source);
```

```
double[] doubles = { 1.5E+104, 9E+103, -2E+103 };
```

```
double min = doubles.Min();
```

```
Console.WriteLine("The smallest number is {0}.", min);
```

Метод Enumerable.Min

```
public static TResult Min<TSource, TResult> (this  
System.Collections.Generic.IEnumerable<TSource> source,  
Func<TSource, TResult> selector);
```

```
double[] doubles = { 1.5E+104, 9E+103, -2E+103 };
```

```
double min = doubles.Min(x => Math.Abs(x));
```

```
Console.WriteLine("The smallest absolute is {0}.",  
min);
```

Метод Enumerable.Max

```
public static decimal Max (this  
System.Collections.Generic.IEnumerable<decimal> source) ;
```

```
double[] doubles = { 1.5E+104, 9E+103, -2E+103 } ;
```

```
double min = doubles.Max() ;
```

```
Console.WriteLine("The biggest number is {0}.", max) ;
```

Метод Enumerable.Max

```
public static TResult Max<TSource, TResult> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource, TResult> selector);

double[] doubles = { 1.5E+104, 9E+103, -2E+103 };

double min = doubles.Max(x => Math.Abs(x));

Console.WriteLine("The biggest absolute is {0}.", min);
```

Метод ElementAt

```
public static TSource ElementAt<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, int index);
```

Метод ElementAt

```
string[] names = { "Hartono, Tommy", "Adams,  
Terry", "Andersen, Henriette Thaulow", "Hedlund,  
Magnus", "Ito, Shu" };
```

```
Random random = new  
Random(DateTime.Now.Millisecond);
```

```
string name = names.ElementAt(random.Next(0,  
names.Length));
```

```
Console.WriteLine("The name chosen at random is  
'{0}'.", name);
```

Метод `ElementAtOrDefault`

```
public static TSource ElementAtOrDefault<TSource>  
(this System.Collections.Generic.IEnumerable<TSource>  
source, int index);
```


Метод `ElementAtOrDefault`

```
int index = 20;
```

```
string name = names.ElementAtOrDefault(index);
```

```
Console.WriteLine("The name chosen at index {0} is  
'{1}'.", index, String.IsNullOrEmpty(name) ? "<no name  
at this index>" : name);
```

Метод Reverse

```
public static  
System.Collections.Generic.IEnumerable<TSource>  
Reverse<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source) ;
```

Метод Reverse

```
char[] apple = { 'a', 'p', 'p', 'l', 'e' };  
char[] reversed = apple.Reverse().ToArray();  
foreach (char chr in reversed)  
{  
    Console.Write(chr + " ");  
}  
Console.WriteLine();
```

Метод Enumerable.OrderBy

```
public static  
System.Linq.IOrderedEnumerable<TSource>  
OrderBy<TSource, TKey> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, Func<TSource, TKey> keySelector);
```

Метод Enumerable.OrderBy

```
public static void OrderByEx1 ()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                  new Pet { Name="Boots", Age=4 },
                  new Pet { Name="Whiskers", Age=1 } };

    IEnumerable<Pet> query = pets.OrderBy(pet => pet.Age);

    foreach (Pet pet in query)
    {
        Console.WriteLine("{0} - {1}", pet.Name, pet.Age);
    }
}
```

Метод Enumerable.OrderBy

```
public static System.Linq.IOrderedEnumerable<TSource>  
OrderBy<TSource, TKey> (this  
System.Collections.Generic.IEnumerable<TSource> source,  
Func<TSource, TKey> keySelector,  
System.Collections.Generic.IComparer<TKey> comparer) ;
```

Метод

Enumerable.OrderByDescending

```
public static System.Linq.IOrderedEnumerable<TSource>  
OrderByDescending<TSource, TKey> (this  
System.Collections.Generic.IEnumerable<TSource> source,  
Func<TSource, TKey> keySelector,  
System.Collections.Generic.IComparer<TKey> comparer) ;
```

Метод

Enumerable.OrderByDescending

```
public static System.Linq.IOrderedEnumerable<TSource>  
OrderByDescending<TSource, TKey> (this  
System.Collections.Generic.IEnumerable<TSource> source,  
Func<TSource, TKey> keySelector) ;
```


Метод Enumerable.OrderBy

```
public static System.Linq.IOrderedEnumerable<TSource>  
OrderBy<TSource, TKey> (this  
System.Linq.IOrderedEnumerable<TSource> source,  
Func<TSource, TKey> keySelector,  
System.Collections.Generic.IComparer<TKey> comparer) ;
```

Метод Enumerable.OrderBy

```
public static System.Linq.IOrderedEnumerable<TSource>  
OrderBy<TSource, TKey> (this  
System.Linq.IOrderedEnumerable<TSource> source,  
Func<TSource, TKey> keySelector);
```

Метод Enumerable.OrderBy

```
string[] fruits = { "grape", "passionfruit", "banana",  
"mango", "orange", "raspberry", "apple", "blueberry" };
```

```
IEnumerable<string> query = fruits.OrderBy(fruit =>  
fruit.Length).ThenBy(fruit => fruit);
```

```
foreach (string fruit in query)  
{  
    Console.WriteLine(fruit);  
}
```

Метод Enumerable.OrderByDescending

```
public class CaseInsensitiveComparer : IComparer<string>
{
    public int Compare(string x, string y)
    {
        return string.Compare(x, y, true);
    }
}
```

Метод

Enumerable.OrderByDescending

```
public static void ThenByDescendingEx1 ()
{
    string[] fruits = { "apPLe", "baNaNA", "apple", "APple",
        "orange", "BAAnana", "ORANGE", "apPLE" };

    IEnumerable<string> query = fruits
        .OrderBy(fruit => fruit.Length)
        .ThenByDescending(fruit => fruit, new
            CaseInsensitiveComparer());
}
```

Метод Take

```
public static  
System.Collections.Generic.IEnumerable<TSource>  
Take<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, int count);
```

Метод Take

```
int[] grades = { 59, 82, 70, 56, 92, 98, 85 };
```

```
IEnumerable<int> topThreeGrades =  
    grades.OrderByDescending(grade =>  
grade).Take(3);
```

```
Console.WriteLine("The top three grades are:");
```

```
foreach (int grade in topThreeGrades)
```

```
{
```

```
    Console.WriteLine(grade);
```

```
}
```

Метод TakeWhile

```
public static  
System.Collections.Generic.IEnumerable<TSource>  
TakeWhile<TSource> (this  
System.Collections.Generic.IEnumerable<TSource> source,  
Func<TSource, bool> predicate) ;
```


Метод TakeWhile

```
string[] fruits = { "apple", "banana", "mango",  
"orange", "passionfruit", "grape" };
```

```
IEnumerable<string> query = fruits.TakeWhile(fruit =>  
String.Compare("orange", fruit, true) != 0);
```

```
foreach (string fruit in query)  
{  
    Console.WriteLine(fruit);  
}
```

Метод TakeWhile

```
public static  
System.Collections.Generic.IEnumerable<TSource>  
TakeWhile<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, Func<TSource,int,bool> predicate);
```

Метод TakeWhile

```
string[] fruits = { "apple", "passionfruit", "banana",  
"mango", "orange", "blueberry", "grape", "strawberry" };
```

```
IEnumerable<string> query =  
    fruits.TakeWhile((fruit, index) => fruit.Length >=  
index);
```

```
foreach (string fruit in query)  
{  
    Console.WriteLine(fruit);  
}
```

Метод Skip

```
public static  
System.Collections.Generic.IEnumerable<TSource>  
Skip<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, int count);
```

Метод Skip

```
int[] grades = { 59, 82, 70, 56, 92, 98, 85 };
```

```
IEnumerable<int> lowerGrades =  
    grades.OrderByDescending(g => g).Skip(3);
```

```
Console.WriteLine("All grades except the top three are:");
```

```
foreach (int grade in lowerGrades)
```

```
{
```

```
    Console.WriteLine(grade);
```

```
}
```

Метод SkipWhile

```
public static  
System.Collections.Generic.IEnumerable<TSource>  
SkipWhile<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, Func<TSource, bool> predicate) ;
```

Метод SkipWhile

```
int[] grades = { 59, 82, 70, 56, 92, 98, 85 };  
IEnumerable<int> lowerGrades =  
    grades  
        .OrderByDescending(grade => grade)  
        .SkipWhile(grade => grade >= 80);  
Console.WriteLine("All grades below 80:");  
foreach (int grade in lowerGrades)  
{  
    Console.WriteLine(grade);  
}
```

Метод SkipWhile

```
public static  
System.Collections.Generic.IEnumerable<TSource>  
SkipWhile<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, Func<TSource,int,bool> predicate);
```


Метод SkipWhile

```
int[] amounts = { 5000, 2500, 9000, 8000, 6500, 4000, 1500,  
5500 };
```

```
IEnumerable<int> query =
```

```
    amounts.SkipWhile((amount, index) => amount > index *  
1000);
```

```
foreach (int amount in query)
```

```
{
```

```
    Console.WriteLine(amount);
```

```
}
```

Метод SkipLast

```
public static  
System.Collections.Generic.IEnumerable<TSource>  
SkipLast<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, int count);
```

Метод Select

```
public static  
System.Collections.Generic.IEnumerable<TResult>  
Select<TSource, TResult> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, Func<TSource, int, TResult> selector);
```

Метод Select

```
string[] fruits = { "apple", "banana", "mango",  
"orange", "passionfruit", "grape" };  
  
var query =  
    fruits.Select((fruit, index) =>  
        new { index, str = fruit.Substring(0, index) });  
  
foreach (var obj in query)  
{  
    Console.WriteLine("{0}", obj);  
}
```

Метод Select

```
public static  
System.Collections.Generic.IEnumerable<TResult>  
Select<TSource, TResult> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, Func<TSource, TResult> selector);
```

Метод Select

```
IEnumerable<int> squares =  
    Enumerable.Range(1, 10).Select(x => x * x);
```

```
foreach (int num in squares)  
{  
    Console.WriteLine(num);  
}
```

Метод SelectMany

```
public static  
System.Collections.Generic.IEnumerable<TResult>  
SelectMany<TSource, TResult> (this  
System.Collections.Generic.IEnumerable<TSource>  
source,  
Func<TSource, System.Collections.Generic.IEnumerable  
<TResult>> selector);
```

Метод SelectMany

```
PetOwner[] petOwners =  
{ new PetOwner { Name="Higa, Sidney",  
                Pets = new List<string>{ "Scruffy", "Sam" } },  
  new PetOwner { Name="Ashkenazi, Ronen",  
                Pets = new List<string>{ "Walker", "Sugar" } },  
  new PetOwner { Name="Price, Vernetta",  
                Pets = new List<string>{ "Scratches", "Diesel" } }  
};
```


Метод SelectMany

```
IEnumerable<string> query1 =  
petOwners.SelectMany(petOwner => petOwner.Pets);
```

```
IEnumerable<List<String>> query2 =  
petOwners.Select(petOwner => petOwner.Pets);
```

Метод SelectMany

```
var query = petOwners .SelectMany(petOwner =>  
petOwner.Pets, (petOwner, petName) => new { petOwner,  
petName })
```

```
IEnumerable<string> query = petOwners.SelectMany((petOwner,  
index) => petOwner.Pets.Select(pet => index + pet));
```

Метод Zip

```
public static  
System.Collections.Generic.IEnumerable<TResult>  
Zip<TFirst, TSecond, TResult> (this  
System.Collections.Generic.IEnumerable<TFirst>  
first,  
System.Collections.Generic.IEnumerable<TSecond>  
second, Func<TFirst, TSecond, TResult>  
resultSelector) ;
```

Метод Zip

```
int[] numbers = { 1, 2, 3, 4 };  
string[] words = { "one", "two", "three" };  
  
var numbersAndWords = numbers.Zip(words, (first,  
second) => first + " " + second);  
  
foreach (var item in numbersAndWords)  
    Console.WriteLine(item);
```

Метод Zip

```
public static  
System.Collections.Generic.IEnumerable<ValueTuple<T  
First,TSecond>> Zip<TFirst,TSecond> (this  
System.Collections.Generic.IEnumerable<TFirst>  
first,  
System.Collections.Generic.IEnumerable<TSecond>  
second) ;
```

Метод Distinct

```
public static  
System.Collections.Generic.IEnumerable<TSource>  
Distinct<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source) ;
```

Метод Distinct

```
List<int> ages = new List<int> { 21, 46, 46, 55,  
17, 21, 55, 55 };
```

```
IEnumerable<int> distinctAges = ages.Distinct();
```

```
Console.WriteLine("Distinct ages:");
```

```
foreach (int age in distinctAges)
```

```
{
```

```
    Console.WriteLine(age);
```

```
}
```

Метод Distinct

```
public class Product : IEquatable<Product>
{
    public string Name { get; set; }
    public int Code { get; set; }
    public bool Equals(Product other)
    {
        if (Object.ReferenceEquals(other, null)) return false;
        if (Object.ReferenceEquals(this, other)) return true;
        return Code.Equals(other.Code) && Name.Equals(other.Name);
    }
}
```


Метод Distinct

```
public override int GetHashCode()  
{  
    int hashProductName = Name == null ? 0 :  
Name.GetHashCode();  
    int hashProductCode = Code.GetHashCode();  
    return hashProductName ^ hashProductCode;  
}
```