Estructuras de datos y de control

Contenido

Introducción:

Como es común al aprender cualquier lenguaje de programación, comenzaremos con un ejercicio básico: imprimir "Hola Mundo" en la pantalla. Este es un paso fundamental para familiarizarse con la sintaxis y la estructura del código en el lenguaje que estamos aprendiendo. Aunque pueda parecer simple, este pequeño programa nos enseñará cómo escribir código, cómo ejecutarlo y cómo ver los resultados.

Código "Hola Mundo" en C#:

```
using System;

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hola Mundo");
    }
}
```

Explicación:

using System;: Esta línea le dice al compilador que estamos utilizando el espacio de nombres System, que contiene la clase Console que necesitamos para imprimir en la consola.

class Program: Aquí declaramos una clase llamada Program, que es donde se encuentra nuestro programa principal.

static void Main(string[] args): Este es el método principal de nuestro programa. Es estático (static) porque es el punto de entrada de nuestra aplicación. Toma un arreglo de argumentos de línea de comandos como parámetro, aunque en este caso no lo estamos utilizando.

Console.WriteLine("Hola Mundo");: Esta línea imprime "Hola Mundo" en la consola y agrega un salto de línea después.

Espacio de nombres (using)

using System; // Este using permite acceder a clases y funciones definidas en el espacio de nombres System.

En C#, los using se utilizan para importar espacios de nombres externos al archivo actual. Esto permite acceder a clases y funciones definidas en esos espacios de nombres sin necesidad de escribir el nombre completo cada vez.

Clase Principal (class Program)

class Program

```
{
    // Esta es la clase principal de la aplicación. Contiene el método Main que es el punto de entrada.
}
```

En C#, un programa comienza ejecutando el método Main, el cual debe estar contenido en una clase. En este caso, la clase se llama Program. Este método es estático (static) porque no es necesario crear una instancia de la clase para llamarlo.

Método Main (static void Main(string[] args))

```
static void Main(string[] args)
{
    // Este método es el punto de entrada del programa. Aquí es donde comienza la ejecución.
}
```

El método Main es el punto de entrada de la aplicación. Es estático porque se invoca sin necesidad de crear una instancia de la clase. Recibe un array de strings como argumento (args), que puede contener argumentos de línea de comandos pasados al programa al iniciarlo.

Llaves ({})

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

Las llaves delimitan bloques de código en C#. En este ejemplo, las llaves encierran el código que se ejecuta si la condición del if se cumple y el código que se ejecuta si no se cumple.

Puntos y coma (;)

Los puntos y coma indican el final de una instrucción en C#. Cada línea de código debe terminar con un punto y coma.

Comentarios

```
// Este es un comentario de una línea


/*
   Este es un comentario de
   múltiples líneas
*/
```

Los comentarios son texto que se incluye en el código fuente pero que el compilador ignora. Se utilizan para hacer anotaciones o explicaciones sobre el código. Los comentarios de una línea comienzan con //, y los comentarios de varias líneas se delimitan con /* y */.

Leer información del teclado (Console.ReadLine())

```
Console.WriteLine("Ingrese su nombre:");

string nombre = Console.ReadLine();
```

Console.ReadLine() espera a que el usuario ingrese una línea de texto por la consola y la devuelve como un string. En este caso, se almacena en la variable nombre.

Mostrar información por consola (Console.WriteLine())

```
Console.WriteLine("Hola " + nombre + "! Tienes " + edad + " años y tu salario es: " + salario);
```

Console.WriteLine() muestra el texto especificado en la consola, seguido de un salto de línea. Puede contener cadenas de texto y variables concatenadas, como en este caso.

Definir variables de tipo string, entero y flotante

```
string cadena = "Hola mundo"; // Variable de tipo string

int entero = 10; // Variable de tipo entero

float flotante = 3.14f; // Variable de tipo flotante
```

En C#, puedes definir variables utilizando el tipo de dato seguido del nombre de la variable y su valor inicial (si lo tiene).

Variable del tipo Random

Para declarar una variable del tipo **Random** en C#, puedes hacerlo de la siguiente manera:

```
Random rnd = new Random();
```

Aquí **rnd** es el nombre de la variable que estás declarando y **new Random()** es la instancia de la clase **Random**. Esto crea un nuevo objeto **Random** que puedes usar para generar números aleatorios.

Una vez que has declarado esta variable, puedes utilizar sus métodos para generar números aleatorios según sea necesario. Por ejemplo:

int numeroAleatorio = rnd.Next(1, 101); // Genera un número aleatorio entre 1 y 100

En este caso, **Next(1, 101)** genera un número aleatorio entre 1 (inclusive) y 101 (exclusivo), lo que significa que el número puede ser 1 pero nunca será 101.

Estructuras de control condicional

Condicional Simple (if)

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

El if ejecuta un bloque de código si una condición dada es verdadera.

Condicional Doble (if-else)

csharp

Copy code

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

El if-else ejecuta un bloque de código si la condición es verdadera y otro bloque de código si la condición es falsa.

Estructura switch

```
switch (opcion)
{
    case 1:
        Console.WriteLine("Opción 1 seleccionada");
        break;
    case 2:
        Console.WriteLine("Opción 2 seleccionada");
        break;
    default:
        Console.WriteLine("Opción no reconocida");
        break;
}
```

El switch selecciona uno de varios bloques de código para ejecutar, dependiendo del valor de una expresión.

Estructuras de control iterativas

Ciclo while

```
while (contador < 10)

{

   Console.WriteLine(contador);

   contador++;

}
```

El ciclo while ejecuta un bloque de código mientras una condición dada sea verdadera.

Ciclo do-while

```
do

{

   Console.WriteLine(contador);

   contador++;

} while (contador < 10);
```

El ciclo do-while es similar al while, pero garantiza que el bloque de código se ejecute al menos una vez antes de evaluar la condición.

Ciclo for

```
for (int i = 0; i < 5; i++)

{

   Console.WriteLine("Iteración " + (i + 1));

}
```

El ciclo for ejecuta un bloque de código un número específico de veces.

Estructura foreach (Recorriendo un string caracter por caracter)

```
string mensaje = "Hola mundo";


foreach (char caracter in mensaje)
```

```
{
    Console.WriteLine(caracter);
}
```

El foreach se utiliza para recorrer colecciones de elementos, como arrays o colecciones. En este caso, recorre cada caracter en la cadena mensaje e imprime cada caracter por separado.

Ejemplo Completo

```
using System; // Espacio de nombres para las clases fundamentales de C#

class Program // Clase principal
{
    static void Main(string[] args) // Método principal, estático porque es el punto de entrada de la aplicación
    {
        // Leer información del teclado
        Console.WriteLine("Ingrese su nombre:");
        string nombre = Console.ReadLine(); // Leer una línea de texto desde el teclado

        Console.WriteLine("Ingrese su edad:");
        int edad = int.Parse(Console.ReadLine()); // Leer una línea de texto y convertirla a entero

        Console.WriteLine("Ingrese su salario:");
        float salario = float.Parse(Console.ReadLine()); // Leer una línea de texto y convertirla a flotante

        // Mostrar información por consola
        Console.WriteLine("Hola " + nombre + "! Tienes " + edad + " años y tu salario es: " + salario);

        // Definir variables de tipo string, entero y flotante
        string cadena = "Hola mundo"; // Variable de tipo string
        int entero = 10; // Variable de tipo entero
        float flotante = 3.14f; // Variable de tipo flotante

        // Estructuras de control condicional
        if (edad >= 18)
        {
            Console.WriteLine("Eres mayor de edad");
        }
        else
        {
            Console.WriteLine("Eres menor de edad");
        }

        // Estructuras de control iterativas
        for (int i = 0; i < 5; i++)
        {
```

```
            Console.WriteLine("Iteración " + (i + 1));
        }

        // Comentarios
        // Este es un comentario de una línea
        /*
            Este es un comentario de
            múltiples líneas
        */

        // Llaves ({}) en el código delimitan bloques de código, como en estructuras condicionales o de repetición
        // Puntos y comas (;) al final de las declaraciones de código indican el fin de una instrucción en C#
    }
}
```

Declaración de Arreglos

En C#, puedes declarar arreglos unidimensionales y bidimensionales.

Arreglos Unidimensionales

int[] arregloUnidimensional = new int[5]; // Arreglo unidimensional de enteros con 5 elementos

Arreglos Bidimensionales

int[,] arregloBidimensional = new int[3, 3]; // Arreglo bidimensional de enteros con 3 filas y 3 columnas

Recorrido de Arreglos

Recorrido con for

```
for (int i = 0; i < arregloUnidimensional.Length; i++)

{

   Console.WriteLine(arregloUnidimensional[i]);

}

for (int i = 0; i < arregloBidimensional.GetLength(0); i++)

{

   for (int j = 0; j < arregloBidimensional.GetLength(1); j++)

   {
```

```
        Console.WriteLine(arregloBidimensional[i, j]);
    }
}
```

Recorrido con foreach

```
foreach (int elemento in arregloUnidimensional)
{
    Console.WriteLine(elemento);
}
```

No se puede usar foreach para recorrer un arreglo bidimensional, solo funciona con arreglos unidimensionales.

Diferencia entre for y foreach

for: Se utiliza cuando necesitas un control más preciso sobre el índice de iteración o necesitas recorrer el arreglo en una dirección específica.

foreach: Es más simple y limpio, y se utiliza cuando solo necesitas recorrer todos los elementos del arreglo sin preocuparte por los índices de iteración.

Modificación de Elementos con foreach

En C#, no es posible modificar los elementos de un arreglo mientras lo recorres con foreach. Esto se debe a que foreach trabaja con una copia de los elementos del arreglo, por lo que cualquier modificación hecha dentro del bucle no afectará al arreglo original.

Declaración y Llamada de Funciones

Declaración de Funciones

```
public static void Saludar(string nombre)

{

    Console.WriteLine("Hola, " + nombre + "!");

}
```

public: Es el modificador de acceso, que indica que la función puede ser accedida desde cualquier lugar.

static: Indica que la función es estática y puede ser llamada sin crear una instancia de la clase que la contiene.

void: Es el tipo de retorno de la función. void significa que la función no devuelve ningún valor.

Saludar: Es el nombre de la función.

string nombre: Es un parámetro de entrada de tipo string llamado nombre.

Llamada de Funciones

```
Saludar("Juan");
```

Uso de static

Cuando un método es static, significa que pertenece a la clase en lugar de a una instancia de la clase. Esto significa que puedes llamar al método directamente desde la clase sin necesidad de crear un objeto de la clase.

Parámetros de entrada y Retorno de la Función

Parámetros: Son valores que se pasan a una función para que pueda realizar su trabajo. Pueden ser de entrada, de salida o de entrada y salida.

Retorno de la Función: Son valores que una función devuelve después de realizar su trabajo, a la función que no devuelve valores de retorno se indicara que devuelve void.

Funciones sin Parámetros

```
public static void ImprimirMensaje()

{

    Console.WriteLine("Este es un mensaje de prueba.");

}
```

Parámetros de Entrada por Valor y por Referencia

- Por Valor: Se pasa una copia del valor original. Los cambios en el parámetro no afectan a la variable original.
- Por Referencia: Se pasa una referencia a la variable original. Los cambios en el parámetro afectan a la variable original.

Por Valor

```
public static void Incrementar(int num)

{

   num++;

}
```

Por Referencia

```
public static void IncrementarRef(ref int num)

{

   num++;

}
```

Parámetros Opcionales

```
public static void Saludar(string nombre, string saludo = "Hola")

{

   Console.WriteLine(saludo + ", " + nombre + "!");

}
```

En este ejemplo, saludo es un parámetro opcional con un valor predeterminado de "Hola". Si no se proporciona un valor para saludo, se usará el valor predeterminado.

Ejemplo Completo

```
csharp
Copy code
using System;

class Program
{
    static void Main()
    {
        Saludar("Juan");
        ImprimirMensaje();

        int numero = 5;
        Incrementar(numero);
        Console.WriteLine("Valor después de Incrementar: " + numero);

        IncrementarRef(ref numero);
        Console.WriteLine("Valor después de IncrementarRef: " + numero);

        Saludar("Pedro", "Hola amigo");
    }

    public static void Saludar(string nombre, string saludo = "Hola")
    {
        Console.WriteLine(saludo + ", " + nombre + "!");
    }

    public static void ImprimirMensaje()
    {
        Console.WriteLine("Este es un mensaje de prueba.");
    }

    public static void Incrementar(int num)
    {
        num++;
    }

    public static void IncrementarRef(ref int num)
    {
        num++;
    }
}
```

Obtener el Mes, Año y Día de una Fecha

DateTime fecha = DateTime.Now;

int mes = fecha.Month; // Obtener el mes (1-12)

int año = fecha.Year; // Obtener el año

int día = fecha.Day; // Obtener el día del mes

Sumar y Restar Días a una Fecha

DateTime fechaActual = DateTime.Now;

DateTime nuevaFecha = fechaActual.AddDays(5); // Sumar 5 días

DateTime otraFecha = fechaActual.AddDays(-3); // Restar 3 días

Determinar el Día de la Semana con una Fecha Dada

DateTime fecha = new DateTime(2024, 3, 12);

DayOfWeek díaSemana = fecha.DayOfWeek; // Obtener el día de la semana (Lunes, Martes, ...)

Tipo DateTime y su Comportamiento

Cuando se declara una variable DateTime sin especificar una hora, se inicializa con la fecha y hora actual del sistema. Si se declara como DateTime fecha;, la variable fecha tendría la fecha y hora actual del momento en que se ejecuta esa línea de código.

Métodos de DateTime

AddDays(int días): Suma una cantidad especificada de días a la fecha.

AddMonths(int meses): Suma una cantidad especificada de meses a la fecha.

AddYears(int años): Suma una cantidad especificada de años a la fecha.

DayOfWeek: Devuelve el día de la semana de la fecha.

Month: Devuelve el mes de la fecha (1-12).

Year: Devuelve el año de la fecha.

Day: Devuelve el día del mes de la fecha.

Ejemplo Completo

```
using System;

class Program
{
    static void Main()
    {
        // Obtener el Mes, Año y Día de una Fecha
        DateTime fecha = DateTime.Now;
        int mes = fecha.Month;
        int año = fecha.Year;
        int día = fecha.Day;

        // Sumar y Restar Días a una Fecha
        DateTime fechaActual = DateTime.Now;
        DateTime nuevaFecha = fechaActual.AddDays(5);
        DateTime otraFecha = fechaActual.AddDays(-3);

        // Determinar el Día de la Semana con una Fecha Dada
        DateTime fechaDada = new DateTime(2024, 3, 12);
        DayOfWeek díaSemana = fechaDada.DayOfWeek;

        // Mostrar resultados
        Console.WriteLine("Mes: " + mes);
        Console.WriteLine("Año: " + año);
        Console.WriteLine("Día: " + día);

        Console.WriteLine("Nueva fecha: " + nuevaFecha);
        Console.WriteLine("Otra fecha: " + otraFecha);

        Console.WriteLine("Día de la semana: " + díaSemana);
    }
}
```