**101个LINQ 例子**

**Restriction Operators**

* [Where - Simple 1](http://msdn2.microsoft.com/aa336760" \l "WhereSimple1)

public void Linq1() {  
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
    var lowNums =  
        from n in numbers  
        where n < 5  
        select n;  
    Console.WriteLine("Numbers < 5:");  
    foreach (var x in lowNums) {  
        Console.WriteLine(x);  
    }  
}

* [Where - Simple 2](http://msdn2.microsoft.com/aa336760" \l "WhereSimple2)

public void Linq2() {  
    List<Product> products = GetProductList();  
    var soldOutProducts =  
        from p in products  
        where p.UnitsInStock == 0  
        select p;  
    Console.WriteLine("Sold out products:");  
    foreach (var product in soldOutProducts) {  
        Console.WriteLine("{0} is sold out!", product.ProductName);  
    }  
}

* [Where - Simple 3](http://msdn2.microsoft.com/aa336760" \l "WhereSimple3)

public void Linq3() {  
    List products = GetProductList();  
    var expensiveInStockProducts =  
        from p in products  
        where p.UnitsInStock > 0 && p.UnitPrice > 3.00M  
        select p;  
    Console.WriteLine("In-stock products that cost more than 3.00:");  
    foreach (var product in expensiveInStockProducts) {  
        Console.WriteLine("{0} is in stock and costs more than 3.00.", product.ProductName);  
    }  
}

* [Where - Drilldown](http://msdn2.microsoft.com/aa336760" \l "WhereDrilldown)

public void Linq4() {  
    List customers = GetCustomerList();  
    var waCustomers =  
        from c in customers  
        where c.Region == "WA"  
        select c;  
    Console.WriteLine("Customers from Washington and their orders:");  
    foreach (var customer in waCustomers) {  
        Console.WriteLine("Customer {0}: {1}", customer.CustomerID, customer.CompanyName);  
        foreach (var order in customer.Orders) {  
            Console.WriteLine(" Order {0}: {1}", order.OrderID, order.OrderDate);  
        }  
    }  
}

* [Where - Indexed](http://msdn2.microsoft.com/aa336760" \l "WhereIndexed)

public void Linq5() {  
    string[] digits = { "zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine" };  
    var shortDigits = digits.Where((digit, index) => digit.Length < index);  
    Console.WriteLine("Short digits:");  
    foreach (var d in shortDigits) {  
        Console.WriteLine("The word {0} is shorter than its value.", d);  
    }  
}

**Projection Operators**

* [Select - Simple 1](http://msdn2.microsoft.com/aa336758" \l "SelectSimple1)

public void Linq6() {  
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
    var numsPlusOne =  
        from n in numbers  
        select n + 1;  
    Console.WriteLine("Numbers + 1:");  
    foreach (var i in numsPlusOne) {  
        Console.WriteLine(i);  
    }  
}

* [Select - Simple 2](http://msdn2.microsoft.com/aa336758" \l "SelectSimple2)

public void Linq7() {  
    List products = GetProductList();  
    var productNames =  
        from p in products  
        select p.ProductName;  
    Console.WriteLine("Product Names:");  
    foreach (var productName in productNames) {  
        Console.WriteLine(productName);  
    }  
}

* [Select - Transformation](http://msdn2.microsoft.com/aa336758" \l "SelectTransformation)

public void Linq8() {  
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
    string[] strings = { "zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine" };  
    var textNums =   
        from n in numbers  
        select strings[n];  
    Console.WriteLine("Number strings:");  
    foreach (var s in textNums) {  
        Console.WriteLine(s);  
    }           
}

* [Select - Anonymous Types 1](http://msdn2.microsoft.com/aa336758" \l "SelectAnonymousTypes1)

public void Linq9() {  
    string[] words = { "aPPLE", "BlUeBeRrY", "cHeRry" };  
    var upperLowerWords =  
        from w in words  
        select new {Upper = w.ToUpper(), Lower = w.ToLower()};  
    foreach (var ul in upperLowerWords) {  
        Console.WriteLine("Uppercase: {0}, Lowercase: {1}", ul.Upper, ul.Lower);  
    }  
}

* [Select - Anonymous Types 2](http://msdn2.microsoft.com/aa336758" \l "SelectAnonymousTypes2)

public void Linq10() {  
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
    string[] strings = { "zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine" };  
    var digitOddEvens =  
        from n in numbers  
        select new {Digit = strings[n], Even = (n % 2 == 0)};  
    foreach (var d in digitOddEvens) {  
        Console.WriteLine("The digit {0} is {1}.", d.Digit, d.Even ? "even" : "odd");  
    }  
}

* [Select - Anonymous Types 3](http://msdn2.microsoft.com/aa336758" \l "SelectAnonymousTypes3)

public void Linq11() {  
    List products = GetProductList();  
    var productInfos =  
        from p in products  
        select new { Name =p.ProductName, Category =p.Category, Price = p.UnitPrice};  
    Console.WriteLine("Product Info:");  
    foreach (var productInfo in productInfos) {  
        Console.WriteLine("{0} is in the category {1} and costs {2} per unit.", productInfo.ProductName, productInfo.Category, productInfo.Price);  
    }  
}

* [Select - Indexed](http://msdn2.microsoft.com/aa336758" \l "SelectIndexed)

public void Linq12() {  
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
    var numsInPlace = numbers.Select((num, index) => new {Num = num, InPlace = (num == index)});  
    Console.WriteLine("Number: In-place?");  
    foreach (var n in numsInPlace) {  
        Console.WriteLine("{0}: {1}", n.Num, n.InPlace);  
    }  
}

* [Select - Filtered](http://msdn2.microsoft.com/aa336758" \l "SelectFiltered)

public void Linq13() {  
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
    string[] digits = { "zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine" };  
    var lowNums =  
        from n in numbers  
        where n < 5  
        select digits[n];  
    Console.WriteLine("Numbers < 5:");  
    foreach (var num in lowNums) {  
        Console.WriteLine(num);  
    }       
}

* [SelectMany - Compound from 1](http://msdn2.microsoft.com/aa336758" \l "SelectManyCompoundfrom1)

public void Linq14() {  
    int[] numbersA = { 0, 2, 4, 5, 6, 8, 9 };  
    int[] numbersB = { 1, 3, 5, 7, 8 };  
    var pairs =  
        from a in numbersA  
        from b in numbersB  
        where a < b  
        select new {a, b};  
    Console.WriteLine("Pairs where a < b:");  
    foreach (var pair in pairs) {  
        Console.WriteLine("{0} is less than {1}", pair.a, pair.b);  
    }  
}

* [SelectMany - Compound from 2](http://msdn2.microsoft.com/aa336758" \l "SelectManyCompoundfrom2)

public void Linq15() {  
    List customers = GetCustomerList();  
    var orders =  
        from c in customers,  
        from o in c.Orders  
        where o.Total < 500.00M  
        select new {c.CustomerID, o.OrderID, o.Total};  
    ObjectDumper.Write(orders);  
}

* [SelectMany - Compound from 3](http://msdn2.microsoft.com/aa336758" \l "SelectManyCompoundfrom3)

public void Linq16() {  
    List customers = GetCustomerList();  
    var orders =  
        from c in customers,  
        from o in c.Orders  
        where o.OrderDate >= new DateTime(1998, 1, 1)  
        select new {c.CustomerID, o.OrderID, o.OrderDate};  
    ObjectDumper.Write(orders);  
}

* [SelectMany - from Assignment](http://msdn2.microsoft.com/aa336758" \l "SelectManyfromAssignment)

public void Linq17() {  
    List customers = GetCustomerList();  
    var orders =  
        from c in customers,  
        from o in c.Orders,  
                total = o.Total  
        where total >= 2000.0M  
        select new {c.CustomerID, o.OrderID, total};  
    ObjectDumper.Write(orders);  
}

* [SelectMany - Multiple from](http://msdn2.microsoft.com/aa336758" \l "SelectManyMultiplefrom)

public void Linq18() {  
    List customers = GetCustomerList();  
    DateTime cutoffDate = new DateTime(1997, 1, 1);  
    var orders =  
        from c in customers  
        where c.Region == "WA"  
        from o in c.Orders  
        where o.OrderDate >= cutoffDate  
        select new {c.CustomerID, o.OrderID};  
    ObjectDumper.Write(orders);  
}

* [SelectMany - Indexed](http://msdn2.microsoft.com/aa336758" \l "SelectManyIndexed)

public void Linq19() {  
    List customers = GetCustomerList();  
    var customerOrders =  
        customers.SelectMany(  
            (cust, custIndex) =>  
            cust.Orders.Select(o => "Customer #" + (custIndex + 1) +  
                                    " has an order with OrderID " + o.OrderID) );  
    ObjectDumper.Write(customerOrders);  
}

**Partitioning Operators**

* [Take - Simple](http://msdn2.microsoft.com/aa336757" \l "TakeSimple)

public void Linq20() {  
            int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
            var first3Numbers = numbers.Take(3);  
            Console.WriteLine("First 3 numbers:");  
            foreach (var n in first3Numbers) {  
                Console.WriteLine(n);  
            }  
        }

* [Take - Nested](http://msdn2.microsoft.com/aa336757" \l "TakeNested)

public void Linq21() {  
            List<Customer> customers = GetCustomerList();  
            var first3WAOrders = (  
                from c in customers  
                from o in c.Orders  
                where c.Region == "WA"  
                select new {c.CustomerID, o.OrderID, o.OrderDate} )  
                .Take(3);  
            Console.WriteLine("First 3 orders in WA:");  
            foreach (var order in first3WAOrders) {  
                ObjectDumper.Write(order);  
            }  
        }

* [Skip - Simple](http://msdn2.microsoft.com/aa336757" \l "SkipSimple)

public void Linq22() {  
            int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
            var allButFirst4Numbers = numbers.Skip(4);  
            Console.WriteLine("All but first 4 numbers:");  
            foreach (var n in allButFirst4Numbers) {  
                Console.WriteLine(n);  
            }  
        }

* [Skip - Nested](http://msdn2.microsoft.com/aa336757" \l "SkipNested)

public void Linq23() {  
            List<Customer> customers = GetCustomerList();  
            var waOrders =  
                from c in customers  
                from o in c.Orders  
                where c.Region == "WA"  
                select new {c.CustomerID, o.OrderID, o.OrderDate};  
            var allButFirst2Orders = waOrders.Skip(2);  
            Console.WriteLine("All but first 2 orders in WA:");  
            foreach (var order in allButFirst2Orders) {  
                ObjectDumper.Write(order);  
            }  
        }

* [TakeWhile - Simple](http://msdn2.microsoft.com/aa336757" \l "TakeWhileSimple)

public void Linq24() {  
            int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
            var firstNumbersLessThan6 = numbers.TakeWhile(n => n < 6);  
            Console.WriteLine("First numbers less than 6:");  
            foreach (var n in firstNumbersLessThan6) {  
                Console.WriteLine(n);  
            }  
        }

* [SkipWhile - Simple](http://msdn2.microsoft.com/aa336757" \l "SkipWhileSimple)

public void Linq26() {  
            int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
            var allButFirst3Numbers = numbers.SkipWhile(n => n % 3 != 0);  
            Console.WriteLine("All elements starting from first element divisible by 3:");  
            foreach (var n in allButFirst3Numbers) {  
                Console.WriteLine(n);  
            }  
        }

* [SkipWhile - Indexed](http://msdn2.microsoft.com/aa336757" \l "SkipWhileIndexed)

public void Linq27() {  
            int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
            var laterNumbers = numbers.SkipWhile((n, index) => n >= index);  
            Console.WriteLine("All elements starting from first element less than its position:");  
            foreach (var n in laterNumbers) {  
                Console.WriteLine(n);  
            }  
        }

**Ordering Operators**

* [OrderBy - Simple 1](http://msdn2.microsoft.com/aa336756" \l "simple1)

publicvoid Linq28() {

string[] words = { "cherry", "apple", "blueberry" };

var sortedWords =

from w in words

orderby w

select w;

Console.WriteLine("The sorted list of words:");

foreach (var w in sortedWords) {

Console.WriteLine(w);

}

}

* [OrderBy - Simple 2](http://msdn2.microsoft.com/aa336756" \l "simple2)

public void Linq29() {  
    string[] words = { "cherry", "apple", "blueberry" };  
    var sortedWords =  
        from w in words  
        orderby w.Length  
        select w;  
    Console.WriteLine("The sorted list of words (by length):");  
    foreach (var w in sortedWords) {  
        Console.WriteLine(w);  
    }  
}

* [OrderBy - Simple 3](http://msdn2.microsoft.com/aa336756" \l "simple3)

public void Linq30() {  
    List products = GetProductList();  
    var sortedProducts =  
        from p in products  
        orderby p.ProductName  
        select p;  
    ObjectDumper.Write(sortedProducts);  
}

* [OrderBy - Comparer](http://msdn2.microsoft.com/aa336756" \l "comparer)

public class CaseInsensitiveComparer : IComparer<string>  
{  
    public int Compare(string x, string y)  
    {  
        return string.Compare(x, y, true);  
    }  
}  
public void Linq31() {  
    string[] words = { "aPPLE", "AbAcUs", "bRaNcH", "BlUeBeRrY", "ClOvEr", "cHeRry"};  
    var sortedWords = words.OrderBy(a => a, new CaseInsensitiveComparer());  
    ObjectDumper.Write(sortedWords);  
}

* [OrderByDescending - Simple 1](http://msdn2.microsoft.com/aa336756" \l "descSimple1)

public void Linq32() {  
    double[] doubles = { 1.7, 2.3, 1.9, 4.1, 2.9 };  
    var sortedDoubles =  
        from d in doubles  
        orderby d descending  
        select d;  
    Console.WriteLine("The doubles from highest to lowest:");  
    foreach (var d in sortedDoubles) {  
        Console.WriteLine(d);  
    }  
}

* [OrderByDescending - Simple 2](http://msdn2.microsoft.com/aa336756" \l "descSimple2)

public void Linq33() {  
    List products = GetProductList();  
    var sortedProducts =  
        from p in products  
        orderby p.UnitsInStock descending  
        select p;  
    ObjectDumper.Write(sortedProducts);  
}

* [OrderByDescending - Comparer](http://msdn2.microsoft.com/aa336756" \l "descComparer)

public class CaseInsensitiveComparer : IComparerspan class="qs-keyword">string>

{

publicint Compare(string x, string y)

{

returnstring.Compare(x, y, true);

}

}

publicvoid Linq34() {

string[] words = { "aPPLE", "AbAcUs", "bRaNcH", "BlUeBeRrY", "ClOvEr", "cHeRry"};

var sortedWords = words.OrderByDescending(a => a, new CaseInsensitiveComparer());

ObjectDumper.Write(sortedWords);

}

* [ThenBy - Simple](http://msdn2.microsoft.com/aa336756" \l "thenBySimple)

publicvoid Linq35() {

string[] digits = { "zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine" };

var sortedDigits =

from d in digits

orderby d.Length, d

select d;

Console.WriteLine("Sorted digits:");

foreach (var d in sortedDigits) {

Console.WriteLine(d);

}

}

* [ThenBy - Comparer](http://msdn2.microsoft.com/aa336756" \l "thenByComparer)

public class CaseInsensitiveComparer : IComparerspan class="qs-keyword">string>

{

publicint Compare(string x, string y)

{

returnstring.Compare(x, y, true);

}

}

publicvoid Linq36() {

string[] words = { "aPPLE", "AbAcUs", "bRaNcH", "BlUeBeRrY", "ClOvEr", "cHeRry"};

var sortedWords =

words.OrderBy(a => a.Length)

.ThenBy(a => a, new CaseInsensitiveComparer());

ObjectDumper.Write(sortedWords);

}

* [ThenByDescending - Simple](http://msdn2.microsoft.com/aa336756" \l "thenByDescendingSimple)

publicvoid Linq37() {

List products = [GetProductList();](http://msdn2.microsoft.com/en-us/vcsharp/aa336753.aspx)var sortedProducts =

from p in products

orderby p.Category, p.UnitPrice descendingselect p;

ObjectDumper.Write(sortedProducts);

}

* [ThenByDescending - Comparer](http://msdn2.microsoft.com/aa336756" \l "thenByDescComparer)

public class CaseInsensitiveComparer : IComparerspan class="qs-keyword">string>

{

publicint Compare(string x, string y)

{

returnstring.Compare(x, y, true);

}

}

publicvoid Linq38() {

string[] words = { "aPPLE", "AbAcUs", "bRaNcH", "BlUeBeRrY", "ClOvEr", "cHeRry"};

var sortedWords =

words.OrderBy(a => a.Length)

.ThenByDescending(a => a, new CaseInsensitiveComparer());

ObjectDumper.Write(sortedWords);

}

* [Reverse](http://msdn2.microsoft.com/aa336756" \l "reverse)

publicvoid Linq39() {

string[] digits = { "zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine" };

var reversedIDigits = (

from d in digits

where d[1] == 'i'

select d)

.Reverse();

Console.WriteLine("A backwards list of the digits with a second character of 'i':");

foreach (var d in reversedIDigits) {

Console.WriteLine(d);

}

}

**Grouping Operators**

* [GroupBy - Simple 1](http://msdn2.microsoft.com/aa336754.aspx" \l "simple1)

public void Linq40() {  
            int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
            var numberGroups =  
                from n in numbers  
                group n by n % 5 into g  
                select new { Remainder = g.Key, Numbers = g };  
            foreach (var g in numberGroups) {  
                Console.WriteLine("Numbers with a remainder of {0} when divided by 5:", g.Remainder);  
                foreach (var n in g.Numbers) {  
                    Console.WriteLine(n);  
                }  
            }  
}

* [GroupBy - Simple 2](http://msdn2.microsoft.com/aa336754.aspx" \l "simple2)

public void Linq41() {  
            string[] words = { "blueberry", "chimpanzee", "abacus", "banana", "apple", "cheese" };  
            var wordGroups =  
                from w in words  
                group w by w[0] into g  
                select new { FirstLetter = g.Key, Words = g };  
            foreach (var g in wordGroups) {  
                Console.WriteLine("Words that start with the letter '{0}':", g.FirstLetter);  
                foreach (var w in g.Words) {  
                    Console.WriteLine(w);  
                }  
            }  
        }

* [GroupBy - Simple 3](http://msdn2.microsoft.com/aa336754.aspx" \l "simple3)

public void Linq42() {  
            List<Product> products = [GetProductList()](http://msdn2.microsoft.com/aa336753);  
            var orderGroups =  
                from p in products  
                group p by p.Category into g  
                select new { Category = g.Key, Products = g };  
            ObjectDumper.Write(orderGroups, 1);  
        }

* [GroupBy - Nested](http://msdn2.microsoft.com/aa336754.aspx" \l "nested)

public void Linq43() {  
            List<Customer> customers = GetCustomerList();  
            var customerOrderGroups =   
                from c in customers  
                select  
                    new {c.CompanyName,   
                         YearGroups =  
                             from o in c.Orders  
                             group o by o.OrderDate.Year into yg  
                             select  
                                 new {Year = yg.Key,  
                                      MonthGroups =   
                                          from o in yg  
                                          group o by o.OrderDate.Month into mg  
                                          select new { Month = mg.Key, Orders = mg }  
                                     }  
                        };  
            ObjectDumper.Write(customerOrderGroups, 3);  
        }

* [GroupBy - Comparer](http://msdn2.microsoft.com/aa336754.aspx" \l "comparer)

public class AnagramEqualityComparer : IEqualityComparer   
{   
public bool Equals(string x, string y) { return getCanonicalString(x) == getCanonicalString(y); }   
public int GetHashCode(string obj) { return getCanonicalString(obj).GetHashCode(); }   
private string getCanonicalString(string word)   
{   
 char[] wordChars = word.ToCharArray(); Array.Sort(wordChars); return new string(wordChars);   
}   
}

publicvoid Linq44()   
{   
string[] anagrams = {"from ", " salt", " earn ", " last ", " near ", " form "};   
var orderGroups = anagrams.GroupBy(w => w.Trim(), new AnagramEqualityComparer());   
ObjectDumper.Write(orderGroups, 1);   
}

* [GroupBy - Comparer, Mapped](http://msdn2.microsoft.com/aa336754.aspx" \l "comparerMapped)

public void Linq45() {  
            string[] anagrams = {"from ", " salt", " earn ", " last ", " near ", " form "};  
            var orderGroups = anagrams.GroupBy(  
                        w => w.Trim(),   
                        a => a.ToUpper(),  
                        new AnagramEqualityComparer()  
                        );  
            ObjectDumper.Write(orderGroups, 1);  
        }  
public class AnagramEqualityComparer : IEqualityComparer<string>  
        {  
            public bool Equals(string x, string y) {  
                return getCanonicalString(x) == getCanonicalString(y);  
            }  
            public int GetHashCode(string obj) {  
                return getCanonicalString(obj).GetHashCode();  
            }  
            private string getCanonicalString(string word) {  
                char[] wordChars = word.ToCharArray();  
                Array.Sort<char>(wordChars);  
                return new string(wordChars);  
            }  
        }

**Set Operators**

* [Distinct - 1](http://msdn2.microsoft.com/aa336761.aspx" \l "distinct1)

publicvoid Linq46() {

int[] factorsOf300 = { 2, 2, 3, 5, 5 };

var uniqueFactors = factorsOf300.Distinct();

Console.WriteLine("Prime factors of 300:");

foreach (var f in uniqueFactors) {

Console.WriteLine(f);

}

}

* [Distinct - 2](http://msdn2.microsoft.com/aa336761.aspx" \l "distinct2)

public void Linq47() {

List products = GetProductList();  
 var categoryNames = (

from p in products

select p.Category)

.Distinct();

Console.WriteLine("Category names:");

foreach (var n in categoryNames) {

Console.WriteLine(n);

}

}

* [Union - 1](http://msdn2.microsoft.com/aa336761.aspx" \l "union1)

publicvoid Linq48() {

int[] numbersA = { 0, 2, 4, 5, 6, 8, 9 };

int[] numbersB = { 1, 3, 5, 7, 8 };

var uniqueNumbers = numbersA.Union(numbersB);

Console.WriteLine("Unique numbers from both arrays:");

foreach (var n in uniqueNumbers) {

Console.WriteLine(n);

}

}

* [Union - 2](http://msdn2.microsoft.com/aa336761.aspx" \l "union2)

publicvoid Linq49() {

List products = [GetProductList();](http://msdn2.microsoft.com/en-us/vcsharp/aa336753.aspx)List customers = GetCustomerList();

var productFirstChars =

from p in products

select p.ProductName[0];

var customerFirstChars =

from c in customers

select c.CompanyName[0];

var uniqueFirstChars = productFirstChars.Union(customerFirstChars);

Console.WriteLine("Unique first letters from Product names and Customer names:");

foreach (var ch in uniqueFirstChars) {

Console.WriteLine(ch);

}

}

* [Intersect - 1](http://msdn2.microsoft.com/aa336761.aspx" \l "intersect1)

publicvoid Linq50() {

int[] numbersA = { 0, 2, 4, 5, 6, 8, 9 };

int[] numbersB = { 1, 3, 5, 7, 8 };

var commonNumbers = numbersA.Intersect(numbersB);

Console.WriteLine("Common numbers shared by both arrays:");

foreach (var n in commonNumbers) {

Console.WriteLine(n);

}

}

* [Intersect - 2](http://msdn2.microsoft.com/aa336761.aspx" \l "intersect2)

publicvoid Linq51() {

List products = GetProductList();  
 List customers = GetCustomerList();

var productFirstChars =

from p in products

select p.ProductName[0];

var customerFirstChars =

from c in customers

select c.CompanyName[0];

var commonFirstChars = productFirstChars.Intersect(customerFirstChars);

Console.WriteLine("Common first letters from Product names and Customer names:");

foreach (var ch in commonFirstChars) {

Console.WriteLine(ch);

}

}

* [Except - 1](http://msdn2.microsoft.com/aa336761.aspx" \l "except1)

public void Linq52() {  
    int[] numbersA = { 0, 2, 4, 5, 6, 8, 9 };  
    int[] numbersB = { 1, 3, 5, 7, 8 };  
    IEnumerable<int> aOnlyNumbers = numbersA.Except(numbersB);  
    Console.WriteLine("Numbers in first array but not second array:");  
    foreach (var n in aOnlyNumbers) {  
        Console.WriteLine(n);  
    }  
}

* [Except - 2](http://msdn2.microsoft.com/aa336761.aspx" \l "except2)

public void Linq53() {  
    List products = GetProductList();  
    List customers = GetCustomerList();  
    var productFirstChars =  
        from p in products  
        select p.ProductName[0];  
    var customerFirstChars =  
        from c in customers  
        select c.CompanyName[0];  
    var productOnlyFirstChars = productFirstChars.Except(customerFirstChars);  
    Console.WriteLine("First letters from Product names, but not from Customer names:");  
    foreach (var ch in productOnlyFirstChars) {  
        Console.WriteLine(ch);  
    }  
}

**Conversion Operators**

* [To Array](http://msdn2.microsoft.com/aa336748" \l "ToArray)

public void Linq54() {  
    double[] doubles = { 1.7, 2.3, 1.9, 4.1, 2.9 };  
    var sortedDoubles =  
        from d in doubles  
        orderby d descending  
        select d;  
    var doublesArray = sortedDoubles.ToArray();  
    Console.WriteLine("Every other double from highest to lowest:");  
    for (int d = 0; d < doublesArray.Length; d += 2) {  
        Console.WriteLine(doublesArray[d]);  
    }   
}

* [To List](http://msdn2.microsoft.com/aa336748" \l "ToList)

public void Linq55() {  
    string[] words = { "cherry", "apple", "blueberry" };  
    var sortedWords =  
        from w in words  
        orderby w  
        select w;  
    var wordList = sortedWords.ToList();  
    Console.WriteLine("The sorted word list:");  
    foreach (var w in wordList) {  
        Console.WriteLine(w);  
    }  
}

* [To Dictionary](http://msdn2.microsoft.com/aa336748" \l "ToDictionary)

public void Linq56() {  
    var scoreRecords = new [] { new {Name = "Alice", Score = 50},  
                                new {Name = "Bob" , Score = 40},  
                                new {Name = "Cathy", Score = 45}  
                              };  
    var scoreRecordsDict = scoreRecords.ToDictionary(sr => sr.Name);  
    Console.WriteLine("Bob's score: {0}", scoreRecordsDict["Bob"]);  
}

* [OfType](http://msdn2.microsoft.com/aa336748" \l "OfType)

public void Linq57() {  
    object[] numbers = { null, 1.0, "two", 3, 4.0f, 5, "six", 7.0 };  
    var doubles = numbers.OfType<double>();  
    Console.WriteLine("Numbers stored as doubles:");  
    foreach (var d in doubles) {  
        Console.WriteLine(d);  
    }  
}

**Element Operators**

* [First - Simple](http://msdn2.microsoft.com/aa336750" \l "FirstSimple)

public void Linq58() {  
    List products = GetProductList();  
    Product product12 = (  
        from p in products  
        where p.ProductID == 12  
        select p )  
        .First();  
    ObjectDumper.Write(product12);  
}

* [First - Indexed](http://msdn2.microsoft.com/aa336750" \l "FirstIndexed)

public void Linq60() {  
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
    int evenNum = numbers.First((num, index) => (num % 2 == 0) && (index % 2 == 0));  
    Console.WriteLine("{0} is an even number at an even position within the list.", evenNum);  
}

* [FirstOrDefault - Simple](http://msdn2.microsoft.com/aa336750" \l "FirstOrDefaultSimple)

public void Linq61() {  
    int[] numbers = {};  
    int firstNumOrDefault = numbers.FirstOrDefault();  
    Console.WriteLine(firstNumOrDefault);  
}

* [FirstOrDefault - Condition](http://msdn2.microsoft.com/aa336750" \l "FirstOrDefaultCondition)

public void Linq62() {  
    List products = GetProductList();  
    Product product789 = products.FirstOrDefault(p => p.ProductID == 789);  
    Console.WriteLine("Product 789 exists: {0}", product789 != null);  
}

* [FirstOrDefault - Indexed](http://msdn2.microsoft.com/aa336750" \l "FirstOrDefaultIndexed)

public void Linq63() {  
    double?[] doubles = { 1.7, 2.3, 4.1, 1.9, 2.9 };  
    double? num = doubles.FirstOrDefault((n, index) => (n >= index - 0.5 && n <= index + 0.5));  
    if (num != null)  
        Console.WriteLine("The value {1} is within 0.5 of its index position.", num);  
    else  
        Console.WriteLine("There is no number within 0.5 of its index position.");  
}

* [ElementAt](http://msdn2.microsoft.com/aa336750" \l "aa336750#)

public void Linq64() {  
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
    int fourthLowNum = (  
        from n in numbers  
        where n < 5  
        select n )  
        .ElementAt(3); // 3 because sequences use 0-based indexing  
    Console.WriteLine("Fourth number < 5: {0}", fourthLowNum);  
}

**Generation Operators**

* [Range](http://msdn2.microsoft.com/aa336752.aspx" \l "range)

public void Linq65() {   
   var numbers =   
      from n in Sequence.Range(100, 50)   
      selectnew {Number = n, OddEven = n % 2 == 1 ? "odd" : "even"};   
   foreach (var n in numbers) {   
      Console.WriteLine("The number {0} is {1}.", n.Number, n.OddEven);   
   }   
}

* [Repeat](http://msdn2.microsoft.com/aa336752.aspx" \l "repeat)

public void Linq66() {   
   var numbers = Sequence.Repeat(7, 10);   
   foreach (var n in numbers) {   
      Console.WriteLine(n);   
   }   
}

**Quantifiers**

* [Any - Simple](http://msdn2.microsoft.com/aa336759.aspx" \l "anySimple)

public void Linq67() {   
   string[] words = { "believe", "relief", "receipt", "field" };   
   bool iAfterE = words.Any(w => w.Contains("ei"));   
   Console.WriteLine("There is a word that contains in the list that contains 'ei': {0}", iAfterE);   
}

* [Any - Indexed](http://msdn2.microsoft.com/aa336759.aspx" \l "anyIndexed)

public void Linq68() {   
   int[] numbers = { -9, -4, -8, -3, -5, -2, -1, -6, -7 };   
   bool negativeMatch = numbers.Any((n, index) => n == -index);   
   Console.WriteLine("There is a number that is the negative of its index: {0}", negativeMatch);   
}

* [Any - Grouped](http://msdn2.microsoft.com/aa336759.aspx" \l "anyGrouped)

public void Linq69() {   
   List products = [GetProductList();](http://msdn2.microsoft.com/aa336753.aspx)  
   var productGroups =   
      from p in products   
      group p by p.Category into g   
      where g.Group.Any(p => p.UnitsInStock == 0)   
      select new {Category = g.Key, Products = g.Group};   
   ObjectDumper.Write(productGroups, 1);   
}

* [All - Simple](http://msdn2.microsoft.com/aa336759.aspx" \l "allSimple)

public void Linq70() {   
   int[] numbers = { 1, 11, 3, 19, 41, 65, 19 };  
   bool onlyOdd = numbers.All(n => n % 2 == 1);  
   Console.WriteLine("The list contains only odd numbers: {0}", onlyOdd);   
}

* [All - Indexed](http://msdn2.microsoft.com/aa336759.aspx" \l "allIndexed)

public void Linq71() {   
   int[] lowNumbers = { 1, 11, 3, 19, 41, 65, 19 };   
   int[] highNumbers = { 7, 19, 42, 22, 45, 79, 24 };   
   bool allLower = lowNumbers.All((num, index) => num < highNumbers[index]);   
   Console.WriteLine("Each number in the first list is lower than its counterpart in the second list: {0}", allLower);   
}

* [All - Grouped](http://msdn2.microsoft.com/aa336759.aspx" \l "allGrouped)

public void Linq72() {   
   List products = [GetProductList();](http://msdn2.microsoft.com/aa336753.aspx)  
   var productGroups =   
      from p in products   
      group p by p.Category into g   
      where g.Group.All(p => p.UnitsInStock > 0)   
      select new {Category = g.Key, Products = g.Group};   
   ObjectDumper.Write(productGroups, 1);   
}

**Aggregate Operators**

* [Count - Simple](http://msdn2.microsoft.com/aa336747" \l "countSimple)

public void Linq73() {   
   int[] factorsOf300 = { 2, 2, 3, 5, 5 };   
   int uniqueFactors = factorsOf300.Distinct().Count();   
   Console.WriteLine("There are {0} unique factors of 300.", uniqueFactors);   
}

* [Count - Conditional](http://msdn2.microsoft.com/aa336747" \l "countConditional)

public void Linq74() {   
   int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };   
   int oddNumbers = numbers.Count(n => n % 2 == 1);   
   Console.WriteLine("There are {0} odd numbers in the list.", oddNumbers);   
}

* [Count - Indexed](http://msdn2.microsoft.com/aa336747" \l "CountIndexed)

public void Linq75() {   
   int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
   int oddEvenMatches = numbers.Count((n, index) => n % 2 == index % 2);  
   Console.WriteLine("There are {0} numbers in the list whose odd/even status " +   
        "matches that of their position.", oddEvenMatches);   
}

* [Count - Nested](http://msdn2.microsoft.com/aa336747" \l "countNested)

public void Linq76() {   
   List customers = GetCustomerList();  
   var orderCounts =   
      from c in customers   
      select new {c.CustomerID, OrderCount = c.Orders.Count()};  
   ObjectDumper.Write(orderCounts);   
}

* [Count - Grouped](http://msdn2.microsoft.com/aa336747" \l "countGrouped)

public void Linq77() {   
   List products = GetProductList();  
   var categoryCounts =   
      from p in products   
      group p by p.Category into g   
      select new {Category = g.Key, ProductCount = g.Group.Count()};  
   ObjectDumper.Write(categoryCounts);   
}

* [Sum - Simple](http://msdn2.microsoft.com/aa336747" \l "sumSimple)

public void Linq78() {   
   int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };   
   double numSum = numbers.Sum();   
   Console.WriteLine("The sum of the numbers is {0}.", numSum);   
}

* [Sum - Projection](http://msdn2.microsoft.com/aa336747" \l "sumProjection)

public void Linq79() {  
   string[] words = { "cherry", "apple", "blueberry" };  
   double totalChars = words.Sum(w => w.Length);  
   Console.WriteLine("There are a total of {0} characters in these words.", totalChars);   
}

* [Sum - Grouped](http://msdn2.microsoft.com/aa336747" \l "sumGrouped)

public void Linq80() {   
   List products = [GetProductList();](http://msdn2.microsoft.com/aa336753.aspx)  
   var categories =   
      from p in products   
      group p by p.Category into g   
      select new {Category = g.Key, TotalUnitsInStock = g.Group.Sum(p => p.UnitsInStock)};   
   ObjectDumper.Write(categories);   
}

* [Min - Simple](http://msdn2.microsoft.com/aa336747" \l "minSimple)

public void Linq81() {   
   int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };   
   int minNum = numbers.Min();   
   Console.WriteLine("The minimum number is {0}.", minNum);   
}

* [Min - Projection](http://msdn2.microsoft.com/aa336747" \l "minProjection)

public void Linq82() {   
   string[] words = { "cherry", "apple", "blueberry" };   
   int shortestWord = words.Min(w => w.Length);   
   Console.WriteLine("The shortest word is {0} characters long.", shortestWord);   
}

* [Min - Grouped](http://msdn2.microsoft.com/aa336747" \l "minGrouped)

public void Linq83() {   
   List products = GetProductList();  
   var categories =   
      from p in products   
      group p by p.Category into g   
      select new {Category = g.Key, CheapestPrice = g.Group.Min(p => p.UnitPrice)};   
   ObjectDumper.Write(categories);   
}

* [Min - Elements](http://msdn2.microsoft.com/aa336747" \l "minElements)

public void Linq84() {   
   List products = [GetProductList();](http://msdn2.microsoft.com/aa336753.aspx)  
   var categories =   
      from p in products   
      group p by p.Category into g   
      from minPrice = g.Group.Min(p => p.UnitPrice)   
      select new {Category = g.Key, CheapestProducts = g.Group.Where(p => p.UnitPrice == minPrice)};  
   ObjectDumper.Write(categories, 1);   
}

* [Max - Simple](http://msdn2.microsoft.com/aa336747" \l "maxSimple)

public void Linq85() {   
   int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
   int maxNum = numbers.Max();   
   Console.WriteLine("The maximum number is {0}.", maxNum);   
}

* [Max - Projection](http://msdn2.microsoft.com/aa336747" \l "maxProjection)

public void Linq86() {   
   string[] words = { "cherry", "apple", "blueberry" };   
   int longestLength = words.Max(w => w.Length);   
   Console.WriteLine("The longest word is {0} characters long.", longestLength);   
}

* [Max - Grouped](http://msdn2.microsoft.com/aa336747" \l "maxGrouped)

public void Linq87() {   
   List products = [GetProductList();](http://msdn2.microsoft.com/aa336753.aspx)  
   var categories =   
      from p in products   
      group p by p.Category into g   
      select new {Category = g.Key, MostExpensivePrice = g.Group.Max(p => p.UnitPrice)};   
   ObjectDumper.Write(categories);   
}

* [Max - Elements](http://msdn2.microsoft.com/aa336747" \l "maxElements)

public void Linq88() {   
   List products = [GetProductList();](http://msdn2.microsoft.com/aa336753.aspx)  
   var categories =   
      from p in products   
      group p by p.Category into g   
      from maxPrice = g.Group.Max(p => p.UnitPrice)   
      select new {Category = g.Key, MostExpensiveProducts = g.Group.Where(p => p.UnitPrice == maxPrice)};   
   ObjectDumper.Write(categories, 1);   
}

* [Average - Simple](http://msdn2.microsoft.com/aa336747" \l "avgSimple)

public void Linq89() {   
   int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };   
   double averageNum = numbers.Average();   
   Console.WriteLine("The average number is {0}.", averageNum);   
}

* [Average - Projection](http://msdn2.microsoft.com/aa336747" \l "avgProjection)

public void Linq90() {   
   string[] words = { "cherry", "apple", "blueberry" };   
   double averageLength = words.Average(w => w.Length);   
   Console.WriteLine("The average word length is {0} characters.", averageLength);   
}

* [Average - Grouped](http://msdn2.microsoft.com/aa336747" \l "avgGrouped)

public void Linq91() {   
   List products = [GetProductList();](http://msdn2.microsoft.com/aa336753.aspx)  
   var categories =   
      from p in products   
      group p by p.Category into g   
      select new {Category = g.Key, AveragePrice = g.Group.Average(p => p.UnitPrice)};   
   ObjectDumper.Write(categories);   
}

* [Fold - Simple](http://msdn2.microsoft.com/aa336747" \l "foldSimple)

public void Linq92() {   
   double[] doubles = { 1.7, 2.3, 1.9, 4.1, 2.9 };   
   double product = doubles.Fold((runningProduct, nextFactor) => runningProduct \* nextFactor);   
   Console.WriteLine("Total product of all numbers: {0}", product);   
}

* [Fold - Seed](http://msdn2.microsoft.com/aa336747" \l "foldSeed)

public void Linq93() {   
   double startBalance = 100.0;   
   int[] attemptedWithdrawals = { 20, 10, 40, 50, 10, 70, 30 };   
   double endBalance =   
      attemptedWithdrawals.Fold(startBalance,   
         (balance, nextWithdrawal) =>   
            ( (nextWithdrawal <= balance) ? (balance - nextWithdrawal) : balance ) );   
   Console.WriteLine("Ending balance: {0}", endBalance);   
}

**Miscellaneous Operators**

* [Concat - 1](http://msdn2.microsoft.com/aa336755" \l "Concat1)

public void Linq94() {  
    int[] numbersA = { 0, 2, 4, 5, 6, 8, 9 };  
    int[] numbersB = { 1, 3, 5, 7, 8 };  
    var allNumbers = numbersA.Concat(numbersB);  
    Console.WriteLine("All numbers from both arrays:");  
    foreach (var n in allNumbers) {  
        Console.WriteLine(n);  
    }  
}

* [Concat - 2](http://msdn2.microsoft.com/aa336755" \l "concat2)

public void Linq95() {  
    List customers = GetCustomerList();  
    List products = GetProductList();  
    var customerNames =  
        from c in customers  
        select c.CompanyName;  
    var productNames =  
        from p in products  
        select p.ProductName;  
    var allNames = customerNames.Concat(productNames);  
    Console.WriteLine("Customer and product names:");  
    foreach (var n in allNames) {  
        Console.WriteLine(n);  
    }  
}

* [EqualAll - 1](http://msdn2.microsoft.com/aa336755" \l "equalAll1)

public void Linq96() {  
    var wordsA = new string[] { "cherry", "apple", "blueberry" };  
    var wordsB = new string[] { "cherry", "apple", "blueberry" };  
    bool match = wordsA.EqualAll(wordsB);  
    Console.WriteLine("The sequences match: {0}", match);  
}

* [EqualAll - 2](http://msdn2.microsoft.com/aa336755" \l "equalAll2)

public void Linq97() {  
    var wordsA = new string[] { "cherry", "apple", "blueberry" };  
    var wordsB = new string[] { "apple", "blueberry", "cherry" };  
    bool match = wordsA.EqualAll(wordsB);  
    Console.WriteLine("The sequences match: {0}", match);  
}

**Custom Sequence Operators**

* [Combine](http://msdn2.microsoft.com/aa336749" \l "combine)

public static class CustomSequenceOperators  
{  
    public static IEnumerable Combine(this IEnumerable first, IEnumerable second, Func func) {  
        using (IEnumerator e1 = first.GetEnumerator(), e2 = second.GetEnumerator()) {  
            while (e1.MoveNext() && e2.MoveNext()) {  
                yield return func(e1.Current, e2.Current);  
            }  
        }  
    }  
}  
public void Linq98() {              
    int[] vectorA = { 0, 2, 4, 5, 6 };  
    int[] vectorB = { 1, 3, 5, 7, 8 };  
    int dotProduct = vectorA.Combine(vectorB, (a, b) => a \* b).Sum();  
    Console.WriteLine("Dot product: {0}", dotProduct);  
}

**Query Execution**

* [Deferred](http://msdn2.microsoft.com/aa336751" \l "Deferred)

public void Linq99() {  
    // Sequence operators form first-class queries that  
    // are not executed until you enumerate over them.  
    int[] numbers = new int[] { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
    int i = 0;  
    var q =  
        from n in numbers  
        select ++i;  
    // Note, the local variable 'i' is not incremented  
    // until each element is evaluated (as a side-effect):  
    foreach (var v in q) {  
        Console.WriteLine("v = {0}, i = {1}", v, i);           
    }   
}

* [Immediate](http://msdn2.microsoft.com/aa336751" \l "Immediate)

public void Linq100() {  
    // Methods like ToList() cause the query to be  
    // executed immediately, caching the results.  
    int[] numbers = new int[] { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };          
    int i = 0;  
    var q = (  
        from n in numbers  
        select ++i )  
        .ToList();  
    // The local variable i has already been fully  
    // incremented before we iterate the results:  
    foreach (var v in q) {  
        Console.WriteLine("v = {0}, i = {1}", v, i);  
    }   
}

* [Query Reuse](http://msdn2.microsoft.com/aa336751" \l "QueryReuse)

public void Linq101() {  
    // Deferred execution lets us define a query once  
    // and then reuse it later after data changes.  
    int[] numbers = new int[] { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
    var lowNumbers =  
        from n in numbers  
        where n <= 3  
        select n;  
    Console.WriteLine("First run numbers <= 3:");  
    foreach (int n in lowNumbers) {  
        Console.WriteLine(n);  
    }  
    for (int i = 0; i < 10; i++) {  
        numbers[i] = -numbers[i];  
    }  
    // During this second run, the same query object,  
    // lowNumbers, will be iterating over the new state  
    // of numbers[], producing different results:  
    Console.WriteLine("Second run numbers <= 3:");  
    foreach (int n in lowNumbers) {  
        Console.WriteLine(n);  
    }  
}