**Easy**

1. **Program**

#include <iostream>

using namespace std;

class Person {

protected:

string name;

int age;

public:

void getData() {

cout << "Enter name: ";

cin >> name;

cout << "Enter age: ";

cin >> age;

}

void displayData() {

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

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

}

};

class Admin : public Person {

protected:

float salary;

public:

void getData() {

Person::getData();

cout << "Enter salary: ";

cin >> salary;

}

void displayData() {

Person::displayData();

cout << "Salary: $" << salary << endl;

}

virtual float bonus() {

return salary \* 0.05; // 5% of salary

}

};

class Account : public Person {

protected:

float balance;

public:

void getData() {

Person::getData();

cout << "Enter balance: ";

cin >> balance;

}

void displayData() {

Person::displayData();

cout << "Balance: $" << balance << endl;

}

virtual float bonus() {

return balance \* 0.1;

}

};

class Master : public Admin, public Account {

public:

void getData() {

Admin::getData();

Account::getData();

}

void displayData() {

Admin::displayData();

Account::displayData();

}

float bonus() {

return Admin::bonus() + Account::bonus();

}

};

int main() {

Master emp;

emp.getData();

cout << "Employee Details:\n";

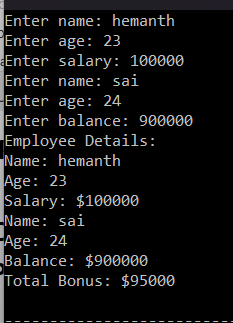
emp.displayData();

cout << "Total Bonus: $" << emp.bonus() << endl;

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

using namespace std;

class Shape {

protected:

float length;

float breadth;

public:

void getData() {

cout << "Enter length: ";

cin >> length;

cout << "Enter breadth: ";

cin >> breadth;

}

};

class Rectangle : public Shape {

public:

float area() {

return length \* breadth;

}

};

class Cuboid : public Rectangle {

float height;

public:

void getData() {

Rectangle::getData();

cout << "Enter height: ";

cin >> height;

}

float volume() {

return length \* breadth \* height;

}

};

int main() {

Cuboid c;

c.getData();

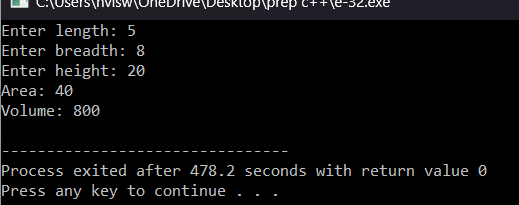
cout << "Area: " << c.area() << endl;

cout << "Volume: " << c.volume() << endl;

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

using namespace std;

class Number {

protected:

int M, N, K;

public:

void getData() {

cout << "Enter M, N, and K: ";

cin >> M >> N >> K;

}

// Getter functions to access the protected member variables

int getM() const { return M; }

int getN() const { return N; }

int getK() const { return K; }

};

class Skipper : public Number {

public:

void printNumbers() {

for (int i = getM(); i <= getN(); i += getK()) {

cout << i << " ";

}

cout << endl;

}

};

int main() {

Skipper s;

s.getData();

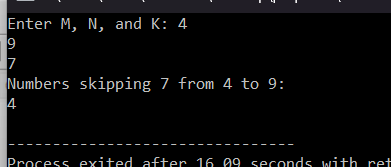
cout << "Numbers skipping " << s.getK() << " from " << s.getM() << " to " << s.getN() << ":\n";

s.printNumbers();

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

#include <string>

using namespace std;

class Grandfather {

private:

double propertyValue;

public:

Grandfather() : propertyValue(500.0) {}

double getProperty() const {

return propertyValue;

}

void setProperty(double newValue) {

propertyValue = newValue;

}

};

int main() {

Grandfather grandpa;

cout << "Grandfather's property value: " << grandpa.getProperty() << " Cr" << endl;

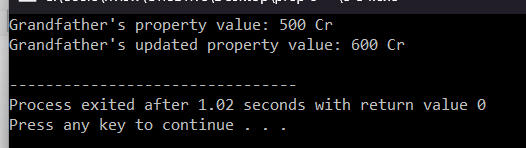
grandpa.setProperty(600.0);

cout << "Grandfather's updated property value: " << grandpa.getProperty() << " Cr" << endl;

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

#include <string>

#include <vector>

using namespace std;

class Author {

private:

string name;

bool contributed;

public:

Author(const string& n, bool contr) : name(n), contributed(contr) {}

string getName() const {

return name;

}

bool hasContributed() const {

return contributed;

}

};

class Paper {

private:

vector<Author> authors;

public:

void addAuthor(const Author& author) {

authors.push\_back(author);

}

void identifyNonContributor() const {

cout << "Authors who didn't contribute to the paper:" << endl;

for (size\_t i = 0; i < authors.size(); ++i) {

const Author& author = authors[i];

if (!author.hasContributed()) {

cout << "- " << author.getName() << endl;

}

}

}

};

int main() {

Paper paper;

paper.addAuthor(Author("Author 1", true));

paper.addAuthor(Author("Author 2", true));

paper.addAuthor(Author("Author 3", false));

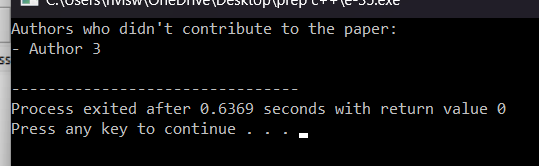
paper.addAuthor(Author("Author 4", true));

paper.identifyNonContributor();

return 0;

}

**Output:**

****

**Medium**

1. **Program**

#include <iostream>

using namespace std;

class AddAmount {

protected:

int amount;

public:

AddAmount() : amount(50) {}

void displayAmount() {

cout << "Total amount in the Piggie Bank: $" << amount << endl;

}

};

class NoAddAmount : public AddAmount {

public:

NoAddAmount() : AddAmount() {}

};

class AddMoreAmount : public AddAmount {

public:

AddMoreAmount(int additional\_amount) {

amount += additional\_amount;

}

};

int main() {

NoAddAmount piggy\_bank\_no\_add;

AddMoreAmount piggy\_bank\_add\_more(30);

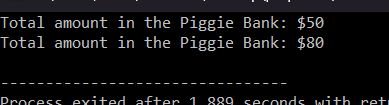
piggy\_bank\_no\_add.displayAmount();

piggy\_bank\_add\_more.displayAmount();

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

#include <string>

using namespace std;

class Person {

protected:

string name;

int age;

public:

virtual void getData() {

cout << "Enter name: ";

cin >> name;

cout << "Enter age: ";

cin >> age;

}

virtual void displayData() {

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

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

}

virtual double calculateBonus() = 0;

};

class Employee : public Person {

protected:

double salary;

double balance;

public:

void getData() override {

Person::getData();

cout << "Enter salary: $";

cin >> salary;

cout << "Enter balance: $";

cin >> balance;

}

void displayData() override {

Person::displayData();

cout << "Salary: $" << salary << endl;

cout << "Balance: $" << balance << endl;

}

double calculateBonus() override {

return salary \* 0.1 + balance \* 0.05;

}

};

int main() {

Employee employee;

employee.getData();

cout << "\nEmployee Details:\n";

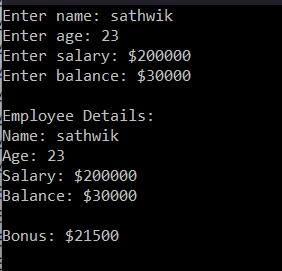
employee.displayData();

cout << "\nBonus: $" << employee.calculateBonus() << endl;

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

#include <string>

using namespace std;

class Employee {

protected:

string name;

int emp\_id;

double basic\_salary;

public:

Employee(const string& emp\_name, int emp\_id, double salary)

: name(emp\_name), emp\_id(emp\_id), basic\_salary(salary) {}

virtual void calculatePay() = 0;

};

class PayCalculator : public Employee {

protected:

double gross\_pay;

double net\_pay;

public:

PayCalculator(const string& emp\_name, int emp\_id, double salary)

: Employee(emp\_name, emp\_id, salary), gross\_pay(0), net\_pay(0) {}

void calculatePay() override {

// Calculate gross pay (assume 10% bonus)

gross\_pay = basic\_salary + basic\_salary \* 0.1;

// Calculate net pay (assume 20% deduction)

net\_pay = gross\_pay - gross\_pay \* 0.2;

}

void displayPay() {

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

cout << "Employee ID: " << emp\_id << endl;

cout << "Basic Salary: $" << basic\_salary << endl;

cout << "Gross Pay: $" << gross\_pay << endl;

cout << "Net Pay: $" << net\_pay << endl;

}

};

int main() {

string emp\_name;

int emp\_id;

double basic\_salary;

cout << "Enter employee name: ";

getline(cin, emp\_name);

cout << "Enter employee ID: ";

cin >> emp\_id;

cout << "Enter basic salary: $";

cin >> basic\_salary;

PayCalculator employee(emp\_name, emp\_id, basic\_salary);

employee.calculatePay();

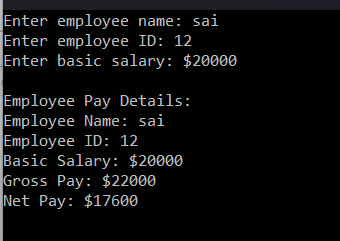
cout << "\nEmployee Pay Details:\n";

employee.displayPay();

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

using namespace std;

class Shape {

protected:

double height;

public:

Shape(double h) : height(h) {}

virtual double calculateVolume() = 0;

};

class Rectangle {

protected:

double length;

double width;

public:

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

double calculateArea() {

return length \* width;

}

};

class Cuboid : public Rectangle, public Shape {

public:

Cuboid(double l, double w, double h)

: Rectangle(l, w), Shape(h) {}

double calculateVolume() override {

return length \* width \* height;

}

};

int main() {

double length, width, height;

cout << "Enter length of cuboid: ";

cin >> length;

cout << "Enter width of cuboid: ";

cin >> width;

cout << "Enter height of cuboid: ";

cin >> height;

Cuboid cuboid(length, width, height);

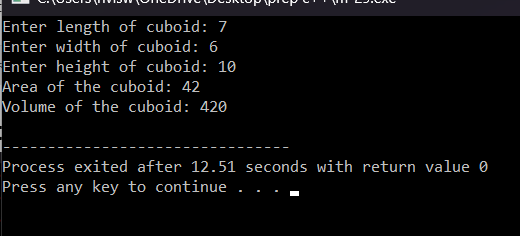
cout << "Area of the cuboid: " << cuboid.calculateArea() << endl;

cout << "Volume of the cuboid: " << cuboid.calculateVolume() << endl;

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

using namespace std;

class Bank {

public:

virtual float getRateOfInterest() { return 0; }

};

class SBI : public Bank {

public:

float getRateOfInterest() override { return 8.4f; }

};

class ICICI : public Bank {

public:

float getRateOfInterest() override { return 7.3f; }

};

class AXIS : public Bank {

public:

float getRateOfInterest() override { return 9.7f; }

};

int main() {

Bank \*b;

SBI sbi;

ICICI icici;

AXIS axis;

b = &sbi;

cout << "SBI Rate of Interest: " << b->getRateOfInterest() << endl;

b = &icici;

cout << "ICICI Rate of Interest: " << b->getRateOfInterest() << endl;

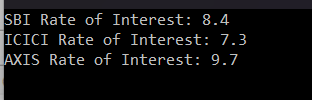
b = &axis;

cout << "AXIS Rate of Interest: " << b->getRateOfInterest() << endl;

return 0;

}

**Output:**

****

**Hard**

1. **Program**

#include <iostream>

using namespace std;

class IncomeTax {

protected:

double income;

public:

IncomeTax(double inc) : income(inc) {}

virtual double calculateTax() = 0;

};

class Slab1 : public IncomeTax {

public:

Slab1(double inc) : IncomeTax(inc) {}

double calculateTax() override {

if (income <= 150000)

return 0;

else if (income <= 300000)

return (income - 150000) \* 0.1;

else

return 15000;

}

};

class Slab2 : public IncomeTax {

public:

Slab2(double inc) : IncomeTax(inc) {}

double calculateTax() override {

if (income <= 300000)

return 0;

else if (income <= 500000)

return (income - 300000) \* 0.2;

else

return 40000;

}

};

class Slab3 : public IncomeTax {

public:

Slab3(double inc) : IncomeTax(inc) {}

double calculateTax() override {

if (income <= 500000)

return 0;

else

return (income - 500000) \* 0.3;

}

};

int main() {

double income;

cout << "Enter your income: ";

cin >> income;

IncomeTax \*tax;

if (income <= 150000)

tax = new Slab1(income);

else if (income <= 300000)

tax = new Slab1(income);

else if (income <= 500000)

tax = new Slab2(income);

else

tax = new Slab3(income);

double tax\_amount = tax->calculateTax();

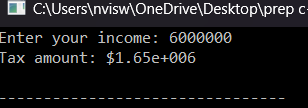
cout << "Tax amount: $" << tax\_amount << endl;

delete tax;

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

using namespace std;

class GradeException {};

class Student {

private:

int marks[4];

int total;

double aggregate;

public:

void enterMarks() {

cout << "Enter marks for four subjects:" << endl;

for (int i = 0; i < 4; ++i) {

cout << "Subject " << i+1 << ": ";

cin >> marks[i];

}

}

void calculate() {

total = 0;

for (int i = 0; i < 4; ++i) {

total += marks[i];

}

aggregate = total / 4.0;

}

void displayGrade() {

try {

if (aggregate < 50)

throw GradeException();

cout << "Grade: Pass" << endl;

} catch (GradeException& e) {

cout << "Grade: Fail" << endl;

}

}

};

int main() {

Student student;

student.enterMarks();

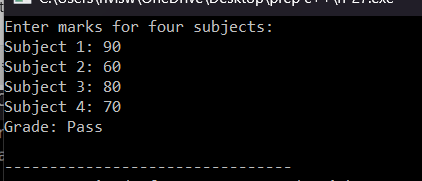
student.calculate();

student.displayGrade();

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

using namespace std;

class DivisionByZeroException {};

int divide(int dividend, int divisor) {

if (divisor == 0)

throw DivisionByZeroException();

return dividend / divisor;

}

int main() {

int num1, num2;

cout << "Enter two numbers: ";

cin >> num1 >> num2;

try {

int result = divide(num1, num2);

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

} catch (DivisionByZeroException& e) {

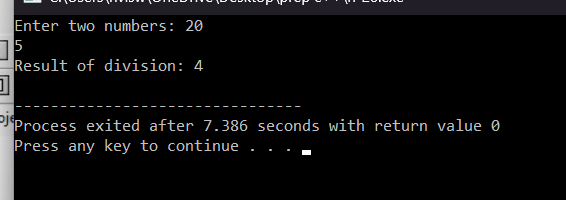
cout << "Error: Division by zero is not allowed." << endl;

}

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

#include <stdexcept>

using namespace std;

template <typename T, size\_t N>

class SafeArray {

private:

T arr[N];

public:

T& operator[](size\_t index) {

if (index >= N)

throw out\_of\_range("Index out of bounds");

return arr[index];

}

};

int main() {

SafeArray<int, 5> arr;

try {

for (int i = 0; i <= 5; ++i) {

arr[i] = i;

cout << "arr[" << i << "] = " << arr[i] << endl;

}

} catch (const out\_of\_range& e) {

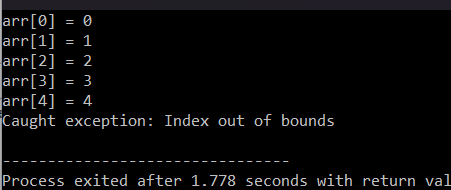
cerr << "Caught exception: " << e.what() << endl;

}

return 0;

}

**Output:**

****

1. **Program**

#include <iostream>

#include <stdexcept>

using namespace std;

void handleException(int choice) {

switch (choice) {

case 1:

throw runtime\_error("Runtime error occurred.");

case 2:

throw logic\_error("Logic error occurred.");

case 3:

throw invalid\_argument("Invalid argument exception occurred.");

default:

throw "Unknown exception occurred.";

}

}

int main() {

int choice;

cout << "Enter a number between 1 and 3: ";

cin >> choice;

try {

handleException(choice);

} catch (const runtime\_error& e) {

cerr << "Caught runtime\_error: " << e.what() << endl;

} catch (const logic\_error& e) {

cerr << "Caught logic\_error: " << e.what() << endl;

} catch (const invalid\_argument& e) {

cerr << "Caught invalid\_argument: " << e.what() << endl;

} catch (const char\* e) {

cerr << "Caught unknown exception: " << e << endl;

} catch (...) {

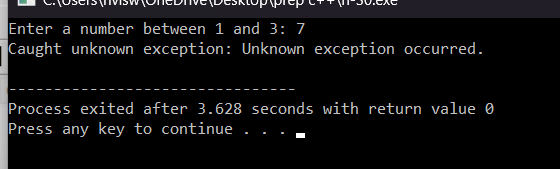
cerr << "Caught unknown exception." << endl;

}

return 0;

}

**Output:**

****