## Practical 1

### AIM: WAP to print “Welcome to C++”.

#include <iostream>

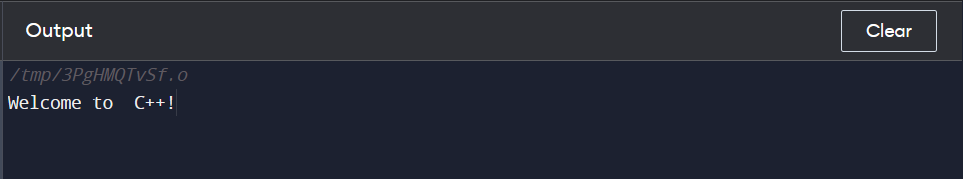
int main() {

std::cout << "Welcome to C++!";

return 0;

}

Output:



### AIM: WAP to compute quotient & remainder (Take dividend & divisor input from the user).

#include <iostream>

using namespace std;

int main() {

int dividend, divisor, quotient, remainder;

cout << "Enter the dividend: ";

cin >> dividend;

cout << "Enter the divisor: ";

cin >> divisor;

quotient = dividend / divisor;

remainder = dividend % divisor;

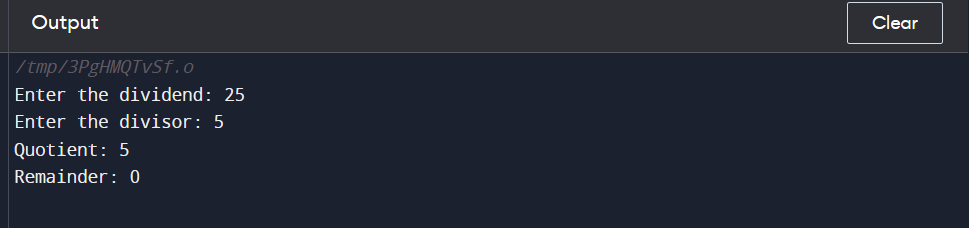
cout << "Quotient: " << quotient << endl;

cout << "Remainder: " << remainder << endl;

return 0;

}

Output:



### AIM: WAP to reverse a number.

#include <iostream>

using namespace std;

int main() {

int num, reverse = 0;

cout << "Enter a number: ";

cin >> num;

while(num > 0) {

int digit = num % 10;

reverse = (reverse \* 10) + digit;

num /= 10;

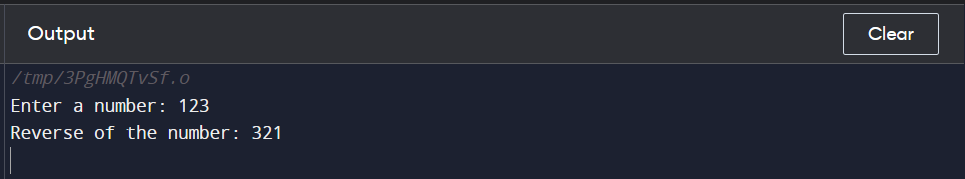
}

cout << "Reverse of the number: " << reverse << endl;

return 0;

}

Output:



### AIM: WAP to swap two numbers

#include <iostream>

using namespace std;

int main() {

int num1, num2;

cout << "Enter two numbers: ";

cin >> num1 >> num2;

cout << "Before swapping: num1 = " << num1 << ", num2 = " << num2 << endl;

int temp = num1;

num1 = num2;

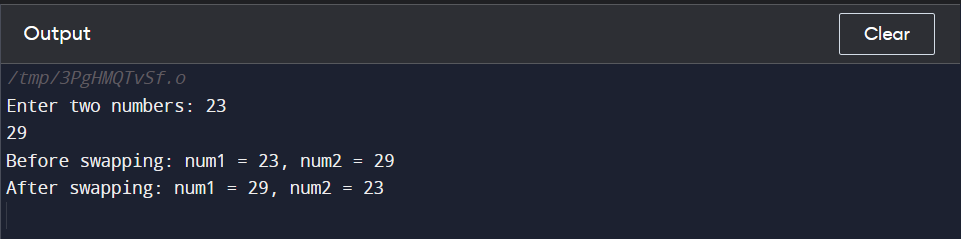
num2 = temp;

cout << "After swapping: num1 = " << num1 << ", num2 = " << num2 << endl;

return 0;

}

Output:



## Practical 2

### AIM: WAP to check whether a character is vowel or consonant.

#include <iostream>

using namespace std;

int main() {

char ch;

cout << "Enter a character: ";

cin >> ch;

if(ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u' ||

ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U') {

cout << ch << " is a vowel." << endl;

} else {

cout << ch << " is a consonant." << endl;

}

return 0;

}

Output:



### AIM: WAP to find factorial of number.

#include <iostream>

using namespace std;

int factorial(int num) {

if (num == 0)

return 1;

else

return num \* factorial(num - 1);

}

int main() {

int num;

cout << "Enter a number: ";

cin >> num;

if (num < 0)

cout << "Error: Factorial of negative numbers is undefined." << endl;

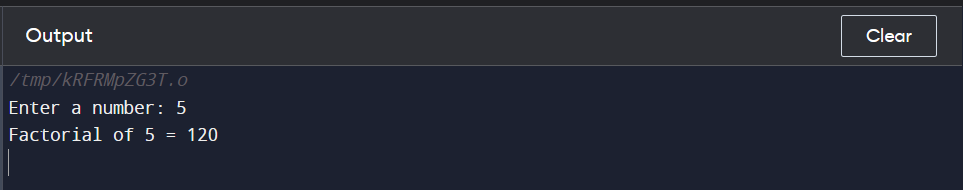
else

cout << "Factorial of " << num << " = " << factorial(num) << endl;

return 0;

}

Output:



### AIM: WAP to check whether a number is palindrome or not.

#include <iostream>

using namespace std;

int reverse(int num) {

int reversedNum = 0, remainder;

while (num != 0) {

remainder = num % 10;

reversedNum = reversedNum \* 10 + remainder;

num /= 10;

}

return reversedNum;

}

bool isPalindrome(int num) {

return num == reverse(num);

}

int main() {

int num;

cout << "Enter a number: ";

cin >> num;

if (isPalindrome(num))

cout << num << " is a palindrome number." << endl;

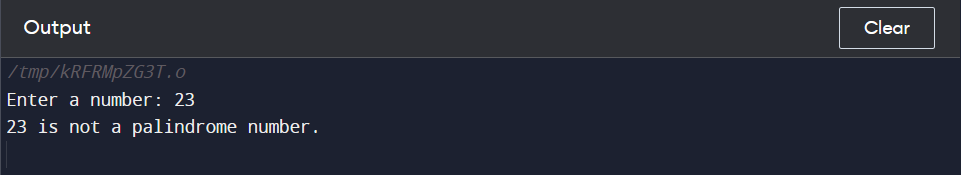
else

cout << num << " is not a palindrome number." << endl;

return 0;

}

Output:



### AIM: WAP to find largest number among three number.

#include <iostream>

using namespace std;

int main() {

int num1, num2, num3, largest;

cout << "Enter three numbers: ";

cin >> num1 >> num2 >> num3;

largest = num1;

if (num2 > largest) {

largest = num2;

}

if (num3 > largest) {

largest = num3;

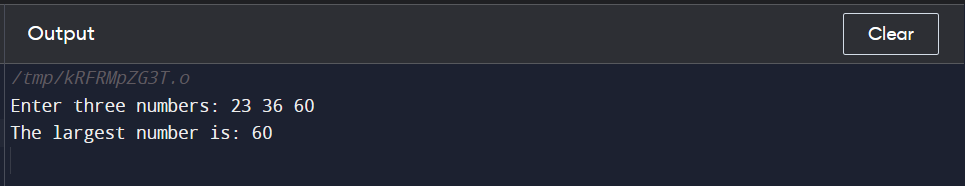
}

cout << "The largest number is: " << largest << endl;

return 0;

}

Output:



## Practical 3

### AIM: WAP to implement function call by value and call by reference.

#include <iostream>

using namespace std;

// Call by value function

void swapByValue(int num1, int num2) {

int temp = num1;

num1 = num2;

num2 = temp;

cout << "Inside swapByValue function: num1 = " << num1 << ", num2 = " << num2 << endl;

}

// Call by reference function

void swapByRef(int& num1, int& num2) {

int temp = num1;

num1 = num2;

num2 = temp;

cout << "Inside swapByRef function: num1 = " << num1 << ", num2 = " << num2 << endl;

}

int main() {

int num1 = 10, num2 = 20;

// Call by value

cout << "Before swapByValue function call: num1 = " << num1 << ", num2 = " << num2 << endl;

swapByValue(num1, num2);

cout << "After swapByValue function call: num1 = " << num1 << ", num2 = " << num2 << endl;

// Call by reference

cout << "Before swapByRef function call: num1 = " << num1 << ", num2 = " << num2 << endl;

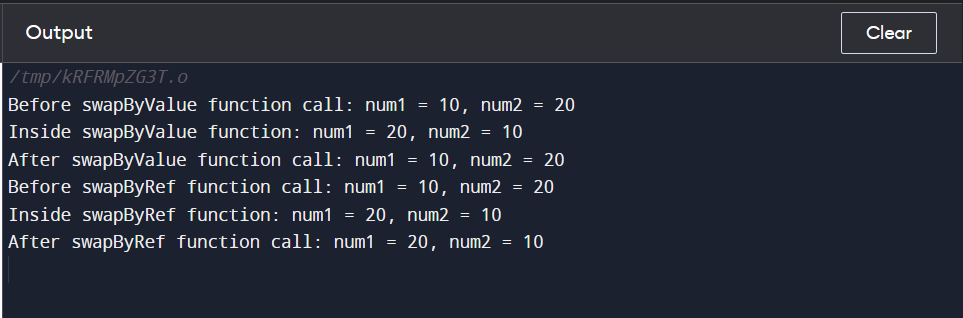
swapByRef(num1, num2);

cout << "After swapByRef function call: num1 = " << num1 << ", num2 = " << num2 << endl;

return 0;

}

Output:



## Practical 4

### AIM: WAP for Illustrating Class Declarations, Definition, and Accessing Class Members (variables & functions).

#include <iostream>

using namespace std;

// Class declaration

class Rectangle {

public:

// Public member variables

double length;

double width;

// Public member functions

double area();

double perimeter();

};

// Class definition

double Rectangle::area() {

return length \* width;

}

double Rectangle::perimeter() {

return 2 \* (length + width);

}

int main() {

// Create an object of the Rectangle class

Rectangle rect;

// Assign values to the member variables

rect.length = 10;

rect.width = 5;

// Access and display the area and perimeter of the rectangle

cout << "Length = " << rect.length << endl;

cout << "Width = " << rect.width << endl;

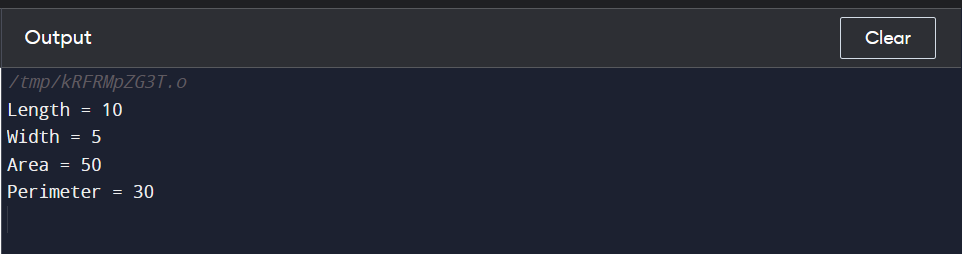
cout << "Area = " << rect.area() << endl;

cout << "Perimeter = " << rect.perimeter() << endl;

return 0;

}

Output:



## Practical 5

AIM: WAP to illustrate default constructor parameterized constructor copy constructor (implicit and explicit) and destructor.

#include <iostream>

#include <string>

using namespace std;

class Person {

private:

string name;

int age;

public:

// Default constructor

Person() {

name = "";

age = 0;

}

// Parameterized constructor

Person(string n, int a) {

name = n;

age = a;

}

// Copy constructor

Person(const Person& p) {

cout << "Using explicit copy constructor" << endl;

name = p.name;

age = p.age;

}

// Destructor

~Person() {

cout << "Destructor called for " << name << endl;

}

void display() {

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

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

}

};

int main() {

Person p1("John", 25);

p1.display();

Person p2 = p1; // Implicit copy constructor

p2.display();

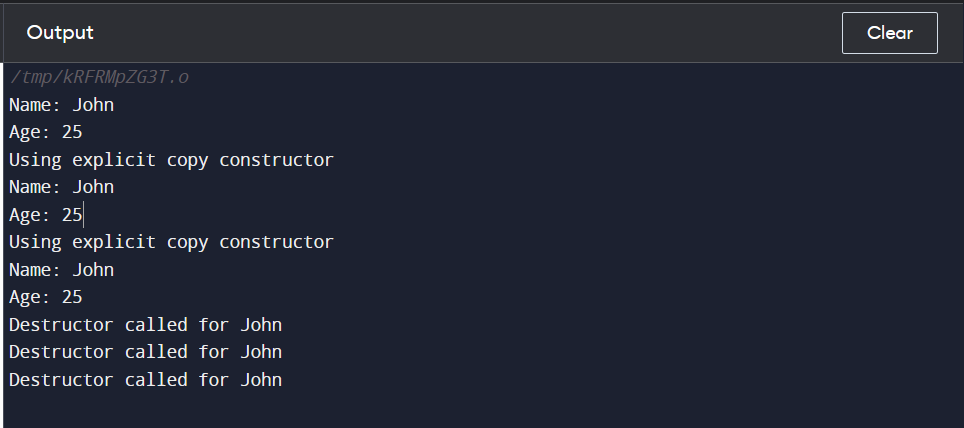
Person p3(p1); // Explicit copy constructor

p3.display();

return 0;

}

Output:



## Practical 6

### AIM: WAP to illustrate the usage of this operator.

#include <iostream>

using namespace std;

class MyClass {

private:

int x;

public:

MyClass(int x) {

this->x = x;

}

void print() {

cout << "x = " << this->x << endl;

}

void setX(int x) {

this->x = x;

}

};

int main() {

MyClass obj(5);

obj.print();

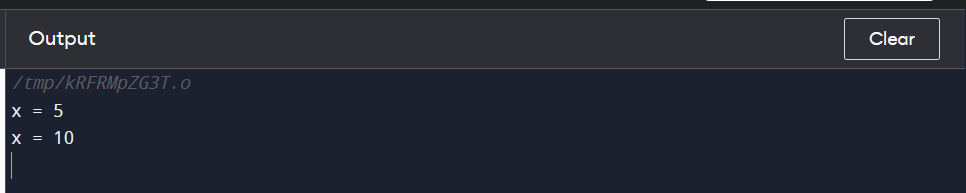
obj.setX(10);

obj.print();

return 0;

}

Output:



## Practical 7

### WAP to illustrate all significance of scope resolution operator.

#include <iostream>

using namespace std;

int x = 10; // global variable

class MyClass {

public:

static int y; // static variable

int x = 20; // member variable

static void printY() { // static function

cout << "y = " << y << endl;

}

void printX() { // member function

cout << "x = " << x << endl;

}

};

int MyClass::y = 30; // static variable definition

int main() {

int x = 40; // local variable

cout << "Global variable x = " << ::x << endl;

cout << "Local variable x = " << x << endl;

cout << "Static variable y = " << MyClass::y << endl;

MyClass::printY();

MyClass obj;

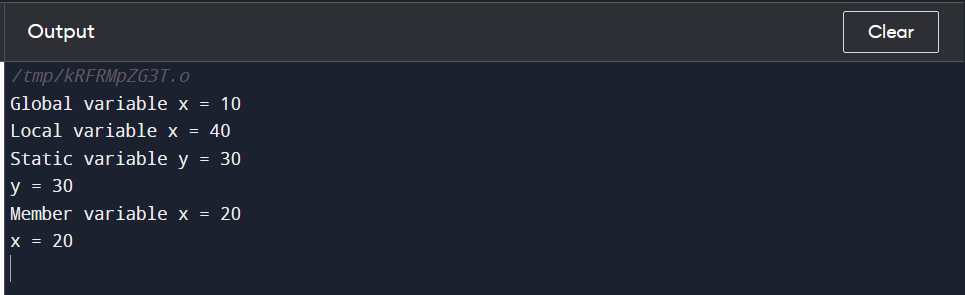
cout << "Member variable x = " << obj.x << endl;

obj.printX();

return 0;

}

Output:



## Practical 8

### AIM: WAP to illustrate single inheritance.

#include <iostream>

using namespace std;

// base class

class Shape {

protected:

int width;

int height;

public:

void setWidth(int w) {

width = w;

}

void setHeight(int h) {

height = h;

}

};

// derived class

class Rectangle : public Shape {

public:

int getArea() {

return width \* height;

}

};

int main() {

Rectangle rect;

rect.setWidth(5);

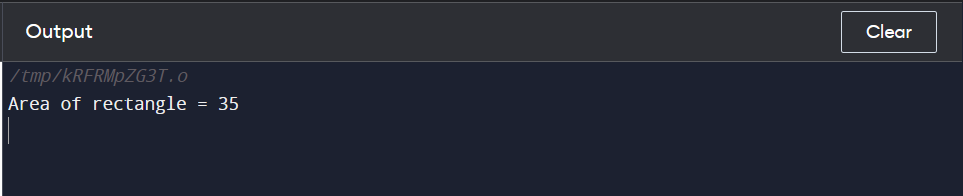
rect.setHeight(7);

cout << "Area of rectangle = " << rect.getArea() << endl;

return 0;

}

Output:



### AIM: WAP to illustrate multiple inheritance.

#include <iostream>

using namespace std;

class A {

public:

void display() {

cout << "This is class A." << endl;

}

};

class B {

public:

void display() {

cout << "This is class B." << endl;

}

};

// derived class C that inherits from A and B

class C : public A, public B {

public:

void show() {

// resolve ambiguity by specifying which display function to use

A::display();

B::display();

}

};

int main() {

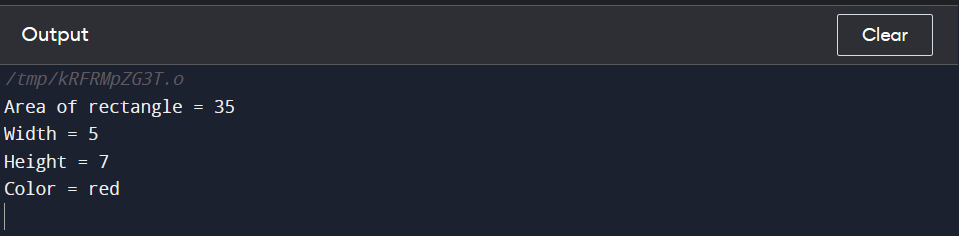
C c;

c.show();

return 0;

}

Output:



### AIM: WAP to illustrate multi level inheritance.

#include <iostream>

using namespace std;

// base class

class Animal {

public:

void eat() {

cout << "Eating..." << endl;

}

};

// derived class

class Dog : public Animal {

public:

void bark() {

cout << "Barking..." << endl;

}

};

// derived class from Dog

class Bulldog : public Dog {

public:

void run() {

cout << "Running..." << endl;

}

};

int main() {

Bulldog b;

b.eat();

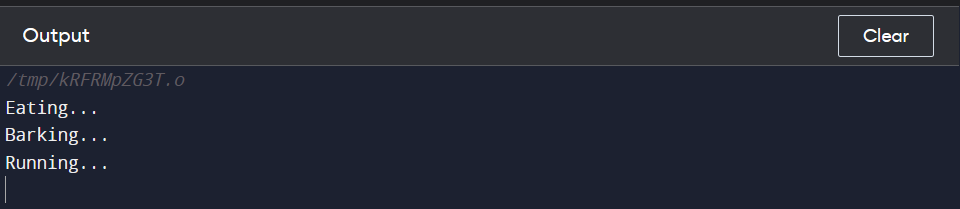
b.bark();

b.run();

return 0;

}

Output:



### AIM: WAP to illustrate hierarchical inheritance.

#include <iostream>

using namespace std;

// base class

class Animal {

public:

void eat() {

cout << "Eating..." << endl;

}

};

// derived class 1

class Dog : public Animal {

public:

void bark() {

cout << "Barking..." << endl;

}

};

// derived class 2

class Cat : public Animal {

public:

void meow() {

cout << "Meowing..." << endl;

}

};

int main() {

Dog d;

Cat c;

d.eat();

d.bark();

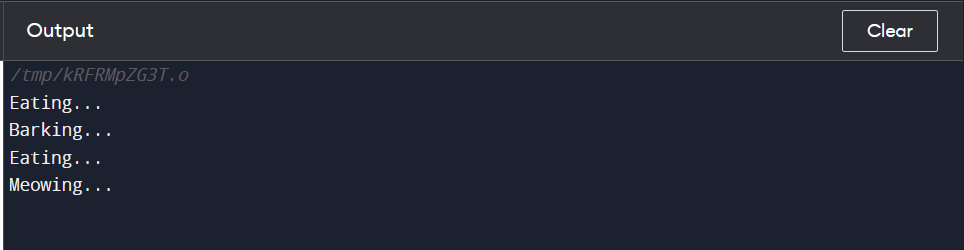
c.eat();

c.meow();

return 0;

}

Output:



AIM: WAP to implement Constructor Overloading

#include<iostream>

using namespace std;

class Student{

public:

string name;

int sid, YOB, age;

Student(){}

Student(string name, int sid, int yob, int age){

this->name = name;

this->sid = sid;

this->YOB = yob;

this->age = age;

}

Student(string name, int sid, int Yob){

this->name = name;

this->sid = sid;

this->YOB = Yob;

this->age = 2023 - this->YOB;

}

void info(){

cout<<"student name "<<this->name;

cout<<" with student id "<<this->sid;

cout<<" and age is "<<this->age<<endl;

}

};

int main(){

Student tannu;

Student raj = Student("Raj", 1001, 2003, 19);

Student rajVerma("Raj Verma", 1002, 2006);

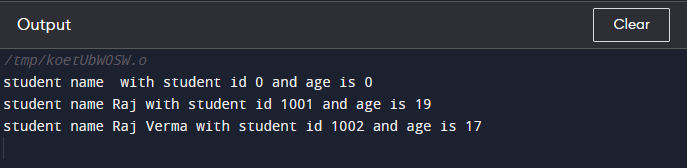
tannu.info();

raj.info();

rajVerma.info();

}

Output:



AIM: WAP to implement Copy Constructor

#include<iostream>

using namespace std;

class Student{

public:

string name;

int sid, YOB, age;

Student(string name, int sid, int Yob){

this->name = name;

this->sid = sid;

this->YOB = Yob;

this->age = 2023 - this->YOB;

}

Student(Student &obj){

this->name = obj.name;

this->sid = obj.sid;

this->age = obj.age;

this->YOB = obj.YOB;

}

void info(){

cout<<"student name "<<this->name;

cout<<" with student id "<<this->sid;

cout<<" and age is "<<this->age<<endl;

}

};

int main(){

Student rajVerma("Raj Verma", 1001, 2003);

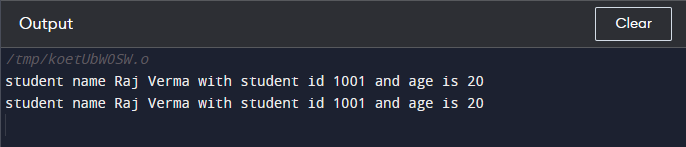
rajVerma.info();

Student rajVerma2 = Student(rajVerma);

rajVerma2.info();

}

Output:



AIM: WAP to implement private inheritance

#include <iostream>

using namespace std;

class Base{

protected:

int a = 0;

int b = -1;

int c = 1;

};

class Derived: private Base{

public:

int get\_a(){

return a;

}

int get\_b(){

return b;

}

int get\_c(){

return c;

}

};

int main() {

Derived obj;

int a = obj.get\_a();

int b = obj.get\_b();

int c = obj.get\_c();

cout<<a<<endl;

cout<<b<<endl;

cout<<c<<endl;

return 0;

}

Output:



AIM: WAP to implement Public Inheritance

#include<iostream>

using namespace std;

class Base{

public:

int pub;

private:

int pvt;

protected:

int pro;

public:

Base(){

pub=0;

pvt=-1;

pro=1;

}

};

class Derived: public Base{

public:

void info(){

cout<<pub<<endl;

//cout<<pvt<<endl;

cout<<pro<<endl;

}

};

int main(){

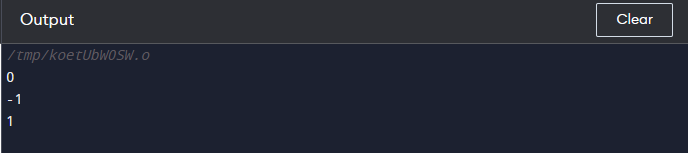
Derived obj;

obj.info();

return 0;

}

Output:



AIM: WAP to implement the use cases of scope resolution operator

#include<iostream>

using namespace std;

int num = 10;

namespace temp{

int num;

};

int main(){

int num;

num = -20;

::num = 10;

temp::num = 0;

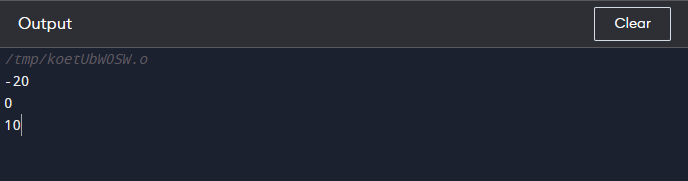
cout<<num<<endl;

cout<<temp::num<<endl;

cout<<::num;

}

Output:



AIM: WAP to take user input in constructor

#include<iostream>

using namespace std;

class Student{

public:

string name;

int sid, YOB, age;

Student(string name, int sid, int Yob){

this->name = name;

this->sid = sid;

this->YOB = Yob;

this->age = 2023 - this->YOB;

}

void info(){

cout<<"student name "<<this->name;

cout<<" with student id "<<this->sid;

cout<<" and age is "<<this->age<<endl;

}

};

int main(){

int id, yob;

string name;

cout<<"Enter name: ";

cin>>name;

cout<<"Enter student id: ";

cin>>id;

cout<<"Enter year of birth: ";

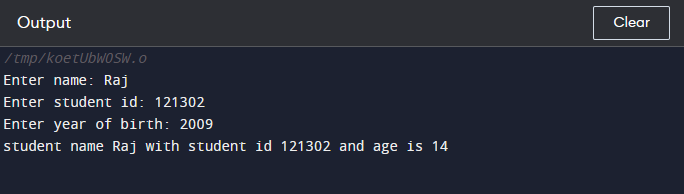
cin>>yob;

Student rajVerma(name, id, yob);

rajVerma.info();

}

Output:



AIM: WAP to implement operator overloading

#include <iostream>

using namespace std;

class Count {

private:

int value;

public:

Count() :{

value = 5;

}

void operator ++ () {

++value;

}

void display() {

cout << "Count: " << value << endl;

}

};

int main() {

Count count1;

++count1;

count1.display();

return 0;

}

Output:

