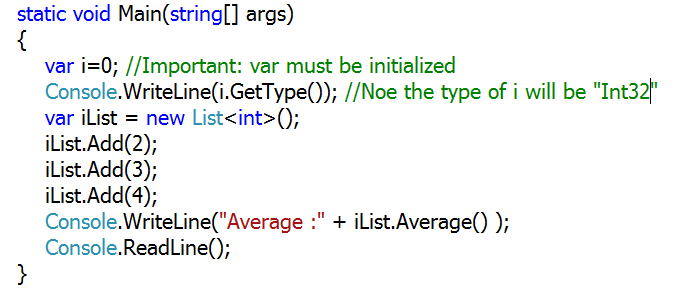
**C# 3.0 New Features**

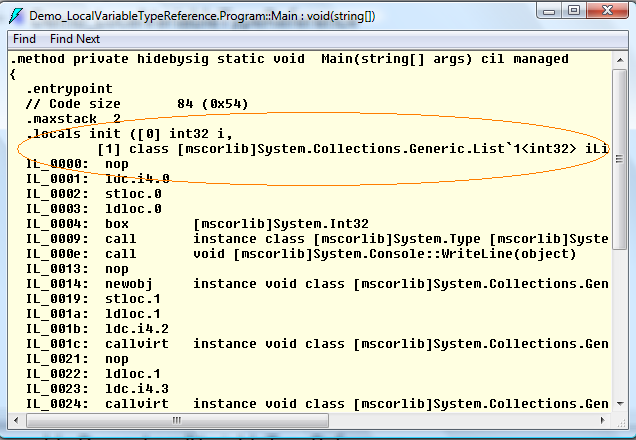
As by now we all know that Microsoft has already released .NET 3.5 Framework, we will now go through some basic details of it. All previous frameworks’ compatible languages are now presented with some new features. In this document we will go through some features of C# 3.0.

1. **Local Variable Type Interface (var):**

This is the new feature released by Microsoft for providing inferred type of “**var**” instead of an explicit type. This keyword instructs to compiler to infer type of the variable based upon the right hand side expression of the initializing statement. The type on the right hand side can be built-in type in .NET type library, user defined type or can also be an anonymous type. Typically “var” is used when declaring anonymous type, this type is generally used in LINQ.

The following example demonstrate how to use var.

The above method clearly specifies the use of “var”.

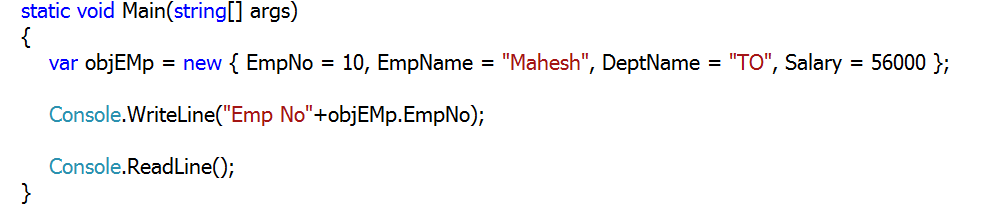


The above diagram shows the IL of the Main method, the variable “i” and the “iList” are inferred by the compiler as int32.

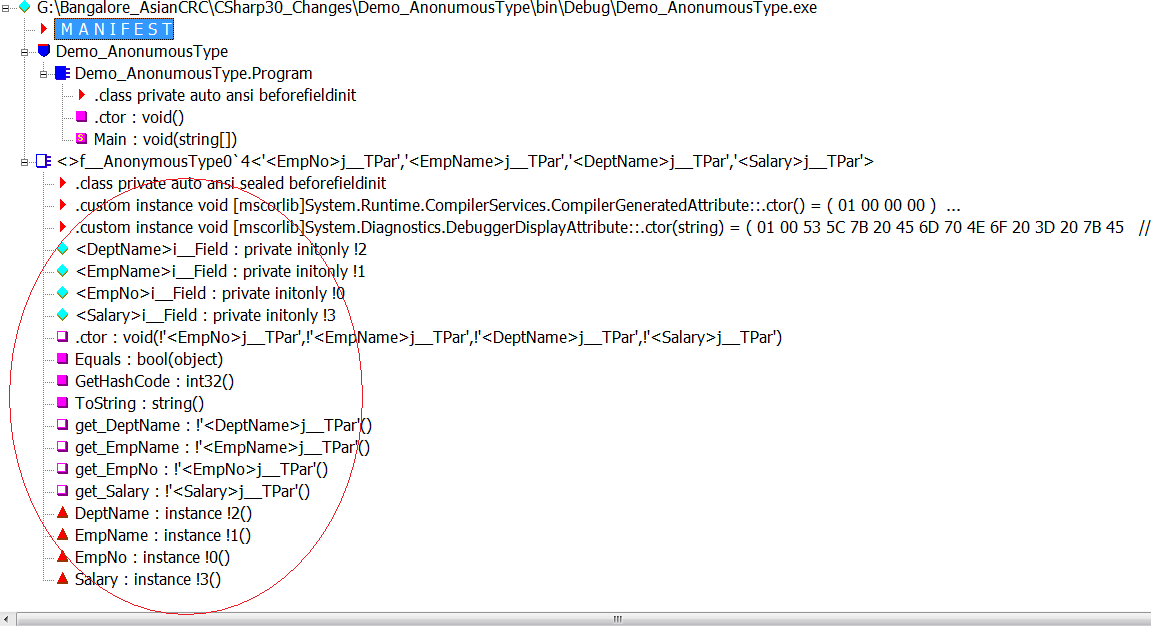
1. **Anonymous Types:**

This is the other facility provided by Microsoft for providing convenient way to encapsulate a set of read-only properties into a single object without having to first explicitly define a type. In this case the type name is generated by the compiler itself, this name is not available at source code level. Type of all properties encapsulated by the anonymous type is inferred by the compiler. Typically these types are used in select clause of LINQ expression to return a subset of the properties from each object in the source sequence.

The following example shows the anonymous type:

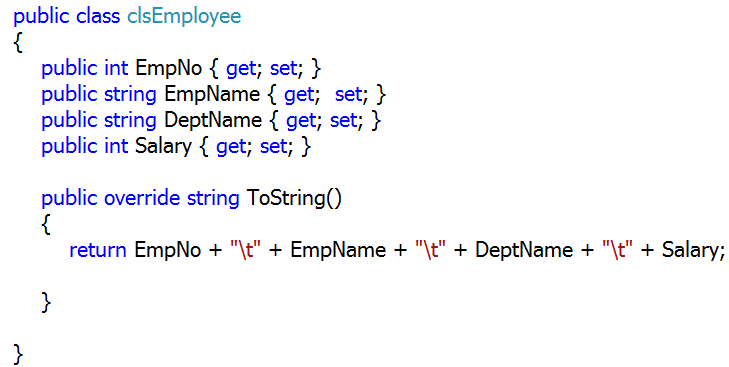


The above example shows how the anonymous type is declared. Properties “EmpNo, EmpName, DeptName and Salary” are inferred by the compiler as below:

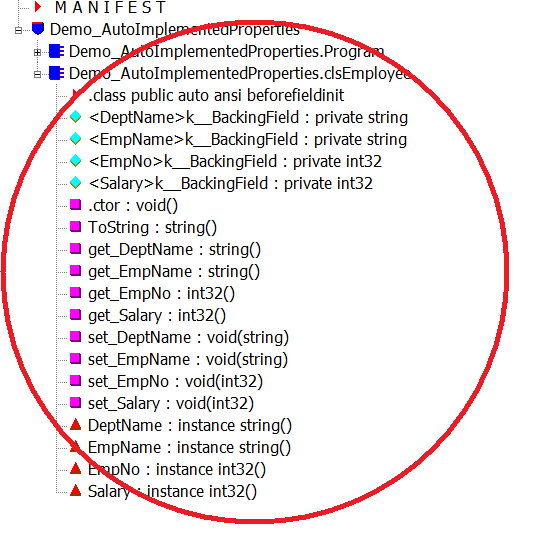


1. **Auto Implemented Properties:**

This is the facility provided to reduce the code of the class if it contains properties only. As we know that properties contains Getter and Setter for returning and setting value parameter, it increases the code length if these properties are not associated with any business logic. In this case the compiler will declare the private fields for every auto-implemented property. The following example shows the declaration of auto-implemented properties:

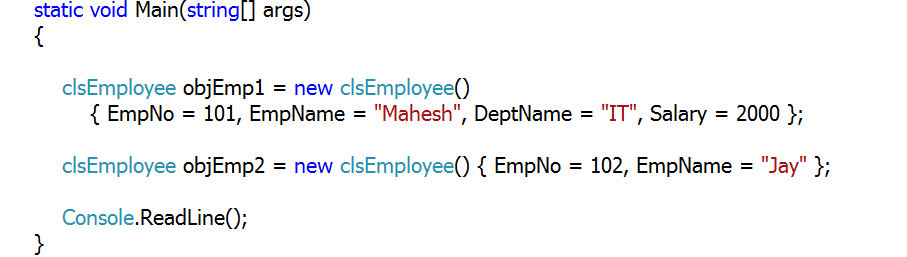


The following IL shows the compiler’s action in case of auto-implemented properties.



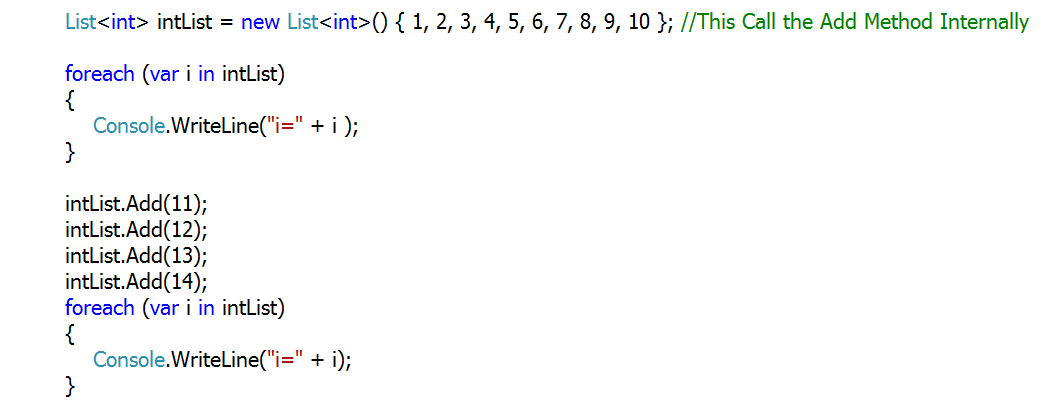
1. **Object and Collection Initializers:**

This facility allows the programmer to initialize object or collection with default values. In this case the object need not required any constructor. The example is shown as below:



The above example uses the “clsEmployee” class declared in the previous example. If you carefully see the code you can easily find that it is not at all necessary to initialize all properties at the time of declaration of the object.

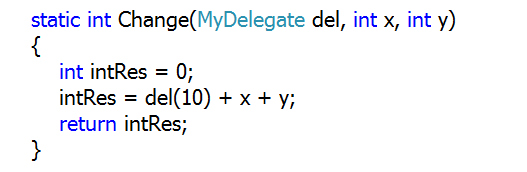
Collection initialize like an array initialization allows us to add default values in collection. In case of collection initialization the “Add()” method of the collection class gets called internally. The example of collection initialization is as below:



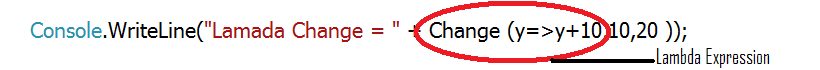
1. **Lambda Expressions**

Lambda expressions are inline expressions similar to anonymous methods, but they are more flexible. Typically they are more used in query expressions, the new concept introduced by Microsoft on .NET 3.5.

Lambda expressions are used for a method which has delegate as parameter passed to it. To understand Lambda expression, lets see an example below:

\

The above method accepts “MyDelegate” as an input parameter. As we all know that delegate is a type-safe pointer used to execute method of matching signature, in the above case we may need to write the separate method which will accepts parameter 10 as an input and will perform some operation on it. But using the Lambda expression the matter can be made much easier as below:

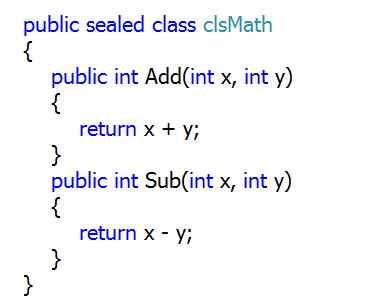


The delegate passed to “Change()” method as first parameter is supposed to increase the value of parameter by 10, the implementation is represented using expression “=>” sign, (pronounced as “Goes into method”).

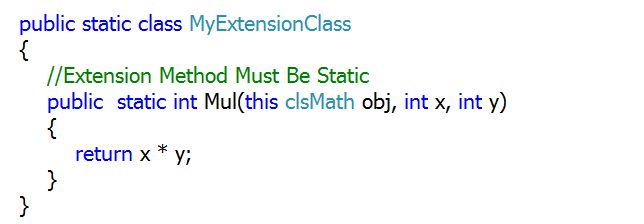
1. **Extension Methods:**

Microsoft has introduced .NET 3.5 using extension methods, most of the new methods present in collection classes are extension methods e.g. Where(), All(),Average(), etc. Technically they are defined as methods made as a part of the class without extending the class or without re-writing it. Typically this concept is used by the developer when he or she wants to create a new method as a part of the class which is written by somebody else in team and class is sealed. To understand the concept clearly we will see the example as below:

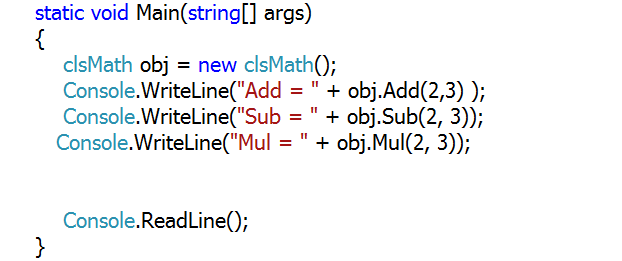
See the math class below:



The class is sealed so cannot be extended, now let’s say a new method of multiply operation of name “Mul” is to be added in Math class then it can be done as below:



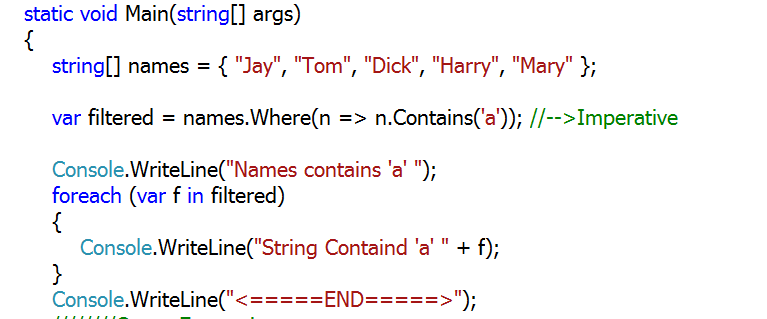
The class in which extension method is to be written must be static at the same time the extension method must also be static and the first parameter of it must be the reference of the class in which class it is to be added logically. So the above example shows that method “Mul()” has the first parameter as the reference of “clsMath” class. Now when this class is called in “Main()” method it will look as below:



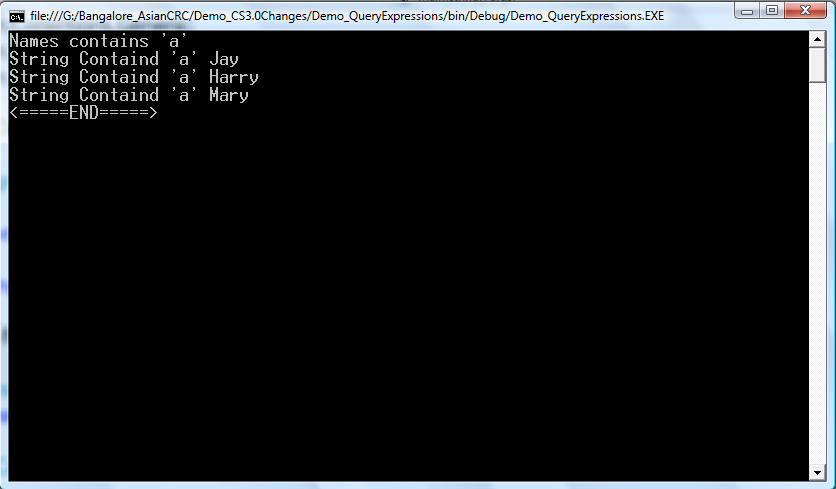
1. **Query Expressions**

This is the facility which is the reason why C# 3.0 is becoming more popular in developers world. Query Expressions make the developer’s life easy by reducing code in the program.

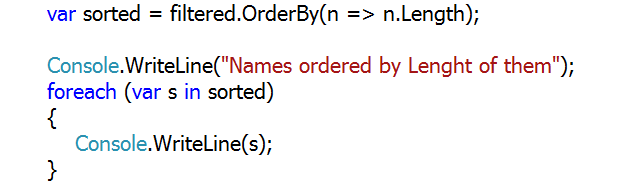
Query Expressions uses, extension methods, Lambda expressions to retrieve values form collection types. To understand the concepts see the below example:



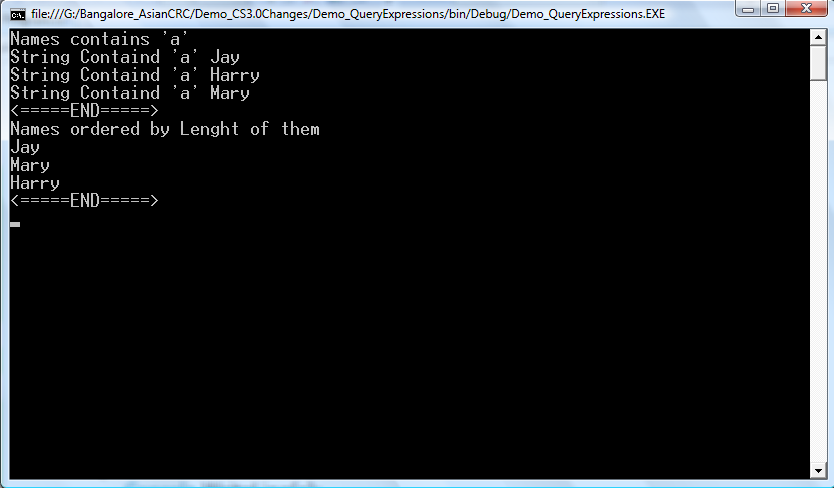
The above example declares string array of names. Now filter only those names which contains ‘a’ we are using the extension method “Where()”, this method takes Lambda expression as an input parameter. The output from the above code is as below:



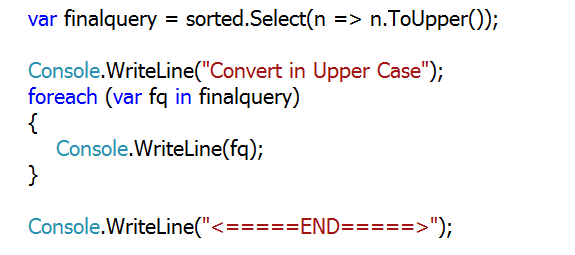
To display the output in the increasing order of the length we can use the query expression as below:



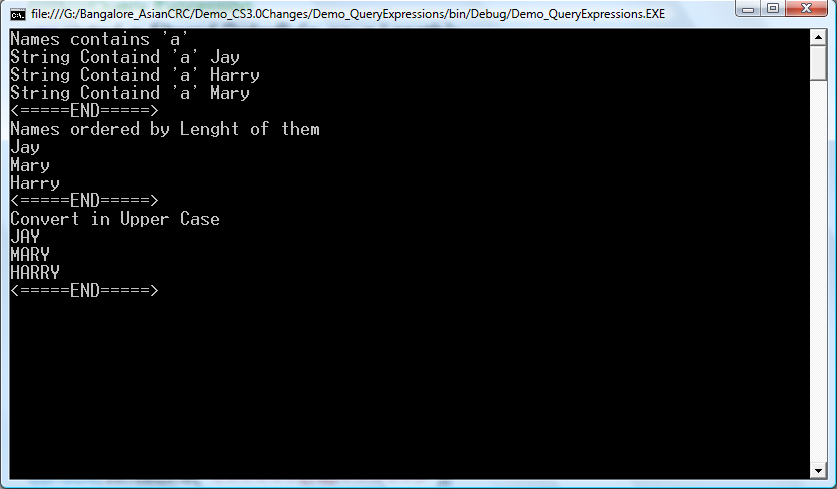
The output will be as below:



Now to convert every string in upper case we will write the following query expression:



The output will be as below:



Now to reduce the code we can put all the expressions together as below:

