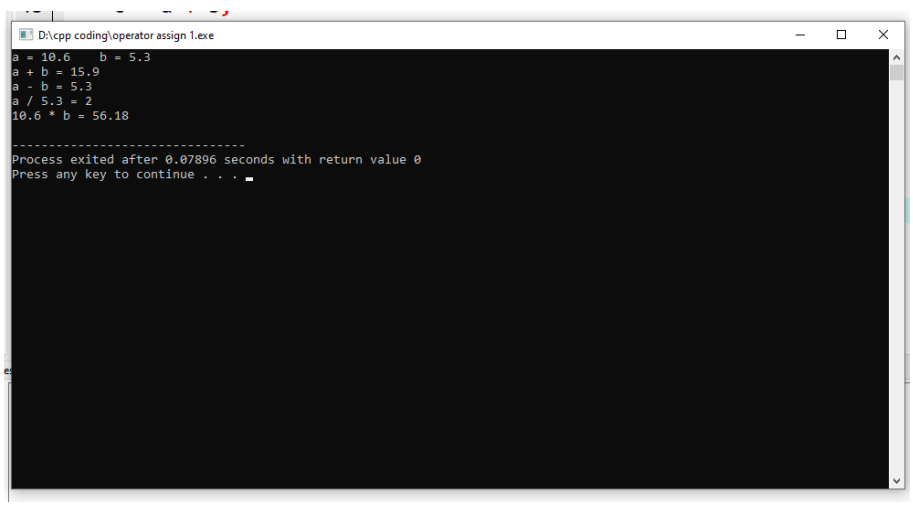
OPERATOR OVERLOADING

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.

→

```
#include<iostream>
using namespace std;
class Float
{
   float i;
   public:
      Float(): i(5) {}
      Float(float x): i(x) {}
      Float operator + (Float a)
      {
         Float temp;
         temp.i = i + a.i;
         return temp;
      }

      Float operator - (Float a)
      {
         Float temp;
         temp.i = i - a.i;
         return temp;
      }

      Float operator / (float a)
      {
         Float temp;
         temp.i = i / a;
         return temp;
      }

      friend Float operator* (float a,Float b)
      {
         Float temp;
         temp.i = a * b.i;
         return temp;
      }

      void show()
```

```
        {
            cout<<i<<endl;
        }
};


int main()
{
    Float a = 10.6,b = 5.3,c;
    cout<<"a = 10.6    b = 5.3\n";
    c = a + b;
    cout<<"a + b = ";
            c.show();
    c = a - b;
    cout<<"a - b = ";
            c.show();
    c = a / 5.3;
    cout<<"a / 5.3 = ";
            c.show();
    c = 10.6 * b;
    cout<<"10.6 * b = ";
            c.show();
    return 0;
}
```



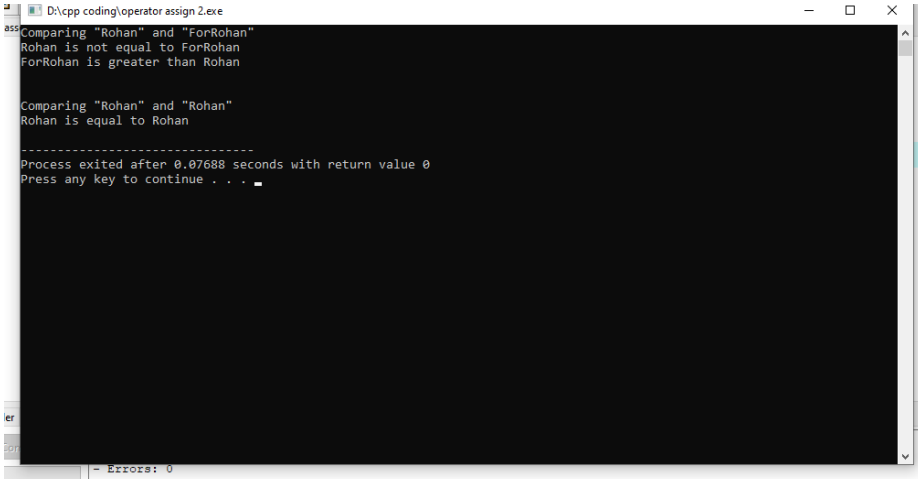
2. Define a class string. Overlaod ==operator to compare 2 strings.

→

```
#include <cstring>
#include <iostream>
#include <string.h>

using namespace std;


class CompareString {

public:

	char str[25];


	CompareString(char str1[])
	{

		strcpy(this->str, str1);
	}


	int operator==(CompareString s2)
	{
		if (strcmp(str, s2.str) == 0)
			return 1;
		else
			return 0;
	}


	int operator<=(CompareString s3)
	{
		if (strlen(str) <= strlen(s3.str))
			return 1;
		else
			return 0;
	}


	int operator>=(CompareString s3)
	{
		if (strlen(str) >= strlen(s3.str))
			return 1;
```

```
		else
			return 0;
	}
};

void compare(CompareString s1, CompareString s2)
{

	if (s1 == s2)
		cout << s1.str << " is equal to "
			<< s2.str << endl;
	else {
		cout << s1.str << " is not equal to "
			<< s2.str << endl;
		if (s1 >= s2)
			cout << s1.str << " is greater than "
				<< s2.str << endl;
		else
			cout << s2.str << " is greater than "
				<< s1.str << endl;
	}
}


void testcase1()
{

	char str1[] = "Rohan";
	char str2[] = "ForRohan";


	CompareString s1(str1);
	CompareString s2(str2);

	cout << "Comparing \"" << s1.str << "\" and \""
		<< s2.str << "\"" << endl;

	compare(s1, s2);
}


void testcase2()
{
```

```
        char str1[] = "Rohan";
        char str2[] = "Rohan";


        CompareString s1(str1);
        CompareString s2(str2);

        cout << "\n\nComparing \"" << s1.str << "\" and \""
                << s2.str << "\"" << endl;

        compare(s1, s2);
}


int main()
{
        testcase1();
        testcase2();

        return 0;
}
```



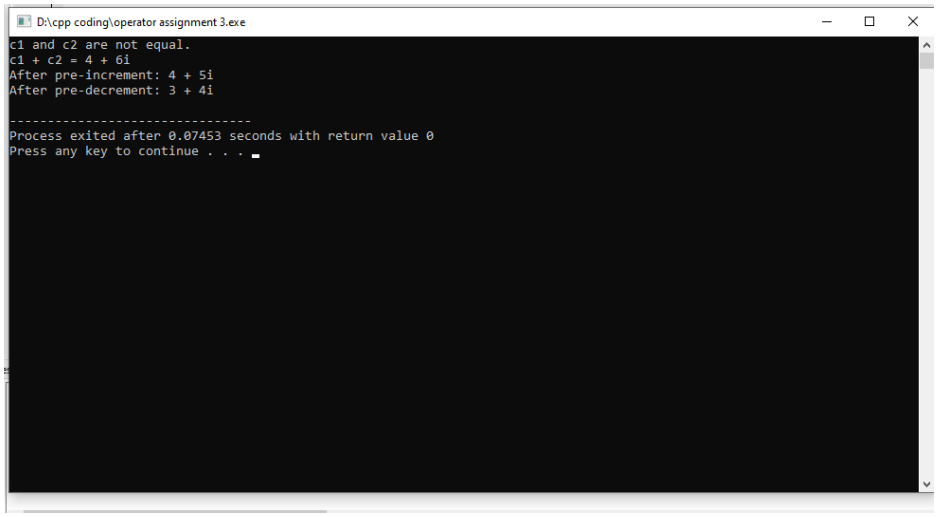
3.Create a Complex class that has real(int) and img(int) as member data, and has getData and showData functions. Then also overload the following operators for Complex class. =, ==, +, ++, --,

```
#include<iostream>

class Complex {
private:
```

```
    int real;
    int img;

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


    void getData(int &r, int &i) const {
        r = real;
        i = img;
    }


    void showData() const {
        if (img >= 0)
            std::cout << real << " + " << img << "i";
        else
            std::cout << real << " - " << -img << "i";
    }


    Complex operator=(const Complex &c) {
        real = c.real;
        img = c.img;
        return *this;
    }


    bool operator==(const Complex &c) const {
        return (real == c.real && img == c.img);
    }


    Complex operator+(const Complex &c) const {
        Complex result;
        result.real = real + c.real;
        result.img = img + c.img;
        return result;
    }


    Complex operator++() {
        ++real;
        ++img;
```

```
        return *this;
    }


    Complex operator--() {
        --real;
        --img;
        return *this;
    }
};

int main() {
    Complex c1(3, 4);
    Complex c2(1, 2);
    Complex c3;


    c3 = c1;


    if (c1 == c2)
        std::cout << "c1 and c2 are equal." << std::endl;
    else
        std::cout << "c1 and c2 are not equal." << std::endl;


    Complex c4 = c1 + c2;
    std::cout << "c1 + c2 = ";
    c4.showData();
    std::cout << std::endl;


    ++c1;
    std::cout << "After pre-increment: ";
    c1.showData();
    std::cout << std::endl;


    --c1;
    std::cout << "After pre-decrement: ";
    c1.showData();
    std::cout << std::endl;

    return 0;
```

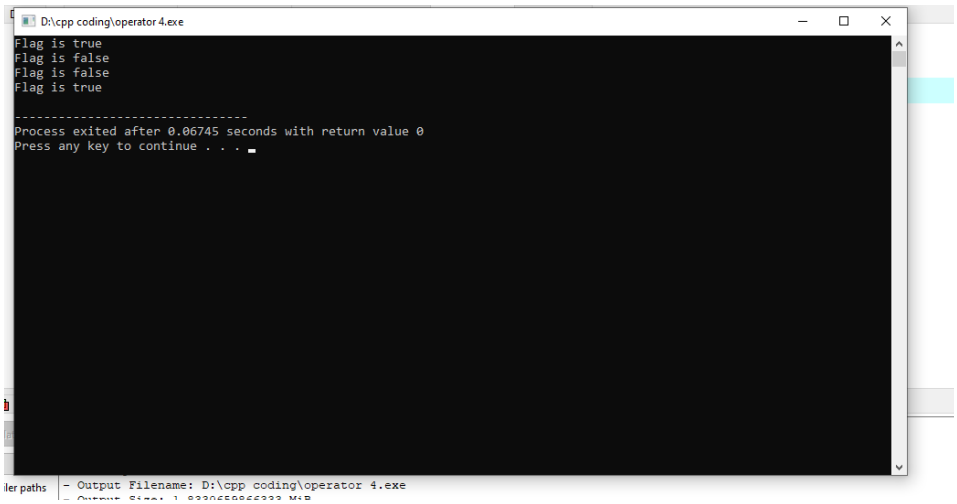
```
}
```



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

```
#include <iostream>
class MyClass {
private:
    bool flag;

public:
    MyClass(bool value) : flag(value) {}


    friend bool operator!(const MyClass& obj);

    void print() {
        std::cout << "Flag is " << (flag ? "true" : "false") << std::endl;
    }
};


bool operator!(const MyClass& obj) {

    return !obj.flag;
}

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

    obj1.print();
```

```
    obj2.print();


    MyClass result1 = !obj1;
    MyClass result2 = !obj2;

    result1.print();
    result2.print();

    return 0;
}
```



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

```
#include<iostream>

using namespace std;

class Distance {

private:

int meter;



friend int addFive(Distance);

public:
```

```
Distance() : meter(0) {}

};


int addFive(Distance d) {


d.meter += 5;

return d.meter;

}

int main() {

Distance D;

cout << "Distance: " << addFive(D);

return 0;

}
```