Today's Topics:

- What is C#?
- Types of application in C#
- What is a variable?
- Data types in C#
- Operator in C#
- Language constructs
- Type Casting

C# was introduced by Andrew Helsibery and it is developed using C and C++.
C# is the combination of C and C++.
Features of C#:

1. Simple
2. Modern
3. Robust (we can develop small application to large applications)

Visual Studio .NET provides a common development environment to develop all types of .NET applications weather they are C# based or Visual Basic.NET based.

This simulation (model teaches you to develop C# application.

In C# simulations, two types of application are discussed which can be created using templates. These are the console based applications and the window based applications.

In this simulation, you are taught how to create a C# project using the console template. The console application is the dos-based application, which does not have any graphical user interface. It is completely command based.

Select

start> Run> DEVENV (Developer Environment)
Or
Start>All Programs> Microsoft Visual Studio 2010> Microsoft Visual studio 2010

File>New> Project or Ctl+Shift+N

Project type:
Visual C#
Windows
Templates
Console Application

Name	<Name of the Project>
Location	<Location of the Project>
Solution name	<by default it tables name of the project>

Structure of console Application:

```
using <Namespace>;
using <Namespace>;
using <Namespace>;

namespace <namespace name>
{
            <Access Specifier> class <class name>
            {
                        <Access Specifier> <return type> Main(String[] args)
                        {

                        }
            }
}
```

Comment [S1]: Directives

Comment [S2]: Public
Private
Protected
Scaled
Abstract

- Every console application makes use of system namespace because the console class is available under system namespace.
- Every console application has a main method from where the execution of the program takes place.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            //To Do: Add code to start Application here
        }
    }
}
```

Comment [S3]: The Opening Curly brace indicates the beginning of the definition of the namespace.

Comment [S4]: Program is the name with which a new class is created in C# programs. This name can be changed later on according to the functions the program is performing.

Comment [S5]: The Open Curly brace signifies the beginning of the definition of the class.

Comment [S6]: This comment block might be replaced with the relevant code for the program.

Comment [S7]: The Closing curly brace signifies the end of the program.

Comment [S8]: The closing brace signifies the end of the namespace

Visual studio automates same code generation for the C# console programs by creating the basic structure of the program. This code corresponds to the **Template** selected earlier.

The class contains the **Main** method. The main method is the block of the code without which a c# program cannot run because it is the execution point of the C# program.

Here the skeleton **Main** method that is the main method without any code is inserted by default in the program. This code can be replaced with the code required of the program.

C# is highly case sensitive.

Example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Welcome to C# Programming");
         Console.ReadLine();
         Console.ReadKey();

      }
   }
}
```

Comment [S9]: The program will terminate only when we press enter button.

Comment [S10]: The program will terminate when we press any key

Folder structure of a console application:

```
<Project Name>
            ProjectName
                        Bin
                                    Debug
                                    Release
                        Object
                                    Debug
                                    Release
```

File Structure

```
Sln is an solutionfile
            .csproj(csharp project)
                        .cs(C# File)
                        . cs(C# File)
                        .cs(C# File) and so on
```

Operators in C#

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C# is rich in built-in operators and provides the following type of operators:

1. Arithmetic Operators

%	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200

/	Divides numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator increases integer value by one	A++ will give 11
--	Decrement operator decreases integer value by one	A-- will give 9

2. Relational Operators

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

3. Logical Operators

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

4. Bitwise Operators

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume if A = 60; and B = 13; now in binary format they will be as follows:
A = 0011 1100
B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101
A^B = 0011 0001
~A = 1100 0011

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12. which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61, which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49, which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61, which is 1100 0011 in 2's complement due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240, which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15, which is 0000 1111

5. Assignment Operators

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2

&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

6. Misc Operators

Operator	Description	Example
sizeof()	Returns the size of a data type.	sizeof(int), will return 4.
typeof()	Returns the type of a class.	typeof(StreamReader);
&	Returns the address of an variable.	&a; will give actual address of the variable.
*	Pointer to a variable.	*a; will pointer to a variable.
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y
is	Determines whether an object is of a certain type.	If(Ford is Car) // checks if Ford is an object of the Car class.

as	Cast without raising an exception if the cast fails.	Object obj = new StringReader("Hello");
		StringReader r = obj as StringReader;

Variable

A variable is an identifier whose value varies according to execution of the program while defining the variables the following points are to be considered.

- It should always begin with a character or underscore.
- It shouldn't begin with number or special characters, but it can be used at middle.
- It shouldn't contain any space.
- It shouldn't be any reserved word or keyword identified by C# compiler.
- The maximum length of a variable name can be of any length.

Note:
All variable names defined in C# highly case sensitive.

Syntax:
<datatype> <variable_name>,<variable_name>..
Example:
```
int x,y;
float a,b;
```
Declared and initialize a variable:
<datatype> <variable>=<value>
Ex: int x=10;

Language Constructs:

if
```
If(expression)
{
        statement(s)
}
```

If..else

```
If(expression)
{
        statement(s)
}
else
{
        statement(s)
}
```

If..else if –else

```
If(expression)
{
        statement(s)
}
else if
{
        statement(s)
}
else
{
        statement(s)
}
```

Switch:

```
switch(expression)
{
        case <value>:
        {
                statement(s)
                break;
        }
        case <value>:
        {
                statement(s)
                break;
        }
        case <value>:
        {
                statement(s)
                break;
        }
}
```

Loops or Interactive Control strcutures:

1. **while Loop**

```
    while(expression)
    {
        statements(s)
    }
```

2. **do-while loop**

```
    do
    {
        statements(s)
```

```
}while(expression)
```

3. **for loop**

```
for(initialization; condition; incrementation)
{
    statements(s)
}
```

4. **for-each loop**

```
foreach(<variable> in <collection>)
{
    Statement(s)
}
```

Comments

1. Single-line comments

```
//This is a sample program
```

2. Multiline comments

```
/*This
 is
a
sample
program*/
```

Data types in C#

CTS(Common Type System)	Type	Represents	Range	Default Value
System.Boolean	bool	Boolean value	True or False	False
System.Byte	byte	8-bit unsigned integer	0 to 255	0
System.Char	char	16-bit Unicode character	U +0000 to U +ffff	'\0'
System.Decimal	decimal	128-bit precise decimal values with 28-29 significant digits	(-7.9 x 10^{28} to 7.9 x 10^{28}) / $10^{0 \text{ ato } 28}$	0.0M
System.Double	double	64-bit double-precision floating point type	(+/-)5.0 x 10^{-324} to (+/-)1.7 x 10^{308}	0.0D
System.Single	float	32-bit single-precision floating point type	-3.4 x 10^{38} to + 3.4 x 10^{38}	0.0F
System.Int32	int	32-bit signed integer type	-2,147,483,648 to 2,147,483,647	0
System.Int64	long	64-bit signed integer type	-923,372,036,854,775,808 to 9,223,372,036,854,775,807	0L
System.sbyte	sbyte	8-bit signed integer type	-128 to 127	0
System.int16	short	16-bit signed integer type	-32,768 to 32,767	0
System.uint32	uint	32-bit unsigned integer type	0 to 4,294,967,295	0
System.uint64	ulong	64-bit unsigned integer type	0 to 18,446,744,073,709,551,615	0
System.uint16	ushort	16-bit unsigned integer type	0 to 65,535	0

Arithmetic Operator Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ArithmeticOperations
{
    class Program
    {
        static void Main(string[] args)
        {
            int A, B, SUM;
            Console.WriteLine("Enter Two Integers");

            A = int.Parse(Console.ReadLine());

            B = int.Parse(Console.ReadLine());
            SUM = A + B;
            Console.WriteLine("Sum  Of  {0}  And  {1}  Is  {2}", A, B, SUM);
            Console.Read();// To prevent console from vanishing
        }
    }
}
```

Relational Operator Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace RelationalOperations
{
    class Program
    {
        static void Main(string[] args)
        {
            int A, B;

            Console.WriteLine("Enter A number");
            A = int.Parse(Console.ReadLine());

            Console.WriteLine("Enter B number");
            B = int.Parse(Console.ReadLine());

            if(A>B)
                Console.WriteLine("The greatest is {0}", A);
            else
                Console.WriteLine("The greatest is {0}", B);
            Console.ReadKey();
        }
    }
}
```

Logical Operators Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LogicalOperations
{
    class Program
    {
        static void Main(string[] args)
        {
            int A, B,C;

            Console.WriteLine("Enter A number");
            A = int.Parse(Console.ReadLine());

            Console.WriteLine("Enter B number");
            B = int.Parse(Console.ReadLine());

            Console.WriteLine("Enter C number");
            C = int.Parse(Console.ReadLine());

            if(A > B && A > C)
                Console.WriteLine("The greatest is {0}", A);
            else if (B >A && B > C)
                Console.WriteLine("The greatest is {0}", B);
            else
```

```
                Console.WriteLine("The greatest is {0}", C);
            Console.ReadKey();
        }
    }
}
```

Loops Example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LoopsExample
{
   class Program
   {
      static void Main(string[] args)
      {
         int i = 0, n;
         Console.Write("Enter a number to generate series:");
         n = int.Parse(Console.ReadLine());
         while (i < n)
         {
            i++;
            Console.WriteLine(i);
         }
         Console.ReadKey();
      }
   }
}
```

Today's Topics:

- Different Types of Type Casting with example

Type Casting:

C#.NET supports 4 types of methods for explicit type casting.

1. C++ style of type casting

   Syntax:

   ```
   Datatype1 <variable>= value
   Datatype2 <variable>=(Datatype2)<variable>
   ```

   Ex:

   ```
   int i=10;
   byte b=(byte)i;
   ```

2. Converting
   - Working with convert class is called converting.
   - Converting can be used to convert from any data type into another data type.
   - Convert is a predefined static class that is a part of FCL (Framework Class Library).
   - Convert class contains a collection of methods.

Convert.ToByte(value)	To Byte Datatype
Convert.ToChar(value)	To CharDatatype
Convert.ToString(value)	To String Datatype

3. Parsing

   All predefined data types are predefined structure. A structure contains **parse()** method that can be used to parse the respective data type.

   Syntax:

   ```
   <datatype>.parse<value>
   ```

   Ex:

   ```
   int.parse(textbox1.text)
   ```

4. Boxing/unboxing (Which we discussed in deligate concept)

Example of various type casting:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ArithmeticOperations
{
    class Program
    {
        static void Main(string[] args)
        {
            //C++ type casting
            int i = 10;
            byte bt = (byte)i;
            Console.WriteLine("The value of i is:" + i);
            Console.WriteLine("The value of bt is:" + bt);


            //converting
            /*int x, y;
            string a, b;
            Console.WriteLine("Enter a value:");
            a = Console.ReadLine();
            Console.WriteLine("Enter b value:");
            b = Console.ReadLine();

            x = Convert.ToInt32(a);
            y = Convert.ToInt32(b);

            Console.WriteLine("The sum of {0} and {1} is {2}",x,y,(x+y));
            Console.ReadKey();
            */

            int x, y;
            string a, b;
            Console.WriteLine("Enter a value:");
            a = Console.ReadLine();
            Console.WriteLine("Enter b value:");
            b = Console.ReadLine();

            x = int.Parse(a);
            y = int.Parse(b);

            Console.WriteLine("The sum of {0} and {1} is {2}", x, y, (x + y));
            Console.ReadKey();


        }
    }
}
```

Console applications have the following drawbacks.

1. It is Character User Interface (CUI).
2. It is not user friendly.
3. We cannot develop complex applications using it.

These drawbacks can be overcome in the windows applications. The following are the advantages of the windows application.

1. Rich Graphical User Interface.
2. User friendly
3. Supports complex applications.

Windows application makes use of System.Window.Forms namespace