

**C# PROGRAMMING FOUNDATIONS:
MODULE
Initial Steps into C# Syntax**



© August 2011 by
ERIC MATTHEWS

Copyright Notice

C# Foundations by Eric Matthews is licensed under a Creative Commons Attribution 3.0 Unported License. This license allows you to

- Copy, distribute and transmit the work
- Adapt the work
- Make commercial use of the work

Under the following conditions:

You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work

With the understanding that:

Waiver — any of the above conditions can be waived if you get permission from the copyright holder.

Public Domain — where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.

Other Rights — in no way are any of the following rights affected by the license:

- Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
- The author's moral rights;
- Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.

Notice — for any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to one or more of the following web pages.

www.DeveloperGeekResources.com

www.EducationAnytime.com

Contents

Copyright Notice	2
<i>Writing Some “Hello World” Programs</i>	5
Bare bones program (for our use now).....	5
Literals.....	6
Compiling your program.....	6
Running your program.....	6
Hello The C# OOP Way.....	7
Variables.....	9
Put into practice – first C# program.....	10
Basic operators.....	10
Conditional logic.....	11
if	11
A quick sidebar on indenting	13
If else	14
Nested if	15
if else if	18
switch (Case logic)	19
Limitations to coding conditional logic in C#?	21
Put into practice – conditional logic.....	21
More About string.....	21
Declaring and initializing a String	22
Determining the length of a String	22
String case conversion	22
Trim	22
Split a String	22
String to a character array	22
String Modification	22
Compare Two Strings	22
Arrays in C#.....	23
Initialize Array and Set values (thus determining size)	23
Initialize Array to Specific Size, Set values later	23
Get # of Elements in Array	24
Dynamic Arrays in C#.....	24
Create a List	25
Getting a Count of List Elements	25
Adding a List Element	25
Removing a List Element	25
Sorting a List	25
Reversing a List	25

Loops26

- for**26
- Increment operator**26
- Decrement operator**.....27
- while**.....27
- do**.....27
- nested loops**29

Put into practice – loops and other stuff we have covered up until now30

Comment Code In C#.....31

- Single line comments**31
- Multiple line comments**.....31

A final word on this subject.....31

WRITING SOME “HELLO WORLD” PROGRAMS

We are going to start learning C# by covering the following subjects:

- Variables/Literals
- Conditional Logic
- Loops
- Arrays
- Compilation
- Running your program

My intent is to give you enough familiarity and practice with the syntax that you are acclimated enough to hit the next module, which is “First Steps in Object Oriented Programming”. I have taken this approach in courses I have taught on structured programming. I got my audience initially comfortable with the syntax so I could help them better focus on what I considered to be mission critical subject...namely writing subroutines and functions and achieving reuse in their code (if only on a personal level).

Since this approach has brought me success in the past I want to employ it again, though applied only to those subjects that are mission critical in learning, and more importantly, thinking in C#. So, your mission here is to get comfortable with some basic C# syntax so that I will have your full attention for the next subject.

We are not going to get into all the intricacies of these topics in this chapter. We are going to build a foundation so that you can begin to write a series of programs that I like to refer to as “hello world” programs.

I have also decided for a variety of reasons to expose you to the language though console (command line) programming. One of the reasons I choose this method is that I wanted to show you C# in its purest form. There is a great deal of myth and legend out there that C# is only for Windows or web programming. This clearly is not the case.

BARE BONES PROGRAM (FOR OUR USE NOW)

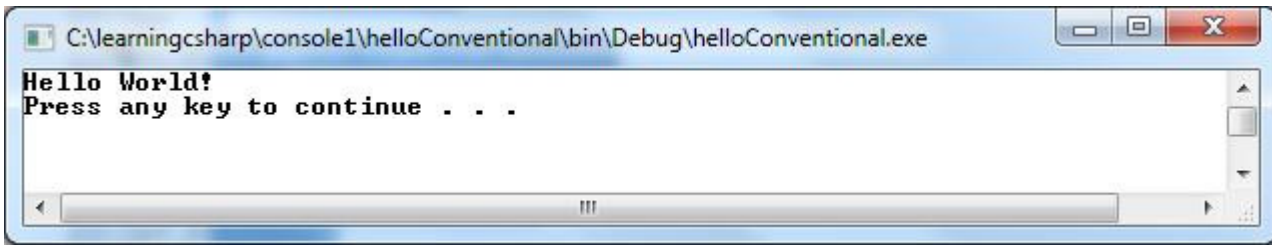
At this point I am not going to explain any of the code contained here. I will later on (I promise). For now, just understand that this is what you will need to write your programs.

```
using System;

class <class_name>
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Hello World!");
    }
}
```

LITERALS

Source HelloConventional.cs

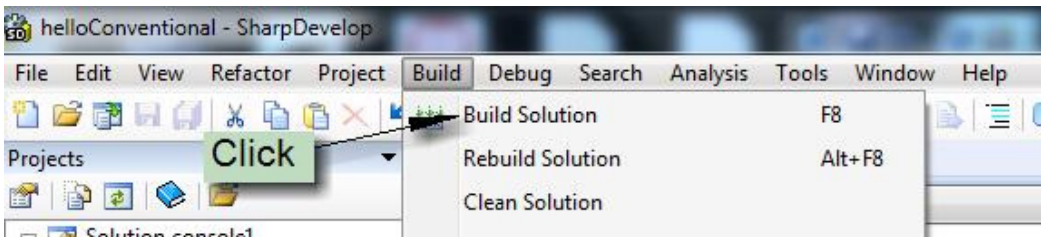


```
using System;

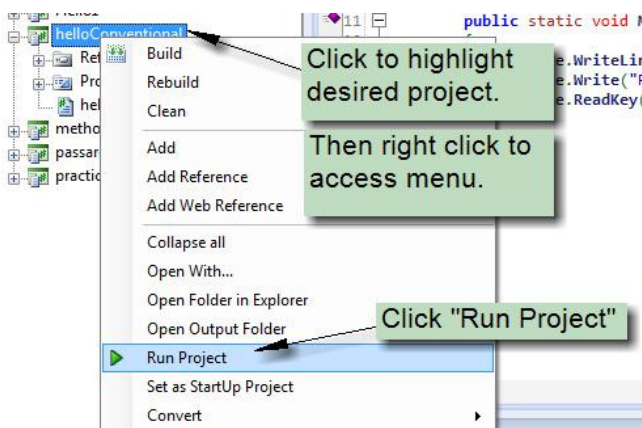
class helloOldStyle
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Hello World!");
        Console.Write("Press any key to continue . . . ");
        Console.ReadKey(true);
    }
}
```

COMPILING YOUR PROGRAM

Please note that this education series is written using SharpDevelop. How to get and install this IDE is covered in the document titled “C#-GettingSetupUsingSharpDevelop.doc.”



RUNNING YOUR PROGRAM



Note: In order to run your console program within SharpDevelop you will need to add the code...

```
Console.WriteLine("Press any key to continue . . . ");  
Console.ReadKey(true);
```

...otherwise the console window will not stay open

Let's go over the syntax.

```
using System;
```

The "using System" directive is required in order to reference the C# api code that allows us to use the console. In fact, this is the only reason for this code why we need to include this directive. Feel free to comment out the directive and see what happens. Then you can comment out the lines the reference "Console" and see that the program will now compile.

The System reference allows us access to a number of classes and methods that we will be using. This is all I am going to say about this for now.

```
class helloOldStyle { }
```

All your code in C# will ultimately be part of a class. We will talk about classes in detail as time progresses.

```
public static void Main(string[] args) { }
```

The Main method is a method that is inherent to every C# class that is created, whether or not you choose to use it. This method is local to the class in which it resides and it is only accessible by that class. If you do not fully understand this point now do not worry.

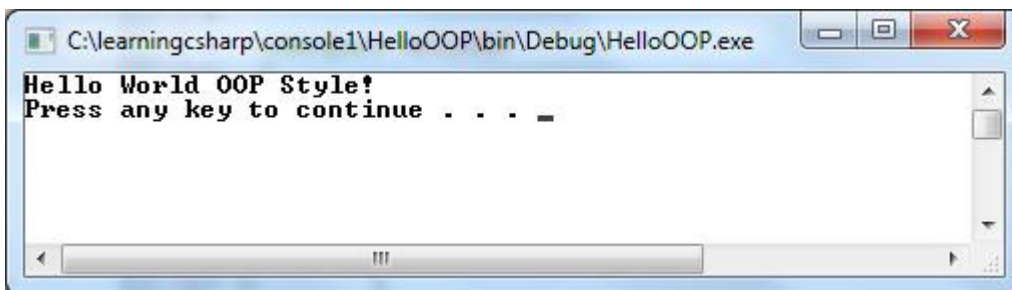
The Main method serves as a nice test wrapper for a class. It also serves us well in our lesson of teaching you the C# language.

```
Console.WriteLine("Hello World!");
```

The Console.WriteLine() method allows you to write data to STDOUT, which in this case is the windows command prompt console.

HELLO THE C# OOP WAY

Source helloOOP.cs



```
using System;

public class run_helloOOP
{
    public static void Main(string[] args)
    {
        helloOOP hi = new helloOOP();
        hi.Greeting();
    }
}

public class helloOOP
{
    public void Greeting()
    {
        Console.WriteLine("Hello World OOP Style!");
        Console.Write("Press any key to continue . . . ");
        Console.ReadKey(true);
    }
}
```

It certainly would seem like a great deal of work to create a hello world program in this manner. If all we were trying to do is echo “Hello World” to the console I would admit that the first program is the most efficient means of doing so.

But good old “hello world” can come in handy for introducing you to some basic oop syntax regarding reuse.

In this program there are two classes

```
public class run_helloOOP
public class helloOOP
```

Both are these classes are declared as public which means they are accessible by anyone. We will cover the notion of public and private later.

If you look at the code in these two classes you will see that the purpose of the first class is to create an instance of the second class. This is accomplished through the statement

```
helloOOP hi = new helloOOP();
```

Since we are creating an instance of the class helloOOP we need to give it a name. I decided on “hi.” I could have named it Fred or pretty much anything I wanted. I prefer to assign meaningful names.

At this point it might be helpful think of a class as document we wish to make a copy of on the copy machine. The class “helloOOP” would represent the original, and the instance of the class “hi” represents the output from the copier.

Now that we have created an instance of our class “helloOOP” we can reference its code. In this case there is not much code to reference.

```
hi.Greeting();
```

Our class “helloOOP” contains one method named Greeting() which contains the code to echo “Hello World” to our console.

Note the syntax

```
<class instance name>.<method>;
```

We will be getting into much more detail regarding oop (object oriented programming) techniques as this course progresses. Our current mission is general syntax familiarization.

VARIABLES

C# supports the following native data types (*Note: This is not a comprehensive list*):

Keyword	Type
char	16 bit unicode
String	Unicode character sequence
bool	true/false
byte	8 bit signed integer
short	16 bit signed integer
int	32 bit signed integer
long	64 bit signed integer
ushort	16 bit unsigned integer
uint	32 bit unsigned integer
ulong	64 bit unsigned integer
float	32 bit unsigned floating point numbers
Double	64 bit unsigned floating point numbers
Decimal	28-29 significant digits

For now I would recommend that you stick with the types I am using in the examples. Later I will get into the intricacies of using these variables types and cover topics like casting.

Source vars1.cs

```
using System;

class dtypes
{
    public static void Main(string[] args)
    {
        String strng = "Hello World";
        char ch = 'Z';
        bool tf = false;
        double dbl = 128.34;

        Console.WriteLine("value of strng is " + strng);
        Console.WriteLine("value of ch is " + ch);
        Console.WriteLine("value of tf is " + tf);
        Console.WriteLine("value of dbl is " + dbl);

        Console.Write("Press any key to continue . . . ");
        Console.ReadKey(true);
    }
}
```

Technically speaking “String” is not a data type, it is a class. However, in terms of application development it is certainly a data type! Frankly in all my years of oop programming and teaching oop I never really saw anyone have a problem learning the language if not exposed to this distinction. I have

seen many looks of bewilderment when trying to explain it at this point in the game. As they get deeper into the game, many folks figure this out on their own.

In analyzing the code keep in mind the following information.

- The equals sign is not an equals sign in C#, it is an assignment operator.
- You should initialize variables that you declare (because it is good practice).
- The char data type can only hold a single character. When initialized the value must be wrapped in single quotes.
- When initializing "String" the value must be wrapped in double quotes.
- A Boolean value is either true or false and is not wrapped in quotes when initialized.
- Integers are initialized to a value that is not wrapped in quotes.

Again, stick with these data types for now. We will cover variable usage in more detail later. We will also cover variable scope later as well.

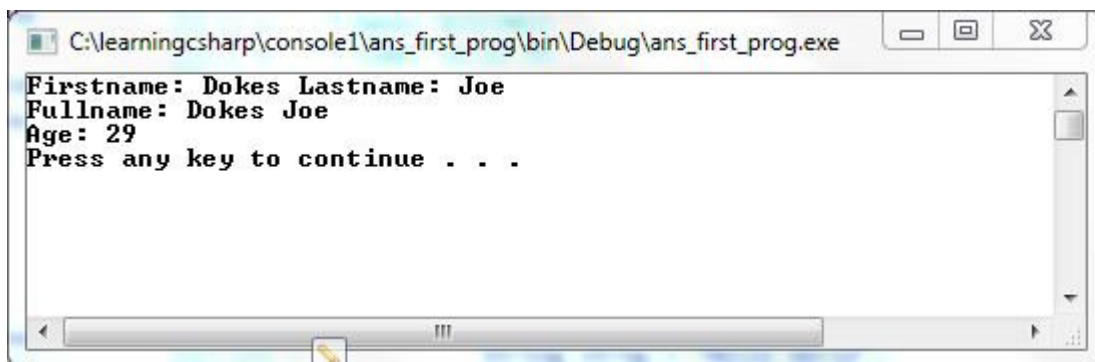
Important note: ... C# is a case sensitive language.

The + sign serves as the concatenation operator in C#. This symbol also doubles as an arithmetic operator. The Java compiler understands the context of when it is used when it tokenizes your code.

PUT INTO PRACTICE – FIRST C# PROGRAM



Using three variables write a C# program that displays "firstname", "lastname", and "age". Your output should look like...



```
C:\learningcsharp\console1\ans_first_prog\bin\Debug\ans_first_prog.exe
Firstname: Dokes Lastname: Joe
Fullname: Dokes Joe
Age: 29
Press any key to continue . . .
```

BASIC OPERATORS

Operator	Meaning
=	Assignment
==	Equals
!=	Not equals

>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

I do not believe any further explanation is required.

CONDITIONAL LOGIC

Of course C# supports if and case logic. “if” statements can be nested, and you can code “if else” logic. I have placed very little narrative in this section as very little should be required.

if

Syntax

```
if (<expression to evaluate>)  
{  
    <statement1>;  
    <optional statement2>;  
    <optional statementN>;  
}
```

The “expression to evaluate” is any expression that evaluates to a Boolean value. If the expression is true the statements get executed, if false they do not. There is nothing new here. What may be new to you is the syntax...

```
if ()  
{  
  
}
```

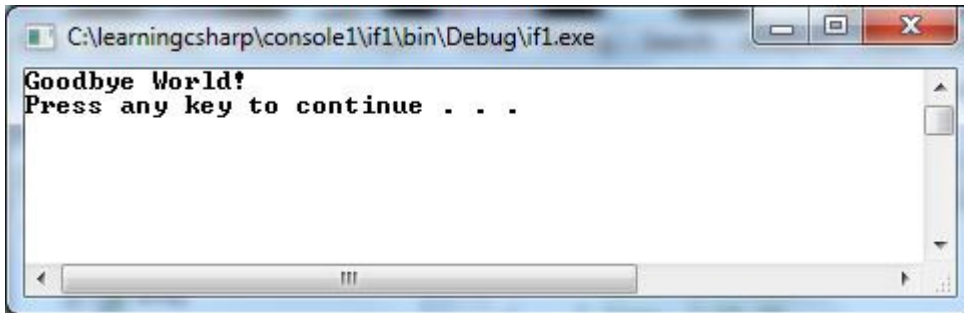
The braces are what’s referred to as a block. If you have programmed in C, C++, Java, Javascript, or Perl then you already know what a block is and you can move on. A block is a very simple concept. It allows you to group a collection of one or more statements together for execution.

On a technical note, if you only have one statement then a block is not required. For example the following two pieces of syntax are equivalent.

```
if (<expression to evaluate>)  
{  
    <statement1>;  
}  
  
if (<expression to evaluate>)  
    <statement1>;
```

While the last “if” is syntactically correct it is good programming style to always include the block.

Source if1.cs



```
using System;

class simpleIf
{
    public static void Main(string[] args)
    {
        String greeting = "Hello World";
        if (greeting == "Hello World")
        {
            Console.WriteLine("Goodbye World!");
        }

        Console.Write("Press any key to continue . . . ");
        Console.ReadKey(true);
    }
}
```

If (no pun intended) you have run the program above and analyzed it you will see that it is pretty straight forward. I would like you to make a copy of this program, change the expression (`strng == "Hello World"`) to (`strng = "Hello World"`). Try to recompile the program. Granted you will make many errors on your own. I did want you to see that substituting the assignment operator for the equals operator produces a compiler error.

```

9
10
11 class simpleIf
12 {
13     public static void Main(string[] args)
14     {
15         String greeting = "Hello World";
16         if (greeting = "Hello World") {
17             Console.WriteLine("Goodbye World!");
18         }
19         Console.Write("Press any key to continue . . . ");
20         Console.ReadKey(true);
21     }
22 }

```

As written, instead of evaluating the expression we are instead attempting to evaluate an assignment statement.

!	Line	Description
✖	14	Cannot implicitly convert type 'string' to 'bool' (CS0029)

Remember the expression in an “if” statement must evaluate true or false. By changing the equal’s operator to an assignment operator you lose your test condition. Assignment is an explicit directive and therefore cannot be evaluated as true or false. I am pointing all this out for people that come from languages where “=” is the equals operator. Some languages allow “=” to be used for both assignment and equivalency. C# does not!

A quick sidebar on indenting

If you have never worked with a language that uses blocks please look at the following two code examples:

One-

```

class simpleIf {
    public static void Main(string[] args){
        String greeting = "Hello World";
        if (greeting == "Hello World"){
            Console.WriteLine("Goodbye World!"); }
            Console.Write("Press any key to continue . . . ");
            Console.ReadKey(true);}}

```

Two-

```
class simpleIf {
    public static void Main(string[] args){
        String greeting = "Hello World";
        if (greeting == "Hello World") {
            Console.WriteLine("Goodbye World!");
        }
        Console.Write("Press any key to continue . . . ");
        Console.ReadKey(true);
    }
}
```

The first example is not real easy to read. It is easy to forget that others will be reading your code. You should strive to write your code in a manner that takes into account that others will be reading it.

Example two is much better. Example two is also the method that many code generators use to produce code. I find this method acceptable. I prefer to line up my opening and closing braces with one another. I find this method more readable when looking at code where there are a lot of nested if's and nested loops.

These two programs are identical to "if1.cs". They all compile and produce identical object code. There is no compelling reason that I can think of to write your program like example one. There are many good ways to write readable code. I am not laying claim to one way. However, I am suggesting that you make your code readable.

If else

Syntax

```
if (<expression to evaluate>){
    <statement1>;
    <optional statement2>;
    <optional statementN>;
}
else {
    <statement1>;
    <optional statement2>;
    <optional statementN>;
}
```

Source simplelfelse.cs

```
using System;

class simple_ifelse
{
    public static void Main(string[] args)
    {
        Console.Write("Enter x to accept \n");
        string strarg0 = Console.ReadLine();
        if (strarg0 == "x") {
            Console.WriteLine(strarg0 + " has been accepted");
        } else {
```

```

        Console.WriteLine("Sorry, " + strarg0 + " is not accepted");
    }
    Console.Write("Press any key to continue . . . ");
    Console.ReadKey(true);
}
}

```

If you are new to programming you should now understand that an if statement by itself only takes actions if the condition being tested is true. Whereas the if/else statement takes action if the statement is true or false.

We have even more flexibility when it comes to conditional logic.

Nested if

Syntax

```

if (<expression to evaluate>)
{
    <statement1>;
    <optional statement2>;
    <optional statementN>;
    if (<expression to evaluate>)
    {
        <statement1>;
        <optional statement2>;
        <optional statementN>;
    }
}

```

Source nestedif.cs

```

using System;

class nestedif
{
    public static void Main(string[] args)
    {
        Console.Write("Enter a number 1 to 10 \n");
        int strarg0 = Int32.Parse(Console.ReadLine());
    }
}

```

```
        Console.WriteLine("Enter another number 1 to 10 \n");
        int strarg1 = Int32.Parse(Console.ReadLine());
        Console.WriteLine("Enter either x or + \n");
        string strarg2 = Console.ReadLine();
        int result;

// evaluate first arg entered
    if (strarg0 > 0 && strarg0 < 11) {
// if ok, evaluate second arg entered
        if (strarg1 > 0 && strarg1 < 11) {
// if ok, evaluate third argument
            if (strarg2 == "x" || strarg2 == "+") {
                if (strarg2 == "x" )
                {
                    result = (strarg0 * strarg1);
                    Console.WriteLine("Answer is " + result);
                }
                else
                {
                    result = (strarg0 + strarg1);
                    Console.WriteLine("Answer is " + result);
                }
            }
//argument 3 entered incorrectly
        } else {
            Console.WriteLine("Sorry, " + strarg2 + " is not a + or x");
        }
//argument 2 entered incorrectly
    } else {
        Console.WriteLine("Sorry, " + strarg1 + " is not acceptable input");
    }
//argument 1 entered incorrectly
} else {
    Console.WriteLine("Sorry, " + strarg0 + " is not acceptable input");
}

    Console.WriteLine("Press any key to continue . . . ");
    Console.ReadKey(true);
}
}
```

Even though this is supposed to be toycode, I chose this example to demonstrate for new programmers the complexity of logic. All we have going is a command line program that accepts three arguments, two numbers and third argument which will either add or multiply the two numbers. You can see the complexity of the code above in just dealing with three command line input arguments. It is that way as the goal of the program was also to provide error handling from the user interface.

We can debate whether this program is written in the best way. You would not need a very persuasive argument to convince me that it is not. However, there are times in your program where you end up with nested logic like this as it is the best means to get the job done.

The program first collects the three input parameters...

```
    Console.WriteLine("Enter a number 1 to 10 \n");
    int strarg0 = Int32.Parse(Console.ReadLine());
```

```

Console.Write("Enter another number 1 to 10 \n");
int strarg1 = Int32.Parse(Console.ReadLine());
Console.Write("Enter either x or + \n");
string strarg2 = Console.ReadLine();

```

We are going to be doing arithmetic operations on the first two input parameters. As such we need to cast this input to the appropriate data type. This is why you see

```
int strarg1 = Int32.Parse(Console.ReadLine());
```

Int32.Parse() method takes the input from the console and casts it as an integer. While we are here it should be noted that while my goal was to error handle the user input, if the user enters a non-numeric value or a number with a decimal a runtime error will occur. I did not trap for these conditions as I felt it would take us too far astray from the missing of demonstrating nested conditionals.

Continuing on, if you examine the code closely you will see that that it is three if/else statements nested inside of each other.

The first if evaluates the first number input. If it evaluates to true we hit the first nested if which evaluated the second number. Of note,

```
if (strarg0 > 0 && strarg0 < 11)
```

The && represents the and operator. For this conditional to be true both statements must be true. If false, we display an input error message. The second if/else block evaluates the second number. If it is true we head to the last nested if/else block. If not we display an input error message.

Our last nested if statement evaluates the third input argument which is asking for a + or x. Again, of note,

```
if (strarg2 == "x" || strarg2 == "+")
```

The || represents the or operator. For this conditional to be true only one of the statements must be true

There is no limit to the number of “and” or “or” operators you can string together. As such, it is important you understand the truth table below so you avoid logic errors in your code. Also, if you have compile and/or in your code make sure you appropriately use parenthesis in your statement.

&& (AND) Truth Table

Arg1	Arg2	...ArgN	Statement evaluates
True	True	True	True
True	True	False	False
True	False	True	False
True	False	False	False
False	True	True	False
False	False	False	False
Etc, etc...			

The guiding principle here is that will AND logic all the conditions must be true in order for the statement to be true.

|| (OR) Truth Table

Arg1	Arg2	...ArgN	Statement evaluates
True	True	True	True
True	True	False	True
True	False	True	True
True	False	False	True
False	True	True	True
False	False	False	False
Etc, etc...			

The guiding principle here is that with OR logic only one of the conditions must be true in order for the statement to be true.

For most of you this is old hat. But, for new programmers this is something that is not well covered anymore. When I was in school in the 60's this was part of what they called this new math. Of course even back then it was not all that new, just newly accepted.

Code like this can sometimes ignite debate as to the best approach for writing code. For me this subject tends to be a slippery slope. My perspective is as follows.

The problem you are tasked to solve and the way your mind thinks really dictate how you need to write your code. I have taught enough programming classes to have observed that some problems I gave students to code would result in valid coding variations that were virtually endless. Other problems would be solved by the masses in essential the same way. I must confess that I spent years trying to figure out if there was a patten to what I observed and was never able to find one.

It is also important to approach coding the way your mind thinks. What is ultimately important is writing code that...

- Is easy for others to maintain
- Lends itself as well as possible to fixes and enhancements

Of course this is all highly subjective stuff we are talking about. But, I can tell you that in all my years working in development I have witnessed way too much time being wasted by people trying to dictate standards for how conditional logic and loops and other structures should be written. This is certainly a debate that will not end anytime soon.

if else if

Syntax

```
if (<expression to evaluate>)\n{\n    <statement1>;\n    <optional statement2>;\n    <optional statementN>;\n}\nelse if (<expression to evaluate>)\n{\n    <statement1>;\n}
```

```
    <optional statement2>;  
    <optional statementN>;  
}
```

Source ifelseif.cs

```
using System;  
  
class Program  
{  
    public static void Main(string[] args)  
    {  
        Console.WriteLine("Enter a, b, or c \n");  
        string strarg0 = Console.ReadLine();  
        if (strarg0 == "a") {  
            Console.WriteLine("you entered a");  
        }  
        else if (strarg0 == "b") {  
            Console.WriteLine("you entered b");  
        }  
        else if (strarg0 == "c") {  
            Console.WriteLine("you entered c");  
        }  
        else {  
            Console.WriteLine("you did not enter a,b,c you entered " + strarg0);  
        }  
  
        Console.WriteLine("Press any key to continue . . . ");  
        Console.ReadKey(true);  
    }  
}
```

This type of logic can also be handled using “switch”, which some believe to be a much more eloquent solution. I prefer to use if/else if logic as it allows for greater latitude in coding.

switch (Case logic)**Syntax**

```
switch(<expression>  
{  
    case <case_constant1> :  
        <statement1>;  
        <optional statement2>;  
        <optional statementN>;  
        break;  
    case <case_constant2> :  
        <statement1>;  
        <optional statement2>;  
        <optional statementN>;  
        break;  
  
    case <case_constantN> :  
        <statement1>;  
        <optional statement2>;
```

```
<optional statementN>;
break;

default:
  <statement1>;
  <optional statement2>;
  <optional statementN>;
  break;
}
```

Source switch1.cs

```
using System;
class switch1
{
    public static void Main()
    {
        Console.WriteLine("Enter one of the following numbers: 1, 2, 3");
        Console.Write("Please enter your selection: ");
        string s = Console.ReadLine();
        int n = int.Parse(s);
        int cost = 0;
        switch(n)
        {
            case 1:
                Console.WriteLine("You entered 1");
                break;
            case 2:
                Console.WriteLine("You entered 2");
                break;
            case 3:
                Console.WriteLine("You entered 3");
                break;
            default:
                Console.WriteLine("Invalid selection. Please select 1, 2, or 3.");
                break;
        }

        Console.Write("Press any key to continue . . . ");
        Console.ReadKey(true);
    }
}
```

Most languages implement switch or case logic in a different manner. There tend to be many quirks and limitations involving its use. Whereas “if” logic tends to be more flexible and consistent across languages. Case in point, “if” logic in C# and Java is pretty much syntactically the same.

Some implementations of switch allow for linear fall through of the code (Java does). In other words a switch statement can have more than one true outcome. C# does not allow for linear fall through in a case statement. Frankly this is not a big deal and can be accommodated in code with multiple individual conditional blocks.

Limitations to coding conditional logic in C#?

As long as you stay within the syntax of the “if” statements you can code some pretty complex logic. The point of the following is not for you to try to figure out the logic (or illogic), nor is it to advocate writing convoluted code

PUT INTO PRACTICE – CONDITIONAL LOGIC



Write a simple program of your choosing that demonstrates your understanding of using conditional logic and anything else you have learned thus far. This is a free form exercise.

MORE ABOUT STRING

Authors note (and peeve): In structured programming you can create functions or subroutines. In oop you can create methods. The terms function, subroutine, and methods all mean the same thing. The terms are interchangeable. So, if you are coming into this from a structured programming background when I use the term method you can think function (or subroutine).

It is now time to delve deeper into the “string” class.

Most structured programming languages provide intrinsic functions for working with character data (trim, substring, etc...) . Well so do oop languages like C#. Use of such methods is slightly different.

Source stringMethods.cs

```
string a = "  foobar  ";

Console.WriteLine("Length of our string: " + a.Length);
//change to uppercase
    Console.WriteLine(a.ToUpper());
    //back to lowercase
    Console.WriteLine(a.ToLower());
    //Console.WriteLine(a.StartsWith());
    a = a.TrimStart();
    Console.WriteLine(a);
    Console.WriteLine("New Length of our string: " + a.Length);
    a = a.TrimEnd();
    Console.WriteLine(a);
    Console.WriteLine("New Length of our string: " + a.Length);
// can also use Trim() to trim both leading and trailing whitespace
// get first three characters of our string
    Console.WriteLine(a.Substring(0,3));
// get last three characters of our string
    Console.WriteLine(a.Substring(3,3));
    // use StartsWith() to capitalize the first letter
    if (a.StartsWith("foo")) {
        a = a.Substring(0,1).ToUpper() + a.Substring(1,5);
        Console.WriteLine(a);
    }
//i want to show you the split function so i am going to insert a comma
```

```
//into the string so we have something proper to split on.
a = a.Substring(0,3) + "," + a.Substring(3,3);
Console.WriteLine("a is now: " + a);
string[] str_arr = new string[2];
str_arr = a.Split(',');
Console.WriteLine("str_arr[0] is: " + str_arr[0] );
Console.WriteLine("str_arr[1] is: " + str_arr[1] );
//using the replace method
a = a.Replace(",","");
Console.WriteLine("a is now: " + a);
string b = "Foobar";
//testing a string against another string
if (a.Equals(b)) {
    Console.WriteLine("b is: " + b);
    Console.WriteLine("a.Equals(b) is true");
}
```

Declaring and initializing a String

```
string <name> = " foobar ";
```

Determining the length of a String

```
<name>.Length
```

String case conversion

```
<name>.ToUpper()
<name>.ToLower()
```

Trim

```
<name> = <name>.TrimStart(); //trim leading spaces
<name> = <name>.TrimStart('0'); //trim leading zeros from a string
<name> = <name>.TrimEnd(); //trim trailing spaces
<name> = <name>.TrimEnd(); //trim all leading and trailing spaces
```

Split a String

```
str_arr = <name>.Split(',');
```

String to a character array

```
string[] str_arr = new string[2];
str_arr = <name>.Split(',');
```

String Modification

```
<name> = <name>.Replace(",","");
```

Compare Two Strings

```
string b = "Foobar";  
if (<name>.Equals(b)) { }
```

ARRAYS IN C#

A C# array once dimensioned cannot be re-dimensioned. If you need an array in C# that is dynamic you use the List<type> class. The code and comments in the code should tell the story without cause for additional narrative.

Source array.cs

```
using System;  
  
class array  
{  
    public static void Main(string[] args)  
    {  
        //create fixed array, set values  
        int[] int_arr = { 0, 1, 2 };  
  
        Console.WriteLine("Array size is: " + int_arr.Length);  
        Console.WriteLine("int_arr[0]=" + int_arr[0] +  
            " int_arr[1]=" + int_arr[1] +  
            " int_arr[2]=" + int_arr[2]);  
  
        //trying to redimension the array will result in a run time error.  
        //comment out code below to see  
        //int_arr[3] = 3;  
  
        //create array of specified size, set values later  
        string[] str_arr = new string[3];  
        str_arr[0] = "Fee";  
        str_arr[1] = "Fi";  
        str_arr[2] = "Fo";  
        Console.WriteLine("str_arr[0]: {0}, str_arr[1]: {1}, str_arr[2]: {2}",  
            str_arr[0], str_arr[1], str_arr[2]);  
  
        Console.Write("Press any key to continue . . . ");  
        Console.ReadKey(true);  
    }  
}
```

Initialize Array and Set values (thus determining size)

```
int[] <name> = { 0, 1, 2 };
```

Initialize Array to Specific Size, Set values later

```
string[] str_arr = new string[3];  
str_arr[0] = "Fee";  
str_arr[1] = "Fi";  
str_arr[2] = "Fo";
```

Get # of Elements in Array

```
<name>.Length;
```

DYNAMIC ARRAYS IN C#

Being locked in to declaring the array size at design time can be limiting. For example, you may need to take the results of a query, or web service where the size is unknown to you. Having to declare the size of your array in this scenario can pose a real problem.

Source dynamic_array.cs

```
using System;
using System.Collections.Generic;

class dynamic_array
{
    public static void Main(string[] args)
    {
        List<string> dynarr = new List<string>();
        dynarr.Add("Fee");
        dynarr.Add("Fi");
        dynarr.Add("Fo");

        Console.WriteLine(dynarr.Count);
        dynarr.Add("Fum");
        dynarr.Add("Fum"); //see this is not a hash
        dynarr.Add("Fum");
        //get an individual element
        Console.WriteLine(dynarr[3]);
        for(int i =0; i < dynarr.Count; i++)
        {
            Console.WriteLine(dynarr[i] + " dynarr[" + i.ToString() + "]");
        }

        dynarr.Remove("Fum");
        Console.WriteLine("Remove element and Loop through Again");
        for(int i =0; i < dynarr.Count; i++)
        {
            Console.WriteLine(dynarr[i] + " dynarr[" + i.ToString() + "]");
        }
        Console.WriteLine("See, only first occurrence was removed. Array gets
redimensioned.");
        for(int i =0; i < dynarr.Count; i++)
        {
            Console.WriteLine(dynarr[i] + " dynarr[" + i.ToString() + "]");
        }
        // now let's remove an array element by indice
        dynarr.RemoveAt(4);
        // reverse array order
        dynarr.Reverse();
        Console.WriteLine("Reverse the order of the array");
        for(int i =0; i < dynarr.Count; i++)
        {
            Console.WriteLine(dynarr[i] + " dynarr[" + i.ToString() + "]");
        }
    }
}
```

```
    }
    // sort array
    dynarr.Sort();
    Console.WriteLine("Sort the array");
    for(int i =0; i < dynarr.Count; i++)
    {
        Console.WriteLine(dynarr[i] + " dynarr[" + i.ToString() + "]");
    }

    Console.Write("Press any key to continue . . . ");
    Console.ReadKey(true);
}
}
```

Notice that we added a new reference in the code.

```
using System.Collections.Generic;
```

This gives us access to the List class. If you are already familiar with hashes (associative arrays) you might think this class is a hash on first glance. It is not.

Create a List

```
List<string> <name> = new List<string>();
```

Getting a Count of List Elements

```
<name>.Count
```

Adding a List Element

```
<name>.Add("Some text or a string variable here");
```

Removing a List Element

```
<name>.Remove("Some text or a string variable here");
```

Sorting a List

```
<name>.Sort();
```

Reversing a List

```
<name>.Reverse();
```

LOOPS

The following two programs demonstrate loops in C#.

for

Syntax

```
for (<loop initializer>; <loop test condition>; <increment loop>)
{
    <statement1>;
    <optional statement2>;
    <optional statementN>;
}
```

Source for_while_loops.cs

```
for (int i=0; i < 10; i++) {
    Console.WriteLine("{0}", i);
}
```

If you have programmed in C, C++, or Perl then you are familiar with this loop. If not, then the syntax for this loop may look a bit different to you. This loop offers some pretty terse syntax. The first part of the loop is the initializer. In the program above we have declared a variable and set its value to one in the loop. The next statement is the test condition for our loop. The loop will execute until the value of “i” is 10. Finally, we need a mechanism for advancing the loop. We do this by applying the iteration operator (++) to our initialized variable.

In the loop above we initialized our variable when we declared it. We iterate the loop within the loop itself. This works the same as the first loop. We could even have written

Increment operator

i++

as

i = (i + 1)

If you were fuzzy on the iteration operator you should now understand its function based on the last statement. I added these code variations to the loop to help clarify how the for loop works as its syntax can be strange if viewed for the first time. I would strongly recommend that you code for loop's the way I first presented as this is the accepted standard.

Note: The iteration operator can be used anywhere in your code. PS, as a note of trivia the language C++ got its name because it was an “iteration” of C. That and fifty cents won't come close to buying you a latte.

Decrement operator

```
int = 9;

i--

//i is now 8
```

while

Syntax

```
while (<expression is true>)
{
    <statement1>;
    <optional statement2>;
    <optional statementN>;
}
```

Source for_while_loops.cs

```
while (x < 10) {
    Console.WriteLine("{0}", x);
    x++;
}
```

This loop executes until the expression is false.

do

Syntax

```
do
{
    <statement1>;
    <optional statement2>;
    <optional statementN>;
} while (<expression is true>;);
```

Source practical_dowhile_loop.cs

```
using System;

class practical_dowhile_loop
{
    public static void Main(string[] args)
    {
        string ui_sel;

        do
        {
            Console.WriteLine("Dummy Menu\n");
```

```
        Console.WriteLine("-----\n");

        Console.WriteLine("A - Menu Item A");
        Console.WriteLine("B - Menu Item B");
        Console.WriteLine("C - Menu Item C");
        Console.WriteLine("D - Menu Item D");
        Console.WriteLine("E - Menu Item E");
        Console.WriteLine("Q - Quit\n");

        Console.WriteLine("Choices: A,B,C,D,E or Q to Exit: ");

        // get ui
        ui_sel = Console.ReadLine();

        // case logic
        switch (ui_sel)
        {
            case "A": case "a":
                Console.WriteLine("You selected menu item A");
                break;
            case "B": case "b":
                Console.WriteLine("You selected menu item B.");
                break;
            case "C": case "c":
                Console.WriteLine("You selected menu item C.");
                break;
            case "D": case "d":
                Console.WriteLine("You selected menu item D.");
                break;
            case "E": case "e":
                Console.WriteLine("You selected menu item E.");
                break;
            case "Q": case "q":
                Console.WriteLine("Later.");
                break;
            default:
                Console.WriteLine("{0} is not a choice",ui_sel);
                break;
        } //end switch

        // refresh ui after selection
        Console.Write("Press Enter to continue...");
        Console.ReadLine();
        Console.WriteLine();
    } while (ui_sel != "Q" && ui_sel != "q"); // iterate until q

    } //end do loop

} //end class
```

The difference between the “do” loop and the “for” and “while” loops is that the “do” loop will always execute at least one time.

nested loops

I have written a lot of different types of parsers as well as code generation tools. I find myself writing nested loops on these types of projects. Nested loops can be a tricky business. I have had many sustaining engineers tell me I will be the one fixing my code if it breaks or needs enhancement in the areas where I have nested loops. I had one engineering team come ask me if it was okay to break my nested loop code out into three functions (it was a triple nested loop). I said “no problem.” I am a staunch advocate that a programmer needs to code the way they think. Not everyone thinks in nested loops. (Unfortunately?) I do.

Below is a rather trivial example of a nested loop.

Source nested_for_loop.cs

```
using System;

class nestedloop
{
    public static void Main(string[] args)
    {
        for (int i=5; i > 0; i--) {
            Console.WriteLine("\nouter loop number is: " + i);
            for (int a=0; a < 5; a++) {
                Console.WriteLine("\n    inner loop number is: " + a);
            }
        }

        Console.WriteLine("\n\nPress any key to continue . . . ");
        Console.ReadKey(true);
    }
}
```

PUT INTO PRACTICE – LOOPS AND OTHER STUFF WE HAVE COVERED UP UNTIL NOW



A palindrome is a word or sentence that is spelled the same backwards or forwards. For example “level” is a palindrome. Write a program that can distinguish these and that works like the one depicted below.

```
arr[0] = l l
arr[1] = e e
arr[2] = v v
arr[3] = e e
arr[4] = l l
We have a palindrome
```

```
arr[0] = g p
arr[1] = e l
arr[2] = t e
arr[3] = h
arr[4] = h
arr[5] = e t
arr[6] = l e
arr[7] = p g
We do not have a palindrome
```

If you are already an experienced programmer this exercise should be pretty easy for you, but will allow you to exercise all apply a great deal of what has been covered. If you are new to programming you will likely find this exercise a challenge.

COMMENT CODE IN C#

Up to now the programs have been trivial enough that they did not require any comment codes. I do not want to introduce comment codes to you.

Single line comments

Syntax

```
// <comment>

// comment line

if (myvar > 30)    //comments can come after code
```

Multiple line comments

Syntax

```
/*
<comments can go anywhere in between>
*/

/*
this is the
type of comment code that
can span
multiple
lines
*/
```

A FINAL WORD ON THIS SUBJECT

You should now have enough familiarity with basic syntax of C# to move into the next subject. Make sure you are comfortable with this material. The next subject is going to expose you to the world of Object Oriented Programming (OOP). This is going to be a new way of thinking about programming for you so you want to be sure that you are comfortable with the subject matter.