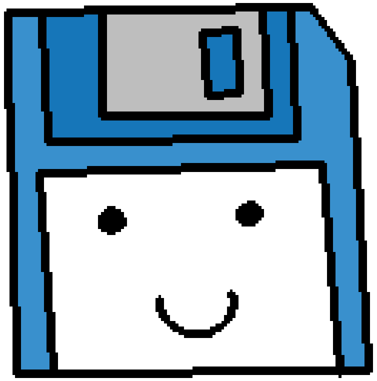
Friends

Written by Rachel J. Morris, last updated 2016-03-05

Friends of Classes

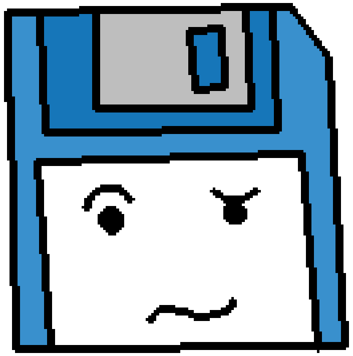


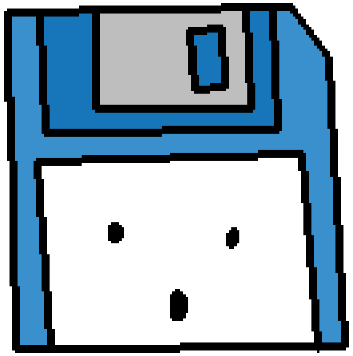
Friends!

In C++, classes can have **friends**.

Friends do not belong to the class itself, but can access members of a class.

A class can have its private and protected members accessible to friend functions and friend classes.





Why have friends?

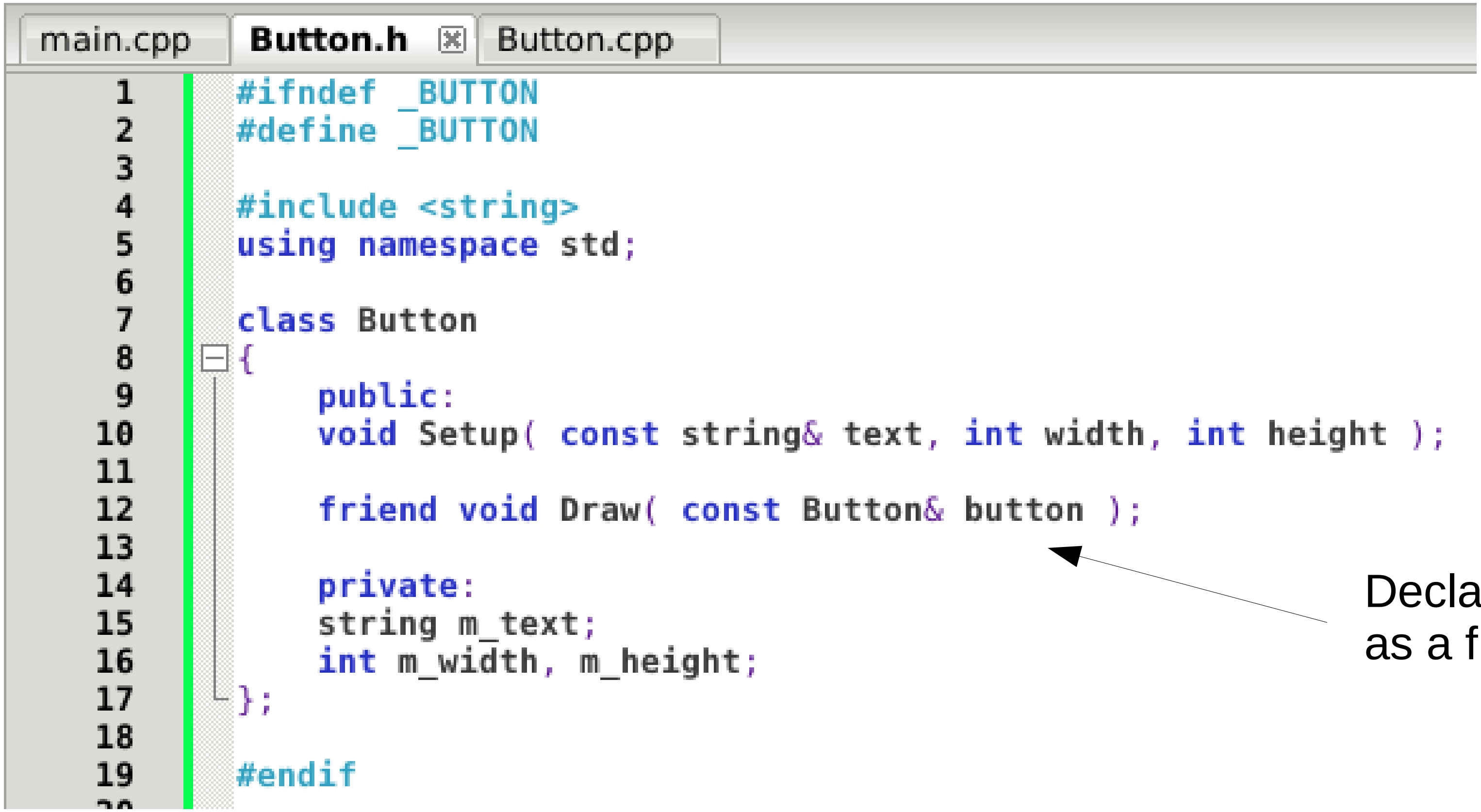
Maybe one class will work with another, and needs to access protected members.

We don't want to add any public functions to access these because only one other class should have access!

Friend functions

A Friend function can access **private members of a class** directly.

The Friend function can be part of another class, or just be a function independent of any objects.

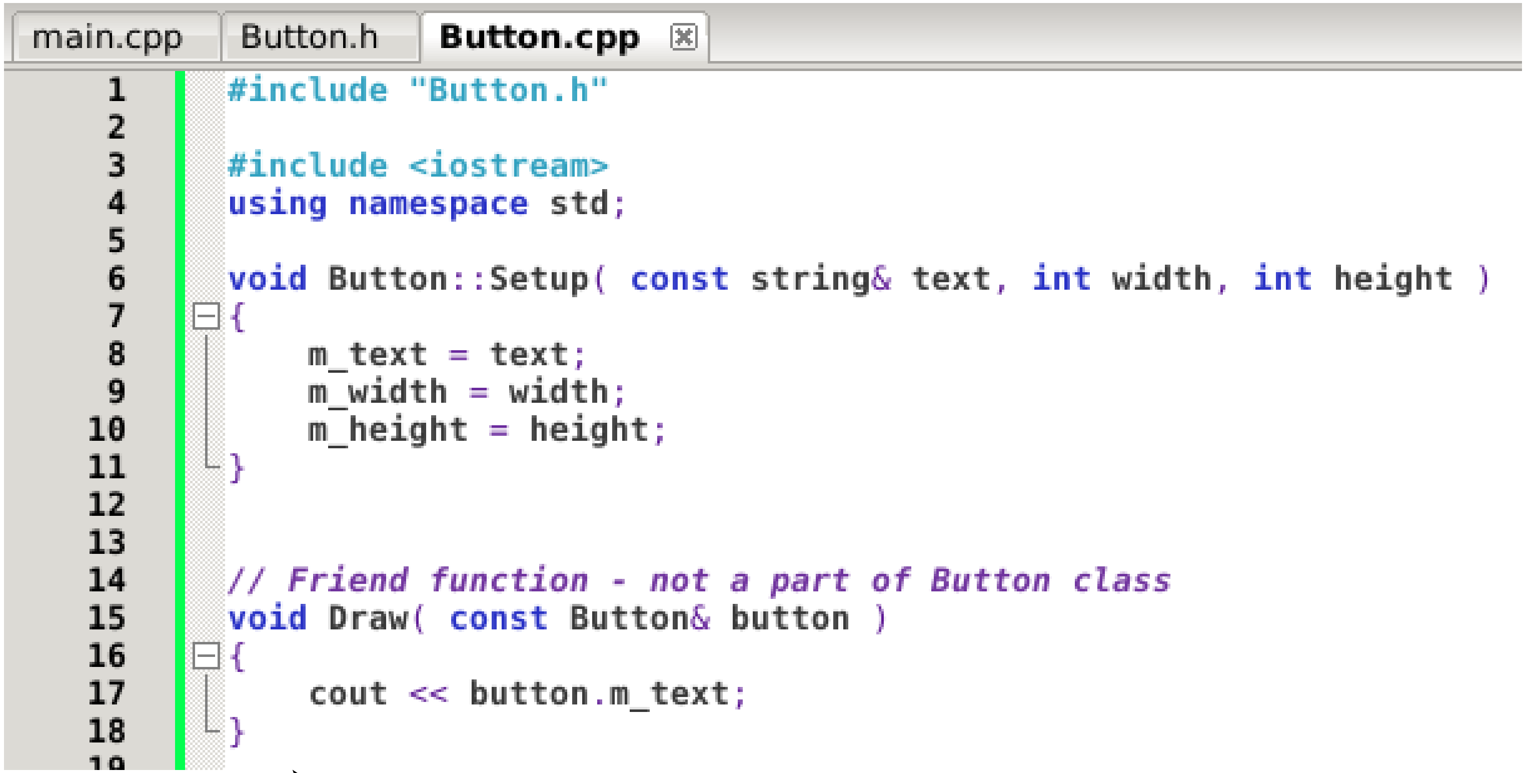


```
#ifndef _BUTTON
#define _BUTTON

#include <string>
using namespace std;

class Button
{
    public:
    void Setup( const string& text, int width, int height );

    friend void Draw( const Button& button );

    private:
    string m_text;
    int m_width, m_height;
};

#endif
```

Declaring a function as a friend

Friend functions

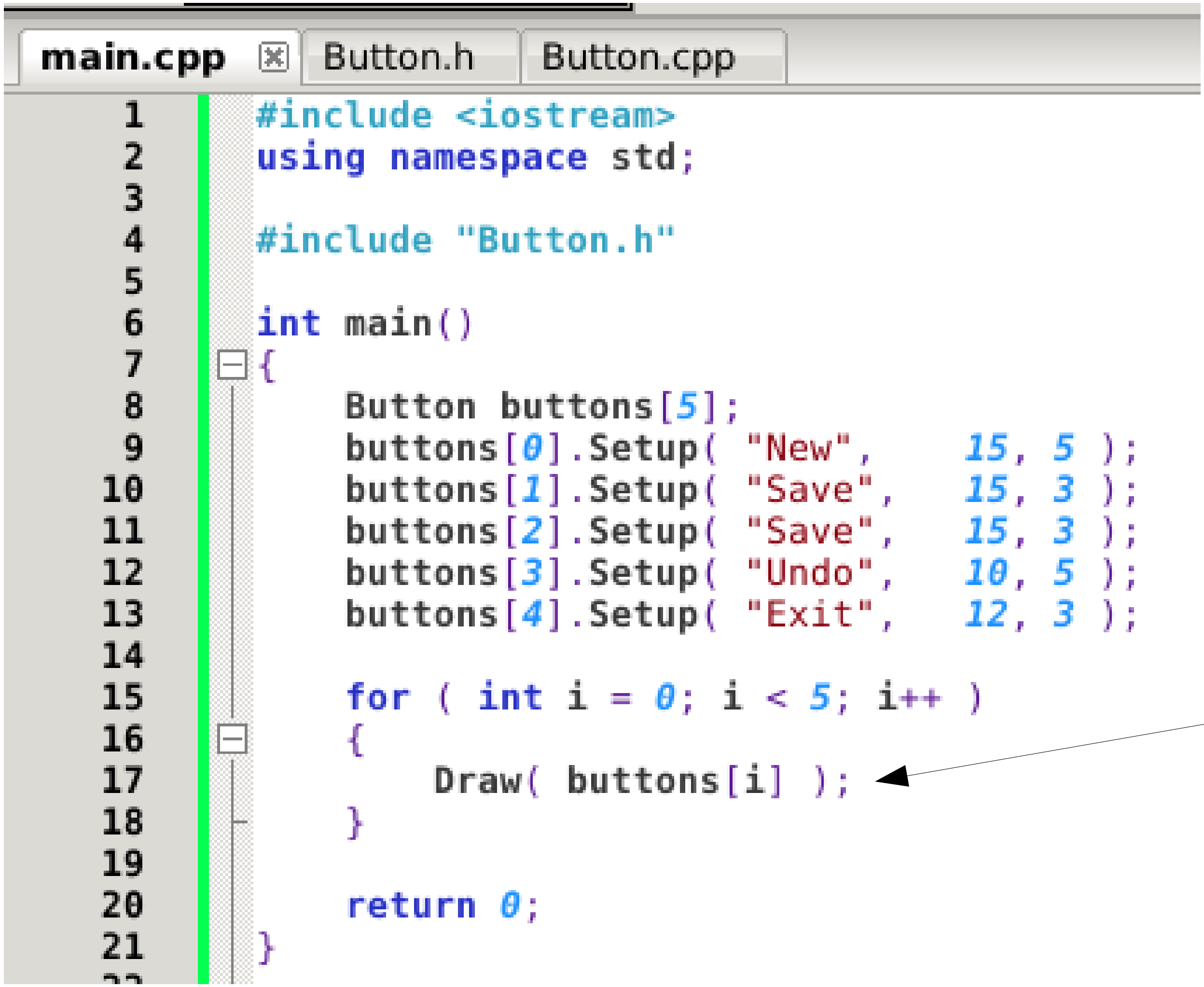
A friend function can be just a normal function that is not a class member



```
#include "Button.h"

#include <iostream>
using namespace std;

void Button::Setup( const string& text, int width, int height )
{
    m_text = text;
    m_width = width;
    m_height = height;
}


// Friend function - not a part of Button class
void Draw( const Button& button )
{
    cout << button.m_text;
}
```

Defining the Friend function inside of the Button.cpp file. Can access private member m_text!

Friend functions



```cpp
#include <iostream>
using namespace std;

#include "Button.h"

int main()
{
    Button buttons[5];
    buttons[0].Setup( "New",    15, 5 );
    buttons[1].Setup( "Save",   15, 3 );
    buttons[2].Setup( "Save",   15, 3 );
    buttons[3].Setup( "Undo",   10, 5 );
    buttons[4].Setup( "Exit",   12, 3 );

    for ( int i = 0; i < 5; i++ )
    {
        Draw( buttons[i] );
    }

    return 0;
}
```

Call the Friend function normally, it will have access to the **argument's** private members.

Friend functions

Say we have a Fraction class. Normally when we write Fractions in math, we have something like:

$$\frac{1}{2}+\frac{4}{5}=\frac{a}{b}$$

Friend functions

If we have a Fraction class in C++, and want to multiply a second Fraction, this *could* be a function that belongs to the Fraction class, but it looks awkward:

```
Fraction.sum = fraction1.Multiply( fraction2 );
```

It looks a little better to have the fractions both be arguments:

```
Fraction.sum = Multiply( fraction1, fraction2 );
```

Friend functions

But if we create an external function to multiply two fractions

```
Fraction.sum = Multiply( fraction1, fraction2 );
```

then we'd need to add Getters & Setters for the fraction...

- Set Numerator
- Set Denominator
- Get Numerator
- Get Denominator

Friend functions

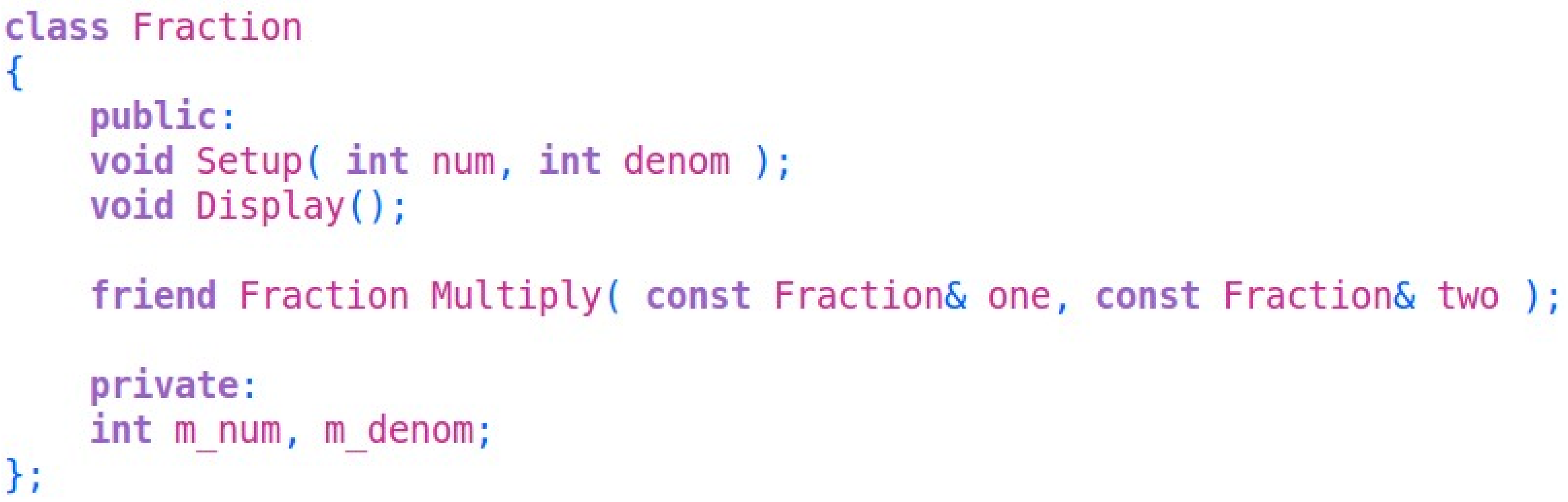
Instead of adding these getters/setters,

- Set Numerator
- Set Denominator
- Get Numerator
- Get Denominator

We could instead make the **`Multiply`** function a friend of the Fraction class, so **`Multiply`** can access the numerators and denominators directly.

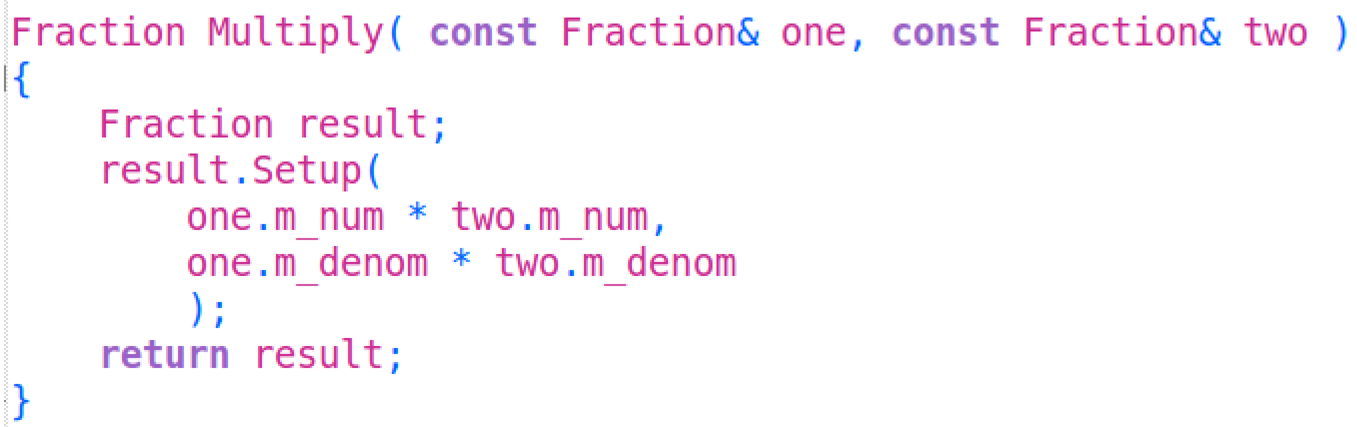
Friend functions

Within the class, we **declare** the function as a **friend**.



```
class Fraction
{
    public:
    void Setup( int num, int denom );
    void Display();

    friend Fraction Multiply( const Fraction& one, const Fraction& two );

    private:
    int m_num, m_denom;
};
```

Outside of the class, we **define** the function.

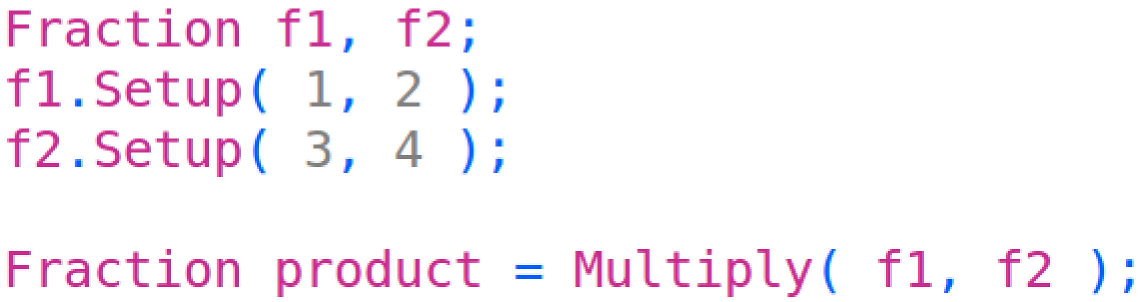


```
Fraction Multiply( const Fraction& one, const Fraction& two )
{
    Fraction result;
    result.Setup(
        one.m_num * two.m_num,
        one.m_denom * two.m_denom
        );
    return result;
}
```

Note that it is <u>not</u> a member of Fraction.

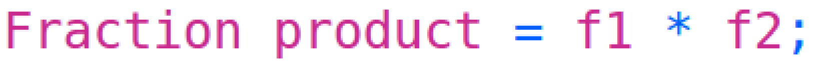
Friend functions

Now we can call it in main:



```
Fraction f1, f2;
f1.Setup( 1, 2 );
f2.Setup( 3, 4 );

Fraction product = Multiply( f1, f2 );
```

And once we get to **operator overloading**, we can do this instead:

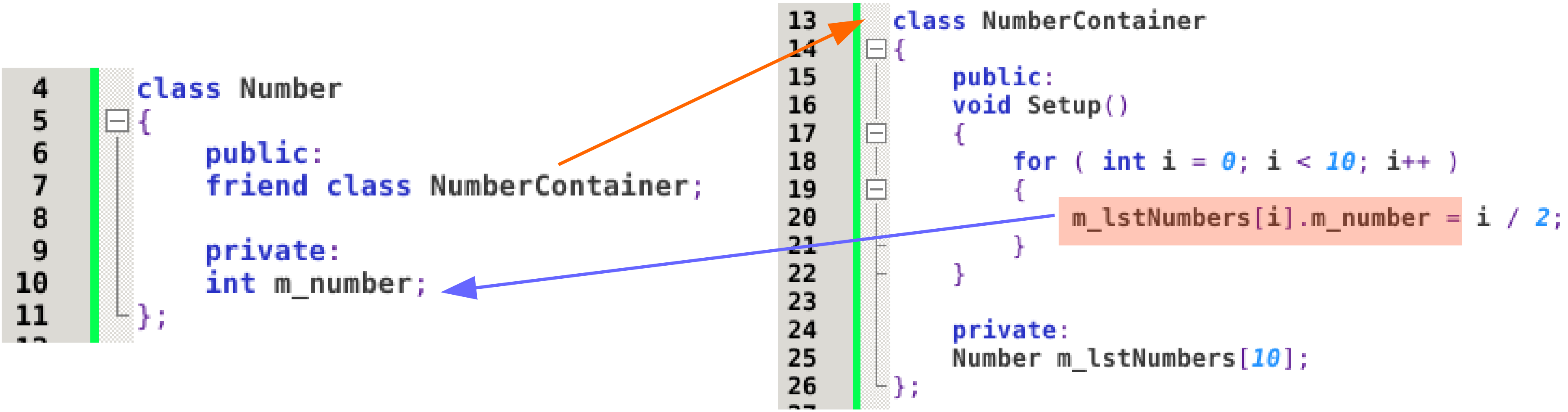


```
Fraction product = f1 * f2;
```

Friend classes

Besides just functions, we can make Classes friends with each other.

The **Friend Class** has access to the other's private members, but not vice versa.



```
class Number
{
    public:
    friend class NumberContainer;

    private:
    int m_number;
};
```

```
class NumberContainer
{
    public:
    void Setup()
    {
        for ( int i = 0; i < 10; i++ )
        {
            m_lstNumbers[i].m_number = i / 2;
        }
    }

    private:
    Number m_lstNumbers[10];
};
```

Notes about friend classes

Friendships go one-way;
if class A is a friend of class B,
it doesn't automatically mean that:
class B is a friend of class A.

Friendships are not transitive
if class A is a friend of class B,
and class B is a friend of class C
it doesn't automatically mean that:
class C is a friend of class A.

Friendships are not inherited;
a child class does not
inherit its parents' friends.

Access due to friendship *is* inherited:
If Class “Email” is a friend of “Attachment”
then Email can access Attachment's
members, including the members inherited
by its children, through the children.

https://en.wikipedia.org/wiki/Friend_class

Friends!

We will see in-code examples of using friend functions in the lecture on **operator overloading**