Lab Sheet 5
Understanding the Concept of Friend Function/Class and Operator Overloading

Friend Function and Class

In some cases you need to access the private data members of a class from non member functions. In such situations you must declare the function as friend function of the class. This friend function seems to violate the data hiding feature of OOP concept. However, the function that accesses private data must be declared as friend function within the class. With friend functions data integrity is still maintained.
Sometimes you may need to make, one or all the member functions of a class friend to other class. For that we declare class or member function as the friend to the other class so that one or all the member functions of the declared class will be friend to the class within which it is declared.
Example:

```
class breadth;

class length

{


    private:

        ......

    public:
        ......

        friend int add(length,    //friend function
        breadth);

    declaration
```

```
};
class breadth

{



    private:

        ......

    public:
        ......

        friend int add(length,    //friend function

        breadth);
    Declaration};

    int add( length l, breadth
    b) { }
```

Operator Overloading

We have already studied that we can make user defined data types behave in much the same way as the built-in types. C++ also permits us to use operators with user defined types in the same way as they are applied to the basic types. We need to represent different data items by objects and operations on those objects by operators. Operator can be overloaded for those different operations. Most programmers implicitly use overloaded operators regularly. For example, the addition operator (+) operates quite differently on integers, float and doubles and other built-in types because operator (+) has been overloaded in the C++ language itself. Operators are overloaded by writing a function definition as you normally would, except that the function name now becomes the keyword operator followed by the symbol for the operator being overloaded. Operator overloading provides a flexible option for the creation of new definitions for most of the C++ operators for your class.

Syntax

```
<return type> operator
<operator_symbol> ( <parameters> )
{
    <statements>;
}
```

example:

```
    complex operator + (complex c1,complex c2)
    { return
    }  complex(c1.real+c2.real,c1.imag+c2.imag);
```

Exercises

1. Write a class for instantiating the objects that represent the two-dimensional Cartesian coordinate system.

A. make a particular member function of one class to friend function in another class for addition

```
#include <iostream>
using namespace std;
class sample2;
class sample1
   {
    int firstd;
    public:
    sample1(int x):firstd(x){}
   friend int sum(sample1
   n1,sample2 n2); };
class sample2
   {
    int secondd;
    public:
    sample2(int y):secondd(y){}
   friend int sum(sample1
   n1,sample2 n2); };
  int sum(sample1 aobj,sample2 bobj)
  {
   return aobj.firstd+bobj.secondd;
  }
int main()
{
  sample1 obja(10);
  sample2 objb(15);
```

```
  cout<<"Sum =
  "<<sum(objа,objb); return 0;
}
```

B. make other three functions to work as a bridge between the classes for multiplication, division and subtraction.

```
#include <iostream>
using namespace std;
class sample2;
class sample1
{
int firstd;
public:
sample1(int x):firstd(x){} friend
int sum(sample1 n1,sample2 n2);
friend int sub(sample1 n1,sample2
n2); friend int mul(sample1
n1,sample2 n2); friend int
div(sample1 n1,sample2 n2); };
class sample2
{
int secondd;
public:
sample2(int y):secondd(y){} friend
int sum(sample1 n1,sample2 n2);
friend int sub(sample1 n1,sample2
n2); friend int mul(sample1
n1,sample2 n2); friend int
div(sample1 n1,sample2 n2); };
int sum(sample1 aobj,sample2 bobj)
{
return aobj.firstd+bobj.secondd;
}
int sub(sample1 aobj,sample2 bobj)
{
return aobj.firstd-bobj.secondd;
}
int mul(sample1 aobj,sample2 bobj)
{
return aobj.firstd*bobj.secondd;
```

```
}
int div(sample1 aobj,sample2 bobj)
{
return aobj.firstd/bobj.secondd;
}
int main()
{
sample1 obja(150);
sample2 objb(10);
cout<<"Sum = "<<sum(obja,objb)<<endl;
cout<<"Subtraction = "<<sub(obja,objb)<<endl;
cout<<"Multiplication =
"<<mul(obja,objb)<<endl; cout<<"Division =
"<<div(obja,objb)<<endl; return 0;
}
```

C. Also write a small program to demonstrate that all the member functions of one class are the friend functions of another class if the former class is made friend to the latter. Make least possible classes to demonstrate all above in single program without conflict.

```
#include <iostream>
using namespace std;
class sample2;
class sample1
{
int firstd;
public:
sample1(int x):firstd(x){}
friend class sample2;
};
class sample2
{
int secondd;
public:
sample2(int y):secondd(y){}
void display (sample1 obja)
{
cout<<"Data form class sample1:
"<<obja.firstd<<endl;
```

```
cout<<"Data form class sample2:
"<<this->secondd<<endl;
cout<<"Sum of alpha and sample1and sample2:
"<<secondd+obja.firstd<<endl;
}
};
int main()
{
sample1 obja(70);
sample2 objb(10);
objb.display(obja);
return 0;
}
```

2. Write a class to store x, y, and z coordinates of a point in three-dimensional space. Using operator overloading, write friend functions to add, and subtract the vectors.

```
#include <iostream>
using namespace std;
class coord2;
class coord1
{
int x1,y1,z1;
public:
void get_cord()
{
cout<<"Enter x1: ";
cin>>x1;
cout<<"Enter y1: ";
cin>>y1;
cout<<"Enter z1: ";
cin>>z1;
}
friend void operator+(coord1 d1,coord2
d2); friend void operator-(coord1
d1,coord2 d2); };
class coord2
{
int x2,y2,z2;
```

```
public:
void get_cord()
{
cout<<"Enter x2: ";
cin>>x2;
cout<<"Enter y2: ";
cin>>y2;
cout<<"Enter z2: ";
cin>>z2;
}
friend void operator+(coord1 d1,coord2
d2); friend void operator-(coord1
d1,coord2 d2); };
void operator+(coord1 d1,coord2 d2)
{
cout<<endl<<"Sum of two
vectors are ("<<d1.x1+d2.x2;
cout<<", "<<d1.y1+d2.y2;
cout<<", "<<d1.z1+d2.z2<<")";
}
void operator-(coord1 d1,coord2 d2)
{
cout<<endl<<"Diffrence of two vectors are
("<<d1.x1-d2.x2;
cout<<", "<<d1.y1-d2.y2;
cout<<", "<<d1.z1-d2.z2<<")";
}
int main()
{
coord1 d1;
coord2 d2;
d1.get_cord();
d2.get_cord();
d1+d2;
d1-d2;
return 0;
}
```

3. Compare the two object that contains integer values that demonstrate the overloading of equality (==), less than (<), greater than (>), not equal (!

=),greater than or equal to (>=) and less than or

equal to(<=) operators.

```
#include <iostream>
using namespace std;
class data2;
class data1
{
int x;
public:
void get_data()
{
cout<<"Enter x: ";
cin>>x;
}
friend void operator==(data1 x,data2
y); friend void operator<(data1
x,data2 y); friend void
operator>(data1 x,data2 y); friend
void operator!=(data1 x,data2 y);
friend void operator>=(data1 x,data2
y); friend void operator<=(data1
x,data2 y); };
class data2
{
int y;
public:
void get_data()
{
cout<<"Enter y: ";
cin>>y;
}
friend void operator==(data1 x,data2
y); friend void operator<(data1
x,data2 y); friend void
operator>(data1 x,data2 y); friend
void operator!=(data1 x,data2 y);
friend void operator>=(data1 x,data2
y); friend void operator<=(data1
x,data2 y); };
```

```
void operator==(data1 d1,data2 d2)
{
if(d1.x==d2.y)cout<<" both are equal.
"<<endl; else cout<<" they are not equal.
"<<endl; }
void operator<(data1 d1,data2 d2)
{
if (d1.x<d2.y) cout<<d1.x<<" is less
than other data."<<endl;
else cout<<d2.y<<" is less than other data."<<endl;
}
void operator>(data1 d1,data2 d2)
{
if (d1.x>d2.y) cout<<d1.x<<" is more
than other data."<<endl;
else cout<<d2.y<<" is more than other data."<<endl;
}
void operator!=(data1 d1,data2 d2)
{
if(d1.x!=d2.y) cout<<" they are not equal.
"<<endl; else cout<<" they are
equal."<<endl; }
void operator>=(data1 d1,data2 d2)
{
if (d1.x>=d2.y) cout<<d1.x<<" is more than
or equal to other data."<<endl;
else cout<<d2.y<<" is more than or equal to
other data."<<endl;
}
void operator<=(data1 d1,data2 d2)
{
if (d1.x<=d2.y) cout<<d1.x<<" is less than
or equal to other data."<<endl;
else cout<<d2.y<<" is less than or equal to
other data."<<endl;
}
int main()
{
data1 d1;
data2 d2;
```

```
d1.get_data();
d2.get_data();
d1==d2;
d1<d2;
d1>d2;
d1!=d2;
d1>=d2;
d1<=d2;
return 0;
}
```

4.Write a class Date that uses pre increment and post increment operators to add 1 to the day in the Date object, while causing appropriate increments to the month and year (use the appropriate condition for leap year). The pre and post increment operators in your Date class should behave exactly as the built in increment operators.

```
#include <iostream>
using namespace std;
class date
{
    int y,m,d;
public:
    void get_data()
    {
        cout<<"Enter valid date.";
        cout<<endl<<"Enter year: ";
        cin>>y;
        cout<<endl<<"Enter month: ";
        cin>>m;
        cout<<endl<<"Enter day: ";
        cin>>d;
    }
    void operator++(int)
    {
```

```
        cout <<y<<":"<<m<<":"<<d++<<endl;
    }
    void operator++()
    {
        ++d;
        if (((y%4==0) && (y%100==0) && (y%400==0)) ||
((y%4==0) && (y%100!=0)))
        {
            if ((m/2==1) && (29<d))
            {
               m++;
               d=d-29;
         }
        }
        else
        {
            if((m/2==1) && (28<d))
              {
                 m++;
                d=d-28;
           }
        }
    if ((m%2==1) && (31<d))
        {
         m++;
         d=d-31;
        }
        if ((m%2==0) && (m/2!=1) && (30<d))
        {
     m++;
     d=d-30;
    }
    if(12<m)
        {
            m=1;
```

```
            y++;
        }
        cout <<y<<":"<<m<<":"<<d<<endl;
    }
};
    int main()
    {
        date yyyy;
        yyyy.get_data();
        cout<<endl<<"Prefix Operator
Overloaded."<<endl;
        yyyy++;
        cout<<endl<<"Postfix Operator
Overloaded."<<endl;
        ++yyyy;
        return 0;
    }
```