Operator overloading and friend function

1. 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, img;
public:
    Complex operator+(Complex);
    Complex operator-(Complex);
    Complex operator*(Complex);
    bool operator==(Complex);
    void setData(int x, int y)
    {
        real = x;
        img = y;
    }
    void showData()
    {
        if (img >= 0)
            cout << real << " + " << img << "i" << endl;
        else
            cout << real << " - " << -img << "i" << endl;
    }
};
Complex Complex::operator-(Complex X)
{
    Complex temp;
    temp.real = real - X.real;
    temp.img = img - X.img;
    return temp;
}
Complex Complex::operator+(Complex X)
{
    Complex temp;
    temp.real = real + X.real;
    temp.img = img + X.img;
    return temp;
}
Complex Complex::operator*(Complex X)
{
    Complex temp;
    temp.real = real * X.real;
    temp.img = real * X.img;
    temp.img = temp.img + (img * X.real);
    temp.real = temp.real + (-(img * X.img));
    return temp;
}
bool Complex::operator==(Complex X)
{
    if(real == X.real && img == X.img)
        return 1;
    else
```

```
        return 0;
}
int main()
{
    Complex C1, C2, C3, C4, C5;
    bool b;
    C1.setData(5, -10);
    C2.setData(5, -10);
    C1.showData();
    C2.showData();
    C3 = C1 + C2;
    C3.showData();
    C4 = C1 - C2;
    C4.showData();
    C5 = C1 * C2;
    C5.showData();
    b = C1==C2;
    cout << b << endl;
    return 0;
}
```

```
Output:
5 - 10i
5 - 10i
10 - 20i
0 + 0i
-75 - 100i
1
```

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

```
#include <iostream>
using namespace std;
class Complex
{
private:
    int real, img;

public:
    void setData(int x, int y)
    {
        real = x;
        img = y;
    }
    void showData()
    {
        cout << "real = " << real << "      "
             << "img = " << img << "i" << endl;
    }
    Complex operator++()
    {
        Complex temp;
        temp.real = ++real;
        temp.img = ++img;
        return temp;
    }
    Complex operator--()
    {
        Complex temp;
        temp.real = --real;
        temp.img = --img;
        return temp;
    }
    Complex operator++(int)
    {
```

```
        Complex temp;
        temp.real = real++;
        temp.img = img++;
        return temp;
    }
    Complex operator--(int)
    {
        Complex temp;
        temp.real = real--;
        temp.img = img--;
        return temp;
    }
    Complex operator-()
    {
        Complex temp;
        temp.real = -real;
        temp.img = -img;
        return temp;
    }
};
int main()
{
    Complex C1, C2, C3;
    C1.setData(5, 15);
    C1 = -C1;
    C1.showData();
    C2.setData(10, 25);
    C2 = C2++;
    C2.showData();
    C3.setData(20, 40);
    C3 = C3--;
    C3.showData();
    C2 = ++C2;
    C2.showData();
    C3 = --C3;
    C3.showData();
    return 0;
}
```

```
================================================================================
Output:
real = -5       img = -15i
real = 10       img = 25i
real = 20       img = 40i
real = 11       img = 26i
real = 19       img = 39i
```

3. 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, img;

public:
    friend Complex operator+(Complex, Complex);
    void setData(int x, int y)
    {
        real = x;
        img = y;
    }
    void showData()
    {
```

```
        if (img >= 0)
            cout << real << " + " << img << "i" << endl;
        else
            cout << real << " - " << -img << "i" << endl;
    }
};
Complex operator+(Complex X, Complex Y)
{
    Complex temp;
    temp.real = X.real + Y.real;
    temp.img = X.img + Y.img;
    return temp;
}

int main()
{
    Complex c1, c2, c3;
    c1.setData(10, 15);
    c2.setData(-25, -62);
    //c3 = c1 + c2;
    c3 = operator+(c1, c2);
    c1.showData();
    c2.showData();
    c3.showData();
    return 0;
}
```

```
Output:
10 + 15i
-25 - 62i
-15 - 47i
```

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

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

   Output -

```
Enter First Time
--------------------
Enter Hours    :  24

Enter Minutes :  30

Enter Seconds :  40

First Time
Hours    :  24
Minutes :  30
Seconds :  40

Enter Second Time
--------------------
Enter Hours    :  24

Enter Minutes :  30

Enter Seconds :  40

Second Time
Hours    :  24
Minutes :  30
Seconds :  40

Times are Same
```

```cpp
#include <iostream>
using namespace std;
class Time
{
private:
    int Hours, Minutes, Seconds;

public:
    friend void operator>>(istream &in, Time &T);
    friend void operator<<(ostream &out, Time &T);
    friend void operator==(Time, Time);
};

void operator>>(istream &in, Time &T)
{
    cout << "----------------------" << endl;
    cout << "Enter Hours    :    ";
    in >> T.Hours;
    cout << endl <<"Enter Minutes :    ";
    in >> T.Minutes;
    cout << endl << "Enter Seconds :    ";
    in >> T.Seconds;
    cout << endl;
}
void operator==(Time T1, Time T2)
{
     if (T1.Hours == T2.Hours && T1.Minutes == T2.Minutes && T1.Seconds ==
T2.Seconds)
        cout << "Time are same";
    else
```

```
        cout << "Time are not same";
}
void operator<<(ostream &out, Time &T)
{
    out << "Hours         :    " << T.Hours << endl;
    out << "Minutes       :    " << T.Minutes << endl;
    out << "Seconds       :    " << T.Seconds << endl;
    out << endl;
}

int main()
{
    Time t1, t2;
    cout << "Enter first time" << endl;
    cin >> t1;
    cout << "First time" << endl;
    cout << t1;
    cout << "Enter second time" << endl;
    cin >> t2;
    cout << "Second time" << endl;
    cout << t2;
    //t1 == t2;
    operator==(t1, t2);
    return 0;
}
```

```
Output:
Enter first time
----------------------
Enter Hours   :    14

Enter Minutes :    45

Enter Seconds :    52

First time
Hours         :    14
Minutes       :    45
Seconds       :    52

Enter second time
----------------------
Enter Hours   :    12

Enter Minutes :    45

Enter Seconds :    52

Second time
Hours         :    12
Minutes       :    45
Seconds       :    52

Time are not same
```

5. 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 setData(int a, int b, int c)
    {
        x = a;
        y = b;
        z = c;
    }
    Numbers operator-()
    {
        Numbers n;
        n.x = -x;
        n.y = -y;
        n.z = -z;
        return n;
    }
    void showData()
    {
        cout << "x    =   " << x << endl;
        cout << "y    =   " << y << endl;
        cout << "z    =   " << z << endl;
    }
};

int main()
{
    Numbers n1, n2;
    n1.setData(5, 20, 9);
    n2 = -n1;
    n1.showData();
    n2.showData();
    return 0;
}
================================================================================
Output:
x    =   5
y    =   20
z    =   9
x    =   -5
y    =   -20
z    =   -9
```

6. 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 c1[50];
    static char c3[100];

public:
    void inputData()
    {
        cout << "Enter String  : ";
        cin.getline(c1, 50);
    }
    char *operator+(CString);
    friend void operator==(CString, CString);
};
char CString::c3[100];
char *CString::operator+(CString S)
{
    int l = (strlen(c1) + strlen(S.c1));
    for (int i = 0; i < l; i++)
    {
        if (i < strlen(c1))
            c3[i] = c1[i];
/*          if (i == strlen(c1))
            c3[i] = ' '; */
        if (i >= strlen(c1))
            c3[i] = S.c1[i - strlen(c1)];
    }
    return c3;
}
void operator==(CString s1, CString s2)
{
    if (strcmp(s1.c1, s2.c1) == 0)
        cout << "\nStrings are equal." << endl;
    else
        cout << "\nStrings are not equal." << endl;
}
int main()
{
    CString s1, s2;
    char *p = NULL;
    s1.inputData();
    s2.inputData();
    p = s1 + s2;
    for (int i = 0; *(p + i); i++)
        cout << *(p + i);
    s1==s2;
    return 0;
}
================================================================================
Output:
Enter String  : ankit
Enter String  : akash
ankitakash
Strings are not equal.
```

7. 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 >>.
Output-

```
f1    :  0/0
f2    :  0/0

Enter 1st Fraction Value

Numerator    :   2

Denominator  :   3

f1++  :  3/4
++f1  :  4/5

Enter 2nd Fraction Value

Numerator    :   1

Denominator  :   2

f2 = ++f1
f1    :   5/6
f2    :   5/6

f2 = f1++
f1    :  6/7
f2    :  5/6
```

```
#include <iostream>
#include <stdio.h>
using namespace std;

class fraction
{
private:
    long numerator, denominator;

public:
    friend void operator>>(istream &in, fraction &f);
    friend void operator<<(ostream &out, fraction &f);
    fraction(long int n = 0, long int d = 0)
    {
        numerator = n;
        denominator = d;
    }
    fraction operator++()
    {
        fraction temp;
        temp.numerator = ++numerator;
        temp.denominator = ++denominator;
        return temp;
    }
    fraction operator++(int)
    {
        fraction temp;
        temp.numerator = numerator++;
        temp.denominator = denominator++;
        return temp;
    }
```

```
};

void operator>>(istream &in, fraction &f)
{
    cout << endl;
    cout << "Numerator      :      ";
    in >> f.numerator;
    cout << "\nDenumerator    :      ";
    in >> f.denominator;
    cout << endl;
}

void operator<<(ostream &out, fraction &f)
{
    out << f.numerator << "/" << f.denominator;
    out << endl;
}

int main()
{
    fraction f1, f2;
    cout << "f1     :       ";
    cout << f1;
    cout << "f2     :       ";
    cout << f2;
    cout << endl;
    cout << "Enter first fraction value" << endl;
    cin >> f1;
    cout << "f1++           :     ";
    f1++;
    cout << f1;
    cout << "++f1           :     ";
    ++f1;
    cout << f1;
    cout << endl;
    cout << "Enter Second fraction value" << endl;
    cin >> f2;
    cout << "f2     =     ++f1" << endl;
    f2 = ++f1;
    cout << "f1     :       ";
    cout << f1;
    cout << "f2     :       ";
    cout << f2;
    cout << "\nf2     =     f1++" << endl;
    f2 = f1++;
    cout << "f1     :       ";
    cout << f1;
    cout << "f2     :       ";
    cout << f2;
    return 0;
}
```

```
================================================================================
Output:
f1     :       0/0
f2     :       0/0

Enter first fraction value

Numerator      :      2

Denumerator    :      3

f1++           :     3/4
++f1           :     4/5
```

```
Enter Second fraction value

Numerator       :       1

Denumerator     :       2

f2      =      ++f1
f1      :        5/6
f2      :        5/6

f2      =      f1++
f1      :        6/7
f2      :        5/6
```

8. Consider a class Matrix
   Class Matrix
   {
   int a[3][3];
   Public:
   //methods;
   };
   Overload the - (Unary) should negate the numbers stored in the object.
   Output -

```
Enter Matrix Element (3 X 3) :

7
8
9
1
2
3
4
5
6

Matrix is :

7       8       9
1       2       3
4       5       6

Matrix is :

-7      -8      -9
-1      -2      -3
-4      -5      -6
```

```cpp
#include <iostream>
using namespace std;
class Matrix
{
private:
    int a[3][3];

public:
    void inputData();
    void showData();
    void operator-();
};
void Matrix ::inputData()
{
    cout << "Enter Matrix Elements (3x3)" << endl;
```

```
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            cin >> a[i][j];
}
void Matrix::showData()
{
    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 Matrix::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.inputData();
    m1.showData();
    -m1;
    m1.showData();
    return 0;
}
```

```
================================================================================
Output:
Enter Matrix Elements (3x3)
7
8
9
1
2
3
4
5
6
Matrix is :
7        8        9
1        2        3
4        5        6
Matrix is :
-7        -8        -9
-1        -2        -3
-4        -5        -6
```

9. 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>
#include <string.h>
using namespace std;

class mystring
{
    private:
        char str[100];
    public:
        void inputData();
        void showData();
        void operator!();
};
void mystring::inputData()
{
    cout << "Enter a string : ";
    cin.getline(str, 100);
}

void mystring::showData()
{
    cout << "String is : " << str << endl;
}

void mystring::operator!()
{
    for (int i = 0; str[i]; i++)
    {
        if (str[i] >= 'a' && str[i] <= 'z')
            str[i] = str[i] - 32;
        else
            str[i] = str[i] + 32;
    }
}

int main()
{
    mystring m;
    m.inputData();
    m.showData();
    !m;
    m.showData();
    return 0;
}
```

```
Output:
Enter a string : ZameeR
String is : ZameeR
String is : zAMEEr
```

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

```
Enter Matrix Element (3 X 3) :
4 5 6 1 2 3 7 8 9

Enter Matrix Element (3 X 3) :
1 2 3 4 5 6 7 8 9

First Matrix :

4       5       6
1       2       3
7       8       9

Second Matrix :

1       2       3
4       5       6
7       8       9

Addition of Matrix :

5       7       9
5       7       9
14      16      18
```

```cpp
#include <iostream>
using namespace std;
class Matrix
{
private:
    int a[3][3];

public:
    void inputData();
    void showData();
    void operator+(Matrix M);
};
void Matrix ::inputData()
{
    cout << "Enter Matrix Elements (3x3)" << endl;
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            cin >> a[i][j];
}
void Matrix::showData()
{
    cout << "\nMatrix is :" << endl;
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            cout << a[i][j] << "\t";
        }
        cout << endl;
    }
}
void Matrix::operator+(Matrix M)
{
    int c[3][3];
```

```
    // sum of two matrix
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            c[i][j] = a[i][j] + M.a[i][j];
        }
    }
    // print sum of two matrix
    cout << "\nSum of two matrices" << endl;
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            cout << c[i][j]  << "\t";
        }
        cout << endl;
    }
}
int main()
{
    Matrix m1, m2;
    m1.inputData();
    m2.inputData();
    m1.showData();
    m2.showData();
    m1 + m2;
    return 0;
}
```

==========

```
Output:
Enter Matrix Elements (3x3)
4 5 6 1 2 3 7 8 9
Enter Matrix Elements (3x3)
1 2 3 4 5 6 7 8 9

Matrix is :
4       5       6
1       2       3
7       8       9

Matrix is :
1       2       3
4       5       6
7       8       9

Sum of two matrices
5       7       9
5       7       9
14      16      18
```