

Backporting to the Future

Andrew Paxie

Pacific++, 18-19 October 2018

"Ex Ignorantia Ad Sapientiam; Ex Luce Ad Tenebras"

Outline

Introduction

- Motivation

- Trompeloeil

- Backporting Goals

C++14 Features Used

- Library

- Language

Backporting to C++11

- Library

- Language

- Issues

Summary

Outline

Introduction

Motivation

Trompeloeil

Backporting Goals

C++14 Features Used

Library

Language

Backporting to C++11

Library

Language

Issues

Summary

Motivation

Project needed more unit tests.

Catch was the unit testing framework.

What would be the mocking framework?

Motivation

Project needed more unit tests.

Catch was the unit testing framework.

What would be the mocking framework?

Requirements for a mock object framework:

- ▶ Integrates with Catch
- ▶ Header-only library
- ▶ Portable: desktop and embedded
- ▶ Thread-safe
- ▶ Supports C++11

Alternatives

- ▶ Google Mock
- ▶ Turtle
- ▶ HippoMocks
- ▶ Fakelt
- ▶ Mockator
- ▶ Trompeloel
- ▶ Upgrade compiler to C++14
- ▶ Avoid mocking frameworks in unit tests

Outline

Introduction

Motivation

Trompeloeil

Backporting Goals

C++14 Features Used

Library

Language

Backporting to C++11

Library

Language

Issues

Summary

Trompeloeil

A header only C++ mocking framework

<https://github.com/rollbear/trompeloeil>



Björn Fahlner
(Code Owner)

<mailto:bjorn@fahller.se>

<https://github.com/rollbear>

<https://playfulprogramming.blogspot.com>



Andrew Paxie
(Contributor)

<mailto:cpp.scribe@gmail.com>

<https://github.com/AndrewPaxie>

<https://blog.andrew.paxie.org>

Trompe l'oeil



(c) 2006, Lee Snider / Alamy Stock Photo

Trompe l'oeil

Paintings that create the illusion of a real object or scene...



(c) 2006, Lee Snider / Alamy Stock Photo

Trompeloeil

Main features:

- ▶ Mock functions
- ▶ Expectations
- ▶ Matchers
- ▶ Modifiers

Trompeloeil

Main features:

- ▶ Mock functions
- ▶ Expectations
- ▶ Matchers
- ▶ Modifiers

Other features:

- ▶ Sequencing expectations
- ▶ Object-lifetime monitoring
- ▶ Integration into test framework reporting
- ▶ Tracing

Trompeloeil

Mock classes and functions

Interface

```
struct Interface
{
    virtual void setValue(int v) = 0;
    virtual int getValue() const = 0;
};
```

Trompeloeil

Mock classes and functions

Interface

```
struct Interface
{
    virtual void setValue(int v) = 0;
    virtual int getValue() const = 0;
};
```

Mock class

```
struct Mock: Interface
{
    MAKE MOCK1(setValue, void(int), override);
    MAKE_CONST MOCK0(getValue, int(), override);
};
```

Trompeloeil

Mock classes and functions

Interface

```
struct Interface
{
    virtual void setValue(int v) = 0;
    virtual int getValue() const = 0;
};
```

Mock class

```
struct Mock: Interface          mock functions ----+
{
    MAKE MOCK1(setValue, void(int), override);    <-+
    MAKE_CONST MOCK0(getValue, int(), override);  <-+
};
```

Trompeloeil

Example

```
struct Interface
{
    virtual void setValue(int v) = 0;
    virtual int  getValue() const = 0;
};
```

```
struct Mock: Interface
{
    MAKE MOCK1(setValue, void(int), override);
    MAKE_CONST MOCK0(getValue, int(), override);
};
```

```
TEST_CASE("Unit test", "[Sample]")
{ // Setup

    Mock obj;

    REQUIRE_CALL(obj, setValue(1+7))
        .WITH(obj.getValue() == 3)
        .RETURN(_1 + 2);
    // ...
}
```


Trompeloeil

Example

```
struct Interface
{
    virtual void setValue(int v) = 0;
    virtual int  getValue() const = 0;
};
```

```
struct Mock: Interface
{
    MAKE MOCK1(setValue, void(int), override);
    MAKE_CONST MOCK0(getValue, int(), override);
};
```

```
TEST_CASE("Unit test", "[Sample]")
{ // Setup

    Mock obj;

    REQUIRE_CALL(obj, setValue(1+7)) <- expectation
        .WITH(obj.getValue() == 3)
        .RETURN(_1 + 2);
    // ...
}
```

Trompeloeil

Example

```
struct Interface
{
    virtual void setValue(int v) = 0;
    virtual int  getValue() const = 0;
};
```

```
struct Mock: Interface
{
    MAKE MOCK1(setValue, void(int), override);
    MAKE_CONST MOCK0(getValue, int(), override);
};
```

```
TEST_CASE("Unit test", "[Sample]")
{ // Setup
    +----- mock object
    |
    |
    |
Mock obj; |
          v
    REQUIRE_CALL(obj, setValue(1+7)) <- expectation
        .WITH(obj.getValue() == 3)
        .RETURN(_1 + 2);
    // ...
}
```

Trompeloeil

Example

```
struct Interface
{
    virtual void setValue(int v) = 0;
    virtual int  getValue() const = 0;
};
```

```
struct Mock: Interface
{
    MAKE MOCK1(setValue, void(int), override);
    MAKE_CONST MOCK0(getValue, int(), override);
};
```

```
TEST_CASE("Unit test", "[Sample]")
{ // Setup

    Mock obj;

    +----- mock object
    |      +----- mock function
    |      |
    |      |
    |      |
    v      v

    REQUIRE_CALL(obj, setValue(1+7)) <- expectation
        .WITH(obj.getValue() == 3)
        .RETURN(_1 + 2);
    // ...
}
```

Trompeloeil

Example

```
struct Interface
{
    virtual void setValue(int v) = 0;
    virtual int  getValue() const = 0;
};
```

```
struct Mock: Interface
{
    MAKE MOCK1(setValue, void(int), override);
    MAKE_CONST MOCK0(getValue, int(), override);
};
```

```
TEST_CASE("Unit test", "[Sample]")
```

```
{ // Setup
```

```

                                +----- mock object
                                |
                                +----- mock function
                                |
                                |         +----- matcher
                                |         |
Mock obj;                       |         |
                                v         v         v
                                v         v         v
```

```
    REQUIRE_CALL(obj, setValue(1+7)) <- expectation
        .WITH(obj.getValue() == 3)
        .RETURN(_1 + 2);
    // ...
```

```
}
```

Trompeloeil

Example

```
struct Interface
{
    virtual void setValue(int v) = 0;
    virtual int  getValue() const = 0;
};
```

```
struct Mock: Interface
{
    MAKE MOCK1(setValue, void(int), override);
    MAKE_CONST MOCK0(getValue, int(), override);
};
```

```
TEST_CASE("Unit test", "[Sample]")
```

```
{ // Setup
```

```

                                +----- mock object
                                |
                                +----- mock function
                                |
                                |
                                +----- matcher
Mock obj;                       |
                                |
                                v
                                v
                                v
```

```
    REQUIRE_CALL(obj, setValue(1+7)) <- expectation
        .WITH(obj.getValue() == 3)    <- modifier
        .RETURN(_1 + 2);
    // ...
```

```
}
```

Trompeloeil

Example

```
struct Interface
{
    virtual void setValue(int v) = 0;
    virtual int  getValue() const = 0;
};
```

```
struct Mock: Interface
{
    MAKE MOCK1(setValue, void(int), override);
    MAKE_CONST MOCK0(getValue, int(), override);
};
```

```
TEST_CASE("Unit test", "[Sample]")
{ // Setup

    +----- mock object
    | +----- mock function
    | | +----- matcher
Mock obj; | | |
          v v v

    REQUIRE_CALL(obj, setValue(1+7)) <- expectation
        .WITH(obj.getValue() == 3)    <- modifier
        .RETURN(_1 + 2);              <- modifier
    // ...
}
```

Trompeloeil

Expectations

- ▶ REQUIRE_CALL(obj, func(params))
- ▶ ALLOW_CALL(obj, func(params))
- ▶ FORBID_CALL(obj, func(params))

```
TEST_CASE("Unit test", "[Sample]")
{
    // Setup
    Mock obj;

    REQUIRE_CALL(obj, setValue(ANY(int)));
    FORBID_CALL(obj, getValue());

    // ...
}
```

Also: Named variants of the above

Trompeloeil

Matchers

Matchers		Notes
-	ANY(type)	wildcards
eq(mark)	ne(mark)	equal, not equal
ge(mark)	le(mark)	[greater or less] than or equal
gt(mark)	lt(mark)	greater than, less than
re(mark, ...)		regular expression
!		negating matcher
*		pointer dereference

Trompeloeil

Modifiers

- ▶ `WITH(condition)`
- ▶ `SIDE_EFFECT(statement)`
- ▶ `RETURN(expression)`
- ▶ `THROW(expression)`

Parameters named using `_1 ... _15` placeholder variables

Also: local reference (`LR_`) versions of the above

Trompeloeil

Example

```
// Interface class
struct IDb
{
    virtual int
        lookup(const char*) = 0;
};
```

```
// Production class
struct Db: IDb
{
    int
        lookup(const char*)
        override;
};
```

```
// Mock class
struct MockDb final: IDb
{
    // Mock function
    MAKE MOCK1(
        lookup,
        int(const char*),
        final);
};
```

```
// System Under Test (SUT)
struct Engine
{
    explicit
        Engine(IDb& db_)
        : db(db_)
    {}

    int
        compute(
            const char* key)
    {
        return
            3 * db.lookup(key);
    }

private:
    IDb& db;
};
```

```
// Unit test
TEST_CASE(
    "Compute, key exists",
    "[Engine]")
{
    // Setup
    const char* key = "k";
    MockDb db;

    REQUIRE_CALL(
        db,
        lookup(eq(key)))
        .RETURN(2);

    Engine sut(db);
    // Exercise
    int ret =
        sut.compute(key);
    // Verify
    REQUIRE(ret == 6);
    // Teardown
}
```

Trompeloeil

Example

```
// Interface class
struct IDb
{
    virtual int
        lookup(const char*) = 0;
};
```

```
// Production class
struct Db: IDb
{
    int
        lookup(const char*)
        override;
};
```

```
// Mock class
struct MockDb final: IDb
{
    // Mock function
    MAKE MOCK1(
        lookup,
        int(const char*),
        final);
};
```

```
// System Under Test (SUT)
struct Engine
{
    explicit
        Engine(IDb& db_)
        : db(db_)
    {}

    int
        compute(
            const char* key)
    {
        return
            3 * db.lookup(key);
    }

private:
    IDb& db;
};
```

```
// Unit test
TEST_CASE(
    "Compute, key exists",
    "[Engine]")
{
    // Setup
    const char* key = "k";
    MockDb db;

    REQUIRE_CALL(
        db,
        lookup(eq(key)))
        .RETURN(2);

    Engine sut(db);
    // Exercise
    int ret =
        sut.compute(key);
    // Verify
    REQUIRE(ret == 6);
    // Teardown
}
```

Outline

Introduction

Motivation

Trompeloeil

Backporting Goals

C++14 Features Used

Library

Language

Backporting to C++11

Library

Language

Issues

Summary

Goals

- ▶ Provide existing API in C++11 mode
 - ▶ Expectations
 - ▶ Matchers
 - ▶ Modifiers
- ▶ Support same compilers, versions
- ▶ Preserve C++14 capability

Outline

Introduction

Motivation

Trompeloeil

Backporting Goals

C++14 Features Used

Library

Language

Backporting to C++11

Library

Language

Issues

Summary

C++14 Features Used: Library

<code>constexpr</code> for <code><tuple></code>	[N3471]
User-defined literals for <code><chrono></code> and <code><string></code>	[N3642]
Transformation Traits Redux, v2	[N3655]
<code>std::make_unique</code>	[N3656]
<code>std::integer_sequence</code>	[N3658]
<code>std::exchange</code>	[N3668]
Dual-range <code>std::equal</code>	[N3671]

C++14 Features Used: Library

<code>constexpr</code> for <code><tuple></code>	[N3471]
User-defined literals for <code><chrono></code> and <code><string></code>	[N3642]
Transformation Traits Redux, v2	[N3655]
<code>std::make_unique</code>	[N3656]
<code>std::integer_sequence</code>	[N3658]
<code>std::exchange</code>	[N3668]
Dual-range <code>std::equal</code> ,	[N3671]

Outline

Introduction

Motivation

Trompeloeil

Backporting Goals

C++14 Features Used

Library

Language

Backporting to C++11

Library

Language

Issues

Summary

C++14 Features Used: Language

`decltype(auto)`

[N3638]

Return type deduction for normal functions

[N3638]

Generalized lambda captures

[N3648]

Generic lambda expressions

[N3649]

Outline

Introduction

Motivation

Trompeloeil

Backporting Goals

C++14 Features Used

Library

Language

Backporting to C++11

Library

Language

Issues

Summary

Backporting: Library

Approach

- ▶ Define a `namespace` detail
 - ▶ Define C++11 versions of the C++14 API.

Backporting: Library

Approach

- ▶ Define a `namespace detail`
 - ▶ Define C++11 versions of the C++14 API.
- ▶ Call the `namespace detail` entities.
 - ▶ `std::make_unique` becomes `detail::make_unique`.

Backporting: Library

Approach

- ▶ Define a `namespace detail`
 - ▶ Define C++11 versions of the C++14 API.
- ▶ Call the `namespace detail` entities.
 - ▶ `std::make_unique` becomes `detail::make_unique`.
- ▶ For C++14 and later, make `std::` entities accessible in `namespace detail`.
 - ▶ Maybe a namespace alias: `namespace detail = std;`
 - ▶ Maybe using declarations in `namespace detail`
 - ▶ Maybe alias templates in `namespace detail`

Backporting: Library

<memory>

- ▶ `make_unique`

Stephan T. Lavavej [\[N3656\]](#)

<type_traits>

- ▶ `conditional_t`
- ▶ `decay_t`
- ▶ `enable_if_t`
- ▶ `remove_pointer_t`
- ▶ `remove_reference_t`

Walter E. Brown [\[N3655\]](#)

<utility>

- ▶ `exchange`
- ▶ `integer_sequence`
- ▶ `index_sequence`
- ▶ `make_integer_sequence`
- ▶ `make_index_sequence`
- ▶ `index_sequence_for`

Jeffrey Yasskin [\[N3668\]](#)

Jonathan Wakely [\[N3658\]](#)

Peter Dimov [\[Boost.mp11\]](#)

Backporting: Library

<memory>

- ▶ `make_unique`

Stephan T. Lavavej [\[N3656\]](#)

<type_traits>

- ▶ `conditional_t`
- ▶ `decay_t`
- ▶ `enable_if_t`
- ▶ `remove_pointer_t`
- ▶ `remove_reference_t`

Walter E. Brown [\[N3655\]](#)

<utility>

- ▶ `exchange`
- ▶ `integer_sequence`
- ▶ `index_sequence`
- ▶ `make_integer_sequence`
- ▶ `make_index_sequence`
- ▶ `index_sequence_for`

Jeffrey Yasskin [\[N3668\]](#)

Jonathan Wakely [\[N3658\]](#)

Peter Dimov [\[Boost.mp11\]](#)

Backporting: Library

<memory>

- ▶ `make_unique`

Stephan T. Lavavej [\[N3656\]](#)

<type_traits>

- ▶ `conditional_t`
- ▶ `decay_t`
- ▶ `enable_if_t`
- ▶ `remove_pointer_t`
- ▶ `remove_reference_t`

Walter E. Brown [\[N3655\]](#)

<utility>

- ▶ `exchange`
- ▶ `integer_sequence`
- ▶ `index_sequence`
- ▶ `make_integer_sequence`
- ▶ `make_index_sequence`
- ▶ `index_sequence_for`

Jeffrey Yasskin [\[N3668\]](#)

Jonathan Wakely [\[N3658\]](#)

Peter Dimov [\[Boost.mp11\]](#)

Outline

Introduction

Motivation

Trompeloeil

Backporting Goals

C++14 Features Used

Library

Language

Backporting to C++11

Library

Language

Issues

Summary

Backporting: Language

- ▶ Generic lambda expressions
- ▶ Generalized lambda captures
- ▶ Return type deduction for normal functions
- ▶ `decltype(auto)`

Generic Lambda Expressions

Definition

- ▶ Lambdas that use `auto` in their parameter specifications

C++14

```
[] (auto x)
{
    return x + x;
}
```

Generic Lambda Expressions

Definition

- ▶ Lambdas that use `auto` in their parameter specifications
- ▶ In C++14, lambdas may also be variadic

C++14

```
[] (auto x)
{
    return x + x;
}
```

C++14

```
[] (auto&&... xs)
{
    return sum(
        std::forward<
            decltype(xs)
        >(xs)...);
}
```

Replace Generic Lambdas

Approach

- ▶ Replace generic lambda with lambda
- ▶ Use a *functor* (*FunctionObject*) with function call operator member template (*generic functor*)
- ▶ Replace functions returning generic lambda with generic functor

Replace Generic Lambdas

Example

C++14

```
[] (auto x)
{
    return x + x;
}
```

Replace Generic Lambdas

Example

C++14

```
[] (auto x)
{
    return x + x;
}
```

C++11/14

```
class ClosureType
{
public:
    template <typename T>
    auto
    operator()(T x) const
    {
        return x + x;
    }
};
```


Replace Generic Lambdas

Trompeloeil

C++14

```
[](  
    std::ostream& os,  
    auto const& value)  
{  
    os << " == ";  
    print(os, value);  
}
```

Replace Generic Lambdas

Trompeloil

C++14

```
[](  
    std::ostream& os,  
    auto const& value)  
{  
    os << " == ";  
    print(os, value);  
}
```

C++11/14

```
struct equal_printer  
{  
    template <typename T>  
    void  
    operator()(  
        std::ostream& os,  
        T const& value)  
    const  
    {  
        os << " == ";  
        print(os, value);  
    }  
};
```

Generalized Lambda Capture

Definition

An *init-capture* may specify

- ▶ A name of the data member in the closure type
- ▶ An expression to initialize that data member

Generalized Lambda Capture

Definition

An *init-capture* may specify

- ▶ A name of the data member in the closure type
- ▶ An expression to initialize that data member

C++14

```
auto p = std::make_unique<std::vector<int>>();  
  
[ptr = std::move(p)]  
{  
    return ptr->empty();  
}
```

Replace Generalized Lambda Capture

Approaches

Create a functor like `ClosureType`

- ▶ Declare member variables
- ▶ Define constructor to initialize members

Replace Generalized Lambda Capture

Approaches

Create a functor like `ClosureType`

- ▶ Declare member variables
- ▶ Define constructor to initialize members

Use `std::bind` [Meyers², Item 32]

- ▶ Move object to be captured into a function object produced by `std::bind`
- ▶ Give the lambda a reference to the captured object

Replace Generalized Lambda Capture

Example

C++14

```
auto p =  
    std::make_unique<  
        std::vector<int>>();  
  
[ptr = std::move(p)]  
{  
    return ptr->empty();  
}
```

Replace Generalized Lambda Capture

Example

C++14

```
auto p =
    std::make_unique<
        std::vector<int>>>();

[ptr = std::move(p)]
{
    return ptr->empty();
}
```

C++11/14

```
struct ClosureType
{
    using T = std::unique_ptr<
        std::vector<int>>>;

    explicit ClosureType(T&& p)
        : ptr(std::move(p))
    {}

    bool operator()() const
    { return ptr->empty(); }

private:
    T ptr;
};
```


Replace Generalized Lambda Capture

Trompeloeil

C++14

```
inline auto
regex_check(std::regex&& re)
{
    return [re = std::move(re)](
        /* args */)
    -> decltype(std::regex_search(
        /* args */,
        re))
    {
        return std::regex_search(
            /* args */,
            re);
    }
}
```

Replace Generalized Lambda Capture

Trompeloil

C++14

```
inline auto
regex_check(std::regex&& re)
{
    return [re = std::move(re)](
        /* args */)
    -> decltype(std::regex_search(
        /* args */,
        re))
    {
        return std::regex_search(
            /* args */,
            re);
    }
}
```

C++11/14

```
struct regex_check
{
    regex_check(std::regex&& re_)
        : re(std::move(re_))
    {}

    template <typename T>
    bool
    operator()(/* args */) const
    {
        return std::regex_search(
            /* args */,
            re);
    }

    std::regex_check re;
};
```

Return Type Deduction

Definition

Jason Merrill, [N3638]:

Write `auto` on your function declaration and have the return type deduced

C++14

```
auto foo(int var)
{
    if (var)
    {
        return 0;
    }
    else
    {
        return var + 1;
    }
}
```

Replace Return Type Deduction

Approach

Use trailing return type

Replace Return Type Deduction

Example

C++14

```
auto  
foo(int var)  
{  
    if (var)  
    {  
        return 0;  
    }  
    else  
    {  
        return var + 1;  
    }  
}
```

Replace Return Type Deduction

Example

C++14

```
auto
foo(int var)
{
    if (var)
    {
        return 0;
    }
    else
    {
        return var + 1;
    }
}
```

C++11/14

```
auto
foo(int var)
-> int
{
    if (var)
    {
        return 0;
    }
    else
    {
        return var + 1;
    }
}
```

Replace Return Type Deduction

Trompeloeil

C++14

```
template <
    typename Kind = wildcard
>
auto
re(
    std::string s,
    match_flag_type match_type)
{
    return make_matcher<Kind>(
        lambdas::regex_check(
            std::regex(s), match_type),
        lambdas::regex_printer(),
        std::move(s));
}
```

Replace Return Type Deduction

Trompeloeil

C++14

```
template <
    typename Kind = wildcard
>
auto
re(
    std::string s,
    match_flag_type match_type)
{
    return make_matcher<Kind>(
        lambdas::regex_check(
            std::regex(s), match_type),
        lambdas::regex_printer(),
        std::move(s));
}
```

C++11/14

```
template <
    typename Kind = wildcard,
    typename R = /* omitted */
>
auto
re(
    std::string s,
    match_flag_type match_type)
-> decltype(R)
{
    return make_matcher<Kind>(
        lambdas::regex_check(
            std::regex(s), match_type),
        lambdas::regex_printer(),
        std::move(s));
}
```


decltype(auto)

Definition

Use the rules of `decltype()` to deduce a type.

Jason Merrill, [N3638]:

Plain `auto` never deduces to a reference, and `auto&` always deduces to a reference. [...] forwarding functions can't use `auto`.

Replace `decltype(auto)`

Approach

- ▶ Replace `decltype(auto)` with explicit type
- ▶ Use `auto` and trailing return type

Replace `decltype(auto)`

Example: Use explicit type

From macro `RETURN(...)`:

C++14

```
handle_return(  
    [&](auto& x)  
    -> decltype(auto)  
    {  
        // Define placeholders  
        // _1 ... _15 from x  
  
        return __VA_ARGS__;  
    })
```

Replace `decltype(auto)`

Example: Use explicit type

From macro `RETURN(...)`:

C++14

```
handle_return(  
    [&](auto& x)  
    -> decltype(auto)  
    {  
        // Define placeholders  
        // _1 ... _15 from x  
  
        return __VA_ARGS__;  
    })
```

C++11/14

```
handle_return(  
    [&](/* :: */& x)  
    -> return_of_t  
    {  
        // Define placeholders  
        // _1 ... _15 from x  
  
        return __VA_ARGS__;  
    })
```

Replace `decltype(auto)`

Example: Use `auto` and trailing return type

From placeholder naming code (simplified):

C++14

```
template <
    int N,
    typename T
>
constexpr
decltype(auto)
arg(
    T* t,
    std::true_type)

{
    return std::get<N-1>(*t);
}
```

Replace `decltype(auto)`

Example: Use `auto` and trailing return type

From placeholder naming code (simplified):

C++14

```
template <
    int N,
    typename T
>
constexpr
decltype(auto)
arg(
    T* t,
    std::true_type)
{
    return std::get<N-1>(*t);
}
```

C++11/14

```
template <
    int N,
    typename T
>
constexpr
auto
arg(
    T* t,
    std::true_type)
-> decltype(
    std::get<N-1>(*t))
{
    return std::get<N-1>(*t);
}
```

Outline

Introduction

Motivation

Trompeloeil

Backporting Goals

C++14 Features Used

Library

Language

Backporting to C++11

Library

Language

Issues

Summary

Issue: Expectation API Changed

Consider the definition of an expectation:

```
REQUIRE_CALL(...)
```

```
.
```

```
RETURN(...)
```


Issue: Expectation API Changed

Consider the definition of an expectation:

`REQUIRE_CALL(...)` an expression yielding an object

.

`RETURN(...)`

Issue: Expectation API Changed

Consider the definition of an expectation:

`REQUIRE_CALL(...)` an expression yielding an object

`.` is the member access operator
(cannot overload)

`RETURN(...)`

Issue: Expectation API Changed

Consider the definition of an expectation:

`REQUIRE_CALL(...)` an expression yielding an object

`.` is the member access operator
(cannot overload)

`RETURN(...)` yields a member function call
invoked with an argument

Issue: Expectation API Changed

Consider the definition of an expectation:

`REQUIRE_CALL(...)` an expression yielding an object

`.` is the member access operator
 (cannot overload)

`RETURN(...)` yields a member function call
 invoked with an argument

The result is still an expression, for more modifiers to be added.

Issue: Have not found a way, so far, to transmit type information between expectation macros and modifiers.

Issue: Expectation API Changed

Changed API to allow access to type. A subtle change:

```
REQUIRE_CALL(obj, foo(_))  
  .WITH(_1 == 42);
```

```
REQUIRE_CALL_V(obj, foo(_),  
  .WITH(_1 == 42));
```

_V is for 'variadic'

Issue: Expectation API Changed

Changed API to allow access to type. A subtle change:

```
REQUIRE_CALL(obj, foo(_))  
    .WITH(_1 == 42);
```

```
REQUIRE_CALL_V(obj, foo(_),  
    .WITH(_1 == 42));
```

`_V` is for 'variadic'

- ▶ API didn't change once but three times.
- ▶ Two APIs are available in C++14 mode or later.
- ▶ Can incrementally transform from C++11 API to C++14 API in unit tests.

Issue: RETURN(...) Macro doesn't Quite Work

- ▶ C++14 API provides type check of expression and gives helpful error messages, via `static_assert`.
- ▶ C++11 API performs implicit conversion of `__VA_ARGS__` to `return_of_t` **before** type checking.

Result: Less helpful compiler error messages in C++11 mode.

Issue: Compilers Have Defects Too

Other limitations because of compiler defects:

- ▶ `<regex>` partial implementation ([PR61582](#))
- ▶ Ambiguous conversion operator error
 - ▶ negating matcher (!)
 - ▶ wildcard matcher (_)
- ▶ `<tuple>` broken ([PR61947](#))

These just for GCC 4.8.3.

Backporting Goals

Review

- ▶ Provide existing API in C++11 mode
 - ▶ Expectations
 - ▶ Matchers
 - ▶ Modifiers
- ▶ Support same compilers, versions
- ▶ Preserve C++14 capability

Backporting Goals

Review

- ▶ Provide existing API in C++11 mode
 - ▶ Expectations ✗
 - ▶ Matchers
 - ▶ Modifiers
- ▶ Support same compilers, versions
- ▶ Preserve C++14 capability

Backporting Goals

Review

- ▶ Provide existing API in C++11 mode
 - ▶ Expectations ✗
 - ▶ Matchers ✓/✗
 - ▶ Modifiers
- ▶ Support same compilers, versions
- ▶ Preserve C++14 capability

Backporting Goals

Review

- ▶ Provide existing API in C++11 mode
 - ▶ Expectations ✗
 - ▶ Matchers ✓/✗
 - ▶ Modifiers ✓/✗
- ▶ Support same compilers, versions
- ▶ Preserve C++14 capability

Backporting Goals

Review

- ▶ Provide existing API in C++11 mode
 - ▶ Expectations ✗
 - ▶ Matchers ✓/✗
 - ▶ Modifiers ✓/✗
- ▶ Support same compilers, versions ✓
- ▶ Preserve C++14 capability

Backporting Goals

Review

- ▶ Provide existing API in C++11 mode
 - ▶ Expectations ✗
 - ▶ Matchers ✓/✗
 - ▶ Modifiers ✓/✗
- ▶ Support same compilers, versions ✓
- ▶ Preserve C++14 capability ✓

Backporting to the Future

How do I port a C++11 program to C++14?

Backporting to the Future

How do I port a C++11 program to C++14?

Reverse the direction of these code transformations
and

Use library and and language features that are also part of C++14
(see [Appendix](#))

Trompeloeil

<https://github.com/rollbear/trompeloeil>



Björn Fahlner
(Code Owner)

<mailto:bjorn@fahller.se>

<https://github.com/rollbear>

<https://playfulprogramming.blogspot.com>



Andrew Paxie
(Contributor)

<mailto:cpp.scribe@gmail.com>

<https://github.com/AndrewPaxie>

<https://blog.andrew.paxie.org>

Appendix: Unused Features from C++14

Library (1)

<code>constexpr</code> for <code><complex></code>	[N3302]
Making operator functors <code>greater<></code>	[N3421]
<code>std::result_of</code> and SFINAE	[N3462]
<code>constexpr</code> for <code><chrono></code>	[N3469]
<code>constexpr</code> for <code><array></code>	[N3470]
Improved <code>std::integral_constant</code>	[N3545]

Appendix: Unused Features from C++14

Library (2)

Null forward iterators	[N3644]
<code>std::quoted</code>	[N3654]
Heterogeneous associative lookup	[N3657]
Shared locking in C++	[N3659]
Fixing <code>constexpr</code> member functions without <code>const</code>	[N3669]
<code>std::get<T></code>	[N3670]

Appendix: Unused Features from C++14

Language

Binary literals	[N3472]
Variable templates	[N3651]
Extended <code>constexpr</code>	[N3652]
Member initializers and aggregates	[N3653]
<code>[[deprecated]]</code> attribute	[N3760]
Single quote as digit separator	[N3781]

Appendix: Unused Features from C++14

Miscellaneous

Tweak to certain contextual conversions

[[N3323](#)]

Clarifying memory allocation

[[N3664](#)]

Sized deallocation

[[N3778](#)]

References

Motivation



Catch,

Available: <https://github.com/catchorg/catch2>

Accessed: 13 October 2018

References

Alternatives (1)



Google Mock

Available: <https://github.com/google/googletest>

Accessed: 10 October 2018



Turtle

Available: <https://github.com/mat007/turtle>

Accessed: 10 October 2018



HippoMocks

Available: http://hippomocks.com/Main_Page

Available: <https://github.com/dascandy/hippomocks>

Accessed: 10 October 2018

References

Alternatives (2)



Fakelt

Available: <https://github.com/eranpeer/FakeIt>

Accessed: 10 October 2018



Mockator

Available: <https://www.cevelop.com/>

Accessed: 10 October 2018



Trompeloeil

Available: <https://github.com/rollbear/trompeloeil>

Accessed: 15 October 2018

References

Trompe l'oeil



Trompe l'oeil definition

Available: [https:](https://www.tate.org.uk/art/art-terms/t/trompe-loeil)

[//www.tate.org.uk/art/art-terms/t/trompe-loeil](https://www.tate.org.uk/art/art-terms/t/trompe-loeil)

Accessed: 13 October 2018

References

C++14 Features Used: Library (1)

-  Benjamin Kosnik and Daniel Krügler,
“Constexpr Library Additions: utilities, v3,” N3471, 18
October 2012.
Available: <http://wg21.link/n3471>
Accessed: 22 September 2018
-  Peter Sommerlad,
“User-defined Literals for Standard Library Types (part 1 -
version 4),” N3642, 18 April 2013.
Available: <http://wg21.link/n3642>
Accessed: 22 September 2018
-  Walter E. Brown,
“TransformationTraits Redux, v2,” N3655, 18 April 2013.
Available: <http://wg21.link/n3655>
Accessed: 22 September 2018

References

C++14 Features Used: Library (2)



Jonathan Wakely,

“Compile-time integer sequences,” N3658, 18 April 2013.

Available: <http://wg21.link/n3658>

Accessed: 22 September 2018



Stephan T. Lavavej,

“make_unique (Revision 1),” N3656, 18 April 2013.

Available: <http://wg21.link/n3656>

Accessed: 22 September 2018

References

C++14 Features Used: Library (3)



Jeffrey Yasskin,

“exchange() utility function, revision 3,” N3668, 19 April 2013.

Available: <http://wg21.link/n3668>

Accessed: 22 September 2018



Mike Spertus and Attila Pall,

“Making non-modifying sequence operations more robust: Revision 2,” N3671, 19 April 2013.

Available: <http://wg21.link/n3671>

Accessed: 22 September 2018

References

C++14 Features Used: Language (1)



Walter E. Brown,

“A Proposal to Tweak Certain C++ Contextual Conversions, v3,” N3323, 8 December 2012.

Available: <http://wg21.link/n3323>

Accessed: 22 September 2018



Jason Merrill,


“Return type deduction for normal functions,” N3638, 17 April 2013.

Available: <http://wg21.link/n3638>

Accessed: 22 September 2018


References

C++14 Features Used: Language (2)

-  Daveed Vandevor and Ville Voutilainen,
“Wording Changes for Generalized Lambda-capture,” N3648,
17 April 2013.

Available: <http://wg21.link/n3648>

Accessed: 22 September 2018

-  Faisal Vali, Herb Sutter, and Dave Abrahams,
“Generic (Polymorphic) Lambda Expressions (Revision 3),”
N3649, 19 April 2013.

Available: <http://wg21.link/n3649>

Accessed: 22 September 2018

References

Generic lambdas



[Meyers[2], Item 33]

“Use `decltype` on `auto&&` parameters to `std::forward` them,” pp. 229-32.



[cppreference.com](https://en.cppreference.com),

“Lambda expressions,” last modified 19 September 2018.

Available:

<https://en.cppreference.com/w/cpp/language/lambda>

Accessed: 22 September 2018

References

Generalized lambda capture



[Meyers², Item 32]

“Use init capture to move objects into closures,” pp. 224-29.

References

Issue: Compilers Have Defects Too



GCC Bugzilla,

“Bug 61582 - C++11 regex memory corruption,” 23 June 2014.

Available:

https://gcc.gnu.org/bugzilla/show_bug.cgi?id=61582

Accessed: 9 November 2017



GCC Bugzilla,

“Bug 61947 - [4.8/4.9 Regression] Ambiguous calls when constructing std::tuple,” 29 July 2014.

Available:

https://gcc.gnu.org/bugzilla/show_bug.cgi?id=61947

Accessed: 8 November 2017




References

Unused Features from C++14: Library (1)(a)

-  Benjamin Kosnik, Gabriel Dos Reis, and Daniel Krügler, “Constexpr Library Additions: complex, v2,” N3302, 20 August 2011.
Available: <http://wg21.link/n3302>
Accessed: 22 September 2018
-  Stephan T. Lavavej, “Making Operator Functors greater<>,” N3421, 20 September 2012.
Available: <http://wg21.link/n3421>
Accessed: 22 September 2018
-  Eric Niebler, Daniel Walker, and Joel de Guzman, “std::result_of and SFINAE,” N3462, 18 October 2012.
Available: <http://wg21.link/n3462>
Accessed: 22 September 2018

References

Unused Features from C++14: Library (1)(b)

-  Benjamin Kosnik and Daniel Krügler,
“Constexpr Library Additions: chrono, v3,” N3469, 18 October 2012.
Available: <http://wg21.link/n3469>
Accessed: 22 September 2018
-  Benjamin Kosnik and Daniel Krügler,
“Constexpr Library Additions: containers, v2,” N3470, 18 October 2012.
Available: <http://wg21.link/n3470>
Accessed: 22 September 2018
-  Walter E. Brown,
“An Incremental Improvement to `integral_constant`,”
N3545, 12 March 2013.
Available: <http://wg21.link/n3545>
Accessed: 22 September 2018

References

Unused Features from C++14: Library (2)(a)



Alan Talbot,

“Null forward iterators,” N3644, 18 April 2013.

Available: <http://wg21.link/n3644>

Accessed: 22 September 2018



Berman Dawes,

“Quoted Strings Library Proposal (Revision 2),” N3654, 19 April 2013.

Available: <http://wg21.link/n3654>

Accessed: 22 September 2018



Jonathan Wakely, Stephan T. Lavavej, and Joaquín M^a López Muñoz,




“Adding heterogeneous comparison lookup to associative containers (rev 4),” N3657, 19 March 2013.

Available: <http://wg21.link/n3657>

Accessed: 22 September 2018

References

Unused Features from C++14: Library (2)(b)

-  Howard Hinnant, Detlef Vollmann, and Hans Boehm,
“Shared locking in C++, Revision 2,” N3659, 19 April 2013.
Available: <http://wg21.link/n3659>
Accessed: 22 September 2018
-  Nicolai Josuttis,
“Fixing constexpr member functions without `const`,” N3669,
19 April 2013.
Available: <http://wg21.link/n3669>
Accessed: 22 September 2018
-  Mike Spertus,
“Wording for Addressing Tuples by Type: Revision 2,” N3670,
19 April 2013.
Available: <http://wg21.link/n3670>
Accessed: 22 September 2018

References

Unused Features from C++14: Language (1)



James Dennett,

“Binary Literals in the C++ Core Language,” N3472, 19 October 2012.

Available: <http://wg21.link/n3472>

Accessed: 22 September 2018



Gabriel Dos Reis,

“Variable Templates (Revision 1),” N3651, 19 April 2013.

Available: <http://wg21.link/n3651>

Accessed: 22 September 2018



Richard Smith,




“Relaxing constraints on constexpr functions, constexpr member functions and implicit const,” N3652, 18 April 2013.

Available: <http://wg21.link/n3652>

Accessed: 22 September 2018



References

Unused Features from C++14: Language (2)

-  Ville Voutilainen and Richard Smith,
“Member initializers and aggregates,” N3653, 17 April 2013.
Available: <http://wg21.link/n3653>
Accessed: 22 September 2018
-  Alberto Ganesh Barbati,
“[[deprecated]] attribute,” N3760, 1 September 2013.
Available: <http://wg21.link/n3760>
Accessed: 22 September 2018
-  Lawrence Cowl, Richard Smith, Jeff Snyder, and Daveed Vandevoorde,
“Single-Quotation-Mark as a Digit Separator,” N3781, 25 September 2013.
Available: <http://wg21.link/n3781>
Accessed: 22 September 2018

References

Unused Features from C++14: Miscellaneous

-  Lawrence Crowl, Chandler Carruth, and Richard Smith, “Clarifying Memory Allocation,” N3664, 19 April 2013.
Available: <http://wg21.link/n3664>
Accessed: 22 September 2018
-  Lawrence Crowl, “C++ Sized Deallocation,” N3778, 27 September 2013.
Available: <http://wg21.link/n3778>
Accessed: 22 September 2018

References

Other



Gerard Meszaros,

“xUnit Test Patterns: Refactoring Test Code,”
Addison-Wesley, Upper Saddle River, NJ, 2007.



Scott Meyers,

“Effective Modern C++,” O’Reilly Media, Inc., Sebastopol,
CA, 2015.



Scott Meyers,

“Modification History and Errata List for ‘Effective Modern
C++’,” last updated 18 May 2018.

Available: [https:](https://www.aristeia.com/BookErrata/emc++-errata.html)

[//www.aristeia.com/BookErrata/emc++-errata.html](https://www.aristeia.com/BookErrata/emc++-errata.html)

Accessed: 23 September 2018

Acknowledgements

Special mention is given to the following individuals,

C++ Meetup
26 September 2018

Biswajit Banerjee
Christian Blume
Nick Sarten
Paul Leslie
Shane Powell
Toby Allsopp
Tom Isaacson

Group Presentation
27 September 2018

Benjamin Deeming
Jacqueline Lewis
Ralf Haeusler
Saeid Shahosseini

Library Testers

Ben Morelli
Bill McCroskey
Cooper Li
Greg Sumner
Edward Peek
Mike Hiatt
Sam McArdle
Shawn Crawford
Sian Phillips
Toby Collett
Vadim Fuchs
Walt Conley
Will Sember

Thank you all for your valuable feedback.

Image Credit



Photo by: Lee Snider / Alamy Stock Photo
Date taken: 22 July 2006
Image ID: E9RNMX
Available: <https://www.alamy.com>

Original wall mural
by Richard Haas, 1978,
depicting the Brooklyn Bridge
and 19th century storefronts
painted on a windowless warehouse
on Peck Slip.