

```
obj : Mock
```

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

Ceci n'est pas une objet

Mock implement member functions.

non-const member function

MAKE MOCKn(name, sig{, spec})

const member function

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. Convenience arguments are **AT_MOST(x)** and **AT_LEAST(x)**

Local objects are const copies

.WITH(condition)

Parameters are _1 .. _15

← when to match →

Local objects are non-const references

.LR_WITH(condition)

.SIDE_EFFECT(statement)

.RETURN(expression)

.THROW(expression)

← What to do when matching →

.LR_SIDE_EFFECT(statement)

.LR_RETURN(expression)

.LR_THROW(expression)

obj : Mock

Trompeloeil cheat sheet for matchers and object life time management.

Ceci n'est pas une objet

Matchers. Substitute for values in parameter list of expectations.

Any type allowing op

—
eq(mark)
ne(mark)
lt(mark)
le(mark)
gt(mark)
ge(mark)
re(mark, ...)



any value

value

==

mark

value

!=

mark

value

<

mark

value

<=

mark

value

>

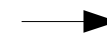
mark

value

>=

mark

match regular
expression /mark/



Disambiguated type

ANY(type)
eq<type>(mark)
ne<type>(mark)
lt<type>(mark)
le<type>(mark)
gt<type>(mark)
ge<type>(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<lifetime_monitor>

NAMED_REQUIRE_DESTRUCTION(*obj)

When to match

.IN_SEQUENCE(s...)



Impose an ordering relation between expectations by using
sequence objects