NAME: SATVIK DANDALE

BATCH: B1

DIV: TY-E

GR No: 1710797

Roll No: 13

/\*

Write a program to extend complex class with following details:

Public:

Operator functions for +, - , multiply

Print complex numbers.

(Operator overloading using member and non-member functions)

\*/

#include<bits/stdc++.h>

using namespace std;

class Complex{

private:

float real\_value, img\_value;

public:

// Constructors

Complex(/\* DEFAULT \*/){

this->real\_value = this->img\_value = 0;

}

Complex(double single){

// In this case, the real and imaginary part will be set to the given number.

this->real\_value = this->img\_value = single;

}

Complex(int single){

// In this case, the real and imaginary part will be set to the given number.

this->real\_value = this->img\_value = single;

}

Complex(double real\_value, double img\_value){

this->real\_value = real\_value;

this->img\_value = img\_value;

}

Complex(int real\_value, double img\_value){

this->real\_value = real\_value;

this->img\_value = img\_value;

}

Complex(double real\_value, int img\_value){

this->real\_value = real\_value;

this->img\_value = img\_value;

}

Complex(int real\_value, int img\_value){

this->real\_value = real\_value;

this->img\_value = img\_value;

}

// Copy Constructor

Complex(Complex &c){

this->real\_value = c.real\_value;

this->img\_value = c.img\_value;

}

// Destructor

~Complex(){

delete this;

}

// SETTERS

void setReal(int real\_value){

this->real\_value = real\_value;

}

void setImg(int img\_value){

this->img\_value = img\_value;

}

void setReal(double real\_value){

this->real\_value = real\_value;

}

void setImg(double img\_value){

this->img\_value = img\_value;

}

// GETTERS

float getReal(){ return this->real\_value; }

float getImg(){ return this->img\_value; }

// Displaying the complex number

void display(){

cout<<this->real\_value<<" + "<<this->img\_value<<"i";

}

};

class ExtendedComplex: public Complex{

public:

// Static Members

static int count;

static void incrementCount(){

count++;

}

// Constructors

ExtendedComplex(): Complex(){incrementCount();}

ExtendedComplex(double single): Complex(single){incrementCount();}

ExtendedComplex(int single): Complex(single){incrementCount();}

ExtendedComplex(double real\_value, double img\_value):Complex(real\_value, img\_value){incrementCount();}

ExtendedComplex(int real\_value, double img\_value):Complex(real\_value, img\_value){incrementCount();}

ExtendedComplex(double real\_value, int img\_value): Complex(real\_value, img\_value){incrementCount();}

ExtendedComplex(int real\_value, int img\_value): Complex(real\_value, img\_value){incrementCount();}

// Copy Constructors

ExtendedComplex(ExtendedComplex &c1): Complex(c1.getReal(), c1.getImg()){incrementCount();}

ExtendedComplex(Complex &c1): Complex(c1){incrementCount();}

// Destructors

~ExtendedComplex(){

delete this;

}

// addComplex()

ExtendedComplex\* add\_complex(ExtendedComplex &c2){

// Two complex numbers, \*this and c2

// Create a new object

static ExtendedComplex \*temp = new ExtendedComplex();

temp->setReal(this->getReal() + c2.getReal());

temp->setImg(this->getImg() + c2.getImg());

// Returns the address of the new Object created

return temp;

}

// subComplex()

ExtendedComplex\* sub\_complex(ExtendedComplex &c2){

// Two complex numbers, \*this and c2

// Create a new object

static ExtendedComplex \*temp = new ExtendedComplex();

temp->setReal(this->getReal() - c2.getReal());

temp->setImg(this->getImg() - c2.getImg());

// Returns the address of the new Object created

return temp;

}

// multiply\_complex()

ExtendedComplex\* multiply\_complex(ExtendedComplex &c2){

// this.r + (this.img)i REAL: this.r IMG: this.img

// c2.r + (c2.img)i REAL: c2.r IMG: c2.img

// RESULT:

// REAL: this.r \* c2.r - this.img \* c2.img

// IMG: this.r \* c2.img + this.img \* c2.r

ExtendedComplex \*temp = new ExtendedComplex();

double real = (this->getReal() \* c2.getReal()) - (this->getImg() \* c2.getImg());

double img = (this->getReal() \* c2.getImg()) + (this->getImg() \* c2.getReal());

temp->setReal(real);

temp->setImg(img);

return temp;

}

// print\_Complex()

void print\_Complex(){

this->display();

}

// get\_count()

static int get\_count(){

return count;

}

};

int ExtendedComplex::count = 0;

class overloadedComplex: public ExtendedComplex{

public:

// Constructors

overloadedComplex():ExtendedComplex(){}

overloadedComplex(double single):ExtendedComplex(single){}

overloadedComplex(int single):ExtendedComplex(single){}

overloadedComplex(double real\_value, double img\_value):ExtendedComplex(real\_value, img\_value){}

overloadedComplex(int real\_value, double img\_value):ExtendedComplex(real\_value, img\_value){}

overloadedComplex(double real\_value, int img\_value):ExtendedComplex(real\_value, img\_value){}

overloadedComplex(int real\_value, int img\_value):ExtendedComplex(real\_value, img\_value){}

// Copy Constructors

overloadedComplex(overloadedComplex &c):ExtendedComplex(c.getReal(), c.getImg()){

// This constructor will take overloadedComplex object and will pass the two values to the parameterized constructor of ExtendedComplex class.

}

overloadedComplex(ExtendedComplex &c):ExtendedComplex(c){}

overloadedComplex(Complex &c):ExtendedComplex(c){}

// operator+

overloadedComplex& operator+(overloadedComplex &c) {

// There are two numbers

// this and c

overloadedComplex \*temp = new overloadedComplex();

temp->setReal(this->getReal() + c.getReal());

temp->setImg(this->getImg() + c.getImg());

return \*temp;

}

// operator-

overloadedComplex& operator-(overloadedComplex &c) {

// There are two numbers

// this and c

overloadedComplex \*temp = new overloadedComplex();

temp->setReal(this->getReal() - c.getReal());

temp->setImg(this->getImg() - c.getImg());

return \*temp;

}

// operator\*

overloadedComplex& operator\*(overloadedComplex &c) {

/\*

this.r + (this.img)i REAL: this.r IMG: this.img

c2.r + (c2.img)i REAL: c2.r IMG: c2.img

RESULT:

REAL: this.r \* c2.r - this.img \* c2.img

IMG: this.r \* c2.img + this.img \* c2.r

\*/

overloadedComplex \*temp = new overloadedComplex();

double real = (this->getReal() \* c.getReal()) - (this->getImg() \* c.getImg());

double img = (this->getReal() \* c.getImg()) + (this->getImg() \* c.getReal());

temp->setReal(real);

temp->setImg(img);

return \*temp;

}

};

int main(){

overloadedComplex \*c1 = new overloadedComplex(1, 4);

overloadedComplex \*c2 = new overloadedComplex(2, 8);

overloadedComplex \*c3 = new overloadedComplex(\*c1 + \*c2);

overloadedComplex \*c4 = new overloadedComplex(\*c1 - \*c2);

overloadedComplex \*c5 = new overloadedComplex(\*c1 \* \*c2);

cout<<"\"c1\" is: ";

c1->display();

cout<<endl<<"\"c2\" is: ";

c2->display();

cout<<endl<<"\"c3\": c1 + c2 is (c1.operator+(c2)) : ";

c3->display();

cout<<endl<<"\"c4\": c1 - c2 is (c1.operator-(c2)) : ";

c4->display();

cout<<endl<<"\"c5\": c1 \* c2 is (c1.operator\*(c2)) : ";

c5->display();

cout<<endl;

}

OUTPUT:

"c1" is: 1 + 4i

"c2" is: 2 + 8i

"c3": c1 + c2 is (c1.operator+(c2)) : 3 + 12i

"c4": c1 - c2 is (c1.operator-(c2)) : -1 + -4i

"c5": c1 \* c2 is (c1.operator\*(c2)) : -30 + 16i