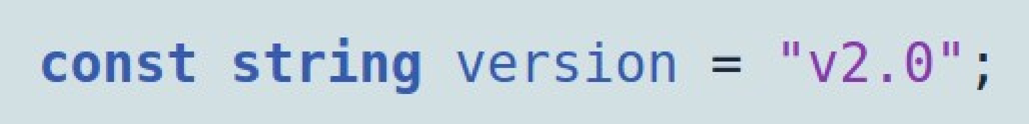
Constants
Part 2

Written by Rachel J. Morris, last updated 2016-03-06

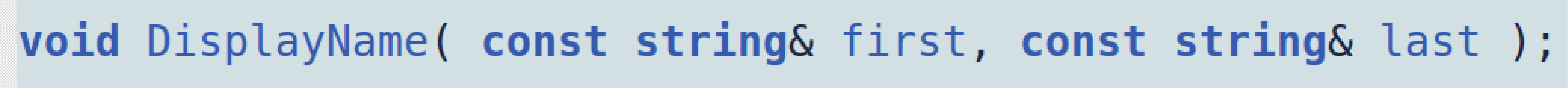
Previously, we've talked about...

Constant variables



```
const string version = "v2.0";
```

Const pass-by-reference



```
void DisplayName( const string& first, const string& last );
```

But now...

Now that we've covered **classes**, there are more ways we can use the **const** keyword.

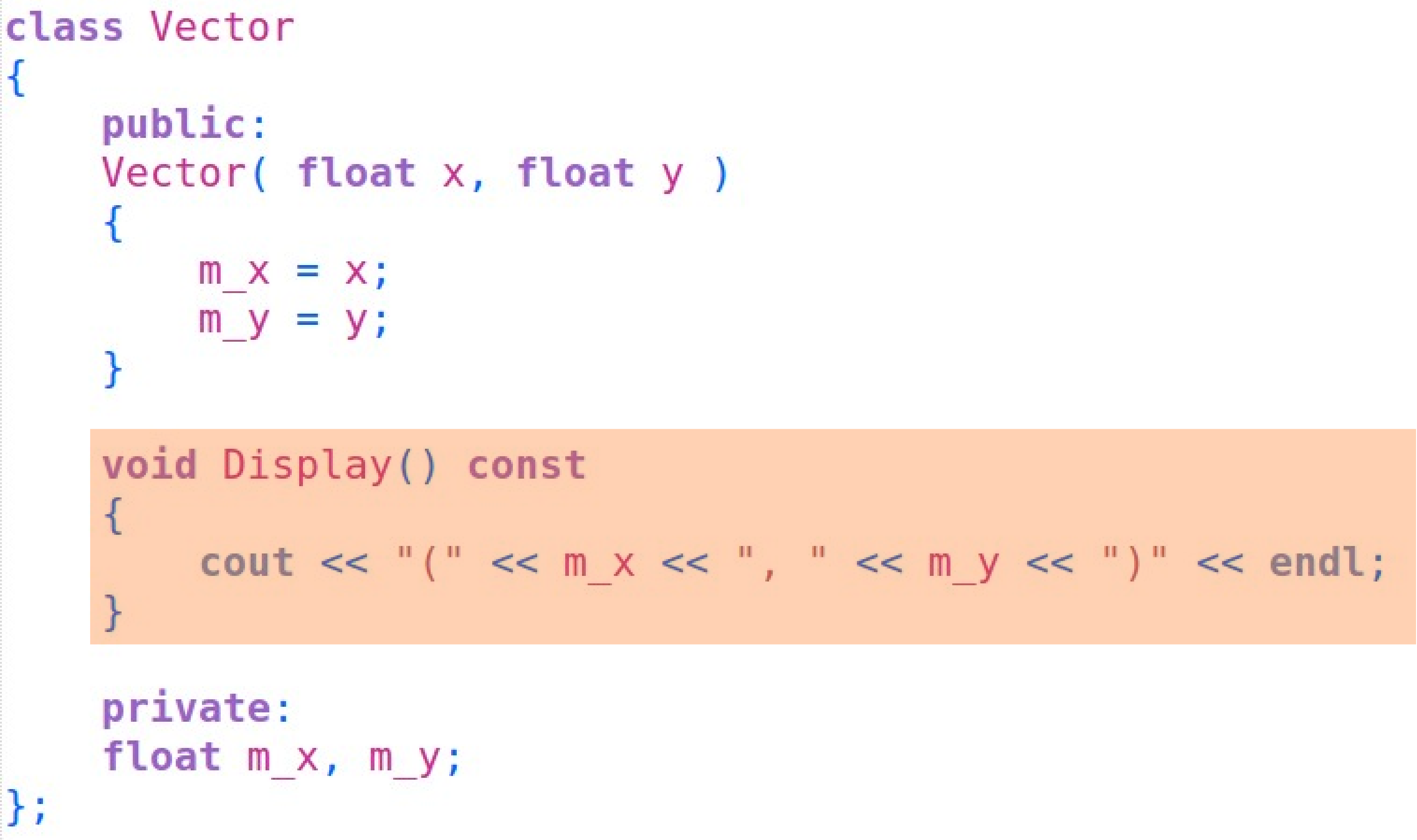
We can make member functions of a class **const**, in order to ensure that member variables are not changed.

We can also return references from functions, and in some cases we may want to return a **const reference.**

We might also want to create a pointer to some address, but ensure the value at that address cannot change.

Const Functions

When declaring a member function within a class, if we
do not want any member variables to change,
we can define the function as **const** by adding
the **const** keyword to the end of the signature.

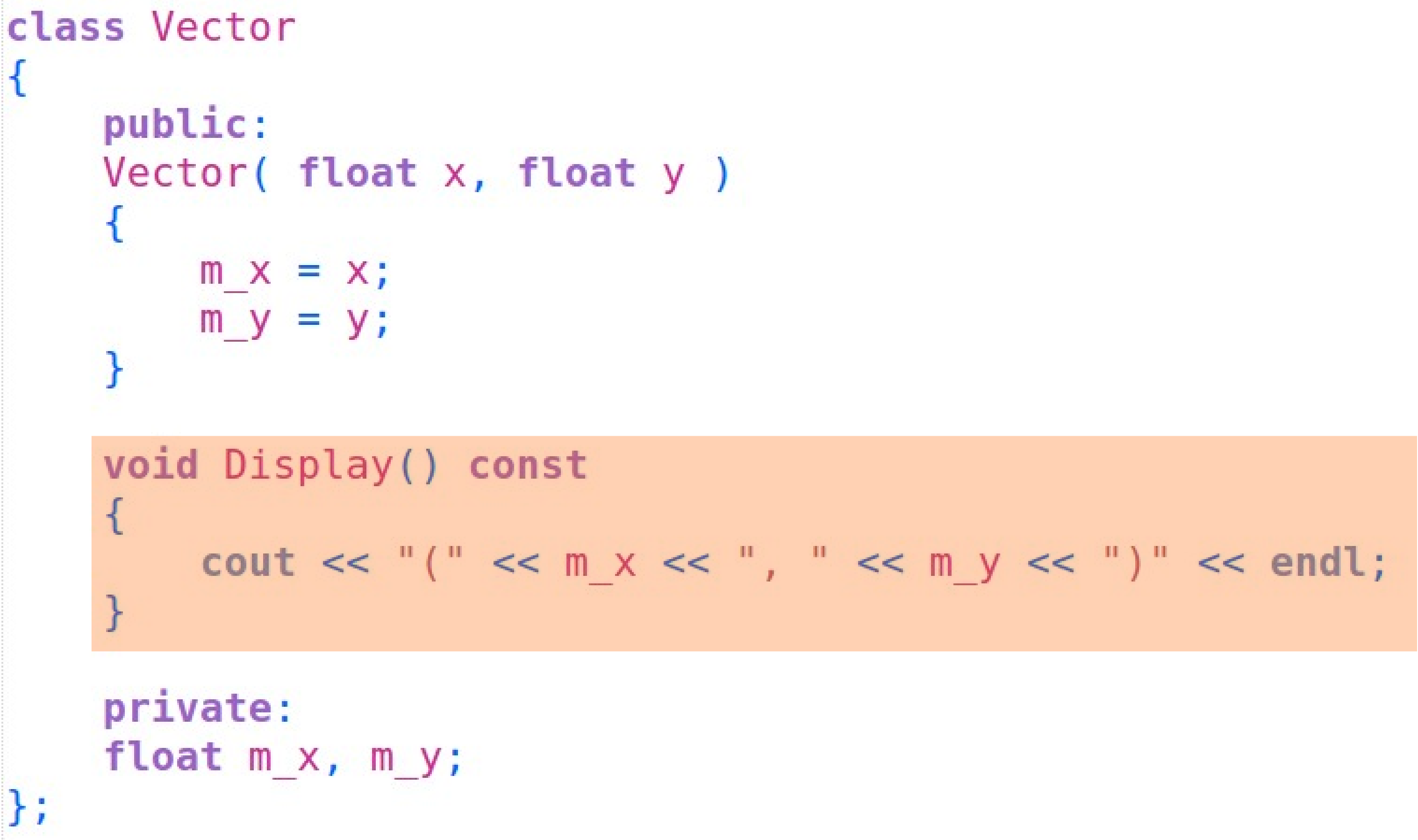


```
class Vector
{
    public:
    Vector( float x, float y )
    {
        m_x = x;
        m_y = y;
    }

    void Display() const
    {
        cout << "(" << m_x << ", " << m_y << ")" << endl;
    }

    private:
    float m_x, m_y;
};
```

Because this function is **const**, it can output the values of
`m_x` and `m_y`,
but it cannot change them – otherwise, a compile error will occur.

Const Functions

It is good to utilize the **const** keyword any time that you're writing a function where class members should not be changed, as good programming practice.

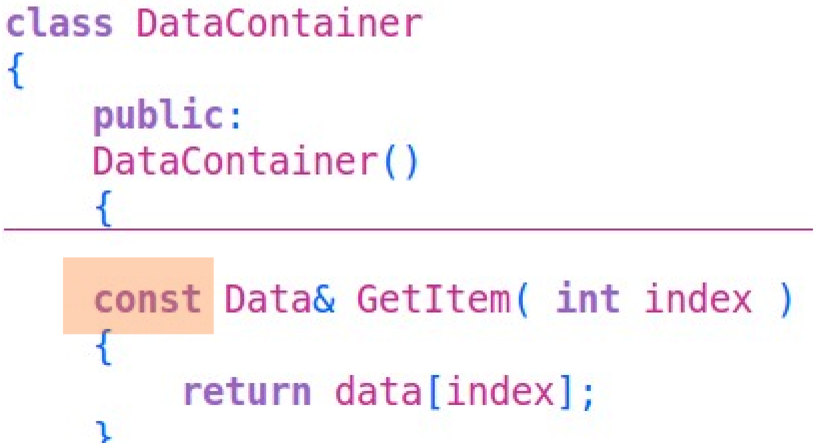
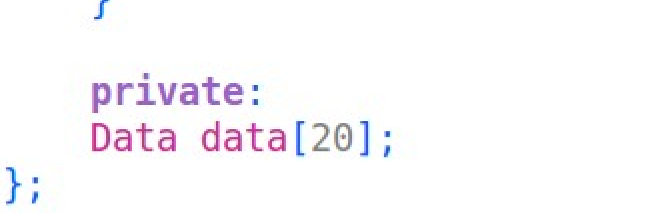


```
class Vector
{
    public:
    Vector( float x, float y )
    {
        m_x = x;
        m_y = y;
    }

    void Display() const
    {
        cout << "(" << m_x << ", " << m_y << ")" << endl;
    }

    private:
    float m_x, m_y;
};
```

Because this function is **const**, it can output the values of `m_x` and `m_y`, but it cannot change them – otherwise, a compile error will occur.

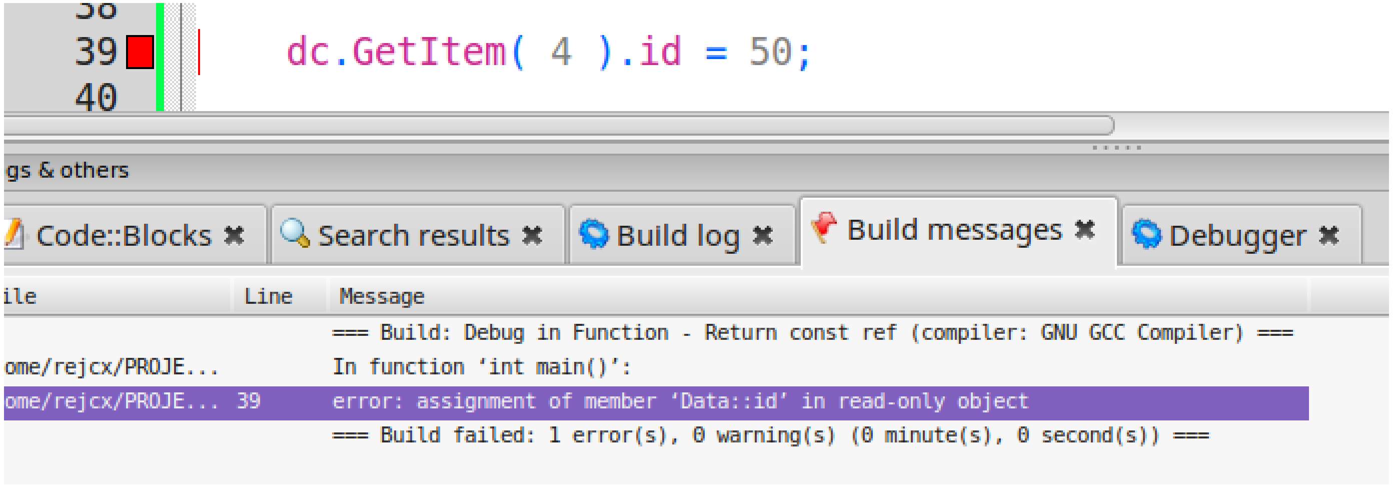
Returning Const

With some functions, you may return a **reference** to an item. However, while you may want to allow access to an item by returning a reference to it, you might not want anything externally to actually change the value.
You can return a const variable from a function.

```
class DataContainer
{
    public:
    DataContainer()
    {

    const Data& GetItem( int index )
    {
        return data[index];
    }

    private:
    Data data[20];
};
```

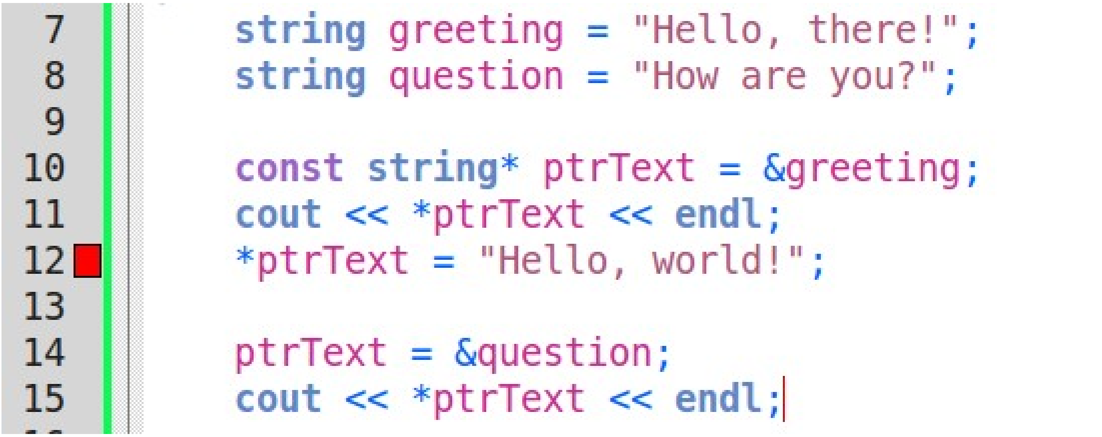
By returning something as a const reference, we can ensure that the compiler will throw an error if we try to modify that referenced item.



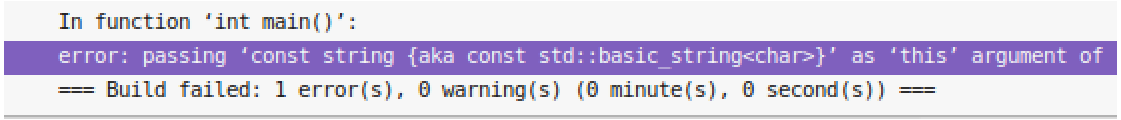
```
      dc.GetItem( 4 ).id = 50;
```

```
ile                  Line   Message
                            === Build: Debug in Function - Return const ref (compiler: GNU GCC Compiler) ===
ome/rejcx/PROJE...          In function ‘int main()’:
ome/rejcx/PROJE...   39     error: assignment of member ‘Data::id’ in read-only object
                            === Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

Const and Pointers

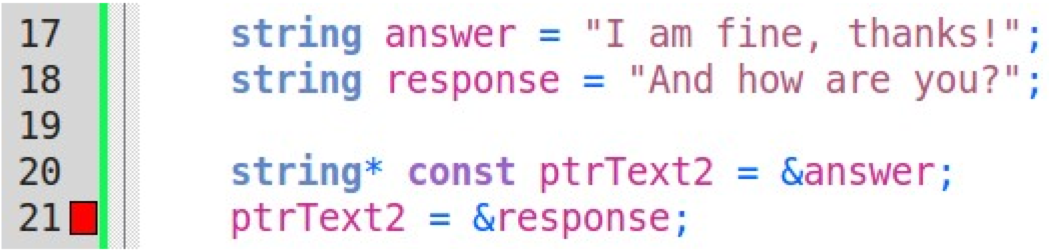


```
    string greeting = "Hello, there!";
    string question = "How are you?";

    const string* ptrText = &greeting;
    cout << *ptrText << endl;
    *ptrText = "Hello, world!";

    ptrText = &question;
    cout << *ptrText << endl;
```

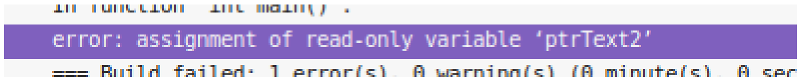


```
In function ‘int main()’:
error: passing ‘const string {aka const std::basic_string<char>}’ as ‘this’ argument of
=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

If we declare a pointer with **const** before the datatype*, then we can still change what our pointer is pointing to, but we cannot change the data it is pointing at.



```
    string answer = "I am fine, thanks!";
    string response = "And how are you?";

    string* const ptrText2 = &answer;
    ptrText2 = &response;
```



```
In function ‘int main()’:
error: assignment of read-only variable ‘ptrText2’
=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 sec
```

If we declare a pointer with **const** after the datatype*, we are saying that we cannot change the address that the pointer is pointing to.

The pointer must be assigned at declaration.

Review

Utilizing **const** is a good way to enforce design decisions, and have the compiler itself police these decisions – if something is written that violates your design, the program will not compile.

Some ways to utilize **const** are...

const variables

Pass-by-const-reference

Return const-reference

Const-pointer

Pointer-to-const