Name : Mukul Kumar

Enrollment number : 201B162

Batch : B5

Subject : OOPL

Lab Assignment number : 6

Semester : 2

1. Create two classes polar and rectangle. Polar class has two data members radius and angle and rectangle has two data members x and y. Write constructors and member functions to get input from user and display the data members in both the classes. Write a program using the concept of type conversion to convert an object of class polar to object of rectangle and vice versa. Also write a function which computes distance between two points represented either in polar or rectangular coordinates.

Hint: formulas to convert polar coordinates to rectangular coordinates:

x = r cosq, y = r sinq

formulas to convert rectangular coordinates to polar coordinates:

r = , q = tan-1( y/x )

**Code:**

#include <iostream>

#include <math.h>

using std::cin;

using std::cout;

using std::endl;

class polar;

class rectangle

{

float x, y;

public:

rectangle() {}

rectangle(float x, float y)

{

this->x = x;

this->y = y;

}

void display() { cout << "x = " << x << " y = " << y << endl; }

int returnx() { return x; }

int returny() { return y; }

void operator=(polar);

};

class polar

{

float radius, angle;

public:

polar() {}

polar(float radius, float angle)

{

this->radius = radius;

this->angle = angle;

}

void display() { cout << "radius = " << radius << " angle = " << angle << endl; }

int returnradius() { return radius; }

int returnangle() { return angle; }

void operator=(rectangle r1)

{

int x = r1.returnx();

int y = r1.returny();

this->radius = sqrt(pow(x, 2) + pow(y, 2));

this->angle = atan(y / x);

}

};

void rectangle::operator=(polar p1)

{

int radius\_ = p1.returnradius();

int angle\_ = p1.returnangle();

this->x = radius\_ \* cos(angle\_);

this->y = radius\_ \* sin(angle\_);

}

float distance(polar p1, polar p2) { return sqrt(pow(p1.returnradius(), 2) + pow(p2.returnradius(), 2) - 2 \* p1.returnradius() \* p2.returnradius() \* cos(p1.returnangle() - p2.returnangle())); }

float distance(rectangle r1, rectangle r2) { return pow(r1.returnx() - r2.returnx(), 2) + pow(r1.returny() - r2.returny(), 2); }

int main()

{

rectangle a(2, 3), b(4, 6);

polar c, d;

c = a;

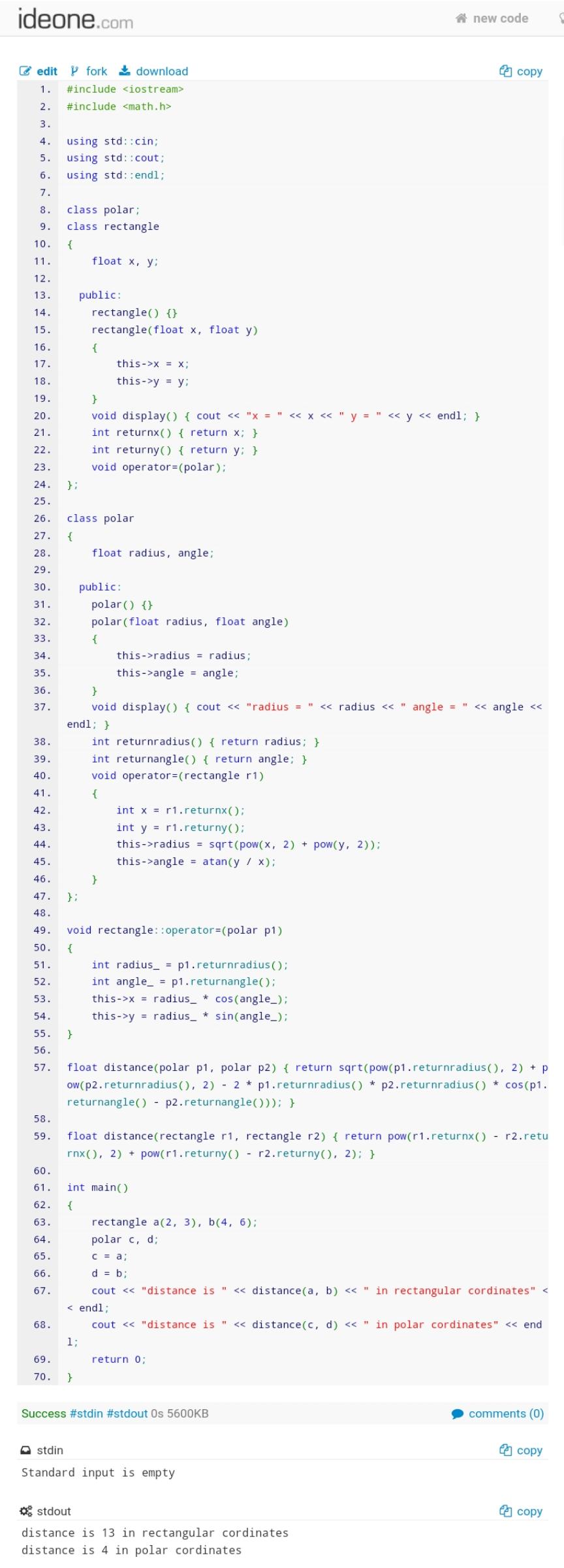
d = b;

cout << "distance is " << distance(a, b) << " in rectangular cordinates" << endl;

cout << "distance is " << distance(c, d) << " in polar cordinates" << endl;

return 0;

}



1. Design a class Distance that includes following data members: feet, inches. It has the following member function:-
   * Constructor, that initializes the distance to 0,0 by default.
   * Give a check so that the inches part is always less than 12.0.
   * Display function
   * Overloaded – operator to subtract 2 distances
   * Overloaded + operator to add 2 distances
   * Overload += and -= operator
   * Overload > and < operators to compare two distances

**Code:**

#include <iostream>

using std::cin;

using std::cout;

using std::endl;

class Distance

{

float feet, inches;

public:

Distance()

{

feet = 0;

inches = 0;

}

Distance(float feet, float inches)

{

this->feet = feet;

this->inches = (int)inches % 12;

this->feet += (inches - (int)inches % 12) / 12;

}

void Display()

{

cout << "feet = " << feet << endl;

cout << "inches = " << inches << endl;

}

friend Distance operator+(Distance d1, Distance d2)

{

Distance temp;

temp.inches = d1.inches + d2.inches;

temp.feet = d1.feet + d2.feet + ((temp.inches - (int)temp.inches % 12) / 12);

temp.inches = ((int)temp.inches % 12) + (temp.inches - (int)temp.inches);

return temp;

}

friend Distance operator-(Distance d1, Distance d2)

{

Distance temp;

temp.inches = d1.inches - d2.inches;

temp.feet = d1.feet - d2.feet + ((temp.inches - (int)temp.inches % 12) / 12);

temp.inches = ((int)temp.inches % 12) + (temp.inches - (int)temp.inches);

return temp;

}

Distance operator+=(int num)

{

this->inches += num;

this->feet += num;

this->feet += (this->inches - ((int)this->inches % 12)) / 12;

this->inches = (int)this->inches % 12;

return \*this;

}

Distance operator-=(int num)

{

this->inches -= num;

this->feet -= num;

return \*this;

}

bool operator>(Distance d)

{

if (feet + (inches / 12) > d.feet + (d.inches) / 12)

return true;

return false;

}

bool operator<(Distance d)

{

if (feet + (inches / 12) < d.feet + (d.inches) / 12)

return true;

return false;

}

};

int main()

{

Distance a(1, 26), b(1, 20), sum, c(1, 1), d(1, 2), e;

a.Display();

b.Display();

sum = a - b;

sum.Display();

c += 15;

c.Display();

d -= 12;

d.Display();

if (a > b)

cout << "hello" << endl;

if (b < a)

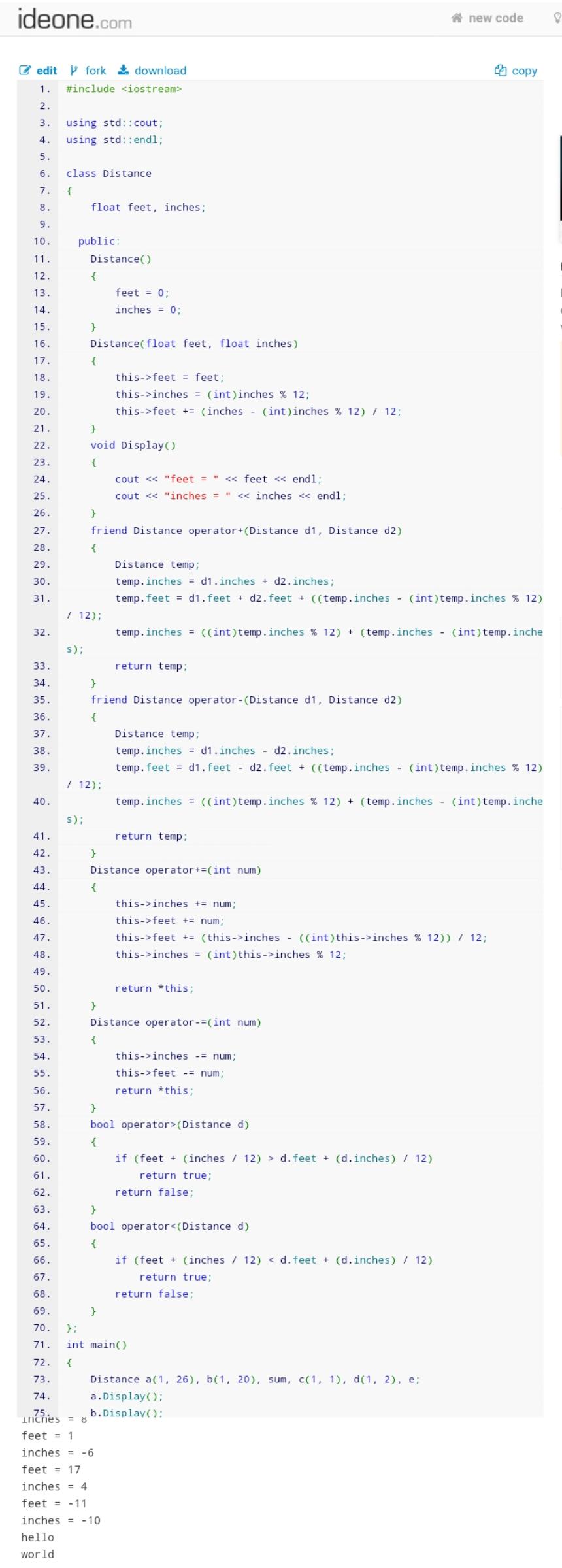
cout << "world" << endl;

if (b > a)

cout << "okay" << endl;

return 0;

}



1. Create a class rational for performing arithmetic with fractions. Use an integer variable to represent the private data of the class-the numerator and denominator. Provide a member function to get input from the user. This function should also check that denominator entered is not 0, if it is zero print invalid input. Provide a function to display the values. Overload +, -, \*, / operators to add, subtract, multiply and divide the objects of this class.

**Code:**

#include <iostream>

using std::cout;

using std::endl;

int hcf(int n1, int n2)

{

while (n1 != n2)

{

if (n1 > n2)

n1 -= n2;

else

n2 -= n1;

}

return n1;

}

class rational

{

int numerator, denominator;

public:

void userinp(int n, int d)

{

numerator = n;

if (!d)

{

cout << "Invalid Input" << endl;

return;

}

denominator = d;

}

void display()

{

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

}

rational operator+(rational n2)

{

rational temp;

temp.numerator = (numerator \* n2.denominator + denominator \* n2.numerator);

temp.denominator = (denominator \* n2.denominator);

int temphcf = hcf(temp.numerator, temp.denominator);

temp.numerator /= temphcf;

temp.denominator /= temphcf;

return temp;

}

rational operator-(rational n2)

{

rational temp;

int temphcf;

temp.numerator = (numerator \* n2.denominator - denominator \* n2.numerator);

temp.denominator = (denominator \* n2.denominator);

if (temp.numerator < 0)

{

temp.numerator \*= -1;

temphcf = hcf(temp.numerator, temp.denominator);

temp.numerator \*= -1;

temp.numerator /= temphcf;

}

else

temphcf = hcf(temp.numerator, temp.denominator);

temp.denominator /= temphcf;

return temp;

}

rational operator\*(rational n2)

{

rational temp;

temp.numerator = (numerator \* n2.numerator);

temp.denominator = (denominator \* n2.denominator);

int temphcf = hcf(temp.numerator, temp.denominator);

temp.numerator /= temphcf;

temp.denominator /= temphcf;

return temp;

}

rational operator/(rational n2)

{

rational temp;

temp.numerator = (numerator \* n2.denominator);

temp.denominator = (denominator \* n2.numerator);

int temphcf = hcf(temp.numerator, temp.denominator);

temp.numerator /= temphcf;

temp.denominator /= temphcf;

return temp;

}

};

int main()

{

rational a, b, c;

a.userinp(1, 2);

a.display();

b.userinp(9, 3);

b.display();

c = a - b;

c.display();

c = a + b;

c.display();

c = a \* b;

c.display();

c = a / b;

c.display();

return 0;

}



1. Include a function that adds two strings to make a third string. Write a program to do the following tasks:
   * Create uninitialized string objects
   * Creates the objects with string constants.
   * Concatenates two strings properly
   * Displays a desired string object

**Code:**

#include <iostream>

#include <cstring>

using std::cout;

using std::endl;

class String

{

private:

char \*ptr= NULL;

int len=0;

public:

String(){}

String(const char \*s)

{

this->len = strlen(s);

this->ptr = new char[len + 1];

strcpy(ptr, s);

}

void display()

{

cout << ptr << endl;

}

friend String operator+(const String a, const String b)

{

String temp;

temp.len = a.len + b.len;

temp.ptr = new char[temp.len + 1];

strcpy(temp.ptr, a.ptr);

strcat(temp.ptr, b.ptr);

return temp;

}

~String() {}

};

int main()

{

String S1("Hello "), S2("World"), S3;

S3 = S1 + S2;

S3.display();

return 0;

}

