A.Ganesh reddy

192110023

**15 th date programs**

1. #include<iostream>

#include<string>

using namespace std;

class Car

{

private:

string model;

public:

Car(){

model="unknown";

}

string getModel()

{

return model;

}

};

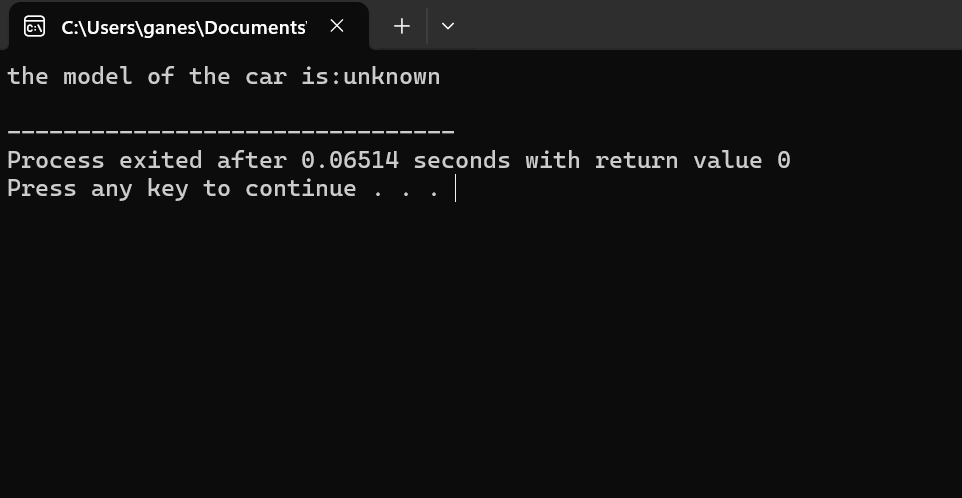
int main(){

Car myCar;

cout<<"the model of the car is:"<<myCar.getModel()<<endl;

return 0;

}



2. #include<iostream>

using namespace std;

class car

{

private:

string model;

public:

car()

{

model="unknown";

}

car(string newModel)

{

model=newModel;

}

string getModel()

{

return model;

}

};

int main()

{

string userInput;

cout<<"enter model of the car:";

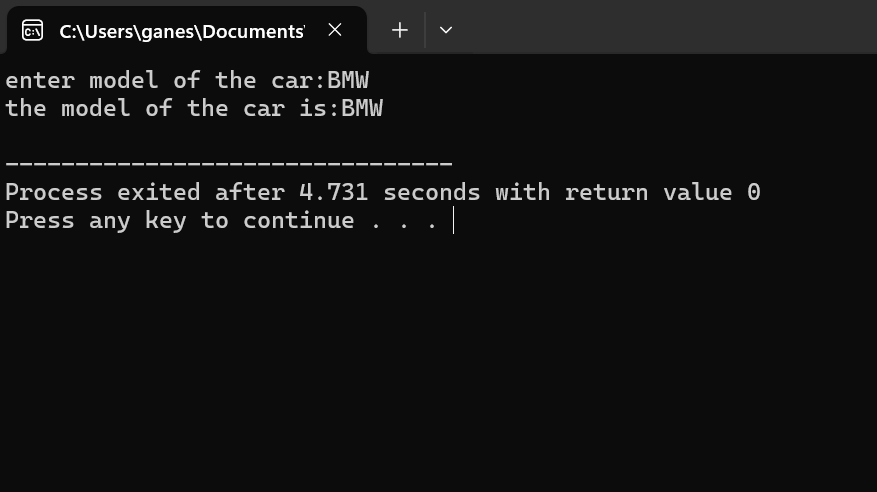
cin>>userInput;

car mycar(userInput);

cout<<"the model of the car is:"<<mycar.getModel()<<"\n";

return 0;

}



3.

#include<iostream>

using namespace std;

class rectangle

{

private:

int length;

int width;

public:

rectangle()

{

length=1;

width=1;

}

rectangle(int len,int wid)

{

length=len;

width=wid;

}

int getLength()

{

return length;

}

int getWidth()

{

return width;

}

};

int main()

{

int userInput1,userInput2;

cout<<"enter length of rectangle:";

cin>>userInput1;

cout<<"enter width of rectangle:";

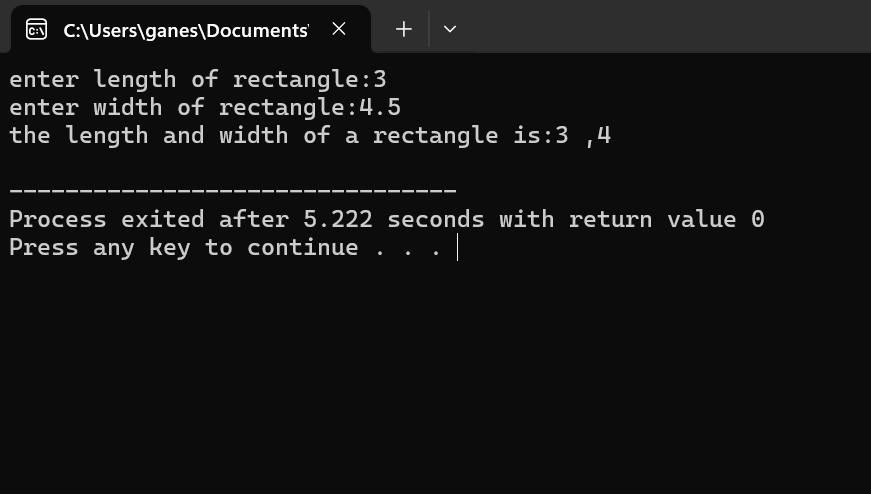
cin>>userInput2;

rectangle myrect(userInput1,userInput2);

cout<<"the length and width of a rectangle is:"<<myrect.getLength()<<" ,"<<myrect.getWidth()<<"\n";

return 0;

}



4.

#include <iostream>

#include <string>

using namespace std;

class Person {

private:

std::string name;

int age;

public:

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

Person(const Person &other) {

name = other.name;

age = other.age;

}

std::string getName() {

return name;

}

int getAge() {

return age;

}

};

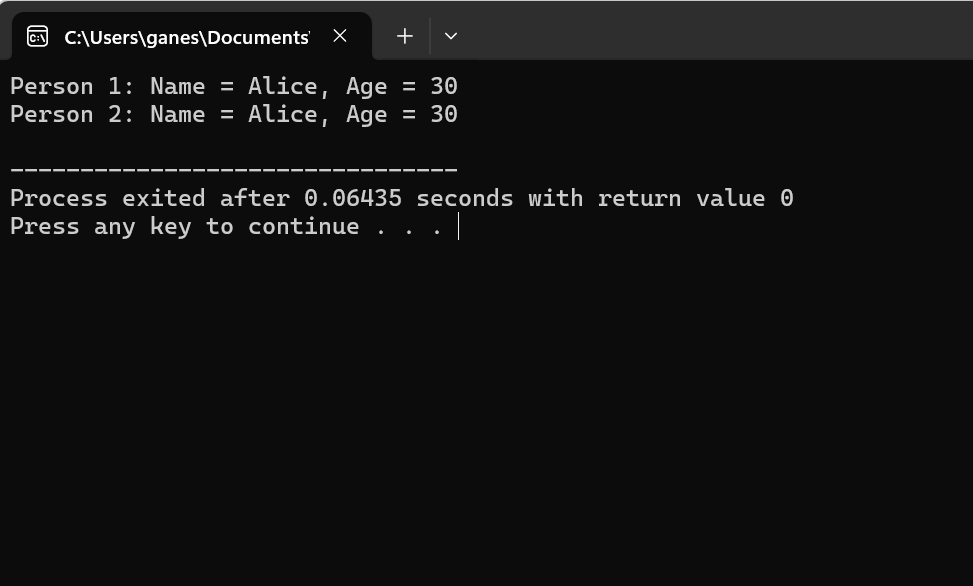
int main() {

Person person1("Alice", 30);

Person person2 = person1;

std::cout << "Person 1: Name = " << person1.getName() << ", Age = " << person1.getAge() << std::endl;

std::cout << "Person 2: Name = " << person2.getName() << ", Age = " << person2.getAge() << std::endl;

return

5.

#include <iostream>

class DynamicArray {

private:

int\* array;

int size;

public:

DynamicArray(int sz) {

size = sz;

array = new int[size];

}

~DynamicArray() {

delete[] array;

}

void displayElements() {

std::cout << "Elements of the DynamicArray: ";

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

std::cout << array[i] << " ";

}

std::cout << std::endl;

}

};

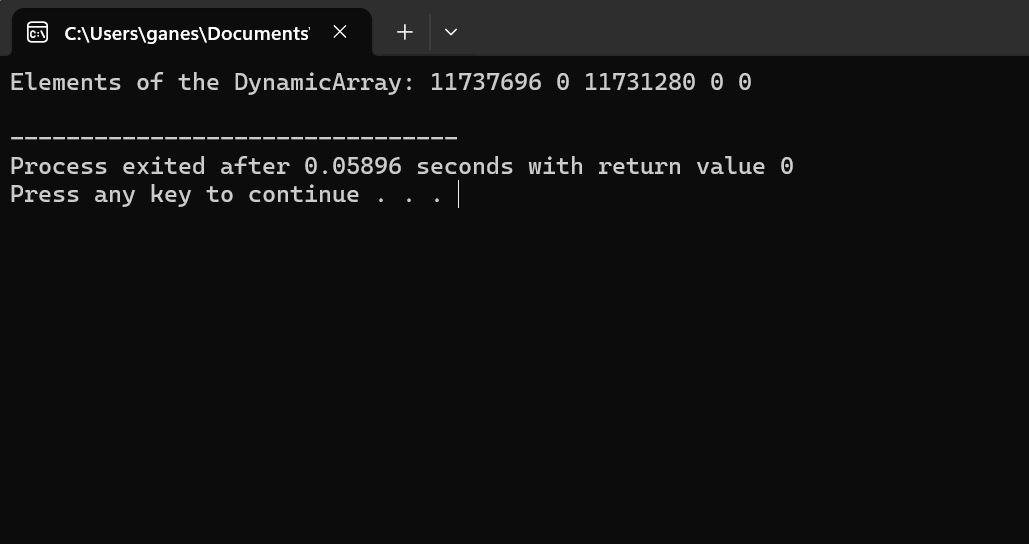
int main() {

DynamicArray dynamicArray(5);

dynamicArray.displayElements();

return 0;

}



6.

#include <iostream>

#include <string>

class Person {

private:

std::string name;

int age;

public:

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

~Person() {

std::cout << "Goodbye, " << name << "!\n";

}

// Method to get name

std::string getName() {

return name;

}

int getAge() {

return age;

}

};

int main() {

// Creating objects of the Person class

Person\* person1 = new Person("Alice", 30);

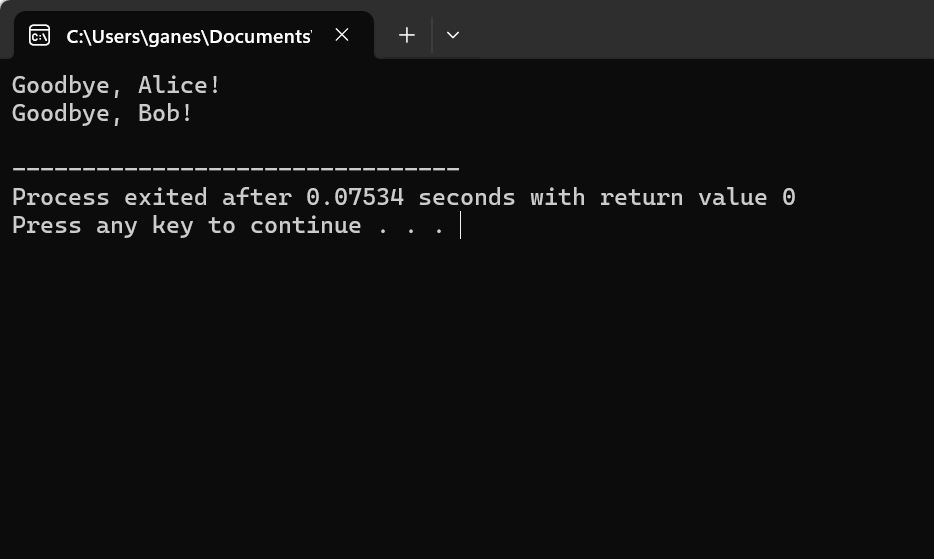
Person\* person2 = new Person("Bob", 25);

delete person1;

delete person2;

return 0;

}



7.

#include <iostream>

class Point {

private:

double x;

double y;

public:

Point(double x\_val = 0.0, double y\_val = 0.0) : x(x\_val), y(y\_val) {}

Point operator+(const Point& other) const {

return Point(x + other.x, y + other.y);

}

void display() const {

std::cout << "(" << x << ", " << y << ")";

}

};

int main() {

Point point1(1.0, 2.0);

Point point2(3.0, 4.0);

Point sum = point1 + point2;

std::cout << "Sum of ";

point1.display();

std::cout << " and ";

point2.display();

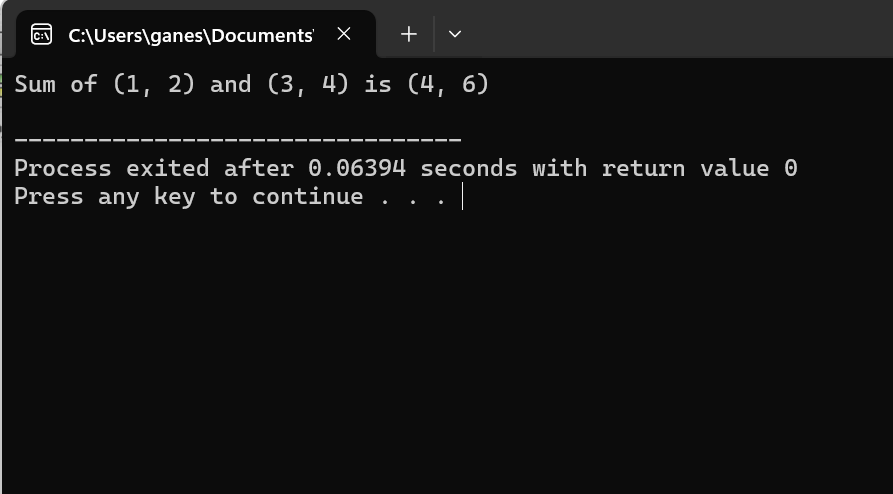
std::cout << " is ";

sum.display();

std::cout << std::endl;

return 0;

}



8.

#include <iostream>

class Rectangle {

private:

double length;

double width;

public:

Rectangle(double len = 1.0, double wid = 1.0) : length(len), width(wid) {}

double area() const {

return length \* width;

}

double operator+(const Rectangle& other) const {

return this->area() + other.area();

}

void displayDimensions() const {

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

}

};

int main() {

Rectangle rect1(3.0, 4.0);

Rectangle rect2(5.0, 6.0);

double totalArea = rect1 + rect2;

std::cout << "Rectangle 1:" << std::endl;

rect1.displayDimensions();

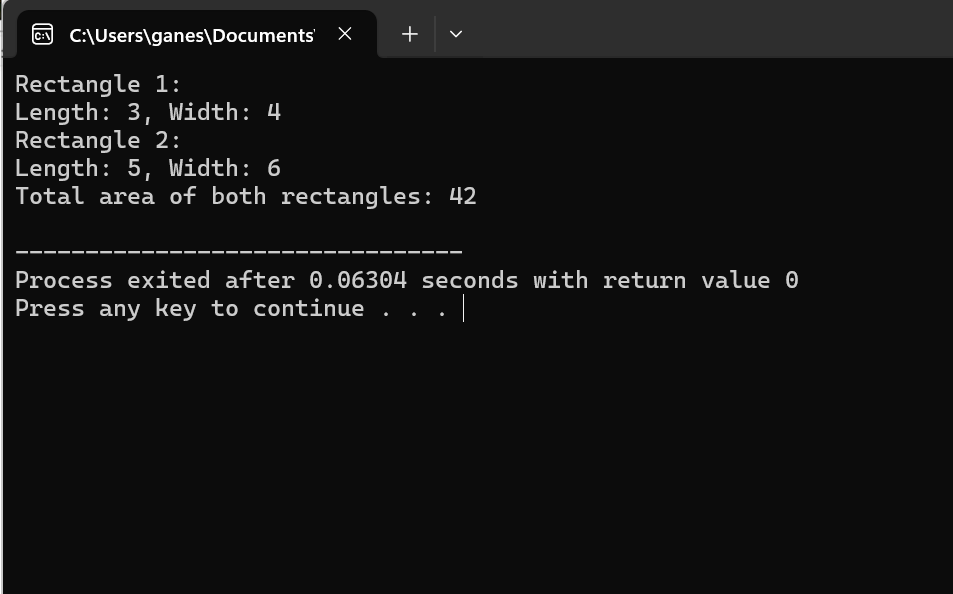
std::cout << "Rectangle 2:" << std::endl;

rect2.displayDimensions();

std::cout << "Total area of both rectangles: " << totalArea << std::endl;

return 0;

}



9.

#include <iostream>

class Complex {

private:

double real;

double imag;

public:

Complex(double r = 0.0, double i = 0.0) : real(r), imag(i) {}

Complex operator+(const Complex& other) const {

return Complex(real + other.real, imag + other.imag);

}

Complex operator-(const Complex& other) const {

return Complex(real - other.real, imag - other.imag);

}

Complex operator\*(const Complex& other) const {

double real\_part = real \* other.real - imag \* other.imag;

double imag\_part = real \* other.imag + imag \* other.real;

return Complex(real\_part, imag\_part);

}

void display() const {

std::cout << real << " + " << imag << "i";

}

};

int main() {

Complex c1(2.0, 3.0);

Complex c2(1.0, -2.0);

Complex sum = c1 + c2;

Complex difference = c1 - c2;

Complex product = c1 \* c2;

std::cout << "c1: ";

c1.display();

std::cout << std::endl;

std::cout << "c2: ";

c2.display();

std::cout << std::endl;

std::cout << "Sum: ";

sum.display();

std::cout << std::endl;

std::cout << "Difference: ";

difference.display();

std::cout << std::endl;

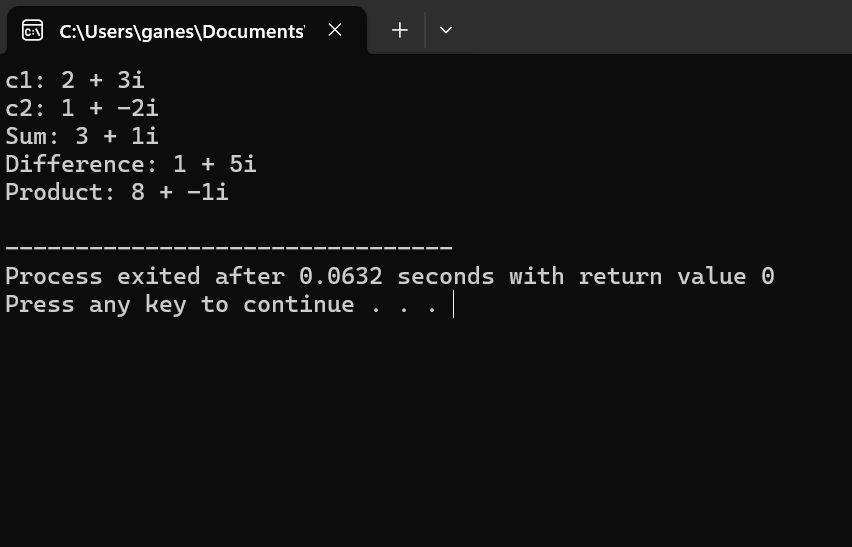
std::cout << "Product: ";

product.display();

std::cout << std::endl;

return 0;

}



10.

#include <iostream>

class Number {

private:

int value;

public:

Number(int val = 0) : value(val) {}

Number operator-() const {

return Number(-value);

}

int getValue() const {

return value;

}

};

int main() {

Number num(10);

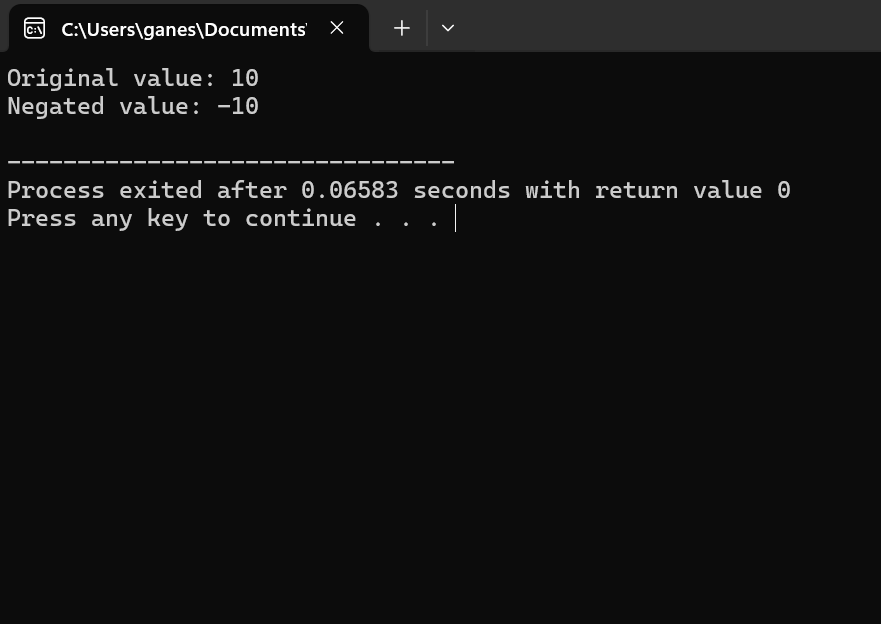
Number negatedNum = -num;

std::cout << "Original value: " << num.getValue() << std::endl;

std::cout << "Negated value: " << negatedNum.getValue() << std::endl;

return 0;

}



11.

#include <iostream>

class Rectangle {

private:

double length;

double width;

public:

Rectangle(double len, double wid) {

length = len;

width = wid;

}

double calculateArea() {

return length \* width;

}

};

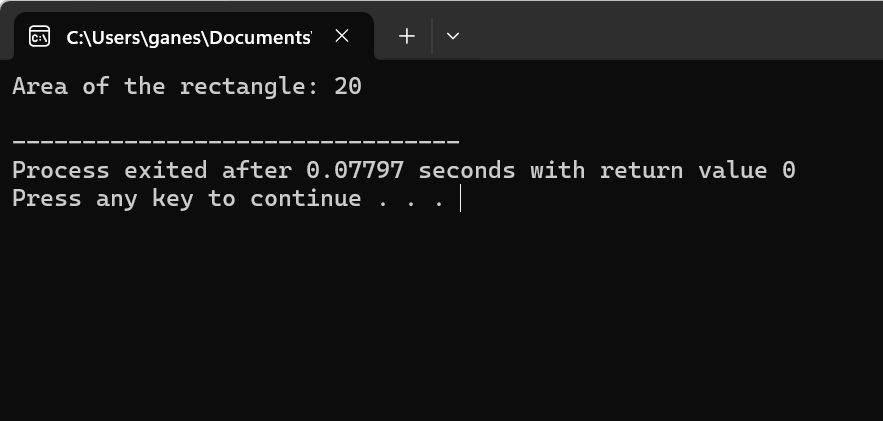
int main() {

Rectangle rect(5, 4);

std::cout << "Area of the rectangle: " << rect.calculateArea() << std::endl;

return 0;

}



12.

#include <iostream>

#include <string>

class Student {

private:

std::string name;

int age;

public:

Student() : name("Unknown"), age(0) {}

Student(std::string newName, int newAge) : name(newName), age(newAge) {}

void displayDetails() {

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

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

}

};

int main() {

Student student1;

Student student2("John", 20);

std::cout << "Student 1 Details:" << std::endl;

student1.displayDetails();

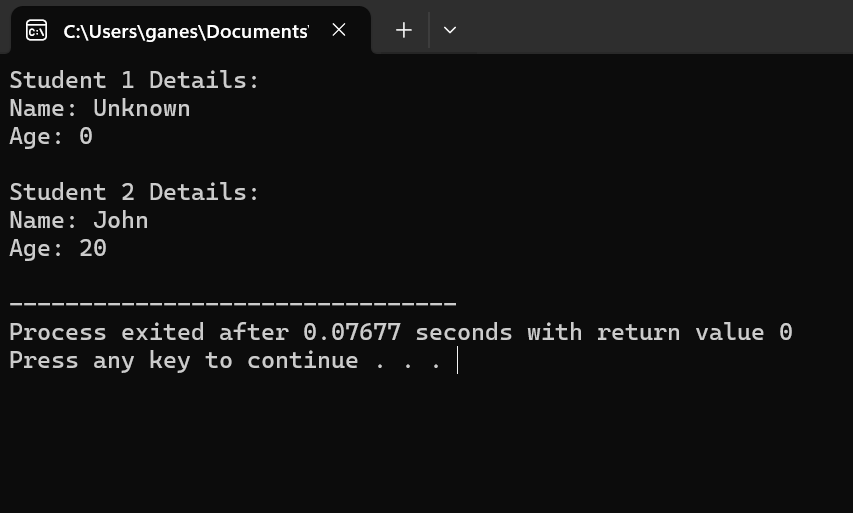
std::cout << std::endl;

std::cout << "Student 2 Details:" << std::endl;

student2.displayDetails();

return 0;

}



13.

#include <iostream>

class Complex {

private:

double real;

double imag;

public:

Complex(double r, double i) : real(r), imag(i) {}

Complex operator+(const Complex& other) {

double newReal = real + other.real;

double newImag = imag + other.imag;

return Complex(newReal, newImag);

}

void display() {

std::cout << real << " + " << imag << "i";

}

};

int main() {

Complex c1(2.5, 3.5);

Complex c2(1.2, 4.8);

Complex sum = c1 + c2;

std::cout << "Sum of ";

c1.display();

std::cout << " and ";

c2.display();

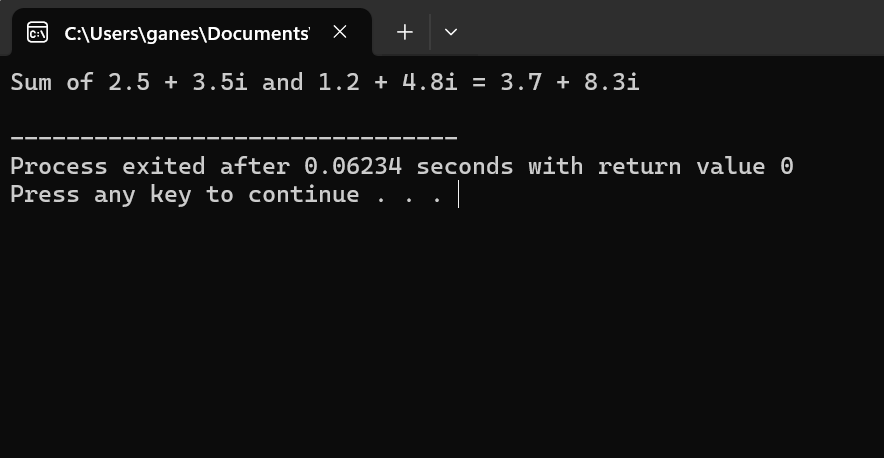
std::cout << " = ";

sum.display();

std::cout << std::endl;

return 0;

}



14.

#include <iostream>

#include <cstring>

class String {

private:

char\* str;

public:

// Constructor

String(const char\* s) {

// Allocate memory for the string and copy characters

str = new char[strlen(s) + 1];

strcpy(str, s);

}

// Copy constructor

String(const String& other) {

// Allocate memory for the string and copy characters

str = new char[strlen(other.str) + 1];

strcpy(str, other.str);

}

// Destructor

~String() {

delete[] str; // Free allocated memory

}

// Function to display the string

void display() {

std::cout << "String: " << str << std::endl;

}

};

int main() {

// Create a String object

String str1("Hello");

// Use copy constructor to create another String object

String str2 = str1;

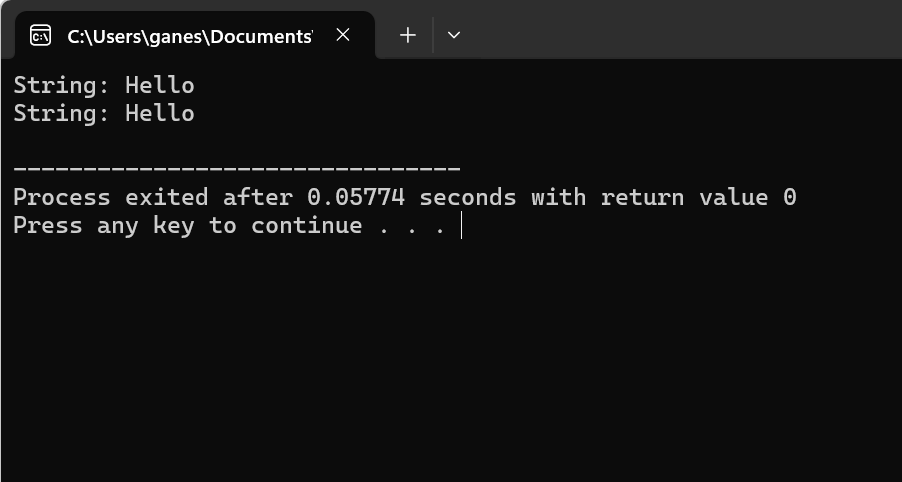
// Display both strings

str1.display();

str2.display();

return 0

}



15.

#include <iostream>

class DynamicArray {

private:

int\* array;

int size;

public:

// Parameterized constructor

DynamicArray(int initialSize) {

size = initialSize;

array = new int[size];

}

// Destructor

~DynamicArray() {

delete[] array; // Free allocated memory

}

// Function to resize the array

void resize(int newSize) {

int\* newArray = new int[newSize];

int minSize = (newSize < size) ? newSize : size;

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

newArray[i] = array[i];

}

delete[] array; // Free old memory

array = newArray;

size = newSize;

}

// Function to get the size of the array

int getSize() const {

return size;

}

// Function to display the elements of the array

void display() const {

std::cout << "Array elements: ";

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

std::cout << array[i] << " ";

}

std::cout << std::endl;

}

};

int main() {

// Create a DynamicArray object with initial size 5

DynamicArray arr(5);

// Display the initial array size and elements

std::cout << "Initial array size: " << arr.getSize() << std::endl;

arr.display();

// Resize the array to size 8

arr.resize(8);

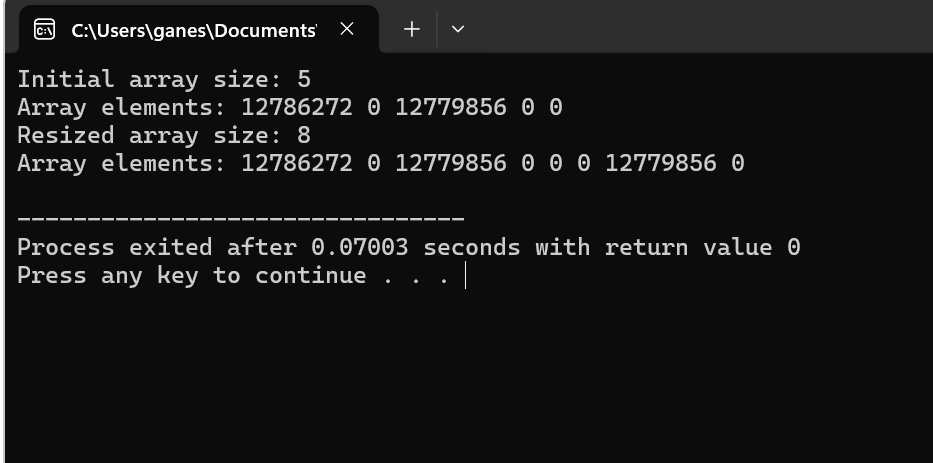
// Display the resized array size and elements

std::cout << "Resized array size: " << arr.getSize() << std::endl;

arr.display();

return 0;

}



16.

#include <iostream>

class Vector2D {

private:

double x;

double y;

public:

// Constructor

Vector2D(double x\_val, double y\_val) : x(x\_val), y(y\_val) {}

// Overloading addition operator

Vector2D operator+(const Vector2D& other) const {

return Vector2D(x + other.x, y + other.y);

}

// Function to display the vector

void display() const {

std::cout << "(" << x << ", " << y << ")";

}

};

int main() {

// Create two Vector2D objects

Vector2D v1(1.5, 2.5);

Vector2D v2(0.5, 1.5);

// Add the two vectors using the overloaded addition operator

Vector2D sum = v1 + v2;

// Display the result

std::cout << "v1 = ";

v1.display();

std::cout << std::endl;

std::cout << "v2 = ";

v2.display();

std::cout << std::endl;

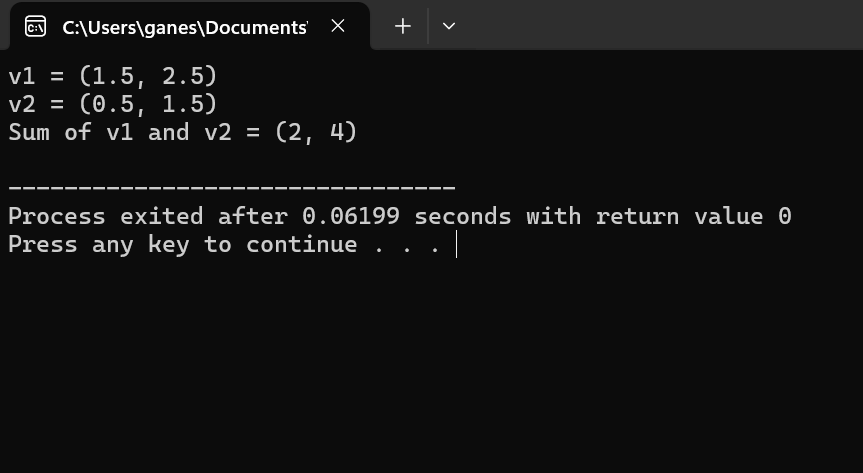
std::cout << "Sum of v1 and v2 = ";

sum.display();

std::cout << std::endl;

return 0;

}



17.

#include <iostream>

#include <string>

class Book {

private:

std::string title;

std::string author;

int pages;

public:

// Parameterized constructor

Book(const std::string& t, const std::string& a, int p)

: title(t), author(a), pages(p) {}

// Overloaded comparison operator (==)

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

return title == other.title;

}

// Overloaded comparison operator (!=)

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

return title != other.title;

}

// Function to display book details

void display() const {

std::cout << "Title: " << title << std::endl;

std::cout << "Author: " << author << std::endl;

std::cout << "Pages: " << pages << std::endl;

}

};

int main() {

// Creating two Book objects

Book book1("Title1", "Author1", 200);

Book book2("Title2", "Author2", 300);

// Comparing books using overloaded operators

if (book1 == book2) {

std::cout << "The books have the same title." << std::endl;

} else {

std::cout << "The books have different titles." << std::endl;

}

if (book1 != book2) {

std::cout << "The books have different titles." << std::endl;

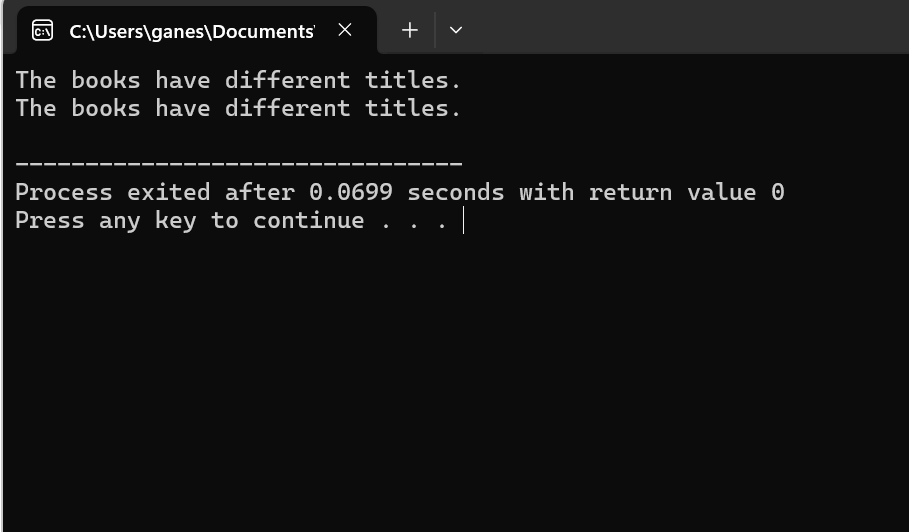
} else {

std::cout << "The books have the same title." << std::endl;

}

return 0;

}



18.

#include <iostream>

class Counter {

private:

int count;

public:

// Default constructor

Counter() : count(0) {}

// Prefix increment operator (++counter)

Counter& operator++() {

++count;

return \*this;

}

// Function to get the count value

int getCount() const {

return count;

}

};

int main() {

// Create a Counter object

Counter counter;

// Display the initial count

std::cout << "Initial count: " << counter.getCount() << std::endl;

// Increment the counter using the overloaded prefix increment operator

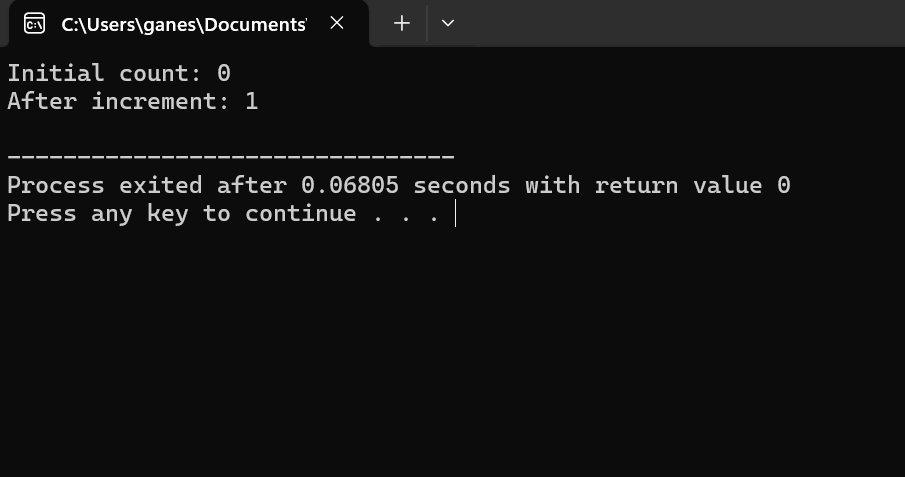
++counter;

// Display the incremented count

std::cout << "After increment: " << counter.getCount() << std::endl;

return 0;

}



19.

#include <iostream>

class Matrix {

private:

double data[2][2];

public:

// Parameterized constructor

Matrix(double a, double b, double c, double d) {

data[0][0] = a;

data[0][1] = b;

data[1][0] = c;

data[1][1] = d;

}

// Overloaded multiplication operator (\*)

Matrix operator\*(const Matrix& other) const {

Matrix result(0, 0, 0, 0);

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

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

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

result.data[i][j] += data[i][k] \* other.data[k][j];

}

}

}

return result;

}

// Function to display the matrix

void display() const {

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

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

std::cout << data[i][j] << " ";

}

std::cout << std::endl;

}

}

};

int main() {

// Create two Matrix objects

Matrix m1(1, 2, 3, 4);

Matrix m2(5, 6, 7, 8);

// Multiply the matrices using the overloaded multiplication operator

Matrix result = m1 \* m2;

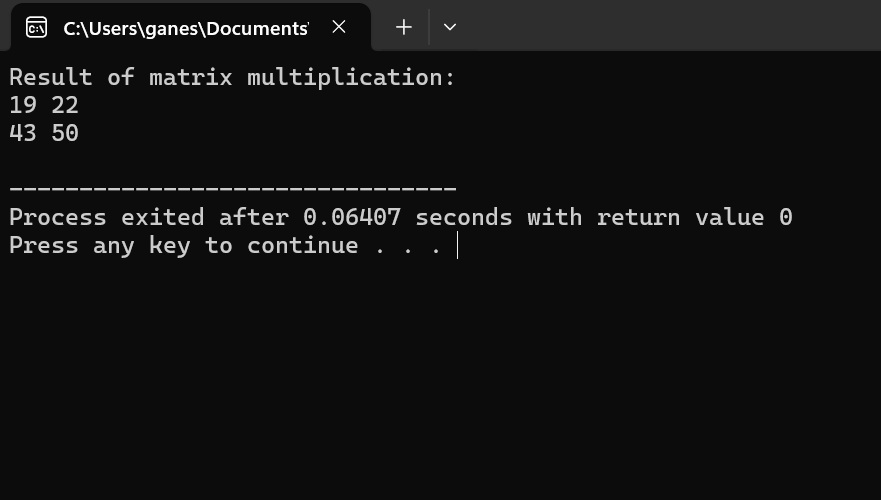
// Display the result

std::cout << "Result of matrix multiplication:" << std::endl;

result.display();

return 0;

}



20.

#include <iostream>

#include <iomanip> // For std::setw

class Date {

private:

int day;

int month;

int year;

public:

// Default constructor

Date() : day(1), month(1), year(2000) {}

// Parameterized constructor

Date(int d, int m, int y) : day(d), month(m), year(y) {}

// Overloading stream insertion operator (<<)

friend std::ostream& operator<<(std::ostream& out, const Date& date) {

out << std::setw(2) << std::setfill('0') << date.day << '/'

<< std::setw(2) << std::setfill('0') << date.month << '/'

<< date.year;

return out;

}

// Overloading stream extraction operator (>>)

friend std::istream& operator>>(std::istream& in, Date& date) {

char delimiter;

in >> date.day >> delimiter >> date.month >> delimiter >> date.year;

return in;

}

};

int main() {

// Create a Date object using default constructor

Date defaultDate;

std::cout << "Default date: " << defaultDate << std::endl;

// Read a Date from standard input

Date inputDate;

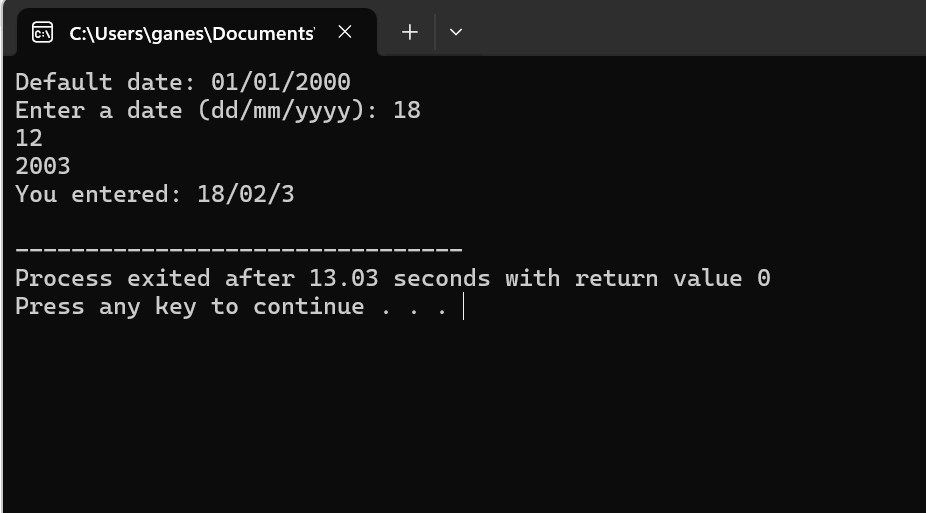
std::cout << "Enter a date (dd/mm/yyyy): ";

std::cin >> inputDate;

std::cout << "You entered: " << inputDate << std::endl;

return 0;

}



21.

#include <iostream>

#include <vector>

class Matrix {

private:

std::vector<std::vector<int>> data;

int rows;

int cols;

public:

// Parameterized constructor

Matrix(int numRows, int numCols) : rows(numRows), cols(numCols) {

data.resize(rows, std::vector<int>(cols, 0));

}

// Copy constructor

Matrix(const Matrix& other) : rows(other.rows), cols(other.cols) {

data = other.data; // Deep copy

}

// Function to set value at a specific position

void setValue(int row, int col, int value) {

if (row >= 0 && row < rows && col >= 0 && col < cols) {

data[row][col] = value;

} else {

std::cerr << "Error: Index out of range!" << std::endl;

}

}

// Function to get value at a specific position

int getValue(int row, int col) const {

if (row >= 0 && row < rows && col >= 0 && col < cols) {

return data[row][col];

} else {

std::cerr << "Error: Index out of range!" << std::endl;

return 0; // Return 0 as a default value

}

}

// Function to display the matrix

void display() const {

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

for (int j = 0; j < cols; ++j) {

std::cout << data[i][j] << " ";

}

std::cout << std::endl;

}

}

};

int main() {

// Create a Matrix object

Matrix mat1(3, 3);

// Set some values

mat1.setValue(0, 0, 1);

mat1.setValue(0, 1, 2);

mat1.setValue(0, 2, 3);

mat1.setValue(1, 0, 4);

mat1.setValue(1, 1, 5);

mat1.setValue(1, 2, 6);

mat1.setValue(2, 0, 7);

mat1.setValue(2, 1, 8);

mat1.setValue(2, 2, 9);

// Display the matrix

std::cout << "Matrix 1:" << std::endl;

mat1.display();

// Create a copy of Matrix object

Matrix mat2 = mat1;

// Display the copied matrix

std::cout << "\nCopied Matrix 2:" << std::endl;

mat2.display();

return 0;

}

22.

#include <iostream>

class Node {

public:

int data;

Node\* next;

Node(int value) : data(value), next(nullptr) {}

};

class LinkedList {

private:

Node\* head;

public:

// Default constructor

LinkedList() : head(nullptr) {}

// Parameterized constructor to initialize with an array of elements

LinkedList(int arr[], int size) : head(nullptr) {

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

insert(arr[i]);

}

}

// Destructor to free the memory allocated for nodes

~LinkedList() {

Node\* current = head;

while (current != nullptr) {

Node\* temp = current;

current = current->next;

delete temp;

}

}

// Function to insert a new element at the end of the list

void insert(int value) {

Node\* newNode = new Node(value);

if (head == nullptr) {

head = newNode;

} else {

Node\* current = head;

while (current->next != nullptr) {

current = current->next;

}

current->next = newNode;

}

}

// Function to display the linked list

void display() const {

Node\* current = head;

while (current != nullptr) {

std::cout << current->data << " ";

current = current->next;

}

std::cout << std::endl;

}

};

int main() {

// Create a LinkedList object using a parameterized constructor

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

int size = sizeof(arr) / sizeof(arr[0]);

LinkedList myList(arr, size);

// Display the linked list

myList.display();

return 0;

}

23.

#include <iostream>

#include <vector>

class Polynomial {

private:

std::vector<int> coefficients;

public:

// Parameterized constructor

Polynomial(const std::vector<int>& coeffs) : coefficients(coeffs) {}

// Overloaded addition operator (+)

Polynomial operator+(const Polynomial& other) const {

int maxDegree = std::max(coefficients.size(), other.coefficients.size());

std::vector<int> resultCoeffs(maxDegree, 0);

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

resultCoeffs[i] += coefficients[i];

}

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

resultCoeffs[i] += other.coefficients[i];

}

return Polynomial(resultCoeffs);

}

// Function to display the polynomial

void display() const {

for (int i = coefficients.size() - 1; i >= 0; --i) {

if (coefficients[i] != 0) {

std::cout << coefficients[i] << "x^" << i;

if (i != 0) {

std::cout << " + ";

}

}

}

std::cout << std::endl;

}

};

int main() {

// Create two Polynomial objects

Polynomial poly1({1, 2, 3}); // 3x^2 + 2x + 1

Polynomial poly2({-1, 0, 4}); // 4x^2 - x

// Add the polynomials using the overloaded addition operator

Polynomial sum = poly1 + poly2;

// Display the result

std::cout << "Polynomial 1: ";

poly1.display();

std::cout << "Polynomial 2: ";

poly2.display();

std::cout << "Sum: ";

sum.display();

return 0;

}

24.

#include <iostream>

#include <string>

class Employee {

private:

std::string name;

int age;

double salary;

public:

// Parameterized constructor

Employee(const std::string& n, int a, double s)

: name(n), age(a), salary(s) {}

// Overloaded comparison operator (<)

bool operator<(const Employee& other) const {

return salary < other.salary;

}

// Overloaded comparison operator (>)

bool operator>(const Employee& other) const {

return salary > other.salary;

}

// Function to display employee details

void display() const {

std::cout << "Name: " << name << ", Age: " << age << ", Salary: " << salary << std::endl;

}

};

int main() {

// Create Employee objects

Employee emp1("John", 30, 50000);

Employee emp2("Alice", 25, 60000);

// Compare employees based on salary

if (emp1 < emp2) {

std::cout << "Employee 1 earns less than Employee 2." << std::endl;

} else if (emp1 > emp2) {

std::cout << "Employee 1 earns more than Employee 2." << std::endl;

} else {

std::cout << "Employee 1 and Employee 2 have the same salary." << std::endl;

}

return 0;

}