The Need for Friend Function:

when a data is declared as private inside a class, then it is not accessible from outside the class. A function that is not a member or an external class will not be able to access the private data. A programmer may have a situation where he or she would need to access private data from non-member functions and external classes. For handling such cases, the concept of Friend functions is a useful tool.

### What is a Friend Function?

A friend function is used for accessing the non-public members of a class. A class can allow non-member functions and other classes to access its own private data, by making them friends. Thus, a friend function is an ordinary function or a member of another class.

### How to define and use Friend Function in C++:

The friend function is written as any other normal function, except the function declaration of these functions is preceded with the [keyword](http://www.exforsys.com/tutorials/c-plus-plus/c-plus-plus-friend-functions.html) friend. The friend function must have the class to which it is declared as friend passed to it in argument.

Some important points to note while using friend functions in C++:

* The keyword friend is placed only in the function declaration of the friend function and not in the function definition.
* It is possible to declare a function as friend in any number of classes.
* When a class is declared as a friend, the friend class has access to the private data of the class that made this a friend.
* A friend function, even though it is not a member function, would have the rights to access the private members of the class.
* It is possible to declare the friend function as either private or public.
* The function can be invoked without the use of an object. The friend function has its argument as objects, seen in example below.

#include <iostream>

class xyz

{

private:

        int a,b;

public:

        void xyz()

        {

                a=100;

                b=200;

        }

        friend int compute(xyz obj);

                *//Friend Function Declaration with keyword friend and with the object of class xyz to which it is friend passed to it*

};

int compute(xyz obj)

{

        *//Friend Function Definition which has access to private data*

        return int(obj.a+obj.b)-5;

}

void main()

{

        xyz obj;

        obj.test();

        cout << "The result is:" << compute(obj);

               *//Calling of Friend Function with object as argument.*

}

The function compute() is a non-member function of the class xyz. In order to make this function have access to the private data a and b of class xyz , it is created as a friend function for the class xyz. As a first step, the function compute() is declared as friend in the class xyz as:

friend int compute (xyz obj)

The keyword friend is placed before the function. The function definition is written as a normal function and thus, the function has access to the private data a and b of the class xyz. It is declared as friend inside the class, the private data values a and b are added, 5 is subtracted from the result, giving 295 as the result. This is returned by the function and thus the output is displayed as shown above.

**Friend functions**

In principle, private and protected members of a class cannot be accessed from outside the same class in which

they are declared. However, this rule does not affect *friends*.

Friends are functions or classes declared as such.

If we want to declare an external function as friend of a class, thus allowing this function to have access to the

private and protected members of this class, we do it by declaring a prototype of this external function within the

class, and preceding it with the keyword friend:

// friend functions

#include <iostream>

using namespace std;

class CRectangle {

int width, height;

public:

void set\_values (int, int);

int area () {return (width \* height);}

friend CRectangle duplicate (CRectangle);

};

void CRectangle::set\_values (int a, int b) {

width = a;

height = b;

}

CRectangle duplicate (CRectangle rectparam)

{

CRectangle rectres;

rectres.width = rectparam.width\*2;

rectres.height = rectparam.height\*2;

return (rectres);

}

int main () {

CRectangle rect, rectb;

rect.set\_values (2,3);

rectb = duplicate (rect);

cout << rectb.area();

return 0;

}

Output-

24

The duplicate function is a friend of CRectangle. From within that function we have been able to access the

members width and height of different objects of type CRectangle, which are private members. Notice that

neither in the declaration of duplicate() nor in its later use in main() have we considered duplicate a member

of class CRectangle. It isn't! It simply has access to its private and protected members without being a member.

The friend functions can serve, for example, to conduct operations between two different classes. Generally, the

use of friend functions is out of an object-oriented programming methodology, so whenever possible it is better to

use members of the same class to perform operations with them. Such as in the previous example, it would have

been shorter to integrate duplicate() within the class CRectangle.

**Friend classes**

Just as we have the possibility to define a friend function, we can also define a class as friend of another one,

granting that first class access to the protected and private members of the second one.

// friend class

#include <iostream>

using namespace std;

class CSquare;

class CRectangle {

int width, height;

public:

int area ()

{return (width \* height);}

void convert (CSquare a);

};

class CSquare {

private:

int side;

public:

void set\_side (int a)

{side=a;}

friend class CRectangle;

};

void CRectangle::convert (CSquare a) {

width = a.side;

height = a.side;

}

int main () {

CSquare sqr;

CRectangle rect;

sqr.set\_side(4);

rect.convert(sqr);

cout << rect.area();

return 0;

}

Out put-

16

In this example, we have declared CRectangle as a friend of CSquare so that CRectangle member functions could

have access to the protected and private members of CSquare, more concretely to CSquare::side, which describes

the side width of the square.

You may also see something new at the beginning of the program: an empty declaration of class CSquare. This is

necessary because within the declaration of CRectangle we refer to CSquare (as a parameter in convert()). The

definition of CSquare is included later, so if we did not include a previous empty declaration for CSquare this class

would not be visible from within the definition of CRectangle.

Consider that friendships are not corresponded if we do not explicitly specify so. In our example, CRectangle is

considered as a friend class by CSquare, but CRectangle does not consider CSquare to be a friend, so CRectangle

can access the protected and private members of CSquare but not the reverse way. Of course, we could have

declared also CSquare as friend of CRectangle if we wanted to.

Another property of friendships is that they are *not transitive*: The friend of a friend is not considered to be a friend

unless explicitly specified.