Programación C#

Ingeniería en Computación

Carlos Castañeda Ramírez

Sintaxis C#

- Contenido
  - Introducción
  - Expresiones
  - Operadores
  - Sentencias
  - Tipos
  - Literales
  - Cadenas (Strings)
  - Formateo de Cadenas
  - Conversión entre tipos

Introducción

- C# es un lenguaje *Typesafe*
  - El código solo puede acceder a la memoria que esta autorizado para acceder
- C# es un lenguaje fuertemente tipado
  - Cada valor o variable debe ser declarado de un tipo específico
- Su sintaxis es muy similar a C++ y Java

Introducción

- La naturaleza orientada a objetos de C# requiere que los programas sean definidos mediante clases.
- Para que el compilador interprete el código, ciertas palabras deben tener un significado especial (reserved words I keywords).

Expresiones

- Están formadas por operandos y operadores.
- Los operadores de una expresión indican la operación a realizar con los operandos. (+, - *, I, etc.)
- Cuando hay múltiples operadores, la precedencia controla el orden de evaluación de los operadores.
- Evalúan valores y convergen hacia un valor.
  - int a,b,c,d,e;
  - a = b = c = d = e = 20;
  - mySecondVariable = myVariable = 57;

Operadores y precedencia (alta a baja)

Category	Operators
Primary	x.y f(x) a[x] x++ x-- new typeof default checked unchecked
Unary	+ - ! ~ ++x --x (T)x
Multiplicative	* / %
Additive	+ -
Shift	<< >>
Relational and type testing	< > <= >= is as
Equality	== !=
Logical AND	&
Logical XOR	^
Logical OR	|
Conditional AND	&&
Conditional OR	||
Null coalescing	??
Conditional	?:
Assignment and lambda expression	= *= /= %= += -= <<= >>= &= ^= |= =>

Sentencias

- Una sentencia completa se llama "statement".
- Cada statement termina con punto y coma (;).
- Es la unidad básica de ejecuci6n de C#.
- Un programa C# esta formado por una secuencia de instrucciones (statements).
- Pueden ser:
  - Declaraciones de variables
  - Expresiones
  - Selección (if, case)
  - Iteración
  - Etc...

```
int myVariable;                  // a statement
myVariable = 23;                 // another statement
int anotherVariable = myVariable; // yet another statement
Console.WriteLine("Hello World");
```

- Pueden ser agrupadas mediante bloques { ... }.

Ejemplos de sentencias

Statement	Example

Local variable declaration	`static void () {`
	`  int a;`
	`  int b = 2, c = 3;`
	`  a = 1;`
	`  Console.WriteLine(a + b + c);`
	`}`

Local constant declaration	`static void () {`
	`  const float pi = 3.1415927f;`
	`  const int r = 25;`
	`  Console.WriteLine(pi * r * r);`
	`}`

Expression statement	`static void () {`
	`  int i;`
	`  i = 123;  // Expression statement`
	`  Console.WriteLine(i); // Expression statement`
	`  i++;  // Expression statement`
	`  Console.WriteLine(i); // Expression statement`
	`}`

if statement	`static void (string[] args) {`
	`  if (args.Length == 0) {`
	`  Console.WriteLine("No arguments");`
	`  }`
	`  else {`
	`  Console.WriteLine("One or more arguments");`
	`  }`
	`}`

Tipos

Category	Bits	Type	Range/Precision
Signed integral	8	`sbyte`	–128...127
	16	`short`	–32,768...32,767
	32	`int`	–2,147,483,648...2,147,483,647
	64	`long`	–9,223,372,036,854,775,808...9,223,372,036,854,775,807
Unsigned integral	8	`byte`	0...255
	16	`ushort`	0...65,535
	32	`uint`	0...4,294,967,295
	64	`ulong`	0...18,446,744,073,709,551,615
Floating point	32	`float`	1.5×10^{-45} to 3.4×10^{38}, 7-digit precision
	64	`double`	5.0×10^{-324} to 1.7×10^{308}, 15-digit precision
Decimal	128	`decimal`	1.0×10^{-28} to 7.9×10^{28}, 28-digit precision

Literales

- Son números o cadenas escritas en el código fuente que representan un valor o conjunto de valores.
- Al ser escritos en el código fuente, sus valores se conocen en tiempo de ejecución.
- El tipo "bool" tiene 2 literales: true y false.
- Para variables del tipo Referencia, el literal null significa que no referencia ningún objeto en memoria.

Ejemplos de literales

```
static void Main()          Literals
{                             ↓
   Console.WriteLine("{0}", 1024);          // int literal
   Console.WriteLine("{0}", 3.1416);        // double literal
   Console.WriteLine("{0}", 3.1416F);       // float literal
   Console.WriteLine("{0}", true);          // boolean literal
   Console.WriteLine("{0}", 'x');           // character literal
   Console.WriteLine("{0}", "Hi there");    // string literal
}
```

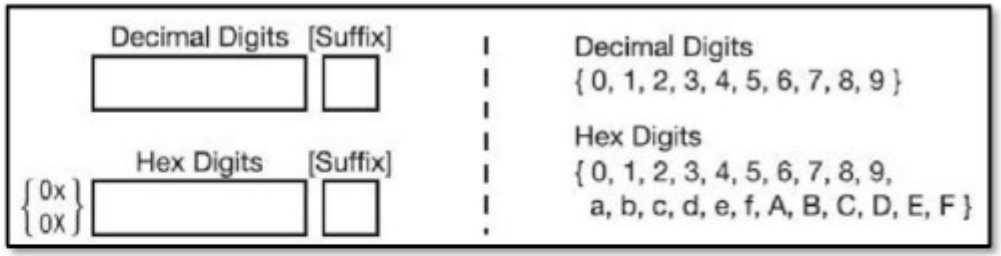
The output of this code is the following:

```
1024
3.1416
3.1416
True
x
Hi there
```

Literales de números enteros

- Son los mas comunes.
- No tienen punto decimal.
- Pueden tener un sufijo para especificar el tipo de entero.
- Pueden ser escritos en forma decimal o hexadecimal.

```
236             // int
236L            // long
236U            // unsigned
236UL           // unsigned long
```

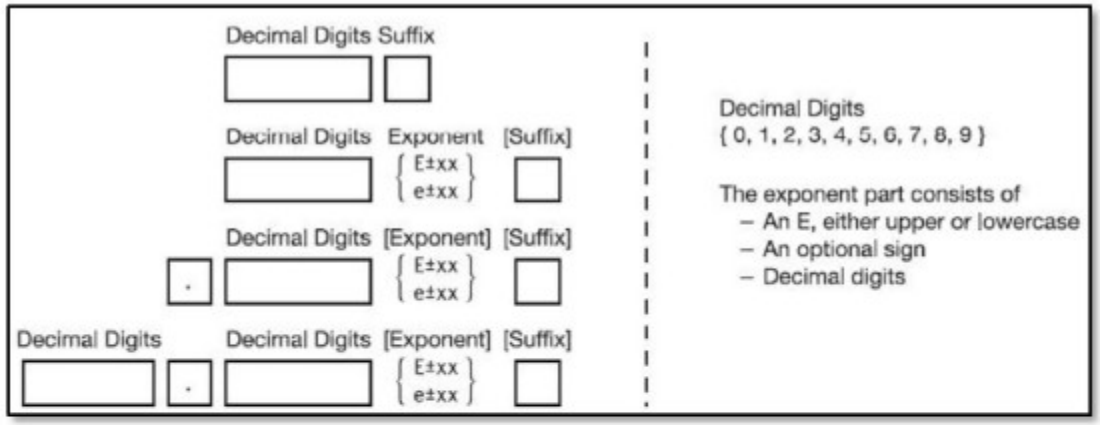


Suffix	Integer Type	Notes
None	int, uint, long, ulong	
U, u	uint, ulong	
L, l	long, ulong	Using the lowercase letter *l* is not recommended, as it is easily mistaken for the digit *1*.
ul, uL, Ul, UL lu, Lu, lU, LU	ulong	Using the lowercase letter *l* is not recommended, as it is easily mistaken for the digit *1*.

Literales de números reales

- Consisten en dígitos decimales, punto decimal (opcional), parte exponencial (opcional) y sufijo (opcional).

```
float  f1 = 236F;
double d1 = 236.714;
double d2 = .35192;
double d3 = 6.338e-26;
```



Suffix	Real Type
None	double
F, f	float
D, d	double
M, m	decimal

Literales de caracteres

- El tipo de un carácter es char.
- Están delimitados por comillas simples (')
- Una secuencia de escape simple esta formada por un backslash (\) seguido de un carácter.
- Una secuencia de escape Hexadecimal esta formada por un backslash+x (\x) seguido de hasta 4 dígitos hexadecimales.
- Una secuencia de escape Unicode esta formada por un backslash+u (\u) seguido de hasta 4 dígitos hexadecimales.

```
char c1 = 'd';                  // Single character
char c2 = '\n';                 // Simple escape sequence
char c3 = '\x0061';             // Hex escape sequence
char c4 = '\u005a';             // Unicode escape sequence
```

Name	Escape Sequence	Hex Encoding
Null	\0	0x0000
Alert	\a	0x0007
Backspace	\b	0x0008
Horizontal tab	\t	0x0009
New line	\n	0x000A
Vertical tab	\b	0x000B
Form feed	\f	0x000C
Carriage return	\r	0x000D
Double quote	\"	0x0022
Single quote	\'	0x0027
Backslash	\\	0x005C

Literales de cadenas

- El tipo de un carácter es string.
- Están delimitados par comillas dobles (")
- Hay dos (2) tipos de literales de cadenas:
- Regulares:
  - Caracteres
  - Secuencias de escape simples
  - Secuencias de escape Unicode y Hexadecimales

```
string st1 = "Hi there!";
string st2 = "Val1\t5, Val2\t10";
string st3 = "Add\x000ASome\u0007Interest";
```

- Verbatim:
  - No evalúan las secuencias de escape.
  - Se escriben como las Regulares pera precedidas par el carácter arroba (@).
  - Todo dentro de las comillas simples se imprime tal cual.
  - La excepción es doble comilla-doble que se interpretan como simple comilla doble.

```
string rst1 = "Hi there!";
string vst1 = @"Hi there!";

string rst2 = "It started, \"Four score and seven...\"";
string vst2 = @"It started, ""Four score and seven...""";

string rst3 = "Value 1 \t 5, Val2 \t 10";    // Interprets tab esc sequence
string vst3 = @"Value 1 \t 5, Val2 \t 10";   // Does not interpret tab

string rst4 = "C:\\Program Files\\Microsoft\\";
string vst4 = @"C:\Program Files\Microsoft\";

string rst5 = " Print \x000A Multiple \u000A Lines";
string vst5 = @" Print
 Multiple
 Lines";
```

```
Hi there!
Hi there!

It started, "Four score and seven..."
It started, "Four score and seven..."

Value 1          5, Val2         10
Value 1 \t 5, Val2 \t 10

C:\Program Files\Microsoft\
C:\Program Files\Microsoft\

 Print
 Multiple
 Lines

 Print
 Multiple
 Lines
```

String (Cadenas)

- El tipo "string" es un alias de System.String.
- Es un tipo "sealed", por lo cual no puede heredarse.
- Representa cadenas de caracteres Unicode.
- Es un tipo de Referencia.
- Pueden crearse desde arrays de caracteres.
- Son inmutables: su secuencia de caracteres no puede ser modificada.
- Cada vez que se modifica un string, se crea una nueva instancia en memoria. El GC se encarga de las anteriores.
- Utilizar StringBuilder para concatenaciones y manejo optimo de cadenas.
- Se puede realizar búsquedas y agrupamiento con expresiones regulares sobre cadenas.
- Utilizar el método Split para partir cadenas.

```
using System;
using System.Text;

public class EntryPoint
{
    static void Main() {
        StringBuilder sb = new StringBuilder();

        sb.Append("StringBuilder ").Append("is ")
            .Append("very... ");

        string built1 = sb.ToString();

        sb.Append("cool");

        string built2 = sb.ToString();

        Console.WriteLine( built1 );
        Console.WriteLine( built2 );
    }
}
```

```
String[] resultArray = s1.Split( delimiters );
foreach ( String subString in resultArray )
{
    Console.WriteLine(ctr++ + ":" + subString);
}
```

Formateo de cadenas (formating)

- Dar formato a cadenas es una tarea común.
- Necesario para mostrar fechas, números, valores decimales, etc.
- C# proporciona un conjunto amplio de opciones de formateo de cadenas.
- El formateo aplicado depende de la localización.
- Se realiza invocando el método Format de la clase string.

```
string formatted = string.Format("The value is {0}", value);
```

```
DateTime.ToString("format specifiers");
```

```
int i = 105263;
textBox1.Text = i.ToString("E4");   // 1.0526E+005
```

Ejemplos de formateo de fechas

```
DateTime dt = DateTime.Now;

Console.WriteLine(string.Format("Default format: {0}", dt.ToString()));
Console.WriteLine(dt.ToString("dddd dd MMMM, yyyy g"));
Console.WriteLine(string.Format("Custom Format 1: {0:MM/dd/yy
hh:mm:sstt}", dt));
Console.WriteLine(string.Format("Custom Format 2: {0:hh:mm:sstt G\\MT
zz}", dt));
```

```
Default format: 9/24/2005 12:59:49 PM
Saturday 24 September, 2005 A.D.
Custom Format 1: 09/24/05 12:59:49PM
Custom Format 2: 12:59:49PM GMT -06
```

Specifier	Description
d	Displays the current day of the month.
dd	Displays the current day of the month, where values < 10 have a leading zero.
ddd	Displays the three-letter abbreviation of the name of the day of the week.
dddd(+)	Displays the full name of the day of the week represented by the given DateTime value.
f(+)	Displays the x most significant digits of the seconds value. The more f's in the format specifier, the more significant digits. This is total seconds, not the number of seconds passed since the last minute.
F(+)	Same as f(+), except trailing zeros are not displayed.
g	Displays the era for a given DateTime (for example, "A.D.")
h	Displays the hour, in range 112.
hh	Displays the hour, in range 112, where values < 10 have a leading zero.
H	Displays the hour in range 023.
HH	Displays the hour in range 023, where values < 10 have a leading zero.
m	Displays the minute, range 059.
mm	Displays the minute, range 059, where values < 10 have a leading zero.
M	Displays the month as a value ranging from 112.
MM	Displays the month as a value ranging from 112 where values < 10 have a leading zero.
MMM	Displays the three-character abbreviated name of the month.
MMMM	Displays the full name of the month.
s	Displays the number of seconds in range 059.
ss(+)	Displays the number of seconds in range 059, where values < 10 have a leading 0.
t	Displays the first character of the AM/PM indicator for the given time.
tt(+)	Displays the full AM/PM indicator for the given time.
y/yy/yyyy	Displays the year for the given time.
z/zz/zzz(+)	Displays the timezone offset for the given time.

Ejemplos de formateo de números

Specifier	String result	Datatype
C[n]	$XX.XX.XX ($XX.XXX.XX)	Currency
D[n]	[-]XXXXXXX	Decimal
E[n] or e[n]	[-]X.XXXXXXE+xxx [-]X.XXXXXXe+xxx [-]X.XXXXXXE-xxx [-]X.XXXXXXe-xxx	Exponent
F[n]	[-]XXXXXXX.XX	Fixed point
G[n]	General or scientific	General
N[n]	[-]XX,XXX.XX	Number
X[n] or x[n]	Hex representation	Hex

```
int i = 654321;
Console.WriteLine("{0:C}",  i); // $654,321.00
Console.WriteLine("{0:C6}", i); // $654,321.000000
Console.WriteLine("{0:D}",  i); // 654321
Console.WriteLine("{0:D8}", i); // 00654321
Console.WriteLine("{0:E}",  i); // 6.543210E+005
Console.WriteLine("{0:F}",  i); // 654321.00
Console.WriteLine("{0:G}",  i); // 654321
Console.WriteLine("{0:N}",  i); // 654,321.00
Console.WriteLine("{0:X}",  i); // 9FBF1
Console.WriteLine("{0:x}",  i); // 9fbf1
```

Format String	Data type	Value	Output
C	Double	12345.6789	$12,345.68
D	Int32	12345	12345
D8	Int32	12345	00012345
E	Double	12345.6789	1.234568E+004
E10	Double	12345.6789	1.2345678900E+004
e	Double	12345.6789	1.2346e+004

Format String	Data type	Value	Output
F	Double	12345.6789	12345.68
F0	Double	12345.6789	123456
F6	Double	12345.6789	12345.678900
G	Double	12345.6789	12345.6789
G7	Double	12345.6789	12345.68
G	Double	0.0000023	2.3E-6

Format String	Data type	Value	Output
G2	Double	1234	1.2E3
G	Double	Math.PI	3.14159265358979
N	Double	12345.6789	12,345.68
N4	Double	123456789	123,456,789.0000
P	Double	.126	12.60 %
r	Double	Math.PI	3.141592653589793

Formateo avanzado de números

```
Console.WriteLine("C: {0}", 39.22M.ToString("C"));
Console.WriteLine("D: {0}", 982L.ToString("D"));
Console.WriteLine("E: {0}", 3399283712.382387D.ToString("E"));
Console.WriteLine("F: {0}", .993F.ToString("F"));
Console.WriteLine("G: {0}", 32.559D.ToString("G"));
Console.WriteLine("N: {0}", 93823713.ToString("N"));
Console.WriteLine("P: {0}", .59837.ToString("P"));
Console.WriteLine("R: {0}", 99.33234D.ToString("R"));
Console.WriteLine("X: {0}", 369329.ToString("X"));
```

La ejecución del código anterior, utilizando la cultura us-EN mostraría:

```
C: $39.22
D: 982
E: 3.399284E+009
F: 0.99
G: 32.559
N: 93,823,713.00
P: 59.84 %
R: 99.33234 X: 5A2B1
```

Format string	Data type	Value	Output
`#####`	Double	123	123
`00000`	Double	123	00123
`(###) ### - ####`	Double	1234567890	(123) 456 – 7890
`#.##`	Double	1.2	1.2
`0.00`	Double	1.2	1.20
`00.00`	Double	1.2	01.20
`#,#`	Double	1234567890	1,234,567,890
`#,,`	Double	1234567890	1235
`#,,,`	Double	1234567890	1
`#,##0,,`	Double	1234567890	1,235
`#0.##%`	Double	0.086	8.6%
`0.###E+0`	Double	86000	8.6E+4
`0.###E+000`	Double	86000	8.6E+004
`0.###E-000`	Double	86000	8.6E004
`[##-##-##]`	Double	123456	[12-34-56]
`##;(##)`	Double	1234	1234
`##;(##)`	Double	-1234	(1234)

Conversiones entre tipos

- Permite que un tipo sea tratado como otro tipo de dato.
- Es importante que los tipos soporten la conversión cuando sea necesario.
- No existe un casting valido entre tipos numéricos y
- booleanos.
- C# brinda flexibilidad para realizar las conversiones entre tipos.

Tipos de Conversión

- Explicit Cast
- Implicit Cast
- Without Casting
  - Utilizando el metodo Parse()
  - Utilizando System.Convert()
  - Utilizando ToString()
  - Utilizando tryParse() (nuevo en NET 2 0)

```
int intNumber = 31416;
long longNumber = (long) intNumber;
```

```
int intNumber = 31416;
long longNumber = intNumber;
```

```
                    -31";
float kgElectronMass = float.Parse(text);
```

```
String middleCText = "278.4375";
double middleC = System.Convert.ToDouble(middleCText);
bool boolean = System.Convert.ToBoolean(middleC);
```

```
bool boolean = true;
string text = boolean.ToString();
System.Console.WriteLine(text);     // Display "True"
```

```
double number;
string input;
                                    a number: ");
input = System.Console.ReadLine();
if (double.TryParse(input, out number)) {
    // Converted correctly, now use number // ... }
 else {
   System.Console.WriteLine( "The text entered was not a valid
number.");
}
```