Módulo 5: C#

AUTOR: ABREU, Pablo

Índice

Introducción

C# (pronunciado "si sharp" en inglés) es un lenguaje de programación moderno, basado en objetos y con seguridad de tipos. que permite a los desarrolladores crear muchos tipos de aplicaciones seguras y sólidas que se ejecutan en el ecosistema de .NET. C# tiene sus raíces en la familia de lenguajes C, y a los programadores de C, C++, Java y JavaScript les resultará familiar inmediatamente.

C# es un lenguaje de programación **orientado a componentes**, orientado a objetos. C# proporciona construcciones de lenguaje para admitir directamente estos conceptos, por lo que se trata de un lenguaje natural en el que crear y usar componentes de software. Desde su origen, C# ha agregado características para admitir nuevas cargas de trabajo y prácticas de diseño de software emergentes.

Todos los tipos de C#, incluidos los tipos primitivos como int y double, se heredan de un único tipo **object** raíz. Todos los tipos comparten un conjunto de operaciones comunes. Los valores de cualquier tipo se pueden almacenar, transportar y operar de forma coherente. Además, C# admite tanto tipos de referencia definidos por el usuario como tipos de valor. C# permite la asignación dinámica de objetos y el almacenamiento en línea de estructuras ligeras. C# admite métodos y tipos genéricos, que proporcionan una mayor seguridad de tipos, así como un mejor rendimiento. C# también proporciona iteradores, gracias a los que los implementadores de clases de colecciones pueden definir comportamientos personalizados para el código de cliente.

Objetivos

El objetivo de este tutorial es introducir la programación del backend y el lenguaje de programación orientado a objeto C#. Nos enfocaremos en las principales características del lenguaje. Finalmente veremos un paso a paso para la creación de un proyecto modo consola.

Historia

C# es un lenguaje de programación que, pese a su "juventud", ha ganado gran popularidad en la industria, pues tiene un gran apoyo de la comunidad y de la infraestructura de Microsoft.

A 20 años de su creación, C#, desarrollado por Anders Hejlsberg, nace para ser rival de Java, ya que Sun (que luego compró Oracle) no quería que Microsoft hiciera cambios en Java, por lo que Microsoft decidió crear su propio lenguaje.

La idea de este lenguaje rondaba desde hace años. Durante el desarrollo de .NET Framework, las bibliotecas de clases se escribieron originalmente utilizando un sistema de compilación de código administrado llamado Simple Managed C (SMC).

En enero de 1999, Anders Hejlsberg formó un equipo para construir un nuevo lenguaje en ese momento llamado Cool, que en inglés significaba "lenguaje orientado a objetos tipo C". Microsoft había considerado mantener el nombre "Cool" como el nombre final del lenguaje, pero decidió no hacerlo por razones de marca registrada.

Para cuando el proyecto .NET se anunció públicamente en la Conferencia de Desarrolladores Profesionales de julio de 2000, el lenguaje había sido renombrado como C#, y las bibliotecas de clases y el tiempo de ejecución de ASP.NET habían sido portados a C #.

El principal diseñador y arquitecto principal de C # en Microsoft es Anders Hejlsberg, en entrevistas y documentos técnicos, ha declarado que las fallas en la mayoría de los lenguajes de programación más importantes (por ejemplo, C ++, Java, Delphi y Smalltalk) impulsaron los fundamentos del Common Language Runtime (CLR), que, a su vez, impulsó el diseño del lenguaje C # sí mismo.[1]

[1] https://recluit.com/historia-del-lenguaje-c/#.YQps8o70nIU

Versiones

La primera versión se lanzó en el año 2002. La última versión, C # 8 , se lanzó en septiembre de 2019.

1. **C# versión 1.0**: publicada con Visual Studio .NET 2002
2. **C# versión 1.2**: publicada con Visual Studio .NET 2003
3. **C# versión 2.0**: publicada con Visual Studio 2005
4. **C# versión 3.0**: publicada con Visual Studio 2008
5. **C# versión 4.0**: publicada con Visual Studio 2010
6. **C# versión 5.0**: publicada con Visual Studio 2012
7. **C# versión 6.0**: publicada con Visual Studio 2015
8. **C# versión 7.0**: se publicó con Visual Studio 2017
9. **C# versión 7.1** :agregó el elemento de configuración y nuevas características
10. **C# versión 7.2**: agrega varias características de lenguaje
11. **C# versión 7.3**: se han agregan nuevas opciones de compilador
12. **C# versión 8.0**: primera versión principal que tiene destino .NET Core
13. **C# versión 9.0**: se publicó con .NET 5

Características de C#:

- Sintaxis sencilla, que facilita al desarrollador la escritura de código.
- Sistema de tipo unificado, permitiendo realizar operaciones comunes y que los valores de todos los tipos se puedan almacenar, transportar y utilizar de manera coherente.
- Orientación a componentes. Hemos dicho que C# es lenguaje orientado a objetos, pero también a componentes porque permite definir propiedades sin necesidad de crear métodos o usar eventos sin tratar con punteros a funciones.
- Espacio de nombres. Se puede aislar o agrupar código mediante
- Bibliotecas. Todos los compiladores de C# tienen un mínimo de biblioteca de clases disponibles para usar.
- Integración con otros lenguajes.
- Multihilo. En C# puedes dividir el código en múltiples hilos de ejecución, trabajar en paralelo y sincronizarlos al final.[2]

[2] https://www.tokioschool.com/noticias/c-que-es/

Sintaxis

A modo general es importante tener en cuenta que cada instrucción del lenguaje C# termina con un punto y coma (;). Además C# se considera un lenguaje **case-sensitive** lo cual permite distinguir los identificadores entre mayúsculas y minúsculas, ejemplo: "MyClass" y "myclass" tienen un significado diferente.

Ejemplo de un programa;

```
using System;


class TestClass
{
    static void Main(string[] args)
    {
        // Muestra el número de argumentos de la línea de comandos.
        Console.WriteLine(args.Length);
    }
}
```

Comentarios

Los comentarios se pueden utilizar para explicar el código C# y hacerlo más legible. También se puede utilizar para evitar la ejecución al probar código alternativo.

Los comentarios de una sola línea comienzan con dos barras diagonales (//).

Ejemplo

```
//C # ignora cualquier texto entre y el final de la línea (no se ejecutará).
```

A continuación un ejemplo de un comentario de una sola línea antes de una línea de código:

Ejemplos:

```
// Esto es un comentario
Console.WriteLine("Hello World!");
```

```
Console.WriteLine("Hello World!");  // Esto es un comentario
```

Comentarios de varias líneas de C #

Los comentarios de varias líneas comienzan con /* y terminan con */. Cualquier texto entre /*y */será ignorado por el compilador de C#.

Este ejemplo utiliza un comentario de varias líneas (un bloque de comentarios) para explicar el código:

Ejemplo:

```
/* El siguiente código imprimirá las palabras Hello World
a la pantalla, y es asombroso */
Console.WriteLine("Hello World!");
```

Variables

Las variables son contenedores para almacenar valores de datos. En C#, existen diferentes tipos de variables (definidas con diferentes palabras clave), por ejemplo:

- **int** - almacena enteros (números enteros), sin decimales, como 123 o -123
- **double** - almacena números de punto flotante, con decimales, como 19,99 o -19,99
- **char** - almacena caracteres individuales, como 'a' o 'B'. Los valores de caracteres están rodeados por comillas simples
- **string** - almacena texto, como "Hola mundo". Los valores de cadena están rodeados por comillas dobles
- **bool** - almacena valores con dos estados: verdadero o falso

Todas las variables de C# deben identificarse con nombres únicos los cuales se denominan identificadores. Los identificadores pueden ser nombres cortos (como **x** e **y**) o nombres más descriptivos (edad, suma, volumen total). Como recomendación es de buen uso utilizar nombres descriptivos para crear un código comprensible y mantenible.

Las reglas generales para construir nombres para variables (identificadores únicos) son:

- Los nombres pueden contener letras, dígitos y el carácter de subrayado (_)
- Los nombres deben comenzar con una letra
- Los nombres deben comenzar con una letra minúscula y no pueden contener espacios en blanco
- Los nombres distinguen entre mayúsculas y minúsculas ("myVar" y "myvar" son variables diferentes)
- Las palabras reservadas (como palabras clave de C #, cómo into double) no se pueden usar como nombres

Para crear una variable, debe especificar el tipo y asignarle un valor:

type variableName = value;

Ejemplos

```
string name = "John";
Console.WriteLine(name);
int myNum = 15;
Console.WriteLine(myNum);
int myNum;
myNum = 15;
```

De igual manera es posible declarar muchas variables en la misma línea:

```
int x = 5, y = 6, z = 50;
```

Constantes

Además, es posible agregar la palabra reservada const para definir una variable a la cual no se desea que otros (o usted mismo) sobrescriban los valores existentes. Esto declarará la variable como "constante", lo que significa inmutable y de solo lectura.

Ejemplo:

```
const int myNum = 15;
myNum = 20; // error
```

Tipos de Datos

Un tipo de datos especifica el tamaño y el tipo de valores de variable. Es importante utilizar el tipo de datos correcto para la variable correspondiente; para evitar errores, para ahorrar tiempo y memoria, pero también hará que su código sea más fácil de mantener y leer.

Ejemplos:

```
int myNum = 5;               // Integer (whole number)
double myDoubleNum = 5.99D;  // Floating point number
char myLetter = 'D';         // Character
bool myBool = true;          // Boolean
string myText = "Hello";     // String
```

Los tipos de datos más comunes son:

int	4 bytes	Almacena números enteros desde -2,147,483,648 hasta 2,147,483,647
long	8 bytes	Almacena números enteros desde -9,223,372,036,854,775,808 hasta 9,223,372,036,854,775,807
float	4 bytes	Almacena números fraccionarios. Suficiente para almacenar de 6 a 7 dígitos decimales
double	8 bytes	Almacena números fraccionarios. Suficiente para almacenar 15 dígitos decimales
bool	1 bit	Almacena valores verdaderos o falsos
char	2 bytes	Almacena un solo carácter / letra, rodeado de comillas simples
string	2 bytes per character	Almacena una secuencia de caracteres, rodeada de comillas dobles.

Numéricos:

Los tipos de números se dividen en dos grupos:

1. Los tipos enteros almacenan números enteros, positivos o negativos (como 123 o -456), sin decimales. Los tipos válidos son int y long. El tipo que debe utilizar depende del valor numérico.
2. Los tipos de coma flotante representan números con una parte fraccionaria, que contienen uno o más decimales. Los tipos válidos son float y double.

INT

En general, el tipo de datos **int** es el tipo de datos preferido cuando creamos variables con un valor numérico:

```
int myNum = 100000;
```

Long

Se utiliza cuando int no es lo suficientemente grande para almacenar el valor. Tenga en cuenta que puede terminar el valor con una "L" (aunque no es obligatorio):

```
long myNum = 15000000000L;
```

Float

El tipo de datos float puede terminar el valor con una "F" (aunque no es obligatorio):

```
float myNum = 5.75F;
```

Double

El tipo de datos double puede terminar el valor con una "D" (aunque no es obligatorio):

```
double myNum = 19.99D;
```

¿Usar float o double?

La precisión de un valor de punto flotante indica cuántos dígitos puede tener el valor después del punto decimal. La precisión de **float** es de solo seis o siete dígitos decimales, mientras que las variables de tipo **double** tienen una precisión de aproximadamente 15 dígitos. Por lo tanto, es más seguro utilizar **double** para la mayoría de los cálculos.

Números científicos

Un número de coma flotante también puede ser un número científico con una "e" para indicar la potencia de 10:

Booleanos

Un tipo de datos booleano se declara con la bool palabra clave y solo puede tomar los valores true o false:

```
bool isCSharpFun = true;
bool isFishTasty = false;
```

Caracteres

El tipo de datos **char** se utiliza para almacenar un solo carácter. El carácter debe estar entre comillas simples, como 'A' o 'c':

```
char myGrade = 'B';
```

Cadenas de caracteres

El string tipo de datos se utiliza para almacenar una secuencia de caracteres (texto). Los valores de cadena deben estar entre comillas dobles:

```
string greeting = "Hello World";
```

Una cadena en C# es en realidad un objeto, que contiene propiedades y métodos que pueden realizar ciertas operaciones en cadenas. Por ejemplo, la longitud de una cadena se puede encontrar con la propiedad Length:

Hay muchos métodos de cadena disponibles, por ejemplo ToUpper() y ToLower(), que devuelve una copia de la cadena convertida a mayúsculas o minúsculas:

```
string txt = "Hello World";
Console.WriteLine(txt.ToUpper());   // Salida: "HELLO WORLD"
Console.WriteLine(txt.ToLower());   // Salida: "hello world"
```

El operador **+** se puede utilizar entre cadenas para combinarlas. Esto se llama concatenación :

```
string firstName = "John ";
string lastName = "Doe";
string name = firstName + lastName;
```

También puede usar el string.Concat()método para concatenar dos cadenas:

```
string firstName = "John ";
string lastName = "Doe";
string sameName = string.Concat(firstName, lastName);
```

Otra opción de concatenación de cadenas es la interpolación de cadenas , que sustituye valores de variables en marcadores de posición en una cadena. Tenga en cuenta que no tiene que preocuparse por los espacios, como con la concatenación:

```
string firstName = "John";
string lastName = "Doe";
string name = $"My full name is: {firstName} {lastName}";
```

Carácter de escape	Resultado	Descripción
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

Otros caracteres de escape útiles en C # son:

Code	Resultado
\n	New Line
\t	Tab
\b	Backspace

Casting de tipo

La conversión de tipos es cuando asigna un valor de un tipo de datos a otro tipo.

En C#, hay dos tipos de conversión:

Conversión implícita (automáticamente): conversión de un tipo más pequeño a un tamaño de letra más grande

char-> int-> long-> float->double

Conversión explícita (manualmente): conversión de un tipo más grande en un tipo de tamaño más pequeño

double-> float-> long-> int->char

Casting implícito

La conversión implícita se realiza automáticamente al pasar un tipo de tamaño más pequeño a un tipo de tamaño más grande:

```
int myInt = 9;
double myDouble = myInt;       // Automatic casting: int to double
```

Casting explicito

La conversión explícita se debe hacer manualmente colocando el tipo entre paréntesis delante del valor:

```
double myDouble = 9.78;
int myInt = (int) myDouble;    // Manual casting: double to int
```

Métodos de conversión de tipos

También es posible convertir tipos de datos de forma explícita mediante el uso de una función de métodos, tales como Convert.ToBoolean, Convert.ToDouble, Convert.ToString, Convert.ToInt32(int) y Convert.ToInt64(long):

```
int myInt = 10;
double myDouble = 5.25;
bool myBool = true;

Console.WriteLine(Convert.ToString(myInt));    // convert int to string
Console.WriteLine(Convert.ToDouble(myInt));    // convert int to double
Console.WriteLine(Convert.ToInt32(myDouble));  // convert double to int
Console.WriteLine(Convert.ToString(myBool));   // convert bool to string
```

Funciones

Las funciones son un elemento central en el desarrollo con C#. En efecto, todas las instrucciones de una aplicación escrita en C# deben situarse en funciones. Cada función representa una unidad de procesamiento reutilizable que puede recibir uno o varios parámetros y devolver un valor.

La escritura de funciones permite estructurar el código desacoplando de manera lógica las funcionalidades desarrolladas. Se recomienda, para una buena legibilidad y una buena mantenibilidad, limitar la longitud de las funciones. Muchos desarrolladores optan, de este modo, por una longitud que permita ver el código completo de una función en "una pantalla". Esta longitud es del todo relativa, pero puede convenir a cada uno. Esta regla no es, evidentemente, absoluta, pero puede ayudar, especialmente en casos de trabajo en equipo o en la relectura y la depuración.

Para alcanzar este objetivo es necesario limitar la responsabilidad de las funciones: cada una debería realizar un tipo de tarea únicamente.[3]

[3] https://www.ediciones-eni.com/open/mediabook.aspx?idR=01786c5432994b5903d498b18d800ddd

Operadores

Operadores aritméticos

+	Addition: Adds together two values	x + y	
-	Subtraction: Subtracts one value from another	x - y	
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	x++
--	Decrement	Decreases the value of a variable by 1	x--

Operadores de asignación

Operador	Examplo	Igual a
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Operadores de comparación

==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Operadores lógicos

&&	Logical and	Returns true if both statements are true

x < 5 && x < 10

||	Logical or	Returns true if one of the statements is true

x < 5 || x < 4

!	Logical not	Reverse the result, returns false if the result is true

!(x < 5 && x < 10)

Creando un Proyecto desde 0.

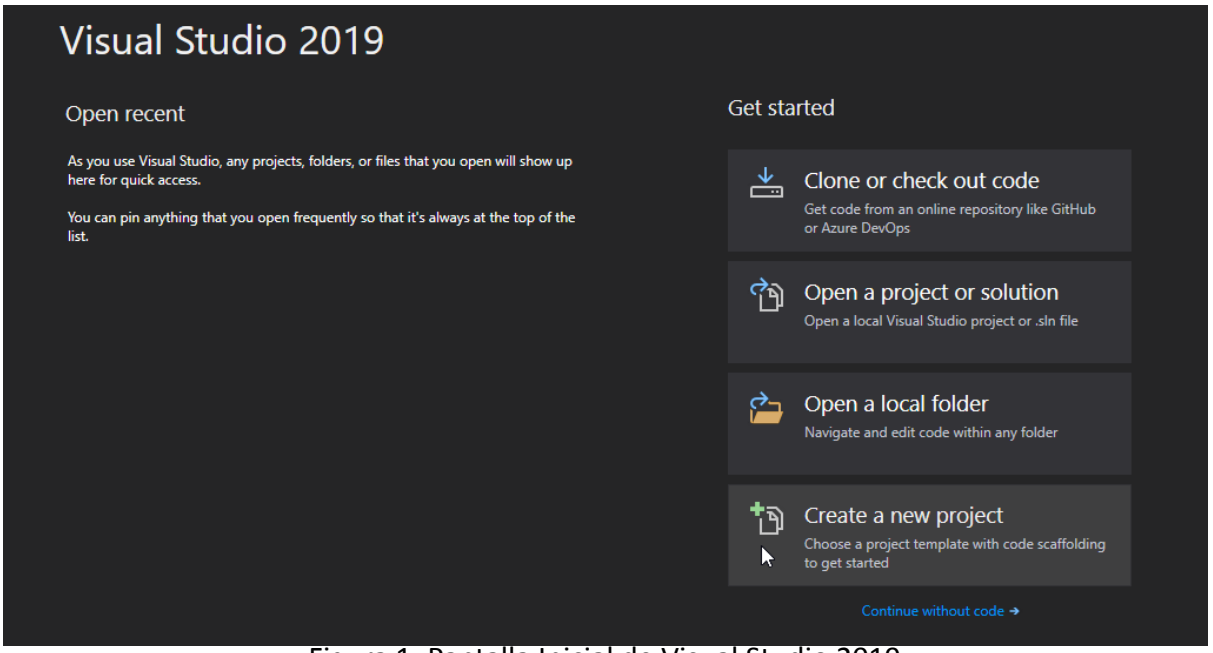

Figura 1: Pantalla Inicial de Visual Studio 2019

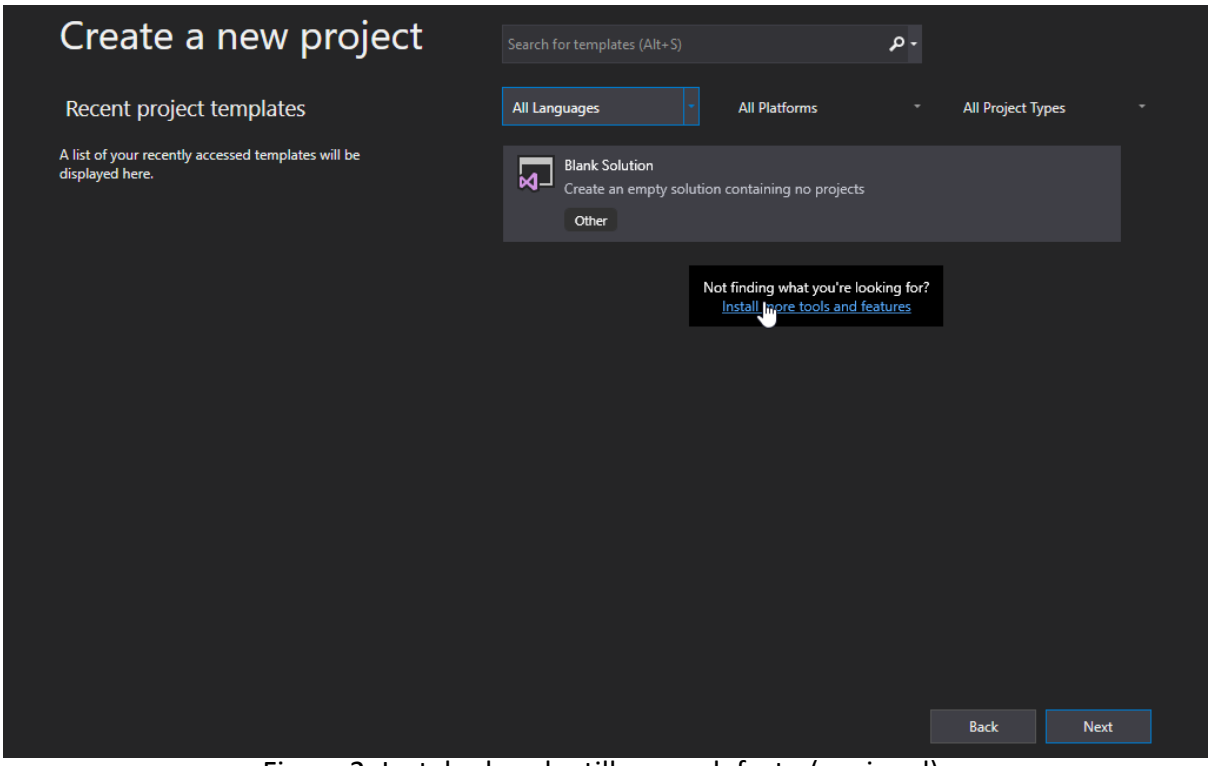

Figura 2: Instalar las plantillas por defecto (opcional).

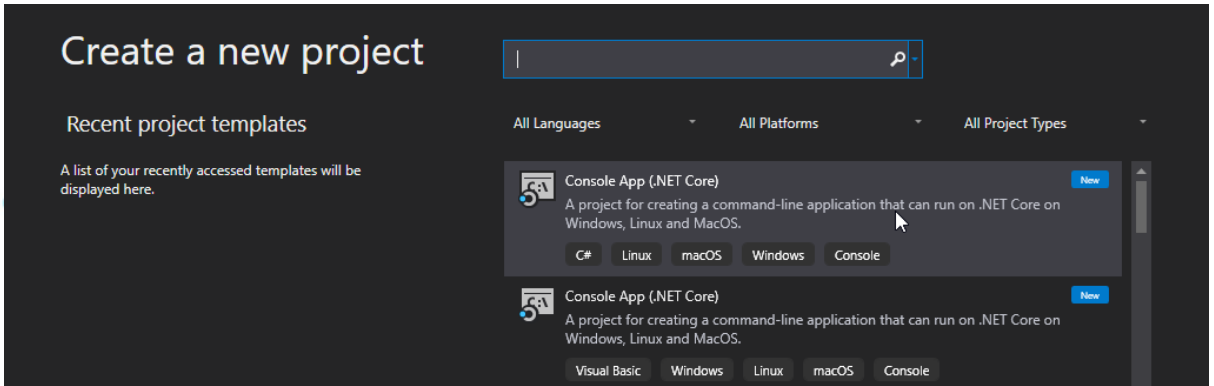

Figura 3: Seleccionar Console APP (.NetCore).

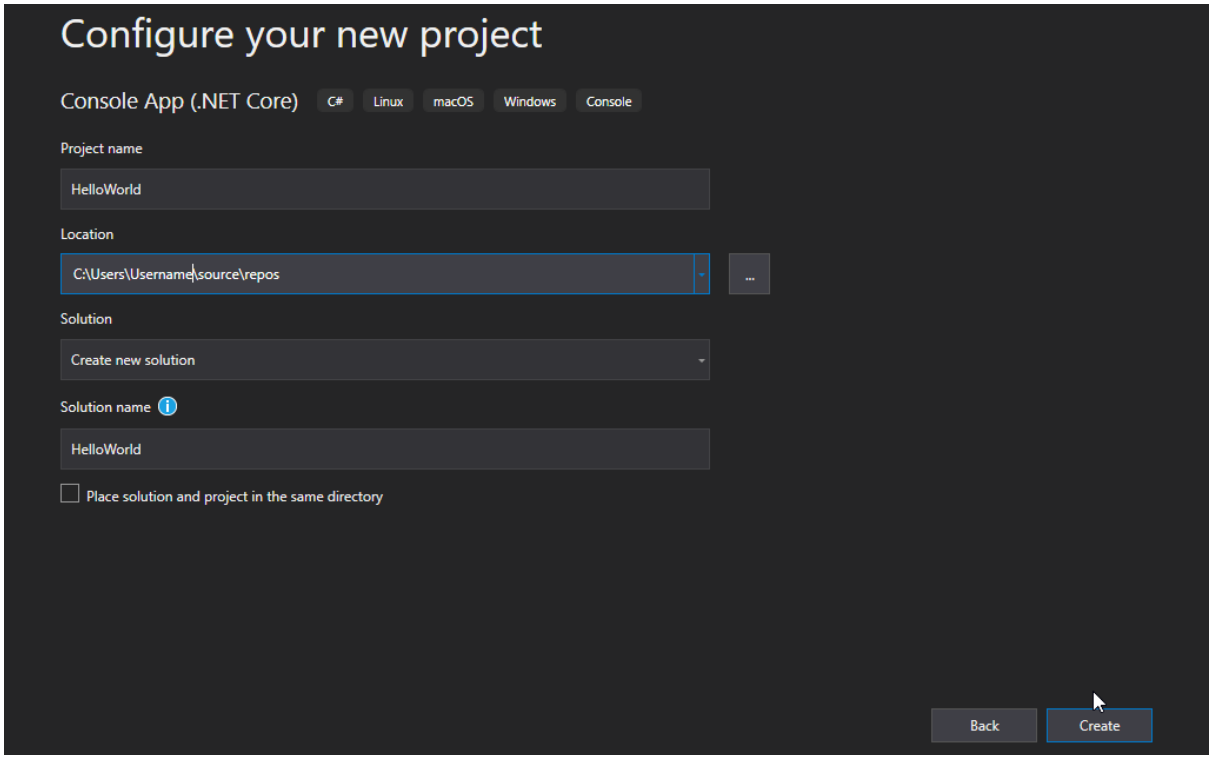

Figura 4: Configurar valores generales del proyecto.

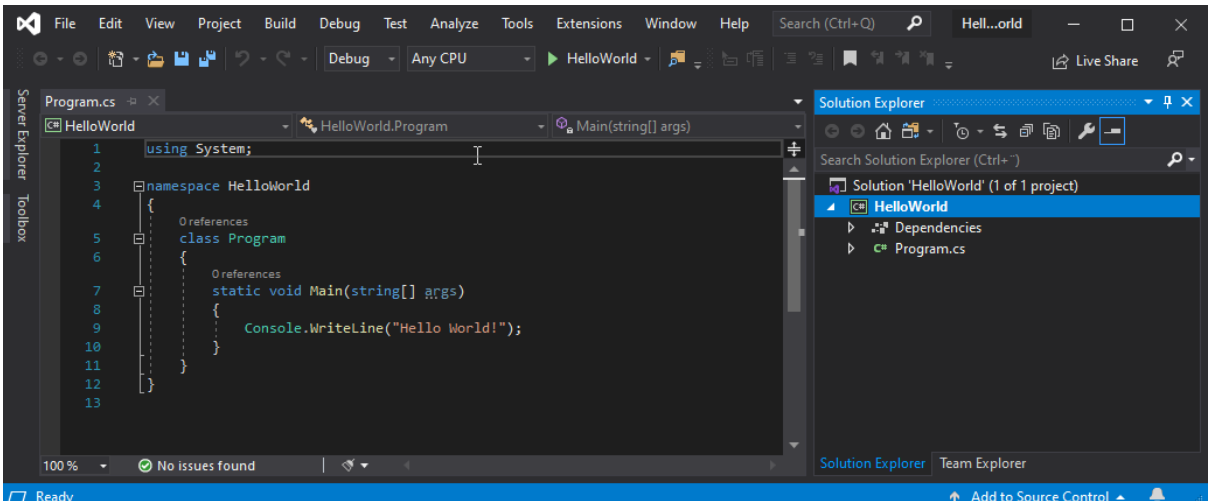

Figura 5: Pantalla inicial del proyecto.

WriteLine o Write

El método WriteLine() se usa a menudo para mostrar valores de variables en la ventana de la consola.

Para combinar texto y una variable, use el carácter +:

```
string name = "John";
Console.WriteLine("Hello " + name);
```

La diferencia es que WriteLine() imprime la salida en una nueva línea cada vez, mientras que Write() imprime en la misma línea (tenga en cuenta que debe recordar agregar espacios cuando sea necesario, para una mejor legibilidad):

Ejemplo:

```
Console.WriteLine("Hello World!");
Console.WriteLine("I will print on a new line.");

Console.Write("Hello World! ");
Console.Write("I will print on the same line.");
```

Entrada de Usuario

Conociendo que Console.WriteLine() se utiliza para imprimir valores. Ahora usaremos Console.ReadLine() para obtener la entrada del usuario:

```
Console.WriteLine("Enter your age:");
```

```
int age = Console.ReadLine();
Console.WriteLine("Your age is: " + age);
```

Si probamos este ejemplo no dará un error de Runtime con el mensaje de: no puede convertir implícitamente el tipo 'cadena' a 'int'. Lo que estaría faltando es poder convertir cualquier tipo de forma explícita, utilizando uno de los métodos Convert.To:

```
Console.WriteLine("Enter your age:");

int age = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Your age is: " + age);
```

Referencias

1. *https://recluit.com/historia-del-lenguaje-c/#.YQps8o70nIU*
2. *https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-version-history*
3. *https://docs.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/*
4. *https://www.tokioschool.com/noticias/c-que-es/*
5. *https://www.ediciones-eni.com/open/mediabook.aspx?idR=01786c5432994b5903d498b18d800ddd*