Ejercicios de Expresiones y Tipos Elementales

Velocidad de escritura

Escriba un programa que le pida al usuario que escriba su nombre en la consola y calcule su velocidad de escritura *(caracteres por segundo).*

> Investigue cómo utilizar `Environment.TickCount` para medir la cantidad de milisegundos transcurridos.

Ordenación con solo Min y Max

Lea tres números enteros de la consola e imprima su ordenación ascendente utilizando solamente los métodos `Math.Min` y `Math.Max`.

Área sombreada

Sean las circunferencias `C1` y `C2` de radio `r`. Lea de la consola el radio `r` (puede ser un número real cualquiera, no solo un entero) y calcule el área sombreada.

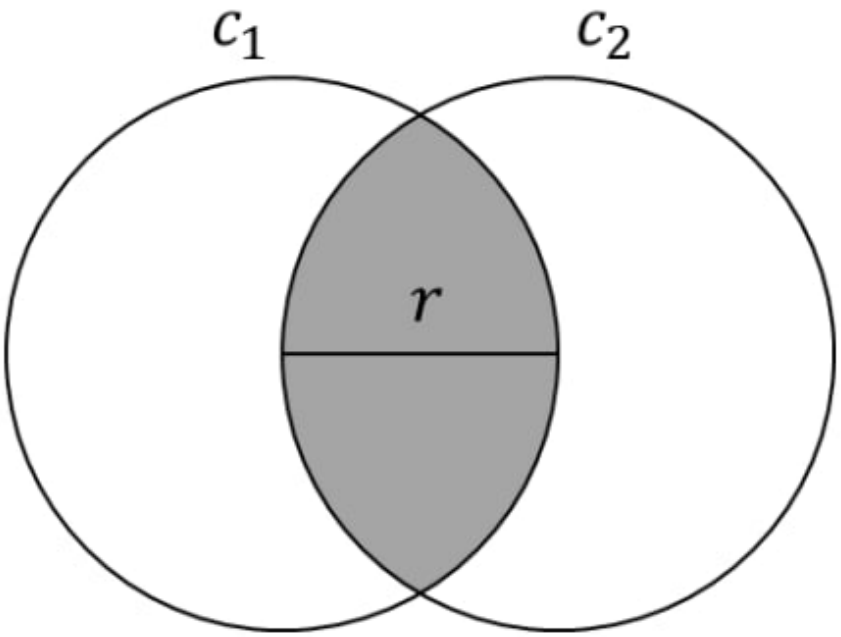

Figure 1: área sombreada

Solución de la ecuación de segundo grado

Lea de la consola los coeficientes **(números reales)** de una ecuación cuadrática y, asumiendo que tiene solución, halle sus soluciones.

$$ax^2 + bx + c = 0$$

Fecha de nacimiento

Lea de la consola el número de identidad de una persona e imprima su fecha de nacimiento. Utilizar operaciones aritméticas (resto y división). NO USAR `string`.

Ejercicios de Condicionales y Ciclos

Determinar sexo

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, en caso de ser par es masculino, femenino en caso contrario.

Mayor, menor y promedio de forma perezosa

Implemente un programa que lea una secuencia de números de la consola (uno por línea) hasta que se escriba una línea en blanco y de estos imprimir:

- El mayor
- El menor
- Su promedio

Fecha válida

Implemente un programa que lea de la consola tres números y compruebe si forman una fecha. En caso de serlo, imprima el día siguiente a la misma con el formato **día/mes/año**.

Tipo de triángulo

Implemente un método que reciba tres números enteros y diga qué tipo de triángulo forman. El método debe devolver `0` si los enteros son lo lados de ningún triángulo, `1` si es un triángulo escaleno, `2` si es isósceles y `3` si es equilátero.

Cree y utilice un `enum` para mejorar la semántica de este método.

Cantidad de dígitos

Implemente un método que reciba un número entero y halle su cantidad de dígitos (no usar la clase `string`)

Representación binaria

Implemente un método que reciba un número entero y devuelva su representación binaria.

Máximo común divisor

Implemente un método que reciba dos números enteros y halle el máximo común divisor entre ellos.

Secuencia de Collatz

Escriba un programa que lea un número entero `n` de la consola e imprima la secuencia de *Collatz* para `n`. La secuencia de *Collatz* se define como:

$$S_0 = nS_{n+1} = \begin{cases} 3S_n + 1 \ si \ S_n \ impar \\ \\ \frac{S_n}{2} \ si \ S_n \ par \end{cases}$$

Dicha secuencia termina cuando se alcanza el número `1` (lo cual siempre ocurre, aunque no se ha podido demostrar aún).

Ejemplo: Para `17` tenemos (`->` división entre `2`, `=>` multiplicación por `3` y adición de `1`)

```
17 => 52 -> 26 -> 13 => 40 -> 20 -> 10 -> 5 => 16 -> 8 -> 4 -> 2 -> 1
```

Días entre dos fechas

Implemente un método que reciba como parámetro dos fechas (tres enteros por cada fecha) y calcule cuántos días hay entre ellas **(no usar ciclos)**.

Factorial

Calcular el factorial de un número.

Ej: `5! = 5 x 4 x 3 x 2 x 1 = 120`

Potencia de un número (sucesivas multiplicaciones)

Calcular la potencia de un número como sucesivas multiplicaciones.

Número perfecto

Determinar si un número es perfecto. Un número es perfecto si la suma de sus divisores propios es igual a él.

Ej: `28 = 1 + 2 + 4 + 7 + 14`

Sucesión de Fibonacci

Hallar el n-ésimo término de la sucesión de Fibonacci:

$$F_0 = 1F_1 = 1F_n = F_{n-1} + F_{n-2}\ para\ n > 1$$

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55$$

Máximo común divisor por el algoritmo de Euclides

Hallar el Máximo Común Divisor entre dos enteros utilizando el algoritmo de Euclides basado en el resto. `MCD(a, b) = MCD(b, r)` donde `r = a % b`.

Descomponer un número en factores primos

Dado un número `n`. Hallar su descomposición en factores primos. Dicha descomposición consiste en el producto de potencias de factores primos tal que se obtenga el número. Ej: $1960 = 2^3 * 5^1 * 7^2$.

Nota: Esta descomposición es única para cada número.

Adicional: Determinar el número primo más cercano a `n`.

Ciclos y Arrays

Segundo mayor

Implemente un método que devuelva el segundo mayor elemento del array.

Es no decreciente?

Implemente un método que determine si el array está ordenado en orden no decreciente.

Evaluación de polinomios

Implemente el método `int Evalua(int[] C, int t)`, que devuelve la evaluación de `p(t)`, donde `p` es un polinomio que tiene como coeficientes los elementos de `C`.

Ejemplo:

$$p(x) = x^3 - 5x + 3C = 1, 0, -5, 3t = 2p(t) = 1$$

Número binario a decimal

Convertir un número de binario a decimal. (El número está representado por un `string` compuesto de `0s` y `1s`).

Rotar arrays

Implemente el método `void Rota(int[] N, int k)` que rote circularmente los elementos de `N, k` posiciones a la derecha.

Es palíndromo?

Implemente un método determine si `s` es palíndromo (se lee igual al derecho que al revés). Ejemplos: `ana`, `anitalavalatina`, `zz`.

Menos sufijo para ser palíndromo

Implemente un método que compute el menor `string t` tal que `s + t` es palíndromo.

Criba de Eratóstenes

Implemente la **Criba de Eratóstenes**. Ver el libro *Empezar a programar. Un enfoque multiparadigma con C#* (p. 104).

Substring

Dados los string `s` y `x`, implemente un método que diga si `x` es substring de `s`. Ejemplo: `""` es substring de toda cadena `"a"` es substring de `"casa"`, `"asap"` no es substring de `"casa"`.

Mezcla ordenada

Implemente el método `int[] MezclaOrdenada(int[] a, int[] b)` que recibe dos array ordenados y devuelve un array ordenado con los elementos de `a` y `b`. Ejemplo: `a = { 1, 2, 4, 7, 10 }`, `b = { 4, 4, 8, 12 }` la mezcla sería `{ 1, 2, 4, 4, 4, 7, 8, 10, 12 }`.

Moda

Dado un array de números enteros implemente un método para calcular su **moda**.

Mediana

Dado un array de números enteros implemente un método para calcular su **mediana**.

n-esima fila del triángulo de Pascal

Dado un número `n`, implemente un método que devuelva la n-ésima fila del triángulo de Pascal.

Clases

Conjunto de enteros

Implemente una clase para representar el comportamiento de un conjunto de enteros. Un conjunto es un colección de elementos no repetidos.

Se deben implementar métodos para realizar las operaciones de:

- Obtener su cardinalidad.
- Realizar la intersección de dos conjuntos.
- Realizar la unión de dos conjuntos.
- Realizar la diferencia de dos conjuntos.
- Consultar si un elemento está en el conjunto.
- Consultar si un conjunto está contenido en otro.
- Consultar si dos conjuntos son iguales.

Fracción

Implemente una clase para representar el comportamiento de una fracción. Una fracción esta conformada por dos enteros denominados numerador y denominador, donde el denominador no puede ser cero. Dos fracciones son iguales si la razón entre su denominador y su numerador es igual. Por ejemplo: `1/2` y `2/4` son iguales. `1/2` y `3/4` no son iguales. Pero `1/2` es su representacion irreducible.

Se deben implementar métodos para realizar las operaciones de:

- Sumar dos fracciones.
- Restar dos fracciones.
- Multiplicar dos fracciones.
- Dividir dos fracciones.
- Consultar si dos fracciones son iguales.

Número complejo

Implemente una clase para representar el comportamiento de un número complejo. Un número complejo esta conformado por dos números, su parte real y su parte imaginaria.

Se deben implementar métodos para realizar las operaciones de:

- Sumar dos números complejos.
- Restar dos números complejos.
- Multiplicar dos números complejos.
- Dividir dos números complejos.

Polinomio

Implemente una clase para representar el comportamiento de un polinomio. Un polinomio es una colección de coeficientes enteros, donde cada coeficiente se corresponde con una potencia de una variable. Por ejemplo: `2x^3 + 3x^2 + 1x + 1` es un polinomio.

Se deben implementar métodos para realizar las operaciones de:

- Obtener el grado del polinomio.
- Obtener el coeficiente de una potencia.
- Sumar dos polinomios.
- Multiplicar dos polinomios.

Ejercicios de Recursión

Menor elemento

Dado un array que no está necesariamente ordenado, implemente el método `Min` que busca recursivamente el menor de los elementos.

```
int Min(int[] elements) {
    // Devuelve le menor de los elementos en el array
}
```

Búsqueda binaria

Dado un *array* ordenado de enteros, implemente el método `BinarySearch` que realiza una búsqueda binaria recursiva en el array.

```
bool BinarySearch(int[] array, int x) {
    // Devuelve true si el elemento `x` está en el array.
}
```

Factorial

El factorial de un número entero, $n!$, se define como la multiplicación de todos los números entre 1 y n:

$$n! = \Pi_{k=1}^{n} k$$

Una posible definición recursiva de $n!$ es la siguiente:

$$n! = n \cdot (n-1)!$$

Tenga en cuenta que por definición, $0! = 1$.

Implemente el método `Factorial` que computa el factorial de un número de forma recursiva.

```
int Factorial(int n) {
    // Devuelve n!
}
```

Fibonacci

La sucesión de Fibonacci se define recursivamente de la siguiente forma:

$$\begin{aligned} F(0) &= 1 \\ F(1) &= 1 \\ F(n) &= F(n-1) + F(n-2) \end{aligned}$$

Implemente el método `Fibonnaci` que calcula el n-ésimo elemento de la sucesión de Fibonnaci.

```
int Fibonacci(int n) {
    // Calcula el n-ésimo elemento de Fibonacci
}
```

Palindromo

Una cadena de caracteres es palíndromo si se lee igual de izquierda a derecha que de derecha a izquierda. Note que una cadena puede ser palíndromo tanto si es de longitud par como si es impar.

Implemente el método `EsPalindromo` que devuelve `true` si la cadena es palíndromo, de forma recursiva.

```
bool EsPalindromo(string s) {
    // Devuelve true is s es palíndromo
}
```

Recorrido del caballo

Se desea saber si existe alguna manera de recorrer un tablero de ajedrez de $n \times n$ celdas con un caballo, tal que empezando en la celda superior izquierda (digamos que es la celda $(0, 0)$) el caballo pase exactamente una vez por cada celda.

Recuerde que el movimiento del caballo es dos celdas en una dirección (horizontal o vertical) y luego una celda en una dirección ortogonal.

Implemente el método `HayRecorridoCaballo` que devuelve `true` si existe al menos una forma de hacer este recorrido en un tablero de tamaño $n \times n$:

```
bool HayRecorridoCaballo(int n) {
    // Devuelve true si existe al menos un recorrido
    // del caballo en un tablero de n x n
}
```

Implemente el método `RecorridosCaballo` que devuelve la cantidad de recorridos posibles en dicho tablero:

```
int RecorridosCaballo(int n) {
    // Devuelve la cantidad total de posibles recorridos
    // del caballo en un tablero de n x n
}
```

Ejercicios de Recursión

Menor elemento

Dado un array que no está necesariamente ordenado, implemente el método `Min` que busca recursivamente el menor de los elementos.

```
int Min(int[] elements) {
    // Devuelve le menor de los elementos en el array
}
```

Búsqueda binaria

Dado un *array* ordenado de enteros, implemente el método `BinarySearch` que realiza una búsqueda binaria recursiva en el array.

```
bool BinarySearch(int[] array, int x) {
    // Devuelve true si el elemento `x` está en el array.
}
```

Factorial

El factorial de un número entero, $n!$, se define como la multiplicación de todos los números entre 1 y n:

$$n! = \Pi_{k=1}^{n} k$$

Una posible definición recursiva de $n!$ es la siguiente:

$$n! = n \cdot (n-1)!$$

Tenga en cuenta que por definición, $0! = 1$.

Implemente el método `Factorial` que computa el factorial de un número de forma recursiva.

```
int Factorial(int n) {
    // Devuelve n!
}
```

Fibonacci

La sucesión de Fibonacci se define recursivamente de la siguiente forma:

$$\begin{aligned} F(0) &= 1 \\ F(1) &= 1 \\ F(n) &= F(n-1) + F(n-2) \end{aligned}$$

Implemente el método `Fibonnaci` que calcula el n-ésimo elemento de la sucesión de Fibonnaci.

```
int Fibonacci(int n) {
    // Calcula el n-ésimo elemento de Fibonacci
}
```

Palindromo

Una cadena de caracteres es palíndromo si se lee igual de izquierda a derecha que de derecha a izquierda. Note que una cadena puede ser palíndromo tanto si es de longitud par como si es impar.

Implemente el método `EsPalindromo` que devuelve `true` si la cadena es palíndromo, de forma recursiva.

```
bool EsPalindromo(string s) {
    // Devuelve true is s es palíndromo
}
```

Recorrido del caballo

Se desea saber si existe alguna manera de recorrer un tablero de ajedrez de $n \times n$ celdas con un caballo, tal que empezando en la celda superior izquierda (digamos que es la celda $(0, 0)$) el caballo pase exactamente una vez por cada celda.

Recuerde que el movimiento del caballo es dos celdas en una dirección (horizontal o vertical) y luego una celda en una dirección ortogonal.

Implemente el método `HayRecorridoCaballo` que devuelve `true` si existe al menos una forma de hacer este recorrido en un tablero de tamaño $n \times n$:

```
bool HayRecorridoCaballo(int n) {
    // Devuelve true si existe al menos un recorrido
    // del caballo en un tablero de n x n
}
```

Implemente el método `RecorridosCaballo` que devuelve la cantidad de recorridos posibles en dicho tablero:

```
int RecorridosCaballo(int n) {
    // Devuelve la cantidad total de posibles recorridos
    // del caballo en un tablero de n x n
}
```

Ejercicios sobre POO

Filtrado genérico

Se tiene la interfaz genérica `IFilter` que encapsula el concepto de un criterio de filtrado. Aquellos elementos para los que el método `Apply` devuelve `true` se consideran que cumplen el criterio asociado.

```
interface IFilter<T> {
    bool Apply(T item);
}
```

A partir de esta interfaz, se quiere implementar la interfaz `IFiltering` que encapsula operaciones a realizar con filtros.

```
interface IFiltering<T> {
    int Count(T[] items, IFilter<T> filter) {
        // Devolver la cantidad de elementos en `items`
        // que cumplen con el criterio de filtro.
    }

    T[] Select(T[] items, IFilter<T> filter) {
        // Devolver los elementos en `items` para los que
        // se cumple el criterio.
    }

    IFilter<T> Complement(IFilter<T> filter) {
        // Devolver una implementación de `IFilter`
        // que encapsule el criterio complementario
        // de `filter`.
    }
}
```

Por ejemplo, una posible implementación es la clase `OddFilter` que representa el criterio de un número que es impar.

```
class OddFilter: IFilter<int> {
    public bool Apply(int item) {
        return item % 2 != 0;
    }
}
```

A partir de este filtro y una implementación sensible de `IFiltering` se podría hacer lo siguiente:

```
IFiltering<int> filtering = new MyFiltering<int>(); // su implementación
IFilter<int> oddFilter = new OddFilter();

int[] items = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

filtering.Count(items, oddFilter); // Devolvería 5

filtering.Select(items, oddFilter);
// Devolvería {1, 3, 5, 7, 9}

filtering.Select(items, filtering.Complement(oddFilter));
// Devolvería {0, 2, 4, 6, 8}
```