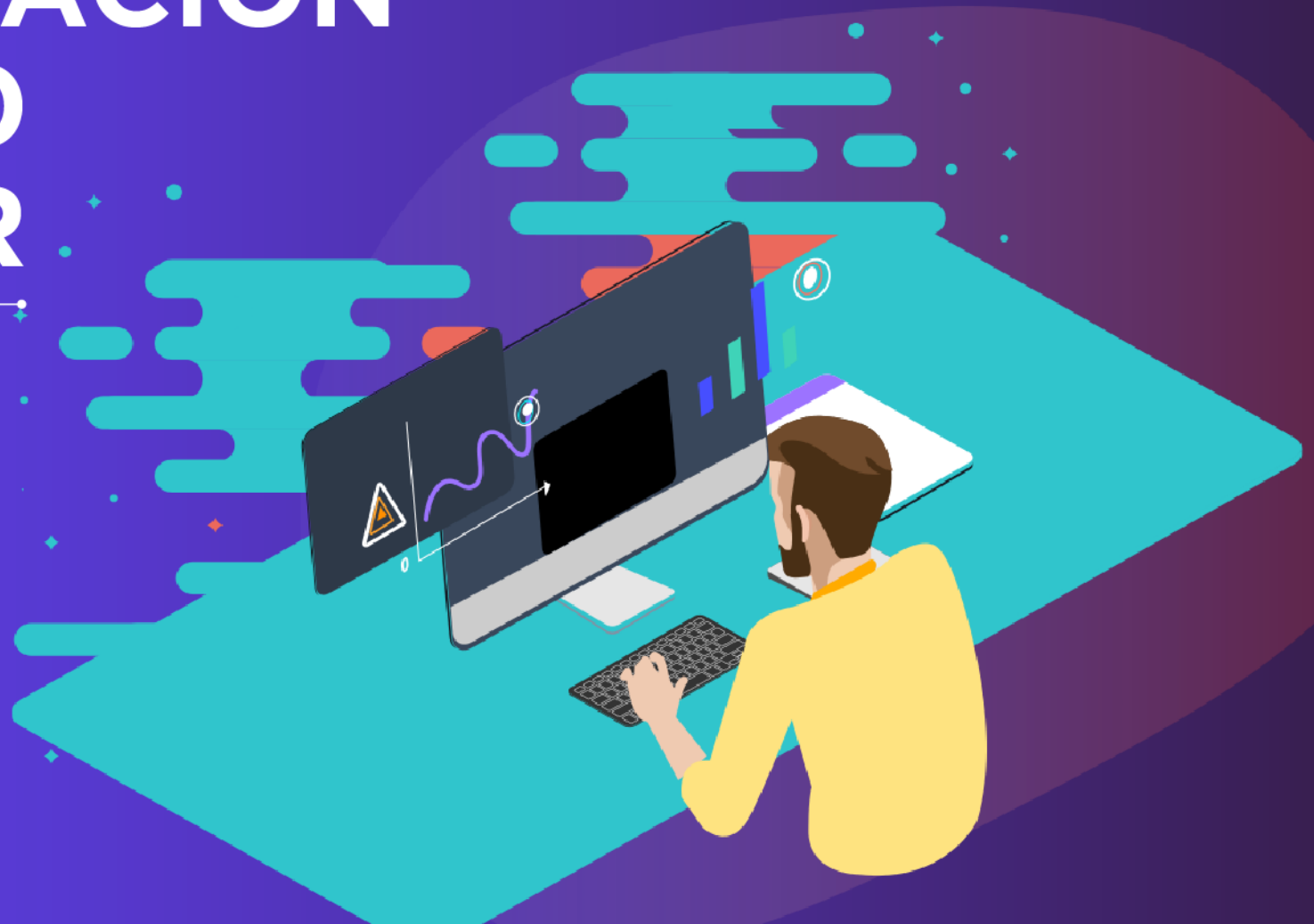
PROGRAMA

.NET ESPECIALIZACIÓN: NET

ESPECIALIZACIÓN ASP.NET 5.0 DEVELOPER



CURSO VIRTUAL

Fundamentos de Programación

Estructuras de control y excepciones

Instructor: Erick Aróstegui
earostegui@galaxy.edu.pe

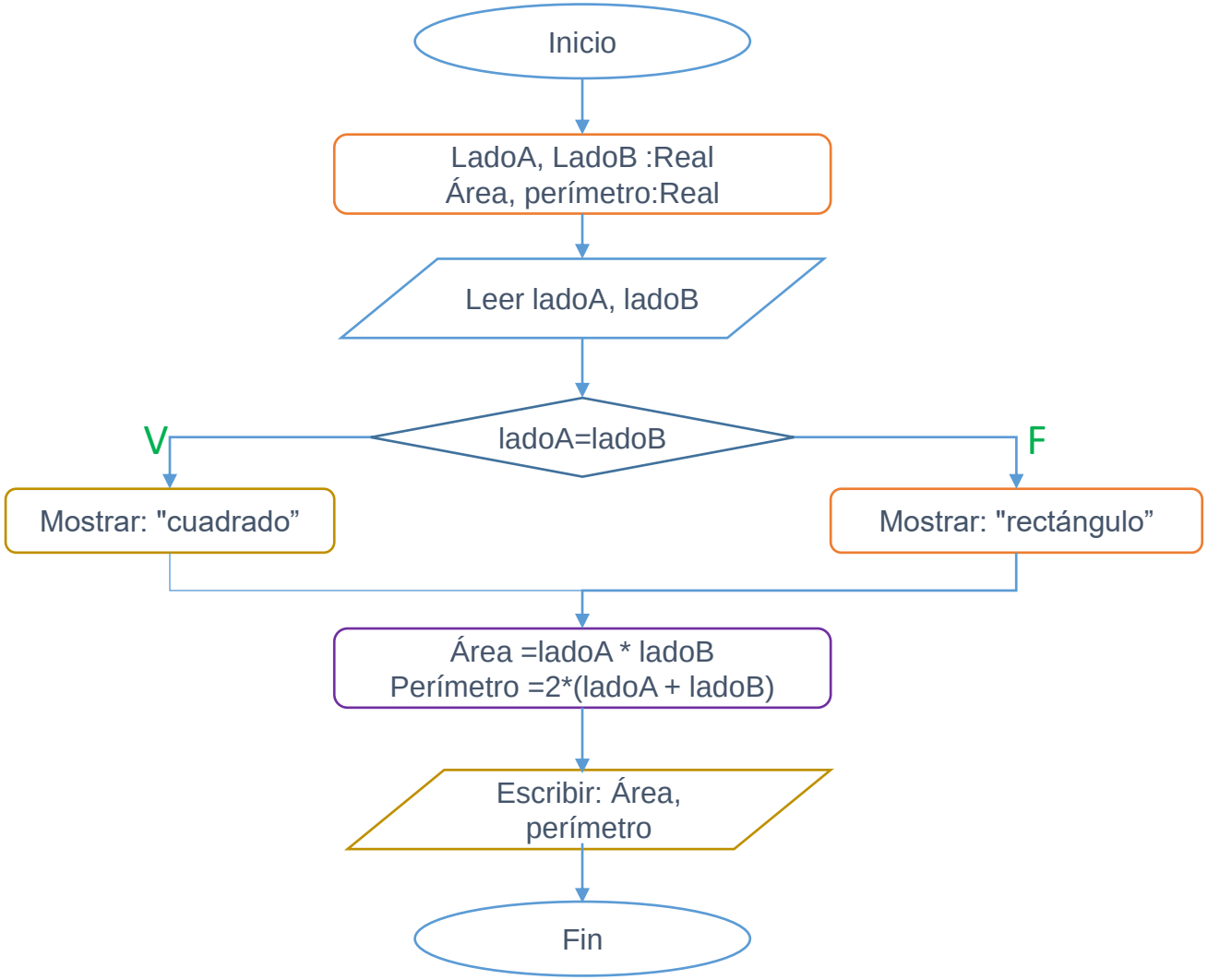
GALAXY TRAINING

TEMAS

Estructuras de control y excepciones

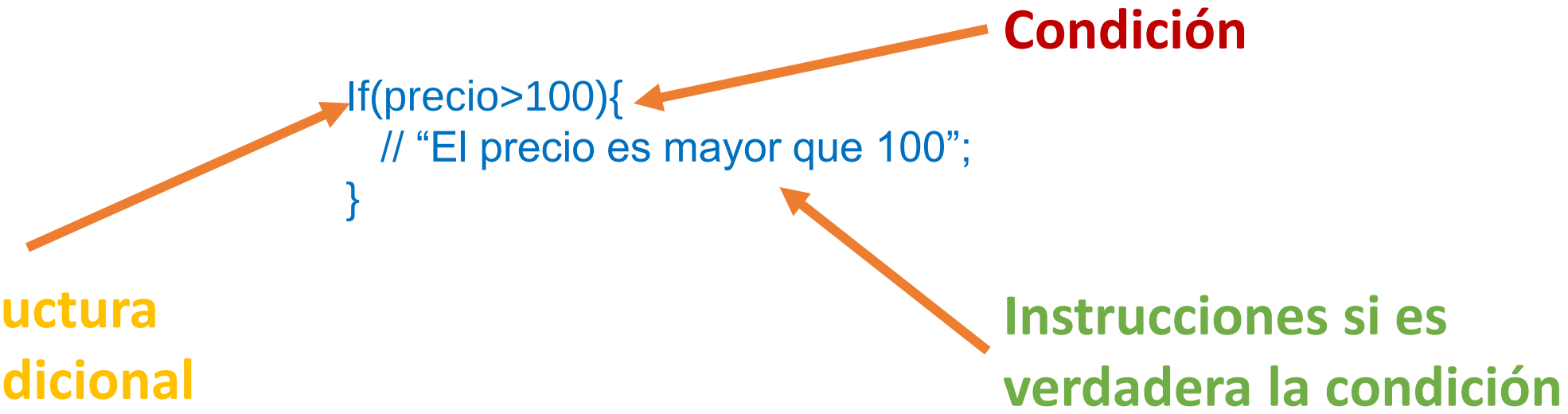
01	Estructuras condicionales
02	Estructuras repetitivas
03	Estructuras selectivas
04	Uso de break, continúe, goto y return
05	Gestión de excepciones estándares y personalizada

Diagrama de Flujo



Estructura If

Evalúa una expresión lógica (condición), si es verdadero ejecuta una determinada instrucción o instrucciones.



Código

```
class EstrucCondi
{
    0 referencias
    static void Main(string[] args)
    {
        int precio = 300;
        if (precio > 100)
        {
            Console.WriteLine("El precio es mayor que 100");
        }
        Console.ReadLine();
    }
}
```

Estructura If-Else If

si la condición del if no es verdadera, ejecute otras instrucciones que estarán dentro de else. Se suele traducir como "Si se cumple esta condición haz esto y sino haz esto".

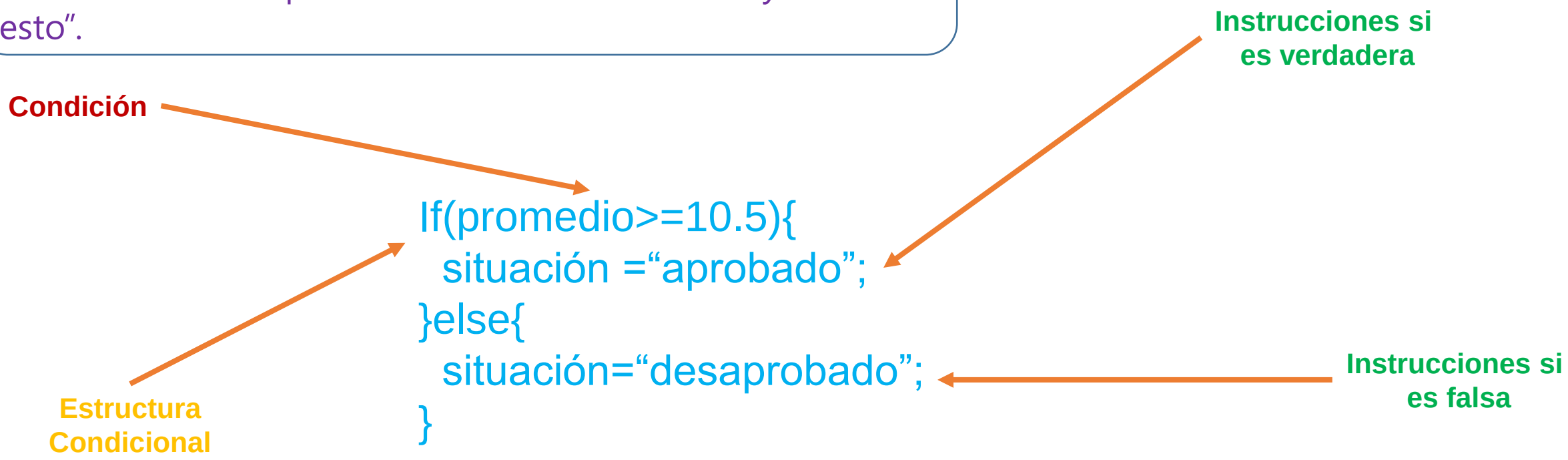
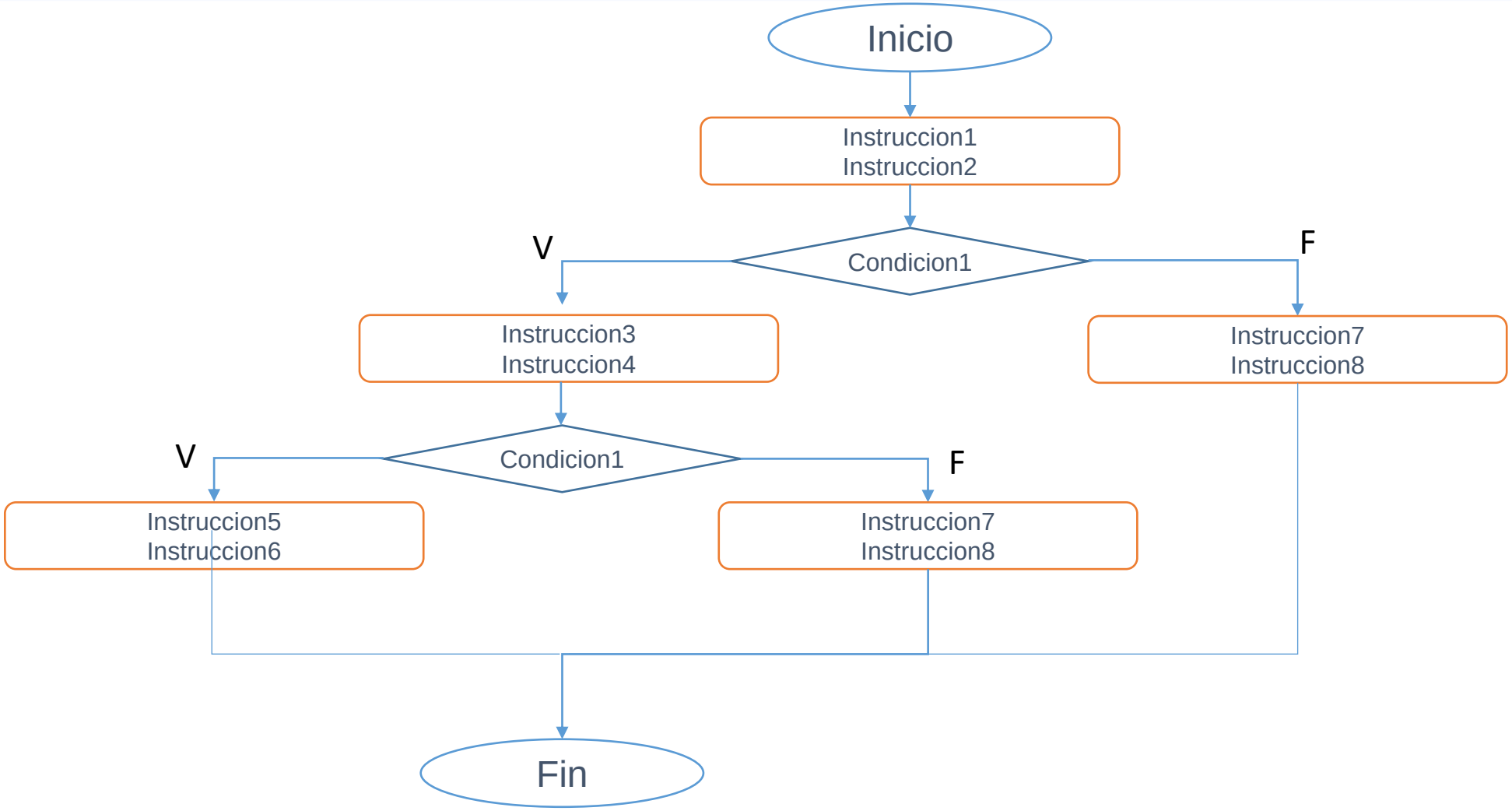


Diagrama de Flujo



Código

```
static void Main()
{
    int a = 1;
    if(a == 1)// si a es igual a 1
    {
        Console.WriteLine("a vale 1");
    }else if (a == 2)// si a es igual a 2
    {
        Console.WriteLine("a vale 2");
    }
    else//sino se cumple ninguna condicion
    {
        Console.WriteLine("a no vale ni 1 ni 2");
    }
    Console.ReadKey();

    /*
```

Estructura If Anidados

•Una sentencia *if* es anidada cuando la sentencia de la rama verdadera o la rama falsa es a su vez una sentencia *if.* Se puede utilizar para implementar decisiones con varias alternativas o multi-alternativas

Código

```
class EstrucCondi
{
    0 referencias
    static void Main()
    {
        Console.WriteLine("Introduce tu edad");
        int edad = Convert.ToInt32(Console.ReadLine());

        if (edad >= 0)
        {

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


        }
        else
        {
            Console.WriteLine("no puede ser negativa");
        }
        Console.ReadKey();
```

Estructuras Repetitiva For

Un bucle for es una estructura de control de repetición, que permite escribir de manera eficiente un bucle que es necesario ejecutar un número determinado de veces.

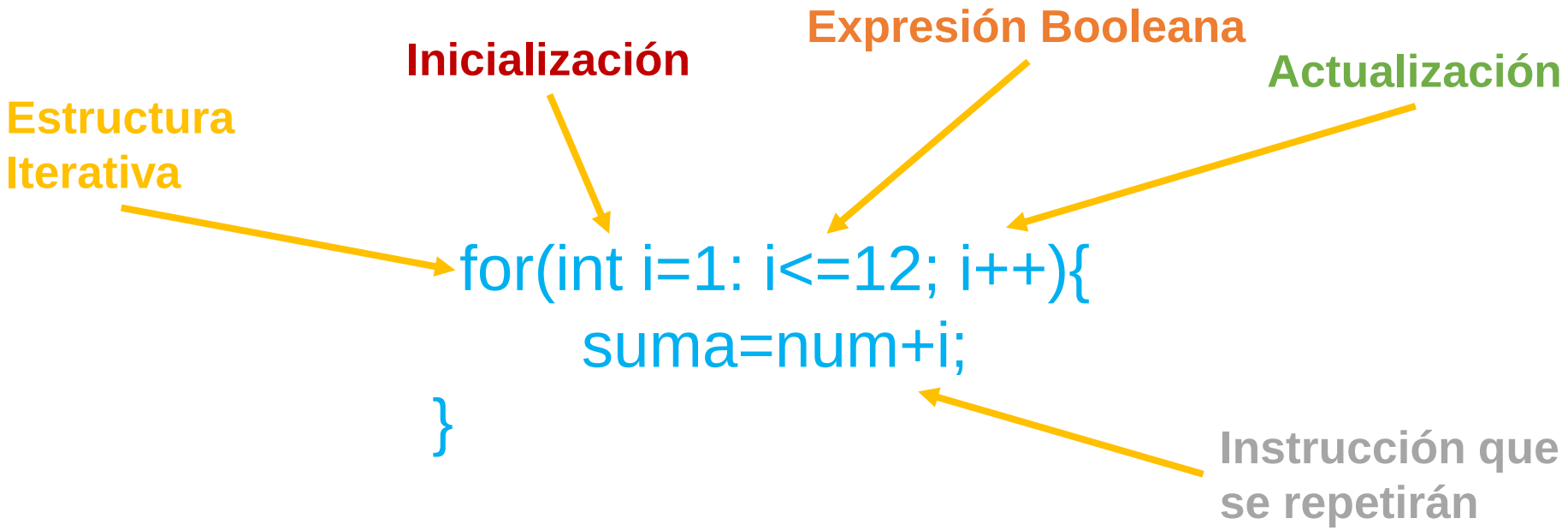
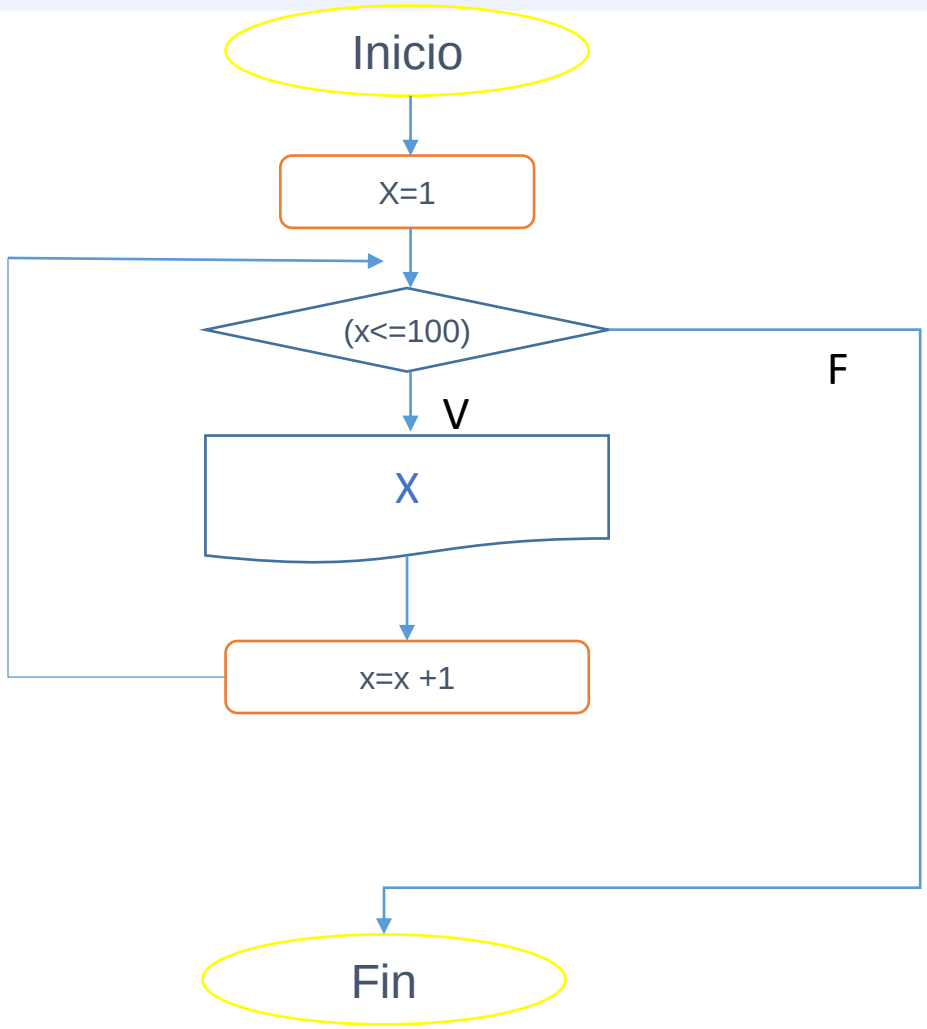


Diagrama de Flujo



Código

Program.cs* — For1 — For1.Program

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace For1
8   {
9       class Program
10      {
11          static void Main(string[] args)
12          {
13              Console.WriteLine("Imprime números del 1 al 10");
14              for(int i = 0; i <= 10; i++)
15              {
16                  Console.WriteLine("i= " + i);
17              }
18              Console.ReadKey();
19          }
20      }
21  }
```

Estructuras Repetitiva While

Una estructura repetitiva que permite repetir una o más instrucciones mientras la condición sea verdadera, cuando la condición es falsa sale del bucle.

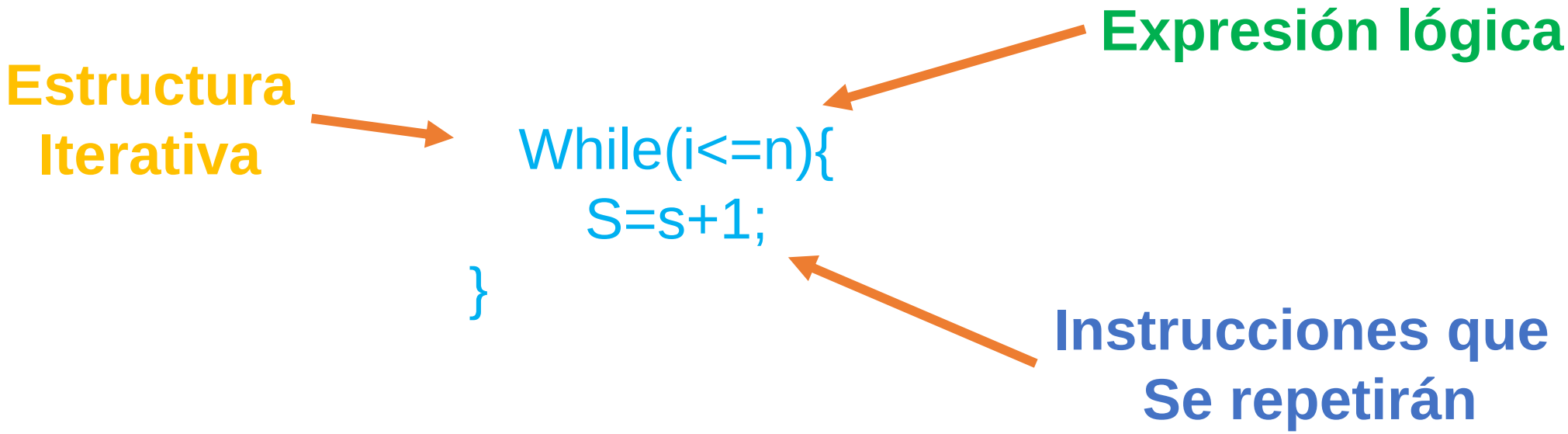
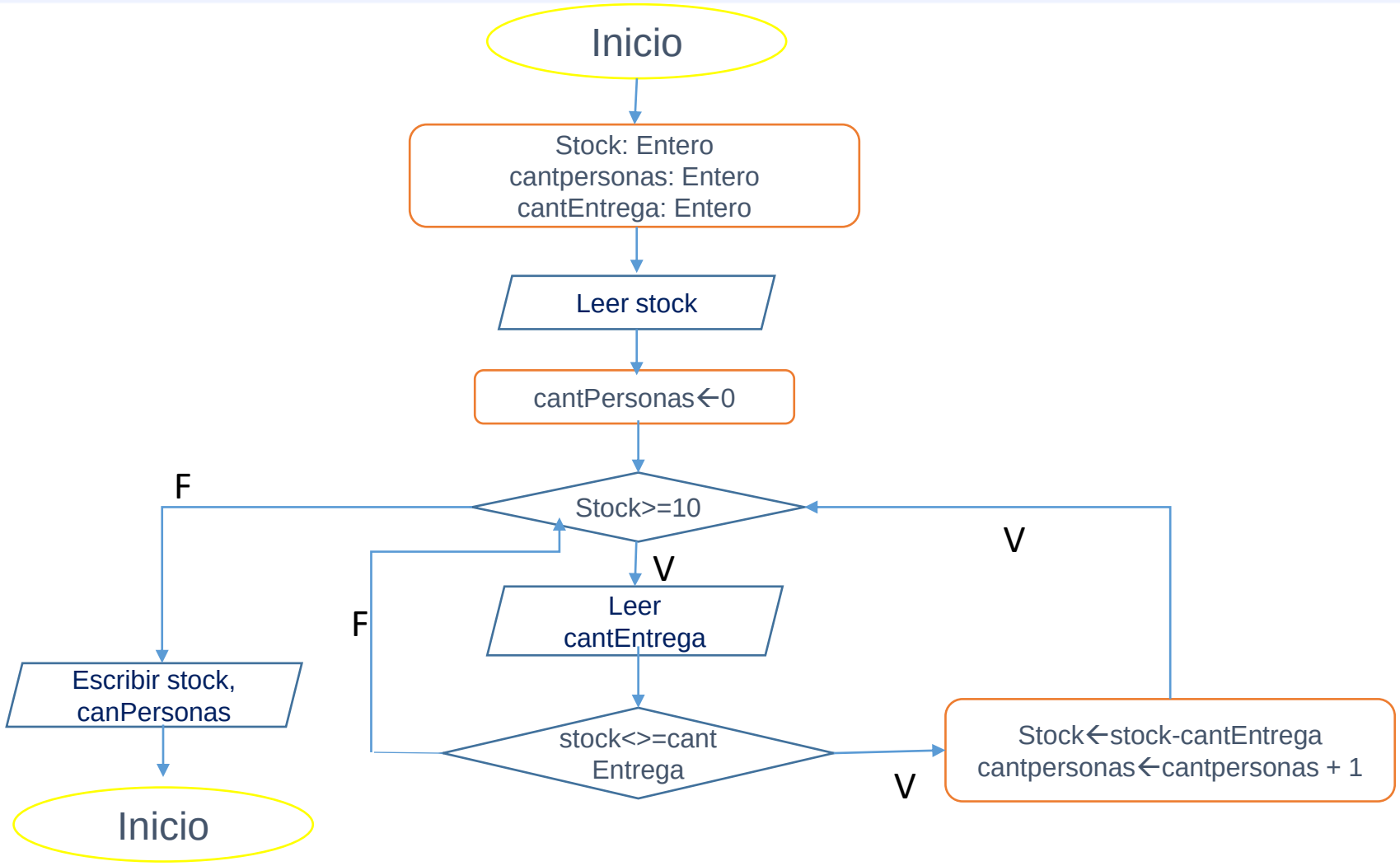


Diagrama de Flujo



Código

```
class EstrucCondi
{
    0 referencias
    static void Main()
    {
        int x, suma, valor, promedio;
        string linea;
        x = 1;
        suma = 0;
        while (x <= 0)
        {
            Console.Write("ingrese su valor " + x + ": ");
            linea = Console.ReadLine();
            valor = int.Parse(linea);
            suma = suma + valor;
            x = x + 1;
        }
        promedio = suma / 8;
        Console.Write("la suma es: ");
        Console.WriteLine(suma);
        Console.Write("el promedio es: ");
        Console.Write(promedio);
        Console.ReadKey();
```

Estructuras Repetitiva Do-While

La estructura do-while es una estructura repetitiva, la cual ejecuta al menos una vez su bloque repetitivo

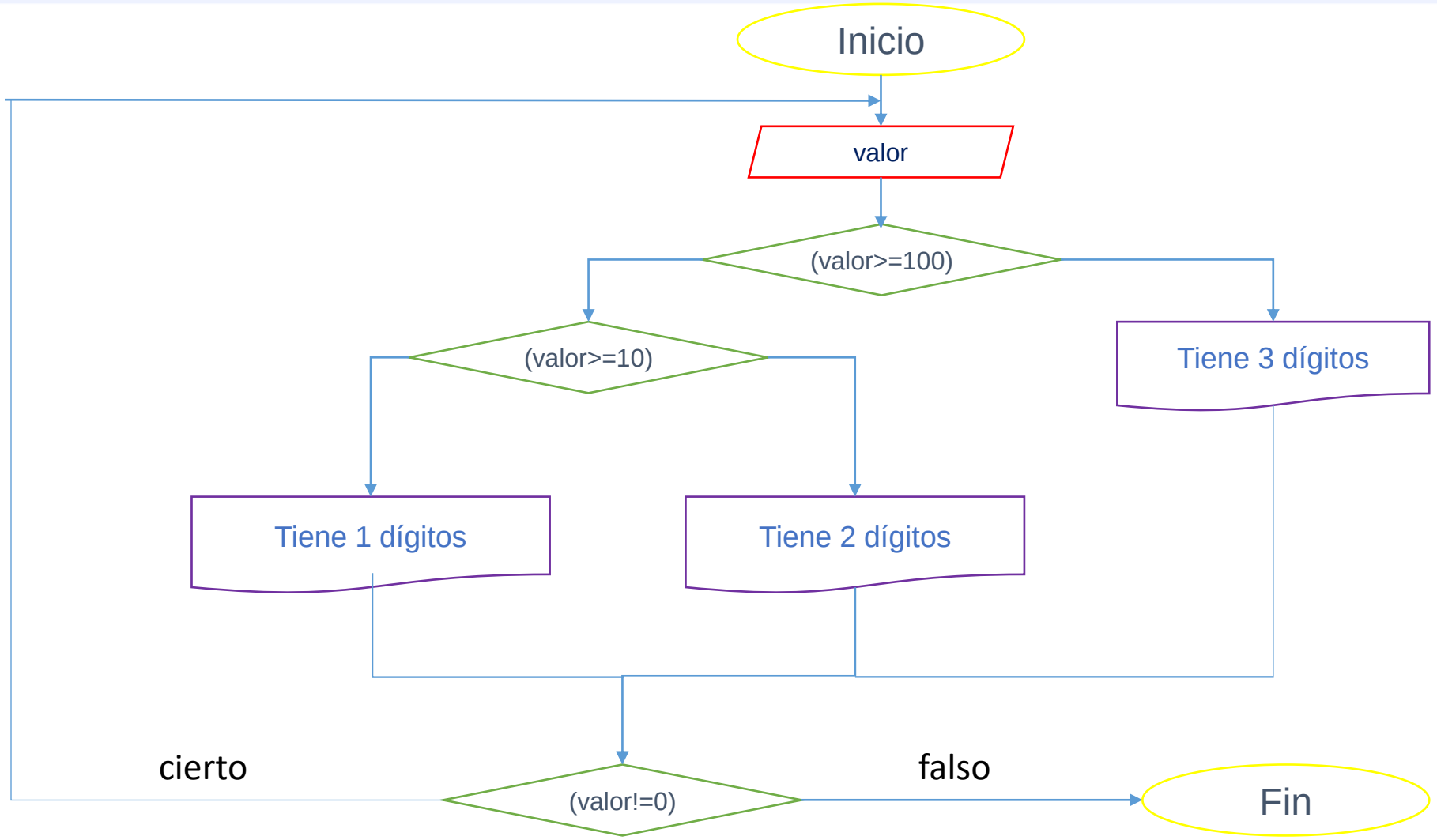
Estructura Iterativa

```
Do{
      nota=nota+1;
}
While(nota<=0|| nota>=20)
```

Instrucciones que se repetirán

Expresión lógica

Diagrama de Flujo



Código

```
static void Main()
{
    int nota;

    do
    {
        Console.Write("Introduzca la nota: ");
        nota = int.Parse(Console.ReadLine());
        if (nota >= 11)
        {
            Console.WriteLine("Aprobaste");
        }
        else
        {
            Console.WriteLine("Desaprobaste");
        }
    } while (nota != 0);
    Console.ReadLine();
```

Estructuras Selectiva Switch

es una declaración de selección que elige una sola *sección de cambio* para ejecutar de una lista de candidatos basada en una coincidencia de patrón con la *expresión de coincidencia*

Estructura selectiva

```
switch(expresión){
  case valor1:
    //instrucciones
    break;
  default:
    //instrucciones
    break;
}
```

Variable a comparar

caso1

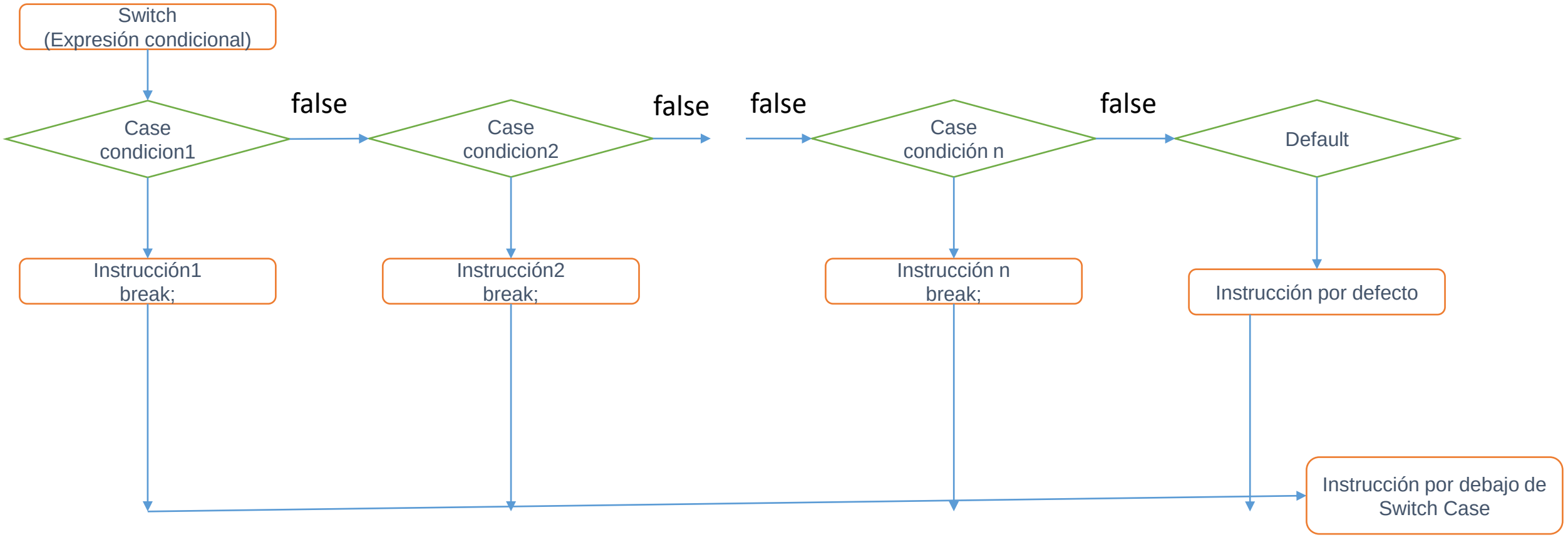
Instrucciones si se cumple el caso 1

Salir del Switch

Caso default

Instrucciones si se cumple el caso default

Diagrama de Flujo



Código

```
public static void Main()
{
    byte numero;
    string dia;
    numero = Convert.ToByte(Console.ReadLine());

    switch (numero)
    {
        case 1:
            dia = "lunes";
            break;
        case 2:
            dia = "martes";
            break;
        case 3:
            dia = "miercoles";
            break;
        case 4:
            dia = "jueves";
            break;
        case 5:
            dia = "viernes";
            break;
        default:
            dia = "no es dia laboral";
            break;

    }
```

Uso de Break

La sentencia break termina la iteración de las sentencias pertenecientes a un ciclo a una sentencia.

detiene un bucle utilizando la palabra break. Detener un bucle significa salirse de él y dejarlo todo como está para continuar con el flujo del programa inmediatamente después del bucle.

Break en for

Este ejemplo la declaración condicional contiene un contador que se supone que cuenta de 1 a 9, la break declaración termina el ciclo después de 3 conteos.

```
0 referencias
static void Main()
{
    for(int i=0; i<10; i++)
    {
        if (i == 4)
        {
            break;
        }
        Console.WriteLine(i);
    }
```

Uso de Switch

En este ejemplo, la instrucción Break solo se usa para salir de la rama actual durante cada iteración del bucle. El propio bucle no se ve afectado por las instancias de Break que pertenecen a la instrucción anidada switch

```
string numero;
int nume;
nume = int.Parse(Console.ReadLine());


switch (nume)
{
    case 1:
        numero = "Uno";
        break;
    case 2:
        numero = "Dos  ";
        break;
    case 3:
        numero = "Tres";
        break;
    case 4:
        numero = "Cuatro";
        break;
    case 5:
        numero = "Cinco";
        break;

    default:
        numero = "error";
        break;
}
```

Uso de Continue

La instrucción Continue transfiere el control a la siguiente iteración (while.do ,for, foreach) que continúe

```
0 referencias
static void Main()
{
    int count = 0;
    for(int i=0; i<10; i++)
    {
        if (i == 5) continue;
        Console.WriteLine(i);
    }
```

Continue en For

Si se usa la instrucción Continue junto con la expresión (i <9), se omiten las instrucciones comprendidas entre Continue y el final de for en las iteraciones donde 1 es menor que 9

```
class ContinueFor
{
    0 referencias
    static void Main()
    {
        for( int i=1; i<=10; i++)
        {
            if (i < 9)
            {
                continue;
            }
            Console.WriteLine(i);
        }
        Console.WriteLine("presiona cualquier tecla para salir");
        Console.ReadKey();
    }
}
```

Continue en While

El código while se usa la instrucción para saltar a la columna siguiente de una matriz

```
int count = 1;
while(count < 10)
{
    count = count + 1;
    if (count < 5)
        continue;
    Console.WriteLine("el valor de 1 es : " + count);
}
```

GOTO

La goto declaración transfiere el control del programa directamente a una declaración etiquetada

Un uso común de Goto es transferir el control de una etiqueta de caja de interruptor específica o la etiqueta predeterminada de una switch declaración

```
static void Main()
{

    Console.WriteLine("Tamaños de cafe: 1=pequeño 2=mediano 3=grande");
    Console.Write("Por favor ingrese su selección: ");
    string s = Console.ReadLine();
    int n = int.Parse(s);
    int costo = 0;
    switch (n)
    {
        case 1:
            costo += 25;
            break;
        case 2:
            costo += 25;
            goto case 1;
        case 3:
            costo += 50;
            goto case 1;
        default:
            Console.WriteLine("Selección invalida.");
            break;
    }
    if (costo != 0)
    {
        Console.WriteLine($"Por favor inserte { costo} centavos.");

    }
    Console.WriteLine("Gracias por hacer negocios .");
    Console.WriteLine(" presiona cualquier tecla para salir.");
    Console.ReadKey();
```

Try-Catch

Cuando se use los bloques try/catch alrededor del código que podría generar una excepción y su código podrá recuperarse de una excepción. Todas las excepciones se derivan del Exception

```
public static void Main()
{
    int n;
    try
    {
        //No inicialice esta variable aquí.
        n = 123;

    }
    catch (Exception e)
    {

    }
    //Error: uso de la variable local 'n' sin asign
    Console.WriteLine();
```

Try-Finally

Mediante el uso de finally, se puede limpiar todos los recursos asignados en un bloque try. Normalmente , las instrucciones del bloque finally se ejecutan cuando el control abandona una instrucción try

```
0 referencias
public static void Main()
{
    int i = 123;
    string s = "Some string";
    object obj = s;

    try
    {
        //    Conversión no válida; obj contiene una cadena, no un tipo numérico.
        i = (int)obj;

        // La siguiente declaración no se ejecuta.
        Console.WriteLine("WriteLine al final del bloque try.");
    }
    finally
    {
        Console.WriteLine("\nExecution del bloque finalmente después de un \n " +
            "El error depende de cómo se desencadena la operación de desenrollado de la excepción.");
        Console.WriteLine("i = {0}", i);
    }
}
```

Try-Catch-Finally

Un uso habitual de catch y finally juntos es obtener y usar recursos de un bloque try, y lidiar con circunstancias excepcionales de un bloque catch y liberar los recursos del bloque finally

```
public static void Main()
{
    try
    {
        int n1, n2;
        int resultado;
        n1= Convert.ToInt32(Console.ReadLine());
        n2 = Convert.ToInt32(Console.ReadLine());
        resultado = n1 / n2;
    }
    catch (FormatException)
    {
        Console.WriteLine("Debe escribir dos números enteros");
    }
    catch (DivideByZeroException)
    {
        Console.WriteLine("No es permitida la division por cero");
    }
    finally
    {
        Console.WriteLine("bloque final");
    }
```

PREPÁRATE
PARA SER EL
MEJOR

+ ENTREMIENTO
EXPERIENCIA

BIENVENIDOS.

GRACIAS
POR TU PARTICIPACIÓN

Por favor, bríndanos tus comentarios
y sugerencias para mejorar nuestros servicios.

GALAXY
TRAINING


GALAXY
TRAINING