**Operator overloading using Friend**

* Friend function using operator overloading offers better flexibility to the class.
* When you overload a unary operator, you have to pass one argument.
* When you overload a binary operator, you have to pass two arguments.
* Friend function can access private members of a class directly.

**Syntax:**

friend return-type operator operator-symbol (Variable 1, Varibale2)  
{  
     //Statements;  
}

Example: Program demonstrating Unary operator overloading using Friend function

#include<iostream>  
using namespace std;  
class UnaryFriend  
{  
     int a=10;  
     int b=20;  
     int c=30;  
     public:  
         void getvalues()  
         {  
              cout<<"Values of A, B & C\n";  
              cout<<a<<"\n"<<b<<"\n"<<c<<"\n"<<endl;  
         }  
         void show()  
         {  
              cout<<a<<"\n"<<b<<"\n"<<c<<"\n"<<endl;  
         }  
         void friend operator-(UnaryFriend &x);      //Pass by reference  
};  
void operator-(UnaryFriend &x)  
{  
     x.a = -x.a;     //Object name must be used as it is a friend function  
     x.b = -x.b;  
     x.c = -x.c;  
}  
int main()  
{  
     UnaryFriend x1;  
     x1.getvalues();  
     cout<<"Before Overloading\n";  
     x1.show();  
     cout<<"After Overloading \n";  
     -x1;  
      x1.show();  
      return 0;  
}

**Output:**  
Values of A, B & C  
10  
20  
30  
  
Before Overloading  
10  
20  
30  
  
After Overloading  
-10  
-20  
-30  
  
In the above program, **operator –** is overloaded using friend function. The **operator ()** function is defined as a **Friend function.** The statement **-x1** invokes the **operator ()** function. The object **x1** is created of class **UnaryFriend.** The object itself acts as a source and destination object. This can be accomplished by sending reference of an object. The object **x1** is a reference of object **x.** The values of object **x1** are replaced by itself by applying negation.