## Program #1

**Basic Inheritance:** Define a base class **Shape** with members to store the dimensions of a shape and a derived class **Circle** that inherits from **Shape**. Add a method to calculate the area of the circle.

#include <iostream>

using namespace std;

class Shape {

protected:

double radius;

public:

Shape (double r): radius(r) {}

};

class Circle: public Shape {

public:

Circle (double r): Shape(r) {}

double calculateArea () {

return 3.14159 \* radius \* radius; // Simple approximation of pi

}

};

int main () {

double radius;

cout << "Enter the radius of the circle: ";

cin >> radius;

Circle myCircle(radius);

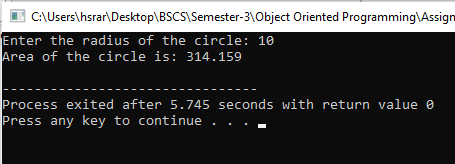
double area = myCircle.calculateArea();

cout << "Area of the circle is: " << area << endl;

return 0;

}

**Output:**

****

## Program #2

**Multiple Inheritance:** Create a base class **Person** with attributes **name** and **age**. Create a derived class **Student** which inherits from **Person** and has additional attributes **studentId** and **grade**. Implement appropriate methods.

#include<iostream>

using namespace std;

class person{

public:

int age;

string name;

void display(){

cout<<"Name:"<<name<<endl;

cout<<"Age:"<<age<<endl;

}

person(string n, int a){

name=n;

age=a;

}

};

class student:public person{

public:

int id;

double grade;

void show(){

person::display();

cout<<"Student i'd:"<<id<<endl;

cout<<"Grade:"<<grade<<endl;

}

student(string name, int age, int id, double grade)

: person(name, age), id(id), grade(grade) {}

};

int main(){

person p1=person("Sarah",17);

cout<<"Person's Information"<<endl;

p1.display();

student s1("Sham",18,50 , 94.30);

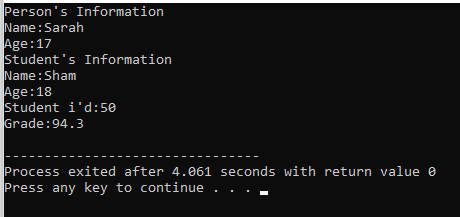
cout<<"Student's Information"<<endl;

s1.show();

return 0;

}

**Output:**

****

## Program #3

**Hierarchical Inheritance:** Create a base class **Animal** with a method **makeSound()**. Derive two classes **Dog** and **Cat** from **Animal** and override the **makeSound()** method to represent the sound each animal makes.

**#include<iostream>**

**using namespace std;**

**class Animal{**

**public:**

**virtual void makeSound(){**

**cout<<"General Animal's sounds:"<<endl;**

**}**

**};**

**class Dog:public Animal{**

**public:**

**void makeSound()override{**

**cout<<"Woof! Woof!"<<endl;**

**}**

**};**

**class Cat:public Animal{**

**public:**

**void makeSound()override{**

**cout<<"Meow! Meow!"<<endl;**

**}**

**};**

**int main() {**

**Animal\*dptr= new Dog(); //dptr is a pointer for class dog**

**Animal\*cptr= new Cat(); //cptr is a pointer for class cat**

**cout << "Animal 1 says: ";**

**dptr->makeSound();**

**cout << "Animal 2 says: ";**

**cptr->makeSound();**

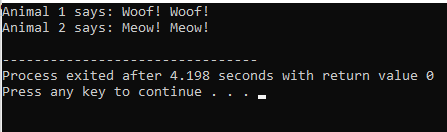
**delete dptr;**

**delete cptr;**

**return 0;**

**}**

**Output:**

****

## Program #4

**Virtual Functions:** Define a base class **Shape** with a virtual function **calculateArea()**. Derive classes **Circle** and **Rectangle** from **Shape** and override **calculateArea()** for each shape.

#include<iostream>

using namespace std;

class Base{

public:

virtual void display(){

cout<<"You are in Base Class."<<endl;

}

};

class Derive:public Base{

public:

void display()override{

cout<<"You are in Derive Class."<<endl;

}

};

int main(){

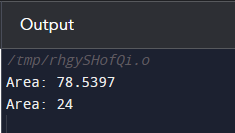
Base \*bptr=new Derive();

bptr->display();

return 0;

}

**Output:**

****

## Program #5

**Pure Virtual Functions:** Create an abstract base class **BankAccount** with a pure virtual function **calculateInterest()**. Derive two classes **SavingsAccount** and **CurrentAccount** from **BankAccount** and implement **calculateInterest()** accordingly.

#include <iostream>

using namespace std;

class BankAccount {

public:

virtual void calculateInterest() = 0; // Pure virtual function

};

class SavingsAccount : public BankAccount {

public:

void calculateInterest() override {

// Implement the interest calculation for a savings account here

cout << "Interest for Savings Account calculated." <<endl;

}

};

class CurrentAccount : public BankAccount {

public:

void calculateInterest() override {

// Implement the interest calculation for a current account here

cout << "Interest for Current Account calculated." <<endl;

}

};

int main() {

SavingsAccount obj1;

CurrentAccount obj2;

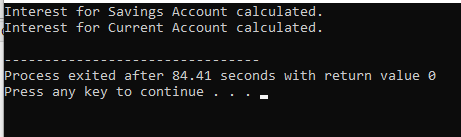
obj1.calculateInterest();

obj2.calculateInterest();

return 0;

}

**Output:**

****

## Program #6

**Access Control:** Define a base class **Base** and a derived class **Derived**. Test the accessibility of different members (public, private, protected) from the derived class.

#include <iostream>

class Base {

public:

int publicMemberBase;

Base() : publicMemberBase(1), protectedMemberBase(2), privateMemberBase(3) {}

void publicFunctionBase() {

std::cout << "Public function in the Base class" << std::endl;

}

void accessMembersInBase() {

std::cout << "Accessing members in the Base class:" << std::endl;

std::cout << "Public Member (publicMemberBase): " << publicMemberBase << std::endl;

std::cout << "Protected Member (protectedMemberBase): " << protectedMemberBase << std::endl;

std::cout << "Private Member (privateMemberBase): " << privateMemberBase << std::endl;

}

protected:

int protectedMemberBase;

private:

int privateMemberBase;

};

class Derived : public Base {

public:

void accessMembersInDerived() {

std::cout << "Accessing members in the Derived class:" << std::endl;

std::cout << "Public Member (publicMemberBase): " << publicMemberBase << std::endl;

std::cout << "Protected Member (protectedMemberBase): " << protectedMemberBase << std::endl;

// Private member from the base class is not accessible in the derived class.

// std::cout << "Private Member (privateMemberBase): " << privateMemberBase << std::endl;

}

void callFunctionInBase() {

publicFunctionBase(); // Public member function from the base class is accessible.

}

};

int main() {

Base baseObject;

Derived derivedObject;

std::cout << "Testing access from the base class:" << std::endl;

baseObject.accessMembersInBase();

std::cout << "\nTesting access from the derived class:" << std::endl;

derivedObject.accessMembersInDerived();

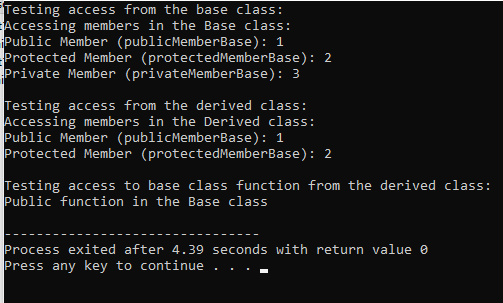
std::cout << "\nTesting access to base class function from the derived class:" << std::endl;

derivedObject.callFunctionInBase();

return 0;

}

**Output:**

****

## Program #7

**Constructor Inheritance:** Create a base class **Vehicle** with attributes like speed and a derived class **Car** that inherits from **Vehicle**. Implement constructors for both classes and display the speed of the car.

**#include<iostream>**

**using namespace std;**

**class Vehicle{**

**public:**

**int speed;**

**public:**

**void display(){**

**cout<<"Speed:"<<speed<<endl;**

**}**

**Vehicle(int s):speed(s){}**

**};**

**class Car:public Vehicle{**

**public:**

**string model, color;**

**void display(){**

**Vehicle::display();**

**cout<<"Model:"<<model<<endl;**

**cout<<"Color:"<<color<<endl;**

**}**

**Car(string m, int s, string c):Vehicle(speed), model(m), color(c){}**

**};**

**int main(){**

**Vehicle v1=Vehicle(55);**

**v1.display();**

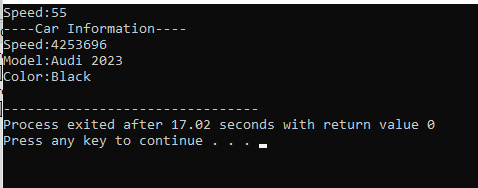
**cout<<"----Car Information----"<<endl;**

**Car c1= Car("Audi 2023",24,"Black");**

**c1.display();**

**}**

**Output:**

****

## Program #8

**Function Overriding:** Create a base class **Shape** with a method **draw()**. Derive classes **Circle**, **Square**, and **Triangle** from **Shape** and override **draw()** to display the shape name.

#include<iostream>

using namespace std;

class Shape{

public:

virtual void draw(){

cout<<"A mathematical Shape."<<endl;

}

};

class Circle:public Shape{

public:

void draw(){

cout<<"I'm a Circle"<<endl;

}

};

class Triangle:public Shape{

public:

void draw(){

cout<<"I'm a Triangle."<<endl;

}

};

int main(){

Shape \*cptr= new Circle();

Shape \*tptr= new Triangle();

cout<<"Shape 1 says:";

cptr->draw();

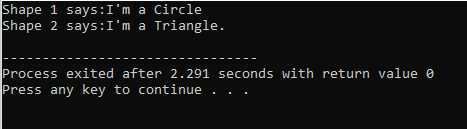
cout<<"Shape 2 says:";

tptr->draw();

return 0;

}

**Output:**

****

## Program #9

**Polymorphism:** Define a base class **Animal** with a virtual function **makeSound()**. Derive classes **Dog**, **Cat**, and **Cow** from **Animal** and implement **makeSound()** to display the sound each animal makes.

#include<iostream>

using namespace std;

class Animal{

public:

virtual void makeSound(){

cout<<"Animal's Voice...";

}

};

class Dog:public Animal{

public:

void makeSound(){

cout<<"Woof! Woof!"<<endl;

}

};

class Cat:public Animal{

public:

void makeSound(){

cout<<"Meow! Meow!"<<endl;

}

};

class Cow:public Animal{

public:

void makeSound(){

cout<<"Gaa! Gaa!"<<endl;

}

};

int main(){

Animal \*dptr= new Dog();

Animal \*cptr= new Cat();

Animal \*aptr= new Cow();

dptr->makeSound();

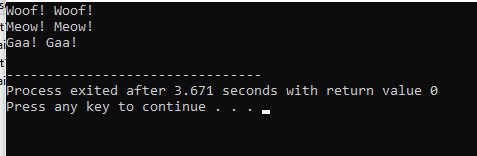
cptr->makeSound();

aptr->makeSound();

return 0;

}

**Output:**



## Program #10

**Dynamic Binding:** Define a base class **Base** with a virtual function **display()**. Derive a class **Derived** from **Base** and override **display()**. Create objects of both classes and call the **display()** method to observe dynamic binding.

#include<iostream>

using namespace std;

class Base{

public:

virtual void display(){

cout<<"You are in Base Class."<<endl;

}

};

class Derive:public Base{

public:

void display()override{

cout<<"You are in Derive Class."<<endl;

}

};

int main(){

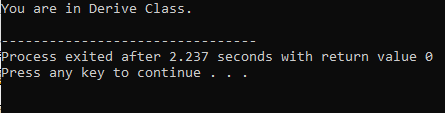
Base \*bptr=new Derive();

bptr->display();

return 0;

}

**Output:**

****