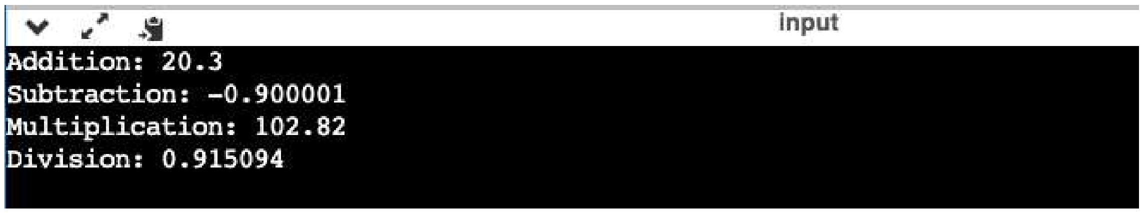
1. Create a class FLOAT that contains one float data member .Overload all the four arithmetic operators so that they operate on the objects of FLOAT.
Answer:

```
#include<iostream>
using namespace std;
class FLOAT {
private:
   float value;

public:
   // Constructor
FLOAT(float val){
   value = val;
}
   // Overloading addition operator (+)
   FLOAT operator+(const FLOAT& add) {
      return FLOAT(value + add.value);
   }

   // Overloading subtraction operator (-)
   FLOAT operator-(const FLOAT& sub)  {
      return FLOAT(value - sub.value);
   }

   // Overloading multiplication operator (*)
   FLOAT operator*(const FLOAT& mul)  {
      return FLOAT(value * mul.value);
   }

   // Overloading division operator (/)
   FLOAT operator/(const FLOAT& div)  {
      if (div.value != 0) {
         return FLOAT(value / div.value);
      } else {
         cout<<"error";

      }
   }
```

```
    // Getter function to retrieve the value
    float getValue()  {
        return value;
    }
};

int main() {
    FLOAT a(9.7);
    FLOAT b(10.6);

    // Addition
    FLOAT result_add = a + b;
    cout << "Addition: " << result_add.getValue() <<endl;

    // Subtraction
    FLOAT result_sub = a - b;
    cout << "Subtraction: " << result_sub.getValue() <<endl;

    // Multiplication
    FLOAT result_mul = a * b;
    cout << "Multiplication: " << result_mul.getValue() <<endl;

    // Division
    FLOAT result_div = a / b;
    cout << "Division: " << result_div.getValue() <<endl;



    return 0;
}
```

OUTPUT :



```
Addition: 20.3
Subtraction: -0.900001
Multiplication: 102.82
Division: 0.915094
```

QUESTION : COMPLEX INSERTION(COUT) OPERATOR .

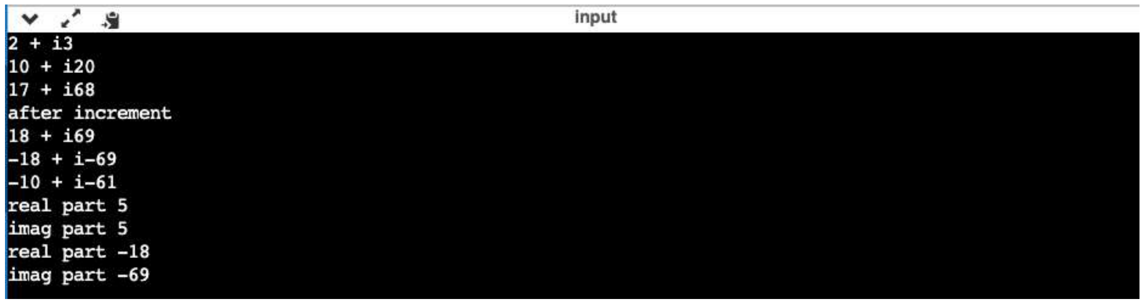
ANSWER :

```
#include <iostream>

using namespace std;
class Complex1
{
          int real;     //data members
          int imag;   //instance variable

        public:
        Complex1(){
          real=0;
          imag=0;
        }
        Complex1(int real,int imag)
        {
            this->real=real;
            this->imag=imag;
        }
        void display()
        {
             cout<<real<<" + i"<<this->imag<<endl;
        }
        Complex1 operator +(const Complex1&);
        Complex1 operator +(int);
        Complex1 operator *(const Complex1&);
         void operator++();
         void operator++(int);
         Complex1 operator-();
       friend Complex1 operator +(int i,const Complex1&);

       //friend function
       friend ostream& operator <<(ostream& ,const Complex1&);
};

Complex1 Complex1::operator +( const Complex1 &c)
 {
     Complex1 t;
     t.real=real+c.real;
     t.imag=imag+c.imag;
     return t;
 }
 Complex1 Complex1::operator +( int i)
 {
     Complex1 t;
     t.real=real+i;
```

```
    t.imag=imag+i;
    return t;
}
Complex1 Complex1::operator *(const Complex1 &c)
{
    Complex1 t;
    t.real=real*c.real;
    t.imag=imag*c.imag;
    return t;
}

 Complex1 Complex1::operator-()
 {
    real=-real;
     imag=-imag;
     return *this;
 }
 void Complex1::operator++()  //preincrement
 {
    ++real;
    ++imag;
 }
  void Complex1::operator++(int)  //postincrement
 {
    ++real;
    ++imag;
 }
 Complex1 operator +(int i,const Complex1& c)
 {
     Complex1 t;
    t.real=i+c.real;
    t.imag=i+c.imag;
    return t;
 }

 //insertion operator
 ostream& operator <<(ostream& o ,const Complex1& c)
 {
     o<<"real part "<<c.real<<endl;
     o<<"imag part "<<c.imag<<endl;
 }
int main()
{

 Complex1 c1(2,3);
Complex1 c2(10,20);
Complex1 c3(1,3);
Complex1 c4;
```

```
 c4=c4+5;
 c1.display();
 c2.display();

  Complex1 c5=c1+c2*c3+c4;   // c3.operator+(c4)
 c5.display();
  c5++;
  cout<<"after increment"<<endl;
  c5.display();
 Complex1 c6=-c5;
  c6.display();
  c6=8+c6;
  c6.display();
  cout<<c4<<c5;;
 return 0;
}
```

OUTPUT :

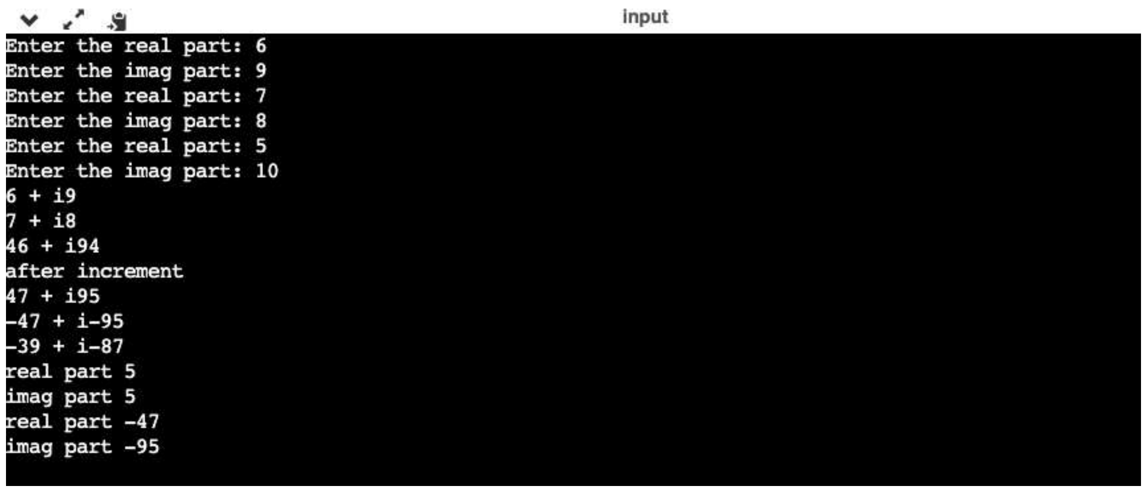


```
2 + i3
10 + i20
17 + i68
after increment
18 + i69
-18 + i-69
-10 + i-61
real part 5
imag part 5
real part -18
imag part -69
```

QUESTION : COMPLEX EXTRACTION(CIN) OPERATOR .
ANSWER:

```
#include <iostream>

using namespace std;

class Complex1 {
    int real;
    int imag;

public:
    Complex1() {
        real = 0;
        imag = 0;
    }

    Complex1(int real, int imag) {
        this->real = real;
        this->imag = imag;
    }

    void display() {
        cout << real << " + i" << imag << endl;
    }

    Complex1 operator +(const Complex1 &);
    Complex1 operator +(int);
    Complex1 operator *(const Complex1 &);
    void operator++();
    void operator++(int);
    Complex1 operator-();
    friend Complex1 operator +(int i, const Complex1 &);

    friend ostream &operator <<(ostream &, const Complex1 &);

    friend istream &operator >>(istream &, Complex1 &);
};

Complex1 Complex1::operator +(const Complex1 &c) {
    Complex1 t;
    t.real = real + c.real;
    t.imag = imag + c.imag;
    return t;
}

Complex1 Complex1::operator +(int i) {
    Complex1 t;
    t.real = real + i;
    t.imag = imag + i;
    return t;
}

Complex1 Complex1::operator *(const Complex1 &c) {
    Complex1 t;
    t.real = real * c.real;
    t.imag = imag * c.imag;
    return t;
}

Complex1 Complex1::operator-() {
    real = -real;
    imag = -imag;
    return *this;
}

void Complex1::operator++() {
    ++real;
    ++imag;
}

void Complex1::operator++(int) {
    ++real;
    ++imag;
}

Complex1 operator +(int i, const Complex1 &c) {
    Complex1 t;
    t.real = i + c.real;
    t.imag = i + c.imag;
    return t;
}

ostream &operator <<(ostream &o, const Complex1 &c) {
    o << "real part " << c.real << endl;
    o << "imag part " << c.imag << endl;
    return o;
}

istream &operator >>(istream &i, Complex1 &c) {
    cout << "Enter the real part: ";
    i >> c.real;
    cout << "Enter the imag part: ";
    i >> c.imag;
    return i;
}

int main() {
    Complex1 c1, c2, c3, c4;
    cin >> c1; // take input for c1
    cin >> c2; // take input for c2
    cin >> c3; // take input for c3

    c4 = c4 + 5;
    c1.display();
    c2.display();

    Complex1 c5 = c1 + c2 * c3 + c4;
    c5.display();
    c5++;
    cout << "after increment" << endl;
    c5.display();
    Complex1 c6 = -c5;
    c6.display();
    c6 = 8 + c6;
    c6.display();
    cout << c4 << c5;

    return 0;
}
```

OUTPUT :



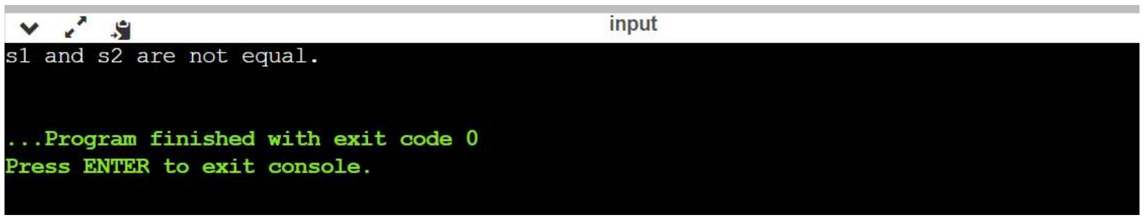
```
Enter the real part: 6
Enter the imag part: 9
Enter the real part: 7
Enter the imag part: 8
Enter the real part: 5
Enter the imag part: 10
6 + i9
7 + i8
46 + i94
after increment
47 + i95
-47 + i-95
-39 + i-87
real part 5
imag part 5
real part -47
imag part -95
```

2. Define a class string. Overlaod ==operator to compare 2 strings.

Answer:

```
#include <iostream>
#include <cstring>
using namespace std;
class String {
private:
    char* str;

public:
    // Constructors
    String() {
        str =0;
    }
    String(const char* s) {
        if (s) {
            str = new char[strlen(s) + 1];
            strcpy(str, s);
        } else {
            str = 0;
        }
    }

    // Destructor
    ~String() {
        delete[] str;
    }

    // Overload equality operator '=='
    bool operator==(const String& other) const {
        return (strcmp(str, other.str) == 0);
    }

    const char* getData() const {
        return str;
    }
};

int main() {
    String s1("Hello");
    String s2("World");


    if (s1 == s2) {
        cout << "s1 and s2 are equal." << endl;
    } else {
        cout << "s1 and s2 are not equal." << endl;
    }


    return 0;
}
```

OUTPUT :



input

```
s1 and s2 are not equal.


...Program finished with exit code 0
Press ENTER to exit console.
```

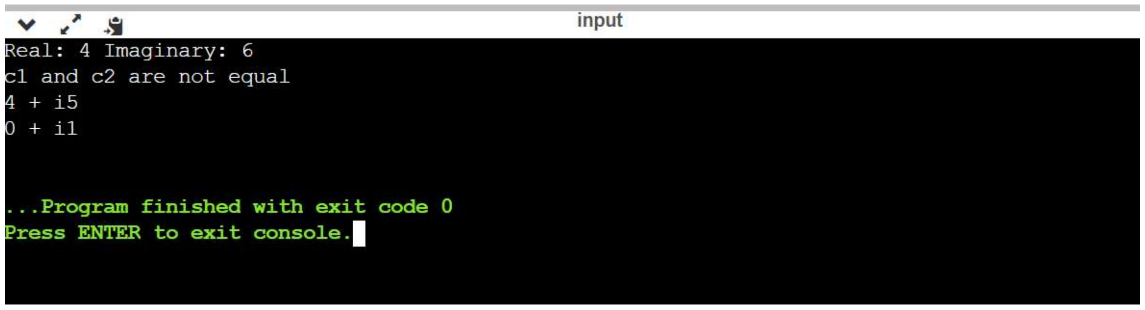
3. Create a Complex class that has real(int) and img(int) as member data, and has getData and showData functions. Then also overload the following operators for Complex class. =, ==, +, ++, --,

Anwer:

```
#include <iostream>
```

```
using namespace std;
class Complex {
private:
    int real;
    int img;

public:
    // Constructors
    Complex(){
        real = 0;
        img = 0;
    }
    Complex(int r, int i){
        real = r;
        img = i;
    }

        void getData() const {
        cout << "Real: " << real << " Imaginary: " << img <<endl;
    }

    // Display function
    void showData() const {
        cout<<real<<" + i"<<this->img<<endl;
    }

    // Overload assignment operator '='
    Complex& operator=(const Complex& other) {
        if (this != &other) {
            real = other.real;
            img = other.img;
        }
        return *this;
    }

    // Overload equality operator '=='
    bool operator==(const Complex& other)  {
        return (real == other.real) && (img == other.img);
    }

    // Overload addition operator '+'
    Complex operator+(const Complex& other) {
        return Complex(real + other.real, img + other.img);
    }

    // Overload pre-increment operator '++'
    Complex& operator++() {
        ++real;
        ++img;
        return *this;
    }

    // Overload pre-decrement operator '--'
    Complex& operator--() {
        --real;
        --img;
        return *this;
    }
};

int main() {
    Complex c1(3, 4);
    Complex c2(1, 2);


    Complex result;

    result = c1 + c2;
    result.getData();

    if (c1 == c2) {
        cout << "c1 and c2 are equal" <<endl;
    } else {
        cout << "c1 and c2 are not equal" <<endl;
    }

    ++c1;
    c1.showData();
    --c2;
    c2.showData();
    return 0;
}
```

OUTPUT :



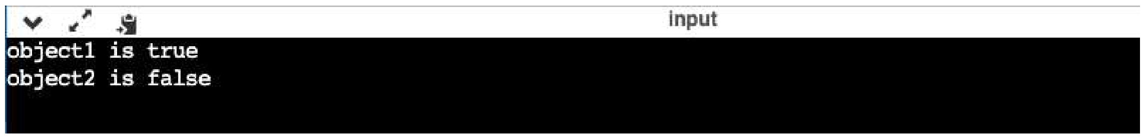
```
Real: 4 Imaginary: 6
c1 and c2 are not equal
4 + i5
0 + i1


...Program finished with exit code 0
Press ENTER to exit console.
```

4. Write a C++ program to overload ‘!’ operator using friend function.
Answer:

```
#include <iostream>
using namespace std;
class Notop {
private:
   bool value;

public:
   Notop(bool val){
   value = val;
}
   //Declare the friend function for overloading
   friend bool operator!(const Notop& obj);
};

 //Define the friend function
bool operator!(const Notop& obj) {

   return !obj.value;
}

int main() {
   Notop obj1(true);
   Notop obj2(false);


   if (!obj1) {
      cout << "object1 is false" << endl;
   } else {
      cout << "object1 is true" << endl;
   }

   if (!obj2) {
      cout << "object2 is false" << endl;
   } else {
      cout << "object2 is true" << endl;
   }
```

```
    return 0;
}
```

OUTPUT :



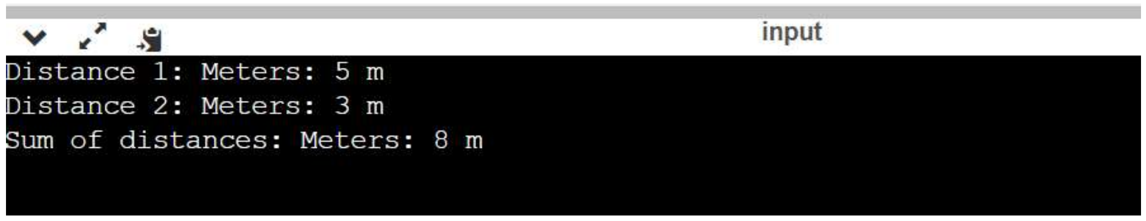
```
object1 is true
object2 is false
```

5. Read a value of distance from one object and add with a value in another object using friend function.

Answer:

```
#include <iostream>
using namespace std;
class Distance {
private:
    int meters;

public:
    Distance(int m) : meters(m) {}

    // Declare the friend function
    friend Distance addDistances(const Distance& dist1, const Distance&
dist2);

    void display() const {
        cout << "Meters: " << meters << " m" <<endl;
    }
};

// Define the friend function to add distances
Distance addDistances(const Distance& dist1, const Distance& dist2) {
    int totalMeters = dist1.meters + dist2.meters;
    return Distance(totalMeters);
}

int main() {
    Distance distance1(5);
    Distance distance2(3);

    // Use the friend function to add distances
    Distance result = addDistances(distance1, distance2);

    cout << "Distance 1: ";
    distance1.display();

    cout << "Distance 2: ";
    distance2.display();

    cout << "Sum of distances: ";
    result.display();

    return 0;
}
```

OUTPUT :



```
Distance 1: Meters: 5 m
Distance 2: Meters: 3 m
Sum of distances: Meters: 8 m
```