Common Variable Declarations

```
const string CLASS_NAME="clsOrder";  //Constant
bool bNewCustomer=true;     //Boolean
char cKeystroke;            //16 bit Unicode Character
int iCount= 0;              //Signed 32-bit integer
long lOrderNumber=0;        //Signed 64-bit integer
double dblOrderTotal=0;     //Double-precision floating-point
single fProductMarkup=0;    //Single-precision floating-point
string sCustomerName = "";  //String

DateTime dtNow= DateTime.Now;   //Current Date
DateTime dtTest = new DateTime(2004,1,23,13,48,39);
// (year, month , day, hours, minutes, seconds)

double[] maLocation;            //Array
maLocation = new double[iMaxRange];  //Dimensioning
```

Loops

```
//While Loop
var i = 1;
while ( i <= 10 )
{     i++;   }

//For Loop
for (int i = 1; i <= 10; i++ )
{
      if (bFound) break;
}

//For Each Loop
int [] ai = {1, 2, 3, 4};
foreach(int i in ai)
{     Console.WriteLine(i); }

//Do Loop
var i = 0;
do
{
      i+=2;
} while(i<20);
```

Error Checking

```
try
{
   iSomeValue= 10/0;
}  //"Catch" the error
catch (Exception e)
{

Console.WriteLine(
      e.Message);
} //Always do this
finally
{
   fInputFile.Close()
}
```

Struct/Classes

```
struct Point {public
      int x, y;}

class MyClass{}
```

Procedure Declaration

```
private double CalcOrderTotal(oOrder)
{
   return oOrder.OrderTotal * dblMarkup;
}
```

Property Declaration

```
public double CustomerName
{
   get
   {
      return msCustomerName;
   }

   set
   {
      msCustomerName= value;
   }
}
```

Branching

```
if ( (x == 1) && (y == 3) )
     sum = y - x;
else
     sum=0;

switch(s)
{
   case "A":  case "B":
      Console.WriteLine("A or B Chosen");
      break;

   case "C":
      Console.WriteLine("C Chosen");
      break;

   /* if all other conditions fail do this
   this is a multi-line comment */
   default;
      Console.WriteLine("Other Chosen");
      break;
}
```

Operators

Computational

-	Negation/Subtraction
++	Increment
--	Decrement
*	Multiplication
/	Division
%	Modulo arithmetic
+	Addition

Logical

!	Logical NOT
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equality
!=	Inequality
&&	Logical AND
||	Logical OR
?:	Conditional (unary)
,	Comma

Assignment

=	Assignment

Compound assignment operators

+=	Addition
-=	Subtraction
*=	Multiplication
/=	Division

Escape Sequences

```
string sInputFile =
      "\\\\server\\fileshare\\in.txt";
string sOutputFile =
      @"\\server\fileshare\out.txt";
```

Access Modifiers

public, internal, private, protected, protected internal

Regular Expressions

```
using System.Text.RegularExpressions;

private static bool ValidAMEX(string sCardType, string sCardNumber)
{
      // AMEX -- 34 or 37 -- 15 length
      if ( (Regex.IsMatch(sCardNumber,"^(34|37)"))
            && (sCardType == "AMEX") )
            return (15==sCardNumber.Length);
}

private static string Filter(string sValue, string sFilter)
{
   string sMask = @sFilter;
   return Regex.Replace(sValue,sMask, "");
}
```

Value/Reference/Output Only/Dynamic Parameters

```
public static void CalcTotal(double dblSubTotal,          // ←Value
                             ref double dblFinanceCharge, // ←Reference
                             out double dblOrderTotal,    // ← Out Only
                             params int[ ] iarr )         // ← Dynamic
{}
```

Enumeration

```
public enum Direction {North, East, West, South}
```

API Usage

```
[DllImport("kernel32")]
private static extern long GetDiskFreeSpaceEx(string sDrive,
   ref long iFreeBytes,
   ref long iTotalBytes,
   ref long iTotalFree);
```

Conversion

Byte[] myBytes = **System.Text.Encoding.ASCII.GetBytes**(sInString);

GAC Location

C:\WINNT\assembly\GAC

C# Web References

http://www.codeproject.com
http://www.dotnet247.com
http://www.csunit.org
http://www.ellkay.com/ConvertVB2CSharp.htm
http://www.c-sharpcorner.com
http://www.gotdotnet.com
http://www.devcity.net/net
http://sourceforge.net/projects/nunit
http://www.regexlib.com