LINQ

Mingsheng Hong
CS 215, Spring 2008

Review

- C# 3.0 language features
  - Implicitly typed variables
  - Automatic properties
  - Initializers
  - Anonymous types
  - Lambda expressions
  - Extension methods

Type Inference & 3.0 Features

- `delegate R Func<A,R>(A arg);`

-
  ```
  //extension method defined in some static class
  public static IEnumerable<S> Select<T,S>(
                  this IEnumerable<T> source,
                  Func<T,S> selector) {
      foreach (T element in source)
            yield return selector(element);
  }
  ```

-
  ```
  var customers = new[] {
      new {Name = "Jack", ID = 8},
      new {Name = "Kate", ID = 15}};
  foreach (var n in customers.Select(c => c.Name)) {
      Console.WriteLine(n);
  }
  ```

Components in LINQ

- LINQ to Object
- LINQ to XML
- LINQ to Dataset
- LINQ to SQL
- LINQ to Entities

Overview of LINQ to Object

- A LINQ query is a composition of operators
  - selection predicate, ordering criteria, output specification, …
  - e.g. retrieve (video game) items with more than 10 letters, sorted alphabetically
- *Sequence* – Input/output data type
  - a collection implementing `IEnumerable<T>`
  - given iterator, can be viewed as a sequence

Two Styles of LINQ Queries

- E.g. Retrieve items with more than 10 letters
  ```
  string[] videoGames = {"Morrowind", "BioShock",
      "Half Life 2: Episode 1", "The Darkness"};

  //1. query expression
  IEnumerable<string> subset = from g in videoGames
      where g.Length > 10 orderby g select g;

  //2. method-based
  IEnumerable<string> subset2 = videoGames.Where(g =>
  g.Length > 10).OrderBy(g => g).Select(g => g);
  ```
- How many methods in `IEnumerable<T>`?

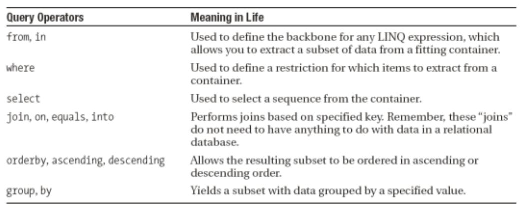
Example Extension Method

```
namespace System.Linq {
  public static class Enumerable {
      public static IEnumerable<T> Where<T>(
            this IEnumerable<T> source,
            Func<T, bool> predicate) {
            foreach (T item in source)
                  if (predicate(item))
                        yield return item;
            }
  }
}
```

Other Extension Methods

- Extension methods in `IEnumerable<T>`
  - `Take/TakeWhile`
  - `Skip/SkipWhile`
  - `Reverse`
  - `Concat`
  - `Intersect/Union/Except`
  - …

LINQ Operators



Query Operators	Meaning in Life
from, in	Used to define the backbone for any LINQ expression, which allows you to extract a subset of data from a fitting container.
where	Used to define a restriction for which items to extract from a container.
select	Used to select a sequence from the container.
join, on, equals, into	Performs joins based on specified key. Remember, these "joins" do not need to have anything to do with data in a relational database.
orderby, ascending, descending	Allows the resulting subset to be ordered in ascending or descending order.
group, by	Yields a subset with data grouped by a specified value.

A Tutorial on LINQ Operators

-
  ```
  var students = new[] {
        new {ID = 100, Name = "Tom", Major = "CS"},
        new {ID = 200, Name = "Dave", Major = "CS"},
        new {ID = 300, Name = "Jane", Major = "ECE"},
  };
  ```
- For more on data management
  - Relational databases, SQL, indexing, transaction, XML, Xquery…
  - Check out CS 330 and CS 432

from, in, select

-
  ```
  var students = new[] {
        new {ID = 100, Name = "Tom", Major = "CS"},
        new {ID = 200, Name = "Dave", Major = "CS"},
        new {ID = 300, Name = "Jane", Major = "ECE"},
  };
  ```

-
  ```
  var result1 = from s in students
                select s;
  ```

-
  ```
  var result2 = from s in students
                select new { s.ID, s.Name };
  ```

where

-
  ```
  var students = new[] {
        new {ID = 100, Name = "Tom", Major = "CS"},
        new {ID = 200, Name = "Dave", Major = "CS"},
        new {ID = 300, Name = "Jane", Major = "ECE"},
  };
  ```

-
  ```
  var result3 = from s in students
                where s.Major == "CS"
                select s;
  ```

orderby

-
  ```
  var students = new[] {
      new {ID = 100, Name = "Tom", Major = "CS"},
      new {ID = 200, Name = "Dave", Major = "CS"},
      new {ID = 300, Name = "Jane", Major = "ECE"},
  };
  ```

-
  ```
  var result4 = from s in students
                where s.Major == "CS"
                orderby s.ID ascending
                select s
  ```
- `ascending` or `descending` keywords optional

group, by

-
  ```
  var students = new[] {
      new {ID = 100, Name = "Tom", Major = "CS"},
      new {ID = 200, Name = "Dave", Major = "CS"},
      new {ID = 300, Name = "Jane", Major = "ECE"},
  };
  ```

-
  ```
  var result5 = from s in students
                group s by s.Major
                into g
                select new {
                      Major = g.Key,
                      Count = g.Count()
                };
  ```

join

-
  ```
  var students = new[] {
      new {ID = 100, Name = "Tom", Major = "CS"},
      new {ID = 200, Name = "Dave", Major = "CS"},
      new {ID = 300, Name = "Jane", Major = "ECE"},
  };
  ```

-
  ```
  var result6 = from s1 in students
                join s2 in students
                on s1.Major equals s2.Major
                select new {
                      Name1 = s1.Name,
                      Name2 = s2.Name
                };
  ```

Exercises

-
  ```
  var students = new[] {
      new {ID = 100, Name = "Tom", Major = "CS"},
      new {ID = 200, Name = "Dave", Major = "CS"},
      new {ID = 300, Name = "Jane", Major = "ECE"},
  };
  ```
- List the IDs of ECE majors
- Sort students alphabetically by name
- List all pairs of students not in the same major

Deferred Execution

-
  ```
  int[] array = { 0, 1, 2 };
  var result = from x in array
               where x % 2 == 0
               select x;
  array[0] = 3;
  foreach (var x in result) {
      Console.WriteLine(x);
  }
  ```
- The LINQ expression is note evaluated until when `result` is iterated over!
- `ToArray<T>` or `ToList<T>` to “cache” query result

Nongeneric Collections

- `OfType<T>` versus `Cast<T>`
  - Extension methods

-
  ```
  ArrayList a = new ArrayList { 0, "1", 2 };
  var a1 = a.OfType<int>();
  foreach (var x in a1) {
      Console.WriteLine(x);
  }
  ```