Section 1 / What is “this”

When using an instance of a C++ `class` or `struct`, you know already that any method you call somehow knows which specific instance you are using. The exception to *this*, of course, are methods which are declared as `static` which do *not* know which specific instance you are referring to - `static` methods are instance-agnostic.

Like many of the cool features of C++, this seemingly magical self-awareness of which instance a non-static method is called with, is accomplished by slight-of-hand.

`this` pointer

Every non-static method call employs a hidden first parameter. That’s it. That’s the slight of hand. The hidden argument is the `this` pointer. It points to the base address of the class or struct. Then, the particular members of the right instance, are accessed the same way as any struct basing memory addresses off a pointer.

Here is the source code to our test harness:

```
#include <stdio.h>                                          // 1
                                                            // 2
class TestClass {                                           // 3
    public:                                                 // 4
    void SetString(char * s);                               // 5
    char * GetString();                                     // 6
    private:                                                // 7
    char * _s = nullptr;                                    // 8
};                                                          // 9
                                                            // 10
void TestClass::SetString(char * s) {                       // 11
    _s = s;                                                 // 12
    printf("String set to: %s\n", _s);                      // 13
}                                                           // 14
                                                            // 15
char * TestClass::GetString() {                             // 16
    return _s;                                              // 17
}                                                           // 18
                                                            // 19
char * test_string = (char *) "Test String";                // 20
                                                            // 21
TestClass tc;                                               // 22
                                                            // 23
int main() {                                                // 24
    tc.SetString(test_string);                              // 25
```

```
    printf("Stored string is: %s\n", tc.GetString());          // 26
    return 0;                                                  // 27
}                                                              // 28
```

There are no surprises in this code.

On line 25, when method `SetString()` is called, a specific instance of the `class` is used, namely the one called `tc`. It looks like there is only one parameter to the method call.

On line 26, when method `GetString()` is called, it will return the a pointer to the very same string that was used in the previous line.

If compiled with:

```
g++ -std=c++11 -O0 -S this.cpp
```

an assembly language file called `this.s` will be produced. In it, after wading through a lot of what seems like gibberish, you will find:

```
    adrp x8, _test_string@PAGE
    ldr x1, [x8, _test_string@PAGEOFF]
    adrp x0, _tc@PAGE
    add x0, x0, _tc@PAGEOFF
    str x0, [sp, #16]                   ; 8-byte Folded Spill
    bl __ZN9TestClass9SetStringEPc
```

followed by:

```
    ldr x0, [sp, #16]                   ; 8-byte Folded Reload
    bl __ZN9TestClass9GetStringEv
```

Note: this was built on a Mac M series. The code, if built on Linux, will be slightly different.

Notice the address of the string winds up in `x1` **not** in `x0` where a first parameter ought to have gone.

What goes in `x0`, i.e. the first parameter slot, is a pointer to the specific instance of the `class` being used.

Magic!