

## 66. Friends

### Objectives

- Make a function a friend of a class
- Make a method a friend of a class
- Make a class a friend of a class

## Friend function

You can include the prototype of a function inside the class declaration to make it a **friend**.

The friend function will then have direct access to private members of the class. This is done usually for **efficiency**.

You can put the friend function in the private or the public section – no difference.

Consider the following example:

```
class C
{
public:
    C(int x):
        x_(x)
    {}

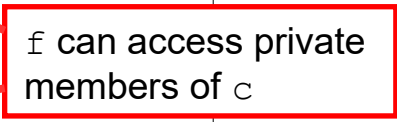
    friend void f(C c);

private:
    int x_;
};

void f(C c)
{
    std::cout << c.x_ << '\n';
}

int main()
{
    C c(23);
    f(c);

    return 0;
}
```



f can access private members of c

**Exercise.** Look at your Date class. Make

`operator<<(std::ostream &, const Date &)` a friend of Date and rewrite it such that it accesses the private members of Date directly.

In general, friend functions break information hiding since the intention is for the friend function to directly access private member variables or directly call private member functions. Therefore you should **limit** the use of friend.

It's OK if the function is a function that is considered an integral part of

the class you are writing. But you definitely do NOT want to make a function a friend if the function is not part of the class. For instance if one of your team mates (in a software project) says “Hey, can you please make this function I’m writing a friend of your class?” you should probably say “No!”

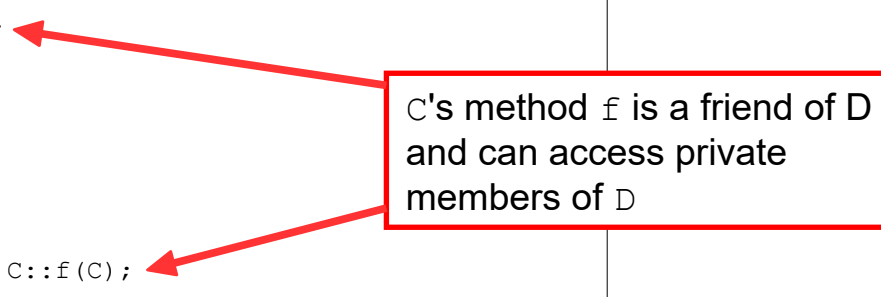
## Friend member function

You can also make a member function of a different class, i.e., a method of a different class, a friend of the current class.

For example:

```
class C
{
public:
    ...
    void f(C c);
    ...
};

class D
{
public:
    ...
    friend void C::f(C);
    ...
};
```



C's method `f` is a friend of D and can access private members of D

## Friend classes

You can make **all** the methods of a class friends to another class. The class with the methods is then called a friend class to the other class.

Make one class a friend of another only when they create objects which are conceptually related.

Example:

```
class D
{
public:
    D(int y)
        : y_(y)
    {}

    friend class C;

private:
    int y_;
};

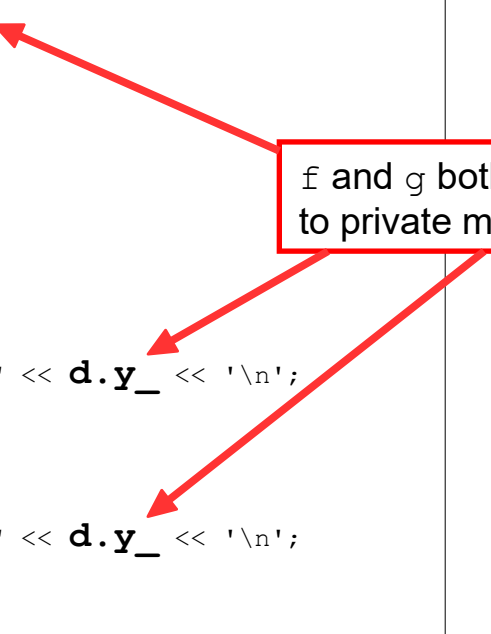
class C
{
public:
    void f(D d)
    {
        std::cout << "f: " << d.y_ << '\n';
    }

    void g(D d)
    {
        std::cout << "g: " << d.y_ << '\n';
    }
};

int main()
{
    D d(42);
    C c;
    c.f(d);
    c.g(d);

    return 0;
}
```

f and g both have access  
to private members of D



You can also make classes **friends of each other**. If you want classes to be friends of each other, you need to declare the classes first, and **then** implement the classes.

```

#include <iostream>

class C;

class D
{
public:
    ...
    void f(C);
    friend class C;
    ...
private:
    int x_;
};

class C
{
public:
    ...
    void f(D);
    friend class D;
    ...
private:
    int x_;
};

void D::f(C c)
{
    std::cout << c.x_ << '\n';
}

void C::f(D d)
{
    std::cout << d.x_ << '\n';
}

int main()
{
    C c;
    D d;
    c.f(d);
    d.f(c);

    return 0;
}

```

... **or** ... make sure  
 “friend class C;” is  
**before** any mention of C:

```

...
public:
    friend class C;
    void f(C);
...

```

**Beware of the following gotcha:**

```

#include <iostream>

class C;

class D
{

```

```

public:
    ...
    void f(C);
    friend class C;
    ...
private:
    int x_;
};

void D::f(C c)
{
    std::cout << c.x << '\n';
}

class C
{
public:
    ...
    void f(D);
    friend class D;
    ...
private:
    int x_;
};

void C::f(D d)
{
    std::cout << d.x_ << '\n';
}

int main()
{
    C c;
    D d;
    c.f(d);
    d.f(c);

    return 0;
}

```

**ERROR: Implementation of `D::f` requires implementation of `C` objects. So put definition of `f` **after** class `C{...}`;**

The same advice goes for friend classes: Don't have too many friend classes. It's OK, you the two classes work hand in hand and are meant to be used together, and especially if you write both of the together.

An example would be a matrix class and a vector class (for those of you who have heard of these terms before). Matrices and vectors for used in physics, engineering, computer graphics, ..., you name it. And they are always used together.

A car class and a car engine class can probably be friends too.