**Easy**

1. **Program**

#include <iostream>

using namespace std;

class CubeCalculator {

private:

int num;

public:

CubeCalculator(int n) : num(n) {}

void displayCubes() {

for (int i = 1; i <= num; ++i) {

std::cout << "Cube of " << i << " is: " << i\*i\*i <<endl;

}

}

~CubeCalculator() {

cout << "Destructor called. Object destroyed." <<endl;

}

};

int main() {

int limit;

cout << "Enter the limit: ";

cin >> limit;

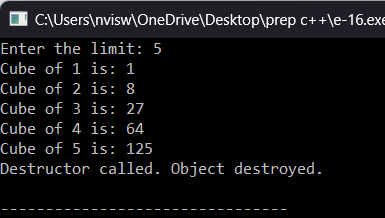
CubeCalculator calculator(limit);

calculator.displayCubes();

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

using namespace std;

class CubeCalculator {

private:

int limit;

public:

CubeCalculator() : limit(0) {}

CubeCalculator(int n) : limit(n) {}

void displayCubes() {

for (int i = 1; i <= limit; ++i) {

cout << "Cube of " << i << " is: " << i \* i \* i <<endl;

}

}

};

int main() {

int limit;

std::cout << "Enter the limit: ";

std::cin >> limit;

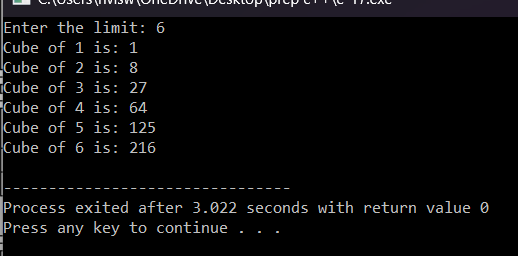
CubeCalculator calculator(limit);

calculator.displayCubes();

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

using namespace std;

class SeriesSum {

private:

int n;

public:

SeriesSum() : n(0) {}

SeriesSum(int num) : n(num) {}

int calculateSum() {

int sum = 0;

for (int i = 1; i <= n; ++i) {

sum += i;

}

return sum;

}

};

int main() {

int limit;

cout << "Enter the limit: ";

cin >> limit;

SeriesSum seriesObj(limit);

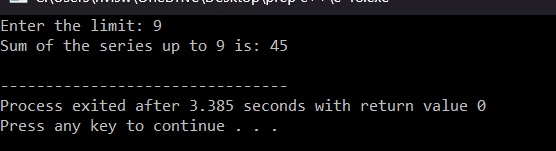
int sum = seriesObj.calculateSum();

cout << "Sum of the series up to " << limit << " is: " << sum << endl;

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

using namespace std;

class RightAngleTriangle {

private:

int rows;

public:

RightAngleTriangle() : rows(0) {}

RightAngleTriangle(int r) : rows(r) {}

void printPattern() {

for (int i = 1; i <= rows; ++i) {

for (int j = 1; j <= i; ++j) {

cout << i << " ";

}

cout << endl;

}

}

};

int main() {

int numRows;

cout << "Enter the number of rows for the right-angle triangle: ";

cin >> numRows;

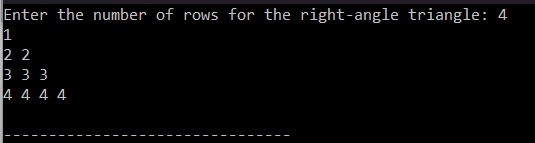
RightAngleTriangle triangle(numRows);

triangle.printPattern();

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

using namespace std;

class ReverseNumber {

private:

int originalNumber;

public:

ReverseNumber() : originalNumber(0) {}

ReverseNumber(int num) : originalNumber(num) {}

int reverse() {

int reversedNum = 0;

int temp = originalNumber;

while (temp != 0) {

int digit = temp % 10;

reversedNum = reversedNum \* 10 + digit;

temp /= 10;

}

return reversedNum;

}

};

int main() {

int num;

cout << "Enter a number to reverse: ";

cin >> num;

ReverseNumber revNum(num);

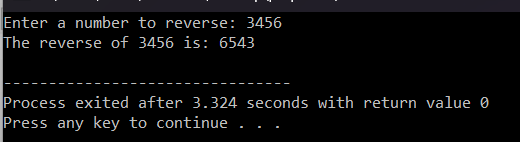
int reversed = revNum.reverse();

cout << "The reverse of " << num << " is: " << reversed << endl;

return 0;

}

**Output:**

****

**Medium**

1. **Program**

#include <iostream>

#include <cmath>

using namespace std;

class Shape {

public:

Shape() {}

virtual float area() { return 0; }

};

class Circle : public Shape {

private:

float radius;

public:

Circle(float r) : radius(r) {}

float area() override {

return M\_PI \* radius \* radius;

}

};

class Rectangle : public Shape {

private:

float length, width;

public:

Rectangle(float l, float w) : length(l), width(w) {}

float area() override {

return length \* width;

}

};

class Triangle : public Shape {

private:

float base, height;

public:

Triangle(float b, float h) : base(b), height(h) {}

float area() override {

return 0.5 \* base \* height;

}

};

int main() {

Circle circle(5);

Rectangle rectangle(4, 6);

Triangle triangle(3, 8);

cout << "Area of Circle: " << circle.area() << endl;

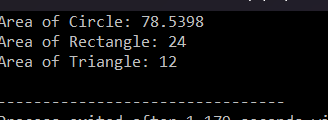
cout << "Area of Rectangle: " << rectangle.area() << endl;

cout << "Area of Triangle: " << triangle.area() << endl;

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

#include <cmath>

using namespace std;

class Shape {

public:

Shape() {}

virtual float volume() { return 0; }

};

class Cube : public Shape {

private:

float side;

public:

Cube(float s) : side(s) {}

float volume() override {

return side \* side \* side;

}

};

class Cylinder : public Shape {

private:

float radius, height;

public:

Cylinder(float r, float h) : radius(r), height(h) {}

float volume() override {

return M\_PI \* radius \* radius \* height;

}

};

int main() {

Cube cube(5);

Cylinder cylinder(3, 7);

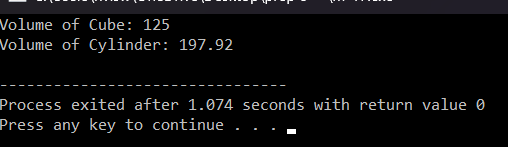
cout << "Volume of Cube: " << cube.volume() << endl;

cout << "Volume of Cylinder: " << cylinder.volume() << endl;

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

using namespace std;

class MyClass {

private:

int num;

public:

MyClass(int n);

void display();

};

MyClass::MyClass(int n) : num(n) {}

void MyClass::display() {

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

}

int main() {

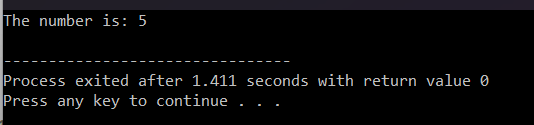
MyClass obj(5);

obj.display();

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

using namespace std;

class FloydTriangle {

private:

int rows;

public:

FloydTriangle(int r) : rows(r) {

int num = 1;

for (int i = 1; i <= rows; ++i) {

for (int j = 1; j <= i; ++j) {

cout << num << " ";

++num;

}

cout << endl;

}

}

~FloydTriangle() {

cout << "Destructor called. Object destroyed." << endl;

}

};

int main() {

int numRows;

cout << "Enter the number of rows for Floyd's Triangle: ";

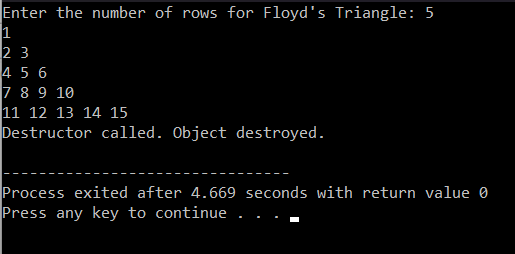
cin >> numRows;

FloydTriangle triangle(numRows);

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

using namespace std;

class DecimalToBinary {

private:

int decimal;

public:

DecimalToBinary(int dec) : decimal(dec) {}

void convertToBinary() {

int binary = 0;

int base = 1;

int temp = decimal;

while (temp > 0) {

int rem = temp % 2;

binary += rem \* base;

temp /= 2;

base \*= 10;

}

cout << "Binary equivalent of " << decimal << " is: " << binary << endl;

}

};

int main() {

int num;

cout << "Enter a decimal number: ";

cin >> num;

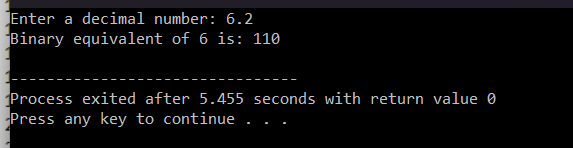
DecimalToBinary converter(num);

converter.convertToBinary();

return 0;

}

**Output:**

****

**Hard**

**1.Program**

#include <iostream>

#include <string>

using namespace std;

class Student {

private:

string name;

public:

Student() : name("Unknown") {}

Student(const string& n) : name(n) {}

void displayName() {

cout << "Student name: " << name << endl;

}

};

int main() {

Student student1;

student1.displayName();

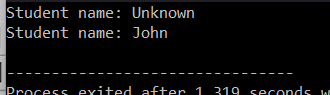
Student student2("John");

student2.displayName();

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

using namespace std;

class Rectangle {

private:

float length, breadth;

public:

Rectangle() : length(0), breadth(0) {}

Rectangle(float l, float b) : length(l), breadth(b) {}

Rectangle(float num) : length(num), breadth(num) {}

float area() {

return length \* breadth;

}

};

int main() {

Rectangle rect1;

Rectangle rect2(4, 6);

Rectangle rect3(5);

cout << "Area of rect1: " << rect1.area() << endl;

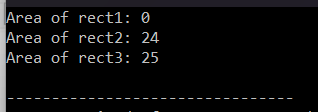
cout << "Area of rect2: " << rect2.area() << endl;

cout << "Area of rect3: " << rect3.area() << endl;

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

using namespace std;

class AddAmount {

private:

int amount;

public:

AddAmount() : amount(50) {}

AddAmount(int add) : amount(50 + add) {}

void displayAmount() {

cout << "Total amount in the piggy bank: $" << amount << endl;

}

};

int main() {

AddAmount piggyBank1;

piggyBank1.displayAmount();

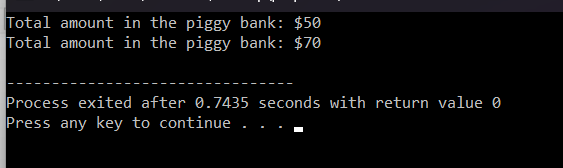
AddAmount piggyBank2(20);

piggyBank2.displayAmount();

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

using namespace std;

class Area {

public:

void printArea(float length, float breadth) {

cout << "Area of rectangle: " << length \* breadth << endl;

}

void printArea(float side) {

cout << "Area of square: " << side \* side << endl;

}

};

int main() {

Area areaObj;

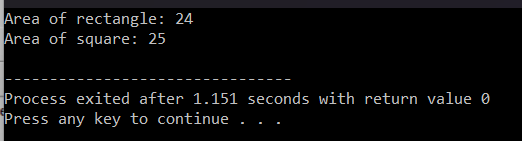
areaObj.printArea(4, 6);

areaObj.printArea(5);

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

using namespace std;

class Add {

private:

int num1, num2;

public:

Add(int n1, int n2) : num1(n1), num2(n2) {}

int operator+() {

return num1 + num2;

}

};

int main() {

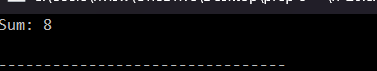
Add addObj(5, 3);

cout << "Sum: " << +addObj << endl;

return 0;

}

**Output:**

****