Ejercicios Guiados sobre Clases en C++

Fundamentos de Programación CC112

Instrucciones

Cada ejercicio tiene un enfoque práctico y está guiado paso a paso para que puedan comprender mejor los conceptos de clases, atributos, métodos, encapsulamiento, constructores, destructores y objetos en C++. Procure agregar comentarios relevantes a su implementación.

Ejercicio 1: Clase Persona

- Define una clase llamada `Persona` con atributos privados: nombre (cadena) y edad (entero).
- Agrega un constructor que inicialice estos atributos.
- Implementa un método `mostrar()` que imprima los valores.
- En el `main`, crea un objeto de tipo `Persona` y muestra sus datos.

Ejercicio 2: Getters y Setters

- A partir del ejercicio anterior, agrega métodos `getNombre()`, `getEdad()`, `setNombre()` y `setEdad()`.
- En el `main`, modifica los valores con los setters y verifica los cambios con los getters.

Ejercicio 3: Cuenta Bancaria

- Crea una clase `CuentaBancaria` con los atributos: titular (cadena) y saldo (double).
- Métodos: `depositar(double)`, `retirar(double)`, `mostrarSaldo()`.
- Evita retiros mayores al saldo disponible.

Ejercicio 4: Constructor y Destructor

- Define una clase `Recurso` que imprima un mensaje en su constructor y otro en su destructor.
- En el `main`, crea objetos de esta clase y observa el orden de ejecución.

Ejercicio 5: Clase Rectángulo

- Atributos: base y altura.
- Métodos: `area()` y `perimetro()`.
- Instancia varios rectángulos en `main()` y muestra sus áreas y perímetros.

Ejercicio 6: Sobrecarga de métodos

- Crea una clase `Calculadora` con dos métodos `suma(int, int)` y `suma(double, double)`.
- Llama ambos desde el `main()` para ver cómo actúa la sobrecarga.

Ejercicio 7: Clase Estudiante con promedio

- Atributos: nombre, tres notas.
- Método `calcularPromedio()` que devuelve el promedio.
- Mostrar el resultado desde el `main()`.

Ejercicio 8: Uso de vectores de objetos

- Define una clase `Producto` con nombre y precio.
- Crea un vector de productos e imprime sus precios totales.

Ejercicio 9: Clase Fecha con validación simple

- Atributos: día, mes, año.
- Verifica si la fecha es válida (día entre 1 y 31, mes entre 1 y 12).
- Método `mostrarFecha()`.

Ejercicio 10: Clase Temperatura con conversión

- Atributo: gradosCelsius.
- Métodos: `aFahrenheit()`, `aKelvin()`.
- Muestra los valores convertidos.

Ejercicio 11: Encapsulamiento con validación

- Crea una clase `Empleado` con atributos privados: nombre y sueldo.
- Usa `setSueldo()` para validar que el sueldo no sea negativo.
- Muestra los datos usando métodos públicos.

Ejercicio 12: Lista de inicialización en constructor

- Implementa una clase `Libro` con título y número de páginas.
- Utiliza una lista de inicialización para asignar valores en el constructor.
- Imprime los datos desde un método `mostrar()`.

Ejercicio 13: Uso del puntero this

- Crea una clase `Caja` con métodos `setDimensiones()` y `comparar(Caja)`.
- Usa el puntero `this` para comparar dos objetos tipo `Caja`.

Ejercicio 14: Constructor por defecto y parametrizado

- Diseña una clase `Punto` con coordenadas x, y.
- Implementa un constructor por defecto y otro con parámetros.
- Crea varios objetos en `main()` y muestra sus coordenadas.

Ejercicio 15: Destructor con mensaje

- Modifica la clase `Punto` para que muestre un mensaje al ser destruida.
- Verifica su ejecución al salir del bloque `main()`.

Ejercicio 16: Clase con atributos constantes

- Implementa una clase `ConstantePi` con un atributo constante de tipo `double`.
- Asigna su valor usando la lista de inicialización.
- Verifica que no puede ser modificado.

Ejercicio 17: Punteros a objetos clase

- Define una clase `Animal` con un método `hablar()`.
- Crea un objeto `Animal* a = new Animal()` y llama a su método mediante el puntero.
- Libera la memoria con `delete`.

Ejercicio 18: Arreglo dinámico de objetos

- Crea una clase `Producto` con nombre y precio.
- Declara un arreglo dinámico de objetos `Producto` usando punteros.
- Inicializa los objetos y muestra sus datos.

Ejercicio 19: Clase con método constante

- Implementa una clase `Circulo` con radio y un método `area()` marcado como `const`.
- Asegúrate de no modificar atributos dentro del método.

Ejercicio 20: Paso de objetos por referencia

- Crea una clase `Jugador` con nombre y puntos.
- Implementa una función externa que reciba un `Jugador&` y modifique sus puntos.
- Observa cómo los cambios afectan al objeto original.

Ejercicio 21 : Clase `Complejo`

Un número complejo consta de dos partes: una **parte real** y una **parte imaginaria**. Por ejemplo, en el número complejo $(4{,}5 + 3{,}0i)$, la parte real es 4,5 y la parte imaginaria es 3,0.

Se desea implementar una clase `Complejo` que permita representar y operar con números complejos. Cada objeto de esta clase almacenará dos números reales: uno para la parte real

y otro para la parte imaginaria.

La clase debe permitir realizar las siguientes operaciones:

- Un método `leerComplejo()` que lea un número complejo desde la entrada estándar.
- Un método `escribirComplejo()` que muestre un número complejo en el formato $(a + bi)$.

Suponga que se tienen dos números complejos $a = (A, Bi)$ y $c = (C, Di)$. Se deben implementar los siguientes métodos como parte de la clase:

- **Suma:** $a + c = (A + C, (B + D)i)$
- **Resta:** $a - c = (A - C, (B - D)i)$
- **Multiplicación compleja:** $a \cdot c = (AC - BD,\ (AD + BC)i)$
- **Multiplicación por escalar real:** $x \cdot c = (xC, xDi)$, donde x es un número real
- **Conjugado:** $\sim a = (A,\ -Bi)$

En el `main()`, escriba un programa que demuestre el uso de la clase `Complejo` leyendo dos números complejos y un escalar real, realizando las operaciones indicadas y mostrando sus resultados.

Ejercicio 22: Clase `Vector3D`

Se desea implementar una clase llamada `Vector3D` que permita representar y manipular vectores tridimensionales con coordenadas (x, y, z). La clase debe cumplir con las siguientes especificaciones:

1. Debe contar con un **constructor en línea** que inicialice las tres coordenadas.
2. Debe implementar una función miembro llamada `igual` que determine si dos vectores son iguales (es decir, si sus componentes x, y, z son idénticas). Esta función debe escribirse en tres versiones:
   - a) Usando paso por valor.
   - b) Usando paso por dirección (puntero).
   - c) Usando paso por referencia.
3. Agregar una función miembro `normaMax` que reciba otro objeto de tipo `Vector3D` y devuelva la norma mayor entre ambos vectores. La norma de un vector $v = (x, y, z)$ se define como:
$$\|v\| = \sqrt{x^2 + y^2 + z^2}$$

4. Añadir también las siguientes funciones miembro:
   - `suma`: que devuelva la suma de dos vectores.
   - `productoEscalar`: que calcule el producto escalar entre dos vectores. Dados:

     $$v_1 = (x_1, y_1, z_1), \quad v_2 = (x_2, y_2, z_2)$$

     el producto escalar está dado por:

     $$v_1 \cdot v_2 = x_1x_2 + y_1y_2 + z_1z_2$$

Implemente un programa principal (`main()`) que permita probar cada una de las funcionalidades implementadas.

Ejercicio 23: Clase `Fecha` con validación

Se desea implementar una clase llamada `Fecha` cuyos objetos representen una fecha del calendario, mediante tres atributos de tipo entero: día, mes y año.

La clase debe cumplir con las siguientes especificaciones:

1. Incluir un **constructor por defecto** que inicialice la fecha al 1 de enero del año 2000.
2. Incluir un **constructor de copia**.
3. Incluir funciones de acceso (`getDia()`, `getMes()`, `getAnio()`).
4. Implementar una función `reiniciar(int d, int m, int a)` que permita modificar la fecha de un objeto ya creado.
5. Implementar una función `adelantar(int d, int m, int a)` que sume los valores indicados al día, mes y año actuales del objeto.
6. Agregar una función `imprimir()` que muestre la fecha en formato `dd/mm/aaaa`.
7. Utilizar una función interna `normalizar()` para garantizar que:
   - El año sea mayor o igual a 1.
   - El mes esté en el rango $1 \leq m \leq 12$.
   - El día no exceda el número de días del mes correspondiente.
8. Para ello, se debe definir una función de utilidad `diasDelMes(int mes, int anio)` que retorne el número de días del mes especificado, considerando años bisiestos cuando corresponda.

Extensión: Modificar la función `diasDelMes()` para que tenga en cuenta los años bisiestos. Un año se considera bisiesto si cumple con las siguientes condiciones:

- Es divisible entre 400, o
- Es divisible entre 4 pero no entre 100.

Por ejemplo, los años 1992 y 2000 son bisiestos, mientras que 1900 y 1997 no lo son.

Implemente un programa principal (`main()`) que cree varios objetos `Fecha`, los modifique usando las funciones disponibles y verifique su comportamiento con fechas válidas e inválidas.

Ejercicios sobre herencia y polimorfismo

1. **Herencia básica: Persona y Estudiante**
   Se desea modelar a una persona y, a partir de ella, un estudiante. Define una clase `Persona` con los atributos: `nombre` (cadena de texto) y `edad` (entero). Luego crea una clase derivada `Estudiante` que herede de `Persona` y agregue un nuevo atributo: `carrera` (cadena de texto). Implementa funciones miembro para ingresar y mostrar los datos.
   Objetivo: Aprender a declarar una clase derivada y acceder a atributos heredados.

2. **Constructores en la herencia**
   Extiende el ejercicio anterior para incluir constructores con parámetros en ambas clases. El constructor de `Estudiante` debe llamar al constructor de `Persona` para inicializar los atributos heredados.
   Objetivo: Comprender cómo se invocan constructores de la clase base desde la clase derivada.

3. **Acceso protegido:** `protected`
   Modifica los atributos de la clase `Persona` para que sean `protected` en lugar de `private`. Luego accede directamente a esos atributos desde la clase derivada `Estudiante`.
   Objetivo: Entender la diferencia entre `private`, `protected` y `public` en herencia.

4. **Sobreescritura de funciones miembro**
   Agrega a `Persona` una función miembro llamada `mostrarDatos()`. Luego redefine esa función en `Estudiante` para mostrar todos los atributos, incluyendo `carrera`. Crea un objeto de `Estudiante` y llama a `mostrarDatos()`.
   Objetivo: Practicar cómo sobrescribir métodos heredados.

5. **Polimorfismo simple: uso de punteros a la clase base**
   Declara `mostrarDatos()` como función `virtual` en `Persona`. Luego, crea un puntero de tipo `Persona*` que apunte a un objeto de tipo `Estudiante`, y llama a `mostrarDatos()` usando ese puntero.
   Objetivo: Entender el funcionamiento del polimorfismo con funciones virtuales.

6. **Herencia múltiple: EstudianteDeportista**
   Crea una clase `Deportista` con un atributo `disciplina`. Define una nueva clase `EstudianteDeportista` que herede tanto de `Estudiante` como de `Deportista`. Implementa métodos que permitan ingresar y mostrar los datos de un estudiante deportista.
   Objetivo: Comprender cómo heredar de múltiples clases y combinar datos.

7. **Constructores y destructores en jerarquías**
   Agrega mensajes a los constructores y destructores de las clases `Persona`, `Estudiante` y `EstudianteDeportista`. Crea un objeto y observa el orden de ejecución al construir y destruir los objetos.
   Objetivo: Observar el flujo de ejecución en la construcción y destrucción de objetos con herencia.

8. **Vectores de objetos derivados**
   Define una clase `Producto` con los atributos `nombre` y `precio`. Crea una clase derivada `ProductoConDescuento` que añade un atributo `descuento` (porcentaje). Redefine un método que calcule el precio final. Luego crea un `vector` de punteros a `Producto` y almacena tanto productos normales como con descuento. Muestra el precio final de todos.
   Objetivo: Utilizar vectores de objetos polimórficos.

9. **Clases abstractas: Figuras geométricas**
   Define una clase abstracta `Figura` con una función virtual pura `double area()`. Crea clases derivadas `Circulo` y `Rectangulo` que implementen la función `area()`. Crea un vector de punteros a `Figura` e imprime el área de cada una.
   Objetivo: Aprender a declarar e implementar clases abstractas.

10. **Jerarquía de clases y polimorfismo profundo**
    Crea una jerarquía con las siguientes clases:

    - `Animal` (clase base) con una función virtual `hablar()`
    - `Mamifero` (hereda de `Animal`)
    - `Perro` (hereda de `Mamifero`) y redefine `hablar()` para imprimir "Guau".

    Crea una función que reciba un puntero a `Animal` y llame a `hablar()`. Muestra cómo el polimorfismo permite llamar a la función correcta según el tipo del objeto.
    Objetivo: Comprender la utilidad del polimorfismo con múltiples niveles de herencia.

11. **Polimorfismo con funciones virtuales**
    Define una clase base `Empleado` que tenga una función virtual llamada `calcularPago()`. Luego crea dos clases derivadas: `EmpleadoAsalariado` y `EmpleadoPorHora`. La primera recibe un salario fijo mensual, la segunda calcula el pago en base a horas trabajadas y pago por hora.
    Tarea: Crea un vector de punteros a `Empleado` que almacene distintos tipos de empleados. Utiliza una función para mostrar cuánto se le debe pagar a cada uno.
    Objetivo: Aplicar el polimorfismo con funciones virtuales.

12. **Asignación dinámica de objetos con punteros**
    Crea una clase `Libro` con atributos `título` y `autor`. Luego permite crear objetos dinámicamente usando el operador `new` y almacenar los objetos en un vector de punteros.
    Tarea: Implementa funciones para ingresar libros y mostrar la información desde el

vector. No olvides liberar la memoria al final usando `delete`.
Objetivo: Manejar correctamente memoria dinámica y punteros a objetos.

13. **Polimorfismo con método virtual puro**
Crea una clase abstracta `Figura3D` con un método virtual puro llamado `volumen()`. Luego crea las clases `Cubo` y `Esfera` que implementan ese método.
Tarea: Crea un vector de punteros a `Figura3D` con diferentes figuras. Recorre el vector y muestra el volumen de cada figura.
Objetivo: Aplicar clases abstractas, funciones virtuales puras y polimorfismo dinámico.

14. **Copias profundas y gestión dinámica de memoria**
Crea una clase `Cadena` que contenga un puntero dinámico a un arreglo de caracteres. Implementa:

    - Constructor por defecto
    - Constructor parametrizado
    - Constructor de copia
    - Operador de asignación (=)
    - Destructor

    Tarea: Verifica que al hacer copia de objetos, se realice una copia profunda y no se compartan direcciones de memoria.
    Objetivo: Comprender el manejo seguro de memoria en clases con punteros.

15. **Polimorfismo en acción: sistema de notificaciones**
Diseña una clase abstracta `Notificacion` con una función virtual pura `enviar()`. Luego crea clases derivadas como `CorreoElectronico`, `SMS` y `NotificacionPush`.
Tarea: Crea una función que reciba un vector de punteros a `Notificacion` y envíe notificaciones simuladas usando `enviar()`. Cada clase derivada debe imprimir un mensaje diferente.
Objetivo: Aplicar polimorfismo para diseñar sistemas flexibles y extensibles.