**DAY 5 LAB**

**MEDIUM**

**1.PROGRAM**

#include <iostream>

using namespace std;

class AddAmount

{

protected:

float amount;

public:

AddAmount() : amount(50) {}

void displayAmount() const

{

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

}

};

class NoAmountAdded : public AddAmount

{

public:

NoAmountAdded() {}

};

class AmountAdded : public AddAmount

{

public:

AmountAdded(float addedAmount)

{

amount += addedAmount;

}

};

int main()

{

NoAmountAdded noAddition;

AmountAdded addition(25);

cout << "For Base class (no amount added):" << endl;

noAddition.displayAmount();

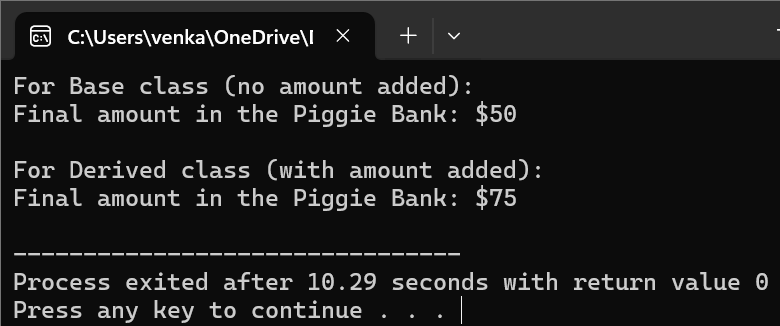
cout << "\nFor Derived class (with amount added):" << endl;

addition.displayAmount();

return 0;

}

**OUTPUT**



**2.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 float calculateBonus()

{

return 0.0;

}

};

class Admin : public Person

{

protected:

float adminBonus;

public:

void getData() override

{

Person::getData();

cout << "Enter admin bonus: ";

cin >> adminBonus;

}

void displayData() override

{

Person::displayData();

cout << "Admin bonus: $" << adminBonus << endl;

}

float calculateBonus() override

{

return adminBonus;

}

};

class Account : public Person

{

protected:

float accountBonus;

public:

void getData() override

{

Person::getData();

cout << "Enter account bonus: ";

cin >> accountBonus;

}

void displayData() override

{

Person::displayData();

cout << "Account bonus: $" << accountBonus << endl;

}

float calculateBonus() override

{

return accountBonus;

}

};

class Master : public Admin, public Account

{

public:

void getData() override

{

Admin::getData();

Account::getData();

}

void displayData() override

{

Admin::displayData();

Account::displayData();

}

float calculateBonus() override

{

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

}

};

int main()

{

Master employee;

cout << "Enter details for employee: " << endl;

employee.getData();

cout << "\nEmployee details: " << endl;

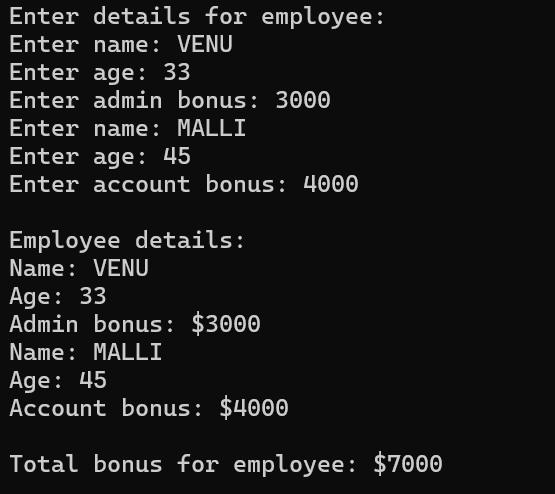
employee.displayData();

cout << "\nTotal bonus for employee: $" << employee.calculateBonus() << endl;

return 0;

}

**OUTPUT**



**3.PROGRAM**

#include <iostream>

using namespace std;

class Shape

{

protected:

float length;

float width;

public:

Shape(float l = 0, float w = 0) : length(l), width(w) {}

virtual float area() const

{

return length \* width;

}

};

class Rectangle : public Shape

{

public:

Rectangle(float l = 0, float w = 0) : Shape(l, w) {}

};

class Cuboid : public Rectangle

{

private:

float height;

public:

Cuboid(float l = 0, float w = 0, float h = 0) : Rectangle(l, w), height(h) {}

float volume() const

{

return length \* width \* height;

}

float area() const override

{

return 2 \* (length \* width + length \* height + width \* height);

}

};

int main()

{

Cuboid cuboid(3, 4, 5);

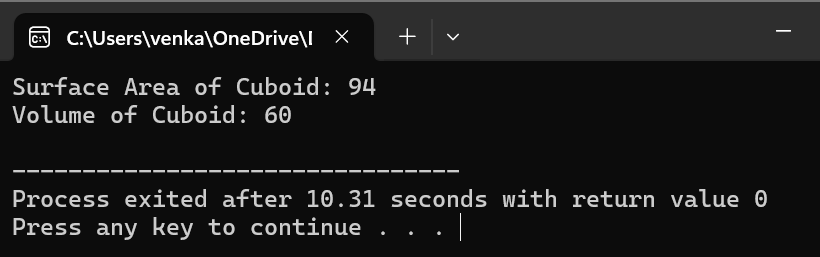
cout << "Surface Area of Cuboid: " << cuboid.area() << endl;

cout << "Volume of Cuboid: " << cuboid.volume() << endl;

return 0;

}

**OUTPUT**



**HARD**

**1.PROGRAM**

#include <iostream>

using namespace std;

class IncomeTax

{

public:

virtual float TDS(float income) = 0;

};

class Slab1 : public IncomeTax

{

public:

float TDS(float income) override

{

if (income <= 150000)

return 0;

else if (income <= 300000)

return (income - 150000) \* 0.1;

else

return 15000 + (income - 300000) \* 0.2;

}

};

class Slab2 : public IncomeTax

{

public:

float TDS(float income) override

{

if (income <= 300000)

return 0;

else if (income <= 500000)

return (income - 300000) \* 0.2;

else

return 40000 + (income - 500000) \* 0.3;

}

};

class Slab3 : public IncomeTax

{

public:

float TDS(float income) override

{

if (income <= 500000)

return 0;

else

return 50000 + (income - 500000) \* 0.3;

}

};

int main()

{

float income;

cout << "Enter income: ";

cin >> income;

IncomeTax \*tax;

if (income <= 300000)

tax = new Slab1();

else if (income <= 500000)

tax = new Slab2();

else

tax = new Slab3();

float tds = tax->TDS(income);

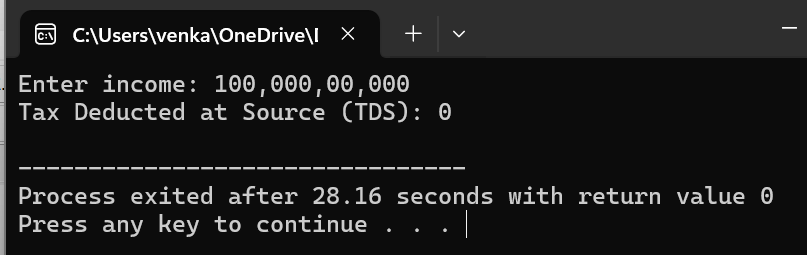
cout << "Tax Deducted at Source (TDS): " << tds << endl;

delete tax;

return 0;

}

**OUTPUT**



**2.PROGRAM**

#include <iostream>

using namespace std;

class GradeCalculator

{

private:

float marks[6];

public:

void inputMarks()

{

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

for (int i = 0; i < 3; ++i)

{

cout << "Subject " << i + 3<< ": ";

cin >> marks[i];

}

}

float calculateTotal()

{

float total = 0;

for (int i = 0; i < 3; ++i)

{

total += marks[i];

}

return total;

}

float calculateAggregate()

{

return calculateTotal() / 40;

}

void calculateGrade()

{

float aggregate = calculateAggregate();

if (aggregate < 50)

throw "fail";

else

cout << "Grade: pass" << endl;

}

};

int main()

{

GradeCalculator student;

student.inputMarks();

try {

float total = student.calculateTotal();

float aggregate = student.calculateAggregate();

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

cout << "Aggregate: " << aggregate << endl;

student.calculateGrade();

}

catch (const char\* msg)

{

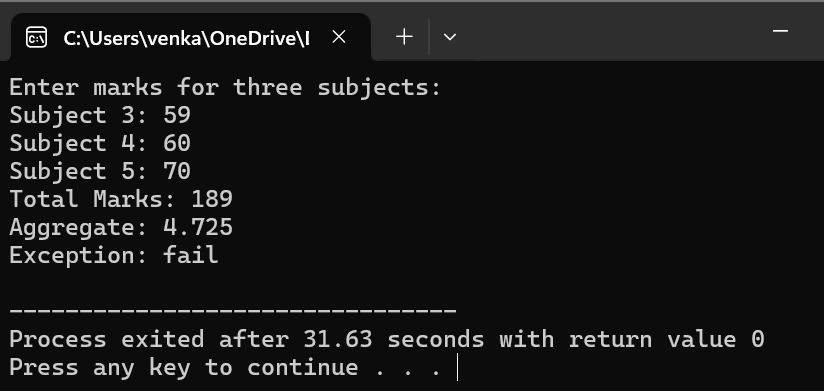
cerr << "Exception: " << msg << endl;

}

return 0;

}

**OUTPUT**



**3.PROGRAM**

#include <iostream>

using namespace std;

int divide(int dividend, int divisor)

{

if (divisor == 0)

throw "Division by zero exception";

return dividend / divisor;

}

int main()

{

int dividend, divisor;

cout << "Enter the dividend: ";

cin >> dividend;

cout << "Enter the divisor: ";

cin >> divisor;

try

{

int result = divide(dividend, divisor);

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

} catch (const char\* msg) {

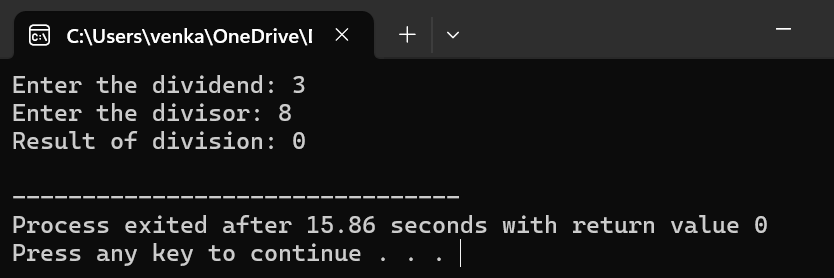
cerr << "Exception: " << msg << endl;

}

return 0;

}

**OUTPUT**



**4.PROGRAM**

#include <iostream>

using namespace std;

int main()

{

int arr[5] = {1, 2, 3, 4, 5};

int index;

cout << "Enter the index: ";

cin >> index;

try

{

if (index < 0 || index >= 5)

throw "Array index out of bounds exception";

else

cout << "Value at index " << index << ": " << arr[index] << endl;

} catch (const char\* msg) {

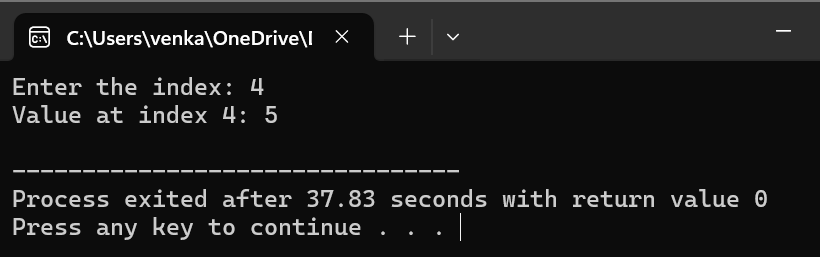
cerr << "Exception: " << msg << endl;

}

return 0;

}

OUTPUT



**5.PROGRAM**

#include <iostream>

using namespace std;

void throwExceptions(int choice)

{

switch(choice)

{

case 1:

throw "Exception 1: Division by zero";

case 2:

throw "Exception 2: Out of memory";

case 3:

throw "Exception 3: Invalid input";

default:

throw "Unknown exception";

}

}

int main()

{

int choice;

cout << "Enter a number (1, 2, or 3): ";

cin >> choice;

try {

throwExceptions(choice);

}

catch (const char\* msg)

{

cerr << "Caught exception: " << msg << endl;

}

catch (...)

{

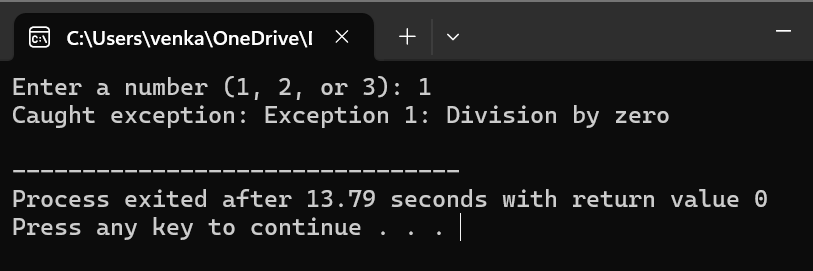
cerr << "Caught unknown exception" << endl;

}

return 0;

}

**OUTPUT**



**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 basicSalary;

public:

void getData()

{

Person::getData();

cout << "Enter basic salary: ";

cin >> basicSalary;

}

void displayData()

{

Person::displayData();

cout << "Basic Salary: " << basicSalary << endl;

}

float calculateBonus()

{

return basicSalary \* 0.1;

}

};

class Account : public Person

{

protected:

float salary;

public:

void getData()

{

Person::getData();

cout << "Enter salary: ";

cin >> salary;

}

void displayData()

{

Person::displayData();

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

}

float calculateBonus()

{

return salary \* 0.08;

}

};

class Master : public Admin, public Account

{

public:

void getData()

{

Admin::getData();

Account::getData();

}

void displayData()

{

Admin::displayData();

Account::displayData();

}

float calculateBonus()

{

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

}

};

int main()

{

Master employee;

cout << "Enter employee details:" << endl;

employee.getData();

cout << "\nEmployee details:" << endl;

employee.displayData();

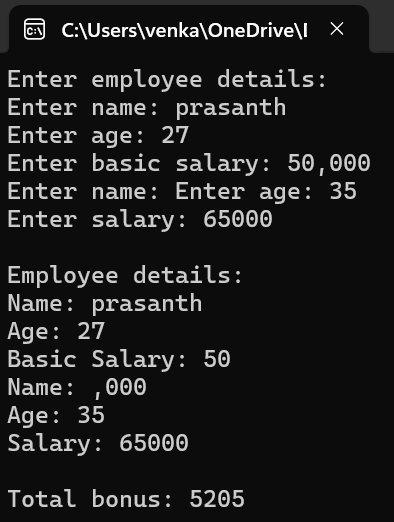
float bonus = employee.calculateBonus();

cout << "\nTotal bonus: " << bonus << endl;

return 0;

}

**OUTPUT**



**2.PROGRAM**

#include <iostream>

using namespace std;

class Shape

{

protected:

string shapeType;

public:

Shape(string type) : shapeType(type) {}

virtual void display()

{

cout << "Shape Type: " << shapeType << endl;

}

};

class Rectangle

{

protected:

float length;

float width;

public:

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

virtual float calculateArea()

{

return length \* width;

}

void display()

{

cout << "Length: " << length << endl;

cout << "Width: " << width << endl;

}

};

class Cuboid : public Rectangle, public Shape

{

private:

float height;

public:

Cuboid(float l, float w, float h) : Rectangle(l, w), Shape("Cuboid"), height(h) {}

float calculateVolume()

{

return length \* width \* height;

}

void display()

{

Rectangle::display();

Shape::display();

cout << "Height: " << height << endl;

}

};

int main()

{

Cuboid cuboid(5, 3, 2);

cout << "Cuboid Details:" << endl;

cuboid.display();

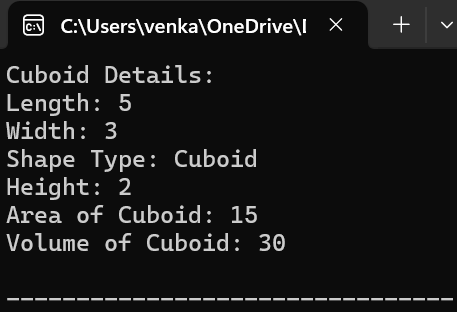
cout << "Area of Cuboid: " << cuboid.calculateArea() << endl;

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

return 0;

}

**OUTPUT**



**3.PROGRAM**

#include <iostream>

using namespace std;

class Number

{

protected:

int m, n, k;

public:

Number(int start, int end, int skip) : m(start), n(end), k(skip) {}

virtual void printNumbers() = 0;

};

class Skipper : public Number

{

public:

Skipper(int start, int end, int skip) : Number(start, end, skip) {}

void printNumbers() override {

cout << "Numbers from " << m << " to " << n << " skipping " << k << " numbers:" << endl;

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

cout << i << " ";

}

cout << endl;

}

};

int main()

{

int start, end, skip;

cout << "Enter starting number (M): ";

cin >> start;

cout << "Enter ending number (N): ";

cin >> end;

cout << "Enter number of numbers to skip (K): ";

cin >> skip;

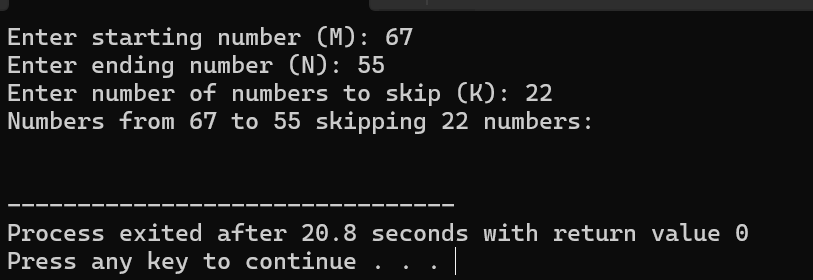
Skipper skipper(start, end, skip);

skipper.printNumbers();

return 0;

}

**OUTPUT**



**4.PROGRAM**

#include <iostream>

using namespace std;

class Grandfather

{

private:

float propertyValue;

protected:

float getProperty()

{

return propertyValue;

}

public:

Grandfather() : propertyValue(500.0) {}

void displayProperty()

{

cout << "Grandfather's property value: " << propertyValue << "Cr" << endl;

}

};

class Grandson : public Grandfather

{

public:

void accessProperty()

{

float property = getProperty();

cout << "Grandson accessed Grandfather's property value: " << property << "Cr" << endl;

}

};

int main()

{

Grandson grandson;

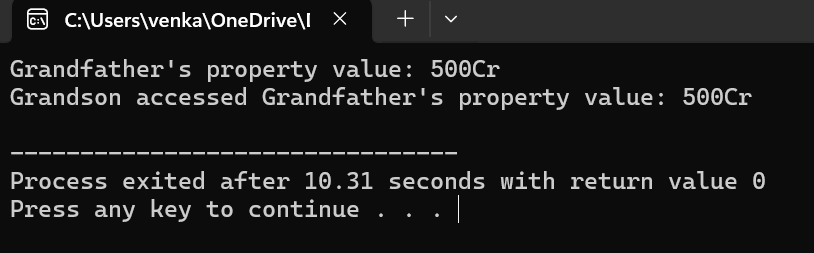
grandson.displayProperty();

grandson.accessProperty();

return 0;

}

**OUTPUT**



**5.PROGRAM**

#include <iostream>

#include <string>

#include <vector>

using namespace std;

class Author

{

private:

string name;

bool worked;

public:

Author(string n) : name(n), worked(true) {}

string getName() const

{

return name;

}

bool hasWorked() const

{

return worked;

}

void markAsNotWorked()

{

worked = false;

}

};

int main()

{

vector<Author> authors;

authors.push\_back(Author("Author 1"));

authors.push\_back(Author("Author 2"));

authors.push\_back(Author("Author 3"));

authors.push\_back(Author("Author 4"));

cout << "Authors of the paper:" << endl;

for (const Author& author : authors)

{

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

}

string notWorkedAuthor;

for (const Author& author : authors)

{

if (!author.hasWorked())

{

notWorkedAuthor = author.getName();

break;

}

}

cout << "Author who didn't work on the paper: " << notWorkedAuthor << endl;

char addFifthAuthor;

cout << "Do you want to add a 5th author? (y/n): ";

cin >> addFifthAuthor;

if (addFifthAuthor == 'y' || addFifthAuthor == 'Y')

{

string name;

cout << "Enter the name of the 5th author: ";

cin >> name;

authors.push\_back(Author(name));

cout << "5th author added successfully!" << endl;

cout << "Updated list of authors:" << endl;

for (const Author& author : authors) {

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

}

} else {

cout << "No 5th author added." << endl;

}

return 0;

}

**OUTPUT**

