**Operator overloading**

**1.Write a c++ program to overload the ++ operator to increment a variable**

// C++ program to demonstrate

// prefix increment operator overloading

#include <bits/stdc++.h>

using namespace std;

class Integer {

private:

int i;

public:

Integer(int i = 0)

{

this->i = i;

}

Integer& operator++()

{

++i;

return \*this;

}

void display()

{

cout << "i = " << i << endl;

}

};

int main()

{

Integer i1(3);

cout << "Before increment: ";

i1.display();

Integer i2 = ++i1;

cout << "After pre increment: " << endl;

cout << "i1: ";

i1.display();

cout << "i2: ";

i2.display();

}

OUTPUT:

Before increment: i = 3

After pre increment:

i1: i = 4

i2: i = 4

**2.Write a c++ program to overload the + operator to add two variables**

#include <iostream>

using namespace std;

class Adder {

private:

int value;

public:

Adder(int v = 0) : value(v) {}

Adder operator+(const Adder& other) const {

Adder result;

result.value = this->value + other.value;

return result;

}

void display() const {

cout << "Value: " << value << endl;

}

};

int main() {

Adder num1(5);

Adder num2(10);

cout << "First value:" << endl;

num1.display();

cout << "Second value:" << endl;

num2.display();

Adder sum = num1 + num2;

cout << "Sum of the two values:" << endl;

sum.display();

return 0;

}

OUTPUT:

First value:

Value: 5

Second value:

Value: 10

Sum of the two values:

Value: 15

**3.Write a c++ program to overload the << operator to print contents of a user defined class**

#include <iostream>

using namespace std;

class MyClass {

private:

int data1;

double data2;

public:

MyClass(int d1 = 0, double d2 = 0.0) : data1(d1), data2(d2) {}

friend ostream& operator<<(ostream& out, const MyClass& obj) {

out << "Data1: " << obj.data1 << ", Data2: " << obj.data2;

return out;

}

};

int main() {

MyClass obj(10, 3.14);

cout << "Object contents: " << obj << endl;

return 0;

}

OUTPUT:

Object contents: Data1: 10, Data2: 3.14

**4.Write a c++ program to overload the == operator to compare two objects of a user defined class**

#include <iostream>

using namespace std;

class MyClass {

private:

int data;

public:

MyClass(int d = 0) : data(d) {}

bool operator==(const MyClass& other) const {

return this->data == other.data;

}

bool operator!=(const MyClass& other) const {

return !(\*this == other);

}

};

int main() {

MyClass obj1(10);

MyClass obj2(20);

MyClass obj3(10);

if (obj1 == obj2) {

cout << "obj1 and obj2 are equal." << endl;

} else {

cout << "obj1 and obj2 are not equal." << endl;

}

if (obj1 == obj3) {

cout << "obj1 and obj3 are equal." << endl;

} else {

cout << "obj1 and obj3 are not equal." << endl;

}

if (obj1 != obj2) {

cout << "obj1 and obj2 are not equal." << endl;

} else {

cout << "obj1 and obj2 are equal." << endl;

}

return 0;

}

OUTPUT:

obj1 and obj2 are not equal.

obj1 and obj3 are equal.

obj1 and obj2 are not equal.

**5.Write a c++ program to overload the \* operator to multiply two matrices**

#include <iostream>

#include <iomanip>

using namespace std;

class Matrix

{

private:

int m[2][2],i,j;

public:

void read()

{

for(i=0;i<2;i++)

for(j=0;j<2;j++)

cin>>m[i][j];

}

void display()

{

for(i=0;i<2;i++)

{

for(j=0;j<2;j++)

cout<<m[i][j]<<setw(2);

cout<<endl;

}

}

friend Matrix operator\*(Matrix a,Matrix b)

{

int i,j,k,sum=0;

Matrix c;

for(i=0;i<2;i++)

{

for(j=0;j<2;j++)

{

for(k=0;k<2;k++)

{

sum=sum+a.m[i][k]\*b.m[k][j];

}

c.m[i][j]=sum;

sum=0;

}

}

return c;

}

};

int main()

{

Matrix a,b,c;

cout<<"Enter 1st matrix:"<<endl;

a.read();

cout<<"Enter 2nd matrix:"<<endl;

b.read();

c=a\*b;

cout<<"The product is:"<<endl;

c.display();

return 0;

}

OUTPUT:  
Enter 1st matrix:

2 3

5 6

Enter 2nd matrix:

2 8

9 7

The product is:

31 37

64 82

**6.write a c++ program to overload the [] operator to access the elements in an array using index values**

#include <iostream>

using namespace std;

class Array {

private:

int\* data;

int size;

public:

Array(int s) : size(s) {

data = new int[size];

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

data[i] = i + 1;

}

}

int& operator[](int index) {

if (index >= 0 && index < size) {

return data[index];

} else {

cout << "Index out of range." << endl;

static int dummy = -1;

return dummy;

}

}

void display() const {

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

cout << data[i] << " ";

}

cout << endl;

}

~Array() {

delete[] data;

}

};

int main() {

Array arr(5);

cout << "Original array:" << endl;

arr.display();

cout << "Accessing elements using [] operator:" << endl;

cout << "Element at index 2: " << arr[2] << endl;

cout << "Element at index 5: " << arr[5] << endl;

cout << "Updated array after modification:" << endl;

arr[1] = 100;

arr.display();

return 0;

}

OUTPUT:  
Original array:

1 2 3 4 5

Accessing elements using [] operator:

Element at index 2: 3

Index out of range.

Element at index 5: -1

Updated array after modification:

1 100 3 4 5

**7.Write a c++ program to overload the () to call a function with arguments**

#include <iostream>

using namespace std;

class Adder {

public:

int operator()(int a, int b) const {

return a + b;

}

};

int main() {

Adder adder;

int result = adder(10, 20);

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

return 0;

}

OUTPUT:  
Result of addition:

**8.write a c++ program to overload the – operator to subtract two variables**

#include <iostream>

using namespace std;

class Subtract {

private:

int value;

public:

Subtract(int v = 0) : value(v) {}

Subtract operator-(const Subtract& other) const {

Subtract result;

result.value = this->value - other.value;

return result;

}

void display() const {

cout << "Value: " << value << endl;

}

};

int main() {

Subtract num1(20);

Subtract num2(10);

cout << "Original values:" << endl;

num1.display();

num2.display();

Subtract diff = num1 - num2;

cout << "Difference of the two values:" << endl;

diff.display();

return 0;

}

OUTPUT:

Original values:

Value: 20

Value: 10

Difference of the two values:

Value: 10

**9.write a c++ program to overload a function to add two integer numbers and two floating point number separately**

#include <iostream>

using namespace std;

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

int main() {

int num1 = 5, num2 = 10;

double num3 = 3.5, num4 = 7.2;

int sum\_int = add(num1, num2);

cout << "Sum of integers: " << sum\_int << endl;

double sum\_float = add(num3, num4);

cout << "Sum of floating-point numbers: " << sum\_float << endl;

return 0;

}

OUTPUT:  
Sum of integers: 15

Sum of floating-point numbers: 10.7

**10.Write a c++ program to overload the += operator to add two objects of a user defined class**

#include <iostream>

using namespace std;

class MyClass {

private:

int value;

public:

MyClass(int v = 0) : value(v) {}

MyClass& operator+=(const MyClass& other) {

this->value += other.value;

return \*this;

}

void display() const {

cout << "Value: " << value << endl;

}

};

int main() {

MyClass obj1(5);

MyClass obj2(10);

cout << "Original values:" << endl;

obj1.display();

obj2.display();

obj1 += obj2;

cout << "After += operation:" << endl;

obj1.display();

return 0;

}

OUTPUT:

Original values:

Value: 5

Value: 10

After += operation:

Value: 15

**11.write a c++ program to overload a function to find the maximum value from two integer numbers and two floating point number, and two characters separately**

#include <iostream>

using namespace std;

int max(int a, int b) {

return (a > b) ? a : b;

}

double max(double a, double b) {

return (a > b) ? a : b;

}

char max(char a, char b) {

return (a > b) ? a : b;

}

int main() {

int intNum1 = 10, intNum2 = 20;

double doubleNum1 = 3.5, doubleNum2 = 7.2;

char char1 = 'A', char2 = 'Z';

int maxInt = max(intNum1, intNum2);

cout << "Maximum between integers: " << maxInt << endl;

double maxDouble = max(doubleNum1, doubleNum2);

cout << "Maximum between doubles: " << maxDouble << endl;

char maxChar = max(char1, char2);

cout << "Maximum between characters: " << maxChar << endl;

return 0;

}

OUTPUT:

Maximum between integers: 20

Maximum between doubles: 7.2

Maximum between characters: Z

**12.write a c++ program to overload a function to concatenate two strings and two characters arrays separately**

#include <iostream>

#include <cstring>

using namespace std;

// Function to concatenate two strings

string concatenate(const string& str1, const string& str2) {

return str1 + str2;

}

// Function to concatenate two character arrays

char\* concatenate(const char\* arr1, const char\* arr2) {

size\_t len1 = strlen(arr1);

size\_t len2 = strlen(arr2);

char\* result = new char[len1 + len2 + 1];

strcpy(result, arr1);

strcat(result, arr2);

return result;

}

int main() {

string str1 = "Hello, ";

string str2 = "world!";

const char\* charArr1 = "Hello, ";

const char\* charArr2 = "C++";

string concatenatedStr = concatenate(str1, str2);

cout << "Concatenated string: " << concatenatedStr << endl;

char\* concatenatedArr = concatenate(charArr1, charArr2);

cout << "Concatenated character array: " << concatenatedArr << endl;

delete[] concatenatedArr;

return 0;

}

OUTPUT:

Concatenated string: Hello, world!

Concatenated character array: Hello, C++

**13.write a c++ program to overload a function to print an integer array, a double array and a character array separately**

#include <iostream>

using namespace std;

void printArray(const int arr[], int size) {

cout << "Integer Array:";

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

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

}

cout << endl;

}

void printArray(const double arr[], int size) {

cout << "Double Array:";

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

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

}

cout << endl;

}

void printArray(const char arr[], int size) {

cout << "Character Array:";

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

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

}

cout << endl;

}

int main() {

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

double doubleArr[] = {1.1, 2.2, 3.3, 4.4, 5.5};

char charArr[] = {'a', 'b', 'c', 'd', 'e'};

printArray(intArr, 5);

printArray(doubleArr, 5);

printArray(charArr, 5);

return 0;

}

OUTPUT:

Integer Array: 1 2 3 4 5

Double Array: 1.1 2.2 3.3 4.4 5.5

Character Array: a b c d e

**14.write a c++ program to overload a function to find a factorial of an integer number and factorial of a floating-point number separately**

#include <iostream>

#include <cmath>

using namespace std;

unsigned long long factorial(int n) {

if (n < 0) {

cerr << "Error: Factorial is not defined for negative integers." << endl;

return 0;

}

unsigned long long result = 1;

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

result \*= i;

}

return result;

}

double factorial(double n) {

return tgamma(n + 1);

}

int main() {

int intNum = 5;

double doubleNum = 5.5;

unsigned long long intFactorial = factorial(intNum);

cout << "Factorial of " << intNum << " (integer): " << intFactorial << endl;

double doubleFactorial = factorial(doubleNum);

cout << "Factorial of " << doubleNum << " (approximated): " << doubleFactorial << endl;

return 0;

}

OUTPUT:  
Factorial of 5 (integer): 120

Factorial of 5.5 (approximated): 287.885

**15.write a c++ program to overload a function to sort an integer array and a double array**

#include <iostream>

#include <algorithm>

using namespace std;

void sortArray(int arr[], int size) {

sort(arr, arr + size);

}

void sortArray(double arr[], int size) {

sort(arr, arr + size);

}

void displayIntArray(int arr[], int size) {

cout << "Integer Array:";

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

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

}

cout << endl;

}

void displayDoubleArray(double arr[], int size) {

cout << "Double Array:";

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

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

}

cout << endl;

}

int main() {

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

double doubleArr[] = {3.5, 1.1, 2.2, 4.4, 5.5};

int intArrSize = sizeof(intArr) / sizeof(int);

int doubleArrSize = sizeof(doubleArr) / sizeof(double);

sortArray(intArr, intArrSize);

cout << "Sorted Integer Array:" << endl;

displayIntArray(intArr, intArrSize);

sortArray(doubleArr, doubleArrSize);

cout << "Sorted Double Array:" << endl;

displayDoubleArray(doubleArr, doubleArrSize);

return 0;

}

OUTPUT:

Sorted Integer Array:

Integer Array: 1 2 3 4 5

Sorted Double Array:

Double Array: 1.1 2.2 3.5 4.4 5.5

**16.write a c++ program to overload a function to calculate the power of an integer number and power of a floating-point number separately**

#include <iostream>

#include <cmath>

using namespace std;

long long power(int base, int exponent) {

long long result = 1;

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

result \*= base;

}

return result;

}

double power(double base, int exponent) {

return pow(base, exponent);

}

int main() {

int intBase = 2;

int intExponent = 5;

double doubleBase = 2.5;

int doubleExponent = 3;

long long intPower = power(intBase, intExponent);

cout << "Power of " << intBase << " raised to " << intExponent << " (integer): " << intPower << endl;

double doublePower = power(doubleBase, doubleExponent);

cout << "Power of " << doubleBase << " raised to " << doubleExponent << " (floating-point): " << doublePower << endl;

return 0;

}

OUTPUT:

Power of 2 raised to 5 (integer): 32

Power of 2.5 raised to 3 (floating-point): 15.625

**17.write a c++ program to overload a function to find an absolute value of an integer number and absolute value of a floating-point number separately**

#include <iostream>

#include <cmath>

using namespace std;

int absolute(int num) {

return abs(num);

}

double absolute(double num) {

return fabs(num);

}

int main() {

int intNum = -5;

double doubleNum = -3.5;

int absInt = absolute(intNum);

cout << "Absolute value of " << intNum << " (integer): " << absInt << endl;

double absDouble = absolute(doubleNum);

cout << "Absolute value of " << doubleNum << " (floating-point): " << absDouble << endl;

return 0;

}

OUTPUT:  
Absolute value of -5 (integer): 5

Absolute value of -3.5 (floating-point): 3.5

**Inheritance and pointers**

1. **Create a base class called Shape with data members for height and width. Derive two classes Rectangle and Triangle from the base class. Write member functions to calculate the area and perimeter of each class**

#include <iostream>

#include <cmath>

using namespace std;

class Shape {

protected:

double height;

double width;

public:

Shape(double h = 0, double w = 0) : height(h), width(w) {}

virtual double calculateArea() const = 0;

virtual double calculatePerimeter() const = 0;

};

class Rectangle : public Shape {

public:

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

double calculateArea() const override {

return height \* width;

}

double calculatePerimeter() const override {

return 2 \* (height + width);

}

};

class Triangle : public Shape {

public:

Triangle(double h, double w) : Shape(h, w) {}

double calculateArea() const override {

return 0.5 \* height \* width;

}

double calculatePerimeter() const override {

double hypotenuse = sqrt(height \* height + width \* width);

return height + width + hypotenuse;

}

};

int main()

{

Rectangle rect(5, 10);

Triangle tri(4, 6);

cout << "Rectangle:" << endl;

cout << "Area: " << rect.calculateArea() << endl;

cout << "Perimeter: " << rect.calculatePerimeter() << endl;

cout << "\nTriangle:" << endl;

cout << "Area: " << tri.calculateArea() << endl;

cout << "Perimeter: " << tri.calculatePerimeter() << endl;

return 0;

}

OUTPUT:

Rectangle:

Area: 50

Perimeter: 30

Triangle:

Area: 12

Perimeter: 17.2111

1. **Create a base class called vehicle with data members for make, model, and year. Derive two classes Car and Truck from the base class. The Car class should have additional data members for seating capacity and fuel type, while the Truck class should have additional data members for payload capacity and towing capacity. Write member functions to get and set the data members for each class**

#include <iostream>

#include <string>

using namespace std;

class Vehicle {

protected:

string make;

string model;

int year;

public:

Vehicle(const string& mk = "", const string& mdl = "", int yr = 0)

: make(mk), model(mdl), year(yr) {}

string getMake() const { return make; }

void setMake(const string& mk) { make = mk; }

string getModel() const { return model; }

void setModel(const string& mdl) { model = mdl; }

int getYear() const { return year; }

void setYear(int yr) { year = yr; }

};

class Car : public Vehicle {

private:

int seatingCapacity;

string fuelType;

public:

Car(const string& mk = "", const string& mdl = "", int yr = 0,

int seats = 0, const string& fuel = "")

: Vehicle(mk, mdl, yr), seatingCapacity(seats), fuelType(fuel) {}

int getSeatingCapacity() const { return seatingCapacity; }

void setSeatingCapacity(int seats) { seatingCapacity = seats; }

string getFuelType() const { return fuelType; }

void setFuelType(const string& fuel) { fuelType = fuel; }

};

class Truck : public Vehicle {

private:

double payloadCapacity;

double towingCapacity;

public:

Truck(const string& mk = "", const string& mdl = "", int yr = 0,

double payload = 0.0, double towing = 0.0)

: Vehicle(mk, mdl, yr), payloadCapacity(payload), towingCapacity(towing) {}

double getPayloadCapacity() const { return payloadCapacity; }

void setPayloadCapacity(double payload) { payloadCapacity = payload; }

double getTowingCapacity() const { return towingCapacity; }

void setTowingCapacity(double towing) { towingCapacity = towing; }

};

int main() {

Car myCar("Toyota", "Camry", 2022, 5, "Gasoline");

Truck myTruck("Ford", "F-150", 2021, 1500.0, 8000.0);

myCar.setMake("Honda");

myCar.setModel("Civic");

myCar.setYear(2023);

myCar.setSeatingCapacity(4);

myCar.setFuelType("Electric");

cout << "Car Details:" << endl;

cout << "Make: " << myCar.getMake() << endl;

cout << "Model: " << myCar.getModel() << endl;

cout << "Year: " << myCar.getYear() << endl;

cout << "Seating Capacity: " << myCar.getSeatingCapacity() << endl;

cout << "Fuel Type: " << myCar.getFuelType() << endl;

myTruck.setPayloadCapacity(2000.0);

myTruck.setTowingCapacity(10000.0);

cout << "\nTruck Details:" << endl;

cout << "Make: " << myTruck.getMake() << endl;

cout << "Model: " << myTruck.getModel() << endl;

cout << "Year: " << myTruck.getYear() << endl;

cout << "Payload Capacity: " << myTruck.getPayloadCapacity() << " lbs" << endl;

cout << "Towing Capacity: " << myTruck.getTowingCapacity() << " lbs" << endl;

return 0;

}

OUTPUT:

Car Details:

Make: Honda

Model: Civic

Year: 2023

Seating Capacity: 4

Fuel Type: Electric

Truck Details:

Make: Ford

Model: F-150

Year: 2021

Payload Capacity: 2000 lbs

Towing Capacity: 10000 lbs

1. **Create a base class called Animal with data members for name, species, and age. Derive two classes Cat and Dog from the base class. The Cat class should have additional data members for color and breed, while the Dog class should have additional data members for weight and breed. Write member functions to get and set the data members for each class**

#include <iostream>

#include <string>

using namespace std;

class Animal {

protected:

string name;

string species;

int age;

public:

Animal(const string& nm = "", const string& sp = "", int ag = 0)

: name(nm), species(sp), age(ag) {}

string getName() const { return name; }

void setName(const string& nm) { name = nm; }

string getSpecies() const { return species; }

void setSpecies(const string& sp) { species = sp; }

int getAge() const { return age; }

void setAge(int ag) { age = ag; }

};

class Cat : public Animal {

private:

string color;

string breed;

public:

Cat(const string& nm = "", const string& sp = "", int ag = 0,

const string& col = "", const string& br = "")

: Animal(nm, sp, ag), color(col), breed(br) {}

string getColor() const { return color; }

void setColor(const string& col) { color = col; }

string getBreed() const { return breed; }

void setBreed(const string& br) { breed = br; }

};

class Dog : public Animal {

private:

double weight;

string breed;

public:

Dog(const string& nm = "", const string& sp = "", int ag = 0,

double wt = 0.0, const string& br = "")

: Animal(nm, sp, ag), weight(wt), breed(br) {}

double getWeight() const { return weight; }

void setWeight(double wt) { weight = wt; }

string getBreed() const { return breed; }

void setBreed(const string& br) { breed = br; }

};

int main() {

Cat myCat("Whiskers", "Cat", 3, "Gray", "Persian");

Dog myDog("Buddy", "Dog", 5, 20.5, "Golden Retriever");

myCat.setName("Mittens");

myCat.setSpecies("Cat");

myCat.setAge(4);

myCat.setColor("Black");

myCat.setBreed("Siamese");

cout << "Cat Details:" << endl;

cout << "Name: " << myCat.getName() << endl;

cout << "Species: " << myCat.getSpecies() << endl;

cout << "Age: " << myCat.getAge() << " years" << endl;

cout << "Color: " << myCat.getColor() << endl;

cout << "Breed: " << myCat.getBreed() << endl;

myDog.setName("Rocky");

myDog.setSpecies("Dog");

myDog.setAge(6);

myDog.setWeight(25.3);

myDog.setBreed("Labrador Retriever");

cout << "\nDog Details:" << endl;

cout << "Name: " << myDog.getName() << endl;

cout << "Species: " << myDog.getSpecies() << endl;

cout << "Age: " << myDog.getAge() << " years" << endl;

cout << "Weight: " << myDog.getWeight() << " kg" << endl;

cout << "Breed: " << myDog.getBreed() << endl;

return 0;

}

OUTPUT:  
Cat Details:

Name: Mittens

Species: Cat

Age: 4 years

Color: Black

Breed: Siamese

Dog Details:

Name: Rocky

Species: Dog

Age: 6 years

Weight: 25.3 kg

Breed: Labrador Retriever

1. **Create a base class called Employee with data members for name, d, and salary Derive two classes Manager and Engineer from the base class. The Manager class should have additional data members for department and bonus, while the Engineer class should have additional data members for specialty and hours. Write member functions to get and set the data members for each class**

#include <iostream>

#include <string>

using namespace std;

class Employee {

protected:

string name;

int id;

double salary;

public:

Employee(const string& n = "", int i = 0, double sal = 0.0)

: name(n), id(i), salary(sal) {}

string getName() const { return name; }

void setName(const string& n) { name = n; }

int getId() const { return id; }

void setId(int i) { id = i; }

double getSalary() const { return salary; }

void setSalary(double sal) { salary = sal; }

};

class Manager : public Employee {

private:

string department;

double bonus;

public:

Manager(const string& n = "", int i = 0, double sal = 0.0,

const string& dept = "", double b = 0.0)

: Employee(n, i, sal), department(dept), bonus(b) {}

string getDepartment() const { return department; }

void setDepartment(const string& dept) { department = dept; }

double getBonus() const { return bonus; }

void setBonus(double b) { bonus = b; }

};

class Engineer : public Employee {

private:

string specialty;

int hours;

public:

Engineer(const string& n = "", int i = 0, double sal = 0.0,

const string& spec = "", int hrs = 0)

: Employee(n, i, sal), specialty(spec), hours(hrs) {}

string getSpecialty() const { return specialty; }

void setSpecialty(const string& spec) { specialty = spec; }

int getHours() const { return hours; }

void setHours(int hrs) { hours = hrs; }

};

int main() {

Manager manager("John Doe", 1001, 75000.0, "Sales", 5000.0);

Engineer engineer("Jane Smith", 2001, 65000.0, "Software Development", 40);

manager.setName("Alice Johnson");

manager.setId(1002);

manager.setSalary(80000.0);

manager.setDepartment("Marketing");

manager.setBonus(6000.0);

cout << "Manager Details:" << endl;

cout << "Name: " << manager.getName() << endl;

cout << "ID: " << manager.getId() << endl;

cout << "Salary: $" << manager.getSalary() << endl;

cout << "Department: " << manager.getDepartment() << endl;

cout << "Bonus: $" << manager.getBonus() << endl;

engineer.setName("Bob Williams");

engineer.setId(2002);

engineer.setSalary(70000.0);

engineer.setSpecialty("Data Science");

engineer.setHours(45);

cout << "\nEngineer Details:" << endl;

cout << "Name: " << engineer.getName() << endl;

cout << "ID: " << engineer.getId() << endl;

cout << "Salary: $" << engineer.getSalary() << endl;

cout << "Specialty: " << engineer.getSpecialty() << endl;

cout << "Hours: " << engineer.getHours() << endl;

return 0;

}

OUTPUT:

Manager Details:

Name: Alice Johnson

ID: 1002

Salary: $80000

Department: Marketing

Bonus: $6000

Engineer Details:

Name: Bob Williams

ID: 2002

Salary: $70000

Specialty: Data Science

Hours: 45

1. **Create a base class called Person with data members for name, age, and gender. Derive two classes Student and Teacher from the base class. The Student class should have additional data members for roll number and class, while the Teacher class should have additional data members for subject and salary. Write member functions to get and set the data members for each class.**

#include <iostream>

#include <string>

using namespace std;

class Person {

protected:

string name;

int age;

char gender;

public:

Person(const string& n = "", int a = 0, char g = ' ')

: name(n), age(a), gender(g) {}

string getName() const { return name; }

void setName(const string& n) { name = n; }

int getAge() const { return age; }

void setAge(int a) { age = a; }

char getGender() const { return gender; }

void setGender(char g) { gender = g; }

};

class Student : public Person {

private:

int rollNumber;

string className;

public:

Student(const string& n = "", int a = 0, char g = ' ',

int roll = 0, const string& cls = "")

: Person(n, a, g), rollNumber(roll), className(cls) {}

int getRollNumber() const { return rollNumber; }

void setRollNumber(int roll) { rollNumber = roll; }

string getClassName() const { return className; }

void setClassName(const string& cls) { className = cls; }

};

class Teacher : public Person {

private:

string subject;

double salary;

public:

Teacher(const string& n = "", int a = 0, char g = ' ',

const string& sub = "", double sal = 0.0)

: Person(n, a, g), subject(sub), salary(sal) {}

string getSubject() const { return subject; }

void setSubject(const string& sub) { subject = sub; }

double getSalary() const { return salary; }

void setSalary(double sal) { salary = sal; }

};

int main() {

Student student1("John Doe", 18, 'M', 101, "12th Grade");

Teacher teacher1("Ms. Smith", 35, 'F', "Mathematics", 60000.0);

student1.setName("Jane Smith");

student1.setAge(17);

student1.setGender('F');

student1.setRollNumber(102);

student1.setClassName("11th Grade");

cout << "Student Details:" << endl;

cout << "Name: " << student1.getName() << endl;

cout << "Age: " << student1.getAge() << " years" << endl;

cout << "Gender: " << student1.getGender() << endl;

cout << "Roll Number: " << student1.getRollNumber() << endl;

cout << "Class: " << student1.getClassName() << endl;

teacher1.setSubject("Computer Science");

teacher1.setSalary(65000.0);

cout << "\nTeacher Details:" << endl;

cout << "Name: " << teacher1.getName() << endl;

cout << "Age: " << teacher1.getAge() << " years" << endl;

cout << "Gender: " << teacher1.getGender() << endl;

cout << "Subject: " << teacher1.getSubject() << endl;

cout << "Salary: $" << teacher1.getSalary() << endl;

return 0;

}

OUTPUT:

Student Details:

Name: Jane Smith

Age: 17 years

Gender: F

Roll Number: 102

Class: 11th Grade

Teacher Details:

Name: Ms. Smith

Age: 35 years

Gender: F

Subject: Computer Science

Salary: $65000

1. **Write a C++ program to create a pointer to an integer and display its value.**

#include <iostream>

using namespace std;

int main() {

int\* ptr;

int number = 42;

ptr = &number;

cout << "Value of integer through pointer: " << \*ptr << endl;

return 0;

}

OUTPUT:  
Value of integer through pointer: 42

1. **Write a C++ program to create a pointer to a float and display its value.**

#include <iostream>

using namespace std;

int main() {

float\* ptr;

float number = 3.14;

ptr = &number;

cout << "Value of float through pointer: " << \*ptr << endl;

return 0;

}

OUTPUT:  
Value of float through pointer: 3.14

1. **Write a C++ program to create a pointer to a char and display its value.**

#include <iostream>

using namespace std;

int main() {

char\* ptr;

char character = 'A';

ptr = &character;

cout << "Value of char through pointer: " << \*ptr << endl;

return 0;

}

OUTPUT:

Value of char through pointer: A

1. **Write a C++ program to create a pointer to a double and display its value.**

#include <iostream>

using namespace std;

int main() {

double\* ptr;

double number = 3.14159;

ptr = &number;

cout << "Value of double through pointer: " << \*ptr << endl;

return 0;

}

OUTPUT:

Value of double through pointer: 3.14159

1. **Write a C++ program to create a pointer to a string and display its value.**

#include <iostream>

using namespace std;

int main() {

char\* ptr;

ptr = "shaik, jigunu";

cout << "Value of string through pointer: " << ptr << endl;

return 0;

}

OUTPUT:

Value of string through pointer: shaik, jigunu

1. **Write a C++ program to create a pointer to an array of elements and display its value.**

#include <iostream>

using namespace std;

int main() {

int numbers[] = {10, 20, 30, 40, 50};

int\* ptr = numbers;

cout << "Array elements through pointer:" << endl;

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

cout << \*(ptr + i) << " ";

}

cout << endl;

return 0;

}

OUTPUT:  
Array elements through pointer:

10 20 30 40 50

1. **Write a C++ program to create a pointer to an array of character and display its value.**

#include <iostream>

int main() {

char my\_array[] = "Hello, World!";

char\* ptr = my\_array;

std::cout << "The value of the pointer is: " << ptr << std::endl;

return 0;

}

OUTPUT:

The value of the pointer is: Hello, World!

1. **Write a C++ program to create a pointer to an array of floats and display its value.**

#include <iostream>

int main() {

float my\_floats[] = {1.23, 4.56, 7.89, 10.11, 12.13};

float\* ptr = my\_floats;

std::cout << "The value of the pointer is: " << ptr << std::endl;

return 0;

}

OUTPUT:

The value of the pointer is: 0x6ffe00

1. **Write a C++ program to create a pointer to an object and display its attributes.**

#include <iostream>

#include <string>

using namespace std;

class Person {

private:

string name;

int age;

public:

Person(const string& n, int a) : name(n), age(a) {}

string getName() const { return name; }

int getAge() const { return age; }

};

int main() {

Person person("Alice", 25);

Person\* ptr = &person;

cout << "Name: " << ptr->getName() << endl;

cout << "Age: " << ptr->getAge() << " years" << endl;

return 0;

}

OUTPUT:

Name: Alice

Age: 25 years

1. **Write a C++ program to create a pointer to a function and call the function using the pointer.**

#include <iostream>

using namespace std;

// Function prototypes

void displayMessage();

int addNumbers(int a, int b);

int main() {

void (\*ptrDisplay)() = &displayMessage;

(\*ptrDisplay)();

int (\*ptrAdd)(int, int) = &addNumbers;

int result = (\*ptrAdd)(5, 10);

cout << "Result of adding numbers: " << result << endl;

return 0;

}

void displayMessage() {

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

}

int addNumbers(int a, int b) {

return a + b;

}

OUTPUT:

Hello, world!

Result of adding numbers: 15

**Polymorphism**

1. **Create a base class called Person with a virtual function work (). Derive two classes Employee and Manager from the base class. Implement the work () function for each class**

#include <iostream>

#include <string>

using namespace std;

class Person {

public:

virtual void work() {

cout << "Person is working." << endl;

}

};

class Employee : public Person {

public:

void work() override {

cout << "Employee is working on assigned tasks." << endl;

}

};

class Manager : public Person {

public:

void work() override {

cout << "Manager is supervising and coordinating work." << endl;

}

};

int main() {

Person\* personPtr;

Employee emp;

Manager mgr;

personPtr = &emp;

personPtr->work();

personPtr = &mgr;

personPtr->work();

return 0;

}

OUTPUT:  
Employee is working on assigned tasks.

Manager is supervising and coordinating work.

1. **Create a base class called Animal with a virtual function eat (). Derive two classes Herbivore and Carnivore from the base class. Implement the eat function for each class.**

#include <iostream>

#include <string>

using namespace std;

class Animal {

public:

virtual void eat() {

cout << "Animal is eating." << endl;

}

};

class Herbivore : public Animal {

public:

void eat() override {

cout << "Herbivore is eating plants." << endl;

}

};

class Carnivore : public Animal {

public:

void eat() override {

cout << "Carnivore is eating other animals." << endl;

}

};

int main() {

Animal\* animalPtr;

Herbivore herb;

Carnivore carn;

animalPtr = &herb;

animalPtr->eat();

animalPtr = &carn;

animalPtr->eat();

return 0;

}

OUTPUT:

Herbivore is eating plants.

Carnivore is eating other animals.

1. **Create a base class called Shape with virtual functions area () and volume (). Derive two classes Sphere and Cylinder from the base class. Implement the area and volume () functions for each class**

#include <iostream>

#include <cmath>

using namespace std;

class Shape {

public:

virtual double area() const = 0;

virtual double volume() const = 0;

};

class Sphere : public Shape {

private:

double radius;

public:

Sphere(double r) : radius(r) {}

double area() const override {

return 4 \* M\_PI \* radius \* radius;

}

double volume() const override {

return (4.0 / 3.0) \* M\_PI \* radius \* radius \* radius;

}

};

class Cylinder : public Shape {

private:

double radius;

double height;

public:

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

double area() const override {

return 2 \* M\_PI \* radius \* (radius + height);

}

double volume() const override {

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

}

};

int main() {

Sphere sphere(5.0);

Cylinder cylinder(3.0, 7.0);

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

cout << "Volume of Sphere: " << sphere.volume() << endl;

cout << "\nArea of Cylinder: " << cylinder.area() << endl;

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

return 0;

}

OUTPUT:

Area of Sphere: 314.159

Volume of Sphere: 523.599

Area of Cylinder: 188.496

Volume of Cylinder: 197.92

1. **Create a base class called Person with a virtual function greet). Derive two classes Student and Teacher from the base class. implement the greet) function for each class**

#include <iostream>

#include <string>

using namespace std;

class Person {

public:

virtual void greet() const {

cout << "Hello, I am a person." << endl;

}

};

class Student : public Person {

public:

void greet() const override {

cout << "Hello, I am a student." << endl;

}

};

class Teacher : public Person {

public:

void greet() const override {

cout << "Hello, I am a teacher." << endl;

}

};

int main() {

Person\* personPtr;

Student student;

Teacher teacher;

personPtr = &student;

personPtr->greet();

personPtr = &teacher;

personPtr->greet();

return 0;

}

OUTPUT:  
Hello, I am a student.

Hello, I am a teacher.

1. **Create a base class called Person with a virtual function greet). Derive two classes Student and Teacher from the base class. implement the greet) function for each class**

#include <iostream>

#include <string>

using namespace std;

class Person {

public:

virtual void greet() const {

cout << "Hello, I am a person." << endl;

}

};

class Student : public Person {

public:

void greet() const override {

cout << "Hello, I am a student." << endl;

}

};

class Teacher : public Person {

public:

void greet() const override {

cout << "Hello, I am a teacher." << endl;

}

};

int main() {

Person\* personPtr;

Student student;

Teacher teacher;

personPtr = &student;

personPtr->greet();

personPtr = &teacher;

personPtr->greet();

return 0;

}

OUTPUT:  
Hello, I am a student.

Hello, I am a teacher.

1. **Create a base class called Shape with virtual functions area( ) and perimeter(). Derive two classes Rectangle and Triangle from the base class. Implement the area () and perimeter () functions for each class.**

#include <iostream>

#include <cmath>

using namespace std;

class Shape {

public:

virtual double area() const = 0;

virtual double perimeter() const = 0;

};

class Rectangle : public Shape {

private:

double width;

double height;

public:

Rectangle(double w, double h) : width(w), height(h) {}

double area() const override {

return width \* height;

}

double perimeter() const override {

return 2 \* (width + height);

}

};

class Triangle : public Shape {

private:

double base;

double height;

public:

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

double area() const override {

return 0.5 \* base \* height;

}

double perimeter() const override {

double side = sqrt(base \* base + 0.25 \* height \* height);

return 2 \* side + base;

}

};

int main() {

Rectangle rect(5, 10);

Triangle tri(4, 6);

cout << "Rectangle:" << endl;

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

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

cout << "\nTriangle:" << endl;

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

cout << "Perimeter: " << tri.perimeter() << endl;

return 0;

}

OUTPUT:  
Rectangle:

Area: 50

Perimeter: 30

Triangle:

Area: 12

Perimeter: 14

1. **Create a base class called Vehicle with a virtual function drive(). Derive two classes Car and Truck from the base class. Implement the drive() function for each class.**

#include <iostream>

using namespace std;

class Vehicle {

public:

virtual void drive() const = 0;

};

class Car : public Vehicle {

public:

void drive() const override {

cout << "Car is driving on the road." << endl;

}

};

class Truck : public Vehicle {

public:

void drive() const override {

cout << "Truck is transporting goods on the highway." << endl;

}

};

int main() {

Car myCar;

Truck myTruck;

myCar.drive();

myTruck.drive();

return 0;

}

OUTPUT:  
Car is driving on the road.

Truck is transporting goods on the highway

1. **Create a base class called Employee with a virtual function calculate Pay(). Derive two classes Manager and Engineer from the base class. Implement the calculatePay () function for each class.**

#include <iostream>

#include <string>

using namespace std;

class Employee {

protected:

string name;

double salary;

public:

Employee(const string& n, double s) : name(n), salary(s) {}

virtual double calculatePay() const = 0;

};

class Manager : public Employee {

private:

double bonus;

public:

Manager(const string& n, double s, double b) : Employee(n, s), bonus(b) {}

double calculatePay() const override {

return salary + bonus;

}

};

class Engineer : public Employee {

private:

double hoursWorked;

double hourlyRate;

public:

Engineer(const string& n, double s, double hw, double hr) : Employee(n, s), hoursWorked(hw), hourlyRate(hr) {}

double calculatePay() const override {

return salary + (hoursWorked \* hourlyRate);

}

};

int main() {

Manager manager("John Doe", 50000, 5000);

Engineer engineer("Jane Smith", 60000, 40, 25);

cout << "Manager's pay: $" << manager.calculatePay() << endl;

cout << "Engineer's pay: $" << engineer.calculatePay() << endl;

return 0;

}

OUTPUT:

Manager's pay: $55000

Engineer's pay: $61000

1. **Create a base class called Animal with a virtual function speak(). Derive two classes Cat and Dog from the base class. Implement the speak() function for each class.**

#include <iostream>

#include <string>

using namespace std;

class Animal {

public:

virtual void speak() const = 0;

};

class Cat : public Animal {

public:

void speak() const override {

cout << "Meow!" << endl;

}

};

class Dog : public Animal {

public:

void speak() const override {

cout << "Woof!" << endl;

}

};

int main() {

Cat myCat;

Dog myDog;

myCat.speak();

myDog.speak();

return 0;

}

OUTPUT:

Meow!

Woof!

1. **Create a base class called Shape with a virtual function area(). Derive two classes Rectangle and Circle from the base class. Implement the area() function for each class.**

#include <iostream>

using namespace std;

class Shape {

public:

virtual double area() const = 0;

};

class Rectangle : public Shape {

private:

double width;

double height;

public:

Rectangle(double w, double h) : width(w), height(h) {}

double area() const override {

return width \* height;

}

};

class Circle : public Shape {

private:

double radius;

public:

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

double area() const override {

return 3.14159 \* radius \* radius;

}

};

int main() {

Rectangle rect(5, 10);

Circle circle(4);

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

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

return 0;

}

OUTPUT:

Area of Rectangle: 50

Area of Circle: 50.2654

**Exception Handling**

1. **Write a c++ program to demonstrate to use of the finally block for handling exceptions**

#include <iostream>

#include <stdexcept>

using namespace std;

class Resource {

public:

Resource() {

cout << "Resource acquired." << endl;

}

void doSomething() {

cout << "Performing operations with acquired resource." << endl;

throw runtime\_error("Error occurred during operation.");

}

~Resource() {

cout << "Resource released." << endl;

}

};

int main() {

try {

Resource res;

res.doSomething();

} catch (const exception& e) {

cerr << "Exception caught: " << e.what() << endl;

}

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

return 0;

}

OUTPUT:

Resource acquired.

Performing operations with acquired resource.

Resource released.

Exception caught: Error occurred during operation.

Program continues execution after exception handling.

1. **Write a c++ program to demonstrate to use of nested try-catch blocks for handling exceptions**

#include <iostream>

#include <stdexcept>

using namespace std;

int main() {

try {

try {

int numerator, denominator;

cout << "Enter numerator: ";

cin >> numerator;

cout << "Enter denominator: ";

cin >> denominator;

if (denominator == 0) {

throw runtime\_error("Division by zero not allowed.");

}

cout << "Result: " << numerator / denominator << endl;

} catch (const exception& e) {

cerr << "Inner Exception caught: " << e.what() << endl;

}

throw runtime\_error("Outer Exception thrown.");

} catch (const exception& e) {

cerr << "Outer Exception caught: " << e.what() << endl;

}

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

return 0;

}

OUTPUT:  
Enter numerator: 54

Enter denominator: 7

Result: 7

Outer Exception caught: Outer Exception thrown.

Program continues execution after exception handling.

1. **Write a c++ program to demonstrate to use of user-defined exception for handling custom exception**

#include <iostream>

#include <stdexcept>

class MyCustomException : public std::runtime\_error {

public:

MyCustomException(const char\* message) : std::runtime\_error(message) {}

};

double divide(double a, double b) {

if (b == 0) {

throw MyCustomException("Division by zero is not allowed");

}

return a / b;

}

int main() {

try {

double result = divide(10, 0);

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

} catch (const MyCustomException& e) {

std::cout << "Custom Exception: " << e.what() << std::endl;

}

return 0;

}

OUTPUT:

Custom Exception: Division by zero is not allowed

1. **Write a c++ program to demonstrate to use of the standard class for handling exceptions**

#include <iostream>

#include <stdexcept>

using namespace std;

int main() {

try {

int x;

cout << "Enter a number greater than 10: ";

cin >> x;

if (x <= 10) {

throw std::out\_of\_range("Input is out of range.");

}

cout << "You entered: " << x << endl;

} catch (const std::exception& e) {

cerr << "Exception caught: " << e.what() << endl;

}

return 0;

}

OUTPUT:  
Enter a number greater than 10: 23

You entered: 23

1. **Write a c++ program to demonstrate to use of the keyword to throw an exception**

#include <iostream>

#include <stdexcept>

using namespace std;

void divide(int numerator, int denominator) {

if (denominator == 0) {

throw std::invalid\_argument("Division by zero is not allowed.");

}

cout << "Result of division: " << numerator / denominator << endl;

}

int main() {

try {

int x, y;

cout << "Enter numerator: ";

cin >> x;

cout << "Enter denominator: ";

cin >> y;

divide(x, y);

} catch (const std::exception& e) {

cerr << "Exception caught: " << e.what() << endl;

}

return 0;

}

OUTPUT:

Enter numerator: 56

Enter denominator: 20

Result of division: 2

1. **Write a c++ program to demonstrate to use of multiple catch blocks for handling different types of exceptions**

#include <iostream>

#include <stdexcept>

using namespace std;

void processInput(int input) {

if (input < 0) {

throw std::invalid\_argument("Negative input not allowed.");

} else if (input == 0) {

throw std::runtime\_error("Zero input encountered.");

} else if (input > 100) {

throw std::out\_of\_range("Input out of range (> 100).");

}

cout << "Input is valid: " << input << endl;

}

int main() {

try {

int num;

cout << "Enter a number: ";

cin >> num;

processInput(num);

} catch (const std::invalid\_argument& e) {

cerr << "Invalid Argument Exception caught: " << e.what() << endl;

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

cerr << "Out of Range Exception caught: " << e.what() << endl;

} catch (const std::runtime\_error& e) {

cerr << "Runtime Error Exception caught: " << e.what() << endl;

} catch (...) {

cerr << "Unknown Exception caught." << endl;

}

return 0;

}

OUTPUT:

Enter a number: 60

Input is valid: 60

1. **Write a c++ program to demonstrate to use of try-catch blocks for handling exceptions**

#include <iostream>

#include <stdexcept>

using namespace std;

int main() {

try {

int numerator, denominator;

cout << "Enter numerator: ";

cin >> numerator;

cout << "Enter denominator: ";

cin >> denominator;

if (denominator == 0) {

throw std::invalid\_argument("Division by zero is not allowed.");

}

cout << "Result: " << numerator / denominator << endl;

} catch (const std::exception& e) {

cerr << "Exception caught: " << e.what() << endl;

}

return 0;

}

OUTPUT:

Enter numerator: 21

Enter denominator: 3

Result: 7