LINQ

[Language Integrated Query]

LINQ

- Language Integrated Query
  Language Integrated = part of programming language syntax
  Query = used for querying data (of different types)

- A programming language syntax in C# and VB.NET that is used to query data from different types of sources including relational databases, objects, XML, etc.

- Two forms of LINQ operations – queries and methods.
  Query syntax is considered more readable and easier to understand.
  Method syntax uses Lambda expressions.

Why We Need LINQ

```
class Student
{
    public int StudentID { get; set; }
    public string FullName { get; set; }
    public int Age { get; set; }
}
```

```
Student[] studentArray =
{
    new Student() { StudentID = 1, FullName = "Bob", Age = 17},
    new Student() { StudentID = 2, FullName = "John", Age = 13},
    new Student() { StudentID = 3, FullName = "Susan", Age = 14},
    new Student() { StudentID = 4, FullName = "Anne", Age = 19},
    new Student() { StudentID = 5, FullName = "Grace", Age = 20},
    new Student() { StudentID = 6, FullName = "David", Age = 21},
    new Student() { StudentID = 7, FullName = "Matt", Age = 19},
    new Student() { StudentID = 8, FullName = "Peter", Age = 12},
    new Student() { StudentID = 9, FullName = "Jean", Age = 16},
    new Student() { StudentID = 10, FullName = "Francis", Age = 11}
};
```

- From an array of Student objects, we need a list of teenage students. What if we wanted this list in alphabetical order, or order of age?

Why We Need LINQ (cont.)

```
// We want a list of teenage students
foreach (Student s in studentArray)
{
    if (s.Age > 12 && s.Age < 20)
    {
        WriteLine("Student Name: {0}, Age: {1}", s.FullName, s.Age);
    }
}
```

Using loop in C#

Why We Need LINQ (cont.)

```
// We want a list of teenage students
foreach (Student s in studentArray)
{
    if (s.Age > 12 && s.Age < 20)
    {
        WriteLine("Student Name: {0}, Age: {1}", s.FullName, s.Age);
    }
}
```

Using loop in C#

```
// LINQ Equivalent of the above loop
var teenageStudents = from s in studentArray where s.Age > 12 && s.Age < 20 select s;
```

Using LINQ Query Syntax

Note: You will need a *using* directive for the System.Linq namespace.

Example

```
class Student
{
    public string First { get; set; }
    public string Last { get; set; }
    public int ID { get; set; }
    public List<int> Scores;
}
```

Auto-implemented properties

```
List<Student> students = new List<Student>
{
    new Student {First="Bob", Last="Jones", ID=111, Scores= new List<int> {97, 92, 81, 60}},
    new Student {First="Claire", Last="Simpson", ID=112, Scores= new List<int> {75, 84, 91, 39}},
    new Student {First="John", Last="Feetham", ID=113, Scores= new List<int> {88, 94, 65, 91}},
    new Student {First="Jonathan", Last="Potts", ID=114, Scores= new List<int> {97, 89, 85, 82}},
    new Student {First="Pepe", Last="Garcia", ID=115, Scores= new List<int> {35, 72, 91, 70}},
    new Student {First="Samantha", Last="Fakhouri", ID=116, Scores= new List<int> {99, 86, 90, 94}},
    new Student {First="Roger", Last="Song", ID=117, Scores= new List<int> {93, 92, 80, 87}},
    new Student {First="Hugo", Last="Garcia", ID=118, Scores= new List<int> {92, 90, 83, 78}},
    new Student {First="Richard", Last="Ammerman", ID=119, Scores= new List<int> {68, 79, 88, 92}},
    new Student {First="Kevin", Last="Adamson", ID=120, Scores= new List<int> {99, 82, 81, 79}},
    new Student {First="Jeet", Last="Singh", ID=121, Scores= new List<int> {96, 85, 91, 60}},
    new Student {First="Michael", Last="Jones", ID=122, Scores= new List<int> {94, 92, 91, 91} }
};
```

Using Object Initializer

Example (cont.)

- Produce a list of all students who scored over 90 in their first test.

```
var studentQuery = from s in students where s.Scores[0] > 90 select s;
```

Implicitly-typed variable
(Compiler determines and assigns the most appropriate type)

The query is not actually executed until you iterate through it.

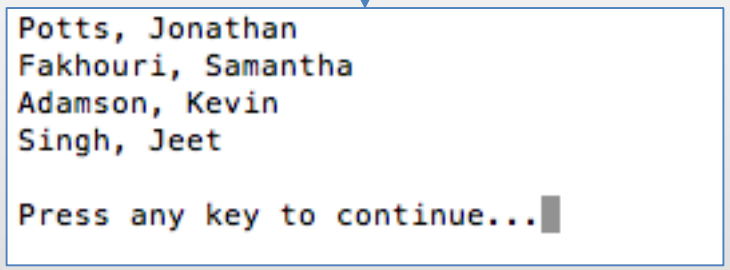
```
foreach (Student s in studentQuery)
{
    Console.WriteLine("{0}, {1}", s.Last, s.First);
}
```

```
Jones, Bob
Potts, Jonathan
Fakhouri, Samantha
Song, Roger
Garcia, Hugo
Adamson, Kevin
Singh, Jeet
Jones, Michael

Press any key to continue...
```

Example (cont.)

- Return all students who scored over 90 in their first test but less than 90 in their second test.

```
// Students who scored over 90 in their first test and less than 90 in their second test
var studentQuery2 = from s in students where s.Scores[0] > 90 && s.Scores[1] < 90 select s;

foreach (Student s in studentQuery2)
{
    Console.WriteLine("{0}, {1}", s.Last, s.First);
}
```



Sorting Results

- We can modify the results to be displayed in any order we want using the **orderby** clause.

```
var studentQuery2 = from s in students
                    where s.Scores[0] > 90 && s.Scores[1] < 90
                    orderby s.Last, s.First ascending
                    select s;
```

```
var studentQuery3 = from s in students
                    where s.Scores[0] > 90 && s.Scores[1] < 90
                    orderby s.Scores[0] + s.Scores[1] descending
                    select s;
```

Grouping Results

- A query using the **group** clause produces a sequence of groups with a key and a sequence of objects.

Key

```
var studentQueryGroup = from s in students group s by s.Last;

foreach (var studentGroup in studentQueryGroup)
{
    WriteLine(studentGroup.Key);
    foreach (Student s in studentGroup)
    {
        WriteLine("   {0}, {1}", s.Last, s.First);
    }
}
```

```
Jones
   Jones, Bob
   Jones, Michael
Simpson
   Simpson, Claire
Feetham
   Feetham, John
Potts
   Potts, Jonathan
Garcia
   Garcia, Pepe
   Garcia, Hugo
Fakhouri
   Fakhouri, Samantha
Song
   Song, Roger
Ammerman
   Ammerman, Richard
Adamson
   Adamson, Kevin
Singh
   Singh, Jeet

Press any key to continue...
```

What do we get if we change the key to s.Last[0]?

Ordering Grouped Results

- To use the **orderby** clause in a grouped result, you need an identifier to the groups created by the **group** clause. You do this using the **into** keyword.

```
var studentQueryGroup = from s in students
                        group s by s.Last[0] into studentGroup
                        orderby studentGroup.Key
                        select studentGroup;
```

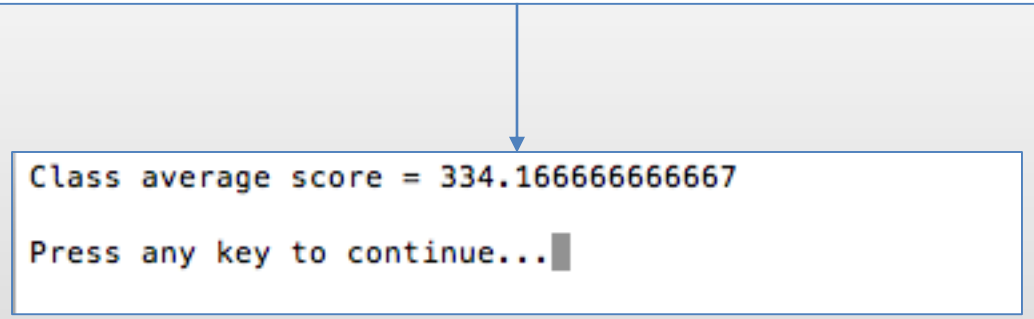
```
foreach (var studentGroup in studentQueryGroup)
{
    WriteLine(studentGroup.Key);
    foreach (Student s in studentGroup)
    {
        WriteLine("   {0}, {1}", s.Last, s.First);
    }
}
```

```
A
   Ammerman, Richard
   Adamson, Kevin
F
   Feetham, John
   Fakhouri, Samantha
G
   Garcia, Pepe
   Garcia, Hugo
J
   Jones, Bob
   Jones, Michael
P
   Potts, Jonathan
S
   Simpson, Claire
   Song, Roger
   Singh, Jeet

Press any key to continue...
```

Performing Calculations

- We can use the let keyword to introduce an identifier for any expression, e.g. to perform calculations such as adding up items and using mathematical formulas.

```
var studentQuery6 = from s in students
let totalScore = s.Scores[0] + s.Scores[1] + s.Scores[2] + s.Scores[3]
select totalScore;

double averageScore = studentQuery6.Average();

Console.WriteLine("Class average score = {0}", averageScore);
```



Joins

- You can use LINQ to perform joins between collections of objects.

```
Person louis = new Person { FirstName = "Louis" };
Person sophie = new Person { FirstName = "Sophie"};
Person luc = new Person { FirstName = "Luc"};

Pet duplo = new Pet { Name = "Duplo", Owner = sophie };
Pet kuro = new Pet { Name = "Kuro", Owner = sophie };
Pet tara = new Pet { Name = "Tara", Owner = louis };
Pet gypsy = new Pet { Name = "Gypsy", Owner = louis };
Pet clover = new Pet { Name = "Clover", Owner = sophie };

List<Person> people = new List<Person> { louis, sophie, luc };
List<Pet> pets = new List<Pet> { duplo, kuro, tara, gypsy, clover };

var queryPairs = from p in people
            join pet in pets on p equals pet.Owner
            select new { OwnerName = p.FirstName, PetName = pet.Name };

foreach (var ownerAndPet in queryPairs)
{
    Console.WriteLine($"\"{ownerAndPet.PetName}\" is owned by {ownerAndPet.OwnerName}");
}
```

Inner Join

Create a collection.
Each element contains
owner name and pet name.

```
"Tara" is owned by Louis
"Gypsy" is owned by Louis
"Duplo" is owned by Sophie
"Kuro" is owned by Sophie
"Clover" is owned by Sophie

Press any key to continue...
```

Note: Luc does not appear on this list. Why not?

Joins (cont.)

- Using our previous example of students, we also have a list of staff. We want to find out how many of our staff are also students.

```
List<Staff> staffmembers = new List<Staff>
{
    new Staff {First="Jeet", Last="Singh", ID=900},
    new Staff {First="Richard", Last="Potter", ID=901},
    new Staff {First="Terry", Last="Woodward", ID=902},
};
```

```
var queryStaffAndStudent = from s in staffmembers
                           join t in students
                           on new { s.First, s.Last }
                           equals new { t.First, t.Last }
                           select s.First + " " + s.Last;

Console.WriteLine("Staff who are also students:");
foreach (string name in queryStaffAndStudent)
{
    WriteLine(name);

}
```

```
Staff who are also students:
Jeet Singh

Press any key to continue...
```

Query v Method Syntax

Query syntax

Method syntax

```
// Students who are teenagers

// Query syntax
var studentTeens = from s in students where s.Age > 12 && s.Age < 20 select s;

// Method syntax
var studentTeens2 = students.Where(s => s.Age > 12 && s.Age < 20).ToList<Student>();
```

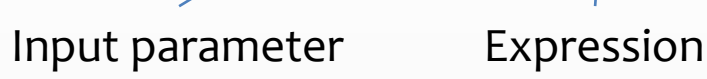
```
int[] numbers = new int[10] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

var EvenNumbers = from num in numbers where (num % 2) == 0 select num;

var EvenNumbers2 = numbers.Where(num => num % 2 == 0).OrderBy(n => n);
```

Method syntax uses Lambda expressions…

- Query syntax is considered more readable (recommended).
- The compiler converts query syntax to method syntax at compile time.

Lambda Expressions*

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

var EvenNumbers = from num in numbers where (num % 2) == 0 select num;

var EvenNumbers2 = numbers.Where(num => num % 2 == 0).OrderBy(n => n);
```



- In this example, we can say that *num* is the input parameter which goes to anonymous function, and this function returns true if the input is even.

More details will be provided later on.

Using LINQ With XML

```
using System.Linq;
using System.Xml.Linq;
```

Need this directive!

```
XDocument document = XDocument.Load("..\\..\\books.xml");

var books = from b in document.Descendants("book")
            select new { Author = b.Element("author").Value, Title = b.Element("title").Value };

foreach (var bk in books)
    WriteLine("{0} ({1})", bk.Title, bk.Author);
```

XML

```
<book id="bk101">
   <author id="1">Gambardella, Matthew</author>
   <title>XML Developer's Guide</title>
   <genre>Computer</genre>
   <price>44.95</price>
   <publish_date>2000-10-01</publish_date>
    <description> An in-depth look at creating applications with XML.</description>
</book>
```

```
XML Developer's Guide (Gambardella, Matthew)
Midnight Rain (Ralls, Kim)
Maeve Ascendant (Corets, Eva)
Oberon's Legacy (Corets, Eva)
The Sundered Grail (Corets, Eva)

Press any key to continue...
```

[Taken from http://www.c-sharpcorner.com/UploadFile/dhananjaycoder/reading-xml-file-through-linq-a-few-tips/]

Exercise

- Using the class definitions and data provided, write a C# code that creates and uses LINQ queries to produce the following:
  - Students who are under 18 years of age (in order of age)
  - Students who are teenagers (alphabetical order by last name)
  - Students who scored 80 or more in their last test (order by score descending)
  - Students who scored over 320 marks in total (across all their tests)
  - Students who scored at least 60 in all of their tests
  - Students grouped by first letter of their last name
  - Average score of each test
  - Students who are also teachers
  - Courses of a duration of 15 weeks
  - Courses held in the Winter semester (order by duration)
  - Courses grouped by semester

Resources

LINQ

- https://msdn.microsoft.com/en-us/library/bb397906(v=vs.120).aspx
- https://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b
  (You need to click on the subheadings and then you can see the examples)

LINQ With XML

- http://www.dotnetcurry.com/linq/564/linq-to-xml-tutorials-examples
  (Section 1 only - Read XML and Traverse the XML Document using LINQ to XML)