**First Approach in LINQ**

**What is LINQ?**

LINQ is an acronym for Language Integrated Query, which is descriptive for where it’s used and what it does. The *Language Integrated* part means that LINQ is part of programming language syntax. In particular, both C# and VB are languages that ship with .NET and have LINQ capabilities.

LINQ is used for querying data. Notice that I used the generic term “data” and didn’t indicate what type of data. That’s because LINQ can be used to query many different types of data, including relational, XML, and even objects. Another way to describe LINQ is that it is programming language syntax that is used to query data.

**LINQ**

LINQ is part of the Microsoft .NET Framework and is used to query data from specific data providers like LINQ to Objects, LINQ to Entities and LINQ to SQL.

Using LINQ for everyday tasks to query objects and whatever is loaded in memory can be very beneficial, as it may replace many trivial for-each loops and make your code easier to read, understand and maintain.

**LINQ to Objects**

The LINQ to Objects provider contains a handy set of standard query operators to work with in-memory IEnumerable<T> collections.

**LINQ Standard Query Classification**

1. Aggregation
2. Conversion
3. Element
4. Generation
5. Grouping
6. Join
7. Ordering
8. Other
9. Partitioning
10. Projection
11. Quantifiers
12. Restriction
13. Set

**1. Aggregate**

**1.1 Aggregate** **(simple)**

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

This Lambda Expression sample aggregates numbers in array by multiplication

**Program:**

private static void Sample\_Aggregate\_Lambda\_Simple()

{

var numbers = new int[] { 1, 2, 3, 4, 5 };

var result = numbers.Aggregate((a, b) => a \* b);

Debug.WriteLine("Aggregated numbers by multiplication:");

Debug.WriteLine(result);

}

**Output:**

Aggregated numbers by multiplication:

120

**1.2 Aggregate (seed)**

Performs a specified operation to each element in a collection, while carrying the result forward. Seed sets initial value.

This Lambda Expression sample aggregates numbers in array by addition, using a seed (seed is an initial value).

**Program:**

private static void Sample\_Aggregate\_Lambda\_Seed()

{

var numbers = new int[] { 1, 2, 3 };

var result = numbers.Aggregate(10, (a, b) => a + b);

Debug.WriteLine("Aggregated numbers by addition with a seed of 10:");

Debug.WriteLine(result);

}

**Output:**

Aggregated numbers by addition with a seed of 10:

16

**1.3 Average**

Computes the average value for a numeric collections.

This Lambda Expression sample calculates the average of values in array.

**Program:**

static void Sample\_Average\_Lambda()

{

int[] numbers = { 10, 10, 11, 11 };

var result = numbers.Average();

Debug.WriteLine("Average is:");

Debug.WriteLine(result);

}

**Output:**

Average is:

10.5

**1.4 Count**

Returns the number of elements in a collection.

This Lambda Expression sample counts names in array.

**Program:**

static void Sample\_Count\_Lambda()

{

string[] names = { "Peter", "John", "Kathlyn", "Allen", "Tim" };

var result = names.Count();

Debug.WriteLine("Counting names gives:");

Debug.WriteLine(result);

}

**Output:**

Counting names gives:

5

**1.5 LongCount**

Returns the number of elements in collections larger than Int32.MaxValue.

This Lambda Expression sample counts elements in array larger than Int32.MaxValue.

**Program:**

/ Use LongCount() when you expect the result to be greater than Int32.MaxValue()

static void Sample\_LongCount\_Lambda()

{

// Create array which is 5 elements larger than Int32.MaxValue

var largeArr = Enumerable.Range(0, Int32.MaxValue).Concat(Enumerable.Range(0, 5));

Int64 result = largeArr.LongCount();

Debug.WriteLine("Counting largeArr elements:");

Debug.WriteLine(result);

}

**Output:**

Counting largeArr elements:

2147483652

**1.6 Max**

Finds the largest value in a collection.

This Lambda Expression sample finds the highest number in array.

**Program:**

static void Sample\_Max\_Lambda()

{

int[] numbers = { 2, 8, 5, 6, 1 };

var result = numbers.Max();

Debug.WriteLine("Highest number is:");

Debug.WriteLine(result);

}

**Output:**

Highest number is:

8

**1.7 Min**

Finds the smallest value in a collection.

This Lambda Expression sample finds the lowest number in array.

**Program:**

static void Sample\_Min\_Lambda()

{

int[] numbers = { 6, 9, 3, 7, 5 };

var result = numbers.Min();

Debug.WriteLine("Lowest number is:");

Debug.WriteLine(result);

}

**Output:**

Lowest number is:

3

**1.8 Sum**

Calculates the total for a numeric collections.

This Lambda Expression sample sums three numbers in array.

**Program:**

static void Sample\_Sum\_Lambda()

{

int[] numbers = { 2, 5, 10 };

var result = numbers.Sum();

Debug.WriteLine("Summing the numbers yields:");

Debug.WriteLine(result);

}

**Output:**

Summing the numbers yields:

17

**2.Conversion**

**2.1 AsEnumerable**

Casts a collection to IEnumerable of same type.

This Lambda Expression sample casts array of strings to its corresponding IEnumerable.

**Program:**

static void Sample\_AsEnumerable\_Lambda()

{

string[] names = { "John", "Suzy", "Dave" };

var result = names.AsEnumerable();

Debug.WriteLine("Iterating over IEnumerable collection:");

foreach (var name in result)

Debug.WriteLine(name);

}

**Output:**

Iterating over IEnumerable collection:

John

Suzy

Dave

**2.2 Cast**

Casts a collection to a specified type.

This Lambda Expression sample casts list of strings to a simple string array.

**Program:**

static void Sample\_Cast\_Lambda()

{

List<string> vegetables = new List<string> { "Cucumber", "Tomato", "Broccoli" };

var result = vegetables.Cast<string>();

Debug.WriteLine("List of vegetables casted to a simple string array:");

foreach (string vegetable in result)

Debug.WriteLine(vegetable);

}

**Output:**

List of vegetables casted to a simple string array:

Cucumber

Tomato

Broccoli

**2.3 OfType**

Filters elements of specified type in a collection.

This Lambda Expression sample takes only objects of type string.

**Program:**

static void Sample\_OfType\_Lambda()

{

object[] objects = { "Thomas", 31, 5.02, null, "Joey" };

var result = objects.OfType<string>();

Debug.WriteLine("Objects being of type string have the values:");

foreach (string str in result)

Debug.WriteLine(str);

}

**Output**:

Objects being of type string have the values:

Thomas

Joey

**2.4 ToArray**

Converts type to a new array.

This Lambda Expression sample converts one array to another.

**Program:**

static void Sample\_ToArray\_Lambda()

{

int[] numbers = { 1, 2, 3, 4 };

var result = numbers.ToArray();

Debug.WriteLine("New array contains identical values:");

foreach (int number in result)

Debug.WriteLine(number);

}

**Output:**

New array contains identical values:

1

2

3

4

**2.5 ToDictionary**(simple)

Converts collection into a Dictionary with Key and Value.

This Lambda Expression sample inserts custom values into a dictionary.

**Program:**

class English2German

{

public string EnglishSalute { get; set; }

public string GermanSalute { get; set; }

}

static void Sample\_ToDictionary\_Lambda\_Simple()

{

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);

Debug.WriteLine("Values inserted into dictionary:");

foreach (KeyValuePair<string, string> dic in result)

Debug.WriteLine(String.Format("English salute {0} is {1} in German", dic.Key, dic.Value));

}

**Output:**

Values put into dictionary:

English salute Good morning is Guten Morgen in German

English salute Good day is Guten Tag in German

English salute Good evening is Guten Abend in German

## **2.6 ToDictionary** (cond.)

Converts collection into a Dictionary with Key and Value.

This Lambda Expression sample uses ToDictionary to make a new array based on condition.

**Program:**

static void Sample\_ToDictionary\_Lambda\_Conditional()

{

int[] numbers = { 1, 2, 3, 4 };

var result = numbers.ToDictionary(k => k, v => (v % 2) == 1 ? "Odd" : "Even");

Debug.WriteLine("Numbers labeled Odd and Even in dictionary:");

foreach (var dic in result)

Debug.WriteLine("Value {0} is {1}", dic.Key, dic.Value);

}

**Output:**

Numbers labeled Odd and Even in dictionary:

Value 1 is Odd

Value 2 is Even

Value 3 is Odd

Value 4 is Even

## **2.7 ToList**

Converts collection into a List.

This Lambda Expression sample converts string array to List of strings.

**Program:**

static void Sample\_ToList\_Lambda()

{

string[] names = { "Brenda", "Carl", "Finn" };

List<string> result = names.ToList();

Debug.WriteLine(String.Format("names is of type: {0}", names.GetType().Name));

Debug.WriteLine(String.Format("result is of type: {0}", result.GetType().Name));

}

**Output:**

names is of type: String[]

result is of type: List`1

## **2.8 ToLookup**

Converts a collection into a Lookup, grouped by key.

This Lambda Expression sample puts array elements into a one-to-many Lookup, where key equals element length.

**Program:**

static void Sample\_ToLookup\_Lambda()

{

string[] words = { "one", "two", "three", "four", "five", "six", "seven" };

var result = words.ToLookup(w => w.Length);

for (int i = 1; i <= 5; i++)

{

Debug.WriteLine(String.Format("Elements with a length of {0}:", i));

foreach (string word in result[i])

Debug.WriteLine(word);

}

}

**Output:**

Elements with a length of 1:

Elements with a length of 2:

Elements with a length of 3:

one

two

six

Elements with a length of 4:

four

five

Elements with a length of 5:

three

seven

## **3.Element**

## **3.1 ElementAt**

Retrieves element from a collection at specified (zero-based) index.

This Lambda Expression sample retrieves second element from an array.

**Program:**

static void Sample\_ElementAt\_Lambda()

{

string[] words = { "One", "Two", "Three" };

var result = words.ElementAt(1);

Debug.WriteLine("Element at index 1 in the array is:");

Debug.WriteLine(result);

}

**Output:**

Element at index 1 in the array is:

Two

## **3.2 ElementAtOrDefault**

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

This Lambda Expression sample retrieves elements at index 1 and 10 from array, and because index 10 is out-of-range, it gets default value (null).

**Program:**

static void Sample\_ElementAtOrDefault\_Lambda()

{

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);

}

**Output:**

Element at index 1 in the array contains:

Green

Element at index 10 in the array does not exist:

True

## **3.3 First** (simple)

Retrieves first element from a collection. Throws exception if collection is empty.

This Lambda Expression sample retrieves first element from an array.

**Program:**

// Note: Operator First will throw an exception, if there is not at least one element in the sequence.

static void Sample\_First\_Lambda\_Simple()

{

string[] fruits = { "Banana", "Apple", "Orange" };

var result = fruits.First();

Debug.WriteLine("First element in the array is:");

Debug.WriteLine(result);

}

**Output:**

First element in the array is:

Banana

## **3.4 First** (conditional)

Retrieves first element from a collection. Throws exception if collection is empty.

This Lambda Expression sample takes first element from collection which is 6 characters long.

**Program:**

static void Sample\_First\_Lambda\_Conditional()

{

string[] countries = { "Denmark", "Sweden", "Norway" };

var result = countries.First(c => c.Length == 6);

Debug.WriteLine("First element with a length of 6 characters:");

Debug.WriteLine(result);

}

**Output:**

First element with a length of 6 characters:

Sweden

## **3.5 FirstOrDefault**

Retrieves first element from a collection, or default value if collection is empty.

This Lambda Expression sample retrieves first element from "countries" array, but from "empty" array it gets default value (null).

**Program:**

// Note: While First() will throw an exception if array...

// ...is empty, FirstOrDefault gracefully returns null.

static void Sample\_FirstOrDefault\_Lambda()

{

string[] countries = { "Denmark", "Sweden", "Norway" };

string[] empty = { };

var result = countries.FirstOrDefault();

var resultEmpty = empty.FirstOrDefault();

Debug.WriteLine("First element in the countries array contains:");

Debug.WriteLine(result);

Debug.WriteLine("First element in the empty array does not exist:");

Debug.WriteLine(resultEmpty == null);

}

**Output:**

First element in the countries array contains:

Denmark

First element in the empty array does not exist:

True

## **3.6 Last**

Retrieves last element from a collection. Throws exception if collection is empty.

This Lambda Expression sample retrieves last element from array.

**Program:**

static void Sample\_Last\_Lambda()

{

int[] numbers = { 7, 3, 5 };

var result = numbers.Last();

Debug.WriteLine("Last number in array is:");

Debug.WriteLine(result);

}

**Output:**

Last number in array is:

5

## **3.7 LastOrDefault** (simple)

Retrieves last element from a collection, or default value if collection is empty.

This Lambda Expression sample retrieves last element from words array, but from empty array it gets default value (null).

**Program:**

// Note: While Last will throw an exception if array...

// ...is empty, LastOrDefault gracefully returns null.

static void Sample\_LastOrDefault\_Simple()

{

string[] words = { "one", "two", "three" };

string[] empty = { };

var result = words.LastOrDefault();

var resultEmpty = empty.LastOrDefault();

Debug.WriteLine("Last element in the words array contains:");

Debug.WriteLine(result);

Debug.WriteLine("Last element in the empty array does not exist:");

Debug.WriteLine(resultEmpty == null);

}

**Output:**

Last element in the words array contains:

three

Last element in the empty array does not exist:

True

## **3.8 LastOrDefault** (cond.)

Retrieves last element from a collection, or default value if collection is empty.

This Lambda Expression sample retrieves last element from "words" array having a length of 3 and 2 respectively. As no elements have a length of 2, "resultNoMatch" array gets default value (null).

**Program:**

// Note: While Last will throw an exception if array...

// ...is empty, LastOrDefault gracefully returns null.

static void Sample\_LastOrDefault\_Conditional()

{

string[] words = { "one", "two", "three" };

var result = words.LastOrDefault(w => w.Length == 3);

var resultNoMatch = words.LastOrDefault(w => w.Length == 2);

Debug.WriteLine("Last element in the words array having a length of 3:");

Debug.WriteLine(result);

Debug.WriteLine("Last element in the empty array having a length of 2 does not exist:");

Debug.WriteLine(resultNoMatch == null);

}

**Output:**

Last element in the words array having a length of 3:

two

Last element in the empty array having a length of 2 does not exist:

True

## **3.9 Single**

Retrieves only element in a collection. Throws exception if not exactly one element in collection.

This Lambda Expression sample retrieves a single element from each array, but from arrays with not exactly one element it throws exception.

**Program:**

// Note: Single will throw an Exception, if there is not exactly one element in the array.

static void Sample\_Single\_Lambda()

{

string[] names1 = { "Peter" };

string[] names3 = { "Peter", "Joe", "Wilma" };

string[] empty = { };

var result1 = names1.Single();

Debug.WriteLine("The only name in the array is:");

Debug.WriteLine(result1);

try

{

// This will throw an exception because array contains no elements

var resultEmpty = empty.Single();

}

catch (Exception e)

{

Debug.WriteLine(e.Message);

}

try

{

// This will throw an exception as well because array contains more than one element

var result3 = names3.Single();

}

catch (Exception e)

{

Debug.WriteLine(e.Message);

}

}

**Output:**

The only name in the array is:

Peter

A first chance exception of type 'System.InvalidOperationException' occurred in System.Core.dll

Sequence contains no elements

A first chance exception of type 'System.InvalidOperationException' occurred in System.Core.dll

Sequence contains more than one element

## **3.10 SingleOrDefault**

Retrieves only element in a collection, or default value if collection is empty.

This Lambda Expression sample retrives a single element from array, but from arrays with not exactly one element it gets default value (null).

**Program:**

// Note: SingleOrDefault retrieves null value if array is empty, and...

// ...throws an exception if array contains more than one element.

static void Sample\_SingleOrDefault\_Lambda()

{

string[] names1 = { "Peter" };

string[] names3 = { "Peter", "Joe", "Wilma" };

string[] empty = { };

var result1 = names1.SingleOrDefault();

var resultEmpty = empty.SingleOrDefault();

Debug.WriteLine("The only name in the array is:");

Debug.WriteLine(result1);

Debug.WriteLine("As array is empty, SingleOrDefault yields null:");

Debug.WriteLine(resultEmpty == null);

try

{

// This will throw an exception as well because array contains more than one element

var result3 = names3.SingleOrDefault();

}

catch (Exception e)

{

Debug.WriteLine(e.Message);

}

}

**Output:**

The only name in the array is:

Peter

As array is empty, SingleOrDefault yields null:

True

A first chance exception of type 'System.InvalidOperationException' occurred in System.Core.dll

Sequence contains more than one element

**4.Generation**

## **4.1 DefaultIfEmpty** (simple)

If a collection is empty, its default value is returned. Default value depends on collection type.

This Lambda Expression sample returns default values for each of the empty collections, while "words" array is returned as is.

**Program :**

static void Sample\_DefaultIfEmpty\_Lambda\_Simple()

{

string[] emptyStr = { };

int[] emptyInt = { };

string[] words = { "one", "two", "three" };

var resultStr = emptyStr.DefaultIfEmpty(); // Default is null for string

var resultInt = emptyInt.DefaultIfEmpty(); // Default is 0 for int

var resultWords = words.DefaultIfEmpty(); // Not empty, so words array is just copied

Debug.WriteLine("resultStr has one element with a value of null:");

Debug.WriteLine(resultStr.Count() == 1 && resultStr.First() == null);

Debug.WriteLine("resultInt has one element with a value of 0:");

Debug.WriteLine(resultInt.Count() == 1 && resultInt.First() == 0);

Debug.WriteLine("resultWords has same content as words array:");

Debug.WriteLine(resultWords.SequenceEqual(words));

}

**Output:**

resultStr has one element with a value of null:

True

resultInt has one element with a value of 0:

True

resultWords has same content as words array:

True

## **4.2 DefaultIfEmpty** (default value)

**Program:**

static void Sample\_DefaultIfEmpty\_Lambda\_DefaultValue()

{

int[] empty = { };

var result = empty.DefaultIfEmpty(5);

Debug.WriteLine("result contains one element with a value of 5:");

Debug.WriteLine(result.Count() == 1 && result.First() == 5);

}

**Output:**

result contains one element with a value of 5:

True

## **4.3 Empty**

Generates an empty collection (with no elements).

This Lambda Expression sample creates an empty sequence of type string.

**Program:**

static void Sample\_Empty\_Lambda()

{

var empty = Enumerable.Empty<string>();

// To make sequence into an array use empty.ToArray()

Debug.WriteLine("Sequence is empty:");

Debug.WriteLine(empty.Count() == 0);

}

**Output:**

Sequence is empty:

True

## **4.4 Range**

Generates sequence of numeric values.

This Lambda Expression sample generates sequence within a range from 0-10.

**Program:**

static void Sample\_Range\_Lambda()

{

var result = Enumerable.Range(0, 10);

Debug.WriteLine("Counting from 0 to 9:");

foreach (int number in result)

Debug.WriteLine(number);

}

**Output:**

Counting from 0 to 9:

0

1

2

3

4

5

6

7

8

9

## **4.5 Repeat**

Creates a collection of repeated elements, where first argument is value to repeat, and second argument is number of times to repeat.

This Lambda Expression sample repeats the word "Banana" 5 times.

**Program :**

static void Sample\_Repeat\_Lambda()

{

string word = "Banana";

var result = Enumerable.Repeat(word, 5);

Debug.WriteLine("String is repeated 5 times:");

foreach (string str in result)

Debug.WriteLine(str);

}

**Output:**

String is repeated 5 times:

Banana

Banana

Banana

Banana

Banana

**5.Grouping**

## **5.1 GroupBy**

Projects elements of a collection into groups by key.

This Lambda Expression sample splits array of numbers into two groups: one which is divisible by 10, and one which is not.­

**Program :**

static void Sample\_GroupBy\_Lambda()

{

int[] numbers = { 10, 15, 20, 25, 30, 35 };

var result = numbers.GroupBy(n => (n % 10 == 0));

Debug.WriteLine("GroupBy has created two groups:");

foreach (IGrouping<bool, int> group in result)

{

if (group.Key == true)

Debug.WriteLine("Divisible by 10");

else

Debug.WriteLine("Not Divisible by 10");

foreach (int number in group)

Debug.WriteLine(number);

}

}

**Output:**

GroupBy has created two groups:

Divisible by 10

10

20

30

Not Divisible by 10

15

25

35

**6.Join**

## **6.1 GroupJoin** (left outer join)

Groups two collections by a common key value, and is similar to left outer join in SQL.

This Lambda Expression sample groups collection "persons" with collection "languages" by a common key.

**Program :**

class Language

{

public int Id { get; set; }

public string Name { get; set; }

}

class Person

{

public int LanguageId { get; set; }

public string FirstName { get; set; }

}

static void Sample\_GroupJoin\_Lambda()

{

Language[] languages = new Language[]

{

new Language {Id = 1, Name = "English"},

new Language {Id = 2, Name = "Russian"}

};

Person[] persons = new Person[]

{

new Person { LanguageId = 1, FirstName = "Tom" },

new Person { LanguageId = 1, FirstName = "Sandy" },

new Person { LanguageId = 2, FirstName = "Vladimir" },

new Person { LanguageId = 2, FirstName = "Mikhail" },

};

var result = languages.GroupJoin(persons, lang => lang.Id, pers => pers.LanguageId,

(lang, ps) => new { Key = lang.Name, Persons = ps });

Debug.WriteLine("Group-joined list of people speaking either English or Russian:");

foreach (var language in result)

{

Debug.WriteLine(String.Format("Persons speaking {0}:", language.Key));

foreach (var person in language.Persons)

{

Debug.WriteLine(person.FirstName);

}

}

}

**Output:**

Group-joined list of people speaking either English or Russian:

Persons speaking English:

Tom

Sandy

Persons speaking Russian:

Vladimir

Mikhail

## **6.2 Join** (inner join)

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

This Lambda Expression sample joins two arrays where elements match in both.

**Program :**

static void Sample\_Join\_Lambda()

{

string[] warmCountries = { "Turkey", "Italy", "Spain", "Saudi Arabia", "Etiobia" };

string[] europeanCountries = { "Denmark", "Germany", "Italy", "Portugal", "Spain" };

var result = warmCountries.Join(europeanCountries, warm => warm, european => european, (warm, european) => warm);

Debug.WriteLine("Joined countries which are both warm and Europan:");

foreach (var country in result) // Note: result is an anomymous type, thus must use a var to iterate.

Debug.WriteLine(country);

}

**Output:**

Joined countries which are both warm and European:

Italy

Spain

**7.Ordering**

## **7.1 OrderBy** (simple - numbers)

Sorts a collection in ascending order.

This Lambda Expression sample sorts array of numbers in ascending order.

**Program :**

static void Sample\_OrderBy\_Lambda\_Numbers()

{

int[] numbers = { 7, 9, 5 };

var result = numbers.OrderBy(n => n);

Debug.WriteLine("Ordered list of numbers:");

foreach (int number in result)

Debug.WriteLine(number);

}

**Output:**

Ordered list of numbers:

5

7

9

## **7.2 OrderBy** (simple - dates)

Sorts a collection in ascending order.

This Lambda Expression sample sorts array of dates in ascending order.

**Program :**

static void Sample\_OrderBy\_Lambda\_Dates()

{

var dates = new DateTime[] {

new DateTime(2015, 2, 15),

new DateTime(2015, 3, 25),

new DateTime(2015, 1, 5)

};

var result = dates.OrderBy(d => d);

Debug.WriteLine("Ordered list of dates:");

foreach (DateTime dt in result)

Debug.WriteLine(dt.ToString("yyyy/MM/dd"));

}

**Output:**

Ordered list of dates:

2015-01-05

2015-02-15

2015-03-25

## **7.3 OrderBy** (simple - objects)

Sorts a collection in ascending order.

This Lambda Expression sample sorts array of cars by "HorsePower", in ascending order.

**Program :**

class Car

{

public string Name { get; set; }

public int HorsePower { get; set; }

}

static void Sample\_OrderBy\_Lambda\_Objects()

{

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 = cars.OrderBy(c => c.HorsePower);

Debug.WriteLine("Ordered list of cars by horsepower:");

foreach (Car car in result)

Debug.WriteLine(String.Format("{0}: {1} horses", car.Name, car.HorsePower));

}

**Output:**

Ordered list of cars by horsepower:

Economy Car: 75 horses

Family Car: 145 horses

Super Car: 215 horses

## **7.4 OrderByDescending**

Sorts a collection in descending order.

This Lambda Expression sample sorts array of names in descending order.

**Program :**

static void Sample\_OrderByDescending\_Lambda()

{

string[] names = { "Ned", "Ben", "Susan" };

var result = names.OrderByDescending(n => n);

Debug.WriteLine("Descending ordered list of names:");

foreach (string name in result)

Debug.WriteLine(name);

}

**Output:**

Descending ordered list of names:

Susan

Ned

Ben

## **7.5 Reverse**

Reverses elements in a collection.

This Lambda Expression sample reverts characters in array.

**Program :**

static void Sample\_Reverse\_Lambda()

{

char[] characters = { 's', 'a', 'm', 'p', 'l', 'e' };

var result = characters.Reverse();

Debug.WriteLine("Characters in reverse order:");

foreach (char character in result)

Debug.WriteLine(character);

}

**Output:**

Characters in reverse order:

e

l

p

m

a

s

## **7.6 ThenBy**

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

This Lambda Expression sample first sorts array by string length of city capital, and then by alphabet.

**Program :**

static void Sample\_ThenBy\_Lambda()

{

string[] capitals = { "Berlin", "Paris", "Madrid", "Tokyo", "London",

"Athens", "Beijing", "Seoul" };

var result = capitals.OrderBy(c => c.Length).ThenBy(c => c);

Debug.WriteLine("Ordered list of capitals, first by length and then alphabetical:");

foreach (string capital in result)

Debug.WriteLine(capital);

}

**Output:**

Ordered list of capitals, first by length and then alphabetical:

Paris

Seoul

Tokyo

Athens

Berlin

London

Madrid

Beijing

## **7.7 ThenByDescending**

Use after earlier sorting, to further sort a collection in descending order.

This Lambda Expression sample first orders a list of dates by year descending, and then by month descending.

**Program :**

static void Sample\_ThenByDescending\_Lambda()

{

var dates = new DateTime[] {

new DateTime(2015, 3, 1),

new DateTime(2014, 7, 1),

new DateTime(2013, 5, 1),

new DateTime(2015, 1, 1),

new DateTime(2015, 7, 1)

};

var result = dates.OrderByDescending(d => d.Year).ThenByDescending(d => d.Month);

Debug.WriteLine("List of dates first ordered by year descending, and then by month descending:");

foreach (DateTime dt in result)

Debug.WriteLine(dt.ToString("yyyy/MM/dd"));

}

**Output:**

List of dates first ordered by year descending, and then by month descending:

2015-07-01

2015-03-01

2015-01-01

2014-07-01

2013-05-01

**8.Other**

## **8.1 Concat** (simple - numbers)

Concatenates (combines) two collections.

This Lambda Expression sample concatenates two arrays of numbers.

**Program :**

static void Sample\_Concat\_Lambda\_Numbers()

{

int[] numbers1 = { 1, 2, 3 };

int[] numbers2 = { 4, 5, 6 };

var result = numbers1.Concat(numbers2);

Debug.WriteLine("Concatenating numbers1 and numbers2 gives:");

foreach (int number in result)

Debug.WriteLine(number);

}

**Output:**

Concatenating numbers1 and numbers2 gives:

1

2

3

4

5

6

## **8.2 Concat** (simple - strings)

Concatenates (combines) two collections.

This Lambda Expression sample concatenates two arrays of strings.

**Program :**

static void Sample\_Concat\_Lambda\_Strings()

{

string[] vegetables = { "Tomato", "Cucumber", "Carrot" };

string[] fruits = { "Apples", "Grapes", "Banana" };

var result = vegetables.Concat(fruits);

Debug.WriteLine("Concatinating vegetables and fruits gives:");

foreach (string piece in result)

Debug.WriteLine(piece);

}

**Output:**

Concatinating vegetables and fruits gives:

Tomato

Cucumber

Carrot

Apples

Grapes

Banana

## **8.3 SequenceEqual**

Checks whether two collections are equal. Use StringComparer.OrdinalIgnoreCase parameter to ignore case.

This Lambda Expression sample shows different methods to test for array equality.

**Program :**

static void Sample\_SequenceEqual\_Lambda()

{

string[] words = { "one", "two", "three" };

string[] wordsSame = { "one", "two", "three" };

string[] wordsOrder = { "two", "three", "one" };

string[] wordsCase = { "one", "TWO", "three" };

var resultSame = words.SequenceEqual(wordsSame);

var resultOrder = words.SequenceEqual(wordsOrder);

var resultCase = words.SequenceEqual(wordsCase);

var resultCaseIgnored = words.SequenceEqual(wordsCase, StringComparer.OrdinalIgnoreCase);

Debug.WriteLine("SequenceEqual on two identical arrays:");

Debug.WriteLine(resultSame);

Debug.WriteLine("SequenceEqual on two differently ordered but otherwise identical arrays:");

Debug.WriteLine(resultOrder);

Debug.WriteLine("SequenceEqual on two differently cased but otherwise identical arrays:");

Debug.WriteLine(resultCase);

Debug.WriteLine("SequenceEqual on two differently cased but otherwise identical arrays, where case is ignored:");

Debug.WriteLine(resultCaseIgnored);

}

**Output:**

SequenceEqual on two identical arrays:

True

SequenceEqual on two differently ordered but otherwise identical arrays:

False

SequenceEqual on two differently cased but otherwise identical arrays:

False

SequenceEqual on two differently cased but otherwise identical arrays, where case is ignored:

True

## **8.4 Zip**

Processes two collections in parallel with func instance, and combines result into a new collection.

This Lambda Expression sample uses Zip to process two arrays in parallel, while each processed pair is summed.

**Program :**

static void Sample\_Zip\_Lambda()

{

int[] numbers1 = { 1, 2, 3 };

int[] numbers2 = { 10, 11, 12 };

var result = numbers1.Zip(numbers2, (a, b) => (a \* b));

Debug.WriteLine("Using Zip to combine two arrays into one (1\*10, 2\*11, 3\*12):");

foreach (int number in result)

Debug.WriteLine(number);

}

**Output:**

Using Zip to combine two arrays into one (1\*10, 2\*11, 3\*12):10

22

36

**9.Partioning**

## **9.1 Skip**

Skips specified number of elements in a collection.

This Lambda Expression sample skips first 4 words in array.

**Program :**

static void Sample\_Skip\_Lambda()

{

string[] words = { "one", "two", "three", "four", "five", "six" };

var result = words.Skip(4);

Debug.WriteLine("Skips the first 4 words:");

foreach (string word in result)

Debug.WriteLine(word);

}

**Output:**

Skips the first 4 words:

five

six

## **9.2 SkipWhile**

Skips elements in a collection while specified condition is met.

This Lambda Expression sample skips words in array, as long as word has length of 3 characters.

**Program :**

static void Sample\_SkipWhile\_Lambda()

{

string[] words = { "one", "two", "three", "four", "five", "six" };

var result = words.SkipWhile(w => w.Length == 3);

Debug.WriteLine("Skips words while the condition is met:");

foreach (string word in result)

Debug.WriteLine(word);

}

**Output:**

Skips words while the condition is met:

three

four

five

six

## **9.3 Take**

Takes specified number of elements in a collection, starting from first element.

This Lambda Expression sample takes only 5 first numbers in array.

**Program :**

static void Sample\_Take\_Lambda()

{

int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

var result = numbers.Take(5);

Debug.WriteLine("Takes the first 5 numbers only:");

foreach (int number in result)

Debug.WriteLine(number);

}

**Output:**

Takes the first 5 numbers only:1

2

3

4

5

## **9.4 TakeWhile**

Takes elements in a collection while specified condition is met, starting from first element.

This Lambda Expression sample takes numbers from array, while number is less than 5.

**Program :**

static void Sample\_TakeWhile\_Lambda()

{

int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

var result = numbers.TakeWhile(n => n < 5);

Debug.WriteLine("Takes numbers one by one, and stops when condition is no longer met:");

foreach (int number in result)

Debug.WriteLine(number);

}

**Output:**

Takes numbers one by one, and stops when condition is no longer met:

1

2

3

4

**10.Projection**

## **10.1 Select** (simple)

Selects, projects and transforms elements in a collection.

This Lambda Expression sample selects and rounds down each number in array.

**Program :**

static void Sample\_Select\_Lambda\_Simple()

{

decimal[] numbers = { 3.4M, 8.33M, 5.225M };

var result = numbers.Select(n => Math.Floor(n));

Debug.WriteLine("Numbers rounded down:");

foreach (int number in result)

Debug.WriteLine(number);

}

**Output:**

Numbers rounded down:

3

8

5

## **10.2 Select** (simple)

Selects, projects and transforms elements in a collection.

This Lambda Expression sample selects and rounds down each number in array.

**Program :**

static void Sample\_Select\_Lambda\_Simple()

{

decimal[] numbers = { 3.4M, 8.33M, 5.225M };

var result = numbers.Select(n => Math.Floor(n));

Debug.WriteLine("Numbers rounded down:");

foreach (int number in result)

Debug.WriteLine(number);

}

**Output:**

Numbers rounded down:

3

8

5

## **10.3 Select** (indexed)

Selects, projects and transforms elements in a collection. Can be overloaded to get element index.

This Lambda Expression sample selects word and element index from array.

**Program :**

static void Sample\_Select\_Lambda\_Indexed()

{

string[] words = { "one", "two", "three" };

var result = words.Select((w, i) => new

{

Index = i,

Value = w

});

Debug.WriteLine("Words with index and value:");

foreach (var word in result)

Debug.WriteLine(String.Format("Index {0} is {1}", word.Index, word.Value));

}

**Output:**

Words with index and value:

Index 0 is one

Index 1 is two

Index 2 is three

## **10.4 SelectMany** (cross join)

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

This Lambda Expression sample cross joins two arrays, and gets cartesian product.

**Program :**

static void Sample\_SelectMany\_Lambda()

{

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);

}

**Output:**

Selecting all values from each array, and mixing them:

Grape, 1

Grape, 2

Grape, 3

Orange, 1

Orange, 2

Orange, 3

Apple, 1

Apple, 2

Apple, 3

**11.Quantifiers**

## **11.1 All**

Checks if all elements in a collection satisfies a specified condition.

This Lambda Expression sample checks whether all names in array start with letter "B".

**Program :**

static void Sample\_All\_Lambda()

{

string[] names = { "Bob", "Ned", "Amy", "Bill" };

var result = names.All(n => n.StartsWith("B"));

Debug.WriteLine("Does all of the names start with the letter 'B':");

Debug.WriteLine(result);

}

**Output:**

Does all of the names start with the letter 'B':

False

## **11.2 Any**

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

This Lambda Expression sample checks whether any names in array start with the letter 'B'.

**Program :**

static void Sample\_Any\_Lambda()

{

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);

}

**Output:**

Does any of the names start with the letter 'B':

True

## **11.3 Contains**

Checks if any elements in a collection satisifies a specified value.

This Lambda Expression sample checks whether array of numbers contains value of 5.

**Program :**

static void Sample\_Contains\_Lambda()

{

int[] numbers = { 1, 3, 5, 7, 9 };

var result = numbers.Contains(5);

Debug.WriteLine("sequence contains the value 5:");

Debug.WriteLine(result);

}

**Output:**

Sequence contains the value 5:

True

**12.Restriction**

## **12.1 Where** (simple - numbers)

Filters elements from a collection to satisfy a specified condition.

This Lambda Expression sample finds numbers with values >= 15 and values <= 25.

**Program :**

static void Sample\_Where\_Lambda\_Numbers()

{

int[] numbers = { 5, 10, 15, 20, 25, 30 };

var result = numbers.Where(n => n >= 15 && n <= 25);

Debug.WriteLine("Numbers being >= 15 and <= 25:");

foreach (var number in result)

Debug.WriteLine(number);

}

**Output:**

Numbers being >= 15 and <= 25:

15

20

25

## **12.2 Where** (simple - objects)

Filters elements from a collection to satisfy a specified condition.

This Lambda Expression sample finds all persons who are 30 years or older.

**Program :**

class Person

{

public string Name { get; set; }

public int Age { get; set; }

}

static void Sample\_Where\_Lambda\_Objects()

{

Person[] persons = {

new Person { Name = "Mike", Age = 25 },

new Person { Name = "Joe", Age = 43 },

new Person { Name = "Nadia", Age = 31 }

};

var result = persons.Where(p => p.Age >= 30);

Debug.WriteLine("Finding persons who are 30 years old or older:");

foreach (Person person in result)

Debug.WriteLine(String.Format("{0}: {1} years old", person.Name, person.Age));

}

**Output:**

Finding persons who are 30 years old or older:

Joe: 43 years old

Nadia: 31 years old

## **12.3 Where** (indexed)

Filters elements from a collection to satisfy a specified condition. Use overloaded Index to pass index.

This Lambda Expression sample finds numbers divisible by 3 and element index >= 5.

**Program :**

static void Sample\_Where\_Lambda\_Indexed()

{

int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

var result = numbers.Where((n, i) => n % 3 == 0 && i >= 5);

Debug.WriteLine("Numbers divisible by 3 and indexed >= 5:");

foreach (var number in result)

Debug.WriteLine(number);

}

**Output:**

Numbers divisible by 3 and indexed >= 5:

6

9

## **13.Set**

## **13.1 Distinct**

Removes duplicate elements from a collection.

This Lambda Expression sample takes only distinct numbers from array.

**Program :**

static void Sample\_Distinct\_Lambda()

{

int[] numbers = { 1, 2, 2, 3, 5, 6, 6, 6, 8, 9 };

var result = numbers.Distinct();

Debug.WriteLine("Distinct removes duplicate elements:");

foreach (int number in result)

Debug.WriteLine(number);

}

**Output:**

Distinct removes duplicate elements:1

2

3

5

6

8

9

## **13.2 Except**

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

This Lambda Expression sample removes numbers from "numbers1", which exist in "numbers2".

**Program :**

static void Sample\_Except\_Lambda()

{

int[] numbers1 = { 1, 2, 3 };

int[] numbers2 = { 3, 4, 5 };

var result = numbers1.Except(numbers2);

Debug.WriteLine("Except creates a single sequence from numbers1 and removes the duplicates found in numbers2:");

foreach (int number in result)

Debug.WriteLine(number);

}

**Output:**

Except creates a single sequence from numbers1 and removes the duplicates found in numbers2:

1

2

## **13.3 Intersect**

Takes only the elements that are shared between two collections.

This Lambda Expression sample creates a new collection with values shared between the two arrays.

**Program :**

static void Sample\_Intersect\_Lambda()

{

int[] numbers1 = { 1, 2, 3 };

int[] numbers2 = { 3, 4, 5 };

var result = numbers1.Intersect(numbers2);

Debug.WriteLine("Intersect creates a single sequence with only the duplicates:");

foreach (int number in result)

Debug.WriteLine(number);

}

**Output:**

Intersect creates a single sequence with only the duplicates:

3

## **13.4 Union**

Combines two collections and removes duplicate elements.

This Lambda Expression sample removes any duplicate values between the two arrays.

**Program :**

static void Sample\_Union\_Lambda()

{

int[] numbers1 = { 1, 2, 3 };

int[] numbers2 = { 3, 4, 5 };

var result = numbers1.Union(numbers2);

Debug.WriteLine("Union creates a single sequence and eliminates the duplicates:");

foreach (int number in result)

Debug.WriteLine(number);

}

**Output:**

Union creates a single sequence and eliminates the duplicates:

1

2

3

4

5