Sir Syed University of Engineering & Technology, Karachi

Programming In C#

Course Code : CS-127

Course Title : Object Oriented Programming

Semester : 2nd

Batch - 2020

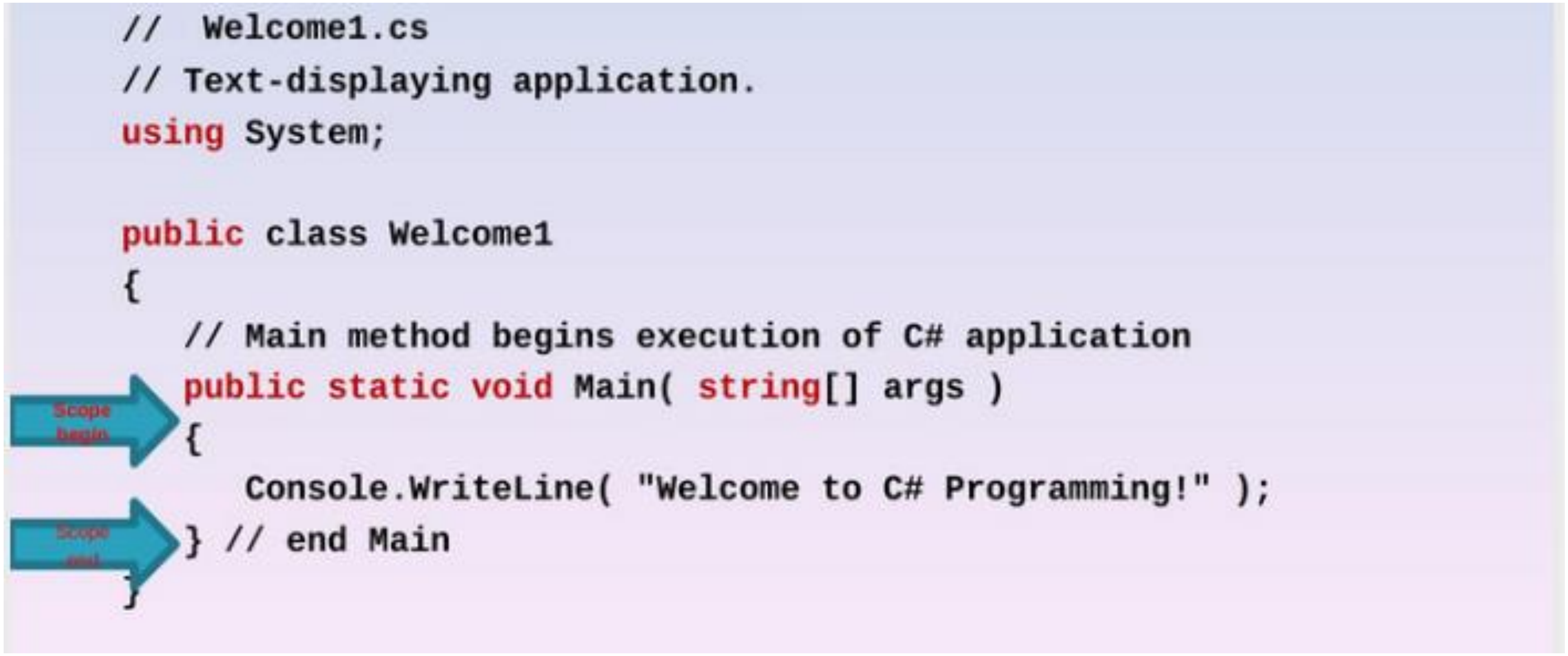
Department of Computer Science / Information Technology

What Is C#?

- C# is type-safe object-oriented language
- Enables developers to build a variety of secure and robust applications
- It was developed by Microsoft within the .NET Framework

C# Application

- A Simple C# Application: Displaying a Line of Text



```
//  Welcome1.cs
// Text-displaying application.
using System;

public class Welcome1
{
   // Main method begins execution of C# application
   public static void Main( string[] args )
   {
      Console.WriteLine( "Welcome to C# Programming!" );
   } // end Main
}
```

A Simple C# Application: Displaying a Line of Text

- Programmers insert comments to document applications.
- Comments improve code readability.
- The C# compiler ignores comments, so they do not cause the computer to perform any action when the application is run.

A Simple C# Application: Displaying a Line of Text

- A comment that begins with // is called a single-line comment, because it terminates at the end of the line on which it appears.

- A // comment also can begin in the middle of a line and continue until the end of that line.

- Delimited comments begin with the delimiter /* and end with the delimiter */. All text between the delimiters is ignored by the compiler.

A Simple C# Application: Displaying a Line of Text

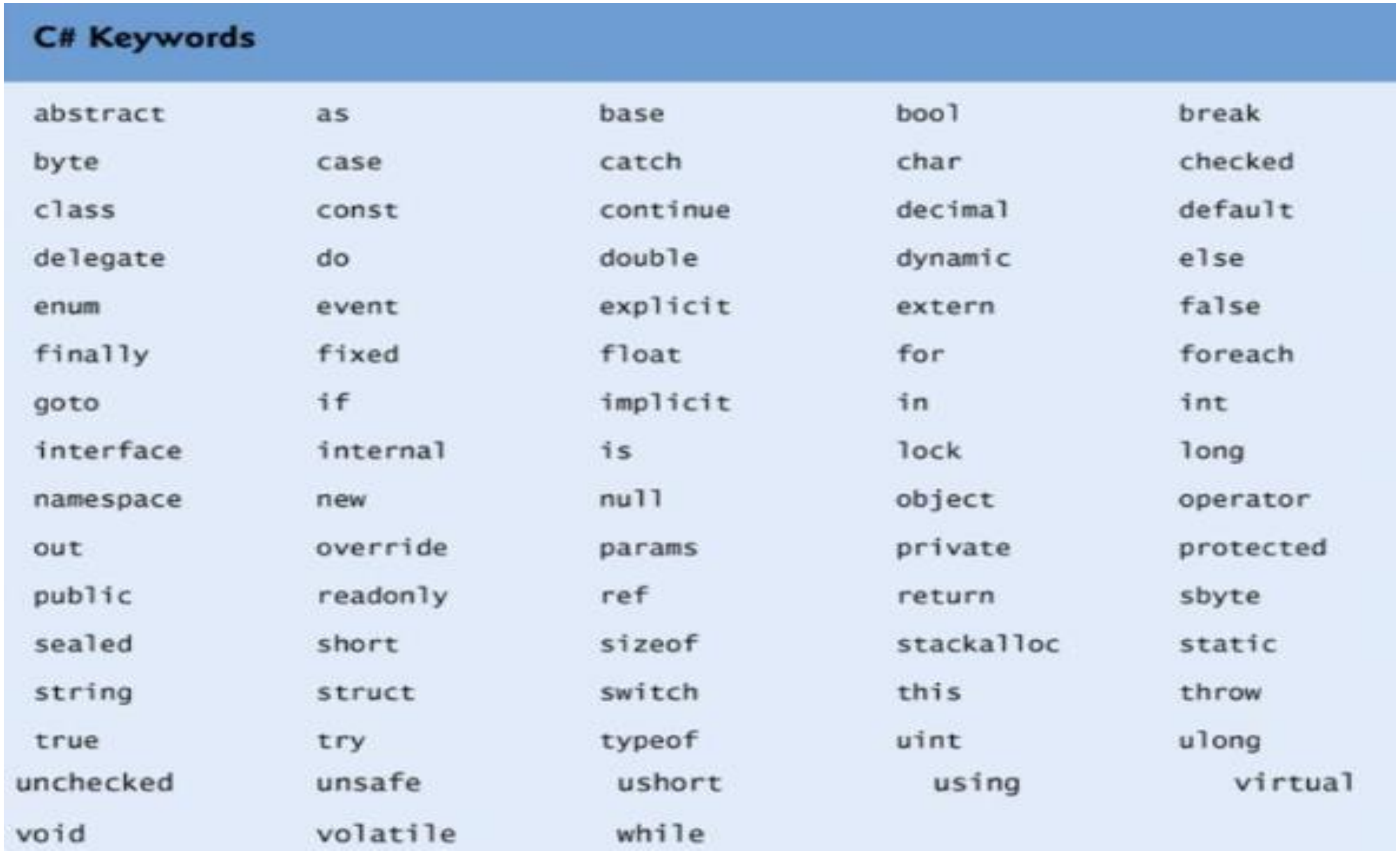
- Programmers use blank lines and space characters to make applications easier to read.
- Together, blank lines, space characters and tab characters are known as whitespace. Whitespace is ignored by the compiler.
- Keywords (sometimes called reserved words) are reserved for use by C# and are always spelled with all lowercase letters.

A Simple C# Application: Displaying a Line of Text

- Every application consists of at least one class declaration that is defined by the programmer. These are known as user-defined classes.

- The class keyword introduces a class declaration and is immediately followed by the class name.

The Complete List of C# Keywords



C# Keywords

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	dynamic	else
enum	event	explicit	extern	false
finally	fixed	float	for	foreach
goto	if	implicit	in	int
interface	internal	is	lock	long
namespace	new	null	object	operator
out	override	params	private	protected
public	readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc	static
string	struct	switch	this	throw
true	try	typeof	uint	ulong
unchecked	unsafe	ushort	using	virtual
void	volatile	while		

A Simple C# Application: Displaying a Line of Text

- C# is case sensitive — that is, uppercase and lowercase letters are distinct, so **a1** and **A1** are different (but both valid) identifiers.

- A left brace, {, begins the body of every class declaration. A corresponding right brace, }, must end each class declaration.

A Simple C# Application: Displaying a Line of Text

- Parentheses after an identifier indicate that it is an application building block called a method. Class declarations normally contain one or more methods.
- Method names usually follow the same casing capitalization conventions used for class names.
- For each application, one of the methods in a class must be called Main; otherwise, the application will not execute.

A Simple C# Application: Displaying a Line of Text

- Methods are able to perform tasks and return information when they complete their tasks. Keyword void indicates that this method will not return any information after it completes its task.

- The body of a method declaration begins with a left brace and ends with a corresponding right brace.

A Simple C# Application: Displaying a Line of Text

- Characters between double quotation marks are strings.
- Whitespace characters in strings are not ignored by the compiler.
- The **Console.WriteLine** method displays a line of text in the console window.
- The string in parentheses is the argument to the **Console.WriteLine** method.

A Simple C# Application: Displaying a Line of Text

- Method **Console.WriteLine** performs its task by displaying (also called outputting) its argument in the console window

- A method is typically composed of one or more statements that perform the method’s task.

- Most statements end with a semicolon.

Sir Syed University of Engineering & Technology, Karachi

Programming In C# Week -1 Session - 2

Course Code	: CS-127
Course Title	: Object Oriented Programming
Semester	: 2nd

Batch - 2020

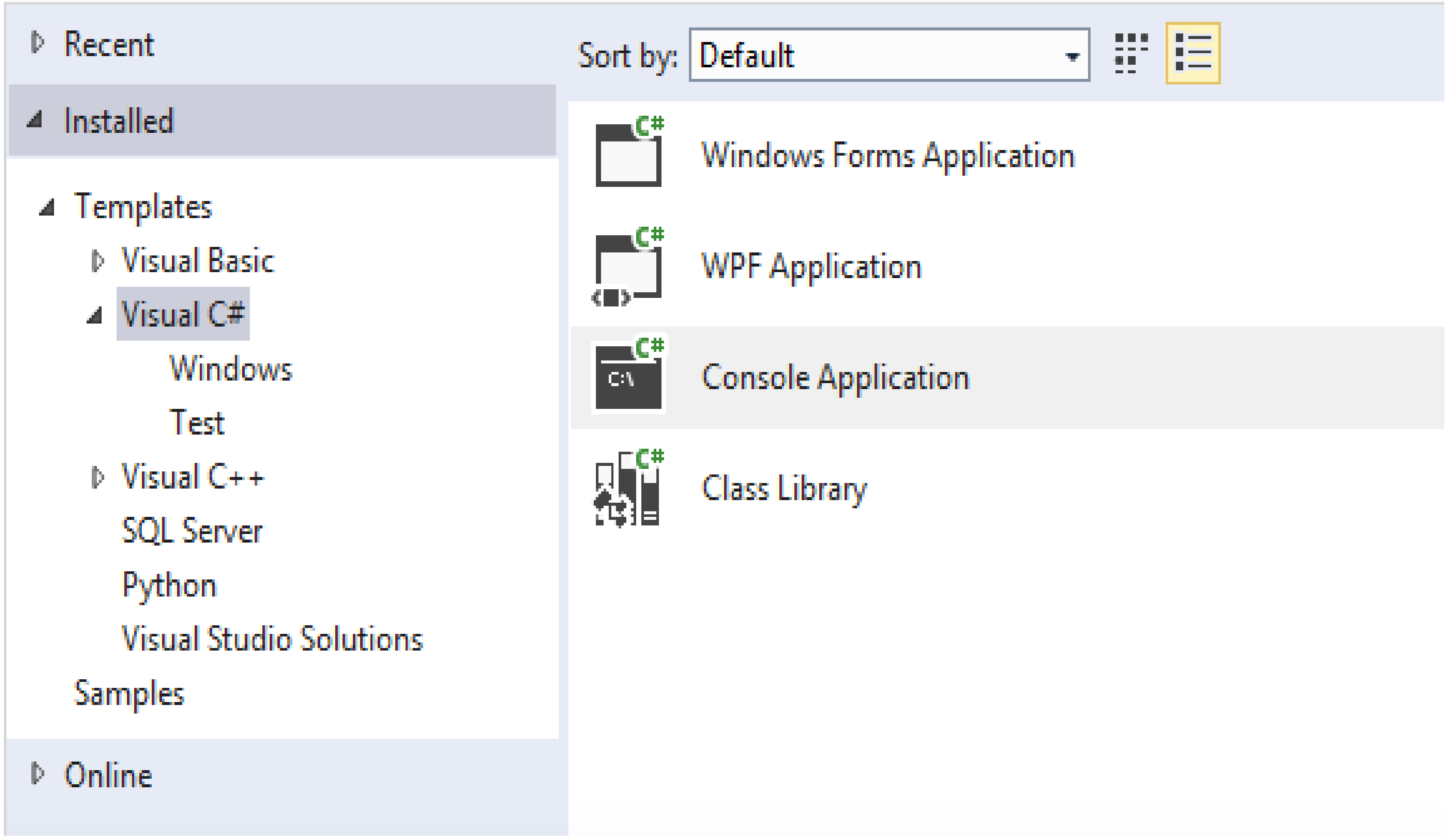
Department of Computer Science / Information Technology

Creating a Simple Application in C#

- To create your first application open Visual Studio and click ‘FILE’ menu, then ‘New Project’. A new window will pop-up. Please select ‘Visual C#’ template on the left panel and a list of C# project types will be presented

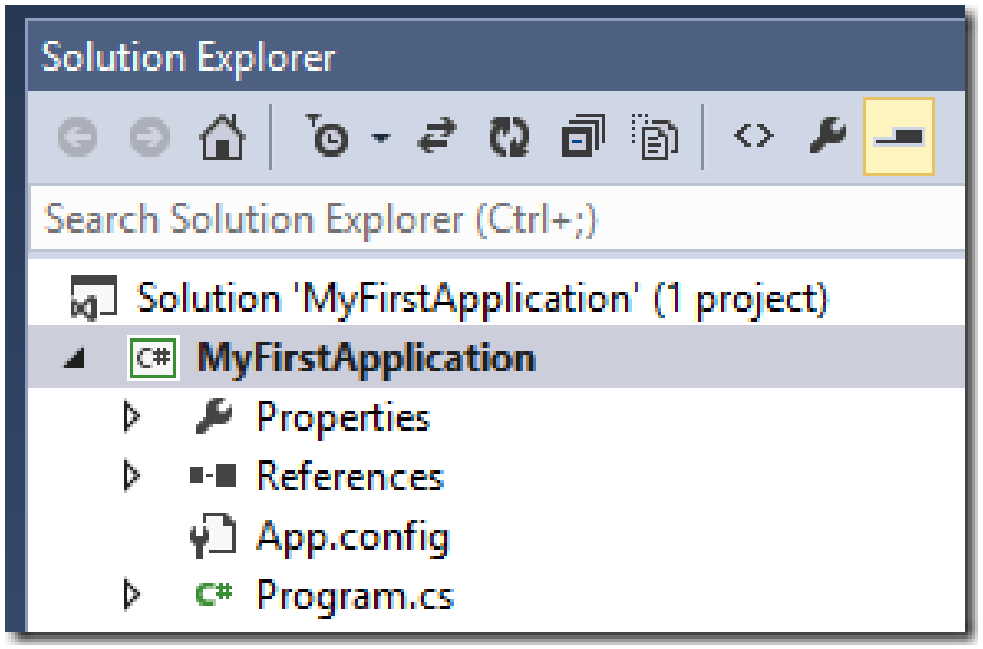
- Choose Console Application and put application name in ‘Name’ field (something like ‘MyFirstApplication’).

Creating a Simple Application in C#



Creating a Simple Application in C#

- After Visual Studio creates the project you will see Solution Explorer on the right side (by default).



Creating a Simple Application in C#

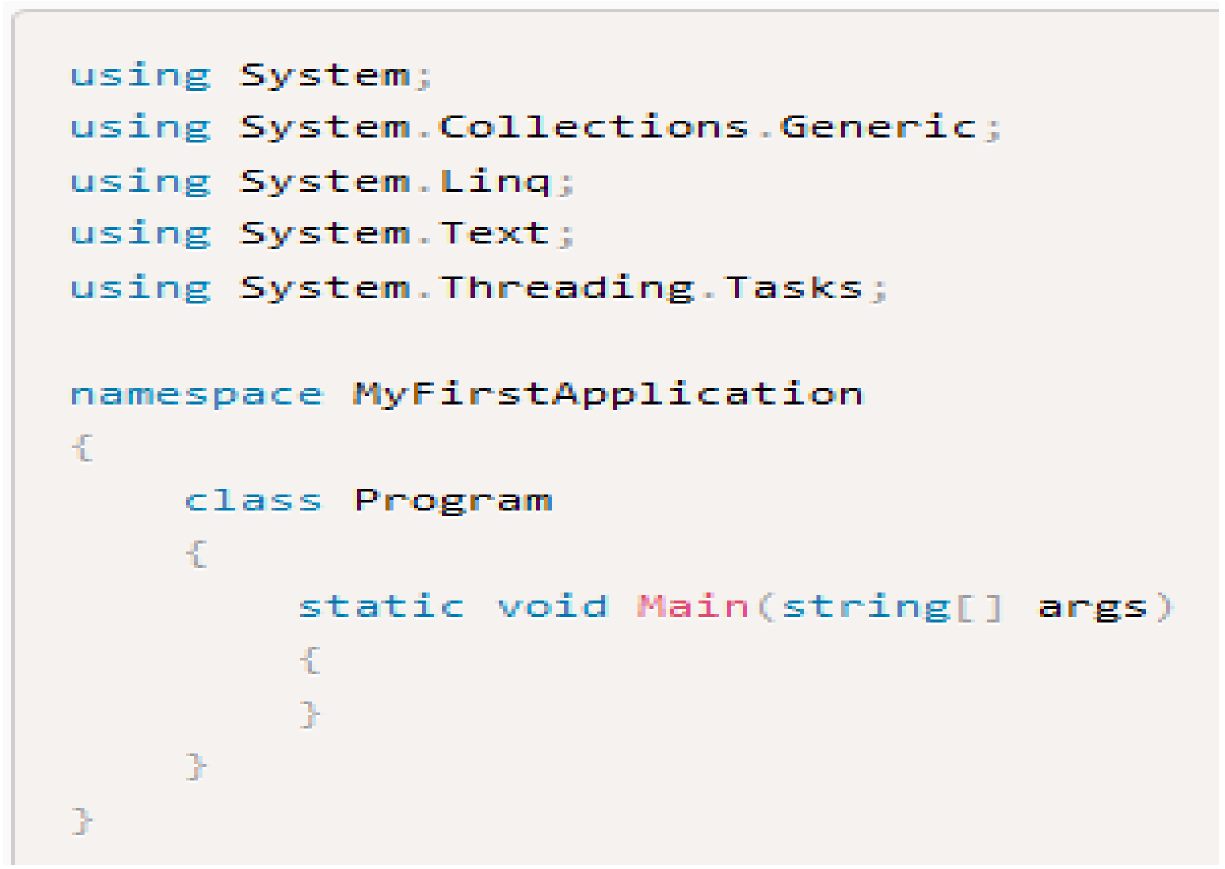
- When you created MyFirstApplication project, Visual Studio also created a solution.

- Solution is just a group of projects. Currently you will see just one project, but you can add more (e.g. by adding new class library). In the Solution Explorer you can see everything that belongs to your project.

Creating a Simple Application in C#

- **Properties** – If you expand it, you will see AssemblyInfo.cs file. It's a metadata file which contains information like: version, product name, company name, copyrights, etc.
- **References** – Contains a list of all libraries required by the project. By default there will be few basic libraries from .NET Framework
- **App.config** – A configuration file. .NET Framework have a nice way to use it.
- **Program.cs** – Finally, the place where you will find proper C# code.

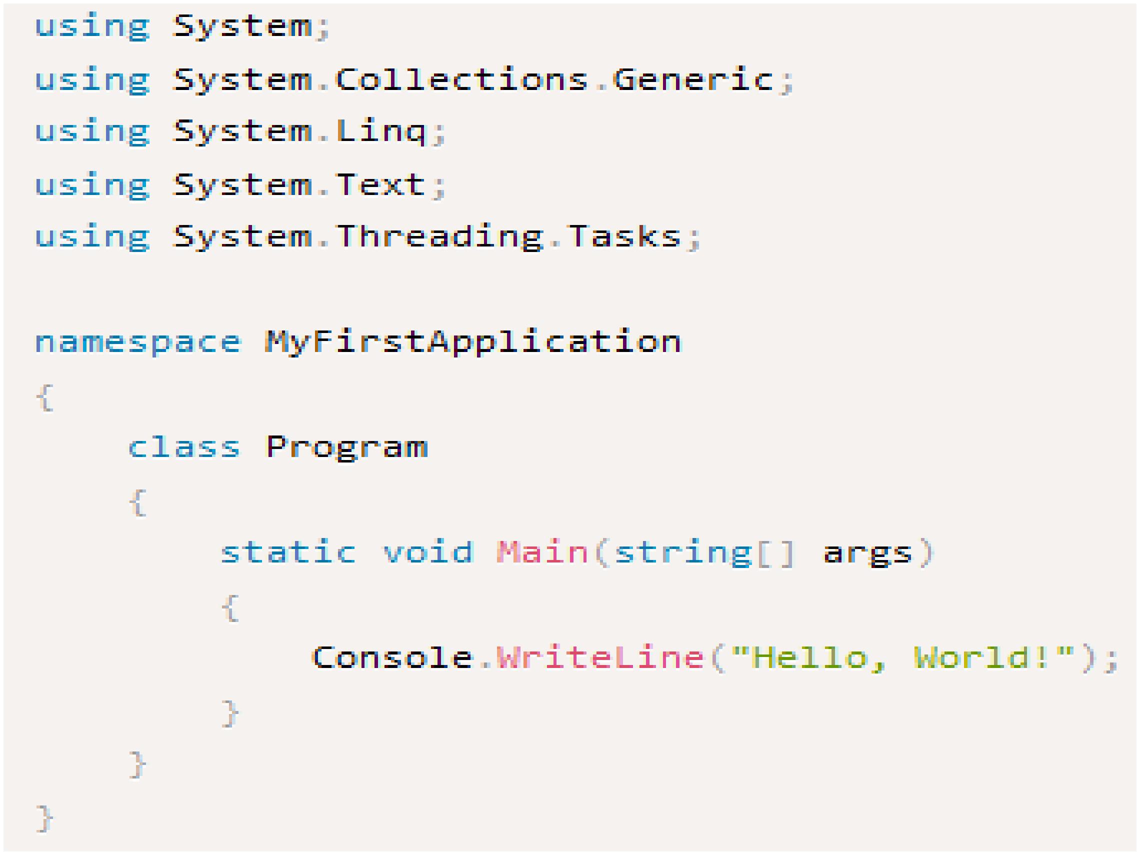
Let’s have a look at Program.cs file



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

namespace MyFirstApplication
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Let’s have a look at Program.cs file

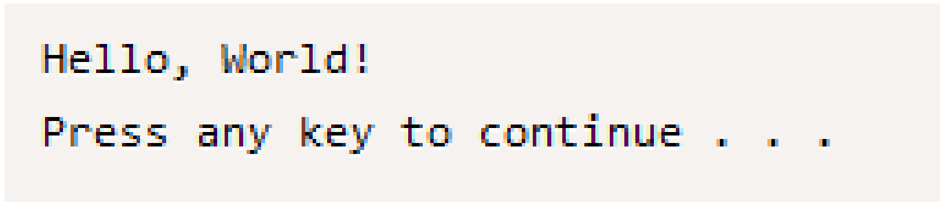
- To print a message insert following code:



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

namespace MyFirstApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, World!");
        }
    }
}
```

Let’s have a look at Program.cs file

- Now press Ctrl+F5 to run the application. You should see a console window with following output.

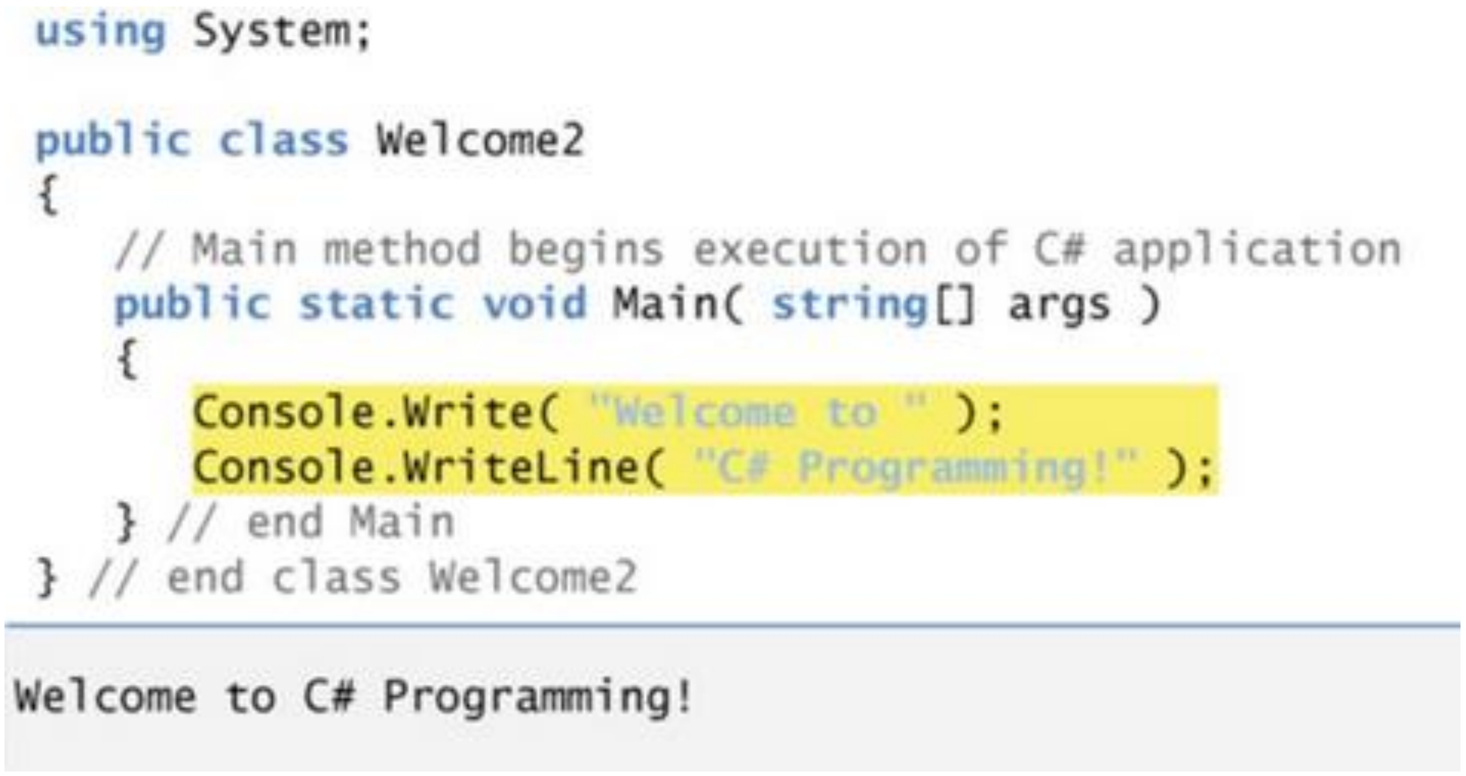


```
Hello, World!
Press any key to continue . . .
```

WriteLine / Write methods

- Class Welcome2, uses two statements to produce the same output as that shown in the previous example.
- Unlike WriteLine, Write method does not position the screen cursor at the beginning of the next line in the console window.

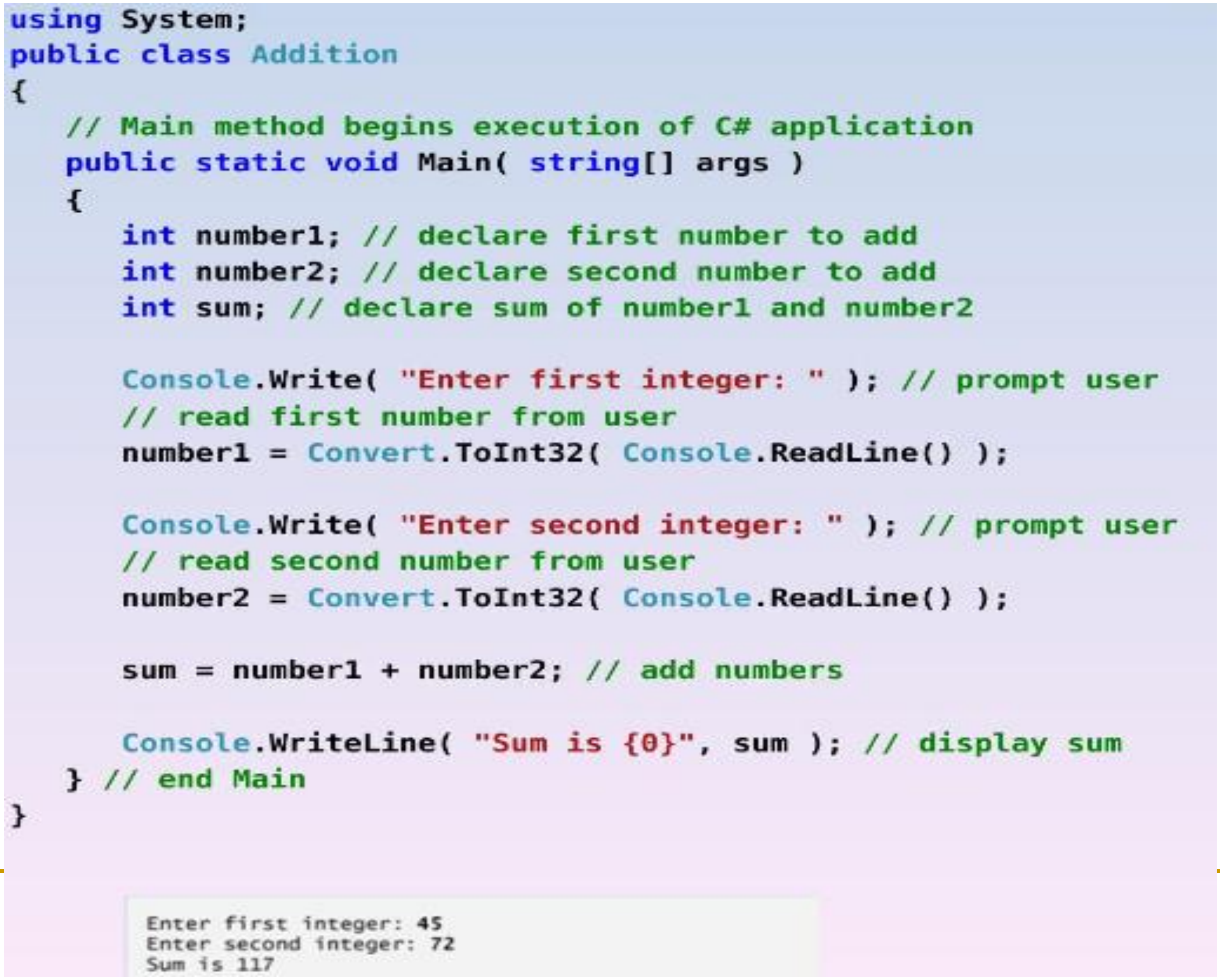
WriteLine / Write methods



```
using System;

public class Welcome2
{
   // Main method begins execution of C# application
   public static void Main( string[] args )
   {
      Console.Write( "Welcome to " );
      Console.WriteLine( "C# Programming!" );
   } // end Main
} // end class Welcome2
```

```
Welcome to C# Programming!
```

Another C# Application: Adding Integers



```
using System;
public class Addition
{
   // Main method begins execution of C# application
   public static void Main( string[] args )
   {
      int number1; // declare first number to add
      int number2; // declare second number to add
      int sum; // declare sum of number1 and number2

      Console.Write( "Enter first integer: " ); // prompt user
      // read first number from user
      number1 = Convert.ToInt32( Console.ReadLine() );

      Console.Write( "Enter second integer: " ); // prompt user
      // read second number from user
      number2 = Convert.ToInt32( Console.ReadLine() );

      sum = number1 + number2; // add numbers

      Console.WriteLine( "Sum is {0}", sum ); // display sum
   } // end Main
}
```

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Another C# Application: Adding Integers

- ReadLine method waits for the user to type a string of characters at the keyboard and press the Enter key.
- ReadLine returns the text the user entered.
- ToInt32 method converts this sequence of characters into data of type int.

Another C# Application: Adding Integers

- ToInt32 returns the int representation of the user’s input.
- A value can be stored in a variable using the assignment operator, =.
- Operator = is called a binary operator, because it works on two pieces of information
- An assignment statement assigns a value to a variable.

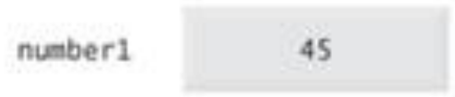
Another C# Application: Adding Integers

- An expression is any portion of a statement that has a value associated with it.
- – The value of the expression number1 + number2 is the sum of the numbers.
- – The value of the expression Console.ReadLine() is the string of characters typed by the user.
- Calculations can also be performed inside output statements.

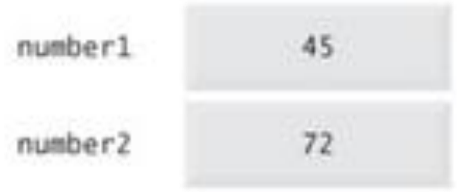
Memory Concepts

- Variable names actually correspond to locations in the computer’s memory. Every variable has a name, a type, a size and a value.
- the computer has placed the value 45 in the memory location corresponding to number1

Memory Concepts

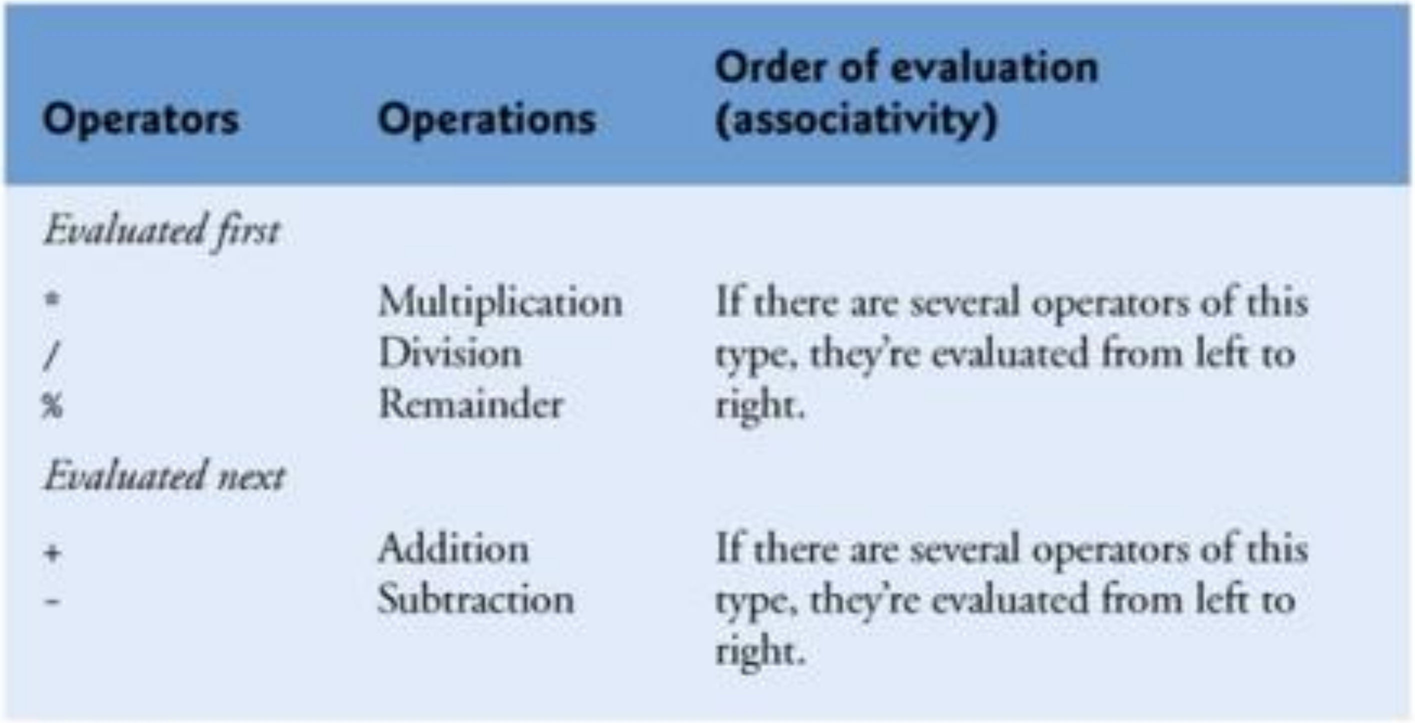


Memory location showing the name and value of variable `number1`.



Memory locations after storing values for `number1` and `number2`.

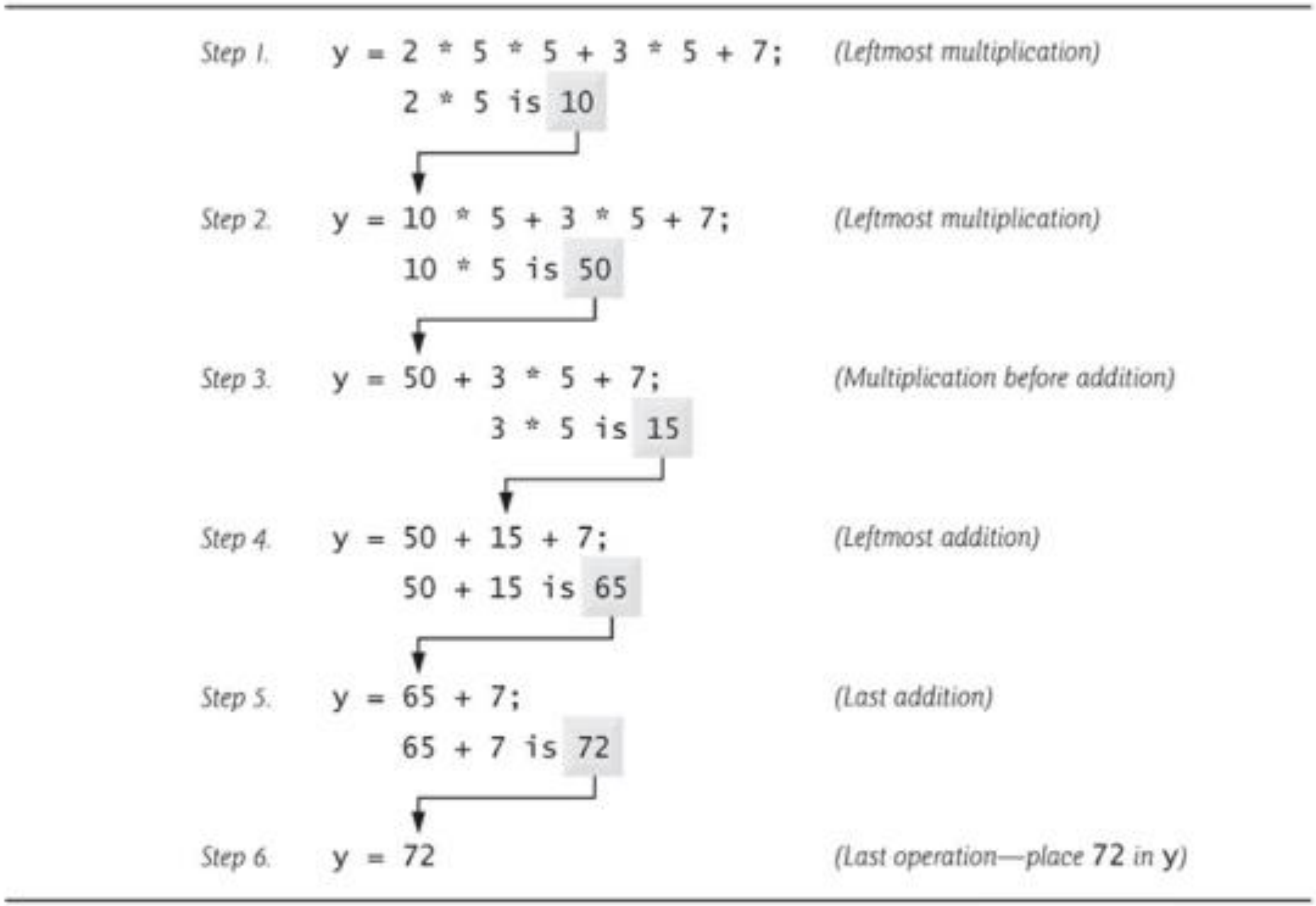
Arithmetic



Operators	Operations	Order of evaluation (associativity)
Evaluated first		
*	Multiplication	If there are several operators of this type, they're evaluated from left to right.
/	Division	
%	Remainder	
Evaluated next		
+	Addition	If there are several operators of this type, they're evaluated from left to right.
-	Subtraction	

Precedence of arithmetic operators.

Arithmetic

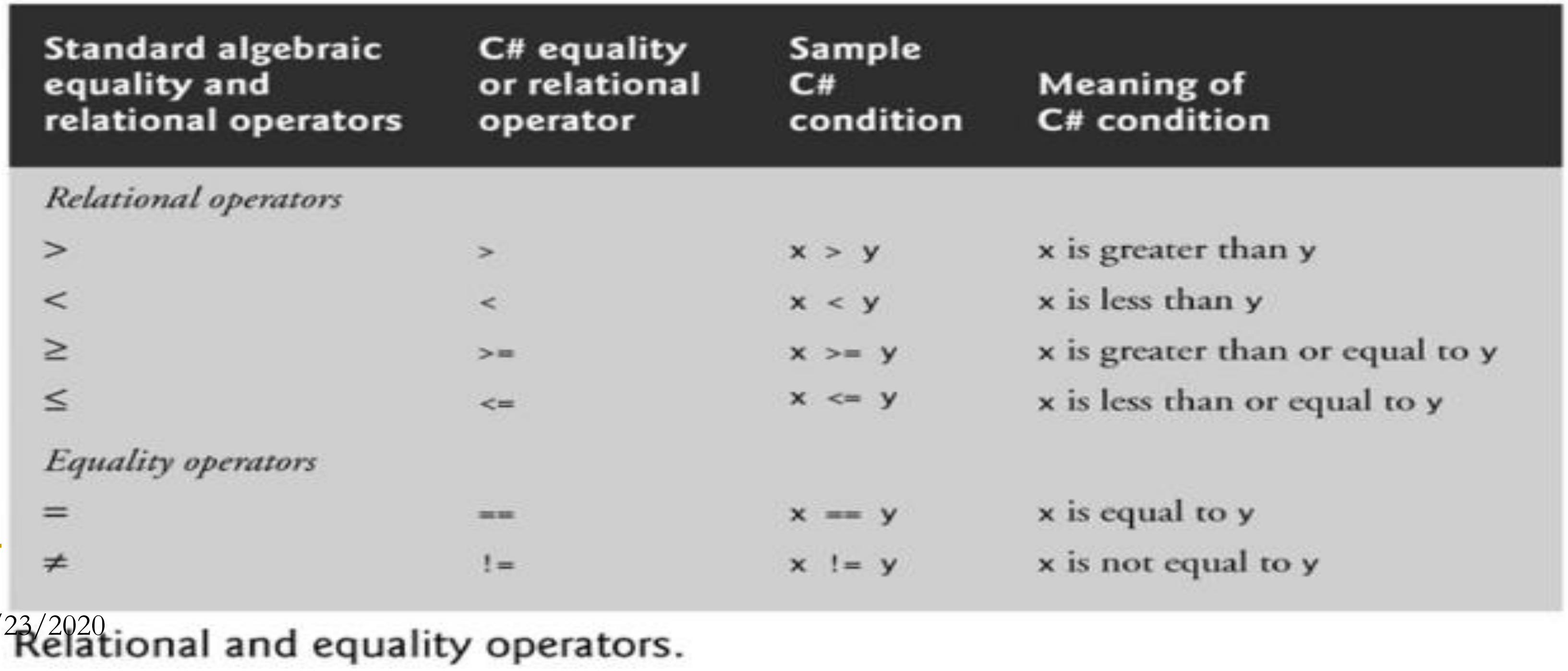


```
Step 1.   y = 2 * 5 * 5 + 3 * 5 + 7;      (Leftmost multiplication)
              2 * 5 is 10

Step 2.   y = 10 * 5 + 3 * 5 + 7;         (Leftmost multiplication)
              10 * 5 is 50

Step 3.   y = 50 + 3 * 5 + 7;             (Multiplication before addition)
                   3 * 5 is 15

Step 4.   y = 50 + 15 + 7;                (Leftmost addition)
              50 + 15 is 65

Step 5.   y = 65 + 7;                     (Last addition)
              65 + 7 is 72

Step 6.   y = 72                          (Last operation—place 72 in y)
```

Order in which a second-degree polynomial is evaluated.

Decision Making: Equality and Relational Operators

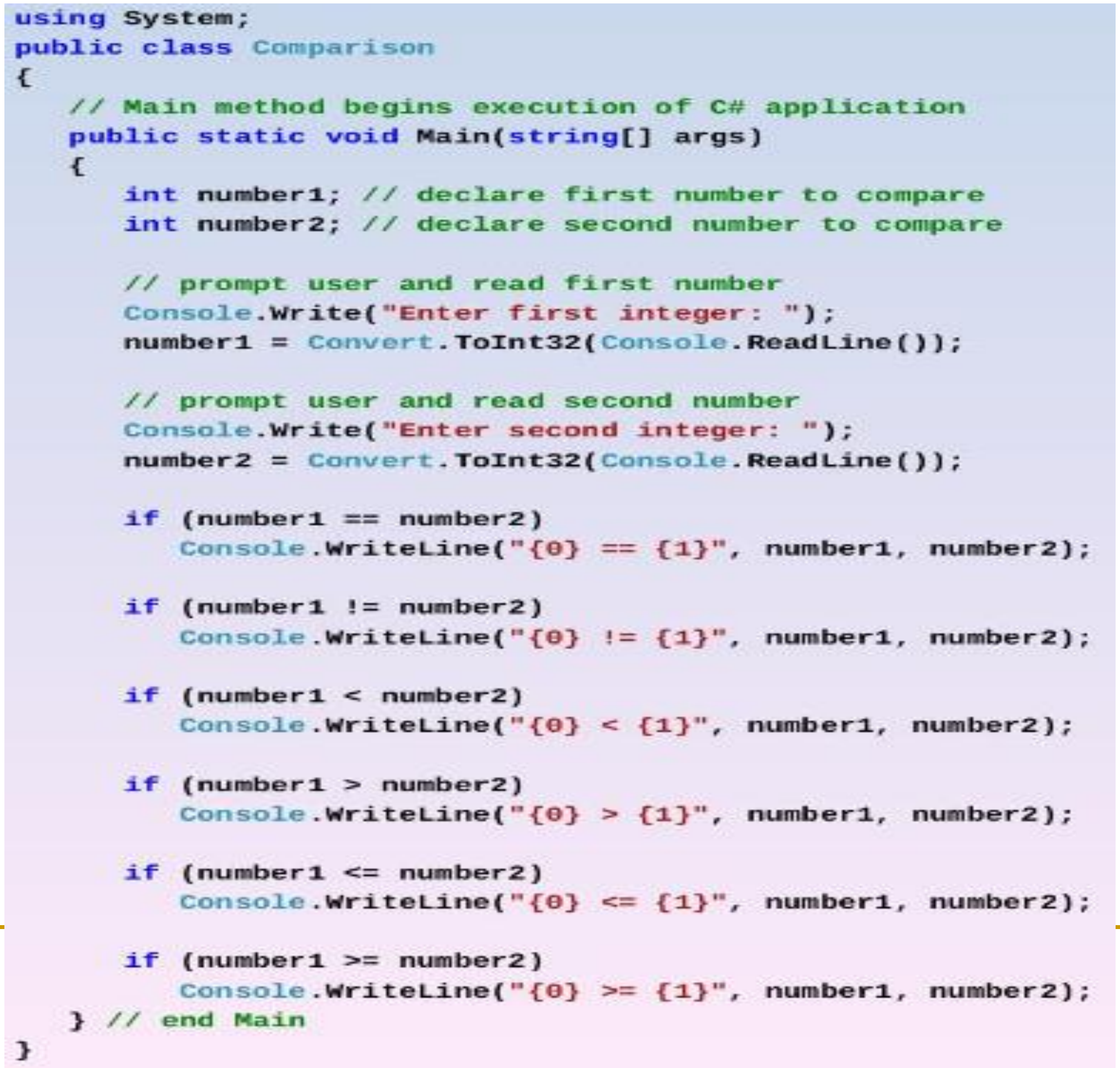
- A condition is an expression that can be either true or false. Conditions can be formed using the equality operators (== and ! =) and relational operators (>, <, >= and <=)



Standard algebraic equality and relational operators	C# equality or relational operator	Sample C# condition	Meaning of C# condition
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y

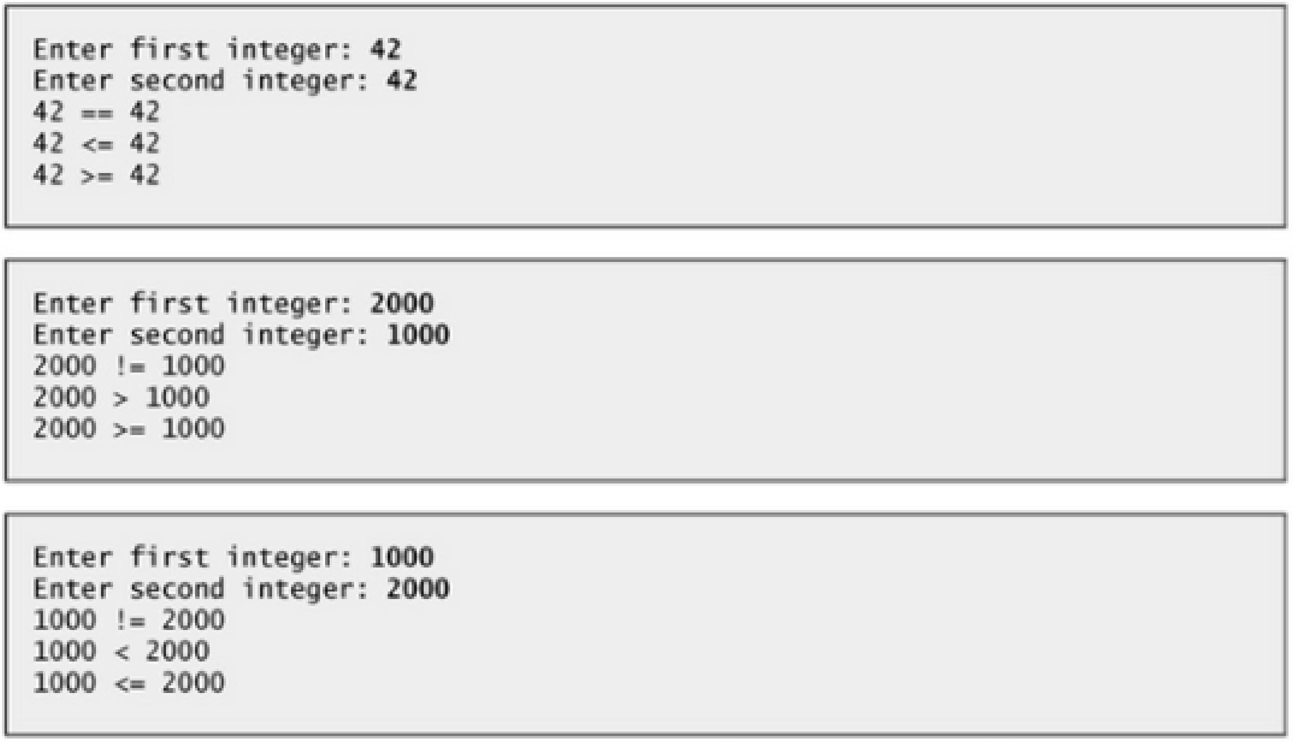
Relational and equality operators.

Decision Making: Equality and Relational Operators



```
using System;
public class Comparison
{
    // Main method begins execution of C# application
    public static void Main(string[] args)
    {
        int number1; // declare first number to compare
        int number2; // declare second number to compare

        // prompt user and read first number
        Console.Write("Enter first integer: ");
        number1 = Convert.ToInt32(Console.ReadLine());

        // prompt user and read second number
        Console.Write("Enter second integer: ");
        number2 = Convert.ToInt32(Console.ReadLine());

        if (number1 == number2)
            Console.WriteLine("{0} == {1}", number1, number2);

        if (number1 != number2)
            Console.WriteLine("{0} != {1}", number1, number2);

        if (number1 < number2)
            Console.WriteLine("{0} < {1}", number1, number2);

        if (number1 > number2)
            Console.WriteLine("{0} > {1}", number1, number2);

        if (number1 <= number2)
            Console.WriteLine("{0} <= {1}", number1, number2);

        if (number1 >= number2)
            Console.WriteLine("{0} >= {1}", number1, number2);
    } // end Main
}
```

Decision Making: Equality and Relational Operators



```
Enter first integer: 42
Enter second integer: 42
42 == 42
42 <= 42
42 >= 42
```

```
Enter first integer: 2000
Enter second integer: 1000
2000 != 1000
2000 > 1000
2000 >= 1000
```

```
Enter first integer: 1000
Enter second integer: 2000
1000 != 2000
1000 < 2000
1000 <= 2000
```

Comparing integers using `if` statements, equality operators and relational operators. (Part 3 of 3.)

Thank you