Introduction to Visual Studio and C#

Lecture 6

CS 638 Web Programming

Lecture overview

- Visual Studio 2005
  - Editing web pages
  - Hello World
  - “Event-driven” hello world
  - Debugging
- Introduction to C#
  - How C# programs are run
  - Control structures and operators
  - Strings, numbers, enumerations
  - Collections using generics

Launching / Start Page

Intellisense

New console application

Intellisense
Intellisense

```csharp
static void Main(string[] args)
{
  Console.WriteLine("Hello World.");
}
```

Hello World

```csharp
using System;
using System.Collections.Generic;
using System.Text;

namespace HelloWorld
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Hello World");
    }
  }
}
```

What types of programs will you write?

- **A naive view**
  - Structure of program
  - Read input
  - Perform computation
  - Produce output
  - The program may use libraries written by others

- **A realistic view**
  - Structure of program
  - Wait for events
  - Find appropriate method for handling the event
  - Handle the event
  - Repeat
  - Often you just add new events and handlers to an existing program

Breakpoints / single stepping

- F9 toggles breakpoints
- F5 start debugging
- F10 step over
- F11 step into
- Shift F5 kills debugging

Watching / rerunning

- Breakpoint indicator
  - Indicates next line to be executed
- Hovering over variable
  - Shows its value
Lecture overview

- Visual Studio 2005
- Editing web pages
- Hello World
- "Event-driven" hello world
- Debugging
- Introduction to C#
  - How C# programs are run
  - Control structures and operators
  - Strings, numbers, enumerations
  - Collections using generics

How C# programs run

- C# was developed specifically for the .NET platform
- Code compiled to machine-independent Microsoft Intermediate Language (MSIL)
  - Similar to Java bytecode
- Programs execute within an environment called Common Language Runtime (CLR)
  - Similar to Java virtual machine
- Both MSIL and CLR designed to support code written in various languages (e.g., Visual Basic .NET)
- Modules written in different languages interact easily

C# has the usual control flow statements

- if (), else, else if ()
- switch(){case :break;}
  - Break, return, or goto must come after each case
- break
- continue
- goto
  - Will discuss exception handling later
- for(;;)
- while()
- do{}while()
- foreach(element in collection)
  - Repeats body of loop for each element in array or collection

C# has the usual operators

- In order of precedence
  - Primary: x.y (x), a[x], x++,x--,new,typeof
  - Unary: -,!,~,++x,--
  - Multiplicative: *,/,%
  - Additive:+,-
  - Shift: <<,>>
  - Relational and type testing:<,>,<=,,>=
  - Equality:==,!=
  - Logical AND: &
  - Logical XOR: ^
  - Logical OR:  |
  - Conditional AND: &&
  - Conditional OR: ||
  - Conditional: ?:
  - Assignment: =,+=,-
  - Some more exotic C# operators omitted

Some C# types

- object **ALL** types derive from object
- int
- string
- char
- double
- float
- bool
- enums eg: DayOfWeek.Friday
- user defined classes and structs

C# uses strong typing

- Pro – Catch / prevent errors at compile time
- Con – Verbosity
Type conversion
- Implicit
  - Obvious relationship exists
  - No loss of information
- Explicit
  - Like a cast in C or C++
  - May cause runtime error when types checked
- Type conversion
  - System.Convert.To________()
  - Used extensively for ADO.NET work.

Strings
- C# strings use 2-byte Unicode characters
  - string s = "some string";
  - string s += " and some other string";
- s.Length – read only attribute
  - s.Trim() – returns string without leading / trailing white space
- Many other members to the string class
  - Split(), SubString(), etc.

Strings
- Can be indexed: char c = s[2];
- Usual escape sequences
  - \n \t etc.
- Precede with @ to make a literal string
  - @"C:\temp\foo" is the same as "C:\temp\foo"
- Well defined logical operators like =, >, etc.
- More on string manipulation later

Arrays, Generics
- Arrays: strongly typed, fixed length
  - int[] i = { 1, 2, 4 };
  - int[] i = int[3];
- Generics collections: strongly typed, variable length
  - Like C++ STL templates
  - Examples: List<T>, Stack<T>, Queue<T>, Dictionary<K,T>
  - Enable by
  - using System.Collections.Generic;

Generics

```csharp
public void fun()
{
    List<string> Flavors = new List<string>();
    Flavors.Add("Vanilla");
    Flavors.Add("Chocolate");
    if (Flavors.Count > 0) {
        if (Flavors.Contains("Chocolate")) {
            Flavors.Insert(0, "Strawberry");
            Flavors.Add(1);
        }
    }
}
```

Lecture overview
- Visual Studio 2005
  - Editing web pages
  - Hello World
  - "Event-driven" hello world
- Debugging
- Introduction to C#
  - How C# programs are run
  - Control structures and operators
  - Strings, numbers, enumerations
  - Collections using generics