

MOCKINGBIRD

C++ Mocking Framework

Mockingbird

Mockingbird is a lightweight, simple and powerful C++ mocking framework.

Mockingbird is a **meta-programmed** framework that augments the mocks it creates with mechanisms to **inject** new method behaviors dynamically at runtime.

Mockingbird shows how expressive C++ and its Preprocessor are.

Why Mockingbird

- A very lightweight mocking framework of about a **100 LoC header**.
- Easy to **integrate**, just include the tiny header.
- Can be **integrated** with any testing framework like Catch2, gtest and Microsoft Unit Testing Framework.
- No **macro** usage after defining the class mock, changing behaviour is done via an injection method call.
- **Cross-Platform**.

Author: Mouaz Chamieh

E-Mail: mouaz.chamieh@gmail.com

License: MIT

Repository: <https://github.com/muazsh/Mockingbird>



Features

- Mock virtual methods and hide non-virtual ones
- Mock overloaded and **const** methods
- Call Once mechanism
- Mock class template
- Spy on original methods
- Count number of calls
- Multiple inheritance mock

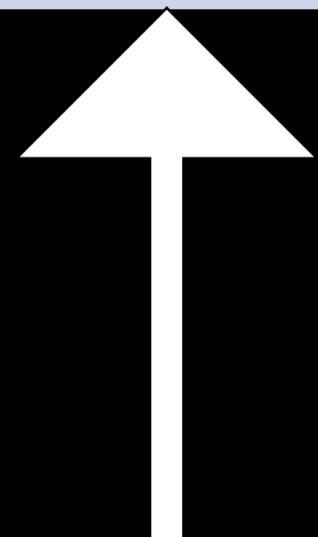
How does Mockingbird work:

- The mock class publicly inherits from the mocked class.
- For each mocked function, Mockingbird:
 - Defines an inner class inside the mock class with:
 - A data member of type function (a placeholder for injection).
 - A list of type function (a placeholder for Call Once mechanism)
 - Defines a data member of that inner class inside the mock class.
 - Defines the function override/hide inside the mock class with its implementation, which calls the functions inside the inner class.
 - Defines utilities to inject new behaviour and to retrieve the call counter.

Generated Structure

Target Class

```
public:  
virtual void DoSomething(int x, int y);
```



Mock Class

DoSomething

T

```
public:  
T m_Func;  
std::list<T> m_CallOnce;
```

```
DoSomething m_DoSomething;  
mutable unsigned m_DoSomethingCallCounter = 0;  
  
public:  
void InjectDoSomething(T substitute);  
void InjectDoSomethingCallOnce(T callOnce);  
int GetDoSomethingCallCounter();  
  
void DoSomething(int x, int y) override;
```

Example

```
struct Point {
    int m_X = 0;
    int m_Y = 0;
    int m_Z = 0;
};

// A class to be mocked
class IPointService {
public:
    virtual Point CreatePoint(int x, int y) = 0;
    virtual Point CreatePoint(int x, int y, int z) const = 0;
    virtual ~IPointService() {}
};

// Mock fixture
START_MOCK(PointServiceMock, IPointService)
FUNCTION(CreatePoint, Point, (int x, int y), , override, return
        Point{}, x, y)
FUNCTION_OVERLOAD(CreatePoint, Point, (int x, int y, int z),
                  const, override, return Point{}, 3D, x, y, z)
END_MOCK(PointServiceMock)

// Injecting and Testing
int main() {
    PointServiceMock pointService;

    auto create2DPointShifted =
        [] (int x, int y) -> Point { return Point{ x + 5, y + 5 }; };
    pointService.InjectCreatePoint(create2DPointShifted);
    auto point2D = pointService.CreatePoint(0, 0);
    assert(5 == point2D.m_X);
    assert(5 == point2D.m_Y);
    assert(0 == point2D.m_Z);

    auto create3DPointShifted =
        [] (int x, int y, int z) -> Point { return Point{ x + 5, y +
5, z + 5 }; };
    pointService.InjectCreatePoint(create3DPointShifted);
    auto point3D = pointService.CreatePoint(0, 0, 0);
    assert(5 == point3D.m_X);
    assert(5 == point3D.m_Y);
    assert(5 == point3D.m_Z);
}
```

