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 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.

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;
}
```

EJERCICIO 3

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 teniendo en cuenta conceptos de Abstracción, Ocultamiento de información y Encapsulamiento. ¿Cuáles serían las ventajas y desventajas de ambas implementaciones?

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 convierte 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 **DateTime** para representar la fecha y la hora. La clase debe contener:

1. un constructor por defecto que inicializa la fecha y hora actual, y un constructor que inicializa a valores pasados por parámetro;
2. un método llamado **timestamp()** para convertir un objeto de tipo DateTime en un entero largo (**long**) que representa el tiempo transcurrido en segundos desde 01/01/1970 00:00 (EPOCH, UNIX Time).
3. los métodos set y get para cada atributo; y,
4. los operadores sobrecargados `<`, `==`, `!=`, `<=`, `>=` y `>` que permitan comparar dos **DateTime**.

EJERCICIO 6

Adaptar el tipo de dato abstracto **EArray** visto en el práctico 1 para que soporte el paradigma orientado a objetos (**struct** → **class**), puede incorporar:

- constructores,
- destructores,
- operadores y
- otros métodos que permitan la manipulación del EArray.