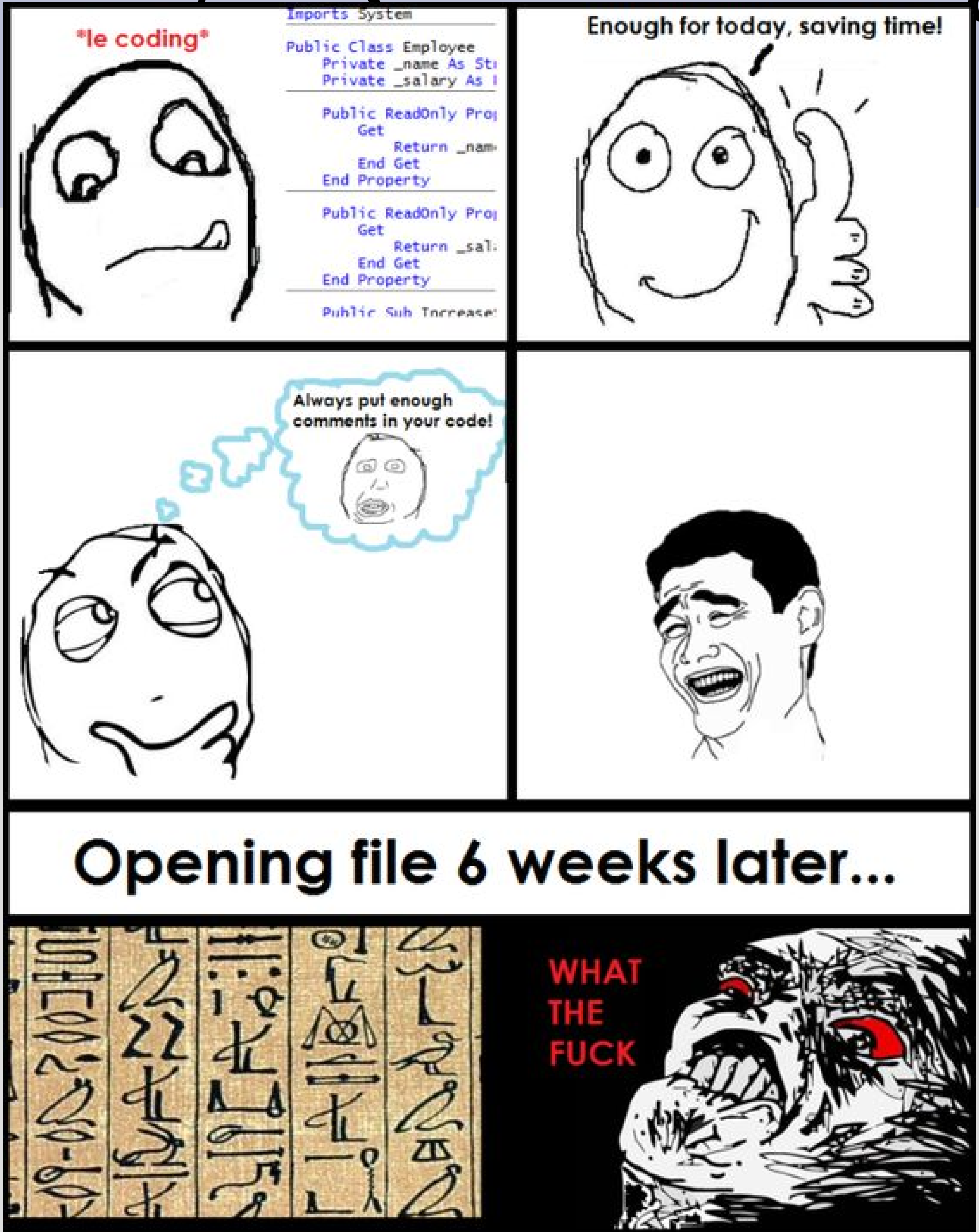
Arrays (and strings)


le coding
Enough for today, saving time!
Always put enough comments in your code!
Opening file 6 weeks later...
WHAT THE FUCK

```
Imports System

Public Class Employee
    Private _name As St
    Private _salary As 

    Public ReadOnly Pro
        Get
            Return _nam
        End Get
    End Property

    Public ReadOnly Pro
        Get
            Return _sal
        End Get
    End Property

    Public Sub Increase
```

Announcements

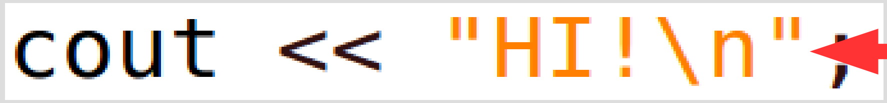
Quiz scores on gradescope (~6pm today)

Test next Wednesday (whole class period)

C-Strings and strings

There are actually two types of “strings” (multiple characters) in C++

A <u>C-String</u> is a char array, and this is what you get when you put quotes around words



```
cout << "HI!\n";
```

C-String

A <u>string</u> (the thing you #include) is a more complicated type called a <u>class</u> (few weeks)

C-Strings and strings

It is fairly easy to convert between C-Strings and strings:

```
char cString[] = "move zig";
string IMAstring = cString;
cout << IMAstring.c_str() << endl;
// above converts it back to C-String
```

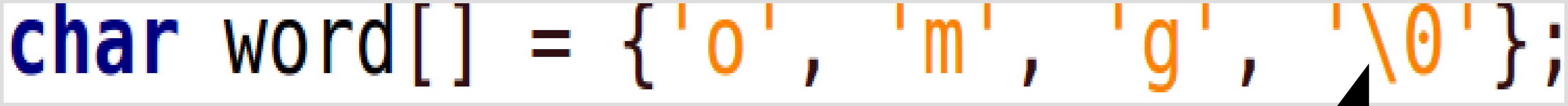
You can also convert between numbers and strings:

```
char number1[20];
string number2;
cin >> number1 >> number2;
cout << "sum is: " << (atof(number1) + stod(number2)) << endl;
```

(see: stringConversion.cpp)

C-Strings and strings

C-Strings are basically strings without the added functions



```
char word[] = {'o', 'm', 'g', '\0'};
```

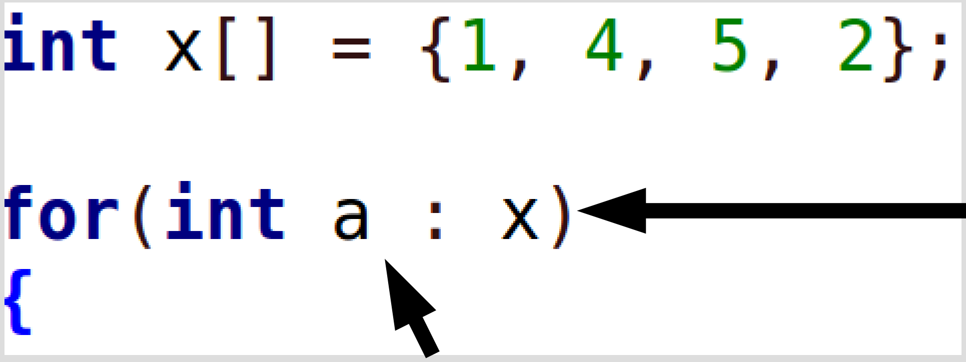
You should end C-Strings with <u>null character</u>, as this tells cout when to stop displaying

This means you can initialize char arrays with quotes (**BUT NOT OTHER ARRAYS**) (see: cstring.cpp)

Arrays - looping

As arrays store multiple elements, we very often loop over those element

There is a special loop that goes over all elements (for each):



```
int x[] = {1, 4, 5, 2};

for(int a : x)
{
```

x is an array

a has the value of x[i] for each i

(See: forEach.cpp)

Partially filled arrays

Arrays are annoying since you cannot change their size

You can get around this by making the array much larger than you need

If you do this you need to keep track of how much of the array you are actually using

(See: partiallyFilled.cpp)

Array - element passing

Each element of an array is the same as an object of that type

For example: `int[] x = {1, 2};`
x[0] is an int, and we can use it identical as if we said: `int x0 = 1;`

(See: maxPassInt.cpp)

Array - array passing

Arrays are references (memory addresses)

This means we can pass the reference as an argument in a method

Then the method can see the whole array, but it won't know the size

(See: maxPassArray.cpp)

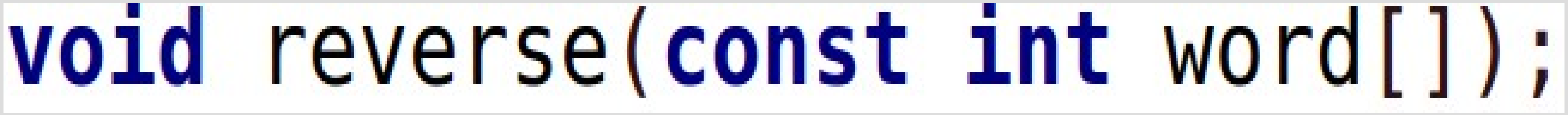
Array - array passing

But wait! This means the function can change the data since we share the memory address

(See: reverse.cpp)

Array - array passing

If we want to prevent a function from modifying an array, we can use const in the function header:



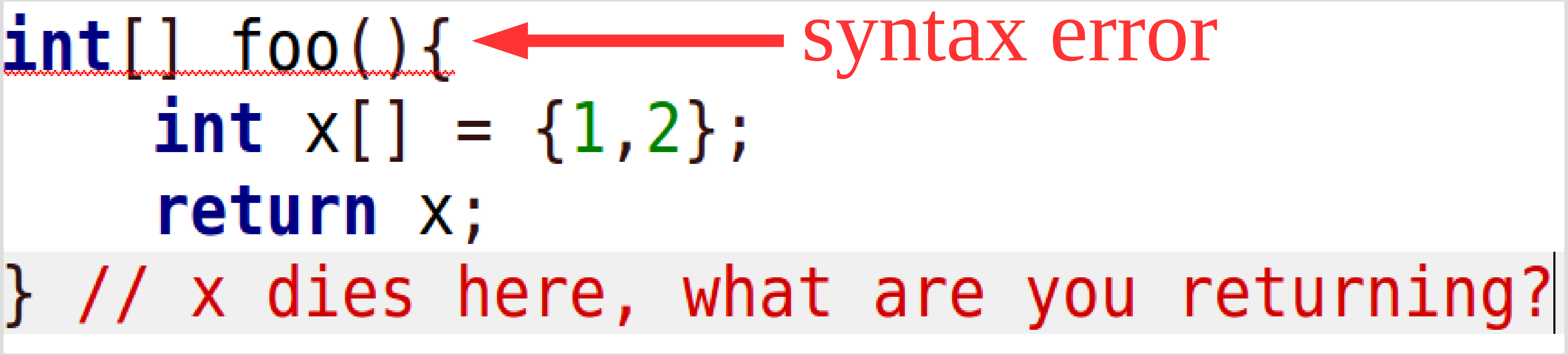
```
void reverse(const int word[]);
```

This also means any function called inside reverse must also use const on this array

(See: reverseFail.cpp)

Array - returning arrays

However, we do not know how to return arrays from functions (yet)



syntax error

```
int[] foo(){
    int x[] = {1,2};
    return x;
} // x dies here, what are you returning?
```

For now, you will have to pass in an array to be changed, much like call-by-reference