C#

Devon Wasson
CSCI 208 – Professor Wittie
Bucknell University, Fall 2015

Write a hello world program in C#:
Please visit http://www.mono-project.com/download/ to install the mono framework, which allows you to run C#. To compile, open up terminal and type mcs [filename.cs]. Once it compiles, type mono [filename.exe] to execute. Here is a basic Hello World function for testing compilation.

Source: http://www.mono-project.com/docs/getting-started/mono-basics/

```
using System;
public class HelloWorld
{
	static public void Main ()
	{
		Console.WriteLine("Hello Mono World");
	}
}
```

Expected output: Hello Mono World

What paradigms does C# cover?
C# is imperative because there are loops and if/else statements.
It is object oriented because it can make classes and has inheritance and polymorphism.
Source: www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf

---

Does C# have a regular expressions library?
Regular Expressions can be implemented in C# with a .NET library. It can be used to parse through text and find character patterns, as one would expect. It can also extract, edit, replace, or delete text substrings. [source]
To use RegEx, you define the pattern you want to identify using the syntax documented in [source2], then call a method that performs one of the above operations.

Source:
https://msdn.microsoft.com/en-us/library/system.text.regularexpressions.regex(v=vs.110).aspx
Source2: https://msdn.microsoft.com/en-us/library/az24scfc(v=vs.110).aspx

---

What primitive types are available?
Source: https://msdn.microsoft.com/en-us/library/ms228360(v=vs.90).aspx
All pieces of information for primitive types are listed below in comments

```
using System;

public class p3 {
   public static void Main() {

       //taken literally exactly from the source above, in the same order
       byte b = 1; // unsigned int of width = 8 and range = 0 - 255 bits
       sbyte sb = 1; // signed int of width = 8 and range = -128 - 127 bits
       int i = 1; // unsigned int of width = 32 and range = -2147483648 - 2147483647
       uint ui = 1; // signed int of width = 32 and range = 0 – 4294967295
       short sh = 1; // signed int of width = 16 and range = -32768 - 32767
       ushort us = 1; // unsigned int of width = 16 and range = 0 - 65535
       long l = 1; // signed int of width = 64 and range = -9223372036854775808 to
           //9223372036854775807
       ulong ul = 1; // unsigned int of width = 64 and range = 0 – 188446744073709551615
       float f = 1.1F; // single precision floating point of width = 32 and range = -3.4x10 ^38 to
           //3.4 x 10^38
       double d = 1.1; // double precision floating point of width = 64 and range = - 1.79 x 10^308 to
           //1.79 x 10^28
       char c = 'a'; // A single unicode character of width = 16 and unicode symbols
       bool bo = true; // Logical Boolean type (true or false) of width = 8 and range true or false
       string s = "Hello world"; // A sequence of characters (not a primitive type)

       //the sizes of each item will now be printed below

       Console.WriteLine("Size of byte: {0}\n", sizeof(byte));
       Console.WriteLine("Size of sbyte: {0}\n", sizeof(sbyte));
       Console.WriteLine("Size of int: {0}\n", sizeof(int));
       Console.WriteLine("Size of uint: {0}\n", sizeof(uint));
       Console.WriteLine("Size of short: {0}\n", sizeof(short));
       Console.WriteLine("Size of ushort: {0}\n", sizeof(ushort));
       Console.WriteLine("Size of long: {0}\n", sizeof(long));
       Console.WriteLine("Size of ulong: {0}\n", sizeof(ulong));
       Console.WriteLine("Size of float: {0}\n", sizeof(float));
       Console.WriteLine("Size of double: {0}\n", sizeof(double));
       Console.WriteLine("Size of char: {0}\n", sizeof(char));
       Console.WriteLine("Size of bool: {0}\n", sizeof(bool));
           }
} //expected output: the correct size of each primitive type
```

What composite types can be made?
C# uses structs [source1], class[source2], and tuples[source3].
Source1: https://msdn.microsoft.com/en-us/library/0taef578.aspx
Source2: https://msdn.microsoft.com/en-us/library/x9afc042.aspx
Source3: https://msdn.microsoft.com/en-us/library/system.tuple(v=vs.110).aspx

Tuples are ordinal. Structs act very closely to classes in the sense that they both have variables and can have methods.

```
using System;

public struct Shirt //struct example
{
	public int size;
	public string color;
	public string brand;
}

public class Friend //class example
{
	public string name;

	public Friend() { }

	public Friend(string n)
	{
		name = n;
	}
	public string getName()
	{
		return name;
	}

}
```

```
public class p4
{
	public static void Main()
	{

		//using the struct
		Shirt shirt = new Shirt();
		shirt.size = 10;
		shirt.color = "red";
		Console.WriteLine("Shirt size:
{0}, shirt color: {1}", shirt.size, shirt.color);

		//using the class
		Friend friend = new
Friend("Elliot");
		Console.WriteLine("My friend is
{0}", friend.getName());

		//using the tuple
		var classroom = new
	Tuple<string, string, int>("Professor
	Wittie", "s208", 30);
		Console.WriteLine("Professor is
{0}, class is {1}, size is {2}.", classroom.Item1,
classroom.Item2, classroom.Item3);

	}

}
```

```
Expected output: Shirt size: 10, shirt color: red
                 My friend is Elliot
                 Professor is Professor Wittie, class is s208, size is 30.
```

How does inheritance work in C#?

Inheritance works like it does in most other OOPLs. A derived class can inherit methods and variables from a parent class. Syntax is derived from the source; the rest is proved with my code.

Source: https://msdn.microsoft.com/en-us/library/ms173149.aspx

```
using System;

public class Person
{
//a person class can only have a name
	public string name;
	public Person() { }
	public Person(string n) //base
constructor for the base class
	{
		this.name = n;
	}
	public string getName()
	{
		return this.name;
	}
}
public class Friend : Person
{
//friend can have an age as well as a name
//name is inherited from the parent class
	public int age;
	public Friend(string n, int a)
//constructor for the derived class
	{
		this.name = n;
		this.age = a;
	}
	public int getAge()
	{
		return this.age;
	}
}
```

```
public class p5
{

	public static void Main()
	{

	Friend friend = new Friend("andy", 20);
	Console.WriteLine("My friend's name is
{0} and he is {1} years old", friend.getName(),
friend.getAge());
	//because we were able to use
friend.getName() which is a method of the
parent class
	//we prove that c# uses inheritance

	}
}
```

Expected output: My friend’s name is andy and he is 20 years old.

Is C# statically or dynamically typed? What about duck typed?

C# requires you to define the type of the variable when writing the code. This is because it is interoperated at compile time. If instead it were determined at run time, type errors would be caught much later. Instead, type errors are caught as soon as you compile. The following program has two versions. One compiles, and the commented one does not. The commented program will not compile if uncommented, proving that C# is not dynamically typed

Source: class notes for understanding, the below program for proof

There is a .NET library for duck typing but it is not specifically included with C#

Source: https://github.com/deftflux/DuckTyping

```
using System;

public class p6

{
   public static void Main()

   {
      int i = 0;

      char b = 'c';

/*
      q = 0;

      a = 'c';

 compiles with these two errors:

p6.cs(29,9): error CS0103: The name `q' does not exist in the current context

p6.cs(30,9): error CS0103: The name `a' does not exist in the current context

*/

   }

}
```

No expected output, just should compile and run without error (might have warnings for unused variables)

Is C# strong, weak, or untyped?
C# is a more strongly typed language. This is proven by the errors produced or not produced below. Because we can convert int to dbl, it is not perfectly strongly typed, but because we can not make any more implicit casts, it is not a weakly typed language.

Source: class notes on strong vs weak typed language and the code below

```
using System;

public class q03
{
    public static void Main()
    {
        double b = 0.0;
        b = 1;
        //because c# is "more" strongly typed,
this is allowed.
        //there is no loss of data, so c# will
convert an int to dbl
        //so this proves that it is not "perfectly"
strongly typed

        int c = 0;
        c = 'c';
        //this will run because there is a
numeric equiv for a char in c#.
        //this is more proof as to why it is not
strongly typed, but still
        //does not prove that it is more weakly
typed.

        //the code above does not produce
errors.
        //the code below produces errors.
        //uncomment the lines below to see the
errors they produce

/*
        int i = 0;
        i = 5.5;
        //because c# is more strongly typed,
this will not run
        //because you cannot convert a double
to an int without a cast


        char a = 'd';
        a = 1;
        //if c# is more strongly typed, this
should produce an error at compile time.
        //if it was more weakly typed, this
would produce an error at run time
        //because c# is more strongly typed,
this will produce an error at compile time
        //because you cannot convert an int to a
char.
*/

    }
}
```

No expected output, just should compile and run without error (might have warnings for unused variables)

Static vs dynamic method binding

Dynamic method binding is when the method being called is looked up at run time. Static method binding is when the method being called is looked up at compile time. If we can have an object be of one type but actually be another type, and have the methods of the second type be attributes of the object then we have dynamic method binding. In the example below, a Bird object is defined as a toucan. If calling bird.fly() prints the method for toucan, then we know C# uses dynamic method binding. The function was called at run time rather than compile time. If it was called at compile time, the compiler wouldn’t "know" the function of the toucan and it would bind the method for the bird to our obj.

Source: class notes/homework on dynamic method binding and the code below

```
using System;

public class q04 {

   public static void Main() {
      Bird bird = new Toucan(); //has type bird
but obj will be toucan
      bird.fly();
         //if bird.fly() produces "Toucan is flying"
then we have dynamic method binding
         //if we get "Generic Birds are flying"
then we have static method binding

   }
}
```

```
public class Bird {

   public Bird() { }

   public virtual void fly() {
      Console.WriteLine("Generic Birds are
Flying");
   }
}

public class Toucan : Bird {

   public Toucan() { }


   public override void fly() {
      Console.WriteLine("Toucan is Flying");
   }
}
```

Expected output: Toucan is Flying

Deep vs shallow copying of objects

C# uses shallow copying of objects. This is proven given my understanding of copying from class and the code below. The code below says a new object is equal to an existing object. If they language used deep copying, altering one object would not change the other. However, because C# uses shallow copying, the new object points to the old object in its assignment so changing one object changes the other, as seen in the code and what prints out below.

Source: class notes and the code below

```
using System;

public class Person
{
//a person class can only have a name
    public string name;

    public Person() { }

    public Person(string n)
    {
        this.name = n;
    }

    public string getName()
    {
        return this.name;
    }

        public void setName(string n)
        {
                this.name = n;
        }

}
```

```
public class p5
{
public static void Main()
    {
        Person person = new Person("andy");

        Person person2 = person;
        person2.setName("Nadeem");
        Console.WriteLine("{0}", person.getName());
        Console.WriteLine("{0}", person2.getName());

        //if the two outputs are the same (Nadeem), then
we know c# does shallow copying
        //if it prints andy and Nadeem, then we know is
does deep copying.

    }


}
```

Expected output: Nadeem Nadeem

Structural reflection in C#
Structural reflection is possible in c# by giving objects the type Type. Reflection dynamically creates an instance of type and will bind it to a pre-existing object or get the type of an existing object and invoke the methods and access its fields.
In the below example, GetType is used to get the type of the variable.

Source: https://msdn.microsoft.com/en-us/library/ms173183.aspx

```
using System;

public class q06{

        public static void Main(){
                int i = 0;
                System.Type t = i.GetType();
                Console.WriteLine(t);
                //prints System.Int32
        }

}
```

Expected output: System.Int32

Static vs dynamic scope in C#
Given the below code, we can infer that c# is static scope. If it were dynamic, when bar calls foo, the value for b would be 3. But because it is static, foo searches the global name space instead of the function that called it.
Source: the below code.

```
using System;

public class q08{

  const int b = 2;

  public static int foo(){
    int a = b + 1;
    return a;
  }

  public static int bar(){
    int b = 3;
    return foo();
  }
```

```
public static void Main(){
    Console.WriteLine(foo()); //should be 3
    Console.WriteLine(bar()); //if static scope, should be 3
                                          //if dynamic, should be 4
    //output: 3 3
  }

}
```

Expected output: 3 3 (warnings for unused variables)

How does C# handle parameter passing?
C# passes arguments by value by default, but can also pass by reference using the ref keyword. The example below highlights this.

Source: https://msdn.microsoft.com/en-us/library/0f66670z.aspx

```
using System;

public class q09{

public static void foo(int x){
  x = 1;
 }

 //overload method for pass by ref
 public static void foo(ref int x){
  x = 1;
 }
```

```
static void Main(){

    //pass by value
    int i = 0;
    foo(i);
    Console.WriteLine(i);
    /* i should still be 0 */

    //pass by reference
    int j = 0;
    foo(ref j);
    Console.WriteLine(j);
    /* j should now be 1 instead of 0 */

  }
}
```

Expected output: 0 1

Does C# have list comprehension?
Yes, by implementing the library Linq. List comprehension is not built in, but can be added. To do so, one should use the keywords below. From 'x' in 'range(a,z)' select 'something'
Source: the below code

```
using System;
using System.Linq;

public class q11{

 public static void Main(){
   int[] list1 = {1,2,3,4,5};

   //gives the square of all numbers in the range of 1 to 5
   var list = from x in list1 select x*x;
   //makes an array of type System.Collections.Generic.IEnumerable<int>
 }

}
```

No expected output, just should compile and run without error (might have warnings for unused variables)

Does C# have anonymous functions?
C# can use lambda expressions and anonymous functions to make delegates or expression tree types. This allows one to write local functions to pass as arguments or return as the value of a function call.

Source: https://msdn.microsoft.com/en-us/library/bb397687.aspx

```
using System;

class q14{
	delegate int del(int i);
	public static void Main(){
		del d = x => x*x;   //this returns the squared for any x
		int i = d(5); //should be 25
		Console.WriteLine(i);
	}

}
```

Excepted output: 25

Does C# support polymorphism?

C # supports polymorphism. The below code demonstrates how we can refer to derived classes as the base class without error.

Source: https://msdn.microsoft.com/en-us/library/ms173152.aspx

```
using System;
public class Plant{
  public string name;
  public Plant(){}
}
public class Tree : Plant{
  public string name = "TREEE";
}
public class Bush : Plant{
  public string name = "bushhhhh";
}
```

```
public class q15{
  public static void Main(){
    Tree t = new Tree();
    Bush b = new Bush();
    Plant[] l = {t, b};
    //this code runs because c# supports
polymorphism.
    //we can have an array of plants, but each
plant can be a derived class
  }
}
```

No expected output, just should compile and run without error (might have warnings for unused variables ect…)

Can you override methods in C#?

C# allows for overriding of methods. The parent class needs to have its method being overwritten labeled as "virtual" The child class needs to inherit the parent class and mark the method they are overriding as "override" Then, the overriding of methods will work

Source: https://msdn.microsoft.com/en-us/library/ebca9ah3.aspx

```
using System;
public class q16 {
   public static void Main() {
      BigDog bigDog  = new BigDog();
      bigDog.bark();
          Dog dog = new Dog();
          dog.bark();
          //because c# allows for overriding,
calling bark on bigDog will print
          // BARK BARK BARK!!!
          //and calling bark on dog will print bark
bark bark
   }
}
```

```
public class Dog {
   public Dog() { }
   public virtual void bark() {
      Console.WriteLine("bark bark bark!");
   }
}
public class BigDog : Dog {
   public BigDog() { }

   public override void bark() {
      Console.WriteLine("BARK BARK BARK!!!");
   }
}
```

Expected output: BARK BARK BARK!!!

```
                    bark bark bark!
```

General information about C#

-c# was first released in the year 2000. [source1] -It was created by Microsoft. [source1] -It is a compiled language. [source2] -It was made to be part of the .NET framework. It was designed to be a modern and simple programming language, capable of all traditional things while implementing the .NET framework. [source1] -Many companies are hiring for c# [source 3] including EPIC healthcare [source4]	**Source1**: http://aboutcsharpprogramming.blogspot.com/2012/09/history-of-c-programming.html **Note**: this looks credible, even though it is a blog. The author has written many articles on c# and they are all intelligent and correct. The author also appears to have lots of experience, as they have been blogging since 2011. **Source2**: the fact that we are using the mono compiler. For windows, there is the Visual C# compiler. http://www.mono-project.com/docs/about-mono/languages/csharp/ **Source3**: https://www.glassdoor.com/Job/c-net-developer-jobs-SRCH_KO0,15.htm **Source4**: https://cse.sc.edu/job/software-developer-epic-systems

Is C# implicitly or explicitly typed?

C# supports both explicit and implicit typing for variables. This is demonstrated by the fact that we can call variables an int, or a var. However, we cannot have implicit function types. Functions must have a return type. This can be checked by uncommenting the bar function and compiling the code.

Source: the below code

```
using System;
public class q19
{
   public static void Main()
   {
        int i = 0;
        var p = 0;
   }
public int foo(){
   return 0;
  }
/*
 public bar(){
   return 0;
  }
*/
}
```

No expected output, just should compile and run without error (might have warnings for unused variables)

Does C# allow super classes?

A constructor for the derived class can call the constructor for a super class, which will allow the derived class to inherit all methods and fields from the super class.

Source: https://msdn.microsoft.com/en-us/library/ms228387(v=vs.90).aspx

```
using System;
public class Animal{

  public string name;

  public Animal(){}

  public Animal(string name){
    this.name = name;
  }

  public void getName(){
    Console.WriteLine(this.name);
  }

}
public class Cat : Animal{
    private int age;
    public Cat(string name, int age){
      this.age = age;
      this.name = name;
    }
    public void getAge(){
      Console.WriteLine(this.age);
    }
}
public class q21{
  public static void Main(){
    Cat cat = new Cat("Mike", 20);
    cat.getName(); //inherited
    cat.getAge(); //from the derived class
  }
}
```

Expected output: Mike 20

Pointers and references in C#

C# allows references in any context and pointers in an "unsafe" context. Pointers can only be used if they are in an unsafe context.[source1] However, my version of mono does not support this and thus cannot be tested. References can be made using the ref keyword. [source2]

Source1: https://msdn.microsoft.com/en-us/library/y31yhkeb.aspx

Source2: https://msdn.microsoft.com/en-us/library/14akc2c7.aspx

```
using System;

public class q22{

 public static void foo(ref int i){

  i++;

 }
```

```
public static void Main(){
  int i = 0;
  foo(ref i);
  Console.WriteLine(i); //should be 1
  //the second line shows we can make a pointer.
  //the method bar shows what happens when we pass an int
by ref
  //because it was by ref, i should now be 1.
 }

}
```

Expected output: 1

Are strings primitive types in C#?

Strings are not primitive types. They are reference types. Strings are series of characters. Chars are primitive types, so a string is a reference to a series of primitive types. [source1] You can use the = and + operators on strings, and can index strings. [source2]

Source1: https://msdn.microsoft.com/en-us/library/ms228360(v=vs.90).aspx

Source2: https://msdn.microsoft.com/en-us/library/362314fe.aspx

```
using System;
public class q24{
 public static void Main(){
  string s = "string";
  string s2 = "strong";
  string s4 = "string";
```

```
Console.WriteLine(s==s2); // false
  Console.WriteLine(s==s4); //true
  string s3 = s+s2;
  Console.WriteLine(s3); //stringstrong
  Console.WriteLine(s[0]); // s
 }
}
```

Expected output: False True stringstrong s

What operations can you do on numbers? Is it different for chars and other int-like types?

You can do + - * / and % operations for math in c# [source]. This works on ints, and ints in other forms. This includes chars. However, chars are implicitly cast to ints when doing math, so to retain a char value, they must be explicitly cast back to char. One should check that the int is a valid value for a char when casting, however. [below code]

Source: https://msdn.microsoft.com/en-us/library/aa691371(v=vs.71).aspx

```
using System;
public class q25{
 public static void Main(){
   int i = 1+1;
   i = 1-1;
   i = 1*1;
   i = 1/1;
   i = 1%1;
   Console.WriteLine(i);
   //you can only do funky math this way.
   //doing math on chars converts to an int
implicetly, so you
   //have to convert explicitely back to a char.
   char c = (char)('c'+'d');
   Console.WriteLine(c);
   //this line doesnt work because it returns -1
and there is no ascii for that
   //c = (char)('c'-'d');
```

```
c = (char)('c'*'d');
   Console.WriteLine(c);
   c = (char)('c'/'d');
   Console.WriteLine(c);
   c = (char)('c'%'c');
   Console.WriteLine(c);

   int h = 0xBE - 0xBA;
   Console.WriteLine(h);
   h = 0xBE + 0xBA;
   Console.WriteLine(h);
   h = 0xBE / 0xBA;
   Console.WriteLine(h);
   h = 0xBE * 0xBA;
   Console.WriteLine(h);
   h = 0xBE % 0xBA;
   Console.WriteLine(h);
 }
}
```

Expected output: 0 Ç ○ [nonASCII] [nonASCII] 4 376 1 35340 4

Does C# support multidimensional arrays?

These behave exactly like regular arrays. A multi dimensional array is just an array whose values are filled with arrays. You can assign it values or just initialize an empty array, like normal.

Source: https://msdn.microsoft.com/en-us/library/2yd9wwz4.aspx

```
using System;
public class q28{
 public static void Main(){
   int[,] twoD = new int[2,2]; //init empty array
   int[,] twoD2 = new int[,] {{1,1}, {2,2}};
```

```
for (int i = 0; i  < 2; i++){
     for (int j = 0; j < 2; j++){
       Console.WriteLine(twoD2[i,j]);
          //prints 1122
     }
   }
 }
}
```

Expected output: 1 1 2 2

How does C# handle the dangling else?

C# uses the in-most if statement when pairing with a dangling else.

In the below example, it is ambiguous to which if statement the else statement belongs. Because C# pairs else statements with the in-most if statement, the else statement in this case belongs to the if (i==0) statement. So in this example, the program will not print anything, which is correct for c#.

Source: the below code

```
using System;

public class q30{

  public static void Main(){

    int i = 0;
if(i == 1)
      if (i == 0)
        Console.WriteLine("Here");
      else
        Console.WriteLine("Here2");
  }
}
```

No expected output which proves the argument for dangling else.

Order of operations in C#

Operations go from highest priority to lowest, and from left to right in the event of two operations having the same priority.

Source: https://msdn.microsoft.com/en-us/library/aa691323(v=vs.71).aspx

Primary: x.y f{x} a[x] x++ x-- new typeof checked unchecked	Logical XOR: ^
Unary: + - ! ~ ++x --x (T)x	Logical OR: |
Multiplicative: * / %	Conditional AND: &&
Additive + -	Conditional OR: ||
Shift: << >>	Conditional: ?:
Relational and type testing: < > <= >= is as	Assignment: = *= /= += -= <<= >>= &= ^= |=
Equality: == !=	
Logical AND: &	

Does it do implicit type conversions? Which ones? Does it allow explicit conversions?

C# allows for both explicit and implicit type conversions. (source1) You can only implicitly cast if there is no precision loss or from a derived class to a base class (source1). You can do an explicit conversion to lose precision or from a base class to derived class (source1). We can also convert chars to nums and nums to chars using explicit casting (source2)

Source1: https://msdn.microsoft.com/en-us/library/ms173105.aspx

Source2: knowing how to cast, and the code below

Chart for all explicit numeric conversions: https://msdn.microsoft.com/en-us/library/yht2cx7b.aspx

Chart for all implicit numeric conversions: https://msdn.microsoft.com/en-us/library/y5b434w4.aspx

```
using System;
public class q33{
        public static void Main(){
                int a = 1;
                double b = a;
                //this works because no loss of
precision
                Console.WriteLine("{0}\n",b);
                double c = 1.1;
                int d = (int) c;
                //we lose precision but we explicitly
casted so it is okay
                Console.WriteLine("{0}\n",d);
                char e = 'e';
                int f = (int)e;
                Console.WriteLine("{0}\n",f);
                int g = 92;
                char h = (char)g;
                Console.WriteLine("{0}\n",h);
        }
}
```

Expected output: 1 1 101 \

In what order are parameters evaluated?

Taken as a direct quote: "During the runtime processing of a function member invocation, the expressions or variable references of an argument list are evaluated in order, form left to right..."

Source: The C# Programming language (Covering C# 4.0), Portable Documents, section 7.5.1.2 Runtime Evaluation of Argument Lists

Exception and error handling in C#

Try and catch blocks can be used to catch code that might cause an error. Finally blocks can be included to specify code to be run regardless of if an exception was thrown or not. You can specify the type of exception to catch with a catch block, which can be derived from exception. Naturally, different catch blocks catching different exceptions can be together.

Source: https://msdn.microsoft.com/en-us/library/ms173162.aspx

```
using System;
public class q35{
  public static void Main(){
    int[] a = {1,2,3};
    try{
      Console.WriteLine(a[3]); //will error out of bounds
    }
    catch (System.IndexOutOfRangeException e){
      Console.WriteLine("Index out of range!");
    }
    finally{
      Console.WriteLine("In the finally block.");
    }
  }
}
```

Expected output: Index out of range!

```
                 In the finally block.
```

Statements vs expressions in C#

A statement is any action that the program takes. Things such as variable assignment, calling methods, if/else/switch/case and loops/iterations/jumps/exceptions are all statements.

An expression is something that can be evaluated into a single value. This includes something like x > 5 because this would evaluate to either true or false. There are no side effects with expressions.

Source(statement): https://msdn.microsoft.com/en-us/library/ms173143.aspx

Source(expression): https://msdn.microsoft.com/en-us/library/ms173144.aspx

Does C# use short circuit evaluation?

Yes it does. The below code demonstrates this [source1]. We can avoid short circuit evaluation by using just a single bar or ampersand [source2].

Source1: the below code

Source2: the below code

```
using System;
public class q39{
  public static void Main(){
    int a = 1;
    int b = 2;
    if ((a == 2) && (b == b++)){
      Console.WriteLine(b); //this will not print because a short circuit evaled to false
    }
    else{
      Console.WriteLine(b); //b will still be 2 because the second arg was never evaluated
    }
if ((a == 2) & (b == b++)){
      Console.WriteLine(b); //this will not print because the condition is not met
    }
    else{
      Console.WriteLine(b); //but b will now be 3 because the entire expression was evaluated
    }
    //expected output: 2 3
  }
}
```

Expected output: 2 3

Does C# use prefix, postfix, or infix, or some combination?

C# uses a combination of these.

Some primary operators are infix: x+y, x-y, x*y ect...

Some unary operators are prefix: +x, -x, !x, ect...

And some use postfix: x++, x--, y[x]

Function calls are prefix: f(x) -> the function call gets evaluated first

Source: https://msdn.microsoft.com/en-us/library/6a71f45d.aspx

What can you overload? Can you overload function names? Can you overload operators yourself? Does the language come with any overloaded operators?

C# lets you overload operators with user-defined types by defining static member functions. Comparison operators must be overloaded in pairs; if == is overloaded, then != must be too. You can overload + - ! ~ ++ -- true false * / % & | ^ << >> == != < > >= <= [source1] and function names[source2]. You can make two different functions with different parameters but the same name, and compiler will check to see which one is being used when compiling.

Source1: https://msdn.microsoft.com/en-us/library/8edha89s.aspx

Source2: the below code

```
using System;
public class Line{
  private int length;
  public Line(){}
  public Line(int i){
    this.length = i;
  }
  public int getLength(){
    return this.length;
  }
  public static Line operator + (Line l1, Line l2){
    Line line = new Line();
    line.length = l1.getLength() + l2.getLength();
    return line;
  }
}
```

```
public class q44{
  static void bar(int x){
    Console.WriteLine("printing an int");
  }
  static void bar(string s){
    Console.WriteLine("printing a str");
  }
  public static void Main(){
    bar(5);  //both will run because overloaded
method
    bar("ello");
    Line l = new Line(1);
    Line l2 = new Line(2);
    Line l3 = new Line();
    l3 = l + l2;
    Console.WriteLine(l3.getLength()); //should
be 3
  }
}
```

Expected output: printing an int. printing a str. 3

What types of tokens does C# have?

Tokens include identifiers, keywords, integral-literals, real-literals, character-literals, string-literals, operator-or-punctuator. Whitespace and comments are not tokens.

Source: https://msdn.microsoft.com/en-us/library/aa664668(v=vs.71).aspx

What control structures does C# use?

C# has selection control and repetition control [source1]. Selection control: c# allows for signal selection (if), double selection (if...else), multiple selection (switch) and inline conditional operations (?:). Repetition control: c# allows for top tested (while), bottom tested (do...while), property or array control (for...in), and counter controlled repetition (for)

Source1: https://msdn.microsoft.com/en-us/library/vstudio/e240yzs4(v=vs.100).aspx
Source for selection control (except ?: and switch structure): knowledge of selection control and below code
Source for switch structure: https://msdn.microsoft.com/en-us/library/06tc147t.aspx
Source for (?:) structure: https://msdn.microsoft.com/en-us/library/ty67wk28.aspx
Source for repetition control (except foreach): knowledge of repetition control and below code
Source for foreach: https://msdn.microsoft.com/en-us/library/ttw7t8t6.aspx

```
using System;
public class q45{
	public static void Main(){
		int a = 0;
		if(a == 0){
			Console.WriteLine("If statement
works!\n");
		}
		int b = 0;
		if (b == 1){
			Console.WriteLine("did not
make it to else statement\n");
		}
		else{
			Console.WriteLine("Else
statement worked!\n");
		}
		int caseSwitch = 2;
		switch (caseSwitch)
		{
			case 1:

	Console.WriteLine("Switch made it to case 1\n");
				break;
			case 2:

	Console.WriteLine("Switch made it to case 2\n");
				break;  //use a break if
you want to stop after a case

			default:

	Console.WriteLine("Default case has been
reached\n");
				break;
		}
string popularity;
		int numFriends = 50;
		popularity = (numFriends > 25) ?
"Popular!" : "Not Popular";
		Console.WriteLine("You are: {0}\n",
popularity);
		//sets the popularity variable to popular
if numFriends > 25,
		//and not popular if less than 25
		//works similar to if else, but can be
done in one line
		int c = 0;
		while (c < 3){
			Console.WriteLine("Writing
while loop line number {0}\n", c);
			c++;
		}
		int d = 0;
		do{
			Console.WriteLine("Writing do
while loop line number {0}\n",d);
			d++;
		} while (d < 1);

		int[] nums = new int[4] {0,1,2,3};
		foreach (int x in nums){
			Console.WriteLine("For x In
array: {0}\n", x);
		}
		for (int e = 0; e < 3; e++ ) {
			Console.WriteLine("For line
number {0}\n", e);
		}
	}
}
```

Expected output: Else statement worked! Switch made it to case 2. You are: Popular! Writing while loop line number 0..1..2. Writing do while loop line number 0. For x in array: 0..1..2..3. For line number 0..1..2.

What are token delimiters in C#?

Strings can be split using the .Split operation. An example of this in action is shown below.

Source: https://msdn.microsoft.com/en-us/library/ms228388.aspx

```
using System;
public class q50{
  public static void Main(){
    char[] deliminatingChars = {' '};
    string text = "Hello world and all who inhabit it.";
    string[] words = text.Split(deliminatingChars); //makes an array of tokens
    for (int i = 0; i < words.Length; i++){
      Console.WriteLine(words[i]); //write each token in the string.
    }
  }
}
```

Expected output: Hello world and all who inhabit it.

Does C# do file IO? (question not on the project page)

Yes it does. It uses a FileStream class, which is derived from the Stream class. Below is an example of how to use this. The below code proves the functionality, but the inspiration for this example came from the source below.
Source: http://www.tutorialspoint.com/csharp/csharp_file_io.htm

```
using System;
using System.IO;

public class nq01{
  public static void Main(){
    FileStream f=new FileStream("example.txt",FileMode.OpenOrCreate,FileAccess.ReadWrite);
    for (int i = 10; i > 0; i--){
            f.WriteByte((byte)i);
    }
    f.Position = 0;
    for (int i = 10; i > 0; i--){
            Console.WriteLine(f.ReadByte());
    }
    f.Close();
  }
}
```

Expected output: 10 9 8 7 6 5 4 3 2 1

Does C# have garbage collection?

Yes, C# does have garbage collection. It is automatic so you do not have to access it yourself. This means you can develop without having to worry about deallocation of memory. All objects are managed efficiently. Objects that are no longer being used are reclaimed and their spaces freed, and it provides memory safety by "making sure that an object cannot use the content of another object."

Source: http://msdn.microsoft.com/en-us/library/ee787088(v=vs.110).aspx

Comments in C#

Comments in c# -> single line comments use //

Multiline comments use /* */

Source: https://msdn.microsoft.com/en-us/library/aa664667(v=vs.71).aspx

What would you change in C#?

If I could change one thing about c#, it would be the way it handles tuples. After working with Haskell, it would be much nicer if it handled them that way. In c#, one has to type Tuple<int, int> to define a tuple type and Tuple.Create (x, y); to actually make it. If it worked like Haskell, it could just be (int, int) for our tuple type, and (x, y) to assign it.

For example, we could go from this to this:

```
Tuple<int, int> t = Tuple.Create(1,2);
```

```
(int, int) t = (1, 2);
```