LINQ 3

Метод Union

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

Метод Union

```
int[] ints1 = { 5, 3, 9, 7, 5, 9, 3, 7 };
int[] ints2 = { 8, 3, 6, 4, 4, 9, 1, 0 };

IEnumerable<int> union = ints1.Union(ints2);

foreach (int num in union)
{
    Console.Write("{0} ", num);
}
```

Метод Union

```
public bool Equals(ProductA other)
    {
        if (other is null)
            return false;
        return this.Name == other.Name && this.Code ==
other.Code;
    }
    public override bool Equals(object obj) =>
Equals(obj as ProductA);
    public override int GetHashCode() => (Name,
Code).GetHashCode();
```

Метод Union

```
IEnumerable<ProductA> union =
  store1.Union(store2);

foreach (var product in union)
    Console.WriteLine(product.Name + " " +
product.Code);
```

Метод Intersect

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

Метод Intersect

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

Метод Except

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

Метод Except

```
double[] numbers1 = { 2.0, 2.0, 2.1, 2.2, 2.3, 2.3, 2.4,
2.5 };
double[] numbers2 = { 2.2 };


IEnumerable<double> onlyInFirstSet =
numbers1.Except(numbers2);


foreach (double number in onlyInFirstSet)
    Console.WriteLine(number);
```

Метод Except

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

Метод Contains

```
public static bool Contains<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
TSource value);
```

```
public static bool Contains<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
TSource value,
System.Collections.Generic.IEqualityComparer<TSource>
comparer);
```

Метод Single

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

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

Метод Single

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

string fruit1 = fruits.Single(fruit => fruit.Length >
10);

Console.WriteLine(fruit1);
```

Метод Single

```
string[] fruits1 = { "orange" };

string fruit1 = fruits1.Single();

Console.WriteLine(fruit1);
```

Метод SingleOrDefault

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

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

Метод Any

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

```
List<int> numbers = new List<int> { 1, 2 };
bool hasElements = numbers.Any();

Console.WriteLine("The list {0} empty.",
    hasElements ? "is not" : "is");
```

Метод Any

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


Pet[] pets = {
  new Pet { Name="Barley", Age=8, Vaccinated=true },
  new Pet { Name="Boots", Age=4, Vaccinated=false },
  new Pet { Name="Whiskers", Age=1, Vaccinated=false }
     };
bool unvaccinated = pets.Any(p => p.Age > 1 &&
p.Vaccinated == false);
```

Метод All

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


Pet[] pets = {
   new Pet { Name="Barley", Age=10 },
   new Pet { Name="Boots", Age=4 },
   new Pet { Name="Whiskers", Age=6 } };


bool allStartWithB = pets.All(pet =>
pet.Name.StartsWith("B"));
```

Метод SequenceEqual

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

```
List<Pet> pets1 = new List<Pet> { pet1, pet2 };
List<Pet> pets2 = new List<Pet> { pet1, pet2 };

bool equal = pets1.SequenceEqual(pets2);
```

Метод SequenceEqual

```
public static bool SequenceEqual<TSource> (this
System.Collections.Generic.IEnumerable<TSource> first,
System.Collections.Generic.IEnumerable<TSource> second,
System.Collections.Generic.IEqualityComparer<TSource>
comparer);
```

Метод SequenceEqual

```
class ProductComparer : IEqualityComparer<Product>
{
    public bool Equals(Product x, Product y)
    {
   if (Object.ReferenceEquals(x, y)) return true;
   if (Object.ReferenceEquals(x, null) ||
Object.ReferenceEquals(y, null))
            return false;
     return x.Code == y.Code && x.Name == y.Name;
    }
```

Метод SequenceEqual (продолжение)

```
public int GetHashCode(Product product)
    {
if (Object.ReferenceEquals(product, null)) return 0;
int hashProductName = product.Name == null ? 0 :
product.Name.GetHashCode();
int hashProductCode = product.Code.GetHashCode();
return hashProductName ^ hashProductCode;
    }
}
```

Метод OfType

```
public static
System.Collections.Generic.IEnumerable<TResult>
OfType<TResult> (this System.Collections.IEnumerable
source);
```

Метод OfType

```
System.Collections.ArrayList fruits = new
System.Collections.ArrayList(4);

fruits.Add("Mango");fruits.Add("Orange");

fruits.Add("Apple");fruits.Add(3.0);fruits.Add("Banana")
;


IEnumerable<string> query1 = fruits.OfType<string>();

Console.WriteLine("Elements of type 'string' are:");

foreach (string fruit in query1)

{

    Console.WriteLine(fruit);
```

Метод Cast

```
public static
System.Collections.Generic.IEnumerable<TResult>
Cast<TResult> (this System.Collections.IEnumerable
source);
```

```
System.Collections.ArrayList fruits = new
System.Collections.ArrayList();

 fruits.Add("mango"); fruits.Add("apple");
fruits.Add("lemon");

IEnumerable<string> query =
fruits.Cast<string>().OrderBy(fruit =>
fruit).Select(fruit => fruit);
```

Метод перевода

```
public static TSource[] ToArray<TSource> (…);
```

```
public static
System.Collections.Generic.Dictionary<TKey,TElement>
ToDictionary<TSource,TKey,TElement> (…);
```

```
public static System.Collections.Generic.List<TSource>
ToList<TSource> (…);
```

```
public static
System.Collections.Generic.HashSet<TSource>
ToHashSet<TSource> (…);
```

Метод Aggregate

```
public static TAccumulate Aggregate<TSource,TAccumulate>
(this System.Collections.Generic.IEnumerable<TSource>
source, TAccumulate seed,
Func<TAccumulate,TSource,TAccumulate> func);
```

```
int[] ints = { 4, 8, 8, 3, 9, 0, 7, 8, 2 };
```

```
int numEven = ints.Aggregate(0, (total, next) =>
                    next % 2 == 0 ? total + 1 : total);
```

```
Console.WriteLine("The number of even integers is: {0}",
numEven);
```

Метод Join

```
public static
System.Collections.Generic.IEnumerable<TResult>
Join<TOuter,TInner,TKey,TResult> (this
System.Collections.Generic.IEnumerable<TOuter> outer,
System.Collections.Generic.IEnumerable<TInner> inner,
Func<TOuter,TKey> outerKeySelector, Func<TInner,TKey>
innerKeySelector, Func<TOuter,TInner,TResult>
resultSelector);
```

Метод Join

```
List<Person> people = new List<Person> { magnus, terry,
charlotte };
List<Pet> pets = new List<Pet> { barley, boots,
whiskers, daisy };


var query = people.Join(pets,
        person => person,
        pet => pet.Owner,
        (person, pet) => new { OwnerName = person.Name,
Pet = pet.Name });
```

Метод GroupJoin

```
public static
System.Collections.Generic.IEnumerable<TResult>
GroupJoin<TOuter,TInner,TKey,TResult> (this
System.Collections.Generic.IEnumerable<TOuter> outer,
System.Collections.Generic.IEnumerable<TInner> inner,
Func<TOuter,TKey> outerKeySelector, Func<TInner,TKey>
innerKeySelector,
Func<TOuter,System.Collections.Generic.IEnumerable<TInne
r>,TResult> resultSelector);
```

Метод GroupJoin

```
List<Person> people = new List<Person> { magnus, terry,
charlotte };
List<Pet> pets = new List<Pet> { barley, boots,
whiskers, daisy };

var query = people.GroupJoin(pets,
   person => person,
   pet => pet.Owner,
   (person, petCollection) => new {
      OwnerName = person.Name,
      Pets = petCollection.Select(pet => pet.Name)
      });
```

Метод GroupBy

```
public static
System.Collections.Generic.IEnumerable<System.Linq.IGrou
ping<TKey,TElement>> GroupBy<TSource,TKey,TElement>
(this System.Collections.Generic.IEnumerable<TSource>
source, Func<TSource,TKey> keySelector,
Func<TSource,TElement> elementSelector);
```

Метод GroupBy

```
List<Pet> pets = new List<Pet>{
   new Pet { Name="Barley", Age=8 },
   new Pet { Name="Boots", Age=4 },
   new Pet { Name="Whiskers", Age=1 },
   new Pet { Name="Daisy", Age=4 } };


IEnumerable<IGrouping<int, string>> query =
pets.GroupBy(pet => pet.Age, pet => pet.Name);
```