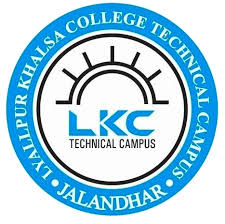
A

PRACTICAL FILE

OF

OBJECT ORIENTED PROGRAMMING SYSTEM



Lyallpur Khalsa College Technical Campus

Dept. of Computer Science& Engineering

Submitted To Submitted By

Ms. Baby Jha Name: Anshu Kumar

Assistant Professor Class: B.Tech DS

Dept. Of Comp. Science Semester-03 & Engineering University Roll No.:2443695

**Index**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.NO.** | **Tasks** | **Page** | **Date** | **Remarks** |
|  | Introduction to C++ | 1 | 08-08-25 |  |
| 1 | Write a program that uses a class where the member functions are defined inside a class. | 2-4 | 08-08-25 |  |
| 2 | Write a program that uses a class where the member functions are defined outside a class. | 5-7 | 13-08-25 |  |
| 3 | Write a program to demonstrate the use of static data members. | 8-9 | 13-08-25 |  |
| 4 | Write a program to demonstrate the use of const data members. | 10-11 | 22-08-25 |  |
| 5 | Write a program to demonstrate the use of zero argument and parameterized constructors. | 12-14 | 27-08-25 |  |
| 6 | Write a program to demonstrate the use of dynamic constructor. | 15-17 | 29-08-25 |  |
| 7 | Write a program to demonstrate the use of explicit constructor. | 18-19 | 10-09-25 |  |
| 8 | Write a program to demonstrate the use of initializer list. | 20-21 | 10-09-25 |  |
| 9 | Write a program to demonstrate the overloading of increment and decrement operators. | 22-25 | 12-09-25 |  |
| 10 | Write a program to demonstrate the overloading of memory management operators. | 26-28 | 12-09-25 |  |
| 11 | Write a program to demonstrate the typecasting of basic type to class type. | 29-30 | 17-09-25 |  |
| 12 | Write a program to demonstrate the typecasting of class type to basic type. | 31-33 | 19-09-25 |  |
| 13 | Write a program to demonstrate the typecasting of class type to class type. | 34-36 | 03-10-25 |  |
| 14 | Write a program to demonstrate the multiple inheritances. | 37-39 | 08-10-25 |  |
| 15 | Write a program to demonstrate the runtime polymorphism. | 40-42 | 10-10-25 |  |
| 16 | Write a program to demonstrate the exception handling. | 43-44 | 15-10-25 |  |
| 17 | Write a program to demonstrate the use of class template. | 45-47 | 24-10-25 |  |
| 18 | Write a program to demonstrate the reading and writing of mixed type of data. | 48-50 | 29-10-25 |  |

**Task-1**

1. **INTRODUCTION TO C++**

C++ is a powerful, general-purpose programming language developed by Bjarne Stroustrup at Bell Labs in the early 1980s. It originated as an extension of the C programming language, initially known as "C with Classes," and later renamed C++.C++ is widely used in various domains, including operating systems, desktop applications, game development, high-frequency trading systems, and scientific computing.

**BASIC STRUCTURE OF C++**

#include <iostream>

// Include the iostream header for input/output operations

int main() {

// The main function, where program execution begins

// This is a single-line comment

std::cout << "Hello, world!" << std::endl;

// Output "Hello, world!" to the console

/\*

\* This is a multi-line comment.

\* It explains the purpose of the return statement.

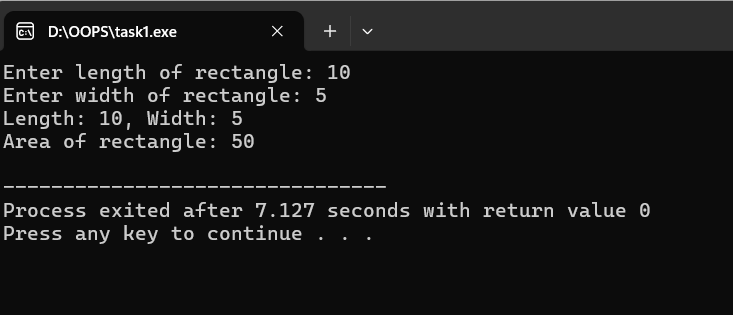
\*/

return 0;

// Indicate successful program execution

}

**Output of Task-2:**

****

**Task-2**

**2.Write a program that uses a class where member function are defined inside a class.**

#include <iostream>

using namespace std;

// Define a class with member functions inside the class

class Rectangle {

private:

double length;

double width;

public:

// Member function to set dimensions

void setDimensions(double l, double w) {

length = l;

width = w;

}

// Member function to calculate area

double area() {

return length \* width;

}

// Member function to display dimensions

void display() {

cout << "Length: " << length << ", Width: " << width << endl;

}

};

int main() {

Rectangle rect;

double l, w;

// Take input from user

cout << "Enter length of rectangle: ";

cin >> l;

cout << "Enter width of rectangle: ";

cin >> w;

// Set dimensions

rect.setDimensions(l, w);

// Display dimensions and area

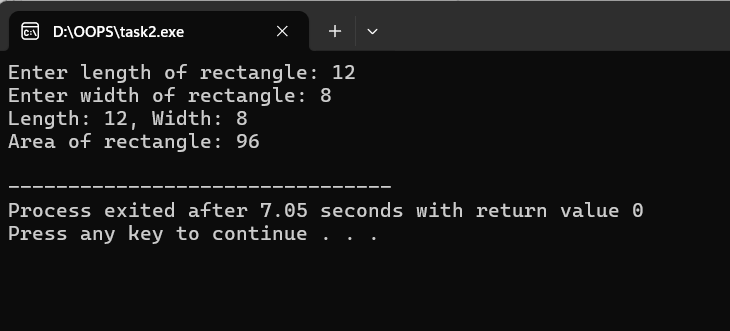
rect.display();

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

return 0;

}

**Output of Task-3:**

****

**Task-3**

**3. Write a program that uses a class where the member function are defined outside a class.**

#include <iostream>

using namespace std;

// Class declaration

class Rectangle {

private:

double length;

double width;

public:

void setDimensions(double l, double w); // Declaration only

double area(); // Declaration only

void display(); // Declaration only

};

// Member function definitions outside the class

void Rectangle::setDimensions(double l, double w) {

length = l;

width = w;

}

double Rectangle::area() {

return length \* width;

}

void Rectangle::display() {

cout << "Length: " << length << ", Width: " << width << endl;

}

int main() {

Rectangle rect;

double l, w;

// Take input from user

cout << "Enter length of rectangle: ";

cin >> l;

cout << "Enter width of rectangle: ";

cin >> w;

// Set dimensions

rect.setDimensions(l, w);

// Display dimensions and area

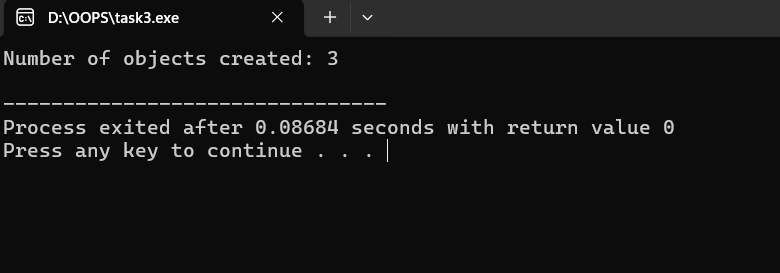
rect.display();

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

return 0;

}

**Output of Task-4:**

****

**Task-4**

**4. Write a program to demonstrate the use of static data members.**

**Input:**

#include <iostream>

using namespace std;

class Counter {

private:

static int count; // Static data member

public:

Counter() {

count++; // Increment count whenever an object is created

}

static void showCount() {

cout << "Number of objects created: " << count << endl;

}

};

// Initialize static member outside the class

int Counter::count = 0;

int main() {

Counter c1;

Counter c2;

Counter c3;

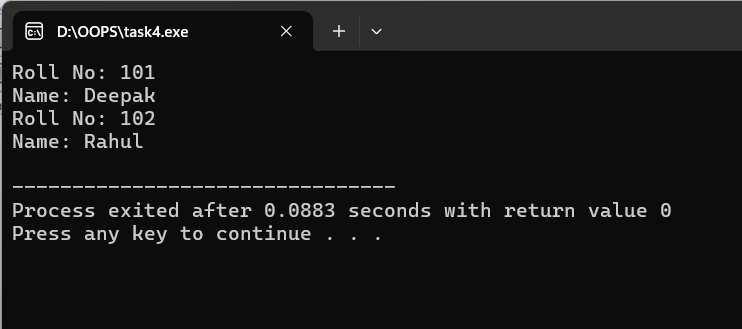
// Call static function to display count

Counter::showCount();

return 0;

}

**Output:**

****

**4. Write a program to demonstrate the use of const data members.**

**Input:**

#include <iostream>

using namespace std;

class Student {

const int rollNo; // const data member

string name;

public:

// Constructor to initialize const data member using initializer list

Student(int r, string n) : rollNo(r), name(n) {}

void display() const {

cout << "Roll No: " << rollNo << endl;

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

}

};

int main() {

Student s1(101, "Deepak");

s1.display();

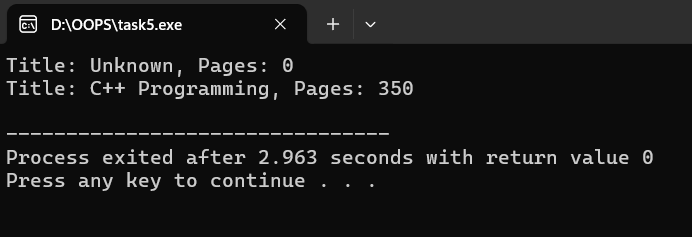
Student s2(102, "Rahul");

s2.display();

return 0;

}

**Output:**

****

**5. Write a program to demonstrate the use of zero argument and parameterized constructors.**

**Input:**

#include <iostream>

using namespace std;

class Book {

private:

string title;

int pages;

public:

// Zero-argument constructor

Book() {

title = "Unknown";

pages = 0;

}

// Parameterized constructor

Book(string t, int p) {

title = t;

pages = p;

}

void display() const {

cout << "Title: " << title << ", Pages: " << pages << endl;

}

};

int main() {

Book b1; // Calls zero-argument constructor

Book b2("C++ Programming", 350); // Calls parameterized constructor

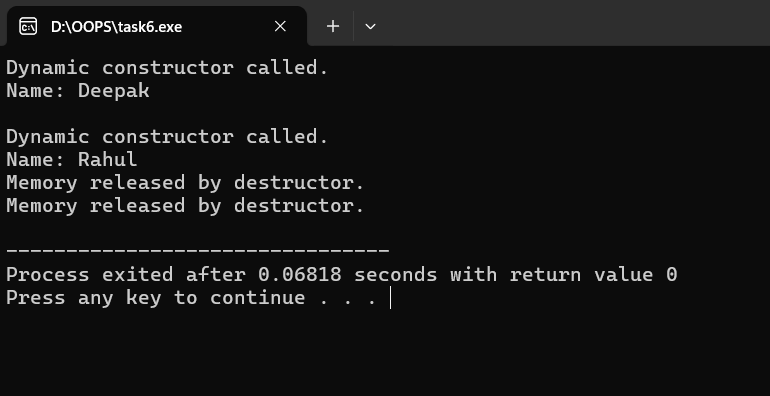
b1.display();

b2.display();

return 0;

}

**Output:**

****

**Task 6: Write a program to demonstrate the use of dynamic constructor**

**Input:**

#include <iostream>

#include <cstring>

using namespace std;

class Student {

char \*name; // pointer to hold name dynamically

int length;

public:

// Dynamic constructor

Student(const char \*n) {

length = strlen(n);

name = new char[length + 1]; // dynamic memory allocation

strcpy(name, n);

cout << "Dynamic constructor called." << endl;

}

void display() {

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

}

// Destructor to free dynamically allocated memory

~Student() {

delete[] name;

cout << "Memory released by destructor." << endl;

}

};

int main() {

Student s1("Deepak");

s1.display();

cout << endl;

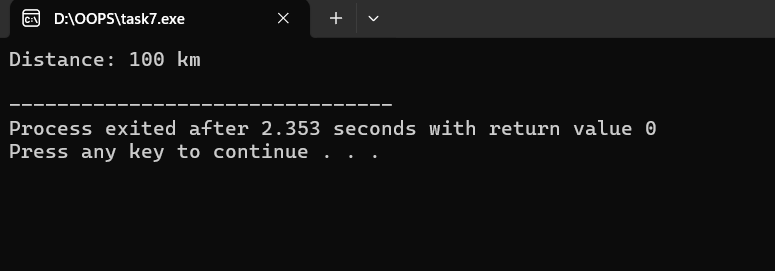
Student s2("Rahul");

s2.display();

return 0;

}

**Output:**

****

**Task 7:** **Write a program to demonstrate the use of explicit constructor.**

**Input:**

#include <iostream>

using namespace std;

class Distance {

private:

int km;

public:

// Explicit constructor

explicit Distance(int d) {

km = d;

}

void show() const {

cout << "Distance: " << km << " km" << endl;

}

};

int main() {

Distance d1(100); // ? OK: direct initialization

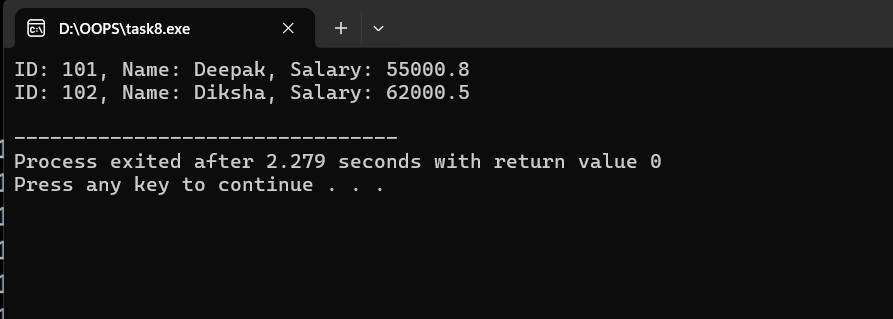
// Distance d2 = 200; // ? Error: implicit conversion not allowed due to 'explicit'

d1.show();

return 0;

}

**Output:**

****

**Task 8: Write a program to demonstrate the use of initializer list.**

**Input:**

#include <iostream>

using namespace std;

class Employee {

private:

const int id; // Must be initialized via initializer list

string name;

double salary;

public:

// Constructor using initializer list

Employee(int i, string n, double s) : id(i), name(n), salary(s) {}

void display() const {

cout << "ID: " << id << ", Name: " << name << ", Salary: " << salary << endl;

}

};

int main() {

Employee e1(101, "Deepak", 55000.75);

Employee e2(102, "Diksha", 62000.50);

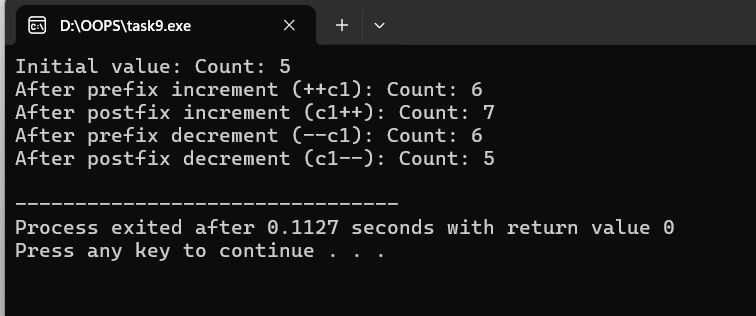
e1.display();

e2.display();

return 0;

}

**Output:**

****

**Task 9: Write a program to demonstrate the overloading of increment and decrement operators.**

**Input:**

#include <iostream>

using namespace std;

class Counter {

int count;

public:

// Constructor

Counter(int c = 0) {

count = c;

}

// Overload prefix increment operator (++obj)

Counter operator++() {

++count;

return \*this;

}

// Overload postfix increment operator (obj++)

Counter operator++(int) {

Counter temp = \*this;

count++;

return temp;

}

// Overload prefix decrement operator (--obj)

Counter operator--() {

--count;

return \*this;

}

// Overload postfix decrement operator (obj--)

Counter operator--(int) {

Counter temp = \*this;

count--;

return temp;

}

void display() {

cout << "Count: " << count << endl;

}

};

int main() {

Counter c1(5);

cout << "Initial value: ";

c1.display();

++c1; // prefix increment

cout << "After prefix increment (++c1): ";

c1.display();

c1++; // postfix increment

cout << "After postfix increment (c1++): ";

c1.display();

--c1; // prefix decrement

cout << "After prefix decrement (--c1): ";

c1.display();

c1--; // postfix decrement

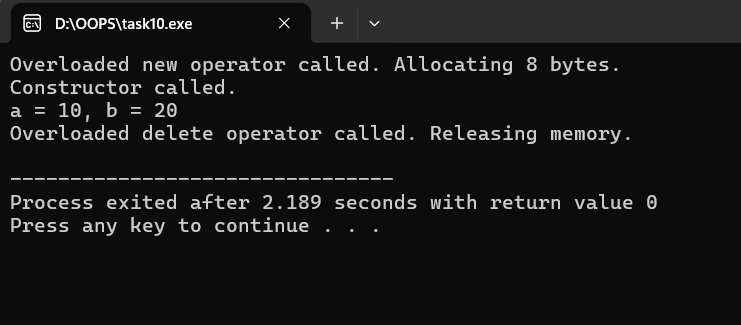
cout << "After postfix decrement (c1--): ";

c1.display();

return 0;

}

**Output:**



**Task 10: Write a program to demonstrate the overloading of memory management operators.**

**Input:**

#include <iostream>

#include <cstdlib> // for malloc() and free()

using namespace std;

class Demo {

int a, b;

public:

// Constructor

Demo(int x = 0, int y = 0) {

a = x;

b = y;

cout << "Constructor called.\n";

}

// Overloading new operator

void\* operator new(size\_t size) {

cout << "Overloaded new operator called. Allocating " << size << " bytes.\n";

void\* ptr = malloc(size); // allocate memory

if (!ptr) {

cout << "Memory allocation failed!\n";

exit(1);

}

return ptr;

}

// Overloading delete operator

void operator delete(void\* ptr) {

cout << "Overloaded delete operator called. Releasing memory.\n";

free(ptr); // deallocate memory

}

void display() {

cout << "a = " << a << ", b = " << b << endl;

}

};

int main() {

// Using overloaded new operator

Demo\* d1 = new Demo(10, 20);

d1->display();

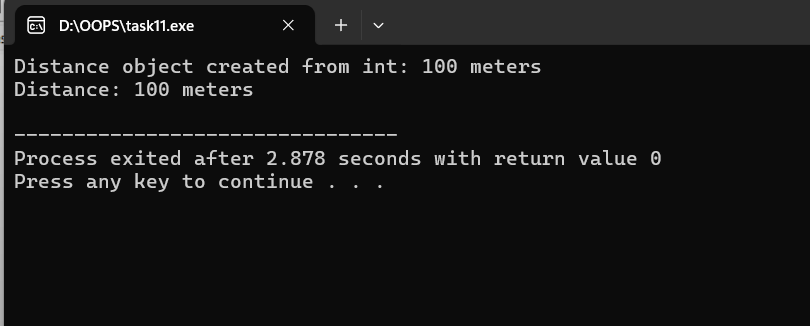
// Using overloaded delete operator

delete d1;

return 0;

}

**Output:**

****

**Task 11: Write a program to demonstrate the typecasting of basic type to class type.**

**Input:**

#include <iostream>

using namespace std;

class Distance {

private:

int meters;

public:

// Constructor to convert int to Distance

Distance(int m) {

meters = m;

cout << "Distance object created from int: " << meters << " meters\n";

}

void display() const {

cout << "Distance: " << meters << " meters" << endl;

}

};

int main() {

int length = 100;

// Typecasting int to class type using constructor

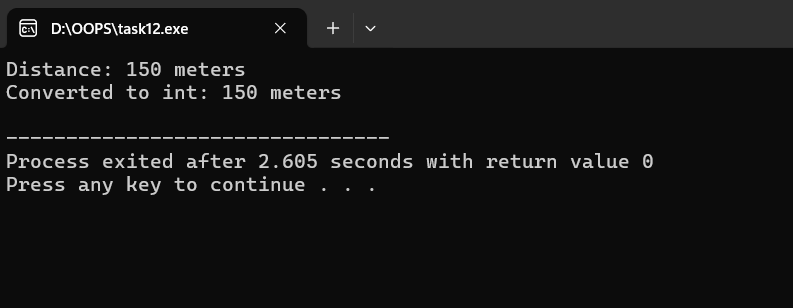
Distance d = length; // Implicit conversion

d.display();

return 0;

}

**Output:**

****

**Task 12: Write a program to demonstrate the typecasting of class type to basic type.**

**Input:**

#include <iostream>

using namespace std;

class Distance {

private:

int meters;

public:

Distance(int m) : meters(m) {}

// Conversion function: class to int

operator int() const {

return meters;

}

void display() const {

cout << "Distance: " << meters << " meters" << endl;

}

};

int main() {

Distance d(150);

d.display();

// Typecasting class object to int

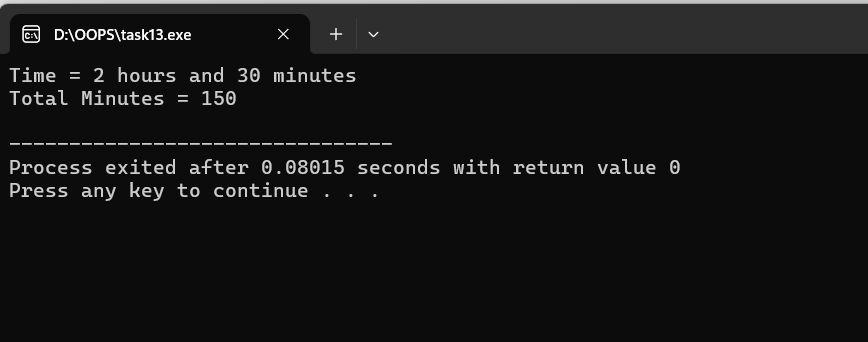
int length = d; // Implicit conversion

cout << "Converted to int: " << length << " meters" << endl;

return 0;

}

**Output:**

****

**Task 13: Write a program to demonstrate the typecasting of class type to class type**

**Input:**

#include <iostream>

using namespace std;

// Source class

class Time {

int hours;

int minutes;

public:

Time(int h = 0, int m = 0) {

hours = h;

minutes = m;

}

int getHours() const { return hours; }

int getMinutes() const { return minutes; }

void display() const {

cout << "Time = " << hours << " hours and " << minutes << " minutes" << endl;

}

};

// Destination class

class Minute {

int minutes;

public:

Minute() { minutes = 0; }

// Conversion constructor: convert Time ? Minute

Minute(Time t) {

minutes = (t.getHours() \* 60) + t.getMinutes();

}

void display() const {

cout << "Total Minutes = " << minutes << endl;

}

};

int main() {

Time t1(2, 30); // 2 hours 30 minutes

t1.display();

// Typecasting: class type (Time) ? class type (Minute)

Minute m1;

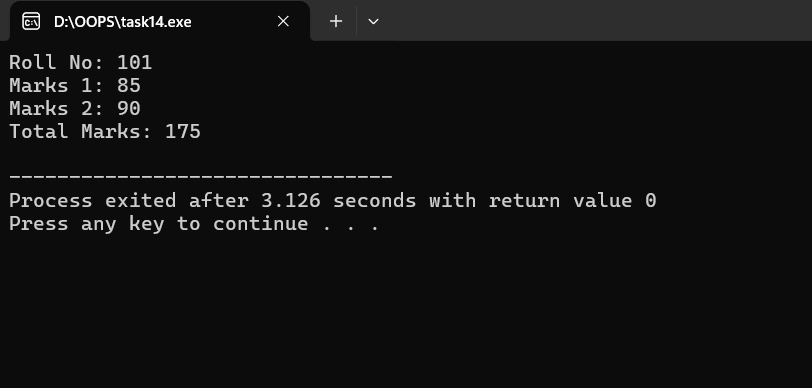
m1 = t1; // invokes Minute(Time) conversion constructor

m1.display();

return 0;

}

**Output:**

****

**Task 14: Write a program to demonstrate the multiple inheritances.**

**Input:**

#include <iostream>

using namespace std;

// Base class 1

class Student {

protected:

int rollNo;

public:

void getStudentData(int r) {

rollNo = r;

}

};

// Base class 2

class Marks {

protected:

int m1, m2;

public:

void getMarks(int a, int b) {

m1 = a;

m2 = b;

}

};

// Derived class inheriting from both Student and Marks

class Result : public Student, public Marks {

int total;

public:

void calculate() {

total = m1 + m2;

}

void display() {

cout << "Roll No: " << rollNo << endl;

cout << "Marks 1: " << m1 << endl;

cout << "Marks 2: " << m2 << endl;

cout << "Total Marks: " << total << endl;

}

};

int main() {

Result r1;

r1.getStudentData(101);

r1.getMarks(85, 90);

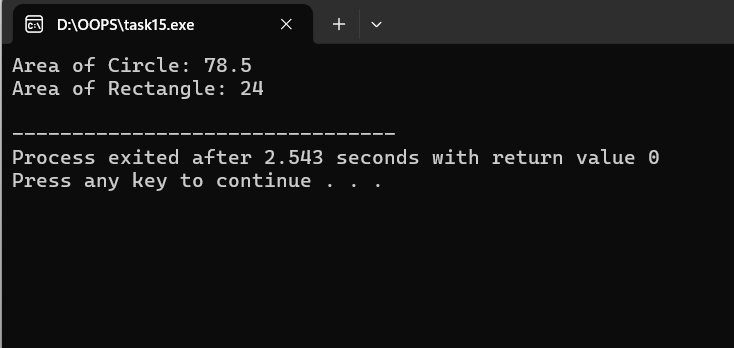
r1.calculate();

r1.display();

return 0;

}

**Output:**

****

**Task 15: Write a program to demonstrate the runtime polymorphism**

**Input:**

#include <iostream>

using namespace std;

class Shape {

public:

// Virtual function for runtime polymorphism

virtual void area() {

cout << "This is a generic shape." << endl;

}

};

class Circle : public Shape {

float radius;

public:

Circle(float r) {

radius = r;

}

void area() override { // overriding base class function

cout << "Area of Circle: " << 3.14 \* radius \* radius << endl;

}

};

class Rectangle : public Shape {

float length, breadth;

public:

Rectangle(float l, float b) {

length = l;

breadth = b;

}

void area() override {

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

}

};

int main() {

Shape \*shapePtr; // base class pointer

Circle c1(5);

Rectangle r1(4, 6);

// Pointing base class pointer to derived class objects

shapePtr = &c1;

shapePtr->area(); // Calls Circle's area() ? runtime binding

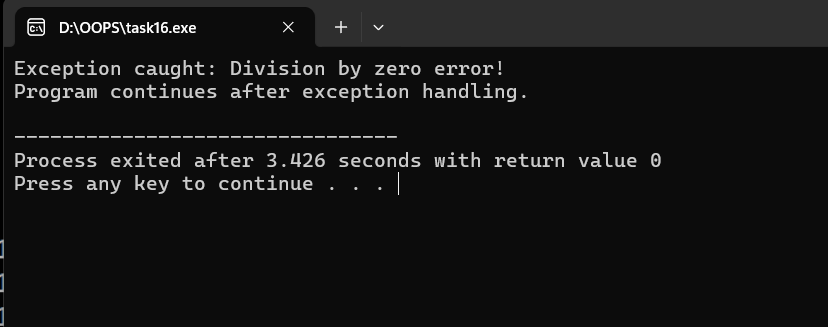
shapePtr = &r1;

shapePtr->area(); // Calls Rectangle's area() ? runtime binding

return 0;

}

**Output:**

****

**Task 16: Write a program to demonstrate the exception handling**

**Input:**

#include <iostream>

using namespace std;

int divide(int a, int b) {

if (b == 0)

throw "Division by zero error!"; // Throwing a string literal

return a / b;

}

int main() {

int x = 10, y = 0;

try {

int result = divide(x, y);

cout << "Result: " << result << endl;

} catch (const char\* msg) {

cout << "Exception caught: " << msg << endl;

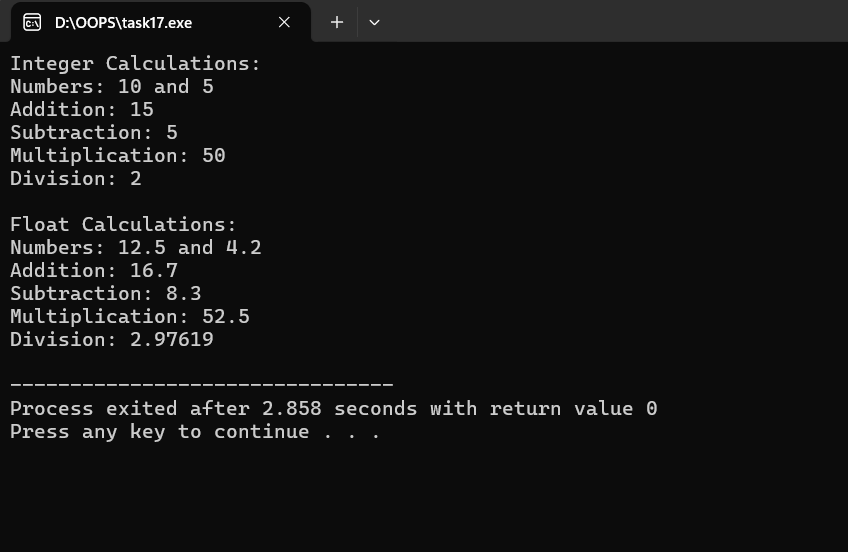
}

cout << "Program continues after exception handling." << endl;

return 0;

}

**Output:**

****

**Task 17: Write a program to demonstrate the use of class template.**

**Input:**

#include <iostream>

using namespace std;

// Class template with one type parameter

template <class T>

class Calculator {

T num1, num2;

public:

// Constructor

Calculator(T n1, T n2) {

num1 = n1;

num2 = n2;

}

void displayResults() {

cout << "Numbers: " << num1 << " and " << num2 << endl;

cout << "Addition: " << num1 + num2 << endl;

cout << "Subtraction: " << num1 - num2 << endl;

cout << "Multiplication: " << num1 \* num2 << endl;

cout << "Division: " << num1 / num2 << endl;

}

};

int main() {

// Creating object of Calculator with int type

Calculator<int> intCalc(10, 5);

cout << "Integer Calculations:\n";

intCalc.displayResults();

cout << "\n";

// Creating object of Calculator with float type

Calculator<float> floatCalc(12.5, 4.2);

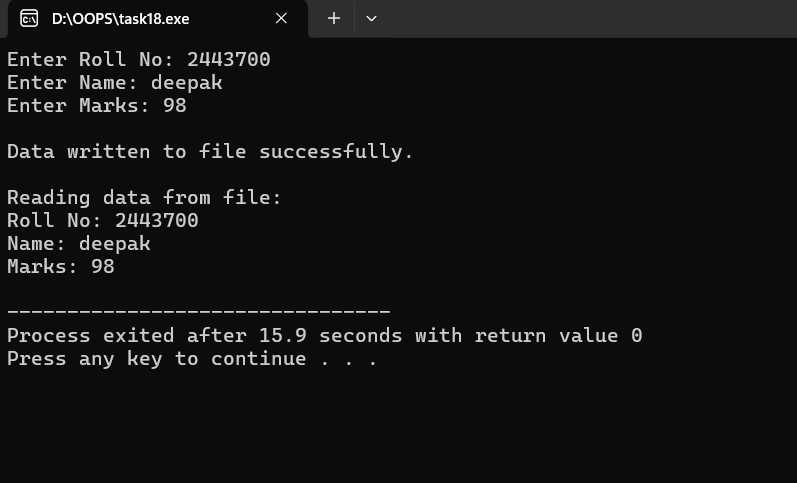
cout << "Float Calculations:\n";

floatCalc.displayResults();

return 0;

}

**Output:**

****

**Task 18: Write a program to demonstrate the reading and writing of mixed type of data.**

**Input:**

#include <iostream>

#include <fstream> // for file handling

using namespace std;

int main() {

// Variables of mixed data types

int rollNo;

string name;

float marks;

// --- Writing data to a file ---

ofstream outFile("student.txt"); // open file for writing

if (!outFile) {

cout << "Error opening file for writing!" << endl;

return 1;

}

cout << "Enter Roll No: ";

cin >> rollNo;

cout << "Enter Name: ";

cin >> name;

cout << "Enter Marks: ";

cin >> marks;

// Write mixed type data to file

outFile << rollNo << " " << name << " " << marks << endl;

outFile.close(); // close file after writing

cout << "\nData written to file successfully.\n" << endl;

// --- Reading data from file ---

ifstream inFile("student.txt"); // open file for reading

if (!inFile) {

cout << "Error opening file for reading!" << endl;

return 1;

}

cout << "Reading data from file:\n";

// Read mixed type data from file

inFile >> rollNo >> name >> marks;

// Display read data

cout << "Roll No: " << rollNo << endl;

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

cout << "Marks: " << marks << endl;

inFile.close(); // close file after reading

return 0;

}