Review

- C# 3 language features
  - Implicitly typed variables
  - Automatic properties
  - Initializers
  - Anonymous types
  - Lambda expressions
  - Extension methods
Today's Agenda

- LINQ
- LINQ operators
LINQ

- LINQ: Language INtegrated Query
- Allows native data querying in .NET
- LINQ defines a set of query operators
  - Can be used to query, project, and filter data
  - Data can be in arrays, enumerables, XML, and databases
  - Querying handled by the LINQ engine
  - Results returned as a collection of in-memory objects that can be iterated on
Quick Example

- To appreciate the awesomeness of LINQ
- LinqCars.cs

as best put by Barney Stinson
Two Styles of LINQ Queries

- As an SQL-like query expression
  - `IEnumerable<Car> result = from car in cars where car.Year > 2003 select car;`

- A method based style
  - `IEnumerable<Car> result = cars.Where(car => car.Year > 2003).Select(car => car);`

- Choose what you find more comfortable
# Main LINQ Operators

<table>
<thead>
<tr>
<th>Query Operator</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>from .. in ..</td>
<td>Used to determine what is the collection of objects we are operating on, and gives a named variable to refer to each element in that collection</td>
</tr>
<tr>
<td>where</td>
<td>Specifies the condition used to match query results</td>
</tr>
<tr>
<td>select</td>
<td>Used to specify what to select from the collection</td>
</tr>
<tr>
<td>orderby .. ascending, descending</td>
<td>Used to reorder the elements in the query result</td>
</tr>
<tr>
<td>group .. by .. into ..</td>
<td>Groups results by certain attributes</td>
</tr>
</tbody>
</table>

- Note: LINQ expressions can be split into multiple lines
  - Can put each operator on a separate line
A Tutorial on LINQ Operators

● Working example

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

● Course plug

- For more on data management
- Relational databases, SQL, XML, Xquery ..etc
- Check out CS 3300 and CS 4320
From .. in .. select ..

- Return everything about all students
  - var result1 = from s in students select s;

- Only return students names and IDs
  - var result2 = from s in students select new
    {s.ID, s.Name};
The where clause

• Return all students that majored in CS

  - var result3 = from s in students
                 where s.Major == “CS”
                 select s;
orderby

- Return CS students and order them by name
  
  ```csharp
  var result4 = from s in students
                where s.Major == "CS"
                orderby s.Name ascending
                select s;
  ```

- ascending or descending keywords optional
• Group students by major

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

The join operator

- The join operator allows you to join multiple collections on a common attribute
- By joining collections you can deal with pairs of data
- You can learn more about this by taking a database course
• var result6 = from s1 in students
  join s2 in students
  on s1.Major equals s2.Major
  select new {
    Name1 = s1.Name,
    Name2 = s2.Name
  };

join
Deferred Execution

- int[] array = {0,1,2};
  var result = from x in array
    where x % 2 == 0
    select x;
  array[0] = 3;
  foreach (int x in result) {
    Console.WriteLine(x);
  }

- LINQ expressions are not evaluated until iterated over!

- Call `ToArray<T>` or `ToList<T>` to “cache” query results