Q1. Define a class Complex with appropriate instance variables and member functions.
Define following operators in the class:
a. +
b. -
c. *
d. ==

```
#include <iostream>
using namespace std;

class Complex {
	private:
		int real, imag;

	public:
		void setData(void) {

			cout << "Enter real part of Complex : ";
			cin >> real;

			cout << "Enter imaginary part of Complex : ";
			cin >> imag;
			cout<<endl<<endl;
		}

		void showData(void){

			cout<<real<<" + "<<imag<<"i"<<endl;
		}

		Complex operator+ (Complex);
		Complex operator- (Complex);
		Complex operator* (Complex);
		int operator== (Complex);
};

int main() {

	Complex c1, c2, ans;
	cout << "Enter data for c1"<<endl;
	c1.setData();

	cout << "Enter data for c2" <<endl;
	c2.setData();

	ans = c1 + c2;

	cout<<"Addition : ";
	ans.showData();
	cout<<endl;
```

```
        ans = c1 - c2;
        cout<<"Subtraction : ";
        ans.showData();
        cout<<endl;

        ans = c1 * c2;
        cout<<"Multiplication : ";
        ans.showData();
        cout<<endl;

        if( c1 == c2 )
                cout<<"c1 is equal to c2";
        else
                cout<<"c1 is not equal to c2";
}

Complex Complex::operator+ (Complex  x){

   Complex ans;
   ans.real = real + x.real;
   ans.imag = imag + x.imag;

   return ans;
}

Complex Complex::operator- (Complex c){

   Complex ans;

   ans.real = real - c.real;
   ans.imag = imag - c.imag;

   return ans;
}

Complex Complex::operator* (Complex c){

   Complex ans;

   ans.real = real * c.real;
   ans.imag = imag * c.imag;
   return ans;
}
int Complex::operator== (Complex c){
   if( (real == c.real)&&(imag == c.imag) )
      return 1;
   else
      return 0;
}
```

Q2. Write a C++ program to overload unary operators that is increment and decrement.

```
#include <iostream>
using namespace std;

class Unary{
        private :
                int a=5;
        public:
                void operator-- (void){
                 cout<<"Pre-Decrement : "<<--a<<endl;
                }

                  void operator-- (int){
                cout<<"Post-Decrement : "<<a--<<endl;
                 }

                    void operator++ (void){
                        cout<<"Pre-Increment : "<<++a<<endl;
                    }

                    void operator++ (int){
                       cout<<"Post-Increment : "<<a++<<endl;
                    }
};

int main(){

   Unary u;

   u--;
   --u;

   u++;
   ++u;
}
```

Q3. Write a C++ program to add two complex numbers using operator overloaded by a friend function.

```
#include <iostream>
using namespace std;

class Complex {
	private:
		int real, imag;
	public:
		void set() {

			cout << "Enter real part : ";
			cin >> real;

			cout << "Enter imaginary part : ";
			cin >> imag;
		}

		void show(void) {

			cout << real << " + " << imag << "i";
		}

		friend  Complex operator+ (Complex, Complex);

};


int main() {

	Complex c1, c2, c3;

	c1.set();
	c2.set();

	c3 = c1 + c2;

	c3.show();
}

Complex operator+ (Complex x, Complex y) {

	Complex tmp;

	tmp.real = y.real + x.real;
	tmp.imag = y.imag + x.imag;

	return tmp;
}
```

Q4. Create a class Time which contains:
- Hours
- Minutes
- Seconds

Write a C++ program using operator overloading for the following:
1. = = : To check whether two Times are the same or not.
2. >> : To accept the time.
3. << : To display the time.*/

```
#include <iostream>
using namespace std;

class Time {

	private:
		int hr, min, sec;
	public:
		friend istream &operator>> (istream &in, Time &x);
		friend ostream &operator<< (ostream &out, Time &x);
		friend int operator== (Time &x, Time &y);


};



int main() {

	Time t1, t2;

	cout << "Enter First Time" << endl;
	cout << "-------------------------------" << endl;
	cin >> t1;
	cout << "First Time" << endl << t1;
	cout << endl;

	cout << "Enter Second Time" << endl;
	cout << "-------------------------------" << endl;
	cin >> t2;
	cout << "Second Time" << endl << t2;
	cout << endl;

	if (t1 == t2)
		cout << "Time are same";
	else
		cout << "Time are not same";

}

istream &operator>> (istream &in, Time &x) {

	cout << "Enter Hour : ";
	in >> x.hr;
	cout << endl;

	cout << "Enter Min : ";
```

```
	in >> x.min;
	cout << endl;

	cout << "Enter Sec : ";
	in >> x.sec;
	cout << endl;

	return in;
}

ostream &operator<< (ostream &out, Time &x) {

	out << "Hour : " << x.hr << endl;
	out << "Minute : " << x.min << endl;
	out << "Second : " << x.sec << endl;

	return out;
}

int operator== (Time &x, Time &y) {
	if (x.hr == y.hr) {
		if (x.min == y.min) {
			if (x.sec == y.sec)
				return 1;
		}
	}
	return 0;
}
```

Q5. Consider following class Numbers

```
class Numbers
{
        int x,y,z;
    public:
        // methods
};
```

Overload the operator unary minus (-) to negate the numbers.

```
#include <iostream>
using namespace std;

class Numbers {

    private:
        int x, y, z;

    public:
        void set(void) {

            cout << "Enter value of x : ";
            cin >> x;

            cout << "Enter value of y : ";
            cin >> y;

            cout << "Enter value of z : ";
            cin >> z;
        }

        void operator- (void) {

            x = -x;
            y = -y;
            z = -z;
        }

        void show(void) {

            cout << "x : " << x << endl;
            cout << "y : " << y << endl;
            cout << "z : " << z;
        }
};

int main() {

    Numbers n;

    n.set();
    n.operator - ();
    n.show();
}
```

Q6. Create a class CString to represent a string.
a) Overload the + operator to concatenate two strings.
b) == to compare 2 strings.

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

using namespace std;

class CString {

	private:
		char str[10];

	public:

		void set(void);

		char *operator+ (const CString);

		int operator== ( CString);
};

int main() {

	CString s1, s2;

	char *p;

	s1.set();
	s2.set();

	cout << endl;

	if ((s1 == s2) == 0)
		cout << "Strings are same";
	else
		cout << "Strings are not same";

	cout << endl;

	p = s1 + s2;
	cout << "concat string = " << p;

}

void CString::set(void) {

	cout << "Enter string : ";
	fgets(str, 10, stdin);

	int len = strlen(str);

	if (str[len - 1] == 10) {

		str[len - 1] = '\0';
```

```
        }

        fflush(stdin);
}

char *CString::operator+ (const CString tmp) {

        return strcat(str, tmp.str);
}

int CString::operator==( CString tmp) {

        return (strcmp(str, tmp.str));
}
```

Q7. Define a C++ class fraction

```
class fraction
{
        long numerator;
        long denominator;
    Public:
        fraction (long n=0, long d=0);
}
```

Overload the following operators as member or friend:
a) Unary ++ (pre and post both)
b) Overload as friend functions: operators << and >>

```
#include <iostream>
using namespace std;

class fraction {

    private:
        long numerator;
        long denominator;

    public:
        fraction (long n = 0, long d = 0) {

            numerator = n;
            denominator = d;
        };

        friend void operator<<(ostream &out, fraction &tmp) {

            out << tmp.numerator << "/" ;
            out << tmp.denominator;
        }

        friend void operator>>(istream &in, fraction &tmp) {

            cout << "Numerator : ";
            in >> tmp.numerator;

            cout << "Denominator : ";
            in >> tmp.denominator;
        }

        fraction operator++(int) {

            fraction tmp = *this;

            numerator++;
            denominator++;

            return tmp;
        }

        fraction &operator++() {
```

```
                ++numerator;
                ++denominator;

                return *this;
        }
};

int main() {

        fraction f1, f2;

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

        cout << "Enter 1st Fraction value" << endl << endl;
        cin >> f1;

        cout << endl;

        cout << "f1++ : ";
        f1++;
        cout << f1 ;
        cout << endl;

        cout << "++f1 : ";
        ++f1;
        cout << f1 ;

        cout << endl << endl;

        cout << "Enter 2nd Fraction value" << endl << endl;
        cin >> f2;

        cout << endl;

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

        cout << endl << endl;

        cout << "f2 = f1++" << endl;
        f2 = f1++;
        cout << "f1 : " << f1;
        cout << endl;
        cout << "f2 : " << f2;
}
```

Q8. Consider a class Matrix

```
Class Matrix
{
        int a[3][3];
    Public:
        //methods;
};
```

Overload the - (Unary) should negate the numbers stored in the object.

```
#include <iostream>
using namespace std;

class Matrix {

    private:
        int a[3][3];

    public:

        void input() {

            cout << "Enter Matrix Element (3x3) : " << endl;

            for (int i = 0; i < 3; i++) {
                for (int j = 0; j < 3; j++) {

                    cin >> a[i][j];
                }
            }
        }

        void disp() {

            cout << "Matrix is : " << endl;

            for (int i = 0; i < 3; i++) {
                for (int j = 0; j < 3; j++) {

                    cout << a[i][j] << " ";
                }
                cout << endl;
            }
        }

        void operator-() {

            for (int i = 0; i < 3; i++) {
                for (int j = 0; j < 3; j++) {

                    a[i][j] = -a[i][j];
                }
            }
        }

};
```

```
int main() {

	Matrix m1;

	m1.input();

	m1.disp();
	cout << endl;

	-m1;
	m1.disp();

}
```

Q9.Consider the following class mystring

```
Class mystring
{
        char str [100];
    Public:
        // methods
};
```

Overload operator "!" to reverse the case of each alphabet in the string (Uppercase to Lowercase and vice versa).

```
#include <iostream>
using namespace std;

class mystring {

    private:
        char str[100];

    public:
        void get() {

            cout << "Enter string : ";
            fgets(str, 100, stdin);
        }

        void display() {

            cout << str;
        }

        void operator!() {

            for (int i = 0; str[i] != '\0'; ++i) {

                if (str[i] >= 65 && str[i] <= 96) {

                    str[i] = str[i] + 32;
                } else if (str[i] >= 97 && str[i] <= 122) {

                    str[i] = str[i] - 32;
                }
            }

        }
};

int main() {
    mystring s1;
    s1.get();
    cout << endl;
    cout << "Converted String : ";
    !s1;
    s1.display();
}
```

Q10.
Class Matrix
{
	int a[3][3];
Public:
	//methods;
};
Let m1 and m2 are two matrices. Find out m3=m1+m2 (use operator overloading).

```
#include <iostream>
using namespace std;

class Matrix {

	private:
		int a[3][3];

	public:
		void set(void);
		void display(void);
		Matrix operator+(Matrix tmp);
};

void Matrix::set() {

	cout << "Enter matrix : ";

	for (int i = 0; i < 3; ++i) {

		for (int j = 0; j < 3; ++j) {

			cin >> a[i][j];
		}
	}
}

void Matrix::display() {

	cout << "Addition is : " << endl;

	for (int i = 0; i < 3; ++i) {

		for (int j = 0; j < 3; ++j) {

			cout << a[i][j] << "  ";
		}
		cout << endl;
	}
}

Matrix Matrix::operator+(Matrix m) {

	Matrix tmp;

	for (int i = 0; i < 3; ++i) {
```

```
            for (int j = 0; j < 3; ++j) {

                tmp.a[i][j] = a[i][j] + m.a[i][j];
            }
        }

        return tmp;
}

int main() {

        Matrix m1, m2, m3;

        m1.set();
        m2.set();

        m3 = m1 + m2;

        m3.display();
}
```