CHAPTER 2

C# Basics using Console Application

Microsoft has introduced **Visual Studio.NET**, which is a tool (also called Integrated Development Environment) for developing .NET applications by using programming languages such as **VB, C#, C++ and J#.** etc.

C# (C Sharp)

- C# is a **object-oriented programming** language developed by Microsoft.
- Runs on the **.NET Framework.**
- Designed for **Common Language Infrastructure (CLI)**.

C# Programming Features

- Simple
- Modern programming language
- Object oriented
- Type safe
- Interoperability
- Scalable and Updateable
- Structured programming language
- Rich Library
- Fast speed

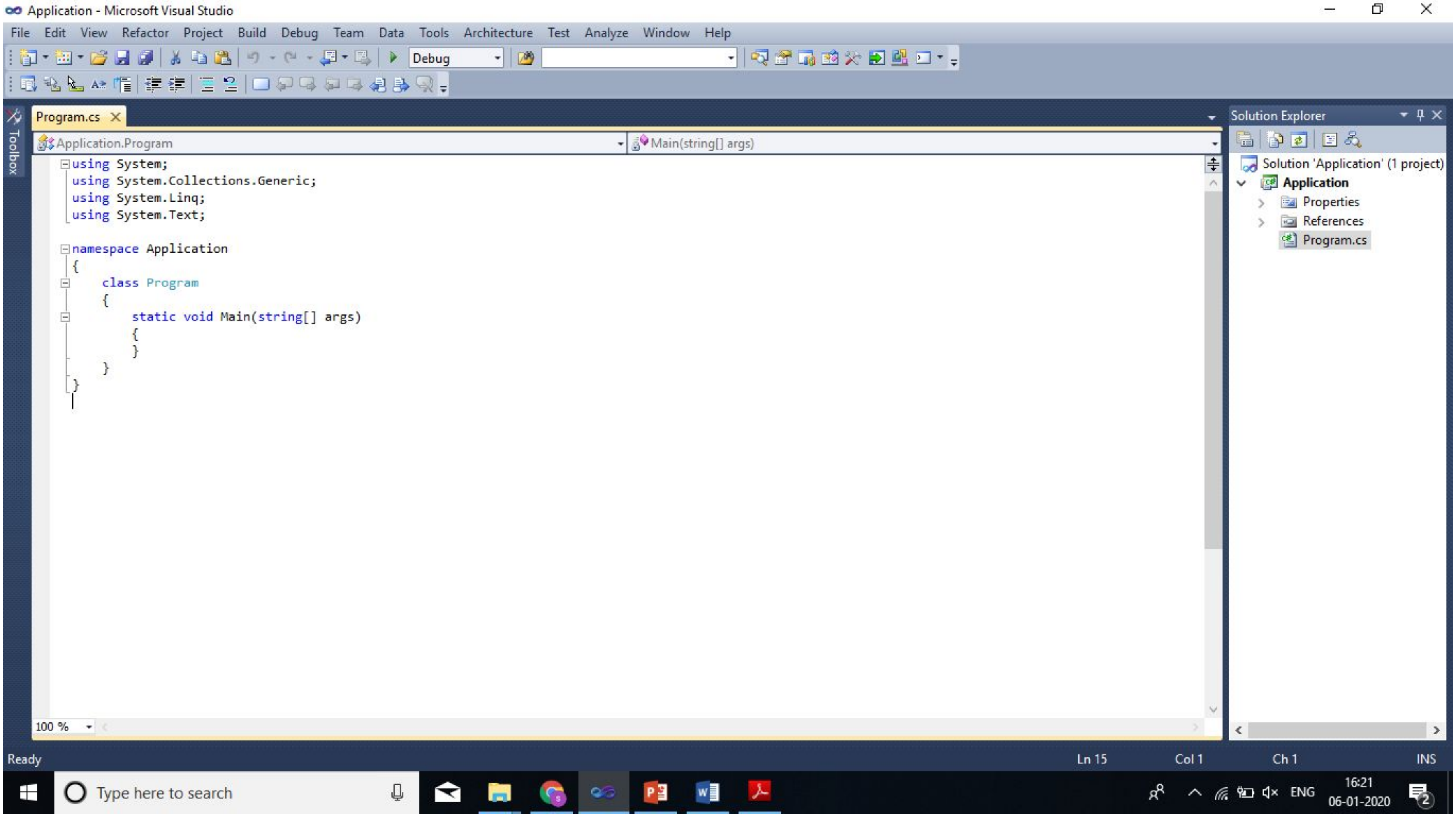
Why C# widely used as professional language?

- It is a modern, general-purpose programming language.
- It is object oriented.
- It is easy to learn.
- It is a structured language.
- It produces efficient programs.
- It can be compiled on a variety of computer platforms.
- It is a part of .Net Framework.

Program Structure

- Namespace declaration
- A class
- Class methods
- Class attributes
- The Main method
- Statements and Expressions
- Comments

Example

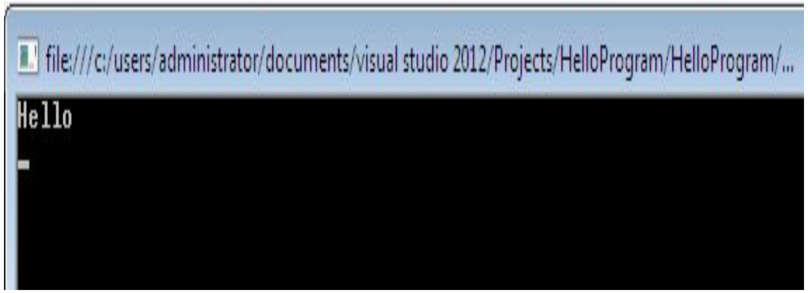


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

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

Sample Program-1

- **HelloProgram.cs**

```
using System;
namespace HelloProgram
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello");
            Console.ReadKey();
        }
    }
}
```

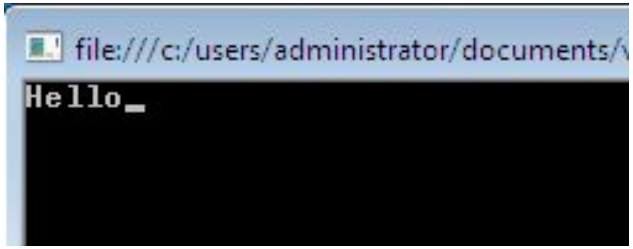


file:///c:/users/administrator/documents/visual studio 2012/Projects/HelloProgram/HelloProgram/...

```
Hello
_
```

Write() and WriteLine()

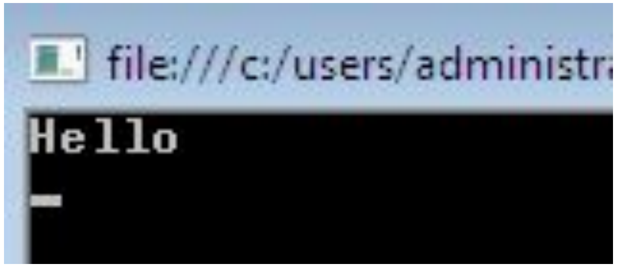
```
using System;
namespace HelloProgram
{
    //This is Hello Program
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Hello");
            Console.ReadKey();
        }
    }
}
```



file:///c:/users/administrator/documents/\

```
Hello_
```

Console.WriteLine("Hello");



file:///c:/users/administra

```
Hello
_
```

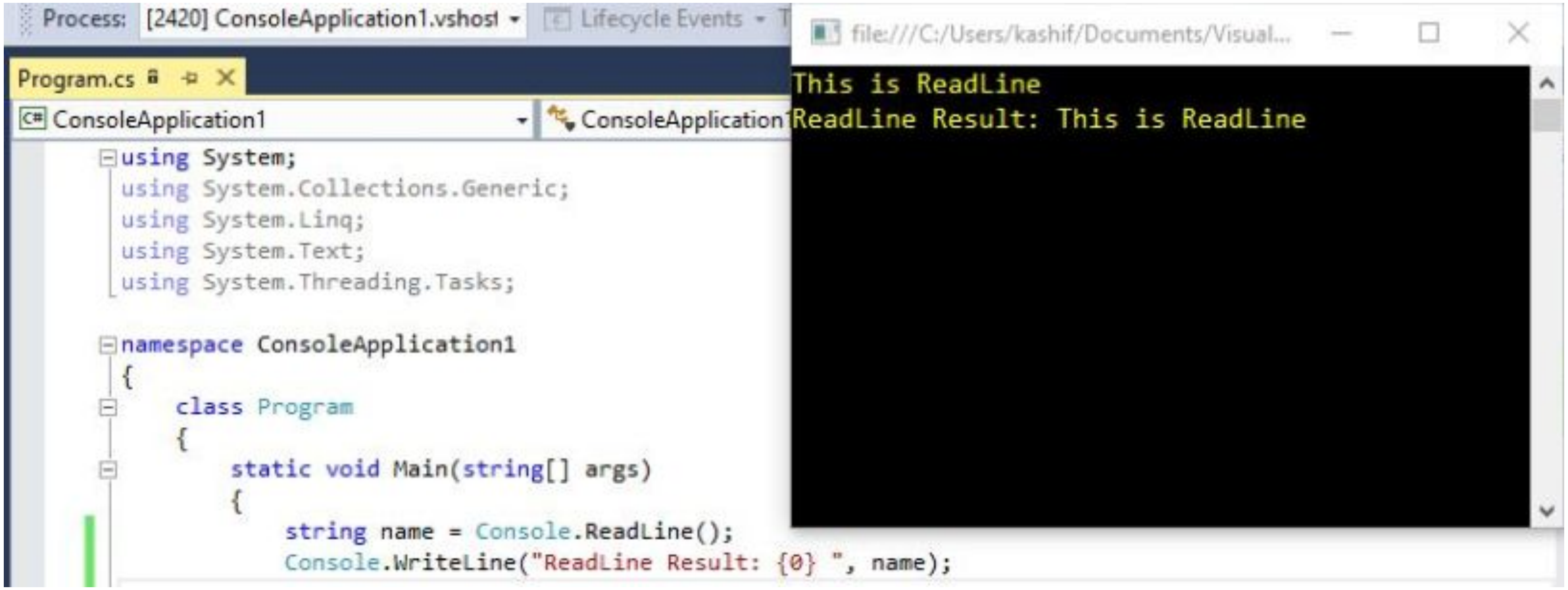
Function used to take User input

- ReadLine()
- Read()
- Readkey()

ReadLine()

ReadLine(): read all the characters from user input and return string.

Note: Data type should be STRING.



Process: [2420] ConsoleApplication1.vshost
Lifecycle Events
Program.cs
ConsoleApplication1
ConsoleApplication
file:///C:/Users/kashif/Documents/Visual...

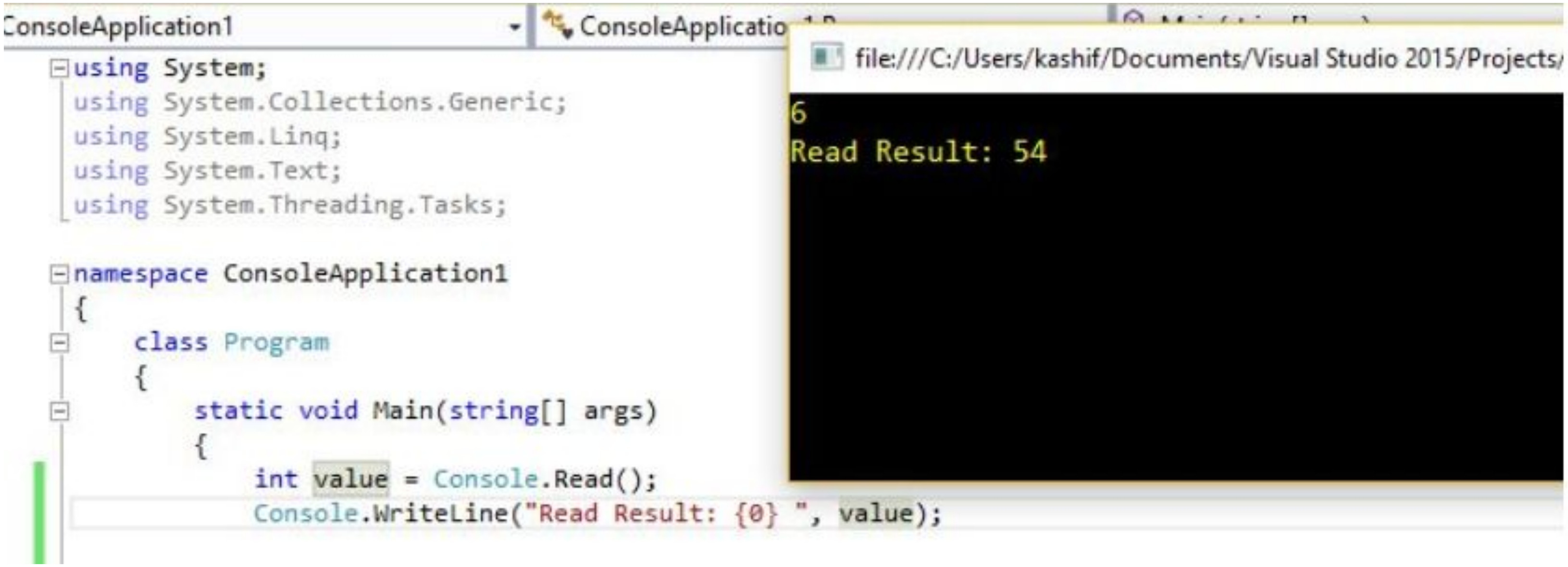
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            string name = Console.ReadLine();
            Console.WriteLine("ReadLine Result: {0} ", name);
```

```
This is ReadLine
ReadLine Result: This is ReadLine
```

Read()

Read(): **accept single character from user input and return its ASCII Code.**

Note: Return Data type must be integer.



ConsoleApplication1
ConsoleApplicatio
file:///C:/Users/kashif/Documents/Visual Studio 2015/Projects/

```
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)
        {
            int value = Console.Read();
            Console.WriteLine("Read Result: {0} ", value);
```

```
6
Read Result: 54
```

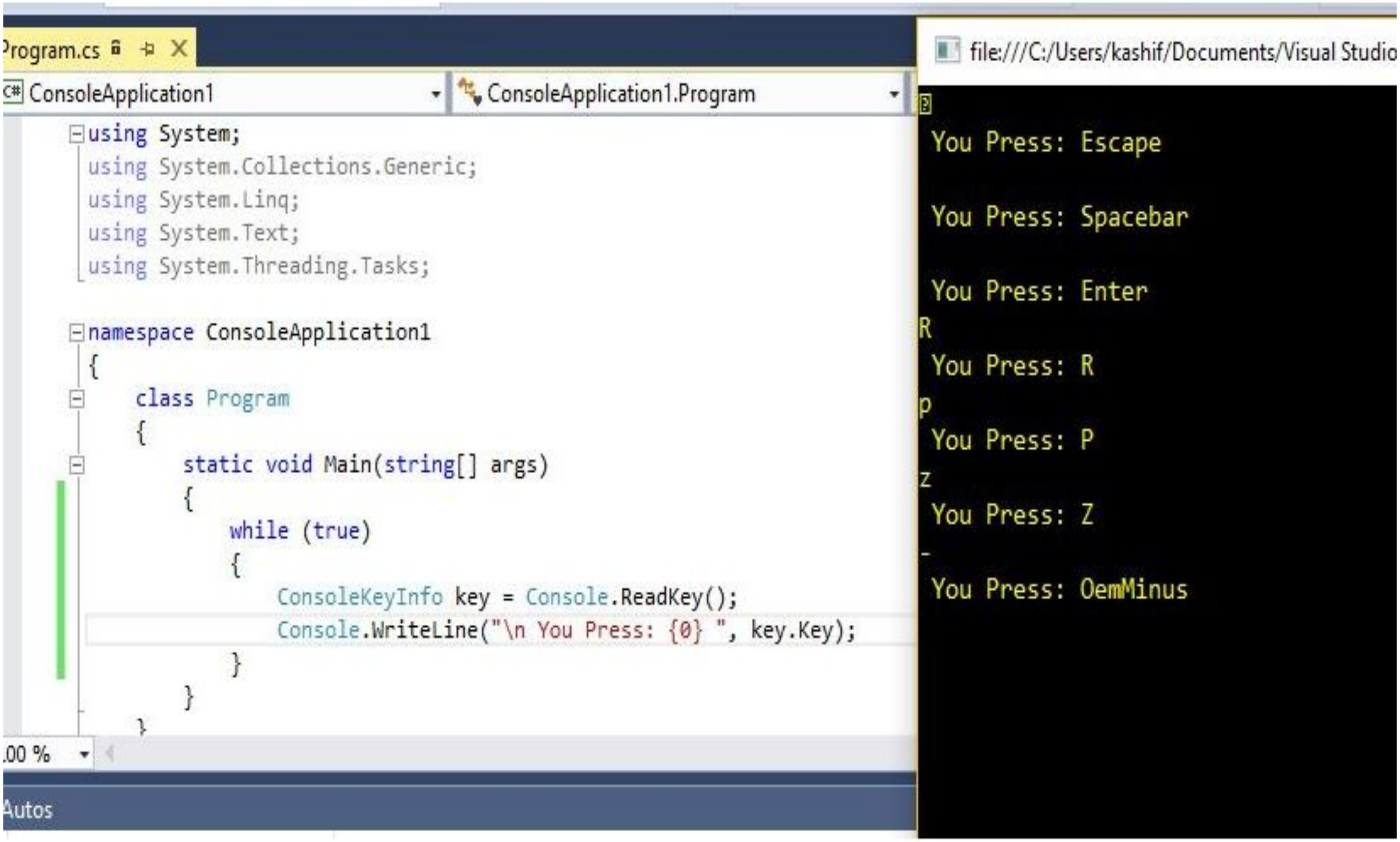
ReadKey()

- It **obtains character or function key pressed** by the user. In simple words, it read that which key is **pressed by user and return its name**.
- **Note: Its return type is ConsoleKeyInfo.**

Declaration:

```
public static ConsoleKeyInfo ReadKey ();
```

ReadKey()



Program.cs
ConsoleApplication1
ConsoleApplication1.Program
00 %
Autos
file:///C:/Users/kashif/Documents/Visual Studio

```
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)
        {
            while (true)
            {
                ConsoleKeyInfo key = Console.ReadKey();
                Console.WriteLine("\n You Press: {0} ", key.Key);
            }
        }
    }
```

```

 You Press: Escape

 You Press: Spacebar

 You Press: Enter
R
 You Press: R
p
 You Press: P
z
 You Press: Z
-
 You Press: OemMinus
```

Variable and constants

- int a;   //declaration of vaiable.
- const float pi=3.14 //Pi is float and constant.

Data Types

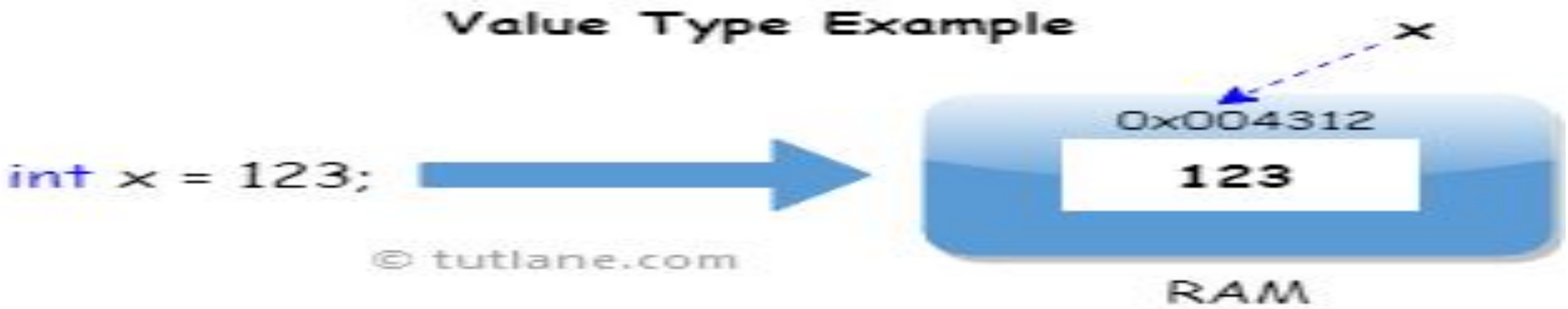
- Three Data Types

1. Value type
2. Reference Type
3. Pointer Type

Data Type (Cont..)

1. **Value type**

- Holds a data value within its **own memory space**
- Value Types will use **Stack memory** to store the variables values.
- It is derived from **System.ValueType.**
- For example, if we define and assign a value to the variable like int x = 123; then the system will use the same memory space of variable ‘x’ to store the value ‘123’.



Value type

bool	Boolean value
byte	8-bit unsigned int
char	16 bit char
decimal	128 bit precise decimal value
double	64 bit double precision floating number
float	32 bit single precision floating
int	32 bit signed integer type
long	64 bit signed integer type
sbyte	8 bit signed int type
short	16 bit signed int type
uint	32 bit unsigned int type
ulong	64 bit unsigned int type
ushort	16 bit unsigned int type

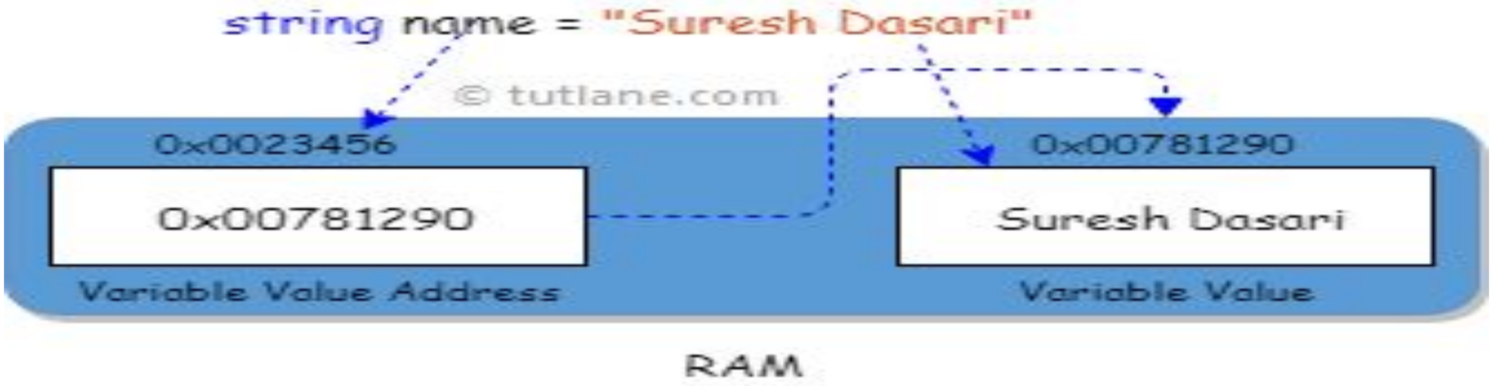
2. Reference type

- It does not contain actual **data stored in variable but it stores the address where the value is being stored**.
- Reference type contains a pointer to another memory location that holds the data.
- If the data in the **memory location is changed by one of the variables,** the other variable automatically reflects this change in value.
- Three types:
  - String type
  - Object type
  - Dynamic type

2. Reference type
1.String type

- Strings are immutable(unchangeable).
- It is derived from object type
- Example:

  string name = "Suresh Dasari";



1.String type

- string columns = "Column 1\tColumn 2\tColumn 3";

  //Output: Column 1 Column 2 Column 3

- string rows = "Row 1\r\nRow 2\r\nRow 3";

```
/* Output: Row 1
           Row 2
                 Row 3 */
```

- string filePath = @"C:\Users\scoleridge\Documents\";

  //Output: C:\Users\scoleridge\Documents\

2. Reference type

2.Object type

- It is base class for all data types.

--It can assigned values of any other data types.

- In .Net Framework all the Object of either **Reference Type or Value Type** comes from Object class.
- Because of this, **every method defined in the Object class is available in all objects in the system.**
- **Type checking for these type of variable takes place at compile time.**

Object type:- Boxing and unboxing

- Process of converting **value type into object type** called **boxing**.
- Ex:

```
int i=100;  Object o=i;
```

- Process of converting **object type into value** type called **unboxing.**
- Ex:

```
Object o=100;  i=(int)o;
```

example

```
static void Main()
      {
          object a = 4.5;
         object b = 1;
         object c = 'A';
         object d = “Hello";
         int i = 100;
         object o = i;  //value to object type
         object p = 200;
         int j = (int)p; //object type to value type
         Console.WriteLine(a);
         Console.WriteLine(b);
         Console.WriteLine(c);
         Console.WriteLine(d);
         Console.WriteLine(o);
         Console.WriteLine(j);
      }
```

<u>Output</u>

```
4.5
1
A
Hello
100
200
```

2. Reference type

3.Dynamic Type

- it store **any type of value.**
- **Type checking for these type of variable takes place** at **run time**.
- A dynamic type **changes its type at runtime** based on the value.
- Example:

```
dynamic d=20;
```

Dynamic type

```
static void Main()
        {
            dynamic x = 1;
            dynamic q = 1.5;
            dynamic r = 'a' ;
            dynamic s = “Hello";
            Console.WriteLine(x);
            Console.WriteLine(q);
            Console.WriteLine(r);
            Console.WriteLine(s);
        }
```

Output

```
1
1.5
a
Hello
```

3. Pointer type

- The Pointer Data Types will contain a **memory address** of the variable value.
- ***ampersand (&)*:** It is Known as **<u>Address Operator</u>**. It is used to determine the **<u>address of a variable</u>**.
- ***asterisk (*)*:** It is known as **<u>Indirection Operator</u>**. It is used to access the **<u>value of an address.</u>**
- Example:

```
int n = 10; // declare variable
int *p = &n; // address of n is assigned to P.

Console.WriteLine((int)p);   //display memory address
Console.WriteLine(*p);   // displays the value at memory address
```

Type Casting

- Two types of type casting:
  1. Implicit type casting
  2. Explicit type casting

Implicit type casting: The values of certain data **types are automatically converted to the different data types in C#.** This is called implicit conversion.It is done by **Compiler.**

Example:

```
int i = 345;
float f = i;  //i is converted from int to float.
Console.WriteLine(f);
```

Explicit type casting(Done by programmer)

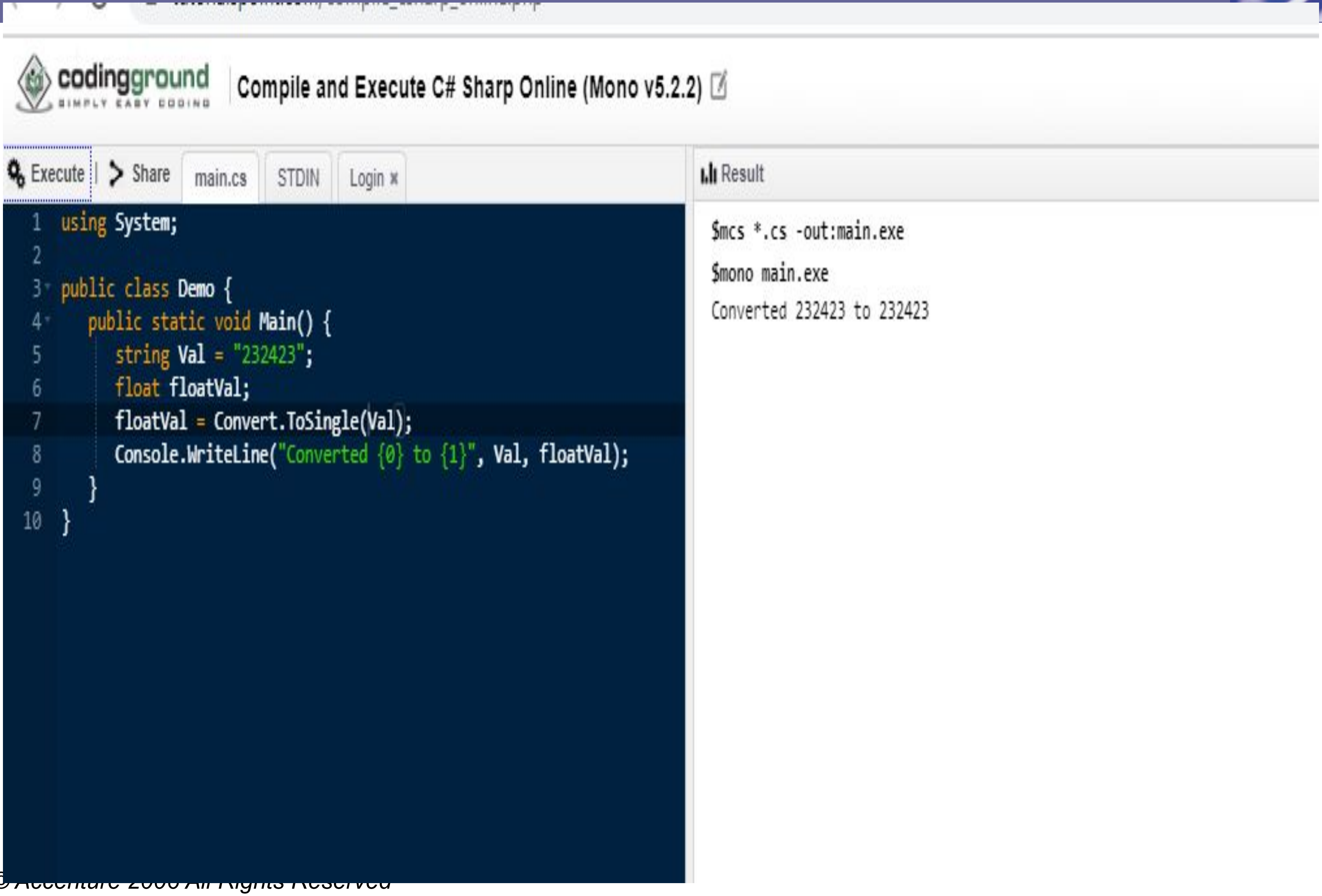
```
class ExplicitConversion {
    static void Main(string[] args) {
        double d = 5673.74;
        int i;
        // cast double to int.
        i = (int)d;
        Console.WriteLine(i);
        Console.ReadKey(); } }
```

//Output: 5673

C# type conversion

- **ToBoolean-**Converts a type to a Boolean value, where possible.
- **ToByte-**Converts a type to a byte.
- **ToChar-**Converts a type to a single Unicode character, where possible.
- **ToDateTime-**Converts a type (integer or string type) to date-time structures.
- **ToDecimal-**Converts a **floating point or integer type to a decimal type**.
- **ToDouble-**Converts a type to a double type.
- **ToInt16-**Converts a type to a **16-bit Signed integer**.
- **ToInt32-**Converts a type to a **32-bit Signed integer**.
- **ToInt64-**Converts a type to a **64-bit Signed integer**.
- **ToSbyte**-Converts a **String** representation of a number in a specified **base.(2,8,10,16)**
- **ToSingle-**Convert a specified value to a **floating-point** number.
- **ToString-**Converts a type to a string.
- **ToUInt16-** Convert a specified value to a **16-bit unsigned integer**
- **ToUInt32-**Convert a specified value to a **32-bit unsigned integer.**
- **ToUInt64-** Convert a specified value to a **64-bit unsigned integer.**

ToSingle()



```
using System;

public class Demo {
   public static void Main() {
      string Val = "232423";
      float floatVal;
      floatVal = Convert.ToSingle(Val);
      Console.WriteLine("Converted {0} to {1}", Val, floatVal);
   }
}
```

```
$mcs *.cs -out:main.exe
$mono main.exe
Converted 232423 to 232423
```

Nullable type

- A **value type** **cannot be assigned a null value.** For example, ***int i = null*** **will give you a compile time error**.
- Nullable types that allow you to assign null to value type variables.
- Syntax: Nullable<t>
- Example: Nullable<int> i = null;

- A nullable type can represent the correct range of values for its underlying value type, **plus an additional *null* value.**

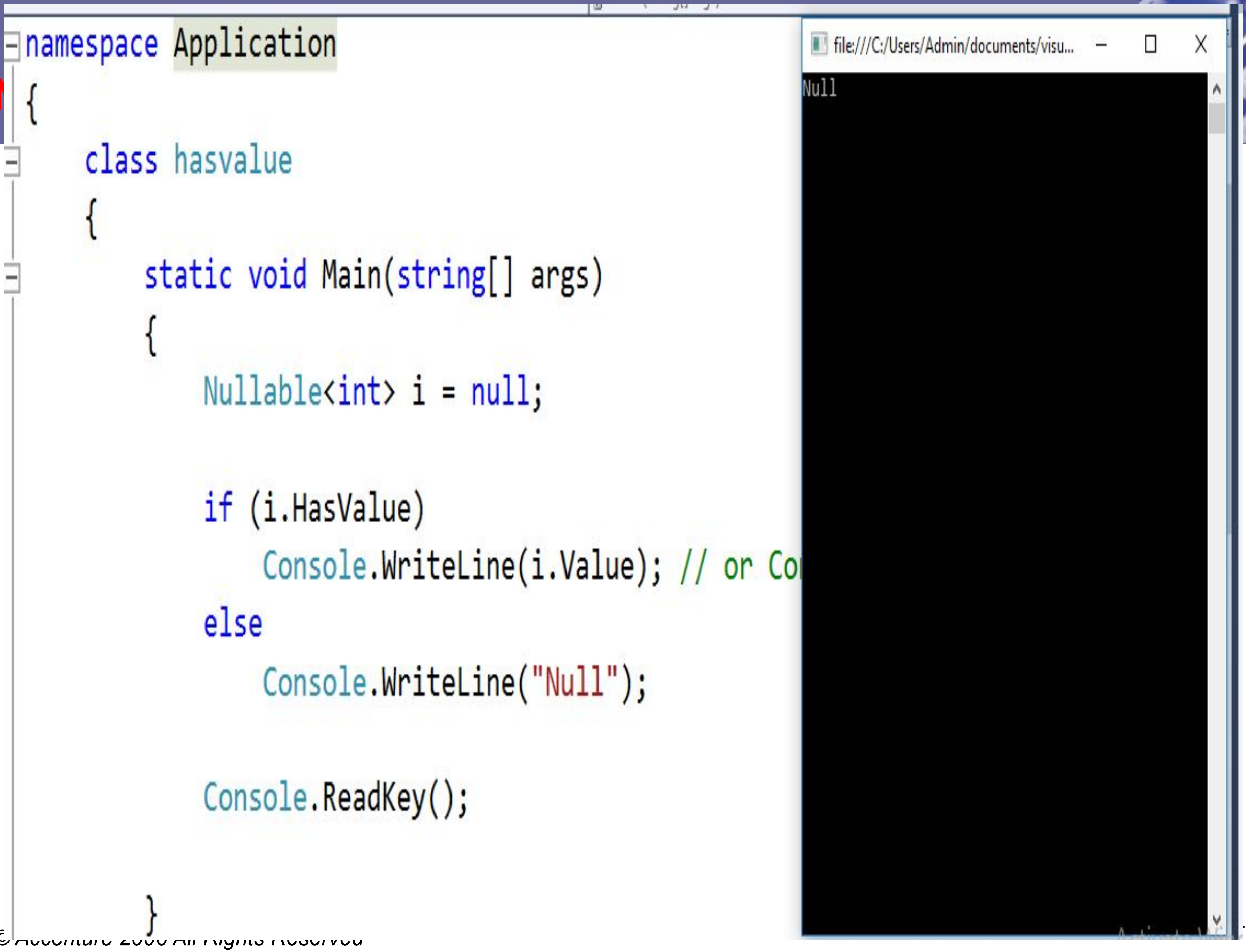
range: -2147483648 to 2147483647 or a null value

Nullable type

- HasValue Property

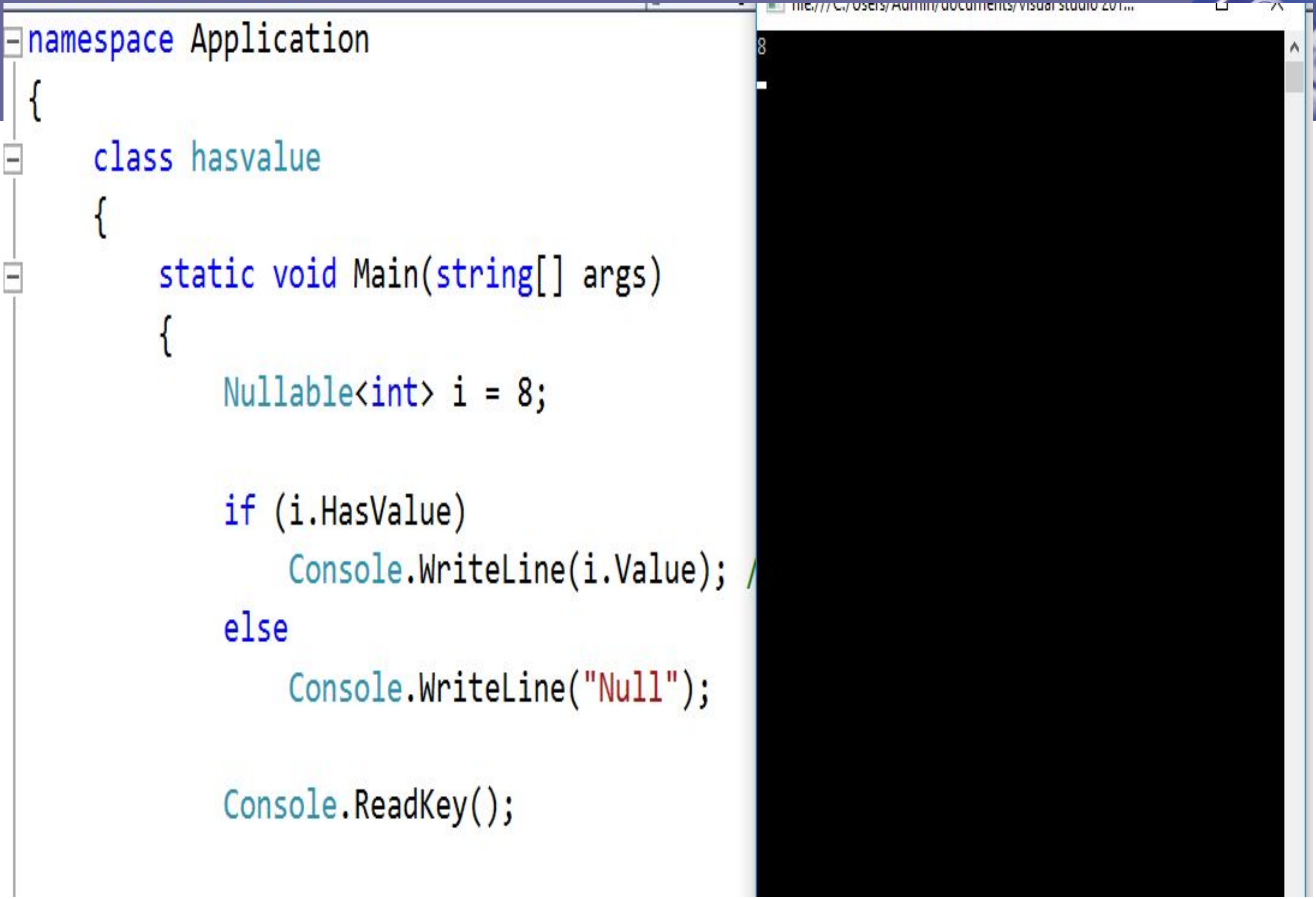
The HasValue returns **true if the object has been assigned a value**;

if it has **not been assigned any value or has been assigned a null value, it will return false.**



```
namespace Application
{
    class hasvalue
    {
        static void Main(string[] args)
        {
            Nullable<int> i = null;

            if (i.HasValue)
                Console.WriteLine(i.Value); // or Co
            else
                Console.WriteLine("Null");

            Console.ReadKey();

        }
```

```
Null
```



```
namespace Application
{
    class hasvalue
    {
        static void Main(string[] args)
        {
            Nullable<int> i = 8;

            if (i.HasValue)
                Console.WriteLine(i.Value); /
            else
                Console.WriteLine("Null");

            Console.ReadKey();

```

```
8
```

Shorthand Syntax for Nullable Types

? operator:

e.g. int?, long? **instead of** using Nullable<T>

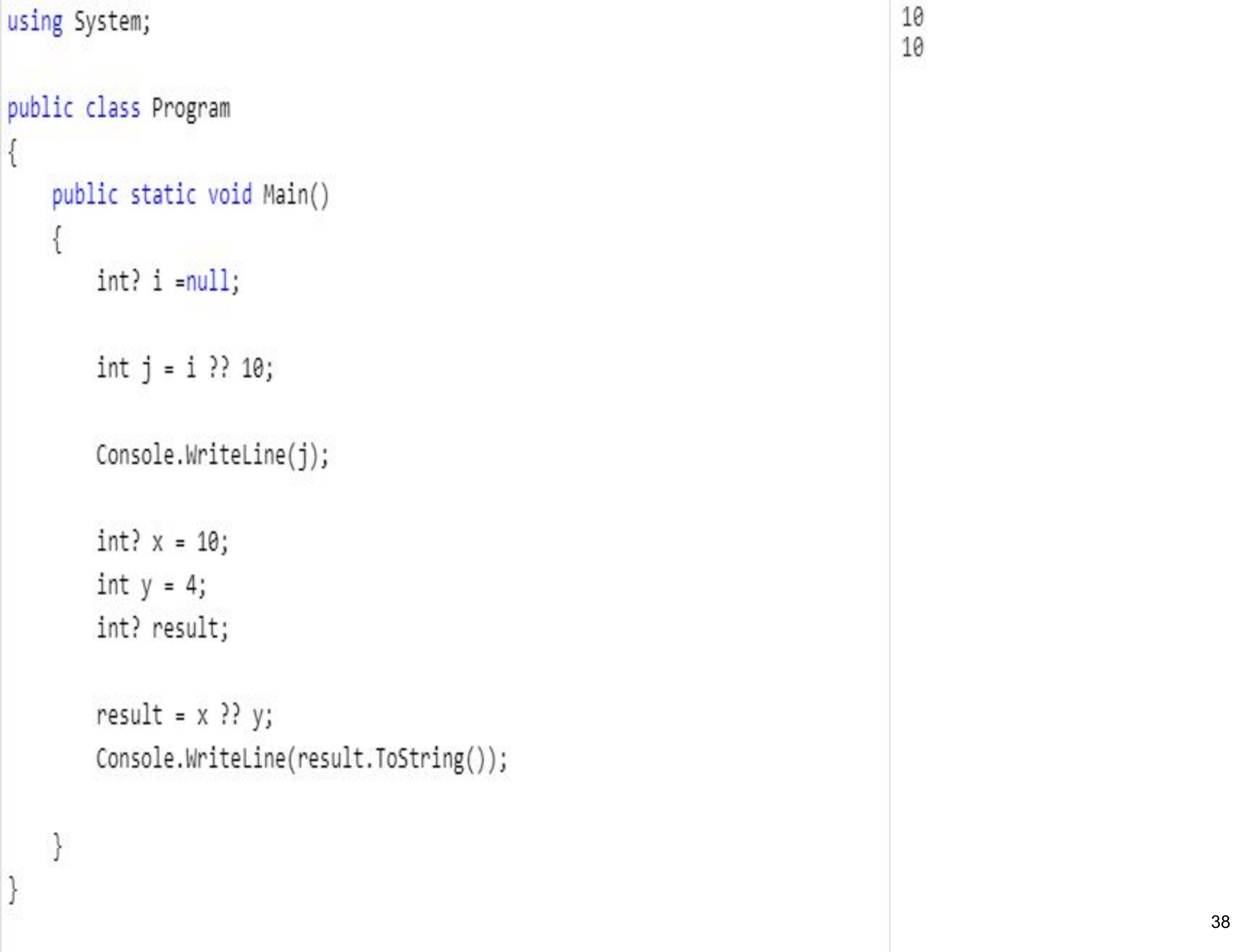
Example: **Nullable<int> x = null;**

```
int? x = null;
double? D = null;
```

Null-Collation(??) Operator

- To assign a **nullable type to non-nullable type.**
- It is also used to define a **default value for nullable value** **types or** **reference types.**
- It returns the **left-hand operand** **if the** **operand is not null;** otherwise, it returns the **right operand.**

```
        int? i = null;
        int j = i ?? 0;    so j =0
Example 2
   int? K=8; int? N=9
    int M = K ?? N                  int? Z=N ?? 0
      What is M =?                 What is Z= ?
```



```
using System;


public class Program
{
    public static void Main()
    {
        int? i =null;

        int j = i ?? 10;

        Console.WriteLine(j);

        int? x = 10;
        int y = 4;
        int? result;

        result = x ?? y;
        Console.WriteLine(result.ToString());


    }
}
```

```
10
10
```

Operators

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

Arithmetic Operators

Operator	Description	Example Let A=10, B=20
+	Adds two operands	A + B = 30
-	Subtracts second operand from the first	A - B = -10
*	Multiplies both operands	A * B = 200
/	Divides numerator by de-numerator	B / A = 2
%	Modulus Operator and remainder of after an integer division	B % A = 0

Relational Operators

Operator	Description	Example Let A=10 B=20
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is false.
!=	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 false.
<	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 false.
<=	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.

Logical Operators

Operator	Description	Example Let A=0 B=1
&&	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.

Program:Logical & conditional operator

Example: Static login page

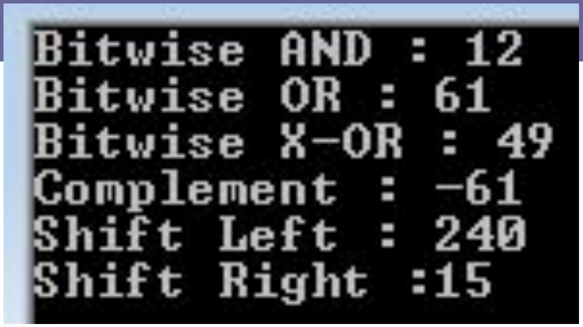
```
static void Main()
{   string username;
    Console.WriteLine("Enter username");
    username = Console.ReadLine();
    int password;
    Console.WriteLine("Enter Password");
    password = Convert.ToInt32(Console.ReadLine());
    string valid = (username == “xyz" && password ==
    123) ? "Welcome" : "Incorrect Username or
    Password";
    Console.WriteLine(valid);
    Console.ReadKey();        }
```

Bitwise Operators

Operator	Description	Example(A=60, B=13)
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = 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 = 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 = 15, which is 0000 1111

Bitwise-Program

```
static void Main()
{
            int a = 60;      /* 60 = 0011 1100 */
            int b = 13;     /* 13 =  0000 1101 */
            int c = 0;
    c = a & b;      /* 12 = 0000 1100 */ Console.WriteLine("Bitwise AND : {0}", c);
   c = a | b;       /* 61 = 0011 1101 */   Console.WriteLine("Bitwise OR : {0}", c);
   c = a ^ b;       / * 49 = 0011 0001 */  Console.WriteLine("Bitwise X-OR : {0}", c);
 c = ~a;          /*-61 = 1100 0011 */       Console.WriteLine("Complement : {0}", c);
  c = a << 2;   /* 240 = 1111 0000 */      Console.WriteLine("Shift Left : {0}", c);
   c = a >> 2;   /* 15 = 0000 1111 */       Console.WriteLine("Shift Right :{0} ", c);
            Console.ReadKey();
}
```



```
Bitwise AND : 12
Bitwise OR : 61
Bitwise X-OR : 49
Complement : -61
Shift Left : 240
Shift Right :15
```

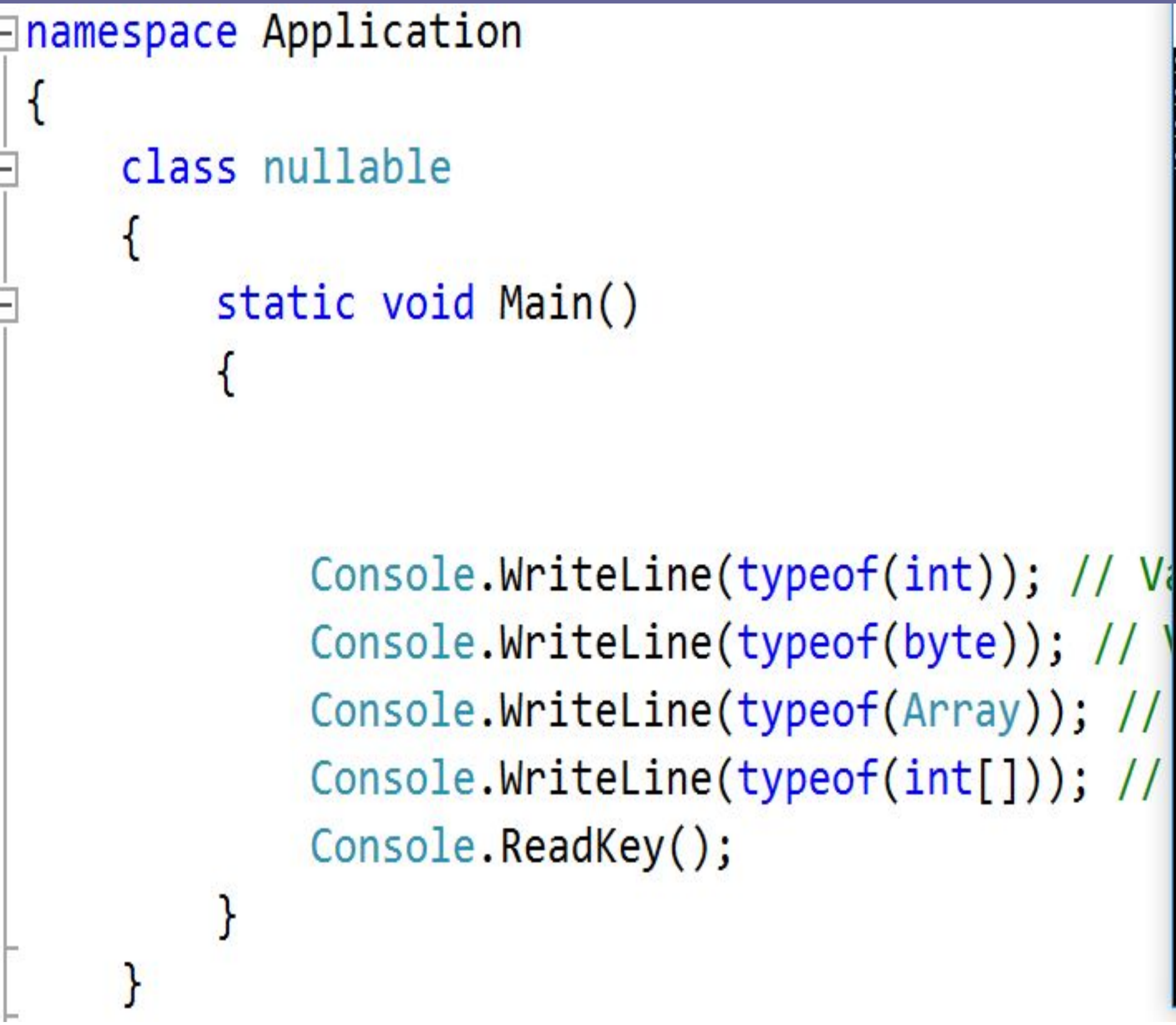
p	q	p & q	p | q	p^q
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Miscellaneous Operators

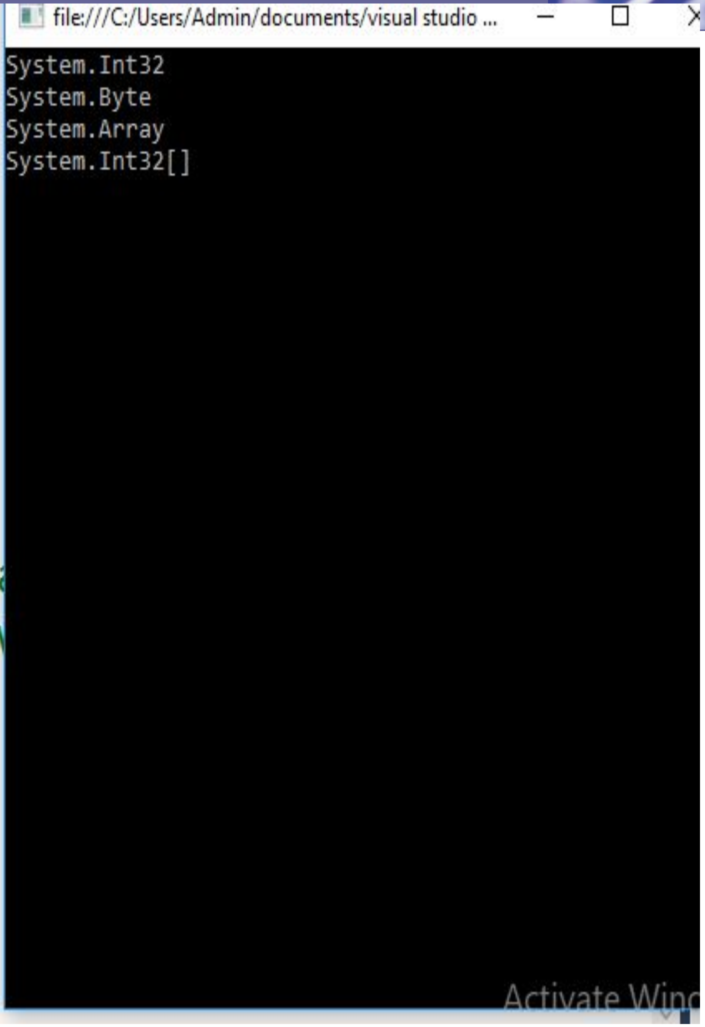
Operator	Description	Example
sizeof()	Returns the size of a data type.	sizeof(int), returns 4.
typeof()	Returns the type of a class.	typeof(int), returns System.Int32
&	Returns the address of an variable.	&a; returns actual address of the variable.
*	Pointer to a variable.	*a; creates pointer named 'a' to a variable.
? :	Conditional Expression	Condition? True : False
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;

typeof OperatorReturns the type of a class or Struct.



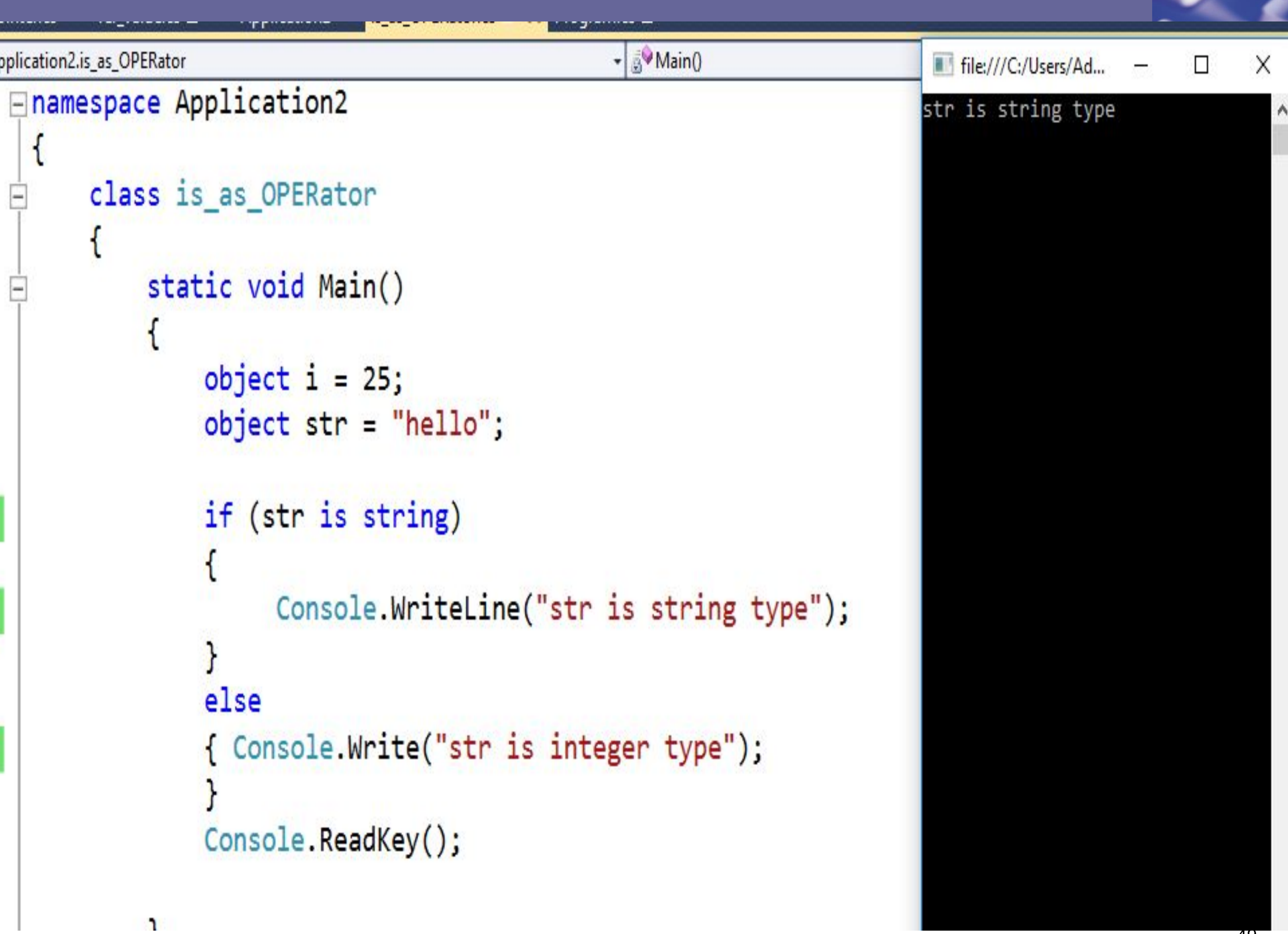
```
namespace Application
{
    class nullable
    {
        static void Main()
        {


            Console.WriteLine(typeof(int)); // Va
            Console.WriteLine(typeof(byte)); // V
            Console.WriteLine(typeof(Array)); //
            Console.WriteLine(typeof(int[])); //
            Console.ReadKey();
        }
    }
```



```
System.Int32
System.Byte
System.Array
System.Int32[]
```

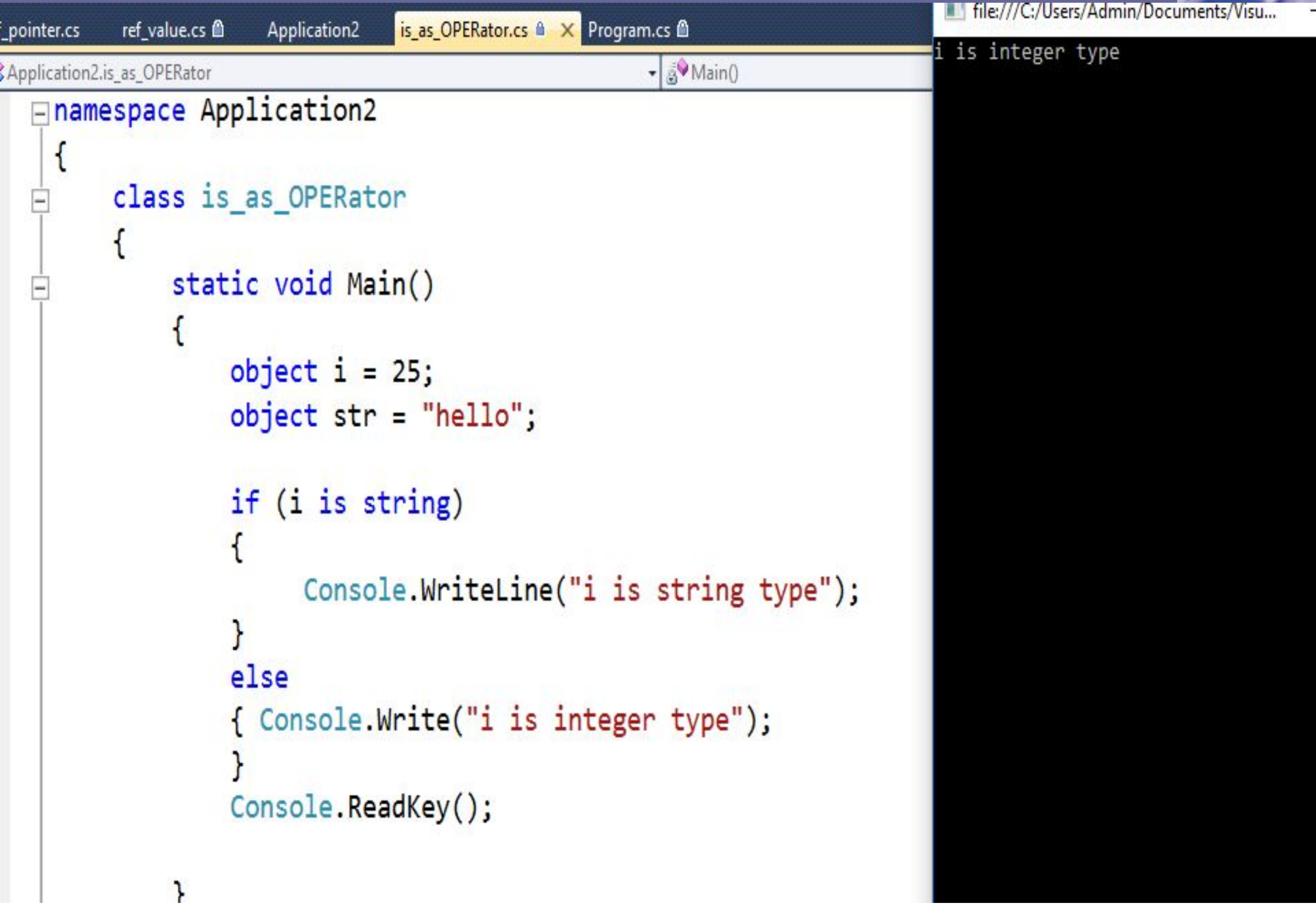
Is Operator

- The ‘is’ operator in C# is used to check the **object type** and it **returns a bool value:**
- It returns **true** if the **object is the same as Class type** and **false if not**.



```
namespace Application2
{
    class is_as_OPERator
    {
        static void Main()
        {
            object i = 25;
            object str = "hello";

            if (str is string)
            {
                 Console.WriteLine("str is string type");
            }
            else
            { Console.Write("str is integer type");
            }
            Console.ReadKey();


        }
```

```
str is string type
```



```
namespace Application2
{
    class is_as_OPERator
    {
        static void Main()
        {
            object i = 25;
            object str = "hello";

            if (i is string)
            {
                Console.WriteLine("i is string type");
            }
            else
            { Console.Write("i is integer type");
            }
            Console.ReadKey();

        }
```

```
i is integer type
```

As Operator

- The **as** operator is used to perform **conversion between** compatible **reference types** or **Nullable types.**
- The ‘**as**‘ operator does the same job of ‘**is**‘ operator but the difference is instead of **bool,** it returns the **object** if they are compatible to that type, else it returns **null**.
  - Exmple: object i = 25;

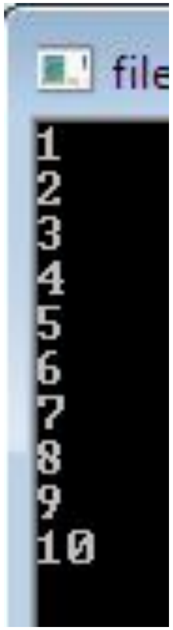
```
object str = “Hello”;
 string str1 = i as string;
 string str2 = str as string;
 Console.Write(str1);    // null
 Console.Write(str2);    // Hello
```

Control Statements

- Controls the flow of execution
- Types of control statements:
  - Conditional Statements
  - Iterative Statements/ Loops
  - Jump Statements [e.g. goto, break, continue]

Goto Statement

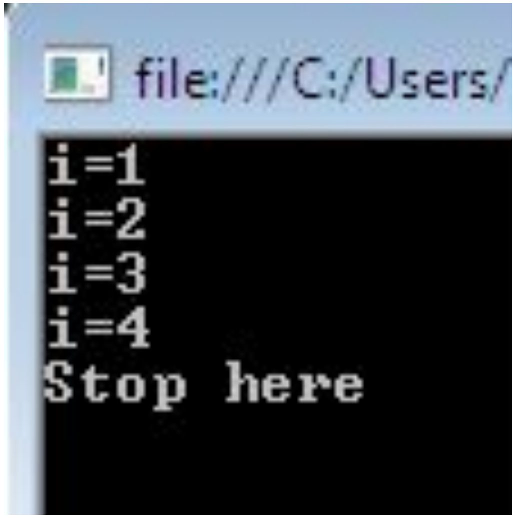
```
using System;
namespace HelloProgram
{
    class Goto
    {
        static void Main()
        {
            int i = 1;
        up:
            Console.WriteLine(i);
            i++;
            if (i <= 10)
                goto up;
            Console.ReadKey(); } } }
```



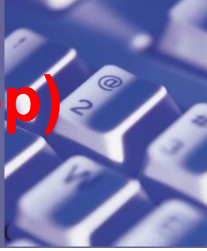
```
1
2
3
4
5
6
7
8
9
10
```

Break Statement(Stop the loop)

```
using System;
namespace HelloProgram
{
    class Break
    {
        static void Main()
        {
            for (int i = 1; i <= 10; i++)
            {
                if (i == 5)
                            break;
                Console.WriteLine("i="+i);
            }
            Console.WriteLine("Stop here");
            Console.ReadKey(); } } }
```

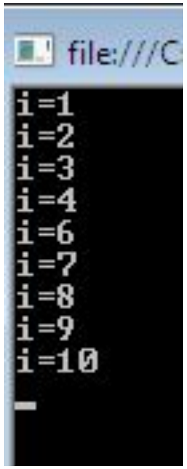


```
i=1
i=2
i=3
i=4
Stop here
```



Continue Statement(Control jumps to the beg. Of loop)

```
using System;
namespace HelloProgram
{
    class Continue
    {
        static void Main()
        {
            for (int i = 1; i <= 10; i++)
            {
                if (i == 5)
                            continue;
                      Console.WriteLine("i=" + i);


                  }
             Console.ReadKey(); } } }
```



```
i=1
i=2
i=3
i=4
i=6
i=7
i=8
i=9
i=10
_
```

Array

- It is used to store **collection of data**
- **Store a fixed size sequential collection** of element of **same data type**.
- Instead of declaring **individual variable**, you **declare one array.**
- Array elements are accessed by its index.
- Types of array:
  1. Single Dimensional Array
  2. Multi Dimensional Array
  3. Jagged Array

1-Dimensional array

Syntax to declare 1-D array:

- Int[] arr1= new int[5];

  arr1[0]=10;
  arr1[1]=20;
  arr1[2]=30;

- Int[] arr2= new int[5] {1,2,3,4,5};

- Int[] arr3= new int[] {1,2,3,4,5};

1-Dimensional array

- Program 1: let take element from user and display
- Program 2: Write a program to find greatest element from integer array.

1-Dimensional array

- Program 1: **let take 10 element from user and display**

```
        int[] arr = new int[10];
        int i;
  Console.Write("\n\nRead and Print elements of an array:\n");
        Console.Write("Input 10 elements in the array :\n");
        for(i=0; i<10; i++)
        {
              arr[i] = Convert.ToInt32(Console.ReadLine());
        }
        Console.Write("\nElements in array are: ");
         for(i=0; i<10; i++)
        {
               Console.Write("{0} ", arr[i]);
        }
```

1-Dimensional Array

Program 2: Write a program to find greatest element from integer array.

1-Dimensional array

Exercises

- **Write a program to find sum of all even elements of an array.**

Multi Dimensional array

- It is also known as rectangular arrays in C#.
- It can be **two dimensional or three dimensional.**
- The data is stored in tabular form **(row * column)** which is also known as **matrix.**

Syntax to declare Multi Dimensional array:

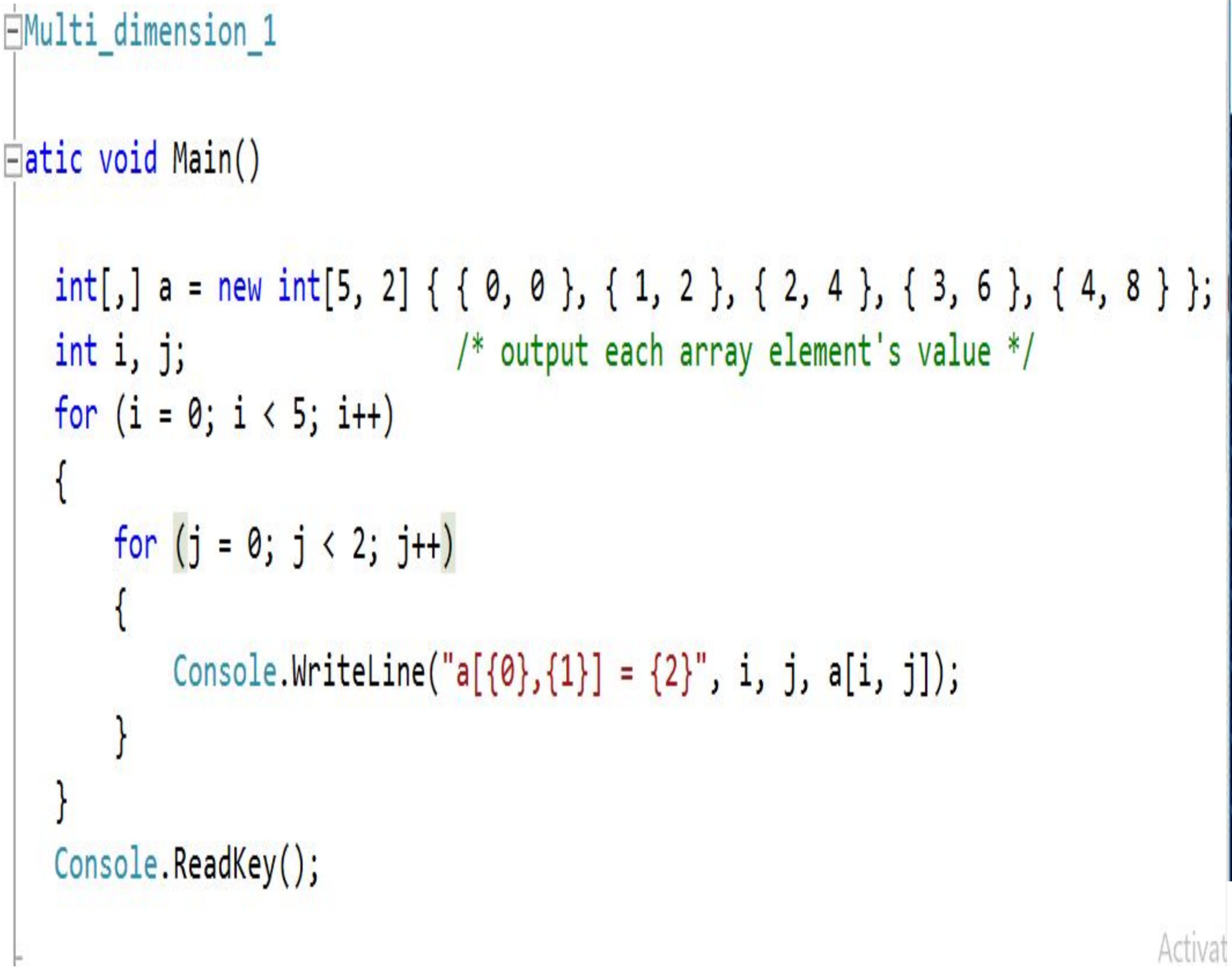
- To create multidimensional array, use comma inside the square brackets.
- int[,] arr=new int[3,3];   //declaration of 2D array
- int[,] ar1=new int[4,3];   //declaration of 2D array

Multi Dimensional array

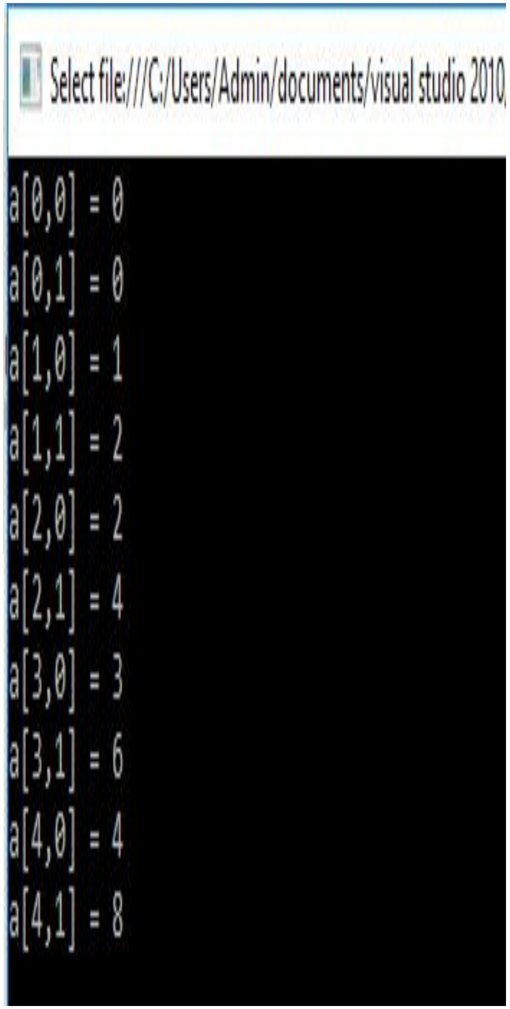
More syntax:

- int[,] array2D = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
- int[,] array2Da = new int[4, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
- string[,] array2Db = new string[3, 2] { { "one", "two" }, { "three", "four" }, { "five", "six" } };

Multi Dimensional array(Store Elements into 2D array statically)



```
Multi_dimension_1

atic void Main()

  int[,] a = new int[5, 2] { { 0, 0 }, { 1, 2 }, { 2, 4 }, { 3, 6 }, { 4, 8 } };
  int i, j;                /* output each array element's value */
  for (i = 0; i < 5; i++)
  {
      for (j = 0; j < 2; j++)
      {
          Console.WriteLine("a[{0},{1}] = {2}", i, j, a[i, j]);
      }
  }
  Console.ReadKey();
```



Select file:///C:/Users/Admin/documents/visual studio 2010

```
a[0,0] = 0
a[0,1] = 0
a[1,0] = 1
a[1,1] = 2
a[2,0] = 2
a[2,1] = 4
a[3,0] = 3
a[3,1] = 6
a[4,0] = 4
a[4,1] = 8
```

Multi Dimensional array

- **Program 2**: Store elements into 2 Dimensional array(Take user input) and display

Multi Dimensional array

- Exercises
- <u>**Program 1**</u>: **Addition of two matrix2*2**
- **<u>Program 2:</u>** Multiplication of two 2-dimentional matrix

Jagged Array

- It is an array **whose elements are arrays.**
- A jagged array is sometimes called an "**array of arrays."**
- **A special type of array is introduced in C#.**
- **A Jagged Array is an array of an array in which the length of each array index can be different.**

Jagged Array

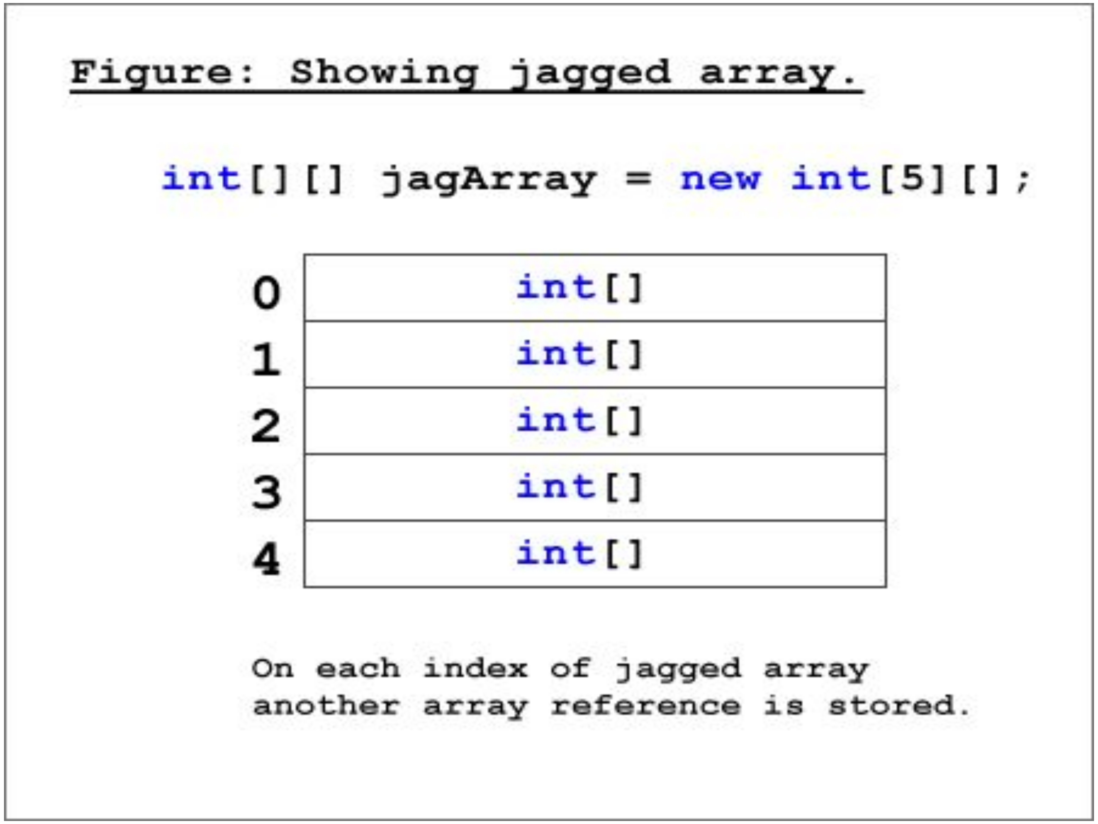


Figure: Showing jagged array.

```
int[][] jagArray = new int[5][];
```

0	int[]
1	int[]
2	int[]
3	int[]
4	int[]

On each index of jagged array
another array reference is stored.

Jagged Array-Syntax

Syntax:

- A jagged array **is initialized** with two square brackets [][].
- The **first bracket** specifies the **rows** of an array.
- The **second bracket** specifies the **column** of the array which is going to be stored as values.

Jagged array with multi dimensional array:

```
1. int[][] jaggedArray = new int[3][]; //declaration
   jaggedArray[0] = new int[3];
   jaggedArray[1] = new int[5];
   jaggedArray[2] = new int[2];

2.  jaggedArray[0] = new int[] { 3, 5, 7 };
    jaggedArray[1] = new int[] { 1, 0, 2, 4, 6 };
    jaggedArray[2] = new int[] { 1, 6 };
```

Q: What will be the output of following statement?

```
Console.WriteLine(jaggedArray[0][1]);
```

Jagged Array-Syntax

Jagged array with multi dimensional array:

- Second bracket [,] indicates multi-dimension.
- Example:

```
int[][,] intJaggedArray = new int[3][,];
intJaggedArray[0] = new int[3, 2] { { 1, 2 }, { 3, 4 }, { 5, 6
} };
intJaggedArray[1] = new int[2, 2] { { 3, 4 }, { 5, 6 } };
intJaggedArray[2] = new int[2, 2];
Console.WriteLine(intJaggedArray[0][1,1]); // 4
Console.WriteLine(intJaggedArray[1][1,0]); // 5
Console.WriteLine(intJaggedArray[1][1,1]); // 6
```

Jagged Array-Example

Program: Generate Pascal Triangle using jagged array.

```
1
1  1
1  2  1
1  3  3  1
1  4  6  4  1
1  5  10  10  5  1
…..
```

J(Coulmn)

I (Row)

	0	1	2	3
0	1			
1	1	1		
2	1	2	1	
3	1	3	3	1

A[3][1]=3=

A[3][1]=a[2][0]+a[2][1]

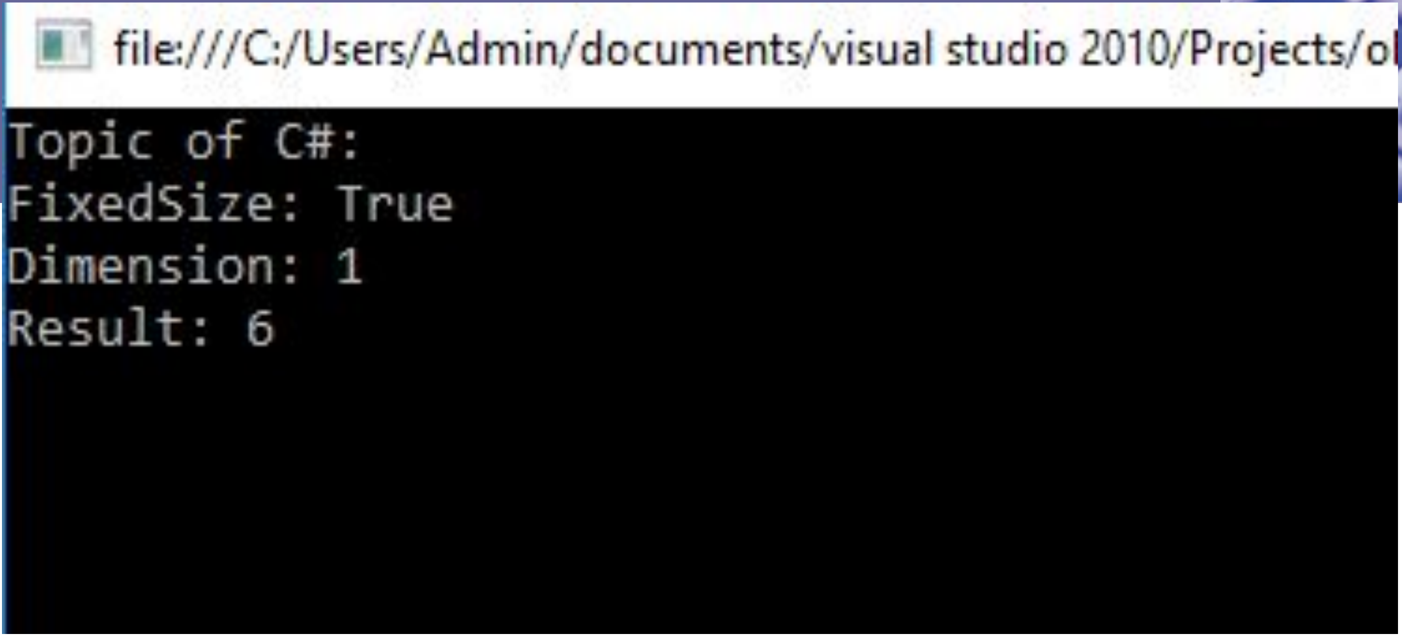
A[i][j]=a[i-1][j-1] + a[i-1][j]

Array Helper Class

- Properties of Array class:
  - IsFixedSize: get value indicating whether the array has fixed size
  - Length: Total number of elements in all the dimensions of the Array.
  - Rank: Get the dimensions(Coulmns) of the array(For 1 D array it returns 1, for 2D array it returns 2).

▪ MOST COMMON PROPERTIES OF ARRAY CLASS

Properties	Explanation	Example
Length	Returns the length of array. Returns integer value.	`int i = arr1.Length;`
Rank	Returns total number of items in all the dimension. Returns integer value.	`int i = arr1.Rank;`
IsFixedSize	Check whether array is fixed size or not. Returns Boolean value	`bool i = arr.IsFixedSize;`
IsReadOnly	Check whether array is ReadOnly or not. Returns Boolean value// Bydefault Returns false for all arrays.	`bool k = arr1.IsReadOnly;`



file:///C:/Users/Admin/documents/visual studio 2010/Projects/ol

```
Topic of C#:
FixedSize: True
Dimension: 1
Result: 6
```

```
tatic void Main()

    string[] topic;

    // allocating memory for topic.
    topic = new string[] { "Array, ", "String, ", "Stack, ", "Queue, ", "Exception, ", "Operators" };

    // Displaying Elements of the array
    Console.WriteLine("Topic of C#:");
    Console.WriteLine("FixedSize: " + topic.IsFixedSize);
    Console.WriteLine("Dimension: " + topic.Rank);
    Console.WriteLine("Result: " + topic.Length);
    Console.ReadKey();
```

Array Helper Methods

- Clear
- Copy(Array, Array,Int32)
- GetLength
- GetLowerBound //First row number of array
- GetUpperBound //Last row number of array
- GetType
- GetValue(Int32)
- IndexOf(Array, Object)
- Reverse(Array)
- SetValue(Object, Int32)
- Sort(Array)

Foreach loop

- It executes a block of code on **each element in an array**.
- The foreach loop is **useful for traversing each item in an array** or a collection of items and displayed one by one.

Foreach loop

- In **foreach loop** the **variable of the loop** will be **same as the type the array.**

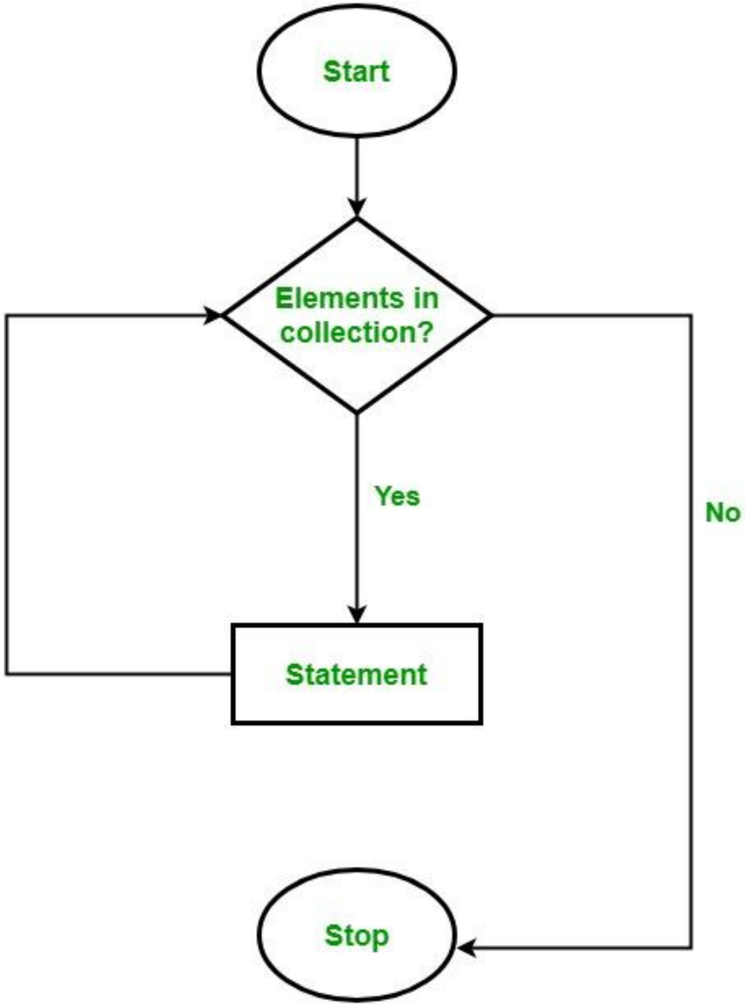
- The **foreach** statement repeats a group of statements for **each element in an array**

- In **foreach loop**, You do not need to specify the loop bounds minimum or maximum.

- **Syntax:**

```
foreach (data_type var_name in collection_variable/Array_variable)
      {
            // statments
      }
```

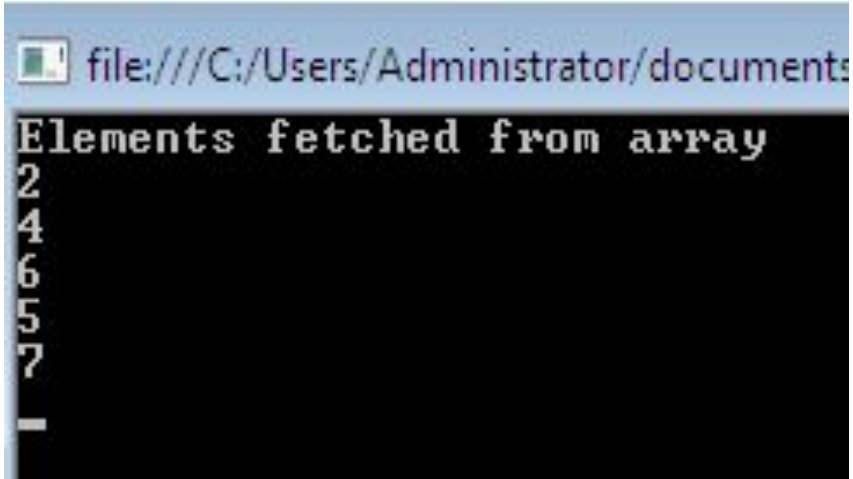
Foreach loop



Flowchart of foreach loop

Foreach loop Program

```
static void Main()
        {
            int[] arr = new int[] { 2,4,6,5,7};
         Console.WriteLine("Elements fetched from array");
            foreach (int item in arr)
            {
                Console.WriteLine(item);
            }
            Console.ReadKey();
        }
```



file:///C:/Users/Administrator/documents

```
Elements fetched from array
2
4
6
5
7
_
```

Difference between for loop and foreach loop

- for loop executes a statement or a block of statement **until the given condition is false.** Whereas *foreach* loop executes a statement or a block of statements **for each element present in the array** and there is **no need to define the minimum or maximum limit.**
- In *for loop*, **we iterate the array in both forward and backward directions,** e.g from index 0 to 9 and from index 9 to 0. But in the **foreach loop, we iterate an array only in the forward direction**, not in a backward direction.

Function

It is a group of statements to perform a task.

Syntax:

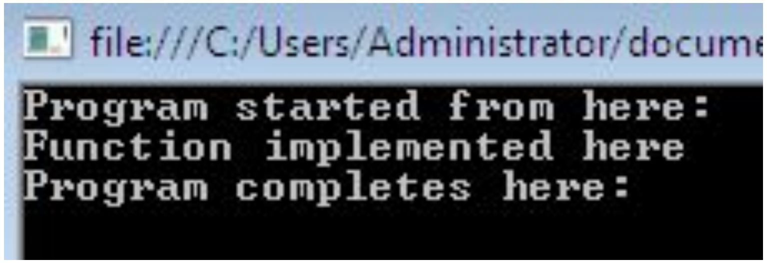
```
 <Access Specifier> <Return Type> Function Name (Parameters)
   {
               //function body with return statement
    }
```

A function consists of the following components:

- **Function name:** It is unique name which is used to call function.
- **Return type:** It is used to specify the data type of function return value.
- **Body:** It is a block that contains executable statements.
- **Access specifier:** It is used to specify function accessibility in the application. **Publlic/Private**
- **Parameters:** It is a list of arguments can pass to function during call.

Function

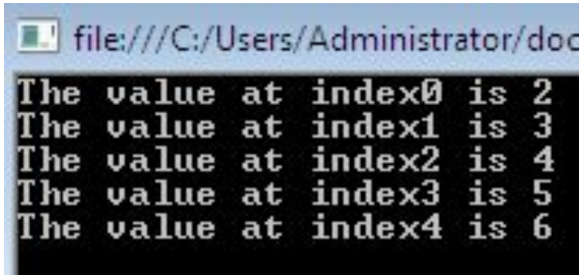
```
static void Main()
        {
            Console.WriteLine("Program started from here:");
            function1();
            Console.WriteLine("Program completes here:");
            Console.ReadKey();
        }
        static void function1()
        {
            Console.WriteLine("Function implemented here");
        }
```



```
Program started from here:
Function implemented here
Program completes here:
```

Array As Function Argument

```
static void Main(string[] args)
{
            int[] arr = { 1, 2, 3, 4, 5 };
            PrintArray(arr);
            Console.ReadKey();
}
static void PrintArray(int[] array)
{
     for (int i = 0; i < array.Length; i++)
      {
             array[i] = array[i] + 1;
             Console.WriteLine("The value at index{0} is {1}", i, array[i]);
       }
}
```



```
The value at index0 is 2
The value at index1 is 3
The value at index2 is 4
The value at index3 is 5
The value at index4 is 6
```

“Params” Keyword

- **In simple function we can allow fixed number of function arguments**

Program:

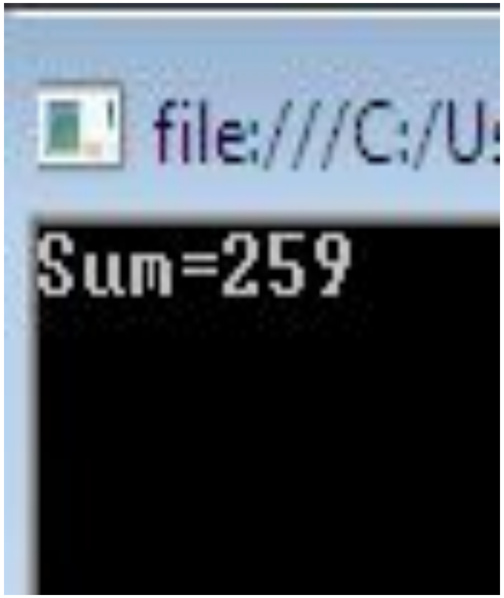
```
class  Program
{
         static  void Main(string[] args)
        {
                int y = Add (12,14,43);
        }
        public static  int Add(int num1, int num2, int num3)
        {
                return  num1+num2+num3;
        }
}
```

- **But in this simple program we want to add more parameters at run time then use “params keyword”**

- **Unknown number of parameters passing to array**

“Params” Keyword-Program1

```
static void Main(string[] args)
{
   int y = Add(12, 14, 43, 34, 56, 100);
  Console.WriteLine("Sum=" + y);
   Console.ReadKey();
 }
public static int Add(params int[] ListNumbers)
  {
      int total = 0;
     foreach (int i in ListNumbers)
            {
                total = i + total;
            }
            return total;
 }
```

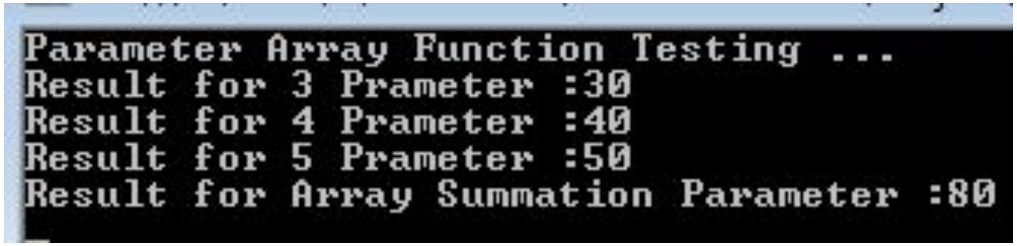


```
Sum=259
```

“Params” Keyword- Program2

```
static int Add(params int[] nums)
      {
         int total=0;
         foreach(int i in nums)
         {
            total = total+i;
         }
         return total;
      }
```

```
Main()
{
int result = 0;
 result = Add(10, 10, 10);
 result = Add(10, 10, 10, 10);
 result = Add(10, 10, 10, 10, 10);
 int[] x = { 10, 10, 10, 10, 10, 10, 10,
     10 };
 result = Add(x);
}
```

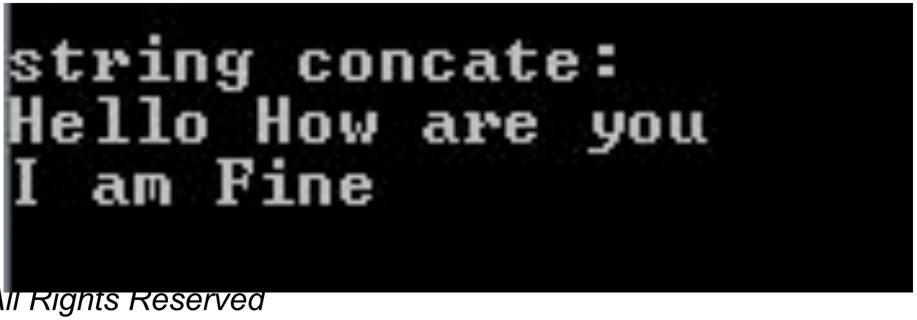


```
Parameter Array Function Testing ...
Result for 3 Prameter :30
Result for 4 Prameter :40
Result for 5 Prameter :50
Result for Array Summation Parameter :80
```

"Params" Keyword- Program3(with String array)

```
static void Main()
{
   Console.WriteLine("string
    concate:");
   ADDparameters1("Hello", " ",
    "How", " ", "are", " ", "you");

   ADDparameters1("I"," ","am","
    ","Fine");
}
```

```
public static void
    ADDparameters1(params string[]  str)
{
         string add = "";
         foreach (string arg in str)
         {
             add += arg;
         }
         Console.WriteLine(add);
}
```



```
string concate:
Hello How are you
I am Fine
```

Function Parameters

Types of function parameters:

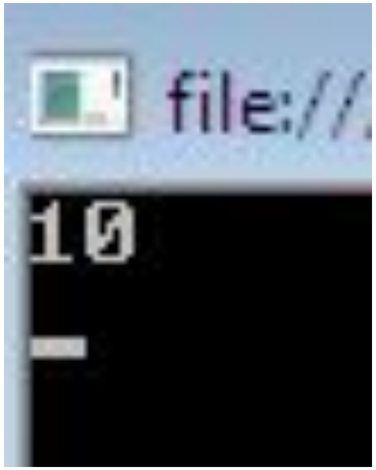
- Value Parameter (Call by value)(In parameter)
- Out Parameter
- Reference Parameter

Value Parameter (Call by value)(In parameter)

- Allows a method to **<u>pass values to value through arguments</u>** given in method call
- This type of assignment **affects only local storage**.
- It has **no affect on actual arguments** being assigned in method call.

Value Parameter (Call by value)(In parameter)-Program

```
using System;
namespace HelloProgram
{
    class CallByValueParameter
    {
        static void add(int v)
        {
            v++;
        }
        static void Main()
        {
            int value = 10;
            add(value);
            Console.WriteLine(value);
            Console.ReadKey();  } } }
```



```
10
_
```

Reference Parameter-Program

- To pass parameter by reference to method 2 ways.

  1.By reference

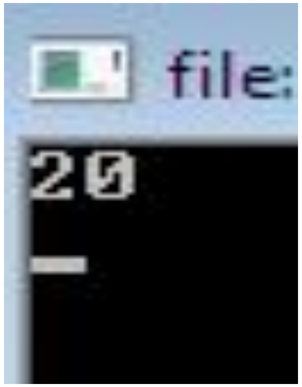
  2. using out

1. **By Reference**

- In this **ref keyword** is used.
- The ref **keyword indicates a value that is passed by reference**

1. Parameter passing by ref keyword

```
using System;
namespace HelloProgram
{
    class ReferenceParameter
    {
        static void set_value(ref int v)
        {
            v = 20;
        }
        static void Main()
        {
            int value = 10;
            set_value(ref value); // method calling
            Console.WriteLine(value);
            Console.ReadKey();   } } }
```



```
20
_
```

Output Parameter

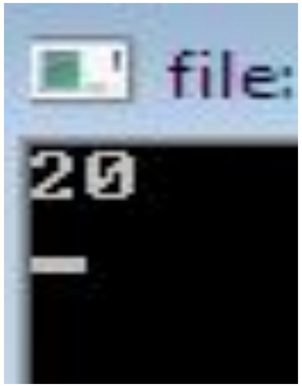
- The **out** is a keyword in C# which is used for the passing the arguments to methods as a **reference type..**
- **Declare with out modifier.**

Difference between ref and out Keyword

- **Every output parameter of method must be assigned inside the method before method return the output.**

Output Parameter-Program-1

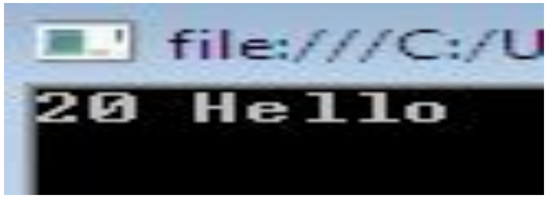
```
using System;
namespace HelloProgram {
    class OutParameter {
        static void set_value(out int v)
    {
            v = 20;
        }
        static void Main()
        {
            int value=5;
            set_value(out value); // method calling
            Console.WriteLine(value);
            Console.ReadKey(); } } }
```



```
20
_
```

Output Parameter-Program-2

```
using System;
namespace HelloProgram{
    class OutParameter
{
         static void value(out  int  i,out string s1, out string s2)
        {
           i =20;
           s1 ="Hello";
           s2 = null;
        }
        static void Main()
        {
           int v;
           string str1, str2;
           value(out v, out str1, out str2); // method calling
           Console.WriteLine(v + " " + str1 + " " + str2 );
           Console.ReadKey();  } } }
```

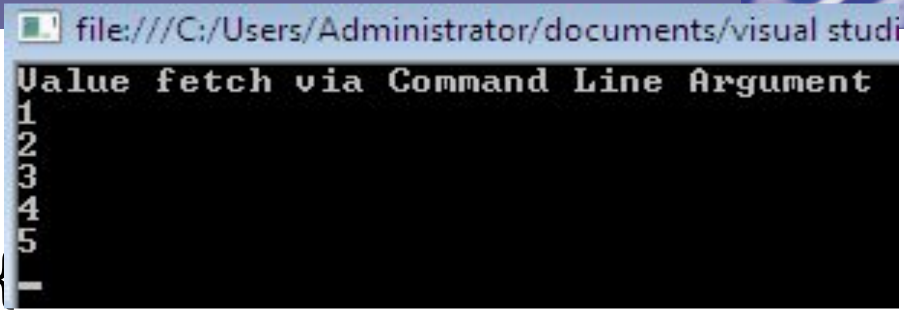


```
20 Hello
```

Summary Out and Ref Parameters

- Both are used to pass **parameters by reference** .
- Using ref keyword we are passing parameter value to methods and that **value is used inside the function &** after **executing function updated parameter value is returned back**.
- Using **out we are passing parameter value to methods but that value is not considered inside method.** We have to **initialize parameter inside method** and after executing the function updated **parameter value is returned by function.**

Command Line Argument-Program-1

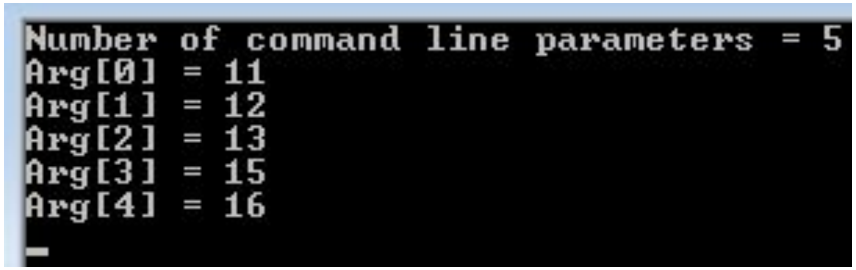


file:///C:/Users/Administrator/documents/visual studi

```
Value fetch via Command Line Argument
1
2
3
4
5
_
```

```
using System;
namespace HelloProgram{
class CommandLineArgument{
static void Main(string[] args)
{
  Console.WriteLine("Value fetch via Command Line
    Argument");
   for (int i = 0; i < args.Length; i++)
   {
            Console.WriteLine(args[i]);
    }
    Console.ReadKey(); } } }
```

Command Line Argument-Program-2

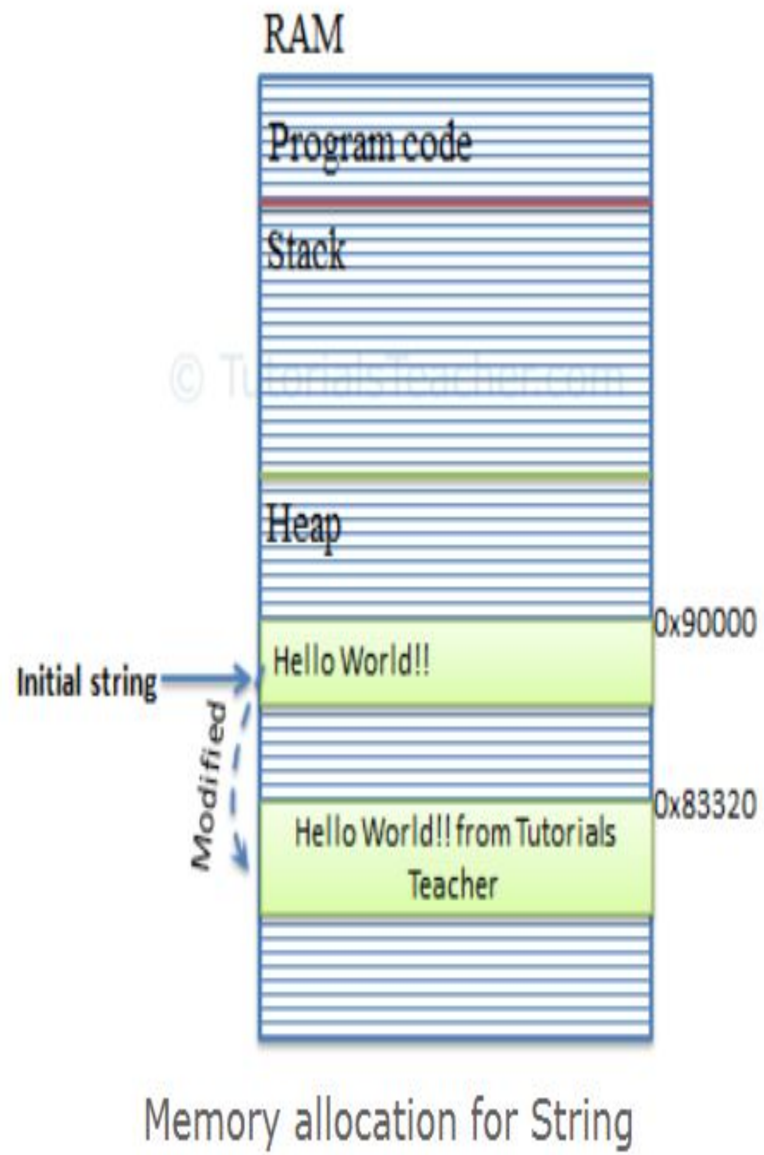


```
Number of command line parameters = 5
Arg[0] = 11
Arg[1] = 12
Arg[2] = 13
Arg[3] = 15
Arg[4] = 16
_
```

```
using System;
namespace HelloProgram
{
    class CommandLineArgument
    {
        static void Main(string[] args)
        {
Console.WriteLine("Number of command line parameters = {0}",
     args.Length);
            for (int i = 0; i < args.Length; i++)
            {
                Console.WriteLine("Arg[{0}] = {1}", i, args[i]);
            }
            Console.ReadKey();
        }
    }
}
```

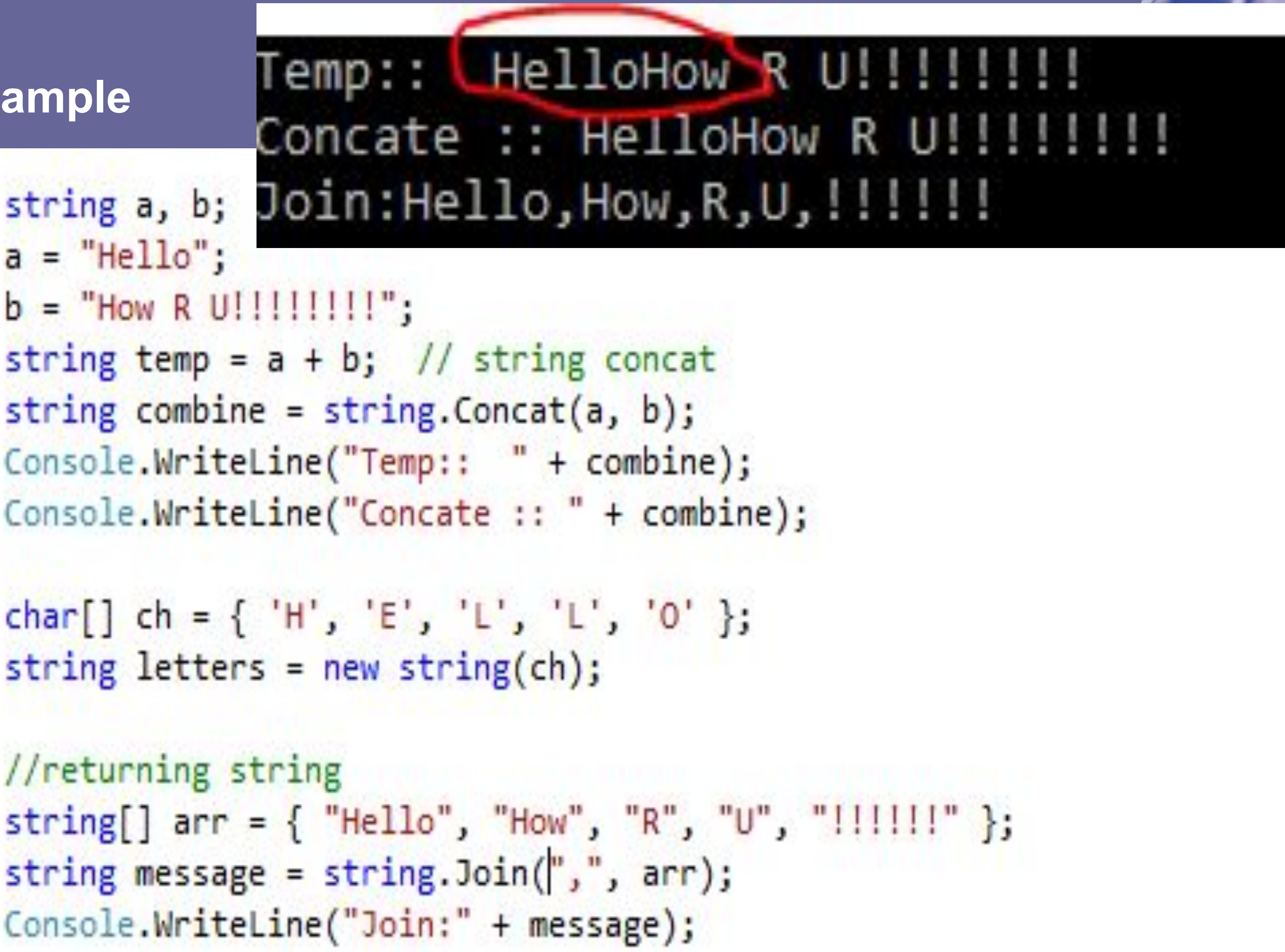
String

- String are immutable.
- Can't Change a string after created it.
- When we update string it **New copy** will be created and stored at some other **memory location** rather than going and updating the same one.



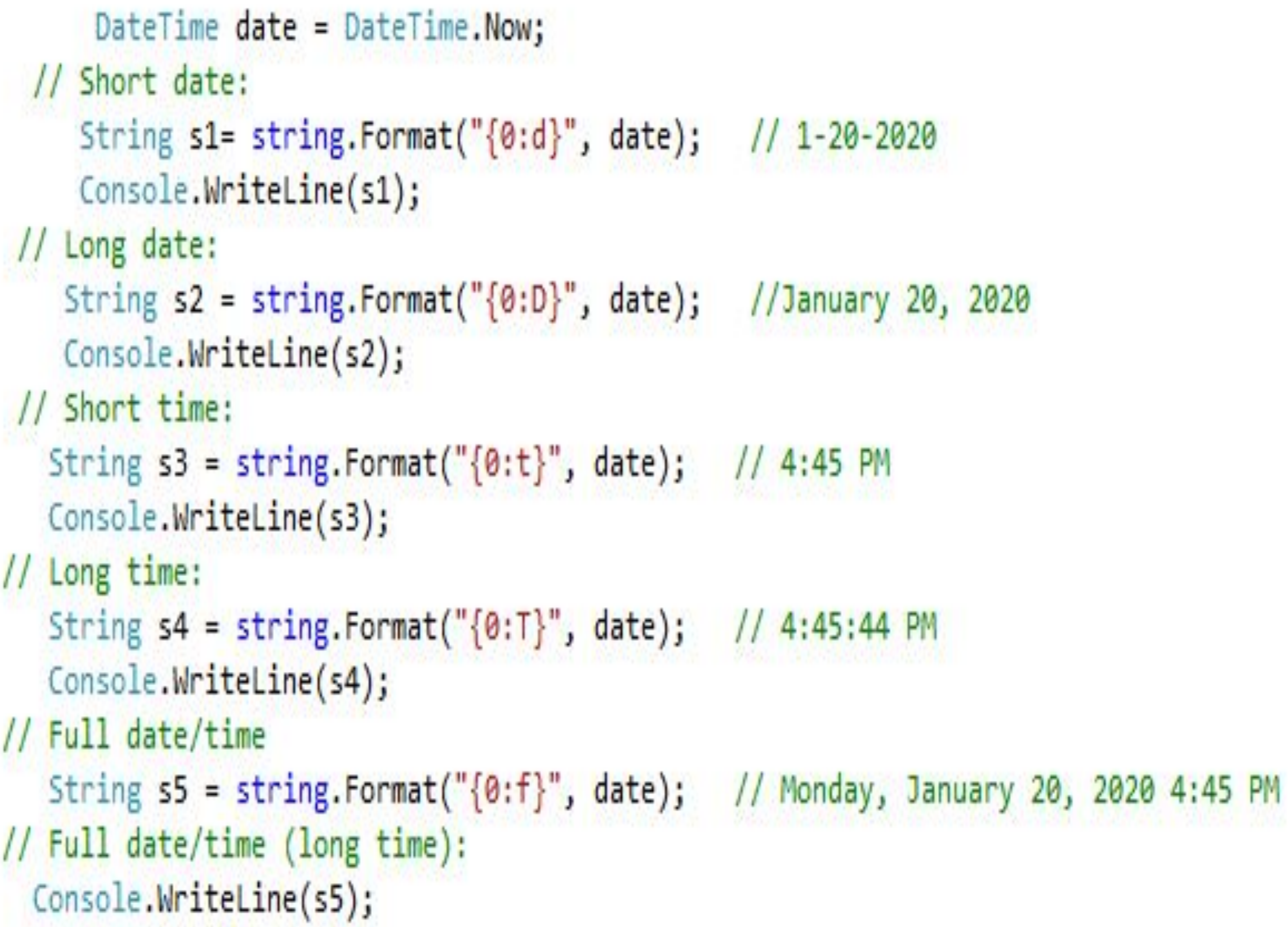
Memory allocation for String

Example



```
Temp::  HelloHow R U!!!!!!!!!
Concate :: HelloHow R U!!!!!!!!!
Join:Hello,How,R,U,!!!!!!
```

```
string a, b;
a = "Hello";
b = "How R U!!!!!!!!!";
string temp = a + b;  // string concat
string combine = string.Concat(a, b);
Console.WriteLine("Temp::  " + combine);
Console.WriteLine("Concate :: " + combine);

char[] ch = { 'H', 'E', 'L', 'L', 'O' };
string letters = new string(ch);

//returning string
string[] arr = { "Hello", "How", "R", "U", "!!!!!!" };
string message = string.Join(",", arr);
Console.WriteLine("Join:" + message);
```



```
        DateTime date = DateTime.Now;
    // Short date:
       String s1= string.Format("{0:d}", date);   // 1-20-2020
       Console.WriteLine(s1);
   // Long date:
      String s2 = string.Format("{0:D}", date);   //January 20, 2020
      Console.WriteLine(s2);
   // Short time:
     String s3 = string.Format("{0:t}", date);   // 4:45 PM
     Console.WriteLine(s3);
  // Long time:
     String s4 = string.Format("{0:T}", date);   // 4:45:44 PM
     Console.WriteLine(s4);
  // Full date/time
     String s5 = string.Format("{0:f}", date);   // Monday, January 20, 2020 4:45 PM
  // Full date/time (long time):
    Console.WriteLine(s5);
```

String

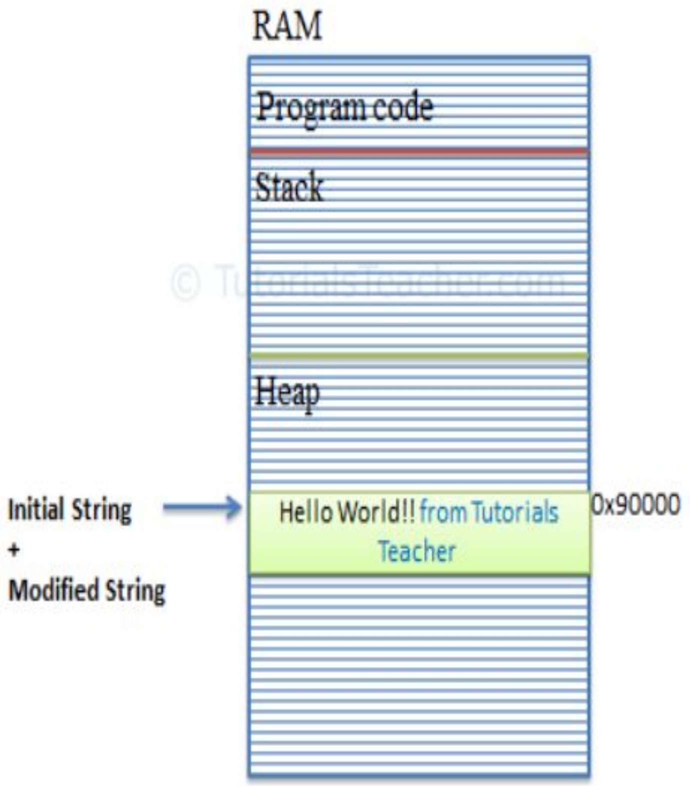
- string S=“Hello World”
- Now replace string S with “Welcome”
- In heap **Hello World** and new object of **Welcome** is created.

- To solve this problem C#, introduced **StringBuilder.**

StringBuilder

- StringBuilder is **mutable**, means if create string builder object then you can perform any operation like insert, replace or append **without creating new instance for every time.**

- It will update string at **one place** in memory doesn't create new space in memory.

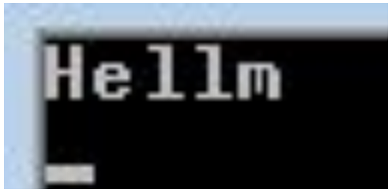


Memory allocation for StringBuilder

- StringBuilder sb = new StringBuilder()

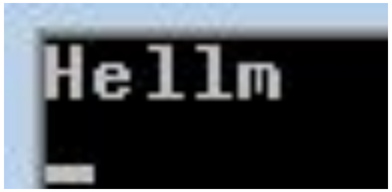
String and StringBuilder

```
 string a = "Hello";
a = a.Replace("o", "m");
Console.WriteLine(a);
```



```
Hellm
_
```

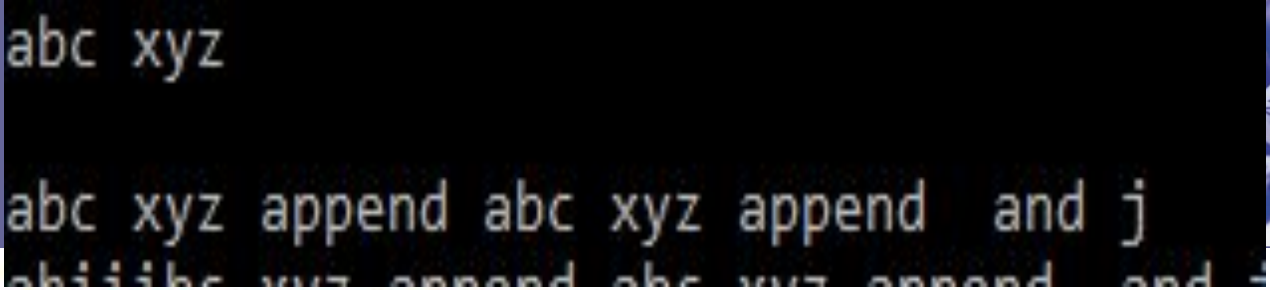
```
StringBuilder sb = new StringBuilder(“Hello”);
a = sb.Replace("o", "m");
Console.WriteLine(a);
```



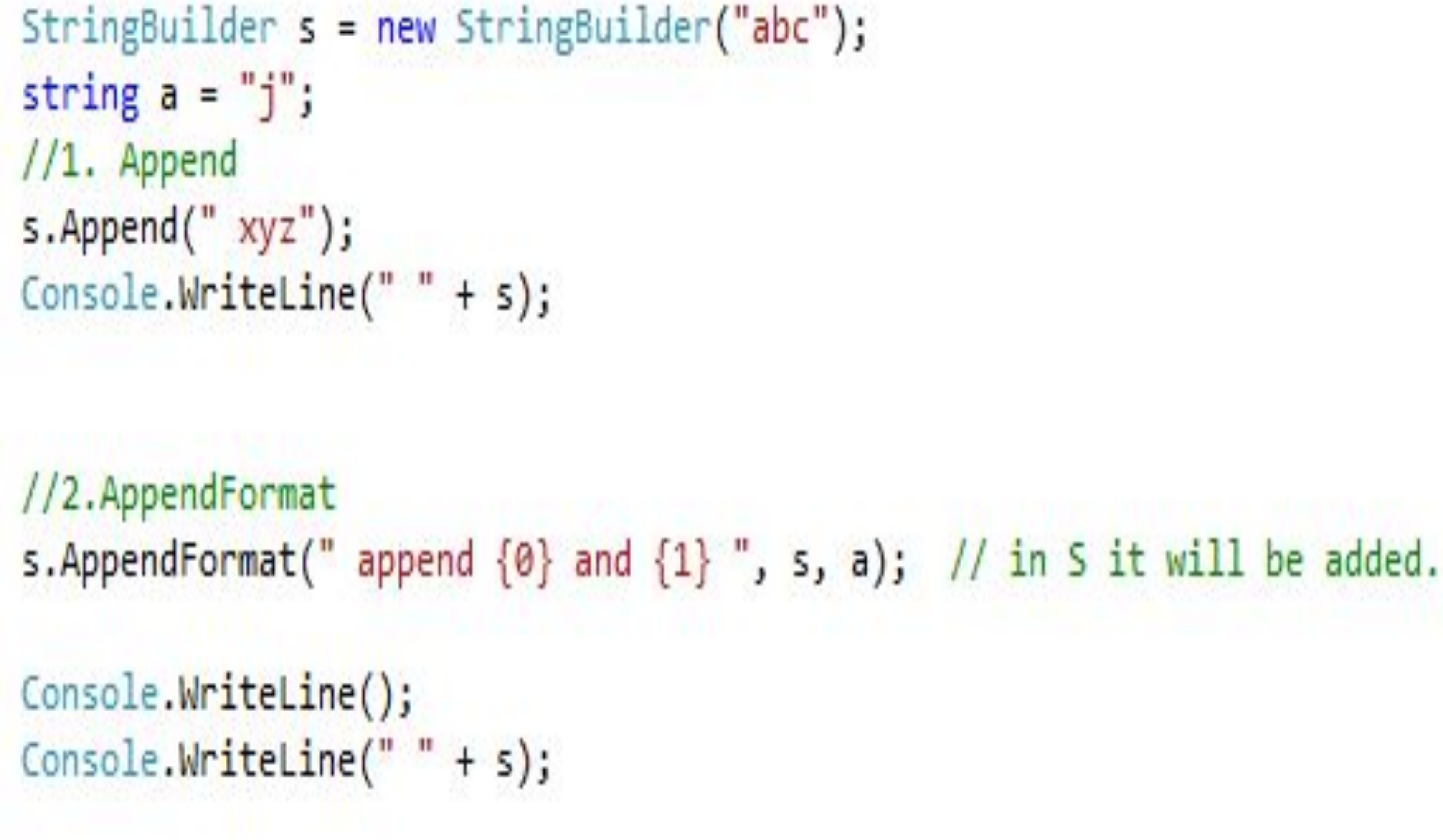
```
Hellm
_
```

StringBuilder Methods

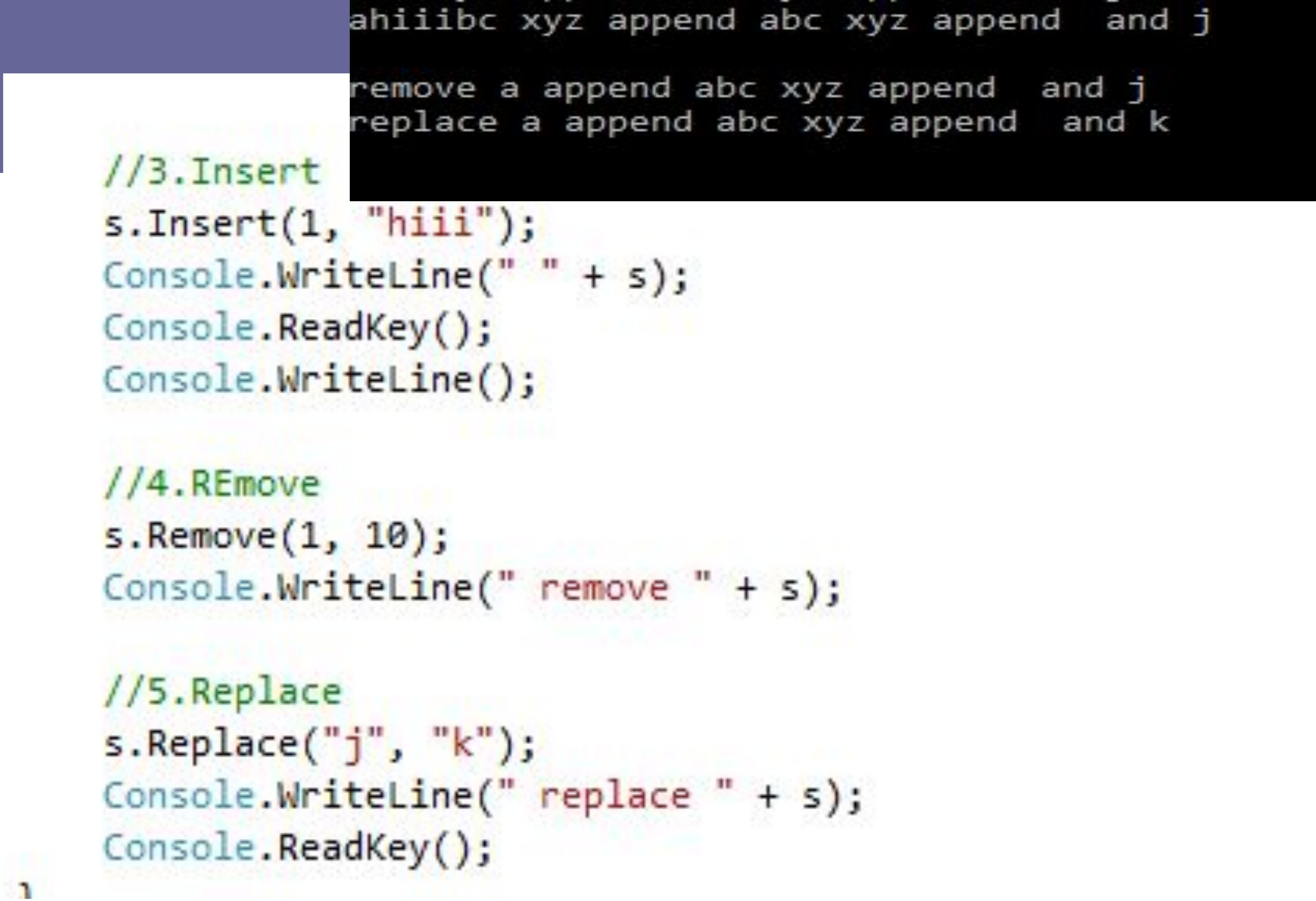
- sb.Append(value)
- sb.Insert(index,value)
- sb.Replce(old value, new value)
- sb.Remove(starting index, length)
- sb.Tostring()



```
abc xyz

abc xyz append abc xyz append  and j
abjjjabc xyz append abc xyz append  and j
```



```
StringBuilder s = new StringBuilder("abc");
string a = "j";
//1. Append
s.Append(" xyz");
Console.WriteLine(" " + s);


//2.AppendFormat
s.AppendFormat(" append {0} and {1} ", s, a);  // in S it will be added.

Console.WriteLine();
Console.WriteLine(" " + s);
```



```
ahiiibc xyz append abc xyz append  and j

remove a append abc xyz append  and j
replace a append abc xyz append  and k
```

```
    //3.Insert
    s.Insert(1, "hiii");
    Console.WriteLine(" " + s);
    Console.ReadKey();
    Console.WriteLine();

    //4.REmove
    s.Remove(1, 10);
    Console.WriteLine(" remove " + s);

    //5.Replace
    s.Replace("j", "k");
    Console.WriteLine(" replace " + s);
    Console.ReadKey();
}
```

StringBuilder Methods

StringBuilder as Indexer

Use: To get or set a character at specified index

Example:

```
StringBuilder sb = new StringBuilder("Hello
 World!!");
  for(int i=0; i< sb.Length; i++)
                 Console.Write(sb[i]);
```

Questions and Comments