**Estructura básica de un programa en C#**

1. **Estructura de un programa**

Cualquier código C# se escribe usando un conjunto de sentencias. Una sentencia es un enunciado que le indica a la aplicación que realice alguna acción. Hay dos tipos de sentencias básicas en C#: las simples y las compuestas.

Una sentencia simple es aquella que termina con punto y coma (;) y usualmente se utilizan para:

* Declarar variables (sentencias de declaración)
* Asignar valores a variables (sentencias de asignación)
* Invocar a un método
* Ramificar código para controlar el flujo del programa

**Notas:**

* Una sentencia simple no necesariamente se escribe en una sola línea.
* En C#, un punto y coma es considerado una sentencia.

Una sentencia compuesta es aquella que contiene una o más sentencias simples englobadas en llaves ({ }). Las sentencias compuestas se emplean típicamente en estructuras de control (de decisión y de repetición) Con excepción de unas cuantas, las sentencias compuestas no terminan con punto y coma.

1. **Control de flujo**

La ramificación de código se usa para mover el flujo del programa en una dirección distinta y luego regresar al punto en el que se quedó, o también para repetir un conjunto de líneas de código para completar una o varias tareas una y otra vez.

**2.1. Instrucciones condicionales**

Las instrucciones condicionales de C# son aquellas que evalúan una condición y luego ejecutan una acción, no realizan ninguna o eligen entre diferentes acciones disponibles para ejecutar. C# posee los siguientes elementos para evaluar condiciones:

* Operadores relacionales
* Expresiones booleanas
* Operadores lógicos
* Un operador condicional (operador ternario)

Las condiciones nos permiten comparar valores, los cuales comúnmente se almacenan en variables, constantes o literales.

Una variable es un espacio de memoria con nombre que se usa para almacenar un valor para su uso posterior. Se llama variable porque podemos cambiar su valor cuando lo necesitemos.

Una constante es similar a una variable, pues también es un espacio de memoria con nombre que almacena un valor, pero dicho valor no puede ser cambiado a voluntad.

Un literal es un valor que se presenta tal como es. Los literales sólo se pueden asignar a variables o a constantes.

Para lograr las comparaciones entre valores se necesitan operadores relacionales o booleanos.

Los operadores relacionales son >, <, >=, <=, ==, !=

Los operadores booleanos más comunes son && (*and*), || (*or*) y ! (*not*)

Una expresión es una sentencia cuya ejecución produce un valor.

Los valores *true* y *false* son de tipo booleano. Para usarlos en un programa C#, hay que declarar una variable de tipo *bool*.

El operador ternario regresa uno de los dos valores incluidos en el operador en base a la evaluación de una expresión booleana. Su sintaxis es: *condición ? valor verdadero : valor falso*

Ejemplo:

Random rnd = new Random();

int num = 0;

num = rnd.Next(100); //genera un número aleatorio entre 1 y 100 y lo asigna a num

string tipo = num % 2 == 0 ? “par” : “impar”

**2.2. Expresiones booleanas**

En términos simples, una expresión booleana es aquélla que resulta en el retorno de un valor de tipo *bool*.

En C#, las expresiones booleanas se forman mediante comparaciones. Estas comparaciones se hacen empleando los operadores relacionales y booleanos.

**2.3. Realizando decisiones en el código**

Un programa puede ejecutar una serie de sentencias sin necesidad de tomar ninguna decisión. Sin embargo, en algún punto deberá evaluarse alguna condición para poder tomar un determinado curso de acción en base al resultado de dicha evaluación. Esto puede ser provocado, por ejemplo: por la introducción de algún valor por parte del usuario, por el hecho de revisar si un disco está presente antes de leer o escribir archivos, por revisar que exista una conexión de red antes de enviar peticiones a un servidor, etc.

**2.3.1. Sentencias if**

**a) Sentencia if simple**

Sintaxis:

if (condición)

{

sentencia(s);

}

sentencias restantes;

Es recomandable que las sentencias estén rodeadas de llaves, aunque sólo se trate de una.

Pueden anidarse las sentencias if para evaluar más de una condición

if (condición1)

{

if (condición2)

{

sentencia(s);

}

}

sentencias restantes;

También, puede usarse más de una condición

if (condición1 && condición2)

{

sentencia(s);

}

sentencias restantes;

**b) Sentencia if-else**

Se usa cuando se desea ejecutar un bloque de código si la condición es verdadera y un bloque distinto si la condición es falsa.

if (condición1)

{

sentencia(s)\_1;

}

else

{

sentencia(s)\_2;

}

sentencias restantes;

Las sentencias if-else también pueden anidarse

if (condición1)

{

sentencia(s)\_1;

}

else if (condición2)

{

sentencia(s)\_2;

}

else if (condición3)

{

sentencia(s)\_3;

}

(...)

else

{

sentencia(s)\_n;

}

**2.3.2. Sentencia switch**

Las sentencias if anidadas son útiles siempre y cuando su número sea limitado. Cuando se rebasa cierto número, es más útil la sentencia switch. Sintaxis:

switch (condición)

{

case 1:

sentencia(s)\_1;

break;

case 2:

sentencia(s)\_2;

break;

case 3:

sentencia(s)\_3;

break;

...

case n:

sentencia(s)\_n;

break;

default:

sentencia(s)\_default;

break;

}

El valor de comparación en la condición puede ser int, string, float e incluso una enumeración.

**2.4. Usando ciclos**

El uso de ciclos permite que una aplicación repita una serie de instrucciones a fin de completar una tarea.

C# proporciona 4 estructuras de repetición:

* Sentencia *for*
* Sentencia *foreach*
* Sentencia *while*
* Sentencia *do-while*

**2.4.1. Sentencia *for***

Esta sentencia permite repetir un conjunto de instrucciones hasta que se cumple una condición determinada. Se compone de tres elementos: inicialización, condición e incremento (o iterador).

for (inicialización; condición; incremento) {

sentencia(s)\_a\_repetir;

}

En la inicialización se puede usar cualquier nombre de variable, pero siempre teniendo en cuenta que:

* No se pueden usar palabras reservadas
* La variable declarada no debe tener el mismo nombre que alguna otra variable usada para un propósito distinto dentro del ciclo
* La variable usada en la inicialización puede usarse dentro del ciclo
* Esta variable no puede usarse fuera del ciclo dado el alcance de la variable

La inicialización ocurre sólo una vez, la primera vez que se ejecuta el ciclo. La condición es revisada en cada iteración del ciclo, incluyendo la primera. Ocurre entre la inicialización y el incremento. Luego, se ejecutan las sentencias que se van a repetir y entonces se actúa sobre la variable de control.

**2.4.2. Sentencia *foreach***

La sentencia foreach es ideal para iterar a través de colecciones de elementos. Las colecciones son arrays tipicos pero tambien otros objetos .NET que implementan las interfaces IEnumerable.

foreach (tipo in colección) {

sentencia(s);

}

En esta sintaxis, tipo es un tipo de dato que estará contenido en la colección.

Este ciclo facilita a los programadores establecer un medio para recorrer todos los elementos de una colección sin tener que preocuparse por conocer por adelantado el número de elementos que contiene.

**2.4.3. Sentecia *while***

La sentencia while es similar a la sentencia for ya que también nos permite repetir un conjunto de sentencias mientras se cumpla determinada condición.

while (condición) {

sentencia(s)\_a\_repetir;

}

La diferencia entre el ciclo for y el while es que el primero establece una variable, una condición y un incremento dentro del paréntesis inicial, mientras que el ciclo while requiere que previamente se establezca una variable y que el incremento ocurra dentro del ciclo.

**2.4.4. Sentencia *do-while***

Esta sentencia opera de forma similar a la sentencia while, con dos diferencias:

1. el bloque de sentencias a repetir se ejecuta al menos una vez, sin importar si la condición se cumple o no, debido a que
2. la condición se evalúa al final de la iteración, no al principio como en la sentencia while.

do {

sentencia(s)\_a\_repetir;

} while (condición);

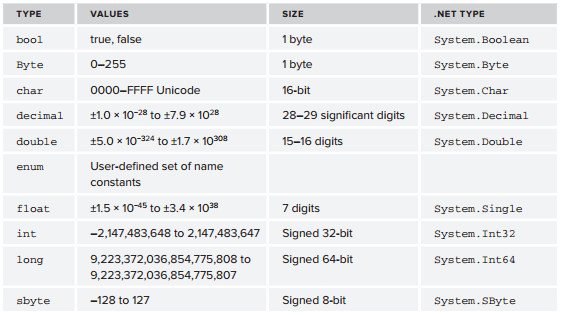
**Trabajando con el sistema de tipos de C#**

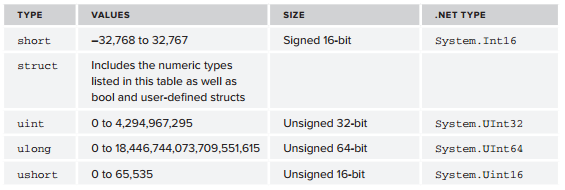
1. **Crear tipos de valor**

C# divide los tipos de valor en dos grandes grupos, como se muestra a continuación.

1. Estructuras (struct)
   1. Valores numéricos
      1. Valores enteros
      2. Valores de punto flotante
      3. Decimales
   2. Booleanos
   3. Estructuras definidas por el usuario
2. Enumeraciones

Los tipos de valor identifican valores específicos. Esta clase de tipos está presente en la mayoría de los lenguajes de programación y sirven para almacenar valores simples. Los tipos de valor se corresponden con tipos del .NET Framework contenidos en el espacio de nombres System.ValueType.





Es necesario comprender correctamente los tipos de datos simples para poder representar los datos que usarán nuestras aplicaciones para representar problemas del mundo real. También, comprender correctamente los tipos de datos nos ayuda a escribir código más eficiente dado que podemos elegir mejor el tipo de dato que se ajuste a las necesidades de almacenamiento de cada tipo de dato que necesitemos.

1. **Trabajando con estructuras de datos**

Las estructuras de datos, o simplemente structs, son tipos de valor que podemos usar para almacenar conjuntos de variables relacionadas entre sí. Las estructuras tienen algunas similaridades con las clases, pero también tienen sus propias restricciones.

Las estructuras pueden contener más que sólo propiedades. También pueden incluir funciones, constructores, constantes, indexadores, operadores, eventos y tipos anidados y también pueden implementar interfaces. Debemos comprender el uso de los constructores en las estructuras porque difieren ligeramente de cómo se emplean en las clases:

* Los constructores son opcionales, pero si se incluyen deben contener parámetros. No se permiten los constructores por defecto.
* Los campos no se pueden inicializar en el cuerpo de la estructura.
* Los campos se pueden inicializar únicamente usando el constructor o después de que el constructor es declarado.
* Los miembros privados se pueden inicializar únicamente mediante el constructor.
* Si se crea una nueva variable de un tipo struct sin usar el operador new, no se llamará al constructor si está presente.
* Si la estructura contiene un tipo de referencia (clase) como uno de sus miembros, se debe llamar explícitamente al constructor de dicho tipo.

1. **Trabajando con enumeraciones**