Basic Introduction to C#

Why C# ?

- Builds on COM+ experience
- Native support for
  - Namespaces
  - Versioning
  - Attribute-driven development
- Power of C with ease of Microsoft Visual Basic®
- Minimal learning curve for everybody
- Much cleaner than C++
- More structured than Visual Basic
- More powerful than Java

C# – The Big Ideas

A component oriented language

- The first "component oriented" language in the C/C++ family
  - In OOP a component is: A reusable program that can be combined with other components in the same system to form an application.
  - Example: a single button in a graphical user interface, a small interest calculator
  - They can be deployed on different servers and communicate with each other
- Enables one-stop programming
  - No header files, IDL, etc.
  - Can be embedded in web pages

C# Overview

- Object oriented
- Everything belongs to a class
  - no global scope
- Complete C# program:

```
using System;
namespace ConsoleTest
{
        class Class1
        {
                static void Main(string[] args)
                {
                }
        }
}
```

C# Program Structure

- Namespaces
  - Contain types and other namespaces
- Type declarations
  - Classes, structs, interfaces, enums, and delegates
- Members
  - Constants, fields, methods, properties, events, operators, constructors, destructors
- Organization
  - No header files, code written “in-line”
  - No declaration order dependence

Simple Types

- Integer Types
  - **byte**, **sbyte**(8bit), **short**, **ushort**(16bit)
  - **int**, **uint**(32bit), **long**, **ulong**(64bit)

- Floating Point Types
  - **float** (precision of 7 digits)
  - **double** (precision of 15–16 digits)

- Exact Numeric Type
  - **decimal** (28 significant digits)

- Character Types
  - **char**(single character)
  - **string**(rich functionality, by-reference type)

- Boolean Type
  - **bool**(distinct type, **not** interchangeable with **int**)

Arrays

- Zero based, type bound
- Built on .NET **System.Array** class
- Declared with type and shape, but no bounds
  - **int [ ] SingleDim;**
  - **int [,] TwoDim;**
  - **int [ ][ ] Jagged;btitle**
- Created using **new** with bounds or initializers
  - **SingleDim = new int[20];**
  - **TwoDim = new int[,]{{1,2,3},{4,5,6}};**
  - **Jagged = new int[1][ ];**
  - **Jagged[0] = new int[ ]{1,2,3};**

Statements and Comments

- Case sensitive (myVar != MyVar)
- Statement delimiter is semicolon ;
- Block delimiter is curly brackets { }
- Single line comment is //
- Block comment is /* */
  - Save block comments for debugging!

Data

- All data types derived from

  System.Object

- Declarations:

  datatype varname;

  datatype varname = initvalue;

- C# does not automatically initialize local variables (but will warn you)!

Value Data Types

- Directly contain their data:
  - int (numbers)
  - long (really big numbers)
  - bool (true or false) (unicode
  - char characters)
  - float (7-digit floating point numbers)
  - string (multiple characters together)

Data Manipulation

=	assignment
+	addition
-	subtraction
*	multiplication
/	division
%	modulus
++	increment by one
--	decrement by one

strings

- Immutable sequence of Unicode characters (char)
- Creation:
  - string s = “Bob”;
  - string s = new String(“Bob”);
- Backslash is an escape:
  - Newline: “\n”
  - Tab: “\t”

string/int conversions

- string to numbers:
  - int i = int.Parse(“12345”);
  - float f = float.Parse(“123.45”);

- Numbers to strings:
  - string msg = “Your number is ” + 123;
  - string msg = “It costs ” +
    string.Format(“{0:C}”, 1.23);

String Example

StringExample.cs

```
using System;  namespace ConsoleTest    {
    class Class1  {
            static void Main(string[ ] args)   {
                nt myInt;
                string myStr = "2";
                bool myCondition = true;

                Console.WriteLine("Before: myStr = " + myStr);
                myInt = int.Parse(myStr);
                myInt++;
                myStr = String.Format("{0}", myInt);
                Console.WriteLine("After: myStr = " + myStr);

                while(myCondition) ;
                }
            }
        }
```

Arrays

- (page 21 of quickstart handout)
- Derived from System.Array
- Use square brackets [ ]
- Zero-based
- Static size
- Initialization:
  - int [ ] nums;
  - int [ ] nums = new int[3]; // 3 items
  - int [ ] nums = new int[ ] {10, 20, 30};

Arrays Continued

- Use Length for # of items in array:
  - nums.Length
- Static Array methods:
  - Sort System.Array.Sort(myArray);
  - Reverse System.Array.Reverse(myArray);
  - IndexOf
  - LastIndexOf

Int myLength = myArray.Length;

System.Array.IndexOf(myArray, “K”, 0, myLength)

Arrays Final

- Multidimensional

```
// 3 rows, 2 columns
int [ , ] myMultiIntArray = new int[3,2]

for(int r=0; r<3; r++)
{
        myMultiIntArray[r][0] = 0;
        myMultiIntArray[r][1] = 0;
}
```

Conditional Operators

==	equals
!=	not equals
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
&&	and
||	or

If, Case Statements

if (*expression*)

{ *statements;* }

else if

{ *statements;* }

else

{ *statements;* }

switch (i) { case 1:

statements;

break; case 2:

statements;

break; default:

statements;

break;

}

Loops

```
for (initialize-statement; condition; increment-statement);
    {
        statements;
    }

while (condition)
    {
        statements;
    }
```

Note: can include ***break*** and ***continue*** statements

Classes, Members and Methods

- Everything is encapsulated in a class
- Can have:
  - member data
  - member methods

```
Class clsName
{
        modifier dataType varName;
        modifier returnType methodName (params)
        {
                statements;  return
                returnVal;
        }
}
```

Class Constructors

- Automatically called when an object is instantiated:

```
public className(parameters)
{
    statements;
}
```

Hello World

```
namespace Sample
{
        using System;

        public class HelloWorld
        {
                public HelloWorld()
                {
                }

                public static int Main(string[]
    args)
                {
                        Console.WriteLine("Hello
    World!");
                        return 0;
                }
        }
```

Constructor

Another Example

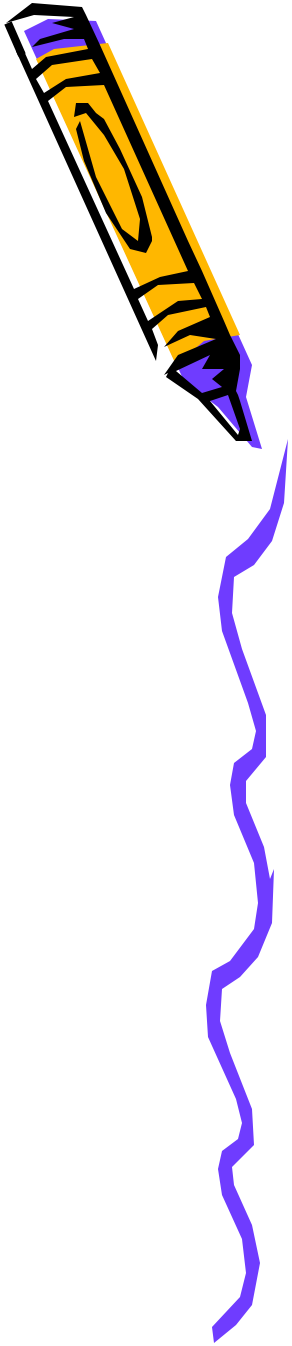
```
using System;
namespace ConsoleTest
{
        public class Class1
        {
                public string FirstName = "Kay";  public
                string LastName = "Connelly";

                public string GetWholeName()
                {
                        return FirstName + " " + LastName;
                }

                static void Main(string[] args)
                {
                        Class1 myClassInstance = new Class1();

                        Console.WriteLine("Name: " + myClassInstance.GetWholeName());

                        while(true) ;

                }
        }
}
```

Summary

- C# builds on the .NET Framework component model
- New language with familiar structure
  - Easy to adopt for developers of C, C++, Java, and Visual Basic applications
- Fully object oriented
- Optimized for the .NET Framework

ASP .Net and C#

- Easily combined and ready to be used in WebPages.
- Powerful
- Fast
- Most of the works are done without getting stuck in low level programming and driver fixing and …



End of The C#