**Assignment 3 Operator Overloading**

1.Write a program to overload arithmetic operators (+, -, / , \* ) using operator function as member and friend function. [Use class Point(int x,int y)]

SOL:-

//Overloading operators as member functions

#include <iostream>

using namespace std;

class Point

{

private:

    int x;

    int y;

public:

    Point(int X, int Y) : x(X), y(Y) {}

    // Member operator functions

    Point operator+(const Point& p)

    {

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

    }

    Point operator-(const Point& p)

    {

        return Point(x - p.x, y - p.y);

    }

    Point operator\*(const Point& p)

    {

        return Point(x \* p.x, y \* p.y);

    }

    Point operator/(const Point& p)

    {

        return Point(x / p.x, y / p.y);

    }

    void print()

    {

        cout << "(" << x << ", " << y << ")" << endl;

    }

};

int main()

{

    Point p1(1, 2);

    Point p2(3, 4);

    // Member operator functions

    Point p3 = p1 + p2; // Uses operator+

    p3.print();

    Point p4 = p1 - p2; // Uses operator-

    p4.print();

    Point p5 = p1 \* p2; // Uses operator\*

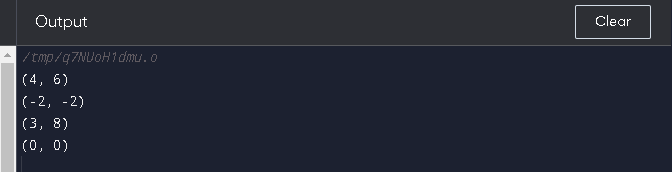
    p5.print();

    Point p6 = p1 / p2; // Uses operator/

    p6.print();

    return 0;

}



//Overloading operators as friend  functions

#include <iostream>

using namespace std;

class Point

{

private:

    int x;

    int y;

public:

    Point(int X, int Y) : x(X), y(Y) {}

    // Friend operator functions

    friend Point operator+(const Point& p1, const Point& p2)

    {

        return Point(p1.x + p2.x, p1.y + p2.y);

    }

    friend Point operator-(const Point& p1, const Point& p2)

    {

        return Point(p1.x - p2.x, p1.y - p2.y);

    }

    friend Point operator\*(const Point& p1, const Point& p2)

    {

        return Point(p1.x \* p2.x, p1.y \* p2.y);

    }

    friend Point operator/(const Point& p1, const Point& p2)

    {

        return Point(p1.x / p2.x, p1.y / p2.y);

    }

    void print()

    {

        cout << "(" << x << ", " << y << ")" << endl;

    }

};

int main()

{

    Point p1(1, 2);

    Point p2(3, 4);

    // friend operator functions

    Point p3 = p1 + p2; // Uses operator+

    p3.print();

    Point p4 = p1 - p2; // Uses operator-

    p4.print();

    Point p5 = p1 \* p2; // Uses operator\*

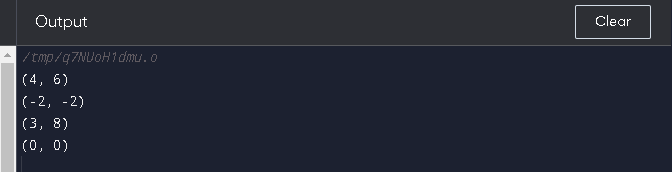
    p5.print();

    Point p6 = p1 / p2; // Uses operator/

    p6.print();

    return 0;

}



2.Write a program to overload Relational operators (>, <, >=, <= !=, == ) using operator function as member and friend function. [Use class Complex(int real, int img)]

#include <iostream>

class Complex

{

private:

    int real;

    int img;

public:

    Complex(int real = 0, int img = 0) : real(real), img(img) {}

     // Overloading the < operator as a friend function

    friend bool operator<(const Complex& c1, const Complex& c2)

    {

        // Compare the magnitudes of the two complex numbers

        return (c1.real \* c1.real + c1.img \* c1.img) < (c2.real \* c2.real + c2.img \* c2.img);

    }

    // Overloading the > operator as a friend function

    friend bool operator>(const Complex& c1, const Complex& c2)

    {

        // Compare the magnitudes of the two complex numbers

        return (c1.real \* c1.real + c1.img \* c1.img) > (c2.real \* c2.real + c2.img \* c2.img);

    }

    // Overloading the <= operator as a friend function

    friend bool operator<=(const Complex& c1, const Complex& c2)

    {

        // Compare the magnitudes of the two complex numbers

        return (c1.real \* c1.real + c1.img \* c1.img) <= (c2.real \* c2.real + c2.img \* c2.img);

    }

    // Overloading the >= operator as a friend function

    friend bool operator>=(const Complex& c1, const Complex& c2)

    {

        // Compare the magnitudes of the two complex numbers

        return (c1.real \* c1.real + c1.img \* c1.img) >= (c2.real \* c2.real + c2.img \* c2.img);

    }

// Overloading the == operator as a friend function

    friend bool operator==(const Complex& c1, const Complex& c2)

    {

        // Check if the real and imaginary parts are equal

        return (c1.real == c2.real) && (c1.img == c2.img);

    }

// Overloading the != operator as a friend function

    friend bool operator!=(const Complex& c1, const Complex& c2)

    {

        // Check if the real and imaginary parts are equal

        return (c1.real != c2.real) || (c1.img != c2.img);

    }

};

int main()

{

    Complex c1(1, 2);

    Complex c2(2, 3);

    Complex c3(1, 2);

    std::cout << (c1 > c2) << std::endl;  // Output: 0

    std::cout << (c1 < c2) << std::endl;  // Output: 1

    std::cout << (c1 >= c2) << std::endl; // Output: 0

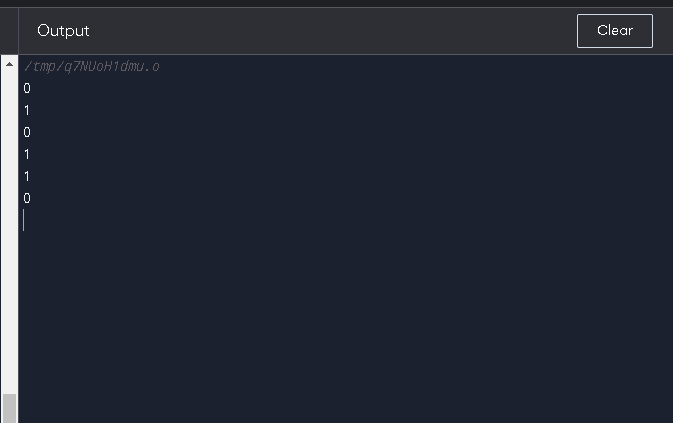
    std::cout << (c1 <= c2) << std::endl; // Output: 1

    std::cout << (c1 != c2) << std::endl; // Output: 1

    std::cout << (c1 == c2) << std::endl; // Output: 0

    return 0;

}



//Overload as member function

#include <iostream>

class Complex

{

private:

    int real;

    int img;

public:

    Complex(int real = 0, int img = 0) : real(real), img(img) {}

    // Overloading the > operator as a member function

    bool operator>(const Complex& other) const

    {

        // Compare the magnitudes of the two complex numbers

        return (real \* real + img \* img) > (other.real \* other.real + other.img \* other.img);

    }

    // Overloading the < operator as a member function

    bool operator<(const Complex& other) const

    {

        // Compare the magnitudes of the two complex numbers

        return (real \* real + img \* img) < (other.real \* other.real + other.img \* other.img);

    }

    // Overloading the >= operator as a member function

    bool operator>=(const Complex& other) const

    {

        // Compare the magnitudes of the two complex numbers

        return (real \* real + img \* img) >= (other.real \* other.real + other.img \* other.img);

    }

    // Overloading the <= operator as a member function

    bool operator<=(const Complex& other) const

    {

        // Compare the magnitudes of the two complex numbers

        return (real \* real + img \* img) <= (other.real \* other.real + other.img \* other.img);

    }

    // Overloading the != operator as a member function

    bool operator!=(const Complex& other) const

    {

        // Check if the real and imaginary parts are different

        return (real != other.real) || (img != other.img);

    }

    // Overloading the == operator as a member function

    bool operator==(const Complex& other) const

    {

        // Check if the real and imaginary parts are equal

        return (real == other.real) && (img == other.img);

    }

};

int main()

{

    Complex c1(1, 2);

    Complex c2(2, 3);

    Complex c3(1, 2);

    std::cout << (c1 > c2) << std::endl;  // Output: 0

    std::cout << (c1 < c2) << std::endl;  // Output: 1

    std::cout << (c1 >= c2) << std::endl; // Output: 0

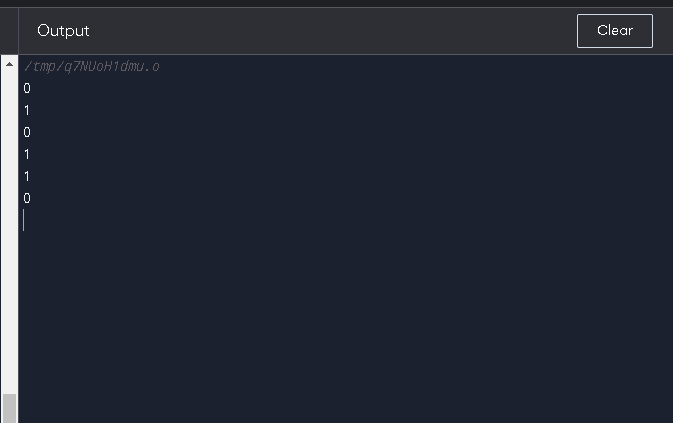
    std::cout << (c1 <= c2) << std::endl; // Output: 1

    std::cout << (c1 != c2) << std::endl; // Output: 1

    std::cout << (c1 == c2) << std::endl; // Output: 0

    return 0;

}



3.Write a program to overload Assignment operator (=). [Use class Result(int m1, int m2, int m3)]

#include <iostream>

class Result {

 public:

  // Constructor

  Result(int m1, int m2, int m3) : m1\_(m1), m2\_(m2), m3\_(m3) {}

  // Overloaded assignment operator

  Result& operator=(const Result& other) {

    m1\_ = other.m1\_;

    m2\_ = other.m2\_;

    m3\_ = other.m3\_;

    return \*this;

  }

  // Getters for the marks

  int m1() const { return m1\_; }

  int m2() const { return m2\_; }

  int m3() const { return m3\_; }

 private:

  int m1\_, m2\_, m3\_;

};

int main() {

  Result r1(50, 60, 70);

  Result r2(80, 90, 100);

  // Assign r2 to r1 using the overloaded assignment operator

  r1 = r2;

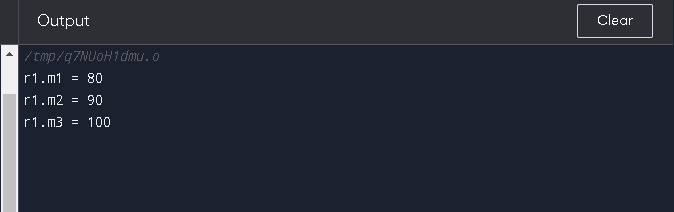
  std::cout << "r1.m1 = " << r1.m1() << std::endl;

  std::cout << "r1.m2 = " << r1.m2() << std::endl;

  std::cout << "r1.m3 = " << r1.m3() << std::endl;

  return 0;

}



4.Write a program to overload insertion and extraction operators (<<,>>). [Use class Distance(inch, feet)]

#include <iostream>

using namespace std;

class Distance {

  public:

    int inch;

    int feet;

    // Constructor

    Distance(int inch, int feet) : inch(inch), feet(feet) {}

    // Overload the stream insertion operator (<<)

    friend ostream& operator<<(ostream& out, const Distance& d) {

      out << d.feet << "' " << d.inch << "\"";

      return out;

    }

    // Overload the stream extraction operator (>>)

    friend istream& operator>>(istream& in, Distance& d) {

      in >> d.feet >> d.inch;

      return in;

    }

};

int main() {

  // Create a Distance object

  Distance d(6, 5);

  // Use the overloaded stream insertion operator to print the Distance object

  cout << d << endl;

  // Use the overloaded stream extraction operator to read values for the Distance object

  cin >> d;

  // Use the overloaded stream insertion operator to print the modified Distance object

  cout << d << endl;

  return 0;

}

