1. Modificadores de acceso a atributos y métodos

- **¿Qué son los modificadores de acceso a atributos y métodos?**
  Los modificadores de acceso en C++ controlan la visibilidad y el acceso a los atributos y métodos de una clase. Permiten definir qué partes del código pueden acceder o modificar estos datos, proporcionando seguridad y encapsulación.
- **¿Cuáles son los diferentes modificadores de acceso disponibles en C++?**
  Los modificadores de acceso en C++ son:
  - public: Accesible desde cualquier parte del programa.
  - protected: Accesible dentro de la clase y las clases derivadas.
  - private: Solo accesible dentro de la clase que lo define.
- **¿Cómo se utilizan los modificadores de acceso para controlar el acceso a los atributos y métodos de una clase?**
  Los atributos y métodos se declaran en la sección correspondiente (public, protected o private) de la clase, y esto determina su accesibilidad. Esto permite limitar el acceso a ciertos datos de la clase, lo cual es fundamental para la encapsulación.

2. Constructor de clase

- **¿Qué es un constructor de clase?**
  Un constructor es un método especial que se ejecuta automáticamente cuando se crea un objeto de la clase. Inicializa el objeto, configurando sus atributos iniciales.
- **¿Cuál es el propósito de un constructor de clase?**
  El propósito de un constructor es asegurar que los objetos se creen en un estado válido, asignando valores iniciales a sus atributos o realizando configuraciones necesarias.
- **¿Cómo se define un constructor de clase en C++?**
  Un constructor se define como una función cuyo nombre coincide con el nombre de la clase y no tiene tipo de retorno. Ejemplo:

```
class Persona {

 public: Persona() {

/* código de inicialización */

}

};
```

3. Destructor de clase

- **¿Qué es un destructor de clase?**
  Un destructor es un método especial que se llama automáticamente cuando un objeto es destruido (por ejemplo, cuando sale de alcance).
- **¿Cuál es el propósito de un destructor de clase?**
  Su propósito es liberar los recursos ocupados por el objeto, como memoria o archivos abiertos, asegurando una correcta gestión de recursos.

- **¿Cómo se define un destructor de clase en C++?**
  Se define como una función cuyo nombre es el nombre de la clase precedido por una tilde (~) y tampoco tiene tipo de retorno. Ejemplo:

```
class Persona {

public: ~Persona()

{ /* código de limpieza */ }

};
```

4. Parte private y public de una clase en C++

- **¿Qué es la parte private y public de una clase en C++?**
  private y public son secciones en una clase que indican la visibilidad de los miembros de la clase (atributos y métodos). private es para miembros solo accesibles dentro de la clase, mientras que public permite acceso desde fuera.
- **¿Cuáles son las diferencias entre las partes private y public de una clase?**
  La principal diferencia es la visibilidad: public es accesible externamente, mientras que private no lo es, permitiendo la encapsulación y protección de datos.
- **¿Cómo se utilizan los modificadores de acceso private y public para controlar el acceso a los atributos y métodos de una clase?**
  Se declara cada miembro en la sección correspondiente (private o public). Esto limita el acceso a atributos sensibles, permitiendo que solo se modifiquen mediante métodos de la clase.

5. Sobrecarga de métodos

- **¿Qué es la sobrecarga de métodos?**
  La sobrecarga de métodos permite que una clase tenga múltiples funciones con el mismo nombre pero diferentes parámetros.
- **¿Cuál es el propósito de la sobrecarga de métodos?**
  Permite realizar operaciones similares con diferentes tipos o números de argumentos, proporcionando flexibilidad y facilidad de uso.
- **¿Cómo se implementa la sobrecarga de métodos en C++?**
  Se implementa declarando varias funciones con el mismo nombre en la misma clase, pero con diferentes listas de parámetros:

```
class Calculadora {

public: int sumar(int a, int b);

 double sumar(double a, double b);

};
```

6. Colaboración de clases

- **¿Qué es la colaboración de clases en C++?**
  La colaboración de clases ocurre cuando una clase utiliza otra clase como parte de su implementación, sin que haya una relación de herencia entre ellas.
- **¿Cuál es el propósito de la colaboración de clases?**
  Facilita la reutilización de código y la modularidad, permitiendo que una clase use las funcionalidades de otra.
- **¿Cómo se implementa la colaboración de clases en C++?**
  Se implementa declarando objetos de una clase como atributos o parámetros en otra clase. Ejemplo:

```
class Motor { /* código */ };

class Coche { Motor motor; // Coche utiliza Motor como parte de su implementación

};
```

7. Herencia

- **¿Qué es la herencia?**
  La herencia es un mecanismo que permite que una clase (clase derivada) herede atributos y métodos de otra clase (clase base).
- **¿Cuáles son los diferentes tipos de herencia?**
  Los tipos de herencia en C++ son:
    - **Herencia pública**: Los miembros public y protected de la clase base se mantienen accesibles en la clase derivada.
    - **Herencia protegida**: Los miembros public de la clase base se convierten en protected en la clase derivada.
    - **Herencia privada**: Todos los miembros de la clase base se convierten en private en la clase derivada.
- **¿Cuáles son las ventajas y desventajas que ofrece?**
  Ventajas: facilita la reutilización de código y establece relaciones jerárquicas.
  Desventajas: puede llevar a dependencias fuertes entre clases y dificultar el mantenimiento.
- **¿Cómo se implementa en C++?**
  Se implementa especificando el tipo de herencia después del nombre de la clase base:

```
class Base { /* código */ };

class Derivada : public Base { /* código adicional */ };
```

8. Diferencia entre colaboración y herencia en programación orientada a objetos

- **¿Cuál es la diferencia entre la colaboración y la herencia en programación orientada a objetos?**

La colaboración es cuando una clase utiliza otra como parte de su funcionalidad (sin heredar de ella), mientras que la herencia establece una relación jerárquica entre clases, permitiendo que una clase derive características de otra. La colaboración se basa en la composición, mientras que la herencia se basa en extender el comportamiento de una clase existente.

Dado el siguiente código:

```
#include <iostream>

using namespace std;


class ClaseBase {

protected:

   int unaVar = 0;

public:

   ClaseBase(int x) : unaVar(x) {}

   void unMetodo(void) {

      cout << "unaVar = " << unaVar << endl;

   }

};


class ClaseDerivada : public ClaseBase {

public:

   ClaseDerivada(int x) : ClaseBase(x) {} // Invoca al constructor de la clase base para inicializar unaVar

};


int main() {

   ClaseDerivada obj1(50); // El constructor de la clase derivada invoca el constructor de la clase base

   obj1.unMetodo();

   return 0;

}
```

I. ¿Compila exitosamente el siguiente programa? ¿Por qué?
II. II. ¿Qué resultados se obtiene luego su ejecución?

- Rta I: Sí, el programa compila exitosamente porque la clase ClaseDerivada hereda correctamente de ClaseBase y el constructor de ClaseBase es invocado con éxito desde el constructor de ClaseDerivada.

- Rta II: El programa imprime: unaVar = 50