Peter Bindels (@dascandy42) TomTom International

Making a mockery of C++

(in 8 minutes)

Situation

```
class IMyBank {
public:
    virtual void PayTheMan(size_t cents) = 0;
};

TEST(TaxSoftPaysTheMan) {
    MockRepository context;
    IMyBank *myTestBank = context.Mock<IMyBank>();
    EXPECT(myTestBank, PayTheMan).With(20);
    TaxSoft myTaxSoftware(myTestBank);
    myTaxSoftware.PayTaxes(50);
    context.Verify();
}
```

Does this work?

```
IMyBank *myTestBank = myTestContext.Mock<IMyBank>();
```

Given no further information, can we create a working mock object?

- No pre-registration → Maintenance nightmare
- No manual mock class creation → Don’t Repeat Yourself
- No scripts running in your build → Unfriendly, slow, buggy...

Does this work?

```
IMyBank *myTestBank = myTestContext.Mock<IMyBank>();
```

Given no further information, can we create a working mock object?

- No pre-registration → Maintenance nightmare
- No manual mock class creation → Don’t Repeat Yourself
- No scripts running in your build → Unfriendly, slow, buggy...

C++ Strict: No

Does this work?

```
IMyBank *myTestBank = myTestContext.Mock<IMyBank>();
```

Given no further information, can we create a working mock object?

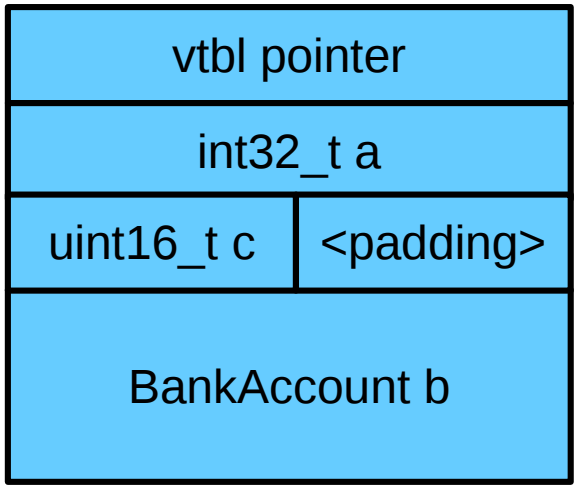
- No pre-registration → Maintenance nightmare
- No manual mock class creation → Don't Repeat Yourself
- No scripts running in your build → Unfriendly, slow, buggy...

C++ Strict: No

Pragmatic: **Yes**

Object layout in memory

- **4/8-byte pointer to vtable**
- **All members in order, with sufficient padding to ensure alignment**

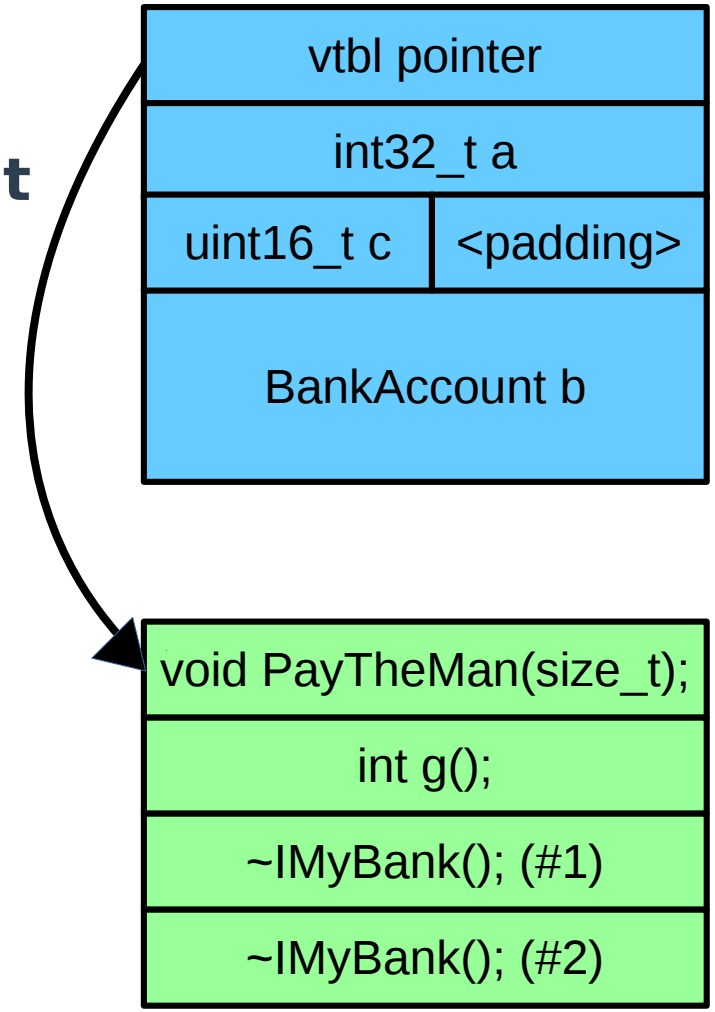


Object layout in memory

- **4/8-byte pointer to vtable**
- **All members in order, with sufficient padding to ensure alignment**

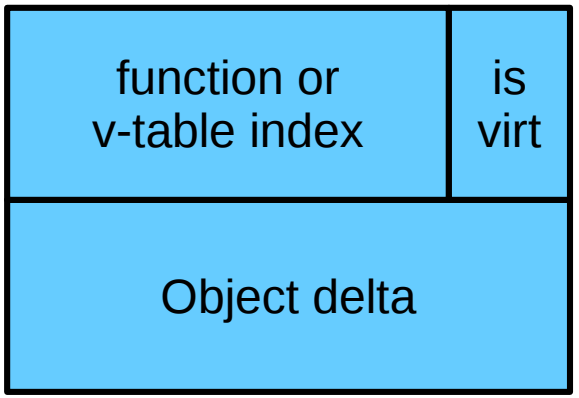
- **Vtable**

  … it’s actually just an array of function pointers.



Virtual member function pointer layout

```
struct MFP {
    union {
        // If nonvirtual, always even
        void (*function)();
        // If virtual, always odd
        size_t vindex_times_two_plus_one;
    };
    size_t delta;
};
```



- **C++ standard does not specify this**
  - So from a C++ standard strict PoV this is UB
- **C++ platform ABIs do specify this**
  - So from a pragmatic C++ PoV this is well-defined

- **May not play well with inlining, LTO or other code flow tools that use the C++ standard interpretation**
- **Does play well with all tested dynamic checkers**

So to mock this...

- **Create a function that just throws an exception.**
  - Only guaranteed OK return from unknown return type
- **Fill a large array with that function.**
- **Alloc an object of at least size T**
- **Initialize all X-sized chunks of memory with “uninitialized functions vtable” pointer**
- **Return reinterpret_cast<IMyBank*>(myMock)**

And then to expect a call

- **Take the expectation**
  - With some magic to know the name and type of the function & to get a unique invocation number
- **Reverse engineer the function pointer to object offset and virtual function index**
- **Replace that vtable with an actual vtable, initialized to “undefined function call”**

- **Replace the function entry with an equivalent function**
  - That calls into some regular administrative backend that keeps track of mock calls
  - Use template magic to synthesize a unique function for putting there... and some horrible casting to force it in
- **Return a handle to a call object referencing this instance of a call**

.With(The, Right, Arguments)

- **Store arguments received in a tuple**
- **Be able to compare a Tuple<As...> to Tuple<Bs...> for sufficiently-equal.**
  - Takes another 20 slides, so not now.

- **Validate at end of test**
  - Require you to call a function
    - But that’s error prone
  - Require integration in your test framework
    - But that’s unnecessarily tight coupling

- **Validate on destruct of MockRepository**
  - If any exception has occurred, the test fails & we don’t mention unsatisfied pending calls.
    - `std::uncaught_exception`. May be the only valid use of it ever – arguably. Although even here `std::uncaught_exceptions` would be better.
  - Otherwise throw an exception if any expectation was not met.

Net result

- **You can test any code without creating lots of DRY violating code**
  - Excellent maintainability
  - Catches many bugs by design
- **It works cross platform**
- **It’s reliable**

- **It would be great if SG7 could make some form of standardese to allow this without C++ UB…**

Mocking C++ made simple

- **Each of these slide omits a bunch of corner cases**

- **Use a mocking library that does this for you!**
  - HippoMocks ← That’s mine, C++03 and up
  - Trompe l’Oeil ← C++14+ only, but cleaner