C# (C-Sharp)

An object oriented C variant
Overview

- "C# is simple, modern, and object-oriented"
- **Used for Windows applications**
  - both GUI and web based
  - Linux/OSX support with Mono Compiler
- **Strongly Typed**
- **Object Oriented**
  - Classes, inheritance, interfaces
- **Syntax similar to C, C++, and Java**
- Compiles in Command Prompt or in IDE
A little history...

- Developed by Microsoft
- Released in 2000
- Runs on the .NET architecture
  - Sort of like JVM
  - VB, VC++, COBOL, Perl, Python, SmallTalk, Transact-SQL all have variants that run on .NET
  - Each language compiles to a MSIL
    - This is run by a "Just in Time" compiler
using System;

public class Scope
{
    private int @int = 1; //the @ allows the use of a keyword as an identifier
    //notice that declarations require type
    private void printGlobal()
    {
        Console.WriteLine("Global int = {0}", @int);
    }
    private void changeGlobal()
    {
        @int = 5; //since @int is not redefined, it finds (and changed) the global @int
    }
    private void useLocal()
    {
        int @int = 42; // since @int is being defined, creates a local variable
        Console.WriteLine("Local variable is {0}", @int); //because there is a local variable, gets that one
    }

    public static void Main()
    {
        Scope test = new Scope(); // @int is 1
        test.printGlobal(); //prints "1"
        test.changeGlobal(); //changes @int to 5
        test.printGlobal(); //prints "5"
        test.useLocal(); //does not change test.@int, prints 42
    }
}
Statements

- For loops:
  - for (int i = 0; i<20; i++) {BLOCK}

- while loops:
  - while(node != null) {BLOCK}

- Not much interesting here…
What makes C# different

- Pointers like C, C++
  - must use "unsafe" keyword, though
- Inheritance like Java
  - can inherit one class
- Generics - like Java, mostly
- Properties
- Delegates
Generics

C#
- List<T> compiles to List<T> in MSIL
- At runtime, a literal class is made each time the generic is called with a new type
  - eg. List<int> l1 = new List<int>();
  - List<char> c1 = new List<char>();
  - both List<int> and List<char> are created

Java
- type erasure
  - List<T> compiles to List
    - eg List<String>
      - compiles to List, String replaces T in code
Properties

- Properties are fields, more or less
- get and set methods are built in
- syntax is same as for fields
  - class.time = "04:30";
  - String Y = class.time;
Delegates

```csharp
using System;

class Test
{
    delegate void Printer(); //delegates require return type and parameters to be indicated

    static public void print()
    {
        Console.WriteLine("This is method in the Test class.");
    }

    static public void Main()
    {
        Printer p = delegate()
        {
            Console.WriteLine("This is a an anonomous method contained by a delegate");
        };
        Printer printer = Test.print; //can assign a class method to a delegate
        printer = p; // can assign a function pointed to be a delegate to another delegate
    }
}
```

- A lot like void* in C
Sources

- http://www.angelikalanger.com/GenericsFAQ/FAQSections/ParameterizedTypes.html#FAQ101