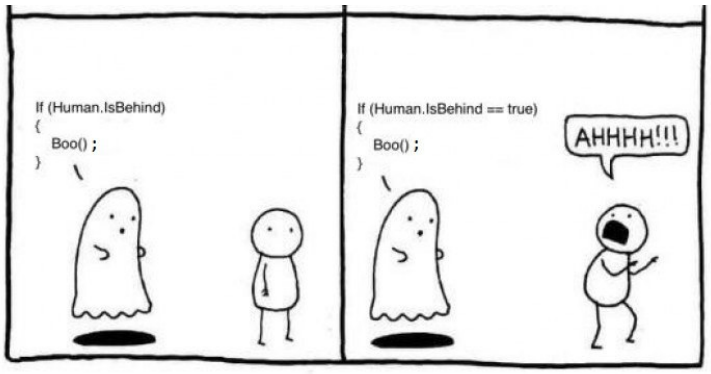
Cp 3 - Condicionales
Curso 2024-2025



1. Divisibles

Implemente un programa que reciba dos enteros y determine si el primero es divisible por el segundo.

Respuesta:

```
Console.WriteLine("Introduce el dividendo:");
int a = int.Parse(Console.ReadLine() ?? string.Empty);
Console.WriteLine("Introduce el divisor:");
int b = int.Parse(Console.ReadLine() ?? string.Empty);

if (b != 0 && a % b == 0)
{
    Console.WriteLine($"{a} es divisible por {b}");
}
else
{
    Console.WriteLine($"{a} no es divisible por {b}");
}
```

2. Carnet, de nuevo

Implemente un programa que le pida al usuario su número de identidad y determine su sexo. Note que el sexo puede determinarse por el penúltimo dígito del número de identidad (par masculino, impar femenino).

Respuesta:

```
Console.WriteLine("Ingrese su número de identidad:");
long idNumber = long.Parse(Console.ReadLine() ?? string.Empty);
long penultimateDigit = idNumber / 10 % 10;
Console.WriteLine(penultimateDigit % 2 == 0 ? "Masculino" : "Femenino");
```

3. Valor absoluto

Implemente un programa que reciba un número entero x de la consola y calcule su valor absoluto. El valor absoluto de un número x se define como el número sin su signo, es decir, la distancia de x al origen en la recta numérica. No utilice Math.Abs.

La función del valor absoluto $|x|$ se define de la siguiente manera:

$$|x| = \begin{cases} x & \text{si } x \geq 0 \\ -x & \text{si } x < 0 \end{cases}$$

Respuesta:

```
Console.WriteLine("Introduce un número entero:");
int x = int.Parse(Console.ReadLine() ?? string.Empty);
int abs = x >= 0 ? x : -x;
Console.WriteLine($"El valor absoluto de {x} es {abs}");
```

4. Forman una fecha

Lea tres números enteros de la consola que representarán día, mes y año respectivamente. Si estos valores pueden formar una fecha, entonces muéstrela en la consola con el formato `día/mes/año`, si no imprima el No es fecha.

Respuesta:

Analicemos qué deben cumplir estos enteros para formar una fecha:

1. El año debe ser positivo.
2. El mes debe estar entre 1 y 12.
3. El día debe estar dentro del rango de días válidos para el mes y el año dados, notemos que tenemos que tener en cuenta el año ya que 29/2/2001 no es una fecha válida mientras que 29/2/2004 sí lo es.

Podemos definir el siguiente método para validar una fecha

```
public static bool ValidateDate(int day, int month, int year)
{
    return year > 0 &&
           month > 0 && month <= 12 &&
           day > 0 && day <= CalculateLastDayInMonth(month, year);
}
```

Notemos que, con una implementación correcta de *CalculateLastDayInMonth*, tendríamos el problema resuelto. Podemos implementar este método así:

```
private static int CalculateLastDayInMonth(int month, int year)
{
    switch (month)
    {
        // Los meses con 30 días
        case 4 or 6 or 9 or 11:
            return 30;
        // Febrero
        case 2:
            return IsLeapYear(year) ? 29 : 28;
        // Los meses con 31 días
```

```
        default:
            return 31;
    }
}
```

Ahora solo nos faltaría implementar *IsLeapYear* que determina si un año es bisiesto o no.

Antes de 1582, los años bisiestos se determinaban según el calendario juliano, donde un año era bisiesto si era divisible por 4. Este sistema acumulaba un pequeño desfase con el tiempo. Con la introducción del calendario gregoriano en 1582, se ajustaron las reglas: un año es bisiesto si es divisible por 4 y no por 100, a menos que también sea divisible por 400. Esto ayuda a mantener el calendario alineado con el ciclo solar y las estaciones del año. Por ejemplo, 1600 y 2000 son años bisiestos, pero 1700, 1800 y 1900 no lo son.

Podemos implementar *IsLeapYear* de la siguiente manera

```
private static bool IsLeapYear(int year)
{
    return year <= 1582
        ? year % 4 == 0
        : (year % 4 == 0 && year % 100 != 0) || year % 400 == 0;
}
```

Finalmente, el código completo nos quedaría:

```
public static class Program
{
    public static void Main()
    {
        Console.WriteLine("Introduce el día:");
        int day = int.Parse(Console.ReadLine() ?? string.Empty);
        Console.WriteLine("Introduce el mes:");
        int month = int.Parse(Console.ReadLine() ?? string.Empty);
        Console.WriteLine("Introduce el año:");
        int year = int.Parse(Console.ReadLine() ?? string.Empty);

        Console.WriteLine(
            ValidateDate(day, month, year)
                ? $"{day}/{month}/{year}"
                : "No es fecha");
    }

    public static bool ValidateDate(int day, int month, int year)
    {
        return year > 0 &&
               month > 0 && month <= 12 &&
               day > 0 && day <= CalculateLastDayInMonth(month, year);
    }

    private static int CalculateLastDayInMonth(int month, int year)
    {
        switch (month)
        {
            // Los meses con 30 días
            case 4 or 6 or 9 or 11:
                return 30;
            // Febrero
            case 2:
                return IsLeapYear(year) ? 29 : 28;
            // Los meses con 31 días
            default:
                return 31;
        }
    }
```

```
    private static bool IsLeapYear(int year)
    {
        return year <= 1582
            ? year % 4 == 0
            : (year % 4 == 0 && year % 100 != 0) || year % 400 == 0;
    }
}
```

5. Triángulos

Implemente un programa que pida al usuario tres números enteros y determine qué tipo de triángulo forman. Debe mostrar en la consola lo siguiente:

- 0 si no pueden ser lados de ningún triángulo
- 1 si es un triángulo escaleno
- 2 si es un triángulo isóceles
- 3 si es un triángulo equilátero

Respuesta:

```
public static class Program
{
    public static void Main()
    {
        Console.WriteLine("Introduce el primer lado:");
        int side1 = int.Parse(Console.ReadLine() ?? string.Empty);
        Console.WriteLine("Introduce el segundo lado:");
        int side2 = int.Parse(Console.ReadLine() ?? string.Empty);
        Console.WriteLine("Introduce el tercer lado:");
        int side3 = int.Parse(Console.ReadLine() ?? string.Empty);

        TypeOfTriangle typeOfTriangle = GetTypeOfTriangle(side1, side2, side3);
        int response = (int)typeOfTriangle;

        Console.WriteLine($"El tipo de triángulo es: {response}");
    }

    private enum TypeOfTriangle
    {
        NotATriangle = 0,
        Scalene = 1,
        Isosceles = 2,
        Equilateral = 3
    }

    private static TypeOfTriangle GetTypeOfTriangle(int side1, int side2, int side3)
    {
        if (side1 + side2 <= side3 || side1 + side3 <= side2 || side2 + side3 <= side1)
            return TypeOfTriangle.NotATriangle;

        if (side1 == side2 && side2 == side3)
            return TypeOfTriangle.Equilateral;

        if (side1 == side2 || side2 == side3 || side1 == side3)
            return TypeOfTriangle.Isosceles;

        return TypeOfTriangle.Scalene;
```

```
    }
}
```

6. El día después

Implemente un programa que reciba una fecha e imprima la fecha correspondiente al día siguiente.

Respuesta:

Habrás notado que en el ejercicio *Forman una fecha* definimos el método CalculateLastDayInMonth que, dado el mes y el año calcula el último día del mes correspondiente, en este ejercicio reutilizaremos este método.

```
public static (int Day, int Month, int Year) CalculateNextDay(int day, int month, int year)
{
    if (!ValidateDate(day, month, year))
        throw new Exception("Invalid date");

    return day == CalculateLastDayInMonth(month, year)
        ? month == 12
            ? (1, 1, year + 1) // si es 31 de diciembre
            : (1, month + 1, year) // si es el último dia de un mes que no es diciembre
        : (day + 1, month, year);
}
```

7. Día de la semana

Implemente un programa que dada una fecha, muestre qué día de la semana cae.

Respuesta:

Para resolver el problema, primero calculamos la cantidad de días que han pasado desde el 1/1/1 hasta la fecha dada, luego usamos la operación resto de la división entre 7 para encontrar el día de la semana, ya que la semana tiene 7 días.

```
public enum DayOfWeek
{
    Sunday = 0,
    Monday = 1,
    Tuesday = 2,
    Wednesday = 3,
    Thursday = 4,
    Friday = 5,
    Saturday = 6,
}

public static DayOfWeek CalculateDayOfWeek(int day, int month, int year)
{
    if (!ValidateDate(day, month, year))
        throw new Exception("Invalid date");

    int daysSinceStartOfEra = CalculateDaysSinceStartOfEra(day, month, year);

    return (DayOfWeek)(daysSinceStartOfEra % 7);
}
```

Aún falta implementar el método *CalculateDaysSinceStartOfEra*. Dada una fecha en el formato $d/m/y$, podemos calcular los días desde el 1/1/1 como la suma de:

1. Los días desde el $1/1/1$ hasta el $31/12$ del año anterior, es decir, $(y-1)$.
2. Los días desde el $1/1/y$, hasta el último día del mes anterior, es decir, $(m-1)$.
3. Los días transcurridos desde el $1/m/y$ hasta $d/m/y$, o sea d.

A continuación el código en C#:

```
public static int CalculateDaysSinceStartOfEra(int day, int month, int year)
{
    if (!ValidateDate(day, month, year))
        throw new Exception("Invalid date");

    int leapYears = (year - 1) / 4 - (year - 1) / 100 + (year - 1) / 400;

    int daysSinceStartOfEra = leapYears * 366 + (year - 1 - leapYears) * 365;

    if (month > 1)
        daysSinceStartOfEra += 31;
    if (month > 2)
        daysSinceStartOfEra += IsLeapYear(year) ? 29 : 28;
    if (month > 3)
        daysSinceStartOfEra += 31;
    if (month > 4)
        daysSinceStartOfEra += 30;
    if (month > 5)
        daysSinceStartOfEra += 31;
    if (month > 6)
        daysSinceStartOfEra += 30;
    if (month > 7)
        daysSinceStartOfEra += 31;
    if (month > 8)
        daysSinceStartOfEra += 31;
    if (month > 9)
        daysSinceStartOfEra += 30;
    if (month > 10)
        daysSinceStartOfEra += 31;
    if (month > 11)
        daysSinceStartOfEra += 30;

    daysSinceStartOfEra += day;

    return daysSinceStartOfEra;
}
```

Este código puede parecer engorroso porque necesitamos redefinir la cantidad de días en cada mes, algo que ya habíamos hecho en el método CalculateLastDayInMonth. Para simplificarlo, podemos usar un ciclo y reutilizar el método definido anteriormente.

```
public static int CalculateDaysSinceStartOfEra(int day, int month, int year)
{
    if (!ValidateDate(day, month, year))
        throw new Exception("Invalid date");

    int leapYears = (year - 1) / 4 - (year - 1) / 100 + (year - 1) / 400;

    int daysSinceStartOfEra = leapYears * 366 + (year - 1 - leapYears) * 365;

    for (int i = 1; i < month; i++)
    {
        daysSinceStartOfEra += CalculateLastDayInMonth(i, year);
    }

    daysSinceStartOfEra += day;
```

```
    return daysSinceStartOfEra;
}
```

Nota: Para facilitar los cálculos asumimos que el calendario gregoriano estuvo en uso desde el año 1, lo cual no es cierto, por tanto los cálculos para fechas anteriores a 1582 no se realizan correctamente.

El calendario juliano, utilizado desde el 46 a.C., tenía un error que acumuló un desfase de 10 días para el siglo XVI. Para corregirlo, el papa Gregorio XIII introdujo el calendario gregoriano, ajustando la duración del año y cambiando la regla de los años bisiestos, además, se eliminaron 10 días del calendario en octubre de 1582.

8. Dos fechas

Implemente un programa que reciba dos fechas (tres enteros por cada fecha) y calcule cuántos días hay entre ellas.

Hint:

Utiliza la cantidad de días entre cada fecha y el 1/1/1.

9. Factorial

Implemente un programa que reciba un número entero no negativo n de la consola y calcule el factorial de ese número. No utilice ciclos.

El factorial de un número n (denotado como $n!$) se define como el producto de todos los números enteros positivos desde 1 hasta n, o lo que es lo mismo:

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n \cdot (n-1)! & \text{si } n > 0 \end{cases}$$

Respuesta:

Al igual que podemos llamar otros métodos, un método puede llamarse a sí mismo. Esto es útil en situaciones como el cálculo del factorial, donde el problema puede definirse en términos de sí mismo.

En CalculateFactorial(int n), el método se llama a sí mismo con un número menor. Sin embargo, es crucial tener un **caso base** para detener las llamadas. En este caso, el caso base es cuando n es 0, devolviendo 1. Sin este caso base, el método se llamaría indefinidamente, causando un bucle infinito.

```
static long CalculateFactorial(int n)
{
    if (n == 0)
        return 1;

    return n * CalculateFactorial(n - 1);
}
```

10. Imprimiendo números

Implemente un método que reciba un entero n y muestre en la consola en orden descendente, todos los números enteros entre n y 0. No utilice ciclos.

Respuesta:

Para imprimir los números de n hasta 0, podemos plantear el problema de la siguiente manera: imprimir el número n y luego imprimir los números de $n-1$ hasta 0. El caso de parada podríamos definirlo como si $n < 0$, no hacer nada (return).

```
static void PrintNumbersDesc(int n)
{
    if (n < 0) return;

    Console.WriteLine(n);

    PrintNumbersDesc(n - 1);
}
```

11. Invierte el orden

Implemente un método que reciba un entero n y muestre en la consola, en orden ascendente, todos los números enteros entre 0 y n. No utilice ciclos.

Respuesta:

Similar al ejercicio anterior, podemos plantear el problema de imprimir los números de 0 hasta n como: imprimir los números de 0 hasta $n-1$ y luego imprimir el número n.

```
static void PrintNumbersAsc(int n)
{
    if (n < 0) return;

    PrintNumbersAsc(n - 1);

    Console.WriteLine(n);
}
```

12. Sumando horas

Implemente un programa que dada una hora en el formato de 24 horas y un intervalo de tiempo representado en hora y minutos, calcule la nueva hora tras sumar el intervalo a la hora inicial. Por ejemplo si se suma 0 horas y 70 minutos a las 3:30, se obtendría 4:40.

13. Avión

Dada la hora de salida de un avión y su hora de llegada, determine el tiempo de vuelo. La hora estará dada por dos enteros, hora y minutos. El máximo del tiempo del viaje es 24 horas y no hay cambio de zonas horarias.

14. Punto interior

Un punto esta formado por dos enteros (coordenadas x,y). Implemente un programa que reciba cuatro puntos de forma que los tres primeros formen un triángulo. Determine si el último punto es o no interior del triángulo.

15. Mayor

Implemente un programa que lea tres enteros de la consola e imprima el mayor.

16. Calculadora

Implemente un programa que lea de la consola dos enteros y un operador (+, -, /, *) y realice la operación correspondiente entre ellos e imprima el resultado en consola.