Boletín de Ejercicios Tema 1

Programación Avanzada

Grado en Ingeniería Informática y Tecnologías Virtuales

Programación III

Grado en Ingeniería del Software

Curso académico 2025/2026

Antonio M. Durán Rosal

1. Ejercicios sobre Sobrecarga de Operadores

Ejercicio 1A - Básico

Defina la clase **Fraccion** que representa fracciones matemáticas del tipo $\frac{a}{b}$:

a) La clase tendrá dos atributos privados: **numerador** y **denominador**, ambos enteros. El denominador debe ser siempre distinto de cero.

b) Implemente constructores, métodos para leer y modificar cada atributo, y una función que simplifique la fracción.

c) Sobrecargue los operadores $+$ y $\ll$ para sumar dos fracciones y mostrar el resultado por pantalla.

Ejercicio 1B - Medio

Defina la clase **Complejo** que representa números complejos en forma binómica $(a+bi)$:

a) La clase tendrá dos atributos privados: **real** e **imaginario**, ambos de tipo **double**.

b) Implemente constructores, métodos de lectura y modificación, y una función que devuelva el módulo del número complejo.

c) Sobrecargue los operadores $+$, $-$, $*$, $==$ y $\ll$.

Ejercicio 1C - Avanzado

Defina la clase **Tiempo** que representa una duración en horas y minutos:

a) La clase tendrá dos atributos privados: **horas** y **minutos**, ambos de tipo **int**.

b) Implemente constructores, métodos para consultar y modificar cada atributo, y una función que normalice el tiempo (por ejemplo, 75 minutos se convierte en 1 hora y 15 minutos).

c) Sobrecargue los operadores $+$, $-$ (para sumar/restar tiempos), $==$ (para comprobar igualdad), y $\ll$ (para mostrar por pantalla en formato `hh:mm`).

2. Ejercicios sobre Templates

Ejercicio 2A - Básico

Implemente una clase que represente a un **Punto2D** (punto de dos dimensiones).

a) La clase tendrá dos variables privadas (**x**, **y**) de tipo **float**.

b) La clase estará dotada de constructores, métodos para obtener y modificar cada coordenada, y una función que calcule la distancia al origen.

c) Todas las funciones miembro se implementarán en el mismo archivo .h.

Ejercicio 2B - Medio

Implemente una plantilla de clase que represente a un **Vector3D** en un espacio tridimensional.

a) La clase tendrá tres atributos privados (**i**, **j**, **k**) que podrán ser de cualquier tipo numérico.

b) La clase incluirá constructores, observadores, modificadores, y métodos para calcular el módulo y el producto escalar con otro vector del mismo tipo.

c) Se debe usar un .cpp separado.

Ejercicio 2C - Avanzado

Implemente una plantilla de clase que represente a un **Punto4D** (punto en un espacio de cuatro dimensiones).

a) La clase tendrá cuatro atributos privados (**x**, **y**, **z**, **w**) de tipo genérico.

b) Se deberán definir constructores, métodos selectores y modificadores, así como una función para calcular el módulo del punto (distancia al origen).

c) Sobrecargue los operadores $==$ y $\ll$ para comparar puntos y mostrarlos por pantalla.

d) Se debe usar un .cpp separado.

3. Ejercicios sobre Excepciones

Ejercicio 3A - Básico

Mostrar el uso de **bad_cast**. Para ello:

a) Defina la clase **BadCastEs** que herede de **bad_cast** en el archivo BadCastEs.h.

b) Redefina los métodos que considere necesario para mostrar el mensaje por pantalla: "Error: tipo BadCastException".

c) Cree un método principal que compruebe que se muestra el mensaje de la clase **BadCastEs** en caso de que ocurra un error de tipo **bad_cast**.

Ejercicio 3B - Medio

Mostrar el uso de **runtime_error**. Para ello:

a) Defina la clase **RuntimeErrorES** que herede de **runtime_error** en el archivo RuntimeErrorES.h.

b) Redefina los métodos que considere necesario para mostrar el mensaje por pantalla: "Error: tipo RuntimeErrorException".

c) Cree un método principal que compruebe que se muestra el mensaje de la clase **RuntimeErrorEs** en caso de que ocurra un error de tipo **runtime_error**.

Ejercicio 3C - Avanzado

a) Defina la clase **EdadInvalidaException** con un método público que muestre por pantalla el mensaje: "La edad introducida no puede ser negativa".

b) Defina una función **leerEdad** que solicite por pantalla un número entero representando la edad de una persona. Si el valor introducido es negativo, debe lanzar la excepción **EdadInvalidaException**.

c) Defina un método principal que solicite la edad de un usuario, valide su valor y, en caso de que sea negativo, lance y capture la excepción mostrando el mensaje correspondiente.

4. Ejercicios STL

Ejercicio 4A - Medio

Escriba un programa que permita crear dos listas (**list**) de nombres (**string**) introducidos por el usuario y que realice las siguientes operaciones (sin permitir duplicados):

a) Lectura y escritura de ambas listas desde teclado, asegurando que no se repitan nombres dentro de cada lista.

b) Eliminar de cada lista los nombres que tengan menos de 4 caracteres.

c) Generar una lista con los nombres que aparecen en ambas listas (intersección).

d) Generar una lista con los nombres que aparecen en la primera lista pero no en la segunda (diferencia).

e) Generar una lista con todos los nombres que empiezan por vocal.

f) Ordenar alfabéticamente las listas y mostrarlas por pantalla.

Nota: asegúrese de eliminar los elementos duplicados antes de realizar las operaciones solicitadas.

Ejercicio 4B - Medio

Repita el ejercicio anterior, pero utilizando conjuntos (**set**) en lugar de listas.

Ejercicio 4C - Avanzado

Escriba un programa que permita gestionar un conjunto de personas, almacenando sus datos en un **map<string, int>** donde la clave es el **nombre** y el valor es la **edad**. El programa debe realizar las siguientes operaciones:

a) Lectura de los datos desde teclado (nombre y edad), evitando nombres repetidos.

b) Eliminar del mapa todas las personas cuya edad sea menor que un valor introducido por teclado.

c) Obtener un nuevo mapa que contenga únicamente las personas cuya edad sea un número primo.

d) Obtener los nombres de las personas cuya edad sea la máxima registrada.

e) Crear un mapa invertido: **map<int, list<string>>** que agrupe los nombres por edad.

f) Mostrar el contenido de ambos mapas ordenadamente por clave.

Nota: el mapa invertido deberá agrupar todos los nombres que comparten la misma edad.

5. Ejercicios Resumen

Ejercicio 5A

Diseñe una clase **CentroDeportivo** con las propiedades: **nombre**, **ciudad**, **número de socios** y **disciplina principal**. Complete la clase con sus métodos correspondientes (constructores, observadores, modificadores, funciones de E/S). Finalmente, implemente una serie de funciones (fuera de la clase) que devuelvan las siguientes estructuras de mapeo (**map**). Todas las funciones recibirán como parámetro un conjunto de objetos **CentroDeportivo** creado en el programa principal **main**.

a) Devolver un **map<string, list<string>>** que relacione cada ciudad con la lista de centros deportivos en esa ciudad.

b) Devolver un **map<string, list<string>>** que relacione cada disciplina con la lista de centros que la ofrecen como principal.

c) Devolver un **map<string, int>** que relacione cada disciplina con el número total de socios que la practican.

d) Devolver un **map<string, double>** que relacione cada ciudad con el número medio de socios por centro deportivo.

e) Devolver un **map<string, string>** que relacione cada ciudad con la disciplina principal más común en esa ciudad.

Ejercicio 5B

Supongamos un tipo **Pelicula** con las siguientes propiedades:

- **string** Título.
- **string** Director.
- **string** Código IMDB: debe tener exactamente 9 caracteres.
- **int** Duración en minutos: debe ser positiva.
- **double** Presupuesto en millones de euros.

Construya una clase **Pelicula** para implementar este tipo. Todos los atributos serán consultables, y modificables sólo la duración y el presupuesto. El orden natural será por título en orden ascendente y, en caso de coincidencia, por director en orden descendente. Proporcione un constructor para **Pelicula** a partir de una cadena de caracteres que contenga todos los valores de sus atributos separados por ampersand (**&**).

Ejercicio 5C

Se quiere representar un catálogo de cine con una clase denominada **CatalogoPeliculas**, definida por un código (**int**), un responsable (**string**) y un conjunto de películas representado por un atributo de tipo set<**Pelicula**>. Proporcione un constructor que reciba un código y un nombre de responsable, e inicialice el conjunto de películas vacío. La clase debe incluir la siguiente funcionalidad:

a) Añadir una película.

b) Eliminar una película.

c) Dada una duración mínima, devolver todas las películas cuya duración sea al menos ese valor.

d) Dado un presupuesto umbral, determinar cuántas películas tienen un presupuesto inferior.

e) Dado un director, calcular el presupuesto medio de sus películas.

f) Dado un director, devolver la película más larga que haya dirigido.

g) Función que calcule la duración media de las películas en el catálogo.

h) Dado un código IMDB, calcular el presupuesto medio de las películas con código menor que el proporcionado.

i) Dado un porcentaje p y un valor v, incrementar en p % el presupuesto de todas las películas cuyo presupuesto sea menor que v.

Ejercicio 5D

A la clase **CatalogoPeliculas** anteriormente definida, añada las siguientes funcionalidades como métodos internos:

a) Un método que devuelva una estructura que relacione cada director con todas las películas (título) que ha dirigido.

b) Un método que devuelva una estructura que acumule la duración total de las películas dirigidas por cada director.

c) Un método que devuelva una estructura que acumule el presupuesto total de películas por director.

d) Un método que devuelva la(s) película(s) con mayor duración (en una lista si hay empates).

e) Un método que devuelva el/los director(es) con el menor número de películas.