Instructivo de C# para principiantes

Carrera: Licenciatura en Producción y Desarrollo de Videojuegos – UNPAZ
Asignatura: Fundamentos de la Programación I
Docente: Lic. Facundo Nicolás Suárez

El objetivo de este instructivo es proporcionar a los estudiantes una **guía básica para iniciarse en la programación en C#**, con ejemplos prácticos de los elementos fundamentales del lenguaje. Este material se presenta como un **resumen introductorio**, que permitirá dar los primeros pasos en la creación de programas y videojuegos en C#.

Existe un texto más amplio, titulado *Fundamentos de la Programación de Videojuegos*, donde abordamos conceptos más amplios de programación, estructuras de control y buenas prácticas de manera más detallada y orientada a la producción de videojuegos. El presente instructivo no reemplaza ese texto, sino que sirve como **guía resumida para los primeros ejercicios prácticos en C#**, facilitando la comprensión de los elementos esenciales antes de abordar proyectos más complejos.

¿Qué es C#?

C# es un lenguaje de programación moderno, **orientado a objetos**, desarrollado por Microsoft. Se utiliza ampliamente en:

- Desarrollo de **aplicaciones de escritorio** (Windows Forms, WPF).
- Creación de **videojuegos** (por ejemplo, con Unity).
- Aplicaciones **web** (con ASP.NET) y **móviles** (Xamarin / MAUI).

Entre sus características más importantes se encuentran:

- Sintaxis clara y consistente.
- Soporte para **programación orientada a objetos** (clases, objetos, herencia, etc.).
- Amplia biblioteca estándar (framework .NET) que incluye funciones para entradas/salidas, matemáticas, fechas, colecciones y más**.**
- Ejecución gestionada, lo que evita errores de memoria frecuentes en lenguajes de bajo nivel.

En este instructivo nos enfocaremos en **programación básica para consola**, es decir, programas que se ejecutan en una ventana de texto, lo que permite concentrarse en **la lógica y las estructuras del lenguaje**, sin necesidad de interfaces gráficas complejas.

Estructura básica de un programa en C#

Todo programa en C# se organiza en **clases y métodos**. El punto de partida siempre es el método `Main`.

```
using System; // Permite utilizar clases básicas del framework .NET

class Program // Clase principal del programa
{
    static void Main(string[] args) // Punto de entrada del programa
    {
        Console.WriteLine("Hola, mundo"); // Muestra un mensaje en la consola
    }
}
```

Explicación

- `using System;`: importa un conjunto de herramientas básicas, incluyendo la consola.
- `class Program`: define una clase donde se organiza el código. En C#, **toda función debe estar dentro de una clase**.
- `static void Main(string[] args)`: método principal del programa. Es el **punto donde comienza la ejecución**.
- `Console.WriteLine()`: imprime un mensaje en la consola.
- `Console.ReadLine()`: lee un texto ingresado por el usuario.

Variables y tipos de datos

Las **variables** son espacios de memoria donde almacenamos datos. Pueden ser de distintos tipos:

```
int edad = 20;          // Número entero
float altura = 1.75f;   // Número decimal
double peso = 70.5;     // Número decimal más preciso
string nombre = "Ana"; // Texto
bool esEstudiante = true; // Valor lógico (true / false)
```

- **Constantes:** valores que no cambian durante la ejecución.

```
const int MAX_VIDA = 10;
```

Consejos:

- Usar nombres descriptivos (`fuerza`, `equilibrio`, `puntaje`)
- Inicializar variables con valores coherentes según su uso.

Operadores

Aritméticos	Comparación	Lógicos

`int a = 5;`
`int b = 2;`
 	
`int suma = a + b;`
`int resta = a - b;`
`int mult = a * b;`
`int div = a / b; // División entera`
`int mod = a % b; // Resto`
	`bool igual = a == b; // false`
	`bool distinto = a != b; // true`
	`bool mayor = a > b; // true`
	`bool menorIgual = a <= b; // false`
		`bool resultado = (a > b) && (b > 0); // true`
		`bool resultado2 = (a < b) || (b > 0); // true`
		`bool negacion = !(a == b); // true`

Entrada y salida de datos

Todo programa necesita comunicarse con el usuario. En C#, esto se logra mediante la consola, utilizando instrucciones que permiten mostrar mensajes en pantalla y leer información ingresada por teclado. A continuación se presentan los métodos básicos para realizar esta interacción.

```
Console.WriteLine("Ingresa tu nombre:");
string nombre = Console.ReadLine();
Console.WriteLine("Hola, " + nombre + "!");
```

Para números, se debe convertir el texto leído. En C#, el método **`int.Parse()`** se utiliza para convertir un texto (string) en un número entero (int).

```
Console.WriteLine("Ingresa tu edad:");
int edad = int.Parse(Console.ReadLine());
```

Condicionales

Las estructuras condicionales permiten que el programa **tome decisiones** según determinadas condiciones. En C#, las más comunes son **`if / else`** y **`switch / case`**, que se utilizan para ejecutar distintos bloques de código dependiendo del valor de una variable o el resultado de una comparación.

`if` / `else`

```
int edad = 18;
if (edad >= 18)
{
    Console.WriteLine("Eres mayor de edad");
}
else
{
    Console.WriteLine("Eres menor de edad");
}
```

`switch` / `case`

```
int opcion = 2;
switch (opcion)
{
    case 1:
        Console.WriteLine("Opción 1 seleccionada");
        break;
    case 2:
        Console.WriteLine("Opción 2 seleccionada");
        break;
    default:
        Console.WriteLine("Opción inválida");
        break;
}
```

Bucles

Los bucles o ciclos permiten **repetir un bloque de instrucciones** varias veces, ya sea una cantidad conocida de veces (`for`) o mientras se cumpla una condición (`while`). Son fundamentales para automatizar tareas y evitar repetir código de forma manual.

`for`

```
for (int i = 1; i <= 5; i++)
{
    Console.WriteLine("Número: " + i);
}
```

`while`

```
int i = 1;
while (i <= 5)
{
    Console.WriteLine("Número: " + i);
    i++;
}
```

Funciones y rutinas

Las funciones, también llamadas **rutinas o métodos**, permiten **organizar el código en bloques reutilizables**. Esto mejora la claridad del programa y evita repetir instrucciones. En C#, las rutinas pueden no devolver un valor (`void`), modificar variables usando `ref`, o devolver un resultado mediante `return`.

En otras palabras, permiten **organizar el código en bloques reutilizables**.

Rutina `void`

```
void Saludar(string nombre)
{
    Console.WriteLine("Hola, " + nombre + "!");
}

// Uso
Saludar("Ana");
```

Rutina con paso por referencia

```
void Aumentar(ref int valor, int cantidad)
{
    valor += cantidad;
}

// Uso
int energia = 5;
```

```
    Aumentar(ref energia, 3); // energia ahora es 8
```

Función con retorno

```
int Sumar(int a, int b)
{
    return a + b;
}

int resultado = Sumar(3, 4); // 7
```

Limitar valores: Clamp

En muchos casos es necesario asegurar que una variable no supere ciertos límites (por ejemplo, que la energía de un personaje no sea menor que 0 ni mayor que 10). En C#, la función `Math.Clamp()` permite mantener un valor dentro de un rango determinado de forma sencilla y segura.

```
int vida = 12;
vida = Math.Clamp(vida, 0, 10); // vida será 10
```

Aleatoriedad

La generación de valores aleatorios es muy útil en videojuegos y simulaciones, por ejemplo para crear eventos impredecibles. En C#, la clase `Random` permite obtener números aleatorios dentro de un rango específico, lo que facilita introducir variabilidad en el comportamiento del programa.

```
Random rnd = new Random();
int numero = rnd.Next(1, 4); // genera 1, 2 o 3
```

Buenas prácticas

- Mantener **nombre de variables claras y significativas**
- Comentar el código para que sea legible
- Separar la lógica en **rutinas** o **funciones**
- Probar el programa por partes
- Usar **estructuras de control y bucles** adecuadamente

Ejercitación en C#

Fundamentos de la Programación I

Carrera: Licenciatura en Producción y Desarrollo de Videojuegos – UNPAZ
Docente: Lic. Facundo Nicolás Suárez

La siguiente consigna es una variación de un ejercicio planteado originalmente por el docente Rod Gonzalez.

Consigna: "Siete días en el Wallmapu"

El siguiente trabajo propone desarrollar un videojuego textual en **C#**, utilizando la **consola** como interfaz de usuario. El objetivo del ejercicio es aplicar los conceptos fundamentales de estructuras de control, rutinas, paso de parámetros, operadores lógicos y condicionales, en el contexto de un juego narrativo basado en la gestión de recursos y toma de decisiones.

Contexto narrativo

El jugador asume el rol de un **joven mapuche** que forma parte de su comunidad en el Wallmapu. Durante el transcurso de una semana (siete días), deberá mantener el equilibrio entre su **fuerza**, su **espíritu** y su relación con el entorno natural y social, representada por la variable **equilibrio**. Cada decisión que tome va a modificar estas energías, pudiendo afectar el desarrollo del ciclo.

El objetivo del juego es **mantener las energías en valores positivos hasta completar los siete días**. Si alguna de las energías llega a cero, el ciclo finaliza y el jugador deberá comenzar nuevamente. Al finalizar los siete días, podrá comenzar un nuevo ciclo, repitiendo la experiencia con nuevos aprendizajes.

Energías del jugador

Energía	Valor inicial	Condición de finalización
Fuerza	10	El personaje colapsa por agotamiento.
Espíritu	10	El personaje pierde la conexión con su propósito.
Equilibrio	10	Se rompe la armonía con el entorno.

Acciones posibles

Cada día, el jugador podrá realizar una acción. Cada acción modifica las energías de la siguiente manera:

Acción	Efecto
Recolectar alimentos	+3 fuerza, -2 equilibrio, +1 espíritu
Compartir con la comunidad	+2 equilibrio, +2 espíritu, -1 fuerza
Caminar por el bosque	-2 fuerza, -1 equilibrio, +3 espíritu, y genera un evento aleatorio
Descansar junto al fogón	+4 fuerza, -2 equilibrio, +1 espíritu

Eventos aleatorios (asociados a "Caminar por el bosque")

Cada vez que el jugador elija la acción "Caminar por el bosque", podrá ocurrir uno de los siguientes **eventos aleatorios**:

Evento	Efecto
Encuentra frutos silvestres	+3 equilibrio
Se topa con un terreno difícil	-4 fuerza
Ve cómo crecen nuevos brotes en un árbol dañado	+2 espíritu

Ciclo del juego

1. Mostrar el día actual y el estado del jugador.
2. Presentar las acciones disponibles.
3. Aplicar los efectos de la acción elegida.
4. Si corresponde, ejecutar un evento aleatorio.
5. Verificar si el ciclo continúa o si alguna energía llegó a cero.

Al completar los siete días, el jugador podrá **reiniciar el ciclo**, comenzando nuevamente con los valores iniciales.

Requisitos técnicos

- El programa debe desarrollarse en **C#** y ejecutarse íntegramente desde la consola.
- Deberán utilizarse estructuras de control tales como **condicionales (if / else)**, **bucles**, **switch**, **rutinas** (`void` y `return`), y **Clamp**.
- Se recomienda el uso de **parámetros pasados por referencia** (`ref`) para modificar las energías del jugador dentro de las rutinas.
- Deberán emplearse las funciones `Console.WriteLine()` y `Console.ReadLine()` para la interacción con el usuario.
- El código deberá estar **correctamente ordenado, comentado y con un formato claro y legible**.

Rutinas sugeridas

```
void RecolectarAlimentos(ref int fuerza, ref int espiritu, ref int equilibrio)
void CompartirComunidad(ref int fuerza, ref int espiritu, ref int equilibrio)
void CaminarBosque(ref int fuerza, ref int espiritu, ref int equilibrio)
void DescansarFogon(ref int fuerza, ref int espiritu, ref int equilibrio)
bool JuegoContinua(int fuerza, int espiritu, int equilibrio, int dia)
```

Mensajes de finalización sugeridos

- **Si completa los siete días:** "Mantuviste el equilibrio del Wallmapu. Tu comunidad reconoce tu esfuerzo y sabiduría. Podés volver a empezar."
- **Si alguna energía llega a cero:** "Perdiste el equilibrio, pero cada ciclo es una oportunidad para aprender y comenzar de nuevo."