Assignment – 27

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

```cpp
        else
            return false;
    }
};

int main()
{
    Complex c1,c2,c3,c4,c5;
    c1.setData(5,6);
    c2.setData(6,7);
    c1.print();
    c2.print();
    c3 = c1+c2;
    c3.print();
    c4 = c1-c2;
    c4.print();
    c5 = c1*c2;
    c5.print();
    if(c1==c2)
        cout<<"True";
    else
        cout<<"False";
    return 0;
}
```

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

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

class operation
{
    private:
        int value;
    public:
    operation(int x)
    {
        value = x;
    }
    void print()
    {
        cout<<value<<endl;
    }
    void operator++()       //++ used as prefix
    {
        ++value;
    }
    void operator++(int)    //++ used as postfix
    {
        value++;
    }
    void operator--()      //++ used as prefix
    {
        --value;
    }
    void operator--(int)    //++ used as postfix
    {
        value--;
    }
};

int main()
{
    operation c1(5);
    c1.print();
    ++c1;
    c1.print();
    c1++;
    c1.print();
    --c1;
    c1.print();
    c1--;
    c1.print();
    return 0;
}
```

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;
        int img;
    public:
    void setData(int x,int y)
    {
        real = x;
        img = y;
    }
    void print()
    {
        if(img>=0)
            cout<<real<<" + "<<img<<"i"<<endl;
        else
            cout<<real<<" - "<<img*(-1)<<"i"<<endl;
    }
    friend Complex operator+(Complex ,Complex);
};

Complex operator+(Complex a,Complex b)
{
    Complex temp;
    temp.real = a.real + b.real;
    temp.img = a.img + b.img;
    return temp;
}

int main()
{
    Complex c1,c2,c3;
    c1.setData(5,8);
    c2.setData(2,3);
    c1.print();
    c2.print();
    c3 = c1+c2;
    c3.print();
    return 0;
}
```

4. 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 h;
        int m;
        int s;
    public:
        Time(int a=0,int b=0,int c=0)
        {
            h = a;
            m = b;
            s = c;
        }
        bool operator==(Time c)
        {
            if((h==c.h)&&(m==c.m)&&(s==c.s))
                return true;
            else
                return false;
        }
        friend ostream & operator<<(ostream &out,const Time &c);
        friend istream & operator>>(istream &in,Time &c);
};
istream & operator>>(istream &in,Time &c)
{
    cout<<"Enter Hours : ";
    in>>c.h;
    cout<<"Enter Minutes : ";
    in>>c.m;
    cout<<"Enter Seconds :";
    in>>c.s;
    return in;
}
ostream & operator<<(ostream &out,const Time &c)
{
    out<<"Hours   : "<<c.h <<endl;
    out<<"Minutes : "<<c.m <<endl;
    out<<"Seconds : "<<c.s <<endl;
    return out;
}

int main()
{
    Time t1,t2;
    cout<<"Enter First Time"<<endl;
    cout<<"--------------------"<<endl;
    cin>>t1;
    cout<<endl;
    cout<<"First Time"<<endl;
    cout<<"---------------"<<endl;
    cout<<t1;
    cout<<endl;
    cout<<"Enter Second Time"<<endl;
    cout<<"--------------------"<<endl;
    cin>>t2;
    cout<<endl;
    cout<<"Second Time"<<endl;
    cout<<"---------------"<<endl;
    cout<<t2;
    cout<<endl;
    if(t1==t2)
        cout<<"Times are same";
    else
        cout<<"Time is not same";
    return 0;
}
```

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 operator-()
        {
            x*=(-1);
            y*=(-1);
            z*=(-1);
        }
        void setData(int a,int b,int c)
        {
            x = a;
            y = b;
            z = c;
        }
        void display()
        {
            cout<<"Numbers are : "<<x<<" , "<<y<<" , "<<z<<endl;
        }
};

int main()
{
    Numbers n1,n2;
    n1.setData(7,-8,10);
    n2.setData(15,5,-7);
    n1.display();
    n2.display();
    -n1;
    -n2;
    n1.display();
    n2.display();
    return 0;
}
```

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<cstring>
using namespace std;

class CString
{
    private:
        char a[100];
    public:
        void setdata()
        {
            cin.getline(a,100);
        }
        void print()
        {
            cout<<a;
        }
        friend CString operator+(CString &str1,const CString &str2) ;
};
CString operator+(CString &str1,const CString &str2)
        {
            CString b;
            strcat(str1.a,str2.a);
            strcpy(b.a,str1.a);
            return b;
        }

int main()
{
    CString s1,s2,s3;
    cout<<"Enter First String : ";
    s1.setdata();
    cout<<"Enter Second String : ";
    s2.setdata();
    s3 =s1+s2;
    s3.print();
    return 0;
}
```

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 >>.

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

class fraction
{
    private:
        long numerator;
        long denominator;
    public:
        fraction(long n=0,long d=0)
        {
            numerator = n;
            denominator = d;
        }
        void setData()
        {
            cout<<"Numerator : ";
            cin>>numerator;
            cout<<"Denominator : ";
            cin>>denominator;
        }
```

```cpp
        void display()
        {
            cout<<numerator<<"/"<<denominator<<endl;
        }
        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;
        }
};

int main()
{
    fraction f1,f2,f3;
    cout<<"f1 : ";
    f1.display();
    cout<<"f2 : ";
    f2.display();
    cout<<endl;
    cout<<"Enter 1st Fraction value : "<<endl<<endl;
    f1.setData();
    cout<<endl;
    f1++;
    cout<<"f1++ : ";
    f1.display();
    ++f1;
    cout<<"++f1 : ";
    f1.display();
    cout<<endl;
    cout<<"Enter 2nd Fraction value : "<<endl<<endl;
    f2.setData();
    cout<<endl;
    cout<<"f2 = ++f1 "<<endl;
    f2 = ++f1;
    cout<<"f1 : ";
    f1.display();
    cout<<"f2 : ";
    f2.display();
    cout<<endl;
    cout<<"f2 = f1++"<<endl;
    f2 = f1++;
    cout<<"f1 : ";
    f1.display();
    cout<<"f2 : ";
    f2.display();
    return 0;
}
```

8. 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 operator-()
        {
            for(int i=0;i<3;i++)
            {
                for(int j=0;j<3;j++)
                    a[i][j]*=(-1);
            }
        }
        void print()
        {
            cout<<"Matrix is : "<<endl<<endl;
            for(int i=0;i<3;i++)
            {
                for(int j=0;j<3;j++)
                {
                    cout<<a[i][j]<<"\t";
                }
                cout<<endl;
            }
        }
        void setData()
        {
            for(int i=0;i<3;i++)
            {
                for(int j=0;j<3;j++)
                    cin>>a[i][j];
            }
        }
};

int main()
{
    Matrix m1;
    cout<<"Enter Matrix Element (3 X 3) : ";
    m1.setData();
    m1.print();
    -m1;
    cout<<endl;
    m1.print();
    return 0;
}
```

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<cstring>
using namespace std;

class mystring
{
    private:
        char str[100];
    public:
        void input()
        {
            cin.getline(str,100);
        }
        void print()
        {
            cout<<str<<endl;
        }
        void operator!()
        {
            for(int i=0;i<strlen(str);i++)
            {
                if(str[i]>=65 && str[i]<=90)
                {
                    str[i] = str[i]+32;
                }
                else if(str[i]>=97 && str[i]<=122)
                {
                    str[i] = str[i]-32;
                }
                else
                    continue;
            }
        }
};
```

```
int main()
{
    mystring s1;
    cout<<"Enter String : ";
    s1.input();
    cout<<endl;
    s1.print();
    !s1;
    cout<<endl;
    s1.print();
    return 0;
}
```

10. 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:
        Matrix operator+(Matrix m)
        {
            Matrix temp;
            for(int i=0;i<3;i++)
            {
                for(int j=0;j<3;j++)
                {
                    temp.a[i][j] = a[i][j] + m.a[i][j];
                }
            }
            return temp;
        }
        void print()
        {
            for(int i=0;i<3;i++)
            {
                for(int j=0;j<3;j++)
                {
                    cout<<a[i][j]<<"\t";
                }
                cout<<endl;
            }
        }
```

```cpp
        void setData()
        {
            for(int i=0;i<3;i++)
            {
                for(int j=0;j<3;j++)
                    cin>>a[i][j];
            }
        }
};

int main()
{
    Matrix m1,m2,m3;
    cout<<"Enter Matrix Element (3 X 3) : ";
    m1.setData();
    cout<<"Enter Matrix Element (3 X 3) : ";
    m2.setData();
    cout<<"First Matrix : ";
    cout<<endl;
    m1.print();
    cout<<endl;
    cout<<"Second Matrix : ";
    cout<<endl;
    m2.print();
    m3 = m1+m2;
    cout<<endl;
    cout<<"Addition of Matrix : ";
    cout<<endl;
    m3.print();
    return 0;
}
```