Práctico 3

TDA - Tipo de Dato Abstracto

NOTA: Los ejercicios deberán entregarse completos, siguiendo los criterios aconsejados por la práctica y, con los controles adecuados, modularizado, etc.
Para todas las clases que considere apropiado especifique los operadores de inserción en el flujo de salida y de extracción en el flujo de entrada `<<` y `>>`.
Cada clase va en un archivo .h con el mismo nombre de la clase pero en minúscula, por ejemplo point.h

EJERCICIO 1

Implementar una clase **Point** que contenga como miembro dato las coordenadas x e y.
La clase debe contener:

1. un constructor por defecto que inicializa el punto en las coordenadas (0,0) y un constructor que inicializa a valores fijos pasados por parámetro;
2. los métodos necesarios para acceder y modificar los datos de un punto; y,
3. un método para visualizar un punto.

EJERCICIO 2

Implementar un TDA simple llamado **Counter** cuya finalidad es contar desde un valor dado o desde cero en caso contrario, el nuevo TDA solo soporta incrementos.

Respetar el siguiente prototipo:

```
#include <iostream>

class Counter {
public:
    Counter();
    Counter(const Counter &c);
    Counter(const unsigned int &c);
    Counter operator=(int i); // "asigna un entero al Counter"
    Counter operator=(const Counter &c);
    friend std::ostream& operator<<(std::ostream& out, Counter c);
    Counter operator++(); // "Pre-increment Operator"
    Counter operator++(int); // "Post-increment Operator"
    Counter& operator+=(int i);
private:
   unsigned int count;
};
```

Nota: El lenguaje C++ resuelve el problema de ambigüedad en los operados Post y Pre incremento utilizando un argumento ficticio de tipo entero, solo sirve para eso.

EJERCICIO 3

Implementar una clase **Hour** que contenga como miembro dato la hora, los minutos y los segundos. La clase debe contener:

- un constructor por defecto que inicializa la hora en 00:00:00 y un constructor que inicializa a valores pasados por parámetro;
- los métodos necesarios para acceder y modificar los datos de una hora
- un método (operador) que sume dos horas
- un operador para visualizar una hora usando **cout**.
- operadores sobrecargados `<`, `==`, `!=`, `<=`, `>=` y `>` que permitan comparar dos horas.

EJERCICIO 4

Implementar una clase **Fraction** parametrizada que permita manejar un número fraccionario. La clase debe contener:

1. Un constructor por defecto que inicializa en 0/1, `Fraction()`.
2. Un constructor que inicializa los dos parámetros (numerador y denominador), `Fraction(T num, T den) T puede ser cualquier tipo de dato entero.`
3. Un constructor de copia, `Fraction(const Fraction &f)`.
4. Métodos seteadores y accesores.
5. Un método `float toFloat()` que devuelva el valor de la fracción en un número real.
6. Un método `bool simplify()` que simplifique la fracción devolviendo si fue posible hacerlo (true).
7. Un método `string toString()` que convierta la fracción en un string.
8. Operadores de sobrecarga para:
    a. multiplicar la fracción por un escalar
    b. multiplicar la fracción por una fracción
    c. restar una fracción
    d. sumar una fracción
    e. dividir una fracción
    f. imprimir una fracción usando **cout**

EJERCICIO 5

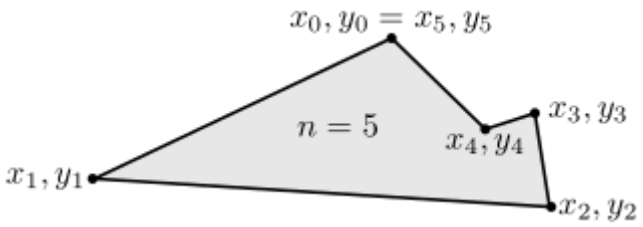
Implementar una clase **Date** que contiene como atributos **day**, **month** y **year**. La clase debe incluir:

- un constructor por defecto que inicializa la fecha actual del sistema;
- un constructor que inicializa según un `string` con el formato **dd/mm/yyyy** pasado por parámetro;
- los métodos set y get para cada atributo; y,
- los operadores sobrecargados `<`, `==`, `!=`, `<=`, `>=` y `>` que permitan comparar dos fechas.

Compare y analice esta solución con la realizada en el Práctico 1 (Ejercicio 2) teniendo en cuenta conceptos de Abstracción, Ocultamiento de información y Encapsulamiento. ¿Cuáles serían las ventajas y desventajas de ambas implementaciones?

EJERCICIO 6

En geometría, un polígono es una figura geométrica plana compuesta por una secuencia finita de segmentos rectos consecutivos que encierran una región en el plano. Estos segmentos son llamados lados, y los puntos en que se intersectan se llaman vértices.
Se pide implementar una clase **Polygon** para representar un polígono simple, defina los constructores, seteadores y accesores, operadores y otros métodos que considere de utilidad.
En la siguiente figura se puede observar un ejemplo de un polígono de 5 segmentos.



Se va a valorar la selección de los tipos de datos y la calidad del código; simplicidad, claridad, documentación, la complejidad de los algoritmos y variedad de métodos que decida incluir en la clase.

Observación
En la definición de los operadores tiene que primar el sentido común y siempre seguir un comportamiento análogo al de los tipos primitivos. Por ejemplo sumar horas es semánticamente y sintácticamente equivalente a sumar enteros.

```
{
    Hour h1, h2, h3;
    h3 = h1 + h2;
    cout << h1 <<  "+" << h2 << "=" << h3;
}
{
    int h1, h2, h3;
    h3 = h1 + h2;
    cout << h1 <<  "+" << h2 << "=" << h3;
}
```

En el caso de la clase Counter podemos encontrar cierta analogía con el tipo int al momento de usar los operadores ++:

```
   {
        Counter c1, c2, c3(1);
        c1 = c2 = c3;
        (c1+=10)+=1;
        c1++;
        cout << endl << "c1++ "<< c1++;
        cout << endl << "++c1 "<< ++c1;
```

```
        cout << endl << "c1: "<< c1;
        cout << endl << "c2: "<< c2;
        cout << endl << "c3: "<< c3;
    }
    {
        cout << endl << endl;
        unsigned int c1, c2, c3 = 1;
        c1 = c2 = c3;
        (c1+=10)+=1;
        c1++;
        cout << endl << "c1++ "<< c1++;
        cout << endl << "++c1 "<< ++c1;
        cout << endl << "c1: "<< c1;
        cout << endl << "c2: "<< c2;
        cout << endl << "c3: "<< c3;
    }
```