

LAB ASSIGNMENTS:

1. Write a C++ program to create a base class Person with attributes name and age. Derive a class Student that adds rollNo. Use constructors to initialize all attributes. Create objects of both classes and display their details to show how Student inherits Person members.

Source Code:

```
#include <iostream>

using namespace std;

class Person{
    public:
        string name;
        int age;
    void input(){
        cout<<"Enter name: ";
        cin>>name;
        cout<<"Enter age: ";
        cin>>age;
    }
};

class Student:public Person{
    public:
        int roll_no;
        void input(){
            Person::input();
            cout<<"Enter roll no: ";
            cin>>roll_no;
        }
        void output(){
```

```

        cout<<"Name: "<<name<<endl<<"Age: "<<age<<endl;
        cout<<"Roll no: "<<roll_no<<endl;
    }
};

int main(){
    Student s;
    s.input();
    s.output();
    return 0;
}

```

Output:

```

Enter name: Suraj
Enter age: 19
Enter roll no: 48
Name: Suraj
Age: 19
Roll no: 48

```

2. Implement a C++ program with a base class Account having a protected attribute balance. Derive a class SavingsAccount that adds an attribute interestRate and a function addInterest() to modify balance. Use user input to initialize attributes and show how the protected balance is accessed in the derived class but not outside.

Source Code:

```

#include <iostream>
#include <string>
using namespace std;
class Account {

```

```

protected:
    double balance;
};

class SavingsAccount : public Account {
public:
    double interestRate;
    void initialize() {
        cout << "Enter initial balance: ";
        cin >> balance;
        cout << "Enter interest rate (e.g., 0.05 for 5%): ";
        cin >> interestRate;
    }
    void addInterest() {
        balance += balance * interestRate;
    }
    void displayBalance() {
        cout << "Current balance: " << balance << endl;
    }
};

int main() {
    SavingsAccount sa;
    sa.initialize();
    cout << "Balance before interest:" << endl;
    sa.displayBalance();
    sa.addInterest();
    cout << "Balance after interest:" << endl;
    sa.displayBalance();
}

```

```
    return 0;
}
```

Output:

```
Enter initial balance: 5000
Enter interest rate (e.g., 0.05 for 5%): 0.04
Balance before interest:
Current balance: 5000
Balance after interest:
Current balance: 5200
```

3. Write a C++ program with a base class Shape having a function draw(). Declare a derived class Circle with an attribute radius initialized via user input. Create a Circle object and call draw() to display a message including radius, demonstrating proper derived class declaration.

Source Code:

```
#include <iostream>

using namespace std;

class Shape{
    public:
    virtual void draw(){
        cout<<"Drawing Shape"<<endl;
    }
};

class Circle:public Shape{
    public:
    void draw(){
        cout<<"Drawing Circle"<<endl;
    }
};

class Square:public Shape{
    public:
```

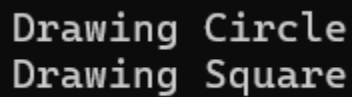
```

void draw(){
    cout<<"Drawing Square"<<endl;
}
};

int main(){
    Shape *s;
    Circle c;
    Square sq;
    s=&c;
    s->draw();
    s=&sq;
    s->draw();
    return 0;
}

```

Output:



```

Drawing Circle
Drawing Square

```

4. Create a C++ program with a base class Vehicle having a function move(). Derive a class Car that overrides move() to indicate driving. Use a base class pointer to call move() on a Car object initialized with user input for attributes like brand. Show that Car is a Vehicle.

Source Code:

```

#include <iostream>
#include <string>
using namespace std;
class Vehicle {
public:

```

```
    virtual void move() {  
        cout << "Vehicle is moving." << endl;  
    }  
};  
  
class Car : public Vehicle {  
private:  
    string brand;  
public:  
    Car(string b) : brand(b) {}  
    void move() override {  
        cout << "The " << brand << " car is driving." << endl;  
    }  
};  
  
int main() {  
    string carBrand;  
    cout << "Enter the car brand: ";  
    getline(cin, carBrand);  
    Car myCar(carBrand);  
    Vehicle* vehiclePtr = &myCar;  
    vehiclePtr->move();  
    return 0;  
}
```

Output:

```
Enter the car brand: BMW  
The BMW car is driving.
```

5. Implement a C++ program with a class Engine having an attribute horsepower. Create a class Car that contains an Engine object (composition) and an attribute model. Initialize all attributes with user input and display details to show that Car has an Engine.

Source Code:

```
#include <iostream>
#include <string>
using namespace std;
class Engine {
public:
    int horsepower;
    Engine(int hp = 0) {
        horsepower = hp;
    }
};
class Car {
public:
    string model;
    Engine engine;
    Car(string m, int hp) : model(m), engine(hp) {}
    void displayDetails() {
        cout << "\n--- Car Details ---" << endl;
        cout << "Model: " << model << endl;
        cout << "Engine Horsepower: " << engine.horsepower << " HP" << endl;
    }
};
int main() {
    string carModel;
```

```

int engineHp;

cout << "Enter the car model: ";

getline(cin, carModel);

cout << "Enter the engine horsepower: ";

cin >> engineHp;

Car myCar(carModel, engineHp);

myCar.displayDetails();

return 0;

}

```

Output:

```

Enter the car model: BMW Sedan
Enter the engine horsepower: 630

--- Car Details ---
Model: BMW Sedan
Engine Horsepower: 630 HP

```

6. Write a C++ program with a base class Base having public, protected, and private attributes (e.g., pubVar, protVar, privVar). Derive three classes using public, protected, and private inheritance, respectively. Demonstrate with user-initialized objects how each inheritance type affects access to base class members.

Source Code:

```

#include <iostream>

using namespace std;

class Base{

private:

    int privateVar;

```


protected:

int protectedVar;

public:

int publicVar;

};

class PublicClass:public Base{

public:

void SetVal(){

publicVar=50;

protectedVar=60;

// privateVar = 1; // private member of base class is not accessible

}

void Display(){

cout<<"Protected variable"<<protectedVar<<" "<<"Public Variable"<<publicVar<<endl;

}};

class ProtectedClass:protected Base{

public:

void SetVal(){

publicVar=50;

protectedVar=60;

// privateVar = 2; // private member of base class is not accessible

}

void Display(){

cout<<"Protected variable as protected: "<<protectedVar<<" "<<"Public Variable as protected: "<<publicVar<<endl;

}};

class PrivateClass:private Base{

```

    public:
void SetVal(){
    publicVar=50;
    protectedVar=60;
    // privateVar = 3; // private member of base class is not accessible
}

void Display(){
    cout<<"Protected variableas private: "<<protectedVar<<" "<<"Public Variable as private:
"<<publicVar<<endl;
    };
int main()
{
    PublicClass P1;
    PrivateClass P2;
    ProtectedClass P3;
    P1.SetVal();
    P2.SetVal();
    P3.SetVal();
    P1.Display();
    P2.Display();
    P3.Display();
    // Accessing public member of PublicClass from main
    P1.publicVar = 100;

    cout << "Accessed and modified public member of PublicClass from main: " << P1.publicVar
<< endl;

    // P2.publicVar = 200; // Not accessible, publicVar is private in PrivateClass
    // P3.publicVar = 300; // Not accessible, publicVar is protected in ProtectedClass

```

```
    return 0;
}
```

Output:

```
Protected variable60 Public Variable50
Protected variableas private: 60 Public Variable as private: 50
Protected variable as protected: 60 Public Variable as protected: 50
Accessed and modified public member of PublicClass from main: 100
```

7. Create a C++ program with a base class Animal having a virtual function sound(). Derive classes Dog and Cat that override sound() to print specific sounds. Use a base class pointer array to call sound() on Dog and Cat objects created with user input, showing runtime polymorphism.

Source Code:

```
#include<iostream>

using namespace std;

class Animal{
protected:
    string name;
public:
    virtual void Sound(){
        cout<<name<<" makes a sound."<<endl;
    }
};

class Dog:public Animal{
public:
    Dog(string s){
        name=s;
    }

    void Sound(){
        cout<<name<<" Barks."<<endl;}
};
```

```

class Cat:public Animal{
public:
    Cat (string s){
        name=s;
    }
    void Sound(){
        cout<<name<<" Meows."<<endl;}
};

int main(){
    string name;
    Animal *animal[2];
    cout<<"Enter name of Dog: ";
    cin>>name;
    animal[0]= new Dog(name);
    cout<<"Enter name of Cat: ";
    cin>>name;
    animal[1]= new Cat(name);
    for(int i=0;i<2;i++){
        animal[i]->Sound();
        cout<<endl;}
    return 0;
}

```

Output:

```

Enter name of Dog: Tommy
Enter name of Cat: Mini
Tommy  Barks.

Mini  Meows.

```

8. Write a C++ program with two base classes Battery and Screen, each with a function showStatus(). Derive a class Smartphone that inherits from both. Resolve ambiguity when calling showStatus() using the scope resolution operator. Initialize attributes with user input and display details.

Source Code:

```
#include<iostream>

using namespace std;

class Battery{
    int Charge;
public:
    void SetCharge(int c){
        Charge=c;
    }
    void ShowStatus(){
        cout<<"Battery Charge:"<<Charge<<endl;
    }
};

class Screen{
    float Size;
public:
    void SetSize(float s){
        Size=s;
    }
    void ShowStatus(){
        cout<<"Screen Size:"<<Size<<"inches"<<endl;
    }
};

class Smartphone:public Battery,public Screen{
```

```
private:
    string model;
public:
    void SetName(string n){
        model=n;
    }
    void Display(){
        cout<<"Phone Model:"<<model<<endl;
        Battery::ShowStatus();
        Screen::ShowStatus();
    }
};

int main(){
    Smartphone S;

    int c;
    float s;
    string n;
    cout<<"Enter Phone Model:";
    cin>>n;
    S.SetName(n);
    cout<<"Enter Battery Status:";
    cin>>c;
    S.SetCharge(c);
    cout<<"Enter Screen Size in inches:";
    cin>>s;
    S.SetSize(s);
    cout<<"---Phone Details---"<<endl;
```

```
S.Display();  
return 0;  
}
```

Output:

```
Enter Phone Model:S21  
Enter Battery Status:89  
Enter Screen Size in inches:6.2  
---Phone Details---  
Phone Model:S21  
Battery Charge:89  
Screen Size:6.2inches
```

9. Implement a C++ program with a base class Person having a parameterized constructor for name and age. Derive a class Employee with an additional attribute employeeID. Use user input to initialize all attributes and show the order of constructor invocation when creating an Employee object.

Source Code:

```
#include<iostream>  
  
#include<string>  
  
using namespace std;  
  
class Person{  
protected:  
    string name;  
    int age;  
public:  
    Person(string n,int a) : name(n), age(a) {  
        cout << "Person Class Called" << endl;  
    }  
    void setName(string n){
```

```

        name=n;
    }
    void setAge(int a){
        age=a;
    }
};

class Employee:protected Person{
protected:
    int employeeID;
public:
    Employee(string n,int a,int id):Person(n,a){
        cout << "Employee Class called." << endl;
        employeeID=id;
    }

    void Display(){
        cout << "Employee Name: " << name << ", Employee ID: " << employeeID << ", Age: "
<< age << endl;
    }
};

int main(){
    string n;
    int id,age;
    cout<<"Enter the name:";
    getline(cin,n);

    cout<<"Enter the age:";

```



```

cin>>age;

cout<<"Enter the ID:";

cin>>id;

Employee e(n,age,id);

e.Display();

return 0;

}

```

Output:

```

Enter the name:Suraj
Enter the age:19
Enter the ID:48
Person Class Called
Employee Class called.
Employee Name: Suraj, Employee ID: 48, Age: 19

```

10. Write a C++ program with a base class Shape and a derived class Rectangle, both with destructors that print messages. Make the base class destructor virtual. Create a Rectangle object through a base class pointer using user input for attributes, and delete it to show proper destructor invocation. Compare with a non-virtual destructor case.

Source Code:

```

#include <iostream>

using namespace std;

class Shape {
public:
    Shape() {}

    virtual ~Shape() {
        cout << "Shape Destructor Called" << endl;
    }
};

```

```
class Rectangle : public Shape {  
    int width, height;  
public:  
    Rectangle(int  $w$ , int  $h$ ) : width( $w$ ), height( $h$ ) {}  
    ~Rectangle() {  
        cout << "Rectangle Destructor Called" << endl;  
    }  
};
```

```
class Shape2 {  
public:  
    Shape2() {}  
    ~Shape2() {  
        cout << "Shape2 Destructor Called" << endl;  
    }  
};
```

```
class Rectangle2 : public Shape2 {  
    int width, height;  
public:  
    Rectangle2(int  $w$ , int  $h$ ) : width( $w$ ), height( $h$ ) {}  
    ~Rectangle2() {  
        cout << "Rectangle2 Destructor Called" << endl;  
    }  
};
```

```

int main() {
    int w, h;
    cout << "Enter width: ";
    cin >> w;
    cout << "Enter height: ";
    cin >> h;
    Shape* s = new Rectangle(w, h);
    delete s;
    Shape2* s2 = new Rectangle2(w, h);
    delete s2;
    return 0;
}

```

Output:

```

Enter width: 2
Enter height: 4
Rectangle Destructor Called
Shape Destructor Called
Shape2 Destructor Called

```

11. Create a C++ program with a base class A having an attribute value. Derive classes B and C from A, and derive class D from both B and C. Use virtual inheritance to avoid duplication of A's members. Initialize value with user input and display it from D to show ambiguity resolution.

Source Code:

```

#include <iostream>

using namespace std;

class A {
protected:
    int value;

```

```
public:
    A() {
        cout << "Enter a value: ";
        cin >> value;
        cout << "Class A constructor called" << endl;
    }
    void displayValue() {
        cout << "Value from class A: " << value << endl;
    }
};

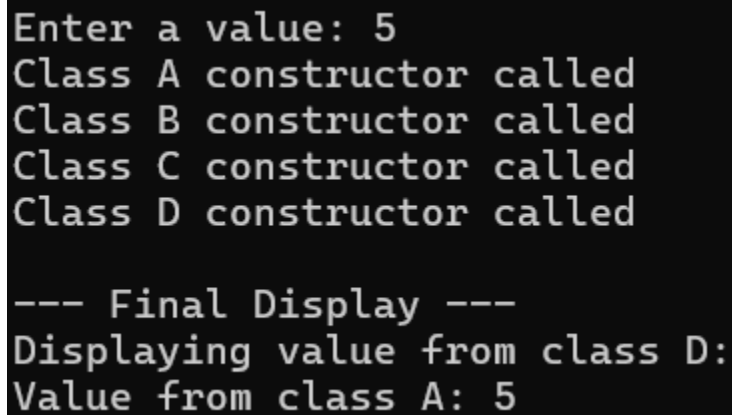
class B : virtual public A {
public:
    B() {
        cout << "Class B constructor called" << endl;
    }
};

class C : virtual public A {
public:
    C() {
        cout << "Class C constructor called" << endl;
    }
};

class D : public B, public C {
public:
    D() {
        cout << "Class D constructor called" << endl;
    }
};
```

```
void showValue() {  
    cout << "Displaying value from class D:" << endl;  
    displayValue(); // No ambiguity due to virtual inheritance  
}  
};  
  
int main() {  
    D obj;  
    cout << "\n--- Final Display ---" << endl;  
    obj.showValue();  
    return 0;  
}
```

Output:

A screenshot of a terminal window with a black background and white text. The output shows the sequence of constructor calls and the final display of values.

```
Enter a value: 5  
Class A constructor called  
Class B constructor called  
Class C constructor called  
Class D constructor called  
  
--- Final Display ---  
Displaying value from class D:  
Value from class A: 5
```

DISCUSSION

In this lab we were able to understand the concept of Inheritance in C++ language, we were able to do various inheritance using programming where we did Multiple, Multilevel, Hierarchical and diamond Inheritance. These practical examples helped us understand how classes can inherit properties and behaviors from other classes, enhancing code reusability and demonstrating the power of object-oriented programming in C++.

CONCLUSION

In this lab, we learned how one class in C++ can get features from another class using something called inheritance. We tried different types like multiple, multilevel, hierarchical, and diamond inheritance. By doing this, we saw how it helps us write less code and keep our programs neat and easy to understand. It was helpful in clarifying the core concept of OOP i.e. Inheritance.