



Tipos de datos en C#

Los tipos de datos en C# se clasifican en: **tipos valor** y **tipos referencia**. Una variable de tipo valor almacena directamente un valor (datos simples), mientras que una variable de un tipo referencia permite almacenar una referencia a un objeto.

Resumiendo podemos decir que los **datos por valor** son variables en las que accedemos a su valor de manera directa. Este tipo de variables no afectan sobre otras variables ya que cada variable accede a un único registro en memoria.

Para declarar un tipo de dato por valor seguimos la sintaxis:

Tipo_de_dato *nombre_identificador*;

Ejemplos

```
int i;  
float x;  
char a='b';  
string="Hola";
```

Los **datos por referencia** son aquellos en las que accedemos a su valor de manera indirecta, mediante su referencia a un registro de la memoria. En este caso si dos variables tienen la misma referencia estarán apuntando al mismo valor.

Para declarar las variables por referencia utilizamos la palabra reservada **new**:

Sintaxis:

Tipo_de_dato *nombre_identificador* = **new** **tipo_de_dato**([constructor]); //Creación de un objeto

Ejemplo: **alumno A1 = new alumno();**

```
Class Program()  
{  
    static void Main(string[] args)  
    {  
        //Creación de una referencia a un objeto a la clase Program  
        Program miObjetoProgram = new Program();  
  
        //Creación de una referencia a un objeto a la clase alumno  
        alumno miObjetoAlumno = new alumno();  
    }  
}  
  
Class alumno  
{  
    String nombre;  
    Int edad;  
}
```

A diferencia de los tipos de referencia, los tipos de valor no pueden incluir el valor **null**.



Conversiones implícitas y explícitas

C# permite convertir un valor de un tipo en un valor de otro tipo. Puede trabajar con conversiones implícitas y explícitas. Las conversiones implícitas son *conversiones directas, que se realizan siempre que mismos tipos puedan hacerlo*, siempre funcionan y nunca pierden información (por ejemplo, puede convertir un int en un long sin perder ningún dato porque un long es mayor que un int).

Las conversiones explícitas son *conversiones en la que debemos forzar la conversión, éstas pueden producir pérdida de datos (por ejemplo, convertir un long en un int puede producir pérdida de datos porque un long puede contener valores mayores que un int)*. Debe escribir un operador **cast** (que se representa por anteponer el tipo de variable al que se quiere convertir cerrado entre paréntesis) en el código para que se produzca una conversión explícita.

Ejemplo:

```
int x;  
double y = 345.35;
```

```
//Conversión implícita  
x= y;
```

```
//Conversión explícita  
y= (int) x;
```

Existen otros métodos que nos permiten hacer conversiones de una cadena a cualquier otro tipo de dato : Convert() y Parse

Alias c#	Método Convert	Método Parse
short	Convert.ToInt16	Int16.Parse()
int	Convert.ToInt32	Int32.Parse()
long	Convert.ToInt64	Int64.Parse()
string	Convert.ToString	No aplica
char	Convert.ToChar	Char.Parse()
float	Convert.ToSingle	Single.Parse()
double	Convert.ToDouble	Double.Parse()
decimal	Convert.ToDecimal	Decimal.Parse()

Ejemplo:

```
float MiFlotante;  
string cadFlotante;
```

```
cadFlotante="3.1416";  
MiFlotante= Convert.ToSingle(cadFlotante);  
MiFlotante=Single.Parse(cadFlotante);
```

Al final de la práctica se anexan las tablas de conversiones válidas entre los distintos tipos de datos.



Ámbito de una variable

El ámbito de una variable se refiere a la parte del programa donde es declarada; determina su accesibilidad desde otras partes del programa. Así como la duración que tiene la variable durante el tiempo que está en ejecución.

Ejemplo:

```
class Program
{
    static mi_variable;

    static void Main(string[] args)
    {
        int X, Y;
        Console.WriteLine("variable={0}",mi_variable)
    }
}
class Otra_clase
{
    public void metodo1()
    {
        int Z;
        Console.WriteLine("variable={0}",Program.mi_variable);
    }
}
```

Ambito de X y Y

Ambito de Z

Ambito de mi_variable

Visibilidad de una variable

Si una clase contiene elementos que son privados (**private**), éstos son sólo visibles para los otros elementos dentro de la misma clase, pero no para otros objetos de otras clases. Por el contrario, si la clase contiene elementos públicos (**public**), éstos pueden ser visibles desde instancias de otras clases. En el caso de las clases, al no explicitar nada, se asume que son public. Por el contrario, en el caso de los elementos de una clase, como atributos o métodos, al no explicitar nada, se asumen como private.

Tenemos adicionalmente otros tipos de acceso a las variables que nos van a permitir controlar el acceso a las mismas, estos son:

protected. Solo pueden acceder a esta variable aquellos objetos de su clase y los objetos de subclases.

internal. Se accede a ella desde cualquier clase contenida dentro del mismo espacio de nombre.

protected internal. Se accede a ellas desde la clase, subclases y las clases que se encuentren dentro del mismo espacio de nombre.



Duración de una variable

La duración de la variable la podemos modificar anteponiendo un **static**.

Por ejemplo una variable declarada dentro de un método tiene como ámbito ese bloque de código y su duración como el tiempo desde que se declara dentro del método hasta que salimos de él.

Todas las variables declaradas como **static** mantendrán invariablemente su espacio reservado en memoria durante toda la ejecución del programa.

La variable **static** existe durante toda la ejecución del programa, sólo existe una copia de la misma sin necesidad de crear un objeto, no afecta su visibilidad.

Ejemplo:

```
class Program
{
    static mi_variable;

    static void Main(string[] args)
    {
        int X, Y;
        Console.WriteLine("variable={0}",mi_variable);
    }
}

class Otra_clase
{
    public void metodo1()
    {
        int Z;
        Console.WriteLine("variable={0}",Program.mi_variable);
    }
}
```

Constantes

Las constantes son valores que se conocen en tiempo de compilación y no cambian. Las constantes se declaran como campo utilizando la palabra clave **const** antes del tipo del campo. Las constantes se deben inicializar tal como se declaran. Este tipo de variables nunca cambian de valor y se consideran implícitamente **static**. Por ejemplo:

```
class calendario
{
    public const int meses = 12;
}

class hora
{
    public const int minutos = 60;
}

class circulo
{
    public float const PI=3.1416F;
}
```



Anexo 1

Tabla de conversiones numéricas explícitas

De	Para
sbyte	byte, ushort, uint, ulong o char
byte	Sbyte o char
short	sbyte, byte, ushort, uint, ulong o char
ushort	sbyte, byte, short o char
int	sbyte, byte, short, ushort, uint, ulong o char
uint	sbyte, byte, short, ushort, int o char
long	sbyte, byte, short, ushort, int, uint, ulong o char
ulong	sbyte, byte, short, ushort, int, uint, long o char
char	sbyte, byte o short
float	sbyte, byte, short, ushort, int, uint, long, ulong, char o decimal
double	sbyte, byte, short, ushort, int, uint, long, ulong, char, float o decimal
decimal	sbyte, byte, short, ushort, int, uint, long, ulong, char, float o double

Tabla de conversiones numéricas implícitas

De	A
sbyte	short, int, long, float, double o decimal
byte	short, ushort, int, uint, long, ulong, float, double o decimal
short	int, long, float, double o decimal
ushort	int, uint, long, ulong, float, double o decimal
int	long, float, double o decimal
uint	long, ulong, float, double o decimal
long	float, double o decimal
char	ushort, int, uint, long, ulong, float, double o decimal
float	double
ulong	float, double o decimal



Anexo 2

Ejemplo variables por valor y referencia

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

namespace referencia_valor
{
    class Program
    {
        int x;
        static void Main(string[] args)
        {
            int var_valor, nuevo_valor;
            Program var_referencia1 = new Program();
            Program var_referencia2;
            var_valor = 10;
            nuevo_valor = var_valor;
            Console.WriteLine("\n***** POR VALOR *****\n");
            Console.WriteLine("Variable por valor {0}, nuevo valor {1}", var_valor, nuevo_valor);
            nuevo_valor = 5;
            Console.WriteLine("Variable por valor {0}, nuevo valor {1}", var_valor, nuevo_valor);

            /*****
            Console.WriteLine("\n***** POR REFERENCIA *****\n");
            var_referencia1.x = 7;
            var_referencia2 = var_referencia1;
            Console.WriteLine("Variable por referencia1 {0}, variable por referencia2 {1}",
            var_referencia1.x, var_referencia2.x);
            var_referencia2.x = 100;
            Console.WriteLine("Variable por referencia1 {0}, variable por referencia2 {1}",
            var_referencia1.x, var_referencia2.x);
            }
        }
    }
}
```



Ejemplo visibilidad de variables

```
using System;
using System.Collections.Generic;
using System.Text;
/* Programa ejemplo para visibilidad de variables */
namespace visibilidad_variables
{
    class Program
    {
        public int Entero1 = 5;
        private int Entero2 = 10;

        static void Main(string[] args)
        {
            int Entero3 = 15;
            Program MiObjeto = new Program();

            Console.WriteLine("Imprimiendo desde el método Main()");
            Console.WriteLine("{0}, {1}, {2}", MiObjeto.Entero1, MiObjeto.Entero2, Entero3);

            MiObjeto.Auxiliar();

            simple MiSimple = new simple();
            MiSimple.Enumerar();

        }
        public void Auxiliar()
        {
            Console.WriteLine("Imprimiendo desde el método Auxiliar() de la clase Program");
            Console.WriteLine("{0}, {1}", Entero1, Entero2);
        }
    }

    class simple
    {
        private int Entero4 = 20;

        public void Enumerar()
        {
            Console.WriteLine("Imprimiendo desde el método Enumerar() de la clase simple");
            Console.WriteLine("{0}", Entero4);
        }
    }
}
```



Ejemplo de vida de variables

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

namespace vida_variables
{
    class Program
    {
        byte Valor_simple;
        static byte Valor_estatico;
        static void Main(string[] args)
        {
            byte Contador;
            for (Contador = 1; Contador <= 10; Contador++)
                MuestraVariables();
            Console.WriteLine("Oprima una tecla para terminar...");
            Console.ReadKey();
        }

        static void MuestraVariables()
        {
            Program MiObjeto = new Program();
            MiObjeto.Valor_simple++;
            Program.Valor_estatico++;
            Console.WriteLine("Valor simple = {0}, Valor estatico = {1}", MiObjeto.Valor_simple,
            Program.Valor_estatico);
        }
    }
}
```



Ejemplo de conversiones

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

namespace conversiones
{
    class Program
    {
        static void Main(string[] args)
        {
            float miFlotante = 3.18F, Flotante;
            char miCaracter = 'a', Caracter;
            int miEntero = 6, Entero;
            decimal miDecimal = 8.99999M, Decimal;
            double miDoble = 35.99, Doble;
            long miLong = 3545678788, Largo;
            string miCadena1 = "45.678", miCadena2="9";
            Console.WriteLine("Mis variables iniciales son: ");
            Console.WriteLine("Entero: {0}, Entero largo = {1}, Flotante: {2}, Doble: {3}, Decimal: {4},
Caracter: {5}, Cadena1: {6}, Cadena2: {7}",
miEntero,miLong,miFlotante,miDoble,miDecimal,miCaracter,miCadena1,miCadena2);
            Flotante = miEntero;
            Entero = miCaracter;
            Largo = miEntero;
            Doble = miFlotante;
            Console.WriteLine("Conversiones implícitas:");
            Console.WriteLine("Flotante: {0}, Entero = {1}, Largo: {2}, Doble: {3}", Flotante, Entero,
Largo, Doble);
            Entero = (int)miFlotante;
            Flotante = (float)miDecimal;
            Decimal = (decimal)miLong;
            Console.WriteLine("Conversiones explícitas:");
            Console.WriteLine("Flotante: {0}, Entero = {1}, Decimal: {2}", Flotante, Entero, Decimal);
            Flotante = Convert.ToSingle(miEntero);
            Entero = Convert.ToInt16(miCaracter);
            Caracter = Convert.ToChar(miEntero);
            Console.WriteLine("Conversiones con la funcion Convert:");
            Console.WriteLine("Flotante: {0}, Entero = {1}, Caracter: {2}", Flotante, Entero, Caracter);
            Flotante = float.Parse(miCadena1);
            Entero = int.Parse(miCadena2);
            Caracter = char.Parse(miCadena2);
            Console.WriteLine("Conversiones con el metodo Parse:");
            Console.WriteLine("Flotante: {0}, Entero = {1}, Caracter = {2}", Flotante, Entero, Caracter);
        }
    }
}
```