TRABAJO PRACTICO N°15

A - Explorar el material adjunto propuesto, complementar buscando información en otras fuentes y luego responder:
1. ¿Qué es una función virtual en C + +?
2. ¿Cuál es la diferencia entre una función virtual y una función normal en C + +? ¿Cuál es la diferencia entre una función virtual y sobrecarga de métodos en C + +?
3. ¿Cómo se declara una función virtual en una clase en C + +?
4. ¿Qué es el polimorfismo en C + +?
5. ¿Cuáles son las ventajas y desventajas del polimorfismo?
6. ¿Se puede tener una función virtual en una clase base en C + +? Justificar
7. ¿Qué es una función virtual pura y cómo se declara?
8. ¿Cómo se logra el polimorfismo en C + + utilizando punteros y referencias a objetos?
9. ¿Qué son las clases abstractas en C + +?
10. ¿Cuándo se debe utilizar el modificador "override" al definir una función virtual en una clase derivada?
B- Leer el material adjunto, ejecutar y analizar el código de ejemplo, y luego resolver las actividades planteadas en el TP Nº15

actividad del libro:

1. Crear una clase base "Figura" con una función virtual "calcularArea()". Luego, crea clases derivadas "Rectangulo" y "Circulo" que implementan la función calcularArea de acuerdo a sus
propias fórmulas.
2. Implementar una clase base "Animal" con una función virtual "hacerSonido()". Luego, crea
clases derivadas "Perro", "Gato" y "Pato" que sobrescribir la función hacerSonido para que emitan el sonido correspondiente.
3. Crear una clase base "Vehículo" con una función virtual "acelerar()". Luego, crea clases derivadas "Coche" y "Moto" que implementan la función acelerar de manera diferente.
4. Implementa una clase base "Empleado" con una función virtual "calcularSalario()". Luego, crea clases derivadas "Gerente" y "Vendedor" que calculen el salario de acuerdo a reglas diferentes (definir al menos).
5. Crea una clase base "InstrumentoMusical" con una función virtual "tocar()". Luego, crea clases derivadas "Piano", "Guitarra" y "Flauta" que implementan la función tocar de manera única para cada instrumento.

RESPUESTAS
1: Una función virtual es una función miembro de una clase que puede ser sobrescrita (overridden) por las clases derivadas. Se usa para implementar el polimorfismo dinámico, dejando que el comportamiento de una función depende del tipo real del objeto apuntado por un puntero o referencia, y no del tipo estático del puntero/referencia.

2: a) Función virtual vs. función normal:

Una función normal está vinculada en tiempo de compilación (resolución estática), mientras que una función virtual utiliza vinculación en tiempo de ejecución (dinámica) si es llamada a través de un puntero o referencia.

Las funciones normales no dejan ser redefinidas dinámicamente por clases derivadas, mientras que las funciones virtuales sí.

b) Función virtual vs. sobrecarga de métodos:

La sobrecarga de métodos se refiere a definir varias funciones con el mismo nombre dentro de una clase pero con diferentes firmas (argumentos). La elección de qué función llamar se hace en tiempo de compilación.

Las funciones virtuales permiten redefinir funciones en clases derivadas y usar la resolución dinámica en tiempo de ejecución para invocar la versión apropiada según el tipo del objeto.

3: Se usa la palabra clave virtual en la declaración dentro de la clase base:

```
class Base {
public:
    virtual void display() {
        std::cout << "Base class display" << std::endl;
    }
```

4: El polimorfismo permite que un mismo nombre de función se comporte de diferentes maneras según el contexto. En C++, el polimorfismo puede ser:

En tiempo de compilación (polimorfismo estático): un ejemplo sería la sobrecarga de funciones.

En tiempo de ejecución (polimorfismo dinámico): un ejemplo sería que en funciones virtuales. Esto se logra con punteros o referencias a clases base que invocan métodos redefinidos en clases derivadas.

5: Ventajas:

Extensibilidad: Las clases derivadas pueden modificar o extender comportamientos sin cambiar el código base.
Reutilización: Facilita la reutilización de código al permitir trabajar con clases base y derivadas de forma genérica.
Mantenimiento: Mejora la claridad del código y reduce la necesidad de condiciones como if/switch.

Desventajas:

Sobrecarga de rendimiento: Las llamadas a funciones virtuales son ligeramente más lentas debido a la tabla virtual (vtable).
Complejidad: Puede hacer el código más difícil de entender, especialmente cuando se usa en exceso.

Uso de memoria: Introduce un ligero aumento en el uso de memoria por la necesidad de mantener la vtable.

6: Sí, una clase base puede tener funciones virtuales, esto es esencial para implementar el polimorfismo dinámico. La idea es declarar una función virtual en la clase base para que las clases derivadas pueden sobrescribirla, y el comportamiento sea determinado en tiempo de ejecución.

Ejemplo:

```
class Base {
public:
    virtual void show() { std::cout << "Base show" << std::endl; }
};

class Derived : public Base {
public:
    void show() override { std::cout << "Derived show" << std::endl; }
};
```

Si show() es llamada a través de un puntero de tipo Base apuntando a un objeto de tipo Derived, se ejecutará la versión de Derived.

7: Una función virtual pura es una función que no tiene implementación en la clase base y se declara con = 0. Hace que la clase se vuelva abstracta.

Ejemplo:

```
class AbstractClass {
public:
    virtual void pureVirtualFunction() = 0; // Declaración de una función virtual pura
};
```

Las clases derivadas deben proporcionar una implementación para la función virtual pura, o también serán abstractas.

8: El polimorfismo dinámico se haceinvocando funciones virtuales a través de punteros o referencias de la clase base que apuntan a objetos de la clase derivada.

Ejemplo:

```
class Base {
public:
    virtual void display() { std::cout << "Base display" << std::endl; }
};

class Derived : public Base {
public:
```

```
    void display() override { std::cout << "Derived display" << std::endl; }
};

int main() {
    Base* basePtr = new Derived();
    basePtr->display(); // Invoca Derived::display() debido al polimorfismo
    delete basePtr;
    return 0;
}
```

9: Una clase abstracta es una clase que tiene al menos una función virtual pura. No se pueden instanciar directamente y se usan como base para otras clases.

Ejemplo:

```
class Abstract {
public:
    virtual void pureFunction() = 0; // Función virtual pura
};

class Concrete : public Abstract {
public:
    void pureFunction() override { std::cout << "Implemented pure function" << std::endl; }
};
```

10: El modificador override se usa al redefinir una función virtual en una clase derivada para indicar explícitamente que se está sobrescribiendo una función de la clase base. Ayuda a evitar errores, como la redefinición accidental con una firma incorrecta.

Ejemplo:

```
class Base {
public:
    virtual void display() const { std::cout << "Base display" << std::endl; }
};

class Derived : public Base {
public:
    void display() const override { std::cout << "Derived display" << std::endl; } // Correcto
};
```

Si se omite override y la firma de la función no coincide exactamente, el compilador no va a marcar como error, y esto puede llevar a comportamientos medios inesperados

B: lo hago abajo.

respuestas de la actividad del libro.

1:

Clase Figura

```
#include <iostream>
#include <cmath>

class Figura {
public:
    virtual double calcularArea() const = 0; // Función virtual pura
    virtual ~Figura() {}
};

class Rectangulo : public Figura {
private:
    double largo, ancho;
public:
    Rectangulo(double l, double a) : largo(l), ancho(a) {}
    double calcularArea() const override {
        return largo * ancho;
    }
};

class Circulo : public Figura {
private:
    double radio;
public:
    Circulo(double r) : radio(r) {}
    double calcularArea() const override {
        return M_PI * radio * radio;
    }
};

int main() {
    Figura* rect = new Rectangulo(5, 3);
    Figura* circ = new Circulo(4);

    std::cout << "Área del rectángulo: " << rect->calcularArea() << std::endl;
    std::cout << "Área del círculo: " << circ->calcularArea() << std::endl;

    delete rect;
    delete circ;
    return 0;
}
```

2. Clase Animal

```
#include <iostream>
```

```
class Animal {
public:
    virtual void hacerSonido() const = 0; // Función virtual pura
    virtual ~Animal() {}
};

class Perro : public Animal {
public:
    void hacerSonido() const override {
        std::cout << "Guau!" << std::endl;
    }
};

class Gato : public Animal {
public:
    void hacerSonido() const override {
        std::cout << "Miau!" << std::endl;
    }
};

class Pato : public Animal {
public:
    void hacerSonido() const override {
        std::cout << "Cuac!" << std::endl;
    }
};

int main() {
    Animal* perro = new Perro();
    Animal* gato = new Gato();
    Animal* pato = new Pato();

    perro->hacerSonido();
    gato->hacerSonido();
    pato->hacerSonido();

    delete perro;
    delete gato;
    delete pato;
    return 0;
}
```

3. Clase Vehículo

```
#include <iostream>

class Vehiculo {
```

```
public:
    virtual void acelerar() const = 0; // Función virtual pura
    virtual ~Vehiculo() {}
};

class Coche : public Vehiculo {
public:
    void acelerar() const override {
        std::cout << "El coche acelera rápidamente." << std::endl;
    }
};

class Moto : public Vehiculo {
public:
    void acelerar() const override {
        std::cout << "La moto acelera más ágilmente." << std::endl;
    }
};

int main() {
    Vehículo* coche = new Coche();
    Vehículo* moto = new Moto();

    coche->acelerar();
    moto->acelerar();

    delete coche;
    delete moto;
    return 0;
}
```

4. Clase Empleado

```
#include <iostream>

class Empleado {
public:
    virtual double calcularSalario() const = 0; // Función virtual pura
    virtual ~Empleado() {}
};

class Gerente : public Empleado {
private:
    double sueldoBase;
public:
    Gerente(double sueldo) : sueldoBase(sueldo) {}
    double calcularSalario() const override {
        return sueldoBase;
```

```
    }
};

class Vendedor : public Empleado {
private:
    double ventas, comision;
public:
    Vendedor(double v, double c) : ventas(v), comision(c) {}
    double calcularSalario() const override {
        return ventas * comision;
    }
};

int main() {
    Empleado* gerente = new Gerente(5000);
    Empleado* vendedor = new Vendedor(10000, 0.1);

    std::cout << "Salario del gerente: $" << gerente->calcularSalario() << std::endl;
    std::cout << "Salario del vendedor: $" << vendedor->calcularSalario() << std::endl;

    delete gerente;
    delete vendedor;
    return 0;
}
```

5. Clase InstrumentoMusical

```
#include <iostream>

class InstrumentoMusical {
public:
    virtual void tocar() const = 0; // Función virtual pura
    virtual ~InstrumentoMusical() {}
};

class Piano : public InstrumentoMusical {
public:
    void tocar() const override {
        std::cout << "El piano suena melodioso." << std::endl;
    }
};

class Guitarra : public InstrumentoMusical {
public:
    void tocar() const override {
        std::cout << "La guitarra toca acordes vibrantes." << std::endl;
    }
};
```

```
class Flauta : public InstrumentoMusical {
public:
    void tocar() const override {
        std::cout << "La flauta produce un sonido suave." << std::endl;
    }
};

int main() {
    InstrumentoMusical* piano = new Piano();
    InstrumentoMusical* guitarra = new Guitarra();
    InstrumentoMusical* flauta = new Flauta();

    piano->tocar();
    guitarra->tocar();
    flauta->tocar();

    delete piano;
    delete guitarra;
    delete flauta;
    return 0;
}
```