8.13 — Friend functions and classes

BY ALEX ON SEPTEMBER 20TH, 2007 | LAST MODIFIED BY ALEX ON MARCH 14TH, 2016

For much of this chapter, we've been preaching the virtues of keeping your data private. However, you may occasionally find situations where you will find you have classes and functions outside of those classes that need to work very closely together. For example, you might have a class that stores data, and a function (or another class) that displays the data on the screen. Although the storage class and display code have been separated for easier maintenance, the display code is really intimately tied to the details of the storage class. Consequently, there isn't much to gain by hiding the storage classes details from the display code.

In situations like this, there are two options:
1) Have the display code use the publicly exposed functions of the storage class. However, this has several potential downsides. First, these public member functions have to be defined, which takes time, and can clutter up the interface of the storage class. Second, the storage class may have to expose functions for the display code that it doesn't really want accessible to anybody else. There is no way to say "this function is meant to be used by the display class only".

2) Alternatively, using friend classes and friend functions, you can give your display code access to the private details of the storage class. This lets the display code directly access all the private members and functions of the storage class, while keeping everyone else out! In this lesson, we'll take a closer look at how this is done.

Friend functions

A **friend function** is a function that can access the private members of a class as though it were a member of that class. In all other regards, the friend function is just like a normal function. A friend function may be either a normal function, or a member function of another class. To declare a friend function, simply use the *friend* keyword in front of the prototype of the function you wish to be a friend of the class. It does not matter whether you declare the friend function in the private or public section of the class.

Here's an example of using a friend function:

```
class Accumulator
{
private:
    int m_value;
public:
    Accumulator() { m_value = 0; }
    void add(int value) { m_value += value; }

    // Make the reset() function a friend of this class
    friend void reset(Accumulator &accumulator);
};

// reset() is now a friend of the Accumulator class
void reset(Accumulator &accumulator)
{
    // And can access the private data of Accumulator objects
    accumulator.m_value = 0;
}
```

In this example, we've declared a function named reset() that takes an object of class Accumulator, and sets the value of m_value to 0. Because reset() is not a member of the Accumulator class, normally reset() would not be able to access the private members of Accumulator. However, because Accumulator has specifically declared this reset() function to be a friend of the class, the reset() function is given access to the private members of Accumulator.

Note that we have to pass an Accumulator object to reset(). This is because reset() is not a member function. It does not have a *this pointer, nor does it have an Accumulator object to work with, unless given one.

Here's another example:

```
class Value
{
private:
    int m_value;
public:
    Value(int value) { m_value = value; }
    friend bool isEqual(const Value &value1, const Value &value2);
};

bool isEqual(const Value &value1, const Value &value2)
{
    return (value1.m_value == value2.m_value);
}
```

In this example, we declare the isEqual() function to be a friend of the Value class. isEqual() takes two Value objects as parameters. Because isEqual() is a friend of the Value class, it can access the private members of all Value objects. In this case, it uses that access to do a comparison on the two objects, and returns true if they are equal.

While both of the above examples are fairly contrived, the latter example is very similar to cases we'll encounter in chapter 9 when we discuss operator overloading!

Multiple friends

A function can be a friend of more than one class at the same time. For example, consider the following example:

```
class Humidity;

class Temperature
{
private:
    int m_temp;
public:
    Temperature(int temp=0) { m_temp = temp; }

    void setTemperature(int temp) { m_temp = temp; }

    friend void printWeather(const Temperature &temperature, const Humidity &humidity);
};

class Humidity
{
private:
    int m_humidity;
public:
    Humidity(int humidity=0) { m_humidity = humidity; }

    void setHumidity(int humidity) { m_humidity = humidity; }

    friend void printWeather(const Temperature &temperature, const Humidity &humidity);
};

void printWeather(const Temperature &temperature, const Humidity &humidity)
{
    std::cout << "The temperature is " << temperature.m_temp <<
       " and the humidity is " << humidity.m_humidity << '\n';
}

int main()
{
    Humidity hum(10);
    Temperature temp(12);

    printWeather(temp, hum);

    return 0;
}
```

There are two things worth noting about this example. First, because PrintWeather is a friend of both classes, it can access the private data from objects of both classes. Second, note the following line at the top of the example:

```
class Humidity;
```

This is a class prototype that tells the compiler that we are going to define a class called Humidity in the future. Without this line, the compiler would tell us it doesn't know what a Humidity is when parsing the prototype for PrintWeather() inside the Temperature class. Class prototypes serve the same role as function prototypes -- they tell the compiler what something looks like so it can be used now and defined later. However, unlike functions, classes have no return types or parameters, so class prototypes are always simply `class ClassName`, where ClassName is the name of the class.

Friend classes

It is also possible to make an entire class a friend of another class. This gives all of the members of the friend class access to the private members of the other class. Here is an example:

```
class Storage
{
private:
    int m_nValue;
    double m_dValue;
public:
    Storage(int nValue, double dValue)
    {
        m_nValue = nValue;
        m_dValue = dValue;
    }

    // Make the Display class a friend of Storage
    friend class Display;
};

class Display
{
private:
    bool m_displayIntFirst;

public:
    Display(bool displayIntFirst) { m_displayIntFirst = displayIntFirst; }

    void displayItem(Storage &storage)
    {
        if (m_displayIntFirst)
            std::cout << storage.m_nValue << " " << storage.m_dValue << '\n';
        else // display double first
            std::cout << storage.m_dValue << " " << storage.m_nValue << '\n';
    }
};

int main()
{
    Storage storage(5, 6.7);
    Display display(false);

    display.displayItem(storage);

    return 0;
}
```

Because the Display class is a friend of Storage, any of Display's members that use a Storage class object can access the private members of Storage directly. This program produces the following result:

```
6.7 5
```

A few additional notes on friend classes. First, even though Display is a friend of Storage, Display has no direct access to the *this pointer of Storage objects. Second, just because Display is a friend of Storage, that does not mean Storage is also a friend of Display. If you want two classes to be friends of each other, both must declare the other as a friend. Finally, if class A is a friend of B, and B is a friend of C, that does not mean A is a friend of C.

Be careful when using friend functions and classes, because it allows the friend function or class to violate encapsulation. If the details of the class change, the details of the friend will also be forced to change. Consequently, limit your use of friend functions and classes to a minimum.

Friend member functions

Instead of making an entire class a friend, you can make a single member function a friend. This is done similarly to making a normal function a friend, except using the name of the member function with the className:: prefix included (e.g. Display::displayItem).

However, in actuality, this can be a little trickier than expected. Let's convert the previous example to make Display::displayItem a friend member function. You might try something like this:

```
class Display; // forward declaration for class Display

class Storage
{
private:
    int m_nValue;
    double m_dValue;
public:
    Storage(int nValue, double dValue)
    {
        m_nValue = nValue;
        m_dValue = dValue;
    }

    // Make the Display class a friend of Storage
    friend void Display::displayItem(Storage& storage); // error: Storage hasn't see the full
declaration of class Display
};

class Display
{
private:
    bool m_displayIntFirst;

public:
    Display(bool displayIntFirst) { m_displayIntFirst = displayIntFirst; }

    void displayItem(Storage &storage)
    {
        if (m_displayIntFirst)
            std::cout << storage.m_nValue << " " << storage.m_dValue << '\n';
        else // display double first
            std::cout << storage.m_dValue << " " << storage.m_nValue << '\n';
    }
};
```

However, it turns out this won't work. In order to make a member function a friend, the compiler has to have seen the full declaration for the class of the friend member function (not just a forward declaration). Since class Storage hasn't seen the full declaration for class Display yet, the compiler will error at the point where we try to make the member function a friend.

To resolve this, we can switch the order of class Display and class Storage. We will also need to move the definition of Display::displayItem() out of the Display class declaration, because it needs to have seen the definition of class Storage first.

```
class Storage; // forward declaration for class Storage

```

```
class Display
{
private:
    bool m_displayIntFirst;

public:
    Display(bool displayIntFirst) { m_displayIntFirst = displayIntFirst; }

    void displayItem(Storage &storage); // forward declaration above needed for this declarati
on line
};

class Storage
{
private:
    int m_nValue;
    double m_dValue;
public:
    Storage(int nValue, double dValue)
    {
        m_nValue = nValue;
        m_dValue = dValue;
    }

    // Make the Display class a friend of Storage (requires seeing the full declaration of cla
ss Display, as above)
    friend void Display::displayItem(Storage& storage);
};

// Now we can define Display::displayItem, which needs to have seen the full declaration of cl
ass Storage
void Display::displayItem(Storage &storage)
{
    if (m_displayIntFirst)
        std::cout << storage.m_nValue << " " << storage.m_dValue << '\n';
    else // display double first
        std::cout << storage.m_dValue << " " << storage.m_nValue << '\n';
}

int main()
{
    Storage storage(5, 6.7);
    Display display(false);

    display.displayItem(storage);

    return 0;
}
```

Now, this will compile, and Display::displayItem is a friend of class Storage.

However, a better solution would have been to put each class declaration in a separate header file, with the function bodies in corresponding .cpp files. That way, all of the class declarations would have been visible immediately, and no rearranging of classes or functions would have been necessary.

Summary

A friend function or class is a function or class that can access the private members of another class as though it were a member of that class. This allows the friend or class to work intimately with the other class, without making the other class expose its private members (e.g. via access functions).

Friending is uncommonly used when two or more classes need to work together in an intimate way, or much more commonly, when defining overloading operators (which we'll cover in chapter 9).

Note that making a class a friend only requires as forward declaration that the class exists. However, making a specific

member function a friend requires the full declaration for the class of the member function to have been seen first.

Quiz time

1) In geometry, a point is a position in space. We can define a point in 3d-space as the set of coordinates x, y, and z. For example, the Point(2.0, 1.0, 0.0) would be the point at coordinate space x=2.0, y=1.0, and z=0.0.

In physics, a vector is a quantity that has a magnitude (length) and a direction (but no position). We can define a vector in 3d-space as an x, y, and z value representing the direction of the vector along the x, y, and z axis (the length can be derived from these). For example, the Vector(2.0, 0.0, 0.0) would be a vector representing a direction along the positive x-axis (only), with length 2.0.

A Vector can be applied to a Point to move the Point to a new position. This is done by adding the vector's direction to the point's position to yield a new position. For example, Point(2.0, 1.0, 0.0) + Vector(2.0, 0.0, 0.0) would yield the point (4.0, 1.0, 0.0).

Points and Vectors are often used in computer graphics (the point to represent vertices of shape, and vectors represent movement of the shape).

Given the following program:

```
#include <iostream>

class Vector3d
{
private:
    double m_x = 0.0, m_y = 0.0, m_z = 0.0;

public:
    Vector3d(double x = 0.0, double y = 0.0, double z = 0.0)
        : m_x(x), m_y(y), m_z(z)
    {

    }

    void print()
    {
        std::cout << "Vector(" << m_x << " , " << m_y << " , " << m_z << ")\n";
    }
};

class Point3d
{
private:
    double m_x=0.0, m_y=0.0, m_z=0.0;

public:
    Point3d(double x = 0.0, double y = 0.0, double z = 0.0)
        : m_x(x), m_y(y), m_z(z)
    {

    }

    void print()
    {
        std::cout << "Point(" << m_x << " , " << m_y << " , " << m_z << ")\n";
    }

    void moveByVector(Vector3d &v)
    {
        // implement this function as a friend of class Vector3d
    }
};

int main()
```

```
45  {
46      Point3d p(1.0, 2.0, 3.0);
47      Vector3d v(2.0, 2.0, -2.0);
48
49      p.print();
50      p.moveByVector(v);
51      p.print();
52
53      return 0;
54  }
```

1a) Make Point3d a friend class of Vector3d, and implement function Point3d::moveByVector()

Show Solution

1b) Instead of making class Point3d a friend of class Vector3d, make member function Point3d::moveByVector a friend of class Vector3d.

Show Solution

 **8.14 -- Anonymous variables and objects**

 **Index**

 **8.12 -- Static member functions**

Share this:

Email Facebook 21 Twitter Google Pinterest

64 comments to 8.13 — Friend functions and classes

sanjay
November 6, 2007 at 12:32 am · Reply

hey its great to have a website like this
i didn't buy my cp book till my exams even then i stood 3rd in my class…

sanjay

pavuluri
January 29, 2015 at 2:26 am · Reply

Friend functions are most useful in C++, very nice post and for more information about the friend functions **Friend functions and the usage in C++ with the examples**

jagadeesh
January 16, 2008 at 7:44 am · Reply

c++ programm.
member function of class can access private members,than we we need friend function,what is the use of this please iam in confusion

Alex
January 16, 2008 at 7:57 am · Reply

Making a function a friend of some class means that function has access to the classes private member variables (even though that function is not a member of the class). The Humidity/Weather example above shows an example of this: PrintWeather() is using the private members of Humidity and Weather (which it would normally not have access to, because they are private). However, it is allowed to do this because it has been made a friend.

nidhi
October 5, 2010 at 9:07 am · Reply

haha

K D Joshi
September 22, 2008 at 11:05 pm · Reply

Is it compulsory to declare a function(which is required to be a friend function) in that class itself? Suppose there is a class named FUN. It has some private and some public elements. Now I want a function called SOME to be a friend of FUN. So, is it MUST to declare SOME in private or public section of FUN? Or is it okey to declare and define it entirely outside the class with prefix friend.

Alex
September 24, 2008 at 2:05 pm · Reply

A friend declaration has to be inside the class declaration itself.

harshitha
February 27, 2010 at 6:55 pm · Reply

a friend function can access the private data members of many classes at a time

hhh
March 16, 2011 at 2:06 am · Reply

fk

lalit
December 15, 2008 at 9:27 am · Reply

```
#include<iostream>
using namespace std;
class x
{
    int a,b;

    public:
        x()
    {
        a=5;
        b=9;
    }

    friend ostream& operator<<(ostream &, x);
};

ostream& operator<<(ostream &t, x x1)
{
    t<<x1.a;
    t<<x1.b;
    return t;
}
void main()
{
    x x1;
    cout<<x1;
}
<!--formatted-->
```

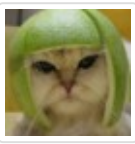
why the errors this above code contain.plz explain .

Alex
December 15, 2008 at 6:12 pm · Reply

Worked fine for me, compiled and ran with MS Visual Studio 2008.

lalit
December 17, 2008 at 1:16 am · Reply

Hello Alix, the above code contain errors like ambigious call to << operator.
a,b are not accessible as these are in private section.
but why it give error.
thxs…



Alex
December 22, 2008 at 10:58 pm · Reply

It should work. Even though a and b are private, operator<< is a friend of class x, and therefore can access x's private member variables directly.

Pathik
January 13, 2009 at 7:21 pm · Reply

Are all the codes you write now object-oriented programming?

Alex
February 5, 2009 at 10:11 pm · Reply

Yup. Once you start using or writing classes, you're doing object-oriented programming. 🙂

tyler
January 22, 2009 at 2:12 pm · Reply

hey Alex;

great lessons, and thx for helping me back with the headers files.

I would like to know how to make my program play music that i have on my computer, and dicplay pictures. when i saw 'Object Oriented Programing' thats what i though it was (useing picures in prgramsect…) gut i see that i was wrong. i would like to learn how to make those types of gams that you see on miniclip. (i know i know, those games are made using flash, but i'm sure you can do simalar thins with C++, like they use C++ for games likie need for speed and the computer version of halo don't they?)

Alex
February 5, 2009 at 10:12 pm · Reply

You will need some kind of add-in library to do this. I would suggest looking into **SDL**, as it has functions to play music, display graphics, etc…

Trevor
January 28, 2009 at 2:40 am · Reply

Heya Alex, I really had better start by thanking you for this excellent site. Thanks.

My question is: is it possible to make only one fuction in a Class A a friend of Class B without making the WHOLE of Class A a friend of Class B. I've constructed a crude example to show why I think it can't be done, perhaps you can tell me if I'm doing something wrong:

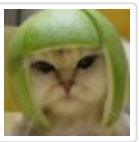
```
1  #include "stdafx.h"
2  #include <iostream>
3
4  class number
5  {
6  private:
7      int no;
8  public:
9      number(int n=0){no = n;}
10     friend class manipulate;
11 };
```

```
12
13  class manipulate
14  {
15  public:
16      int add(number &a,number &b)
17      {
18          return a.no + b.no;
19      }
20      int sub(number &a,number &b)
21      {
22          return a.no - b.no;
23      }
24  };
25
26  int main()
27  {
28      using namespace std;
29
30      number a(4),b(2);
31      manipulate name;
32
33      cout << name.add(a,b) << endl;
34  }
```

The code above works exactly as expected, but if I replace “friend class manipulate” with “friend int add(number &a,number &b)” or “friend int manipulate::add(number &a,number &b)”, I get compiler errors along the lines of “cannot access private member declared in class ‘number’”.

Cheers,

Trevor



Alex
February 8, 2016 at 2:35 pm · Reply

Yes, it is possible, although with a little difficulty.

```
1   class number;
2   class manipulate
3   {
4   public:
5       int add(number &a, number &b);
6       int sub(number &a, number &b);
7   };
8
9   class number
10  {
11  private:
12      int no;
13  public:
14      number(int n = 0) { no = n; }
15      friend int manipulate::add(number&, number&);
16      friend int manipulate::sub(number&, number&);
17  };
18
19  int manipulate::add(number &a, number &b)
20  {
21      return a.no + b.no;
22  }
23  int manipulate::sub(number &a, number &b)
24  {
25      return a.no - b.no;
26  }
27
28  int main()
29  {
```

```
30      using namespace std;
31
32      number a(4), b(2);
33      manipulate name;
34
35      cout << name.add(a, b) << endl;
36  }
```

I've updated the lesson with an example similar to this.

asif
December 27, 2009 at 6:53 am · Reply

```
1   #include <iostream>
2   using namespace std;
3
4
5   class two;
6
7
8
9   class one
10  {
11     private:
12      int data1;
13     public:
14      one()
15      {
16        data1 = 100;
17      }
18     friend int two::accessboth(one a);
19  };
20
21
22
23
24      class two
25  {
26     private:
27      int data2;
28     public:
29      two()
30      {
31        data2 = 200;
32      }
33     int accessboth(one a);
34      };
35
36       int two::accessboth(one a)
37        {
38      return (a.data1 + (*this).data2);
39    }
40
41
42
43  int main()
44  {
45     one a;
46     two b;
47     cout << b.accessboth(a);
48     return 0;
49  }
```

can some one help me with this …..i tried compiling in Gnu G++ environment…and following are the errors…can

someone tell me whats wrong with above code!!!111
friend2.cpp:30: error: invalid use of incomplete type ‘struct two’
friend2.cpp:5: error: forward declaration of ‘struct two’
friend2.cpp: In member function ‘int two::accessboth(one)’:
friend2.cpp:24: error: ‘int one::data1’ is private
friend2.cpp:55: error: within this context

Alex
February 8, 2016 at 2:41 pm · Reply

Class one needs to see the full declaration of class two for a friend member function to work.

You have a couple of options here:
1) Use a friend class instead.
2) Switch the order of the classes and define the functions outside of the classes (after the declarations).

lolo 33
June 5, 2010 at 8:17 pm · Reply

I’m so lucky to find this websit =)

thnx so much Alex ,i have c++ exam now

bye

Aki
November 8, 2012 at 5:49 am · Reply

Normal friend functions can access private members of other classes.
But if the friend function is a member function of another class then it cannot access the other class’s private members.

```
1
2  #include <iostream>
3  class A
4  {
5      int x;
6      public:
7      friend int boo(A &a);
8      A()
9      {
10         x = 5;
11     }
12 };
13
14 //int boo(A &a)
15 //{
16 //  return a.x;
17 //}
18 class B
19 {
20     public:
21     int y;
22     int boo(A &a)
23     {
24         y = a.x;
25         return y;
26     }
27 };
28
29 int main()
```

```
{
    using namespace std;
    A a;
    B b;
    cout << b.boo(a) << endl;
}
<!--formatted-->
It gives the following error

```

friends.cpp: In member function ‘int B::boo(A&)’:
friends.cpp:4:6: error: ‘int A::x’ is private
friends.cpp:23:9: error: within this context

prakash
December 25, 2012 at 5:25 pm · Reply

// This works.

#include
#include

class A; // forward declaration

class B
{
public:
int y;
int boo(A &a) ; // only declare. define later. as friendship has not came in to play yet.

};

class A
{
int x;
public:
friend int B::boo(A &a);
A()
{
x = 5;
}
};

// now we define the function boo which belongs to class B only. just defined outside.
// if using header files and implementation files separately, it makes crystal clear.

int B::boo(A &a){
y = a.x;
return y;
}

int main()
{
using namespace std;
A a;
B b;
cout << b.boo(a) << endl;
}

prakash
December 25, 2012 at 5:30 pm · Reply

Aki,

you can add more functions in class B, same as foo. i.e. declare first and define later.
but keep friendship to foo.
other functions will not have access to private x

-prakash

dinhpq
April 16, 2013 at 8:49 pm · Reply

Hi Alex,
I have confused “A friend function may or may not be a member of another class” It mean member function Reset() of class Value may or may not be a friend class Accumulator, is that true? I have an example:

```
1  #include
2  using namespace std;
3
4  class Accumulator
5  {
6  private:
7      int m_nValue;
8  public:
9      Accumulator() { m_nValue = 0; }
10     void Add(int nValue) { m_nValue += nValue; }
11     friend void Reset(Accumulator &cAccumulator);
12 };
13
14 class Value
15 {
16 private:
17     int m_nValue;
18 public:
19     Value(int nValue) { m_nValue = nValue; }
20     void Reset(Accumulator &cAccumulator);
21 };
22 void Reset(Accumulator &cAccumulator)
23 {
24     cout << "Reset Accumulator" <<endl;
25     cAccumulator.m_nValue = 0;
26 }
27 int main()
28 {
29     Accumulator cAccumulator;
30     Value cValue(10);
31     cValue.Reset(cAccumulator);
32    return 0;
33 }
```

The compiler error: undefined reference to `Value::Reset(Accumulator&)’
Can you explain?

Alex
February 8, 2016 at 2:52 pm · Reply

I mean that a friend function may be either a normal function, or a member function of another class.

In the above code, you’ve declared Reset() as a member of class Value, but you’ve defined Reset to not belong

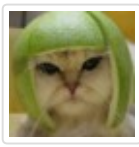
to a class. You will need to fix this, and then you will need to flip the order of the classes around, so class value is declared first (or put it in a header file).

samiullah
July 17, 2014 at 1:38 am · Reply

Thank U so much Alex;

Vaibhav
October 15, 2014 at 6:01 am · Reply

Hi Alex,
Just a question.
Can a friend function be const or volatile.
eg. friend void Reset(Accumulator &cAccumulator) const;
friend volatile void Reset((Accumulator &cAccumulator);



Alex
February 8, 2016 at 2:53 pm · Reply

Yes.

Chienli Ma
February 14, 2015 at 6:31 am · Reply

Hi, this is an awesome website.
I learn C++ here much faster than any other places.

But I just got a little confuse here.
The friend function, in my opinion, is something like a Getter or Setter.
Is there any other usage which makes the friend function so important that it needs to be a feature of Cpp?
Or, what's the difference between a friend function and a (normal function + Getter + Setter)

Thanks 😀
Ma

Danny
December 30, 2015 at 6:56 am · Reply

The getters and setters are usually member functions within a class. The setter initializes the data members, and getter can probably display the values set by the setter. For the friend functions, consider a case where you have all your data encapsulated within a class, but there is also a non-class function that needs to access the data in that class. (I used non-class function to clearly set the difference between friend functions and other normal functions)
How would the non-class function access the data in the class? Making the non-class function a friend to the class by declaring its prototype in the class is the only. Otherwise,trying to access data would give you an error.
I hope this helps.

Enrique
July 7, 2015 at 12:56 am · Reply

this website is a jewell for C++ programers. Thanks a lot !

anvekar
July 16, 2015 at 7:20 am · Reply

```
1  class Accumulator
2  {
3  private:
4      int m_nValue;
5  public:
6      Accumulator() { m_nValue = 0; }
7      void Add(int nValue) { m_nValue += nValue; }
8
9      // Make the Reset() function a friend of this class
10     friend void Reset(Accumulator &cAccumulator); // IS THE ARGUMENT NECESSARY?
11 };
```

Why is it necessary to use an argument in the function prototype inside the class?
As it is just a prototype, only the argument type should be sufficient right?
Example: friend void Reset(Accumulator&); // prototype for the friend function Reset()

Alex
February 8, 2016 at 2:57 pm · Reply

In the above case, the argument is needed because function Reset() is not a member of class Accumulator.

If Reset() were defined as a member of class Accumulator, then making it a friend would not be necessary, as it would already be able to access the private members of class Accumulator!

Quang
September 20, 2015 at 12:59 am · Reply

Hello Alex! I want to thank you for what you have done to the programming community! I have a little bit problem about the friend functions:
- In 8.10 — Const class objects and member functions, you use functions like getDate, getMonth, getYear and i wonder whats the differences between those and friend function (may be getSomething just for the return !?) and what is the proper situation to use each of them?
Thank you for reading this!! Keep up the great work

Alex
September 20, 2015 at 5:30 pm · Reply

Friend functions have direct access to the private data of a class. Non-friend functions have to use public accessor functions (like getMonth()).

Friend functions are mostly useful for overloading operators, which we talk about in the next chapter. It's generally good to keep your friend functions to a minimum and use accessors if you can.

Mr D
October 7, 2015 at 1:38 pm · Reply

Hi Alex,

Regarding the first code example from this lesson:

If i understand correctly, the reason you're using a reference (Accumulator &cAccumulator for example) is because

there's also an int main() part that's implicit, but that you didn't write out?
So for example, the whole thing might look something like this?!:

```
1  class Accumulator
2  {
3  private:
4      int m_nValue;
5  public:
6      Accumulator() { m_nValue = 0; }
7      void Add(int nValue) { m_nValue += nValue; }
8
9      friend void Reset(Accumulator &cAccumulator);
10
11     int get_m_nValue() { return m_nValue; }
12
13     void setm_nValue(int set) { m_nValue = set; }
14 };
15
16 void Reset(Accumulator &cAccumulator)
17 {
18         cAccumulator.m_nValue = 0;
19 }
20
21 int main()
22 {
23     Accumulator cAccumulator;
24     cAccumulator.setm_nValue(55);
25     std::cout << cAccumulator.get_m_nValue() << "n";
26     Reset(cAccumulator);
27     std::cout << cAccumulator.get_m_nValue() << "n";
28 }
```

Alex
October 8, 2015 at 8:55 am · Reply

No, reset is passed in as a reference so that the function can modify the actual Accumulator argument passed in, not a copy of it. If you were to pass cAccumulator by value, you'd end up resetting a copy of the argument, and the original argument would be unchanged. That would defeat the purpose of calling the function. 🙂

Danny
December 30, 2015 at 7:13 am · Reply

I have read in one of your tutorials that passing by reference saves the overhead that would rather be incurred when passing by value.
I tried the same program example with passing by value and it works perfectly fine.

Alex
December 31, 2015 at 5:41 pm · Reply

> I have read in one of your tutorials that passing by reference saves the overhead that would rather be incurred when passing by value.

True.

> I tried the same program example with passing by value and it works perfectly fine.

Which example are you referring to?

Bassam
February 23, 2016 at 2:05 am · Reply

Passing classes by reference is good for performance. see Passing arguments by reference

Reaversword
January 8, 2016 at 9:05 pm · Reply

I've been trying to get reciprocal "friendship" between two classes. I don't know if this is nonsense, is just for make the try. But I'm unable to get it working. The only way to get one single class being friend of another one is with this "order":

1)forward declaration of the friend class (the one able to access the other one's data)
2)definition of the class gonna be accessed (the one which accepts friendship)
3)definition of the friend class.

If I go out of that structure, it doesn't works (as swap 2 & 3 steps).

This is the rule for friendship between classes?.

A quick example:

```
#include <iostream>
using std::cout;
using std::cin;

class Bclass;
class Aclass
{
    friend class Bclass;

    private:
        int aVar = 0;

    public:
        int getB(Bclass &Bc)
        {
            //return Bc.bVar;//This line is a problem!
            return 0;
        }
};

class Bclass
{
    friend class Aclass;

    private:
        int bVar = 2;

    public:
        int getA(Aclass &Ac)
        {
            return Ac.aVar;
        }
};

int main( )
{
    cin.ignore( );
    return 0;
}
```

VisualStudio throws the error:

C2027 use of undefined type ‘Bclass’
and
C2228 left of ‘.bVar’ must have class/struct/union

So, what’s wrong?
There is a way to get both classes friend of each other?

Alex
January 9, 2016 at 6:28 pm · Reply

The problem here is that the forward declaration tells the compiler about the existence of a class, but nothing about it’s details. So the compiler can’t validate what members the class has.

The best way to solve this is to move the function definitions out of the class declarations. That way, by the time the compiler compiles them, it will have already seen the class declarations and should know how to resolve all of the names.

Reaversword
January 9, 2016 at 7:36 pm · Reply

Now I understand. Wow!, I was trusting too much in the forward declaration sentence. I really didn’t expect that. Thank you Alex, one more time, brilliant explanation, consecuent solution.

I leave the code, if somebody wants take a look:

```
1  #include <iostream>
2  using std::cout;
3  using std::cin;
4
5  class Bclass;
6  class Aclass
7  {
8      friend class Bclass;
9
10     private:
11         int aVar = 0;
12
13     public:
14         int getB(Bclass &Bc);
15 };
16
17 class Bclass
18 {
19     friend class Aclass;
20
21     private:
22         int bVar = 2;
23
24     public:
25         int getA(Aclass &Ac);
26 };
27
28 //The function definitions AFTER the declarations of BOTH classes
29 //(In this way compiler finds "Bc.bVar").
30     int Bclass::getA(Aclass &Ac)
31     {
32         return Ac.aVar;
33     }
34
35     int Aclass::getB(Bclass &Bc)
36     {
```

```
        return Bc.bVar;
    }

int main( )
{
    Aclass ac;
    Bclass bc;
    cout<<ac.getB(bc)<<".\n";
    cout<<bc.getA(ac)<<".\n";
    cin.ignore( );
    return 0;
}
```

JonM
January 15, 2016 at 2:56 pm · Reply

First I want to thank you on a fantastic tutorial.

Instead of using Friend class, could one not just get the same result, with just accessing the friend class through getters and setters? And no risk of violating encapsulation would be at risk?

Alex
January 19, 2016 at 1:14 pm · Reply

Yes, if the class provides getters and setters sufficient to do the job. However, often this is not the case.

Lokesh
February 9, 2016 at 7:09 am · Reply

In the "multiple friends" section the setHumidity() and setTemperature() functions used in main() are not defined inside the class definitions.

Alex
February 10, 2016 at 5:19 pm · Reply

Fixed! Thanks.

Lokesh
February 10, 2016 at 4:26 am · Reply

In first code example in this lesson,

```
// And can access the private data of Accumulator objects
    accumulator.value = 0;
```

should be

```
// And can access the private data of Accumulator objects
    accumulator.m_value = 0;
```

Alex
February 10, 2016 at 5:39 pm · Reply

Thanks. Fixed!

Lokesh
February 10, 2016 at 9:53 am · Reply

Quiz 1a. solution - line 20

```
1 friend class Point3d; // Vector3d is now a friend of class Point3d
```

should be

```
1 friend class Point3d; // Point3d is now a friend of class Vector3d
```

Alex
February 11, 2016 at 9:53 am · Reply

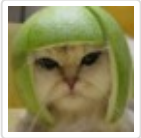
Fixed, thanks.

Meric
February 19, 2016 at 3:20 am · Reply

Hmmm. The compiler (g++ with -std=c++0x) kept complaining until I made the following changes to the code given under "Multiple Friends". Any idea why?

```
1  #include <iostream>
2
3  class Humidity;
4
5  class Temperature
6  {
7  private:
8    int m_temp;
9  public:
10   Temperature(int temp) {m_temp = temp;}
11   int setTemperature(int temp) {return m_temp = temp;} // entered return type
12   friend void printWeather(const Temperature &temperature, const Humidity &humidity);
13 };
14
15 class Humidity
16 {
17 private:
18   int m_humidity;
19 public:
20   Humidity(int humidity) {m_humidity = humidity;}
21   int setHumidity(int humidity) { return m_humidity = humidity;} // entered return type
22   friend void printWeather(const Temperature &temperature, const Humidity &humidity);
23 };
24
25 void printWeather(const Temperature &temperature, const Humidity &humidity)
26 {
27   std::cout << "The temperature is " << temperature.m_temp << " and the humidity is " << hu
28 midity.m_humidity << '\n';
29 }
30
31 int main()
32 {
33   // These had to be initialized.
34
35   Humidity hum(0);
36   Temperature temp(0);
37
38   //
39
40   hum.setHumidity(10);
```

```
41    temp.setTemperature(12);
42
43    printWeather(temp, hum);
44
45    return 0;
}
```



Alex
February 20, 2016 at 10:51 am · Reply

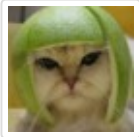
Error on my part -- not sure how that slipped past my compiler. I've updated the example so it should compile now. 🙂

Bassam
February 23, 2016 at 2:24 am · Reply

Good work, The code has not been updated. you should put int before the function name

```
1 setTemperature(int temp) { m_temp = temp; }
```

```
1 setHumidity(int humidity) { m_humidity = humidity; }
```



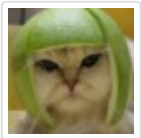
Alex
February 24, 2016 at 5:01 pm · Reply

Oops. Example updated. Those two functions now return void.

malt
March 3, 2016 at 6:28 am · Reply

On the solution for the 1b) i got this error:
main.cpp 22:7: error: extra qualification 'Point3d::' on member 'moveByVector' [-fpermissive]
void Point3d::moveByVector(Vector3d &v); // so we can use Vector3d here

I use g++ -std=c++11 main.cpp for compile.

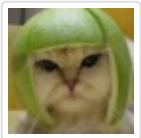


Alex
March 6, 2016 at 2:11 pm · Reply

Oops, my bad. I accidentally left a Point3d:: prefix on moveByVector. Visual Studio didn't complain about it, so I didn't notice. The example is fixed now.

Bryan
March 11, 2016 at 1:23 am · Reply

Thanks for the tutorial.
I have a question, how to do when Storage and Display classes are in separate files(Storage.h, Storage.cpp, Display.h, Display.cpp)? The compiler shows error: invalid use of incomplete type 'class Storage'



Alex
March 14, 2016 at 1:24 pm · Reply

Make sure you #include "Storage.h" from any file that references the Storage class.

Xhevo
March 12, 2016 at 10:58 am · Reply

Hi, Alex

Typo to line 11 of friend member functions:

11. void Display::displayItem(Storage &storage);// forward declaration above needed for this declaration line

should be:
11. void displayItem(Storage &storage); // forward declaration above needed for this declaration line

Thanks again!

Alex
March 14, 2016 at 1:50 pm · Reply

Fixed. Thanks!

Mekacher Anis
March 17, 2016 at 8:04 am · Reply

hey , sorry but I forgot in which lesson you told as about the compiler adding *this pointer in the member functions , can u remind me of the lesson please ?