Complex Numbers (C++) - Define a class Complex to represent complex numbers with member variables for real and imaginary parts. Overload the +, -, and \* operators for complex number addition, subtraction, and multiplication.

#include <iostream>

using namespace std;

class Complex{ // create class

private:

double real ;

double imaginary;

public :

Complex()

{

real = 0.0;

imaginary = 0.0;

}

Complex(double r, double i)

{

real = r;

imaginary = i;

}

Complex operator+(const Complex& other) const

{

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

}

Complex operator-(const Complex& other) const

{

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

}

Complex operator\*(const Complex& other) const

{

return Complex(real \* other.real, imaginary \* other.imaginary);

}

friend std::ostream& operator<<(std::ostream& os, const Complex& complex);

};

std::ostream& operator<<(std::ostream& os, const Complex& complex)

{

os << complex.real << " + " << complex.imaginary << "i";

return os;

}

int main()

{

Complex c1(3.5,3.0); // create instance

Complex c2(2.5, -1.5); //create another instance with argument

Complex sum = c1 + c2;

std::cout << "Sum: " << sum << std::endl;

Complex sub = c1 - c2;

std::cout << "Sub: " << sub << std::endl;

Complex mult = c1 \* c2;

std::cout << "Multiply: " << mult << std::endl;

    return 0;

}

2. Point2D (Java) - Create a class Point2D with x and y coordinates. Overload the + operator to return a new Point2D object representing the sum of two points.

#include <iostream>

class Point2D

{

private:

double x;

double y;

public:

Point2D(double a, double b)

{

x = a;

y= b;

}

Point2D operator+(const Point2D other) const

{

return Point2D(x + other.x, y + other.y);

}

friend std::ostream& operator<<(std::ostream& os, const Point2D& point)

{

os << "(" << point.x << ", " << point.y << ")";

return os;

}

};

int main()

{

Point2D p1(1.0, 2.0);

Point2D p2(3.0, 4.0);

Point2D sum = p1 + p2;

std::cout << "Sum of " << p1 << " and " << p2 << " is " << sum << std::endl;

    return 0;

}

3. Rectangle Class: Define a class Rectangle with member variables for width and height. Overload the + operator to return a new Rectangle object representing the sum of the areas of two rectangles.

#include <iostream>

class Rectangle

{

private:

double width;

double height;

public:

Rectangle() : width(0.0), height(0.0) {}

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

double getWidth(){

return width;

}

double getHeight(){

return height;

}

Rectangle operator+( Rectangle R) const

{

double newWidth = width + R.width;

double newHeight = height + R.height;

return Rectangle(newWidth, newHeight);

}

double area() const

{

return width \* height;

}

};

int main()

{

Rectangle rect1(3.0, 4.0);

Rectangle rect2(5.0, 2.0);

Rectangle sumRect = rect1 + rect2;

std::cout << " width = " << rect1.getWidth() << ", height = " << rect1.getHeight() << ", area = " << rect1.area() << std::endl;

std::cout << " width = " << rect2.getWidth() << ", height = " << rect2.getHeight() << ", area = " << rect2.area() << std::endl;

std::cout << "Sum Rectangle: Width = " << sumRect.getWidth() << ", Height = " << sumRect.getHeight() << ", Area = " << sumRect.area() << std::endl;

    return 0;

}

4. Date (C#) - Implement a class Date with year, month, and day. Overload the comparison operators (== and !=) to compare two Date objects.

#include<iostream>

using namespace std;

class Date{

private :

int year;

int month;

int day;

public :

Date(int y, int m, int d){

year = y;

month = m;

day = d;

}

bool operator==(Date D){

return(year == D.year && month == D.month && day == D.day);

}

bool operator!=(Date D)

{

return(year != D.year || month != D.month && day != D.day);

}

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

{

os << date.year << "-" << date.month << "-" << date.day;

return os;

}

};

int main()

{

Date d1(2024,03,04);

Date d2(2024,03,04);

Date d3(2024,03,04);

if(d1 == d2)

{

std::cout << d1 << " is equal to " << d2 << std::endl;

}

else

{

std::cout << d1 << " is not equal to "<< d2 << std::endl;

}

if(d1 != d2)

{

std::cout << d1 << "is not equal to " << d3 << std::endl;

}

else

{

std::cout << d1 << "is equal to " << d3 << std::endl;

    }

}

5. Math Functions: Design overloaded functions factorial and power that can handle integer and floating-point input for calculating factorials and raising a number to a power.

#include <iostream>

#include <cmath>

class MathFunctions

{

public:

long long factorial(int n)

{

if (n < 0)

{

std::cerr << "number must be greater than 0" << std::endl;

return 0;

}

long long result = 1;

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

{

result \*= i;

}

return result;

}

double power(double base, double exponent)

{

return std::pow(base, exponent);

}

};

int main()

{

MathFunctions math;

int n;

std::cout << "Enter number : " << std::endl;

std::cin >> n;

std::cout << "Factorial " << math.factorial(n) << std::endl;

std::cout << "2 to the power of 3: " << math.power(2, 3) << std::endl;

std::cout << "2.5 to the power of 2.0: " << math.power(2.5, 2.0) << std::endl;

    return 0;

}

6. Rectangle Class: Define a class Rectangle with member variables for width and height. Overload the + operator to return a new Rectangle object representing the sum of the areas of two rectangles.

#include <iostream>

class Rectangle {

private:

double width;

double height;

public:

Rectangle() : width(0.0), height(0.0) {}

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

double getWidth(){

return width;

}

double getHeight(){

return height;

}

Rectangle operator+( Rectangle R) const {

double newWidth = width + R.width;

double newHeight = height + R.height;

return Rectangle(newWidth, newHeight);

}

double area() const {

return width \* height;

}

};

int main() {

Rectangle rect1(3.0, 4.0);

Rectangle rect2(5.0, 2.0);

Rectangle sumRect = rect1 + rect2;

std::cout << " width = " << rect1.getWidth() << ", height = " << rect1.getHeight() << ", area = " << rect1.area() << std::endl;

std::cout << " width = " << rect2.getWidth() << ", height = " << rect2.getHeight() << ", area = " << rect2.area() << std::endl;

std::cout << "Sum Rectangle: Width = " << sumRect.getWidth() << ", Height = " << sumRect.getHeight() << ", Area = " << sumRect.area() << std::endl;

    return 0;

}