



LAB 11 MANUAL

Course Code: CL-1004	Course: Object Oriented Programming Lab
Instructor(s):	Shazia Paras Shaikh

Contents:

- Friend Functions
- Friend Classes
- Operator Overloading
- Examples

Working with Friend Functions

Data hiding is a fundamental concept of object-oriented programming. It restricts the access of private members from outside of the class.

However, there is a feature in C++ called friend functions that break this rule and allow us to access member functions from outside the class.

They are defined globally outside the class scope. Friend functions are not member functions of the class. So, what exactly is the friend function?

A friend function in C++ is a function that is declared inside a class but is capable of accessing the private and protected members of the class. There could be situations in programming wherein we want two classes to share their members. These members may be data members, class functions or function templates. In such cases, we make the desired function, a friend to both these classes which will allow accessing private and protected data of members of the class.

Generally, non-member functions cannot access the private members of a particular class. Once declared as a friend function, the function is able to access the private and the protected members of these classes.

Why do we need friend functions?

As discussed, we require friend functions whenever we have to access the private or protected members of a class. This is only the case when we do not want to use the objects of that class to access these private or protected members.

To understand this better, let us consider two classes: Tokyo and Rio. We might require a function, metro(), to access both these classes without any restrictions. Without the friend function, we will require the object of these classes to access all the members. Friend functions in c++ help us avoid the scenario where the function has to be a member of either of these classes for access.

In special cases **when a class's private data needs to be accessed directly without using objects of that class, we need friend functions.** For instance, let's consider two classes: Director and Doctor. We may want the function gross_salary to operate the objects of both these classes. The function does not need to be a member of either of the classes.

Characteristics of Friend Function in C++

- The function is not in the 'scope' of the class to which it has been declared a friend.
- Friend functionality is not restricted to only one class
- Friend functions can be a member of a class or a function that is declared outside the scope of class.
- It cannot be invoked using the object as it is not in the scope of that class. • We can invoke it like any normal function of the class.
- **Friend functions have objects as arguments.**
- **It cannot access the member names directly and has to use dot membership operator and use an object name with the member's name.** • We can declare it either in the 'public' or the 'private' part.

Syntax of friend functions:

To make a function that is declared outside the class “friendly” to that class, we have to declare the function as a friend function, as seen below:

```
class className{  
    // Other Declarations  
    friend returnType functionName(arg list);  
};
```

As we can see above, the friend function should be declared inside the class whose private and protected members are to be accessed.

Let's breakdown the syntax:

friend is a keyword to denote that this function is a friend

function. returnType is the function's return type.

functionName is the name of the function being made a friend of the

class. arg list is the arguments that are passed.

The friend function definition is found outside the class like a normal member function. The friend function is not defined using the friend keyword or use the scope resolution operator: as it is not a member of the class in which it has been declared. A friend function can be declared in several classes.

Ways to Implementing Friend Functions

Friend Functions can be implemented in two ways:

A method of another class:

We declare a friend function when we want to access the non-public data members of a particular class from another class or function. This method involves declaring the friend function within the class definition.

A global function:

A 'global friend function' allows you to access all the private and protected members of the class declaration. This method involves declaring the friend function outside the class definition, but still declaring it as a friend of the class.

Example 1: A simple example of a C++ friend

```
function #include <iostream>

using namespace std;

class Temperature
{
    int celsius;
public:
    Temperature()
    {
        celsius = 0;
    }
    friend int temp( Temperature ); // declaring friend function
};

int temp( Temperature t ) // friend function definition
{
    t.celsius = 40;
    return t.celsius;
}

int main()
{
    Temperature tm;
    cout << "Temperature in celsius : " << temp( tm ) << endl;
    return 0;
}
```

Here we declared a function 'temp' as the friend function of the class 'Temperature'. In the friend function, we directly accessed the private member Celsius of the class 'Temperature'.

When the first statement of the main function created an object 'tm' of the class 'Temperature' thus calling its constructor and assigning a value 0 to its data member Celsius.

The second statement of the main function called the function 'temp' which assigned a value 40 to Celsius.

Example 2: the same friend function of two classes

```
#include <iostream>

using namespace std;

class B; //declaration of class B

class A
{
    int value;
    public:
        A()
        {
            value = 5;
        }
        friend int sum(A, B); // declaring friend function
};

class B
{
    int value;
    public:
        B()
        {
            value = 3;
        }
        friend int sum(A, B); // declaring friend function
};

int sum( A v1, B v2 ) // friend function definition
{
    return (v1.value + v2.value);
}

int main()
{
    A a;
    B b;
    cout << "Sum : " << sum( a, b ) << endl;
    return 0;
}
```

In this example, we declared `sum()` as a friend function of the classes A and B. So, this function can now access the private and protected members of both these classes. The objects of both the classes are passed into as an argument to the function.

First, we created an object for both the classes, thus calling their respective constructors and initializing the values of their respective data member value to 5 and 3. The function 'sum' then returned the sum of the data members of the two classes by calling the data members of the two classes by their respective objects.

Note that we declared class B before defining class A because in the body of class A, the friend function takes the parameters 'A' and 'B' (forward declaration).

Normal friend function :

```
class MyClass {
private:
    int privateData;

public:
    MyClass(int data) : privateData(data) {}

    // Declare a normal friend function
    friend void myFriendFunction(MyClass obj);
};

// Define the friend function outside of the class
void myFriendFunction(MyClass obj) {
    // Access the private data of MyClass through the friend function
    std::cout << "Private data of MyClass: " << obj.privateData << std::endl;
}

int main() {
    MyClass obj(5);
    myFriendFunction(obj); // Call the friend function
    return 0;
}
```

Global friend function:

```
class MyClass {
private:
    int privateData;

public:
    MyClass(int data) : privateData(data) {}

    // Declare a global friend function
    friend void myGlobalFriendFunction(MyClass obj);
};

// Define the global friend function outside of any class
void myGlobalFriendFunction(MyClass obj) {
    // Access the private data of MyClass through the global friend function
    std::cout << "Private data of MyClass: " << obj.privateData << std::endl;
}
```

```
int main() {
    MyClass obj(5);
    myGlobalFriendFunction(obj); // Call the global friend function
    return 0;
}
```

Friend Class

A friend class can have access to the data members and functions of another class in which it is declared as a friend. They are used in situations where we want a certain class to have access to another class's private and protected members.

Classes declared as friends to any another class will have all the member functions become friend functions to the friend class. Friend functions are used to work as a link between the classes.

When a class is declared a friend class, all the member functions of the friend class become friend functions.

```
class ClassB;

class ClassA {
    // ClassB is a friend class of ClassA
    friend class ClassB;
    ... ..
}

class ClassB {
    ... ..
}
```

Since ClassB is a friend class, we can access all members of ClassA from inside ClassB. }

However, we cannot access members of ClassB from inside ClassA. It is because friend relation in C++ is only granted, not taken.

To make you understand in detail:

- If class A is a friend of class B, then class B is not a friend of class A. • Also, if class A is a friend of class B, and then class B is a friend of class C, class A is not a friend of class C.
- If Base class is a friend of class X, subclass Derived is not a friend of class X; and if class X is a friend of class Base, class X is not a friend of subclass Derived

Example of friend class:

```
using namespace std;

// forward declaration
class ClassY;

class ClassX {
    int digit1;

    // friend class declaration
    friend class ClassY;

public:
    // constructor to initialize num1 to 10
    ClassX() : digit1(10) {}
};

class ClassY {
    int digit2;

public:
    // constructor to initialize num2 to 5
    ClassY() : digit2(5) {}

    // member function to multiply num1
    // from ClassX with num2 from ClassY
    int multiply() {
        ClassX m;
        return m.digit1 * digit2;
    }
};

int main() {
    ClassY n;
    cout << "Multiplication: " << n.multiply();
    return 0;
}
```

The output will be: Multiplication: 50

In the program above, we have declared two classes: X and Y. ClassY is a friend class of ClassX. Therefore, ClassY has access to the member function of ClassX. In ClassY, we have created a function multiply() that returns the multiplication of digit1 and digit2.

ClassY being a friend class enables us to create objects of ClassX inside of ClassY. This is possible through forward declaration of the ClassY.

Class Members as Friend

We can also make a function of one class as a friend of another class. We do this in

```
the class A; // forward declaration of A needed by B
```

```
class B
{
    display( A a ); //only specified. Body is not declared
};
```

```
class A
{
    friend void B::display( A );
};
```

```
void B::display(A a) //declaration here
{
}
class ClassB {
    ... ..
}
```

same way as we make a function as a friend of a class. The only difference is that we need to write `class_name ::` in the declaration before the name of that function in the class whose friend it is being declared. The friend function is only specified in the class and its entire body is declared outside the class. It will be clear from the example given below.

```
class A; // forward declaration of A needed by B
```

```
class B
{
    display( A a ); //only specified. Body is not declared
};
```

```
class A
{
    friend void B::display( A );
};
```

```
void B::display(A a) //declaration here
{
```

```

    }
8      ClassB {
class    ... ..

```

CL-1004 - Object Oriented Programming Lab Lab Manual - 10

```

void B::display(A obj)
{
    cout << obj.x << endl;
}

```

```

int main()
{
    A a;
    B b;
    b.display(a);
    return 0;
}

```

Advantages of friend function in C++

- **Friend function in c++** provide a degree of freedom in the interface design option
- A friend function is used to access all the non-public members of a class. • You can use a friend function to bridge two classes by operating objects of two different classes.
- It increases the versatility of **overloading operators**.
- It enhances encapsulation. Only the programmer who has access to the class's source code can make a function friend to that class.
- You may declare a member function of a class as a friend of another class.
- It works symmetrically with all its friends.

C++ Function Overloading Using Friend Function

The property of function overloading allows two or more functions in C++ to have the same names, but they should have different signatures. This means there should be a difference in those functions in terms of parameters (and)or return types. Such functions are called Overloaded Functions.

The following is the sample code

```

class Complex
{
private:
int real;
int img;
public:
Complex (int r = 0, int i = 0)
{
real = r;
img = i;
}
Complex add (Complex x)
{
Complex temp;
temp.real = real + x.real;
temp.img = img + x.img;
return temp;
}
void Display()
{
cout << real << "+i" << img << endl;
}
};
int main()
{
Complex C1 (3, 7);
C1.Display();
Complex C2 (5, 2);
C2.Display();
Complex C3;
C3 = C1.add (C2); // C2.add(C1);
C3.Display();
}

```

Suppose we want to add two complex numbers i.e. C1 and C2,

C3 = C1 + C2;

We have created add function in class Complex. At that time the idea was that either C1 will add to C2 or C2 will add to C1. But now we want somebody else to add two complex numbers.

We have also given you an example that if 'X' has some money and 'Y' also has some money and they wanted to add their money. So 'X' can add money or 'Y' can add money or they can also take help from another person i.e. friends. If their friend is adding the money then they both have to give their money to him as the parameter. Then only their friend can add the money. So, the same approach will follow for the friend function. Here we are writing the Complex class,

```
class Complex{
private:
int real;
int img;
public:
friend Complex operator + (Complex C1, Complex C2);
};
```

In the above example, we have created two integer type private data members real and img. Then we overloaded the + operator with two parameters C1 and C2. We have not defined the body here. We have made it a friend by using the friend function. This is the prototype of the friend function in C++. This function will return an object of type Complex. So, this friend function will take two complex numbers as parameters and return a Complex number.

C3 = C1 + C2;

It is just like there is a function that will take C1 and C2 as parameters and add them and return the result. So, neither C1 nor C2 adding but someone else is adding. This friend function has to be written outside the class without using scope resolution. Now let us write the body of the friend function 'operator +' outside the class,

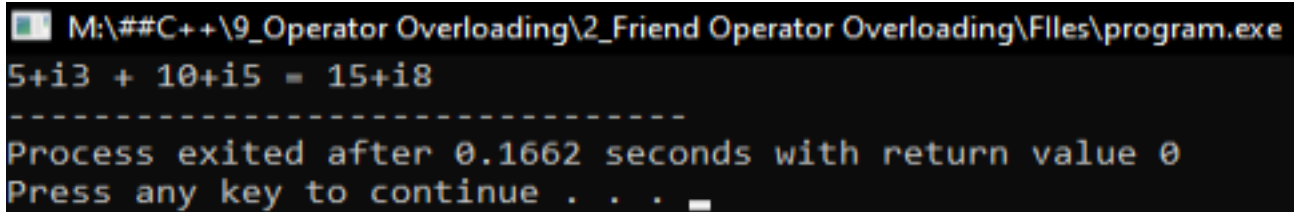
```
Complex operator + (Complex C1, Complex C2){
Complex t;
t.real = C1.real + C2.real;
t.img = C1.img + C2.img;
return t;
}
```

This function doesn't belong to the class but it is a friend of the Complex class. So, we don't use any scope resolution operator. So, this is another approach to overloading operators in C++. So, operators, we can overload as a member function as well as we can overload them as friend functions. Now let us write the complete program in C++.

Example to Understand Operator Overloading in C++ Using Friend Function

```
#include <iostream>
using namespace std;
class Complex
{
private:
int real;
int img;
public:
Complex (int r = 0, int i = 0)
{
real = r;
img = i;
}
void Display ()
{
cout << real << "+" << img;
}
friend Complex operator + (Complex c1, Complex c2);
};
Complex operator + (Complex c1, Complex c2)
{
Complex temp;
temp.real = c1.real + c2.real;
temp.img = c1.img + c2.img;
return temp;
}
int main ()
{
Complex C1(5, 3), C2(10, 5), C3;
C1.Display();
cout << " + ";
C2.Display();
cout << " = ";
C3 = C1 + C2;
C3.Display();
}
```

Output:



```

M:\##C++\9_Operator Overloading\2_Friend Operator Overloading\Files\program.exe
5+i3 + 10+i5 = 15+i8
-----
Process exited after 0.1662 seconds with return value 0
Press any key to continue . . . _

```

Overloading Unary Operator using a Friend function: Using Friend

Function to Overload Unary Operator in C++:

We can also overload a unary operator in C++ by using a friend function. The overloaded ++ operator relative to the Test class using a member function is shown in the below example.

```

#include <iostream>
using namespace std;
class Test
{
    int a, b, c;
public:
    Test()
    {
        a = b = c = 0;
    }
    Test(int i, int j, int k)
    {
        a = i;
        b = j;
        c = k;
    }
    // use a reference to overload the ++
    friend Test operator ++ (Test & op1);
    friend Test operator ++ (Test & op1, int not_used);
    void Display();
};

/* Overload prefix ++ using a friend function.
   This requires the use of a reference parameter. */
Test operator ++(Test & op1)
{
    op1.a++;
    op1.b++;
    op1.c++;
    return op1;
}

```

```

Test operator ++ (Test & op1, int not_used)
{
    Test temp = op1;
    op1.a ++;
    op1.b ++;
    op1.c ++;
    return temp;
}
// Display a, b, c coordinates.
void Test::Display()
{
    cout << a << ", ";
    cout << b << ", ";
    cout << c << "\n";
}
int main()
{
    Test a (12, 22, 33);
    a.Display();
    ++a; // prefix increment
    a.Display();
    a++; // postfix increment
    a.Display();
    return 0;
}

```

Points to Remember While Overloading Operator using Friend Function:

We need to remember the following pointers while working with Operator Overloading in C++ Using Friend Function.

- The Friend function in C++ using operator overloading offers better flexibility to the class.
- The Friend functions are not a member of the class and hence they do not have 'this' pointer.
- When we overload a unary operator, we need to pass one argument. • When we overload a binary operator, we need to pass two arguments. • The friend function in C++ can access the private data members of a class directly.
- An overloaded operator friend could be declared in either the private or public section of a class.

When redefining the meaning of an operator by operator overloading the friend function, we cannot change its basic meaning. For example, we cannot redefine minus operator + to multiply two operands of a user-defined data type.

Tasks

Question 1:

Write a C++ program that simulates a TV scenario using a friend class connecting multiple classes.

The program has two classes, TV and Remote.

The TV class has three private member variables: model, isOn, and volume. model stores the model name of the TV as a string, isOn is a boolean variable that indicates whether the TV is on or off, and volume stores the current volume level of the TV.

The TV class has a constructor that takes a string argument for the model variable and initializes isOn to false and volume to 0. The class also has a display() function that prints out the current status of the TV, including its model name, whether it is on or off, and the current volume level.

The Remote class has a TV object as a private member variable, which allows it to access the TV class's private member variables using the friend function mechanism.

The Remote class has a constructor that takes a TV object as an argument and initializes the tv variable to that object. The class has three member functions, togglePower(), increaseVolume(), and decreaseVolume(), that correspond to the power button, volume up button, and volume down button on a remote control, respectively.

The togglePower() function toggles the value of isOn, so the TV switches between on and off. It then prints a message indicating the current status of the TV.

The increaseVolume() function increases the value of volume by one if the current volume is less than 10. It then prints a message indicating the new volume level.

The decreaseVolume() function decreases the value of volume by one if the current volume is greater than 0. It then prints a message indicating the new volume level.

The Remote class also has a display() function that calls the display() function of the TV object to print the current status of the TV.

Finally, in the main() function, the program creates a TV object with the model name "Samsung" and a Remote object that takes the TV object as an argument. The program then calls various functions of the Remote object to simulate pressing buttons on a remote control, such as toggling the power on, increasing the volume, and decreasing the volume. The program then calls the display() function of the Remote object to print the current status of the TV.

Question 2:

Write a c++ program in which two classes, Document and Printer. The Document class has a private member variable called content, which is a string that holds the text of the document. The class has a constructor that takes a string argument, which is used to initialize the content member.

The Printer class has a private member variable called model, which is a string that holds the name of the printer model. The class has a constructor that takes a string argument, which is used to initialize the model member.

The code then defines a friend function called printDocument, which takes a Document object and a Printer object as arguments. This function is declared as a friend of both the Document and Printer classes, which means it has access to the private members of both classes.

Inside the printDocument function, the model of the printer and the content of the document are output to the console.

In main, a Printer object is created with the name "Epson", and a Document object is created with the content "This is a test document." The printDocument function is then called with the Printer and Document objects as arguments, which prints the content of the document to the console using the specified printer.

Question 3:

Write a c++ program in which there are two classes: OperatingSystem and Process. The OperatingSystem class represents an operating system and has a name and a version attribute. The Process class represents a running process and has a name, an id, and a cpu usage attribute.

Declare two friend functions: display(OperatingSystem os) and display(Process p). These functions are declared as friends of both classes.

The display(OperatingSystem os) function takes an OperatingSystem object as an argument and displays the name and version of the operating system.

The display(Process p) function takes a Process object as an argument and displays the name, id, and CPU usage of the process.

In main(), we create an OperatingSystem object with the name "Windows" and the version number 10. We also create a Process object with the name "chrome.exe", the ID 1234, and a CPU usage of 50%.

Call the display() function with both the OperatingSystem and Process objects as arguments. Because display() is a friend function of both classes, it has access to their private members and can display their attributes.

Question 4:

Car class has three private member variables - make, model, and year. The equality operator == is overloaded as a friend function of the Car class. The overloaded operator== function takes two const Car references as arguments and returns a boolean value indicating whether the two Car objects have the same make, model, and year values. In the main function, we create three Car objects and compare them using the overloaded == operator. The output of the program is:

Car 1 is not equal to Car 2.

Car 1 is equal to Car 3.

Question 5:

Write a c++ Program in which there are three classes. The Mobile class has a model attribute and an isCharging attribute, and it also has a friend class Display and a friend function Charger::charge().

The Charger class has a charge() method that takes a Mobile object by reference and sets its isCharging attribute to true. This method is declared as a friend function of the Mobile class.

The Mobile class also has a display() method that displays the model and charging status of the mobile, and a friend class Display that can access the model attribute of the Mobile class.

In the main() function, create a Mobile object m, a Charger object c, and a Display object d, call m.display() to display the initial status of the mobile, call d.showModel(m) to display the model of the mobile using the friend class Display, and call c.charge(m) to charge the mobile using the friend function Charger::charge(). Finally, call m.display() again to display the updated charging status of the mobile.