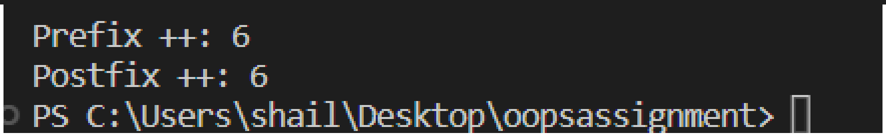
OOPS LABORATORY ASSIGNMENT-7

NAME: SHAILY SHEKHAR
ROLL NUMBER: 21CS8197

1. Implement a Code to execute Overload ++ when used as prefix and postfix.
CODE:

```
#include <iostream>

class Number {
    private:
        int value;

    public:
        Number(int value = 0) : value(value) {}


        Number& operator++() {
            ++value;
            return *this;
        }


        Number operator++(int) {
            Number temp = *this;
            ++(*this);
            return temp;
        }

        int getValue() const {
            return value;
        }
};

int main() {
    Number n1(5);

    ++n1;
    std::cout << "Prefix ++: " << n1.getValue() << std::endl;


    Number n2 = n1++;
    std::cout << "Postfix ++: " << n2.getValue() << std::endl;

    return 0;
}
```



```
Prefix ++: 6
Postfix ++: 6
PS C:\Users\shail\Desktop\oopsassignment>
```

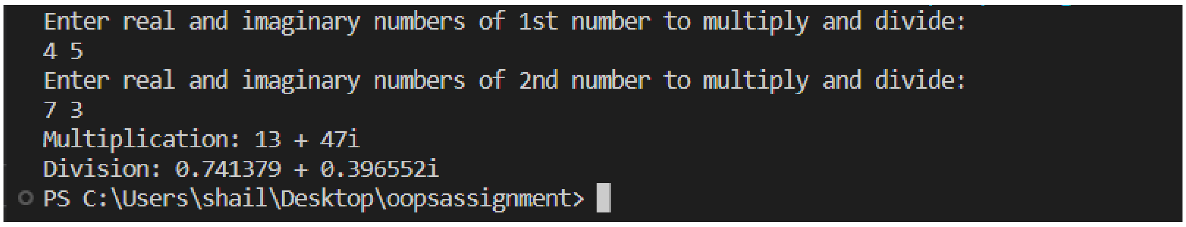
2. Implement a Complex class which performs the following operations:
Multiply (*), Divide (/).

CODE:

```
#include <iostream>
using namespace std;
class Complex {
private:
    double real;
    double imag;
public:
    Complex(double r = 0, double i = 0) : real(r), imag(i) {}

    // Multiply operator overload
    Complex operator*(const Complex& other) const {
        double r = real * other.real - imag * other.imag;
        double i = real * other.imag + imag * other.real;
        return Complex(r, i);
    }

    // Divide operator overload
    Complex operator/(const Complex& other) const {
        double r = (real * other.real + imag * other.imag) /
(other.real * other.real + other.imag * other.imag);
        double i = (imag * other.real - real * other.imag) /
(other.real * other.real + other.imag * other.imag);
        return Complex(r, i);
    }

    void print() const {
        std::cout << real << " + " << imag << "i" << std::endl;
    }
};
```

```
int main() {
    int a,b,c,d;
    cout<<"Enter real and imaginary numbers of 1st number to multiply
and divide:\n";
    cin>>a>>b;

    cout<<"Enter real and imaginary numbers of 2nd number to multiply
and divide:\n";
    cin>>c>>d;

    Complex c1(a,b);
    Complex c2(c,d);

    // Multiply
    Complex c3 = c1 * c2;
    std::cout << "Multiplication: ";
    c3.print();

    // Divide
    Complex c4 = c1 / c2;
    std::cout << "Division: ";
    c4.print();

    return 0;
}
```

OUTPUT:



```
Enter real and imaginary numbers of 1st number to multiply and divide:
4 5
Enter real and imaginary numbers of 2nd number to multiply and divide:
7 3
Multiplication: 13 + 47i
Division: 0.741379 + 0.396552i
PS C:\Users\shail\Desktop\oopsassignment>
```

3. Implement a Fraction class which performs the following operations:
   Add (+), Subtract (-), Multiply (*), Divide (/).

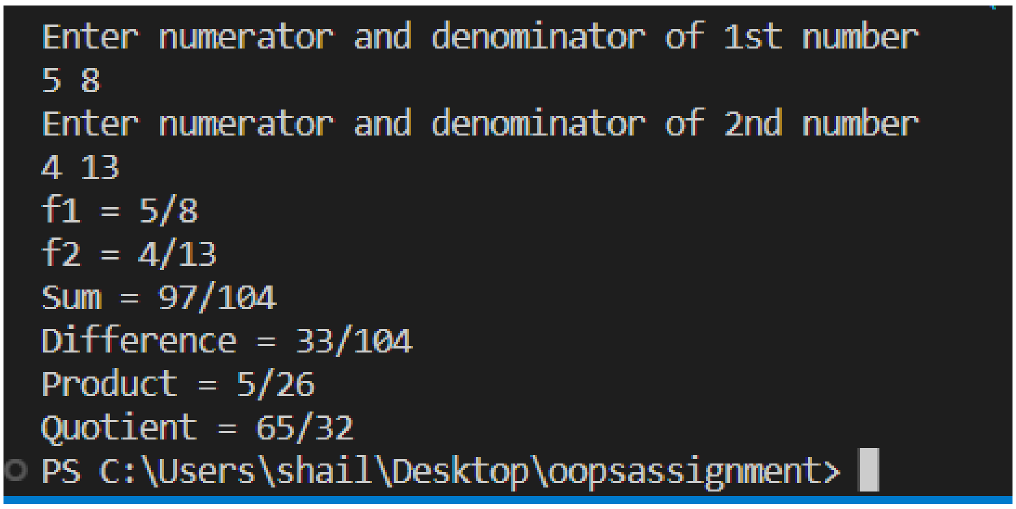
CODE:

```
#include <iostream>
using namespace std;
class Fraction {
private:
    int numerator;
```

```cpp
    int denominator;

    void simplify() {
        int gcd = getGCD(numerator, denominator);
        numerator /= gcd;
        denominator /= gcd;
    }

    int getGCD(int a, int b) {
        if (b == 0) {
            return a;
        }
        return getGCD(b, a % b);
    }

public:
    Fraction(int num, int denom) {
        if (denom == 0) {
            throw std::invalid_argument("Denominator cannot be zero");
        }
        numerator = num;
        denominator = denom;
        simplify();
    }

    Fraction operator+(const Fraction& other) const {
        int num = numerator * other.denominator + other.numerator *
denominator;
        int denom = denominator * other.denominator;
        return Fraction(num, denom);
    }

    Fraction operator-(const Fraction& other) const {
        int num = numerator * other.denominator - other.numerator *
denominator;
        int denom = denominator * other.denominator;
        return Fraction(num, denom);
    }

    Fraction operator*(const Fraction& other) const {
        int num = numerator * other.numerator;
        int denom = denominator * other.denominator;
        return Fraction(num, denom);
    }

    Fraction operator/(const Fraction& other) const {
        if (other.numerator == 0) {
            throw std::invalid_argument("Cannot divide by zero");
        }
        int num = numerator * other.denominator;
        int denom = denominator * other.numerator;
        return Fraction(num, denom);
    }

    friend std::ostream& operator<<(std::ostream& os, const Fraction&
f) {
        os << f.numerator << "/" << f.denominator;
        return os;
    }
};
int main() {
    int a,b;
    int c,d;
    cout<<"Enter numerator and denominator of 1st number\n";
    cin>>a>>b;
    cout<<"Enter numerator and denominator of 2nd number\n";
    cin>>c>>d;
    Fraction f1(a, b);
    Fraction f2(c, d);

    std::cout << "f1 = " << f1 << std::endl;
    std::cout << "f2 = " << f2 << std::endl;

    Fraction sum = f1 + f2;
    std::cout << "Sum = " << sum << std::endl;

    Fraction diff = f1 - f2;
    std::cout << "Difference = " << diff << std::endl;

    Fraction prod = f1 * f2;
    std::cout << "Product = " << prod << std::endl;

    Fraction quot = f1 / f2;
    std::cout << "Quotient = " << quot << std::endl;

    return 0;
```

```
}
```

OUTPUT:

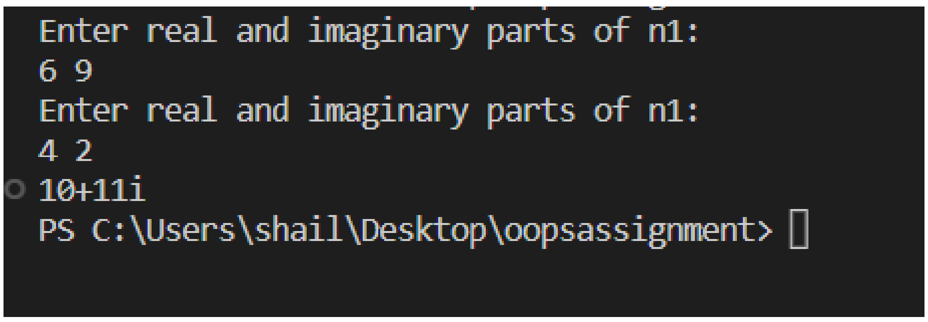


```
Enter numerator and denominator of 1st number
5 8
Enter numerator and denominator of 2nd number
4 13
f1 = 5/8
f2 = 4/13
Sum = 97/104
Difference = 33/104
Product = 5/26
Quotient = 65/32
PS C:\Users\shail\Desktop\oopsassignment>
```

4. Implement a Code to Overload ++ binary operator overloading.
CODE:

```
#include <bits/stdc++.h>
using namespace std;

class BinaryOperator{

  int img;
  int real;

  public :

  BinaryOperator()
  {
    img=0;
    real=0;

  }

  BinaryOperator(int r,int i)
  {
    real=r;
    img=i;
  }
  BinaryOperator operator + (BinaryOperator c)
{
    BinaryOperator temp;
    temp.img=img+c.img;
    temp.real=real+c.real;

    return temp;
}

void Print()
{
    cout<<real<<"+"<<img<<"i";
    cout<<endl;
}


};

int main()
{
 int a,b,c,d;
 cout<<"Enter real and imaginary parts of n1:\n";
 cin>>a>>b;
 cout<<"Enter real and imaginary parts of n1:\n";
 cin>>c>>d;
 BinaryOperator b1(a,b);
 BinaryOperator b2(c,d);

 BinaryOperator b3= b1+b2;

 b3.Print();
return 0;

}
```

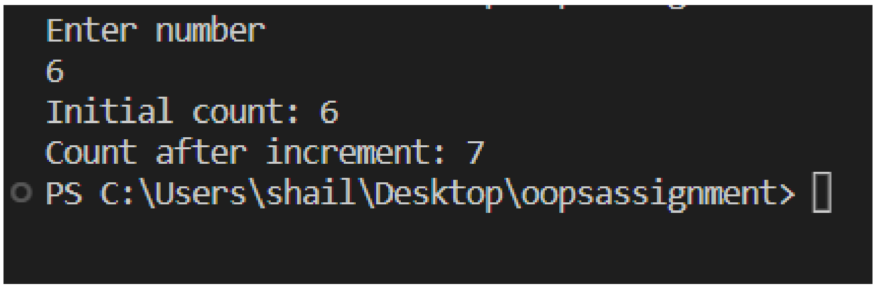
OUTPUT:



```
Enter real and imaginary parts of n1:
6 9
Enter real and imaginary parts of n1:
4 2
10+11i
PS C:\Users\shail\Desktop\oopsassignment> 
```

5. Implement a Code to Overload ++ Unary operator overloading.
CODE:

```
#include <iostream>
using namespace std;
class Counter {
private:
    int count;

public:
    Counter(int c = 0) {
        count = c;
    }

    Counter operator++() {
        ++count;
        return *this;
    }

    int getCount() const {
        return count;
    }
};

int main() {

    int num;
    cout<<"Enter number\n";
    cin>>num;
    Counter c(num);

    cout << "Initial count: " << c.getCount() << endl;

    ++c;
```

```cpp
    cout << "Count after increment: " << c.getCount() << endl;

    return 0;
}
```
OUTPUT:

```
Enter number
6
Initial count: 6
Count after increment: 7
PS C:\Users\shail\Desktop\oopsassignment>
```