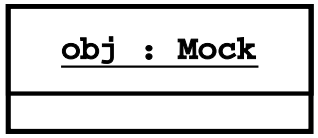


Ceci n'est pas un obj ;

Trompeloeil cheat sheet for implementing mock functions and placing expectations on them.

Mock implement member functions.

non-const member function	*const member function*
`MAKE_MOCKn(name, sig{, spec})`	`MAKE_CONST_MOCKn(name, sig{, spec})`

Place expectations. *Matching expectations are searched from youngest to oldest. Everything is illegal by default.*

Anonymous local object

```
REQUIRE_CALL(obj, func(params))
ALLOW_CALL(obj, func(params))
FORBID_CALL(obj, func(params))
```

std::unique_ptr<expectation>

```
NAMED_REQUIRE_CALL(obj, func(params))
NAMED_ALLOW_CALL(obj, func(params))
NAMED_FORBID_CALL(obj, func(params))
```

Refine expectations.

When to match

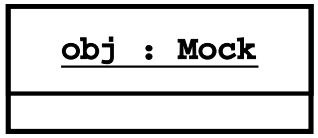
```
.IN_SEQUENCE(s...)
.TIMES(min {, max} )
```

Impose an ordering relation between expectations by using **`sequence`** objects

Define how many times an expectation must match. Default is 1. Conveniency arguments are **`AT_MOST(x)`** and **`AT_LEAST(x)`**

Parameters are _1 .. _15

Local objects are const copies		*Local objects are non-const references*
`.WITH(condition)`	← when to match →	`.LR_WITH(condition)`
`.SIDE_EFFECT(statement)`	← What to do when matching →	`.LR_SIDE_EFFECT(statement)`
`.RETURN(expression)`	← What to do when matching →	`.LR_RETURN(expression)`
`.THROW(expression)`	← What to do when matching →	`.LR_THROW(expression)`



Trompeloeil cheat sheet for matchers and object life time management.

Ceci n'est pas un obj

Matchers. Substitute for values in parameter list of expectations.

Any type allowing op		*Disambiguated type*
_	any value	ANY(type)
eq(mark)	value == mark	eq<type>(mark)
ne(mark)	value != mark	ne<type>(mark)
lt(mark)	value < mark	lt<type>(mark)
le(mark)	value <= mark	le<type>(mark)
gt(mark)	value > mark	gt<type>(mark)
ge(mark)	value >= mark	ge<type>(mark)
re(mark, ...)	match regular expression /mark/	re<type>(mark, ...)

Use **operator*** to dereference pointers. E.g. ***ne(mark)** means parameter is pointer (like) and ***parameter != mark**.

Object life time management

auto obj = new deathwatched<my_mock_type>(params);

***obj** destruction only allowed when explicitly required. Inherits from **my_mock_type**

Anonymous local object
REQUIRE_DESTRUCTION(*obj)

std::unique_ptr<expectation>
NAMED_REQUIRE_DESTRUCTION(*obj)

When to match
.IN_SEQUENCE(s...)

Impose an ordering relation between expectations by using **sequence** objects