accu 2021 VIRTUAL EVENT

Bloomberg Engineering

undo

mosaic CONSULTANTS TO FINANCIAL SERVICES

Future of Testing With C++20

Kris Jusiak

https://boost-ext.github.io/ut/accu-2021

kris@quantlab.com | https://www.quantlab.com/careers

Agenda

Agenda

- Motivation

Agenda

- Motivation
- Implementation (Simplified / C++20)

Agenda

- Motivation
- Implementation (Simplified / C++20)
- UT - Unit Testing Framework

Agenda

- Motivation
- Implementation (Simplified / C++20)
- UT - Unit Testing Framework
- Benchmarks

Agenda

- Motivation
- Implementation (Simplified / C++20)
- UT - Unit Testing Framework
- Benchmarks
- Summary

Agenda

- Motivation
- Implementation (Simplified / C++20)
- UT - Unit Testing Framework
- Benchmarks
- Summary

`darkblue background - something to remember` ✓

Agenda

- Motivation
- Implementation (Simplified / C++20)
- UT - Unit Testing Framework
- Benchmarks
- Summary

---

darkblue background - something to remember ✓

green background - achivement unlocked 👍

Agenda

- Motivation
- Implementation (Simplified / C++20)
- UT - Unit Testing Framework
- Benchmarks
- Summary

---

darkblue background - something to remember ✓

green background - achivement unlocked 

POWERED BY COMPILER EXPLORER

Motivation

Motivation

```
constexpr auto sum(auto... args) { return (args + ...); }
```

Motivation

```
constexpr auto sum(auto... args) { return (args + ...); }
```

```
int main() {
  // should sum numbers
  {
    assert(3 == sum(1, 2));
  }
}
```

Motivation - Problems

Motivation - Problems

- No ability to list/run specific tests

Motivation - Problems

- No ability to list/run specific tests
- No useful information when the test failed

Motivation - Problems

- No ability to list/run specific tests
- No useful information when the test failed
- Hard to scale/integrate/debug

Motivation - Problems

- No ability to list/run specific tests
- No useful information when the test failed
- Hard to scale/integrate/debug
- Easy to make mistakes (implicit conversions, ...)

Motivation - Problems

- No ability to list/run specific tests
- No useful information when the test failed
- Hard to scale/integrate/debug
- Easy to make mistakes (implicit conversions, ...)
- Hard to follow good practises such as TDD/BDD

Motivation - Problems

- No ability to list/run specific tests
- No useful information when the test failed
- Hard to scale/integrate/debug
- Easy to make mistakes (implicit conversions, ...)
- Hard to follow good practises such as TDD/BDD
- ...

Existing Solutions

Existing Solutions

- GoogleTest

Existing Solutions

- GoogleTest
- Boost.Test

Existing Solutions

- GoogleTest
- Boost.Test
- Catch2

Existing Solutions

- GoogleTest
- Boost.Test
- Catch2
- DocTest

Existing Solutions

- GoogleTest
- Boost.Test
- Catch2
- DocTest
- ...

Existing Solutions - Problems

Existing Solutions - Problems

* *Generalized*

Existing Solutions - Problems

- Macro based*

** Generalized*

Existing Solutions - Problems

- Macro based*
- Boilerplate*

** Generalized*

Existing Solutions - Problems

- Macro based*
- Boilerplate*
- Slow to compile*

** Generalized*

Existing Solutions - Problems

- Macro based*
- Boilerplate*
- Slow to compile*
- Hard to integrate*

** Generalized*

Existing Solutions - Problems

- Macro based*
- Boilerplate*
- Slow to compile*
- Hard to integrate*
- ...

** Generalized*

Future (C++20)?

Future (C++20)?

- No macros

Future (C++20)?

- No macros
- Minimal boilerpalte

Future (C++20)?

- No macros
- Minimal boilerpalte
- Minimal learning curve

Future (C++20)?

- No macros
- Minimal boilerpalte
- Minimal learning curve
- Easy integration

Future (C++20)?

- No macros
- Minimal boilerpalte
- Minimal learning curve
- Easy integration
- Flexible/Scalable

Future (C++20)?

- No macros
- Minimal boilerpalte
- Minimal learning curve
- Easy integration
- Flexible/Scalable
- Fast to compile/execute

The Goal

The Goal

```
constexpr auto sum(auto... args) { return (0 + ... + args); }
```

The Goal

```
constexpr auto sum(auto... args) { return (0 + ... + args); }
```

```
01 import ut;                  // C++20 module
02
03 int main() {
04   "sum"_test = [] {         // Running... sum
05     sum(1, 2) == 42_i;      // sum.cpp:5:FAILED [ 3 == 42 ]
06   };                        // tests:   1 | 1 failed
07 }                           // asserts: 1 | 0 passed | 1 failed
```

The Goal

```
constexpr auto sum(auto... args) { return (0 + ... + args); }
```

```
01 import ut;                  // C++20 module
02
03 int main() {
04   "sum"_test = [] {         // Running... sum
05     sum(1, 2) == 42_i;      // sum.cpp:5:FAILED [ 3 == 42 ]
06   };                        // tests:   1 | 1 failed
07 }                           // asserts: 1 | 0 passed | 1 failed
```

```
01 import ut;
02
03 suite sums = [] {
04  "sum with no args"_test        = [] { expect(sum() == 0_i); };
05  "sum with single arg"_test     = [] { expect(sum(42) == 42_i); };
06  "sum with multiple args"_test = [] { expect(sum(1, 2) == 3_i); };
07 };
08
09 int main() {                  // tests:   3 | 0 failed
10 }                            // asserts: 3 | 3 passed | 0 failed
```

Implementation*

** Simplified (~200 LOC) / C++20*

ut.module

ut.module

```
import ut;
```

ut.module

```
import ut;
```

---

ut.hpp

ut.module

```
import ut;
```

---

ut.hpp

```
export module ut; // module interface unit
```

ut.module

```
import ut;
```

---

ut.hpp

```
export module ut; // module interface unit
```

```
import std;
```

ut.module

```
import ut;
```

---

ut.hpp

```
export module ut; // module interface unit
```

```
import std;
```

```
export namespace ut::inline v1 {
  // ...
}
```

ut.module

ut.module

---

main.cpp

---

ut.module

main.cpp

```
01 import ut;
02
03 int main() {
04   // ...
05 }
```

ut.module

main.cpp

```
01 import ut;
02
03 int main() {
04   // ...
05 }
```

```
$CXX $CXXFLAGS -emit-module-interface -c ut.hpp -o ut.pcm
```

ut.module

main.cpp

```
01 import ut;
02
03 int main() {
04   // ...
05 }
```

```
$CXX $CXXFLAGS -emit-module-interface -c ut.hpp -o ut.pcm
```

```
$CXX $CXXFLAGS main.cpp ut.pcm -o main
```

ut.module

main.cpp

```
01 import ut;
02
03 int main() {
04   // ...
05 }
```

```
$CXX $CXXFLAGS -emit-module-interface -c ut.hpp -o ut.pcm
```

```
$CXX $CXXFLAGS main.cpp ut.pcm -o main
```

```
./main
 -> ✔
```

Modules / C++20 / C++draft/module

2 A *module interface unit* is a module unit whose *module-declaration* starts with *export-keyword*; any other module unit is a *module implementation unit*. A named module shall contain exactly one module interface unit with no *module-partition*, known as the *primary module interface unit* of the module; no diagnostic is required.

Modules / C++20 / C++draft/module

2 A *module interface unit* is a module unit whose *module-declaration* starts with *export-keyword*; any other module unit is a *module implementation unit*. A named module shall contain exactly one module interface unit with no *module-partition*, known as the *primary module interface unit* of the module; no diagnostic is required.

- "Similar" to Precompiled Headers (PCH)

Modules / C++20 / C++draft/module

2 A *module interface unit* is a module unit whose *module-declaration* starts with *export-keyword*; any other module unit is a *module implementation unit*. A named module shall contain exactly one module interface unit with no *module-partition*, known as the *primary module interface unit* of the module; no diagnostic is required.

- "Similar" to Precompiled Headers (`PCH`)
- Allow to hide implementation details (detail namespace)

Modules / C++20 / C++draft/module

2 A *module interface unit* is a module unit whose *module-declaration* starts with *export-keyword*; any other module unit is a *module implementation unit*. A named module shall contain exactly one module interface unit with no *module-partition*, known as the *primary module interface unit* of the module; no diagnostic is required.

- "Similar" to Precompiled Headers (`PCH`)
- Allow to hide implementation details (detail namespace)
- Faster compilation times (across translation units)

ut::test

ut::test

```
01 import ut;                  // ✔
02
03 int main() {
04   "sum"_test = [] {         //  ut::test / ""_test
```

ut::test - concept

ut::test - concept

```
template <class T, auto expr = []{}> concept test =
```

ut::test - concept

```
template <class T, auto expr = []{}> concept test =
  requires(T test) {




  };
```

ut::test - concept

```
template <class T, auto expr = []{}> concept test =
  requires(T test) {
    { test.name } -> printable;              // for "Running... test"


  };
```

ut::test - concept

```
template <class T> concept printable =
  requires(std::ostream &os, T t) {
    os << t;
  };
```

```
template <class T, auto expr = []{}> concept test =
```

```
  requires(T test) {
```

```
    { test.name } -> printable;              // for "Running... test"
```

```
  };
```

ut::test - concept

```
template <class T> concept printable =
  requires(std::ostream &os, T t) {
    os << t;
  };
```

```
template <class T, auto expr = []{}> concept test =

  requires(T test) {

    { test.name } -> printable;              // for "Running... test"

    { test = expr } -> std::same_as<void>; // for test = []{};

  };
```

Concepts / C++20 / C++draft/concepts

Concepts / C++20 / C++draft/concepts

- Type constraints (act like documentation for interfaces)

Concepts / C++20 / C++draft/concepts

- Type constraints (act like documentation for interfaces)
- Better error messages (Point Of Instantiation - POI)

Concepts / C++20 / C++draft/concepts

- Type constraints (act like documentation for interfaces)
- Better error messages (Point Of Instantiation - POI)
- Faster compilation times (In comparison to Substitution Failure is Not an Error - SFINAE)

Concepts / C++20 / C++draft/concepts

- Type constraints (act like documentation for interfaces)
- Better error messages (Point Of Instantiation - POI)
- Faster compilation times (In comparison to Substitution Failure is Not an Error - SFINAE)

```
template<class T>
concept Auto = true; // the least constraint concept (same as auto)
```

Concepts / C++20 / C++draft/concepts

- Type constraints (act like documentation for interfaces)
- Better error messages (Point Of Instantiation - POI)
- Faster compilation times (In comparison to Substitution Failure is Not an Error - SFINAE)

```
template<class T>
concept Auto = true; // the least constraint concept (same as auto)
```

EQUIVALENT SYNTAX

```
                                  auto foo(auto arg);
template<class T> requires Auto<T> auto foo(T arg);
template<class T>                 auto foo(T arg) requires Auto<T>;
template<Auto T>                  auto foo(T arg);
                                  auto foo(Auto auto arg);
```

ut::test

ut::test

```
struct test {
  std::string_view name{}; // test case name
```

```
};
```

ut::test

```
struct test {
  std::string_view name{}; // test case name
```

```
  auto operator=(std::invocable auto test) -> void {
    std::clog << "Running... " << name << '\n';
    test();
  }
```

```
};
```

---

ut::test

```
struct test {
  std::string_view name{}; // test case name
```

```
  auto operator=(std::invocable auto test) -> void {
    std::clog << "Running... " << name << '\n';
    test();
  }
```

```
};
```

```
[[nodiscard]] constexpr concepts::test auto operator""_test(
  const char* name, std::size_t size) {
  return test{.name = {name, size}};
}
```

ut::test

ut::test

```
01 import ut;                  // ✔
02
03 int main() {
04   "sum"_test = [] {         // ✔
05   };
06 }
```

---

ut::test

```
01 import ut;                  // ✔
02
03 int main() {
04   "sum"_test = [] {         // ✔
05   };
06 }
```

```
Running... sum
 -> ✔
```

Assertion

```
01 import ut;                  // ✔
02
03 int main() {
04   "sum"_test = [] {         // ✔
05       sum(1, 2) == 42_i; // ut::operator==, ut::""_i
06   };
07 }
```

Operator overloading / C++draft/over.match.oper

12.4 Overload resolution [over.match]

12.4.1 Candidate functions and argument lists [over.match.funcs]

12.4.1.2 Operators in expressions [over.match.oper]

1 If no operand of an operator in an expression has a type that is a class or an enumeration, the operator is assumed to be a built-in operator and interpreted according to [expr.compound]. [*Note:* Because `.`, `.*`, and `::` cannot be overloaded, these operators are always built-in operators interpreted according to [expr.compound]. `?:` cannot be overloaded, but the rules in this subclause are used to determine the conversions to be applied to the second and third operands when they have class or enumeration type ([expr.cond]). — *end note*]

Operator overloading - https://godbolt.org/z/G6Pzb3

Operator overloading - https://godbolt.org/z/G6Pzb3

```
constexpr auto operator==(int, int) -> bool { return {}; }
```

Operator overloading - https://godbolt.org/z/G6Pzb3

```
constexpr auto operator==(int, int) -> bool { return {}; }
```

```
 -> error: overloaded 'operator==' must have at least
      one parameter of class or enumeration type
```

Operator overloading - https://godbolt.org/z/G6Pzb3

```
constexpr auto operator==(int, int) -> bool { return {}; }
```

```
 -> error: overloaded 'operator==' must have at least
      one parameter of class or enumeration type
```

```
template< auto N > constexpr auto operator==(
  std::integral_constant< int, N >, int) -> bool {
  return {};
}
```

-> ✔

ut::operator==

---

ut::operator==

```
template <class TLhs, class TRhs>
  requires concepts::op<TLhs> or concepts::op<TRhs>
constexpr concepts::op auto operator==(TLhs lhs, TRhs rhs) {



}
```

ut::operator==

```
template <class TOp> concept op =
  requires printable<TOp> and (
    requires(TOp op) { // eq, ...
      op.lhs; // left-hand side operator
      op.rhs; // right-hand side operator
      requires std::convertible_to<TOp, bool>; // expect
    } or
    requires(TOp op) { // integral_constant, 42_i
      typename TOp::value_type;
      op.value;
    }
  );
```

```
template <class TLhs, class TRhs>
  requires concepts::op<TLhs> or concepts::op<TRhs>
constexpr concepts::op auto operator==(TLhs lhs, TRhs rhs) {



}
```

ut::operator==

```
template <class TOp> concept op =
  requires printable<TOp> and (
    requires(TOp op) { // eq, ...
      op.lhs; // left-hand side operator
      op.rhs; // right-hand side operator
      requires std::convertible_to<TOp, bool>; // expect
    } or
    requires(TOp op) { // integral_constant, 42_i
      typename TOp::value_type;
      op.value;
    }
  );
```

```
template <class TLhs, class TRhs>
  requires concepts::op<TLhs> or concepts::op<TRhs>
constexpr concepts::op auto operator==(TLhs lhs, TRhs rhs) {
```

```
  return eq{lhs, rhs};
```

```
}
```

ut::eq

ut::eq

```
template <class TLhs, class TRhs>
struct eq final {
```

```
};
```

```
template <class TLhs, class TRhs> eq(TLhs, TRhs) -> eq<TLhs, TRhs>;
```

ut::eq

```
template <class TLhs, class TRhs>
struct eq final {
```

```
  TLhs lhs{}; // left-hand side operator
  TRhs rhs{}; // right-hand side operator
```

```
};
```

```
template <class TLhs, class TRhs> eq(TLhs, TRhs) -> eq<TLhs, TRhs>;
```

ut::eq

```
template <class TLhs, class TRhs>
struct eq final {
```

```
  TLhs lhs{}; // left-hand side operator
  TRhs rhs{}; // right-hand side operator
```

```
  ~eq() noexcept {
    if (not *this) { error(*this); }
  }
```

```
};
```

```
template <class TLhs, class TRhs> eq(TLhs, TRhs) -> eq<TLhs, TRhs>;
```

ut::eq

```
template <class TLhs, class TRhs>
struct eq final {
```

```
  TLhs lhs{}; // left-hand side operator
  TRhs rhs{}; // right-hand side operator
```

```
  ~eq() noexcept {
    if (not *this) { error(*this); }
  }
```

```
  [[nodiscard]] constexpr explicit(false) operator bool() const {
    return value(lhs) == value(rhs);
  }
```

```
  friend auto& operator<<(auto& os, const eq& op) {
    return (os << value(op.lhs) << " == " << value(op.rhs));
  }
```

```
};
```

```
template <class TLhs, class TRhs> eq(TLhs, TRhs) -> eq<TLhs, TRhs>;
```

ut::detail

ut::detail

```
auto error(const auto& expr) -> void {
  std::cerr << "FAILED: " << expr << '\n';
}
```

---

ut::detail

```
auto error(const auto& expr) -> void {
  std::cerr << "FAILED: " << expr << '\n';
}
```

```
constexpr auto value(auto op) { // underlying value
```

```
}
```

ut::detail

```
auto error(const auto& expr) -> void {
  std::cerr << "FAILED: " << expr << '\n';
}
```

```
constexpr auto value(auto op) { // underlying value
```

```
  if constexpr (requires { op.value; }) {
    return op.value;
  } else {
    return op;
  }
```

```
}
```

Design By Introspection - https://d.godbolt.org/z/n8e4de

Design By Introspection - https://d.godbolt.org/z/n8e4de

```
constexpr auto foo(auto v) {




}
```

Design By Introspection - https://d.godbolt.org/z/n8e4de

```
constexpr auto foo(auto v) {
```

```
  if constexpr (requires{ v.foo; }) {
    return v.foo;
  } else {
    return 0;
  }
```

```
}
```

Design By Introspection - https://d.godbolt.org/z/n8e4de

```
constexpr auto foo(auto v) {
```

```
  if constexpr (requires{ v.foo; }) {
    return v.foo;
  } else {
    return 0;
  }
```

```
}
```

```
constexpr struct { int foo{42}; } f;
static_assert(42 == foo(f));
```

Design By Introspection - https://d.godbolt.org/z/n8e4de

```
constexpr auto foo(auto v) {
```

```
  if constexpr (requires{ v.foo; }) {
    return v.foo;
  } else {
    return 0;
  }
```

```
}
```

```
constexpr struct { int foo{42}; } f;
static_assert(42 == foo(f));
```

```
constexpr struct { int bar{42}; } b;
static_assert(0 == foo(b));
```

Design By Introspection - https://d.godbolt.org/z/n8e4de

```
constexpr auto foo(auto v) {
```

```
  if constexpr (requires{ v.foo; }) {
    return v.foo;
  } else {
    return 0;
  }
```

```
}
```

```
constexpr struct { int foo{42}; } f;
static_assert(42 == foo(f));
```

```
constexpr struct { int bar{42}; } b;
static_assert(0 == foo(b));
```

Andrei Alexandrescu - https://www.youtube.com/watch?v=HdzwvY8Mo-w

Assertion

Assertion

```
01 import ut;                  // ✔
02
03 int main() {
04   "sum"_test = [] {         // ✔
05       sum(1, 2) == 42_i; // ut::operator== ✔, ut::""_i
06   };
07 }
```

User Defined Literals (UDL) - C++draft/over.literal

3 The declaration of a literal operator shall have a *parameter-declaration-clause* equivalent to one of the following:

```
const char*
unsigned long long int
long double
char
wchar_t
char8_t
char16_t
char32_t
const char*, std::size_t
const wchar_t*, std::size_t
const char8_t*, std::size_t
const char16_t*, std::size_t
const char32_t*, std::size_t
```

User Defined Literals (UDL) - C++draft/over.literal

3 The declaration of a literal operator shall have a *parameter-declaration-clause* equivalent to one of the following:

```
const char*
unsigned long long int
long double
char
wchar_t
char8_t
char16_t
char32_t
const char*, std::size_t
const wchar_t*, std::size_t
const char8_t*, std::size_t
const char16_t*, std::size_t
const char32_t*, std::size_t
```

```
constexpr auto operator""_i(int) -> int;
```

User Defined Literals (UDL) - C++draft/over.literal

3 The declaration of a literal operator shall have a *parameter-declaration-clause* equivalent to one of the following:

```
const char*
unsigned long long int
long double
char
wchar_t
char8_t
char16_t
char32_t
const char*, std::size_t
const wchar_t*, std::size_t
const char8_t*, std::size_t
const char16_t*, std::size_t
const char32_t*, std::size_t
```

```
constexpr auto operator""_i(int) -> int;
```

```
-> error: invalid literal operator parameter type 'int',
     did you mean 'unsigned long long'?
```

ut::""_i - User Defined Literals (`UDL`) - https://godbolt.org/z/cZsSK3

---

ut::""_i - User Defined Literals (UDL) - https://godbolt.org/z/cZsSK3

```
template <char... Cs>
[[nodiscard]] constexpr auto operator""_i() { // int
```

```
}
```

ut::""_i - User Defined Literals (UDL) - https://godbolt.org/z/cZsSK3

```
template <char... Cs>
[[nodiscard]] constexpr auto operator""_i() { // int
```

```
  return []<auto... Ns>(std::index_sequence<Ns...>) {
```

```
  }(std::make_index_sequence<sizeof...(Cs)>{});
```

```
}
```

---

ut::""_i - User Defined Literals (UDL) - https://godbolt.org/z/cZsSK3

```
template <char... Cs>
[[nodiscard]] constexpr auto operator""_i() { // int
```

```
  return []<auto... Ns>(std::index_sequence<Ns...>) {
```

```
    return std::integral_constant<int,
      ((std::pow(10, sizeof...(Ns) - Ns - 1)) * (Cs - '0')) + ...)
    >{};
```

```
  }(std::make_index_sequence<sizeof...(Cs)>{});
```

```
}
```

---

ut::""_i - User Defined Literals (UDL) - https://godbolt.org/z/cZsSK3

```
template <char... Cs>
[[nodiscard]] constexpr auto operator""_i() { // int
```

```
  return []<auto... Ns>(std::index_sequence<Ns...>) {
```

```
    return std::integral_constant<int,
      ((std::pow(10, sizeof...(Ns) - Ns - 1)) * (Cs - '0')) + ...)
    >{};
```

```
  }(std::make_index_sequence<sizeof...(Cs)>{});
```

```
}
```

```
static_assert(0_i  == 0);
static_assert(42_i == 42);
```

Immediately-Invoked Function Expression (IIFE) - https://godbolt.org/z/aM58T1

Immediately-Invoked Function Expression (IIFE) - https://godbolt.org/z/aM58T1

```
template<auto N>
constexpr auto unfold = [](auto expr) {
  [expr]<auto ...Is>(std::index_sequence<Is...>) {
    ((expr(), void(Is)), ...);
  }(std::make_index_sequence<N>{});
};
```

Immediately-Invoked Function Expression (IIFE) - https://godbolt.org/z/aM58T1

```
template<auto N>
constexpr auto unfold = [](auto expr) {
  [expr]<auto ...Is>(std::index_sequence<Is...>) {
    ((expr(), void(Is)), ...);
  }(std::make_index_sequence<N>{});
};
```

```
int main() {
  unfold<2>([]{ std::puts("ACCU-2021!"); });
}
```

Immediately-Invoked Function Expression (IIFE) - https://godbolt.org/z/aM58T1

```
template<auto N>
constexpr auto unfold = [](auto expr) {
  [expr]<auto ...Is>(std::index_sequence<Is...>) {
    ((expr(), void(Is)), ...);
  }(std::make_index_sequence<N>{});
};
```

```
int main() {
  unfold<2>([]{ std::puts("ACCU-2021!"); });
}
```

```
.LC0: .string "ACCU-2021!"
main:
  sub     rsp, 8
  mov     edi, OFFSET FLAT:.LC0
  call    puts
  mov     edi, OFFSET FLAT:.LC0
  call    puts
  xor     eax, eax
  add     rsp, 8
  ret
```

Assertion

Assertion

```
01 import ut;                  // ✔
02
03 int main() {
04   "sum"_test = [] {         // ✔
05     sum(1, 2) == 42_i; // ut::operator== ✔, ut::""_i ✔
06   };
07 }
```

Assertion

```
01 import ut;                  // ✔
02
03 int main() {
04   "sum"_test = [] {         // ✔
05       sum(1, 2) == 42_i; // ut::operator== ✔, ut::""_i ✔
06   };
07 }
```

```
Running... sum
FAILED: [3 == 42]
 -> It's missing FILE:LINE!
```

Source Location / C++20 / C++draft/support.srcloc#source.location.syn

17.8.2 Class source_location

```
namespace std {
  struct source_location {
    // source location construction
    static consteval source_location current() noexcept;
    constexpr source_location() noexcept;

    // source location field access
    constexpr uint_least32_t line() const noexcept;
    constexpr uint_least32_t column() const noexcept;
    constexpr const char* file_name() const noexcept;
    constexpr const char* function_name() const noexcept;

  private:
    uint_least32_t line_;                // exposition only
    uint_least32_t column_;              // exposition only
    const char* file_name_;              // exposition only
    const char* function_name_;          // exposition only
  };
}
```

Source Location / C++20 / C++draft/support.srcloc#source.location.syn

17.8.2 Class source_location

```
namespace std {
  struct source_location {
    // source location construction
    static consteval source_location current() noexcept;
    constexpr source_location() noexcept;

    // source location field access
    constexpr uint_least32_t line() const noexcept;
    constexpr uint_least32_t column() const noexcept;
    constexpr const char* file_name() const noexcept;
    constexpr const char* function_name() const noexcept;

  private:
    uint_least32_t line_;          // exposition only
    uint_least32_t column_;        // exposition only
    const char* file_name_;        // exposition only
    const char* function_name_;    // exposition only
  };
}
```

Macro-free replacement for `__FILE__`, `__LINE__`, `__function__`

Source Location / C++20 - https://godbolt.org/z/G6Pzb3

Source Location / C++20 - https://godbolt.org/z/G6Pzb3

```
constexpr auto operator==(
  auto, auto, const std::source_location& location =
                  std::source_location::current()
) -> bool {
  return {};
}
```

```
 -> error: parameter of overloaded 'operator=='
      cannot have a default argument
```

Source Location / C++20

Source Location / C++20

```
auto error(const auto& expr, const auto& location) -> void {
  std::cerr << location.file_name() << ':' << location.line()
            << ":FAILED: " << expr << '\n';
}
```

Source Location / C++20

```
auto error(const auto& expr, const auto& location) -> void {
  std::cerr << location.file_name() << ':' << location.line()
            << ":FAILED: " << expr << '\n';
}
```

```
template<class TLhs, class TRhs>
struct eq final {
  constexpr eq(TLhs lhs, TRhs rhs,
    std::source_location& location = // FILE:LINE -> ut.hpp:128
    std::source_location::current());
```

```
  TLhs lhs_{};
  TRhs rhs_{};
  std::source_location location_{};
};
```

Source Location / C++20

```
auto error(const auto& expr, const auto& location) -> void {
  std::cerr << location.file_name() << ':' << location.line()
            << ":FAILED: " << expr << '\n';
}
```

```
template<class TLhs, class TRhs>
struct eq final {
  constexpr eq(TLhs lhs, TRhs rhs,
    std::source_location& location = // FILE:LINE -> ut.hpp:128
    std::source_location::current());
```

```
  ~eq() noexcept {
    if (not *this) {
      error(*this, location_);
    }
  }
```

```
  TLhs lhs_{};
  TRhs rhs_{};
  std::source_location location_{};
};
```

Assertion

Assertion

```
01 import ut;                  // ✔
02
03 int main() {
04   "sum"_test = [] {         // ✔
05       sum(1, 2) == 42_i; // ut::operator== ✔, ut::""_i ✔
06   };
07 }
```

Assertion

```
01 import ut;                  // ✔
02
03 int main() {
04   "sum"_test = [] {         // ✔
05       sum(1, 2) == 42_i; // ut::operator== ✔, ut::""_i ✔
06   };
07 }
```

```
Running... sum
ut.hpp:128:FAILED: [3 == 42]
 -> Wrong FILE:LINE!
```

ut::value_location

ut::value_location

```
template <class TValue> struct value_location {
```

```
  TValue value{};
  std::source_location location{};
};
```

ut::value_location

```
template <class TValue> struct value_location {
```

```
  constexpr explicit(false)
    value_location(TValue value,
                   const std::source_location &location =
                         std::source_location::current())
    : value{value}, location{location} {}
```

```
  TValue value{};
  std::source_location location{};
};
```

ut::value_location

ut::value_location

```
template <op T>
constexpr auto operator==(
  value_location<typename T::value_type> lhs, T rhs) {
```

```
}
```

ut::value_location

```
template <op T>
constexpr auto operator==(
  value_location<typename T::value_type> lhs, T rhs) {
```

```
  struct eq : detail::eq<decltype(lhs), decltype(rhs)> {
    ~eq() noexcept {
      if (not *this) {
        error(*this, lhs.location);
      }
    }
  };

  return eq{lhs, rhs};
```

```
}
```

Assertion

Assertion

```
01 import ut;                  // ✔
02
03 int main() {
04   "sum"_test = [] {         // ✔
05       sum(1, 2) == 42_i; // ut::operator== ✔, ut::""_i ✔
06   };
07 }
```

Assertion

```
01 import ut;                  // ✔
02
03 int main() {
04   "sum"_test = [] {         // ✔
05     sum(1, 2) == 42_i; // ut::operator== ✔, ut::""_i ✔
06   };
07 }
```

```
Running... sum
test.cpp:05:FAILED: [3 == 42]
```
-> ✔

Achievement Unlocked (First goal)

ut::suite / ut::expect

ut::suite / ut::expect

```
01 import ut;// ✔
02
03 suite sums = [] { // ut::suite
04  "sum with no args"_test        = [] { expect(sum() == 0_i); };
05  "sum with single arg"_test     = [] { expect(sum(42) == 42_i); };
06  "sum with multiple args"_test = [] { expect(sum(1, 2) == 3_i); };
07 };
08
09 int main() {
10 }
```

ut::expect

```
01 import ut; // ✔
02
03 int main() {
04   expect(sum(1, 2) == 21_i) << "?"; // ut::expect
05 }
```

ut::expect

ut::expect

```
constexpr concepts::streamable auto& expect(
  concepts::expression auto expr,
  const std::source_location &location =
        std::source_location::current()) {
```

```
}
```

ut::expect

```
template <class TExpr> concept expression =
  std::convertible_to<TExpr, bool> and printable<TExpr>;
```

```
constexpr concepts::streamable auto& expect(
  concepts::expression auto expr,
  const std::source_location &location =
        std::source_location::current()) {
```

```
}
```

---

ut::expect

```
template <class TExpr> concept expression =
  std::convertible_to<TExpr, bool> and printable<TExpr>;
```

```
constexpr concepts::streamable auto& expect(
  concepts::expression auto expr,
  const std::source_location &location =
        std::source_location::current()) {
```

```
  if (not expr) {
    error(expr, location);
  }
  return std::cerr;
```

```
}
```

---

ut::expect

```
template <class TExpr> concept expression =
  std::convertible_to<TExpr, bool> and printable<TExpr>;
```

```
constexpr concepts::streamable auto& expect(
  concepts::expression auto expr,
  const std::source_location &location =
        std::source_location::current()) {
```

```
  if (not expr) {
    error(expr, location);
  }
  return std::cerr;
```

```
}
```

```
template <class TLhs, class TRhs>
  requires concepts::op<TLhs> or concepts::op<TRhs>
constexpr concepts::op auto operator==(TLhs lhs, TRhs rhs);
```

ut::expect - https://godbolt.org/z/6Nk5Mi

---

ut::expect - https://godbolt.org/z/6Nk5Mi

```
01 import ut;                  // ✔
02
03 int main() {
04   expect(sum(1, 2) == 21_i) << '?';
05 }
```

---

ut::expect - https://godbolt.org/z/6Nk5Mi

```
01 import ut;                  // ✔
02
03 int main() {
04   expect(sum(1, 2) == 21_i) << '?';
05 }
```

```
expect.cpp:04:FAILED [3 == 21] ?
 -> ✔
```

ut::suite

ut::suite

```
struct suite {




};
```

ut::suite

```
struct suite {
```

```
  [[nodiscard]] constexpr explicit(false)
  suite(concepts::suite auto suite) {
    suite();
  }
```

```
};
```

ut::suite

```
template <class TSuite>
concept suite = std::invocable<TSuite>;
```

```
struct suite {
```

```
  [[nodiscard]] constexpr explicit(false)
  suite(concepts::suite auto suite) {
    suite();
  }
```

```
};
```

ut::suite / ut::expect

ut::suite / ut::expect

```
01 import ut;          // ✔
02
03 suite sums = [] { // ✔
04  "sum with no args"_test        = [] { expect(sum() == 0_i); };
05  "sum with single arg"_test     = [] { expect(sum(42) == 42_i); };
06  "sum with multiple args"_test = [] { expect(sum(1, 2) == 3_i); };
07 };
08
09 int main() {
10 }
```

ut::suite / ut::expect

```
01 import ut;          // ✔
02
03 suite sums = [] { // ✔
04  "sum with no args"_test        = [] { expect(sum() == 0_i); };
05  "sum with single arg"_test     = [] { expect(sum(42) == 42_i); };
06  "sum with multiple args"_test = [] { expect(sum(1, 2) == 3_i); };
07 };
08
09 int main() {
10 }
```

```
Running... sum with no args
Running... sum with single arg
Running... sum with multiple args
 -> ✔
```

Achievement Unlocked (Second goal)

Full implementation at:

Full implementation at:

- https://github.com/boost-ext/ut/tree/gh-pages/accu-2021/example

Full implementation at:

- https://github.com/boost-ext/ut/tree/gh-pages/accu-2021/example
- Header - https://godbolt.org/z/a7Mses

Full implementation at:

- https://github.com/boost-ext/ut/tree/gh-pages/accu-2021/example
- Header - https://godbolt.org/z/a7Mses
- Module - https://wandbox.org/permlink/LRz6YCZJX9aff34x

UT - Unit Testing Framework

https://github.com/boost-ext/ut

UT - Overview

UT - Overview

- C++20 (GCC-9+, Clang-9.0+, Apple Clang-11.0.0+, MSVC-2019+*)

- (*) Limitations may apply

UT - Overview

- C++20 (GCC-9+, Clang-9.0+, Apple Clang-11.0.0+, MSVC-2019+*)
- Single header/module (2k LOC) with no external dependencies

---

- (*) Limitations may apply

UT - Overview

- C++20 (GCC-9+, Clang-9.0+, Apple Clang-11.0.0+, MSVC-2019+*)
- Single header/module (2k LOC) with no external dependencies
- Macro-free (✔)

- (*) Limitations may apply

UT - Overview

- C++20 (GCC-9+, Clang-9.0+, Apple Clang-11.0.0+, MSVC-2019+*)
- Single header/module (2k LOC) with no external dependencies
- Macro-free (✔)
- Features (Assertions, Suites, Tests, Sections, BDD, Matchers, Logging, ...)

- (*) Limitations may apply

UT - Overview

- C++20 (GCC-9+, Clang-9.0+, Apple Clang-11.0.0+, MSVC-2019+*)
- Single header/module (2k LOC) with no external dependencies
- Macro-free (✔)
- Features (Assertions, Suites, Tests, Sections, BDD, Matchers, Logging, ...)

- (*) Limitations may apply
- UT is not an official Boost library

Hello World - https://godbolt.org/z/Y43mXz

```cpp
import boost.ut; // #include <boost/ut.hpp>

auto sum(auto... args) { return (args + ...); }

int main() {
  using namespace boost::ut;

  "sum"_test = [] {
    sum(0) == 0_i;
    sum(1, 2) == 3_i;
    sum(1, 2) > 0_i and 41_i == sum(40, 2);
  };
}
```

```
Running "sum"...
  sum.cpp:11:FAILED [(3 > 0 and 41 == 42)]
FAILED
================================================================================
tests:   1 | 1 failed
asserts: 3 | 2 passed | 1 failed
```

UT main - https://godbolt.org/z/f1ba1j

```
auto ut_main(int argc, const char** argv) -> int {
  cfg<override> = {.filter = argv[1]};
  return cfg<override>.run();
}
```

main - https://godbolt.org/z/f1ba1j

main - https://godbolt.org/z/f1ba1j

```
auto main(int argc, const char** argv) -> int {
```

```
  return ut_main(argc, argv);
}
```

main - https://godbolt.org/z/f1ba1j

```
auto main(int argc, const char** argv) -> int {
```

```
  "empty"_test        = []{};
  "single"_test       = []{};
  "many"_test         = []{
    "equal"_test      = []{};
    "not equal"_test = []{};
  };
```

```
  return ut_main(argc, argv);
}
```

main - https://godbolt.org/z/f1ba1j

```
auto main(int argc, const char** argv) -> int {
```

```
  "empty"_test        = []{};
  "single"_test       = []{};
  "many"_test         = []{
    "equal"_test      = []{};
    "not equal"_test = []{};
  };
```

```
  return ut_main(argc, argv);
}
```

```
./main "*"  # Regex-like syntax is supported
```

```
 -> All tests passed (0 asserts in 3 tests)
```

main - https://godbolt.org/z/f1ba1j

```
auto main(int argc, const char** argv) -> int {
```

```
  "empty"_test        = []{};
  "single"_test       = []{};
  "many"_test         = []{
    "equal"_test      = []{};
    "not equal"_test = []{};
  };
```

```
  return ut_main(argc, argv);
}
```

```
./main "*"  # Regex-like syntax is supported
```

```
 -> All tests passed (0 asserts in 3 tests)
```

```
./main "single*"
```

```
 -> All tests passed (0 asserts in 1 tests)
```

main - https://godbolt.org/z/f1ba1j

```
auto main(int argc, const char** argv) -> int {
```

```
  "empty"_test        = []{};
  "single"_test       = []{};
  "many"_test         = []{
    "equal"_test      = []{};
    "not equal"_test = []{};
  };
```

```
  return ut_main(argc, argv);
}
```

```
./main "*"  # Regex-like syntax is supported
```
```
 -> All tests passed (0 asserts in 3 tests)
```

```
./main "single*"
```
```
 -> All tests passed (0 asserts in 1 tests)
```

```
./main "many.equal*" # Dot(.) is used to distinguish sub-tests
```
```
 -> All tests passed (0 asserts in 1 tests)
```

Tags - https://godbolt.org/z/X3_kG4

Tags - https://godbolt.org/z/X3_kG4

```
"performance"_test= [] {
  expect(42_i == 42);
};
```

Tags - https://godbolt.org/z/X3_kG4

```
tag("nightly") / tag("slow") /
```

```
"performance"_test= [] {
  expect(42_i == 42);
};
```

Tags - https://godbolt.org/z/X3_kG4

```
tag("nightly") / tag("slow") /
```

```
"performance"_test= [] {
  expect(42_i == 42);
};
```

```
"don't run"_test = [] {
  expect(42_i == 43) << "should not fire!";
};
```

Tags - https://godbolt.org/z/X3_kG4

```
tag("nightly") / tag("slow") /
```

```
"performance"_test= [] {
  expect(42_i == 42);
};
```

```
skip /
```

```
"don't run"_test = [] {
  expect(42_i == 43) << "should not fire!";
};
```

Tags - https://godbolt.org/z/X3_kG4

```
tag("nightly") / tag("slow") /
```

```
"performance"_test= [] {
  expect(42_i == 42);
};
```

```
skip /
```

```
"don't run"_test = [] {
  expect(42_i == 43) << "should not fire!";
};
```

```
cfg<override> = {.tag = {"nightly"}};
```

Tags - https://godbolt.org/z/X3_kG4

```
tag("nightly") / tag("slow") /
```

```
"performance"_test= [] {
  expect(42_i == 42);
};
```

```
skip /
```

```
"don't run"_test = [] {
  expect(42_i == 43) << "should not fire!";
};
```

```
cfg<override> = {.tag = {"nightly"}};
```

```
-> All tests passed (1 asserts in 1 tests)
   1 tests skipped
```

Assertions - https://godbolt.org/z/jaFK8w

Assertions - https://godbolt.org/z/jaFK8w

```
1_i == 2;                                   // Terse syntax
  -> assertions.cpp:1:FAILED [1 == 2]
```

Assertions - https://godbolt.org/z/jaFK8w

```
1_i == 2;                                   // Terse syntax
  -> assertions.cpp:1:FAILED [1 == 2]

expect(2 == 1_i);                           // Expect syntax
  -> assertions.cpp:1:FAILED [2 == 1]
```

Assertions - https://godbolt.org/z/jaFK8w

```
1_i == 2;                                   // Terse syntax
  -> assertions.cpp:1:FAILED [1 == 2]

expect(2 == 1_i);                           // Expect syntax
  -> assertions.cpp:1:FAILED [2 == 1]

expect(that % 1 == 2);                      // Matchers syntax
  -> assertions.cpp:1:FAILED [1 == 2]
```

Assertions - https://godbolt.org/z/jaFK8w

```
1_i == 2;                                   // Terse syntax
  -> assertions.cpp:1:FAILED [1 == 2]

expect(2 == 1_i);                           // Expect syntax
  -> assertions.cpp:1:FAILED [2 == 1]

expect(that % 1 == 2);                      // Matchers syntax
  -> assertions.cpp:1:FAILED [1 == 2]

std::vector v{1l, 2l, 3l};
(4_ul == std::size(v)) >> fatal;            // Fatal assertion
v[3] == 4_l;                                // Not executed
  -> assertions.cpp:2:FAILED [4 == 3]
```

Assertions - https://godbolt.org/z/jaFK8w

```
1_i == 2;                                   // Terse syntax
```

```
  -> assertions.cpp:1:FAILED [1 == 2]
```

```
expect(2 == 1_i);                           // Expect syntax
```

```
  -> assertions.cpp:1:FAILED [2 == 1]
```

```
expect(that % 1 == 2);                      // Matchers syntax
```

```
  -> assertions.cpp:1:FAILED [1 == 2]
```

```
std::vector v{1l, 2l, 3l};
(4_ul == std::size(v)) >> fatal;            // Fatal assertion
v[3] == 4_l;                                // Not executed
```

```
  -> assertions.cpp:2:FAILED [4 == 3]
```

```
41.10_d == 42.101 and "a" == "b"sv;     // Compound expression
                                        // with floating-point
```

```
  -> assertions.cpp:1:FAILED [42.1 == 42.101 and a == b]
```

Misc/Logging - https://godbolt.org/z/rnanxr

Misc/Logging - https://godbolt.org/z/rnanxr

```
(1_i == 2) << "should equal?";
  -> assertions.cpp:1:FAILED [1 == 2] should equal?
```

Misc/Logging - https://godbolt.org/z/rnanxr

```
(1_i == 2) << "should equal?";
  -> assertions.cpp:1:FAILED [1 == 2] should equal?
```

```
expect(1_i == 2) << "should equal?";
  -> assertions.cpp:1:FAILED [1 == 2] should equal?
```

Misc/Logging - https://godbolt.org/z/rnanxr

```
(1_i == 2) << "should equal?";
  -> assertions.cpp:1:FAILED [1 == 2] should equal?

expect(1_i == 2) << "should equal?";
  -> assertions.cpp:1:FAILED [1 == 2] should equal?

log << "I'm here!";
  -> assertions.cpp:1:I'm here!
```

Sections - https://godbolt.org/z/y9m5vF

Sections - https://godbolt.org/z/y9m5vF

```
"[vector]"_test = [] {

};
```

Sections - https://godbolt.org/z/y9m5vF

```
"[vector]"_test = [] {
```

```
  // set up (1)
  std::vector<int> v(5);
  (5_ul == std::size(v)) >> fatal;
```

```
};
```

Sections - https://godbolt.org/z/y9m5vF

```
"[vector]"_test = [] {
```

```
  // set up (1)
  std::vector<int> v(5);
  (5_ul == std::size(v)) >> fatal;
```

```
  should("resize bigger") = [v] { // section (2.1)
    mut(v).resize(10);
    10_ul == std::size(v);
  };
```

```
};
```

Sections - https://godbolt.org/z/y9m5vF

```
"[vector]"_test = [] {

  // set up (1)
  std::vector<int> v(5);
  (5_ul == std::size(v)) >> fatal;

  should("resize bigger") = [v] { // section (2.1)
    mut(v).resize(10);
    10_ul == std::size(v);
  };

  (5_ul == std::size(v)) >> fatal; // (3)

};
```

Sections - https://godbolt.org/z/y9m5vF

```
"[vector]"_test = [] {

  // set up (1)
  std::vector<int> v(5);
  (5_ul == std::size(v)) >> fatal;

  should("resize bigger") = [v] { // section (2.1)
    mut(v).resize(10);
    10_ul == std::size(v);
  };

  (5_ul == std::size(v)) >> fatal; // (3)

  should("resize smaller") = [v] { // section (2.2)
    mut(v).resize(0);
    0_ul == std::size(v);
  };



};
```

Sections - https://godbolt.org/z/y9m5vF

```
"[vector]"_test = [] {

  // set up (1)
  std::vector<int> v(5);
  (5_ul == std::size(v)) >> fatal;

  should("resize bigger") = [v] { // section (2.1)
    mut(v).resize(10);
    10_ul == std::size(v);
  };

  (5_ul == std::size(v)) >> fatal; // (3)

  should("resize smaller") = [v] { // section (2.2)
    mut(v).resize(0);
    0_ul == std::size(v);
  };

  // tear down (4)

};
```

Suites - https://godbolt.org/z/7P3Ph1

Suites - https://godbolt.org/z/7P3Ph1

```
suite errors = [] {

};
```

Suites - https://godbolt.org/z/7P3Ph1

```
suite errors = [] {
```

```
  "exception"_test = [] {
    expect(throws([] { throw 0; })) << "throws any exception";
    expect(throws<std::runtime_error>([] {
      throw std::runtime_error{"error"}; })
    );
  };

  "failure"_test = [] {
    expect(aborts([] { assert(false); }));
   };
```

```
};
```

Suites - https://godbolt.org/z/7P3Ph1

```
suite errors = [] {
```

```
  "exception"_test = [] {
    expect(throws([] { throw 0; })) << "throws any exception";
    expect(throws<std::runtime_error>([] {
      throw std::runtime_error{"error"}; })
    );
  };

  "failure"_test = [] {
    expect(aborts([] { assert(false); }));
   };
```

```
};
```

```
int main() { }
```

```
 -> All tests passed (3 asserts in 1 tests)
```

Parameterized - https://godbolt.org/z/6FHtpq

Parameterized - https://godbolt.org/z/6FHtpq

```
for (auto i : std::vector{1, 2, 3}) {
  test("args " + std::to_string(i)) = [i] {
    expect(arg > 0_i) << "all values greater than 0";
  };
}
```

Parameterized - https://godbolt.org/z/6FHtpq

```
for (auto i : std::vector{1, 2, 3}) {
  test("args " + std::to_string(i)) = [i] {
    expect(arg > 0_i) << "all values greater than 0";
  };
}
```

```
-> All tests passed (3 asserts in 3 tests)
```

Parameterized - https://godbolt.org/z/6FHtpq

```
for (auto i : std::vector{1, 2, 3}) {
  test("args " + std::to_string(i)) = [i] {
    expect(arg > 0_i) << "all values greater than 0";
  };
}
```

```
-> All tests passed (3 asserts in 3 tests)
```

```
"args and types"_test =
  []<class TArg>(TArg arg) {
    expect(std::is_integral_v<TArg>);
    expect(type<TArg> == type<int> or type<TArg> == type<bool>);
  }
```

Parameterized - https://godbolt.org/z/6FHtpq

```
for (auto i : std::vector{1, 2, 3}) {
  test("args " + std::to_string(i)) = [i] {
    expect(arg > 0_i) << "all values greater than 0";
  };
}
```

```
-> All tests passed (3 asserts in 3 tests)
```

```
"args and types"_test =
  []<class TArg>(TArg arg) {
    expect(std::is_integral_v<TArg>);
    expect(type<TArg> == type<int> or type<TArg> == type<bool>);
  }
```

```
| std::tuple{true, 42};
 -> All tests passed (4 asserts in 2 tests)
```

Template metaprogramming - https://godbolt.org/z/9oh97x

Template metaprogramming - https://godbolt.org/z/9oh97x

```
01 constexpr auto i = 0;
02 constant<42_i == i> and type<void> == type<int>;
```

Template metaprogramming - https://godbolt.org/z/9oh97x

```
01 constexpr auto i = 0;
02 constant<42_i == i> and type<void> == type<int>;
```

```
-> meta-programming.cpp:2:FAILED [(42 == 0 and void == int)]
```

Spec - https://godbolt.org/z/6jKKzT

Spec - https://godbolt.org/z/6jKKzT

```
describe("vector") = [] {




};
```

Spec - https://godbolt.org/z/6jKKzT

```
describe("vector") = [] {
```

```
  std::vector<int> v(5);
  expect((5_ul == std::size(v)) >> fatal);
```

```
};
```

Spec - https://godbolt.org/z/6jKKzT

```
describe("vector") = [] {
```

```
  std::vector<int> v(5);
  expect((5_ul == std::size(v)) >> fatal);
```

```
  it("should resize bigger") = [v] {
    mut(v).resize(10);
    expect(10_ul == std::size(v));
  };
```

```
};
```

Spec - https://godbolt.org/z/6jKKzT

```
describe("vector") = [] {

  std::vector<int> v(5);
  expect((5_ul == std::size(v)) >> fatal);

  it("should resize bigger") = [v] {
    mut(v).resize(10);
    expect(10_ul == std::size(v));
  };

};
```

```
-> All tests passed (2 asserts in 1 tests)
```

Behavior Driven Development (BDD) - https://godbolt.org/z/4Mdo3K

Behavior Driven Development (BDD) - https://godbolt.org/z/4Mdo3K

```
feature("vector") = [] {
  scenario("size") = [] {
```

```
  };
};
```

Behavior Driven Development (BDD) - https://godbolt.org/z/4Mdo3K

```
feature("vector") = [] {
  scenario("size") = [] {
```

```
    given("I have a vector") = [] {
      std::vector<int> v(5);
      expect((5_ul == std::size(v)) >> fatal);
```

```
  };
};
```

Behavior Driven Development (BDD) - https://godbolt.org/z/4Mdo3K

```
feature("vector") = [] {
  scenario("size") = [] {
```

```
    given("I have a vector") = [] {
      std::vector<int> v(5);
      expect((5_ul == std::size(v)) >> fatal);
```

```
      when("I resize bigger") = [v] {
        mut(v).resize(10);
```

```
      };
```

```
  };
};
```

Behavior Driven Development (BDD) - https://godbolt.org/z/4Mdo3K

```
feature("vector") = [] {
  scenario("size") = [] {

    given("I have a vector") = [] {
      std::vector<int> v(5);
      expect((5_ul == std::size(v)) >> fatal);

      when("I resize bigger") = [v] {
        mut(v).resize(10);

        then("The size should increase") = [v] {
          expect(10_ul == std::size(v));
        };

      };

  };
};
```

Behavior Driven Development (BDD) - https://godbolt.org/z/4Mdo3K

```
feature("vector") = [] {
  scenario("size") = [] {

    given("I have a vector") = [] {
      std::vector<int> v(5);
      expect((5_ul == std::size(v)) >> fatal);

      when("I resize bigger") = [v] {
        mut(v).resize(10);

        then("The size should increase") = [v] {
          expect(10_ul == std::size(v));
        };

      };

  };
};
```

```
-> All tests passed (2 asserts in 1 tests)
```

Gherkin - https://godbolt.org/z/jb1d8P

Gherkin - https://godbolt.org/z/jb1d8P

VECTOR.FEATURE

Gherkin - https://godbolt.org/z/jb1d8P

VECTOR.FEATURE

```
Feature: Vector
  Scenario: Resize
    Given I have a vector
     When I resize bigger
     Then The size should increase
```

Gherkin - https://godbolt.org/z/jb1d8P

Gherkin - https://godbolt.org/z/jb1d8P

VECTOR.CPP

Gherkin - https://godbolt.org/z/jb1d8P

VECTOR.CPP

```
gherkin::steps steps = [](auto& steps) {

};
```

Gherkin - https://godbolt.org/z/jb1d8P

VECTOR.CPP

```
gherkin::steps steps = [](auto& steps) {
```

```
  steps.feature("Vector") = [&] {
    steps.scenario("*") = [&] {
```

```
    };
  };
```

```
};
```

Gherkin - https://godbolt.org/z/jb1d8P

VECTOR.CPP

```
gherkin::steps steps = [](auto& steps) {
```

```
  steps.feature("Vector") = [&] {
    steps.scenario("*") = [&] {
```

```
      steps.given("I have a vector") = [&] {
        std::vector<int> v(5);
        steps.when("I resize bigger") = [&] { v.resize(10); };
        steps.then("The size should increase") = [&] {
          expect(10_ul == std::size(v));
        };
      };
```

```
    };
  };
```

```
};
```

Gherkin - https://godbolt.org/z/jb1d8P

Gherkin - https://godbolt.org/z/jb1d8P

```
int main() {



}
```

Gherkin - https://godbolt.org/z/jb1d8P

```
int main() {
```

```
  "gherkin"_test = steps | "vector.feature"_file;
```

```
}
```

Gherkin - https://godbolt.org/z/jb1d8P

```
int main() {

  "gherkin"_test = steps | "vector.feature"_file;

}
```

```
-> All tests passed (2 asserts in 1 tests)
```

Parallel execution - https://godbolt.org/z/M7z1qv

Parallel execution - https://godbolt.org/z/M7z1qv

```
suite parallel_1 = [] {
  "test.1.1"_test = [] { expect(1_i == 1); };
  "test.1.2"_test = [] { expect(2_i == 2); };
};
```

Parallel execution - https://godbolt.org/z/M7z1qv

```
suite parallel_1 = [] {
  "test.1.1"_test = [] { expect(1_i == 1); };
  "test.1.2"_test = [] { expect(2_i == 2); };
};
```

```
suite parallel_2 = [] {
  "test.2.1"_test = [] { expect(1_i == 1); };
  "test.2.2"_test = [] { expect(2_i == 2); };
};
```

Parallel execution - https://godbolt.org/z/M7z1qv

```
suite parallel_1 = [] {
  "test.1.1"_test = [] { expect(1_i == 1); };
  "test.1.2"_test = [] { expect(2_i == 2); };
};
```

```
suite parallel_2 = [] {
  "test.2.1"_test = [] { expect(1_i == 1); };
  "test.2.2"_test = [] { expect(2_i == 2); };
};
```

```
template <> auto cfg<ut::override> = parallel_runner{};
```

Parallel execution - https://godbolt.org/z/M7z1qv

```
suite parallel_1 = [] {
  "test.1.1"_test = [] { expect(1_i == 1); };
  "test.1.2"_test = [] { expect(2_i == 2); };
};
```

```
suite parallel_2 = [] {
  "test.2.1"_test = [] { expect(1_i == 1); };
  "test.2.2"_test = [] { expect(2_i == 2); };
};
```

```
template <> auto cfg<ut::override> = parallel_runner{};
```

```
./parallel    ./parallel
test.2.1      test.1.1
test.2.2      test.1.2
test.1.1      test.2.1
test.1.2      test.2.2
```

Parallel execution - https://godbolt.org/z/M7z1qv

```
suite parallel_1 = [] {
  "test.1.1"_test = [] { expect(1_i == 1); };
  "test.1.2"_test = [] { expect(2_i == 2); };
};
```

```
suite parallel_2 = [] {
  "test.2.1"_test = [] { expect(1_i == 1); };
  "test.2.2"_test = [] { expect(2_i == 2); };
};
```

```
template <> auto cfg<ut::override> = parallel_runner{};
```

```
./parallel    ./parallel
test.2.1      test.1.1
test.2.2      test.1.2
test.1.1      test.2.1
test.1.2      test.2.2
```

```
 -> All tests passed (4 asserts in 4 tests)
```

Benchmarking - https://godbolt.org/z/WqznMn

Benchmarking - https://godbolt.org/z/WqznMn

```
"string creation"_benchmark = [] {
  std::string created_string{"hello"};
  do_not_optimize(created_string);
}
```

Benchmarking - https://godbolt.org/z/WqznMn

```
auto do_not_optimize(auto expr) -> void {
  asm volatile("" :: "m"(expr) : "memory");
}
```

```
"string creation"_benchmark = [] {
  std::string created_string{"hello"};
  do_not_optimize(created_string);
}
```

Benchmarking - https://godbolt.org/z/WqznMn

```
auto do_not_optimize(auto expr) -> void {
  asm volatile("" :: "m"(expr) : "memory");
}
```

```
"string creation"_benchmark = [] {
  std::string created_string{"hello"};
  do_not_optimize(created_string);
}
```

```
[string creation] 3749 ns
```

```
 -> All tests passed (0 asserts in 1 tests)
```

And much more at:

And much more at:

https://github.com/boost-ext/ut

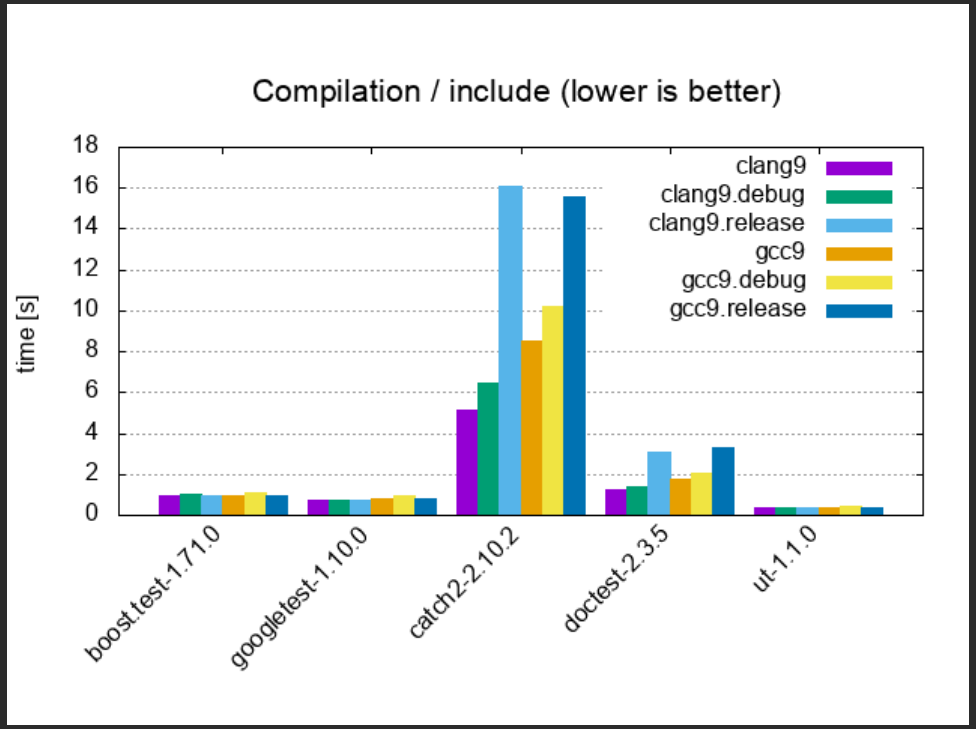
Benchmarks

https://github.com/cpp-testing/ut-benchmark

Benchmarks - Frameworks

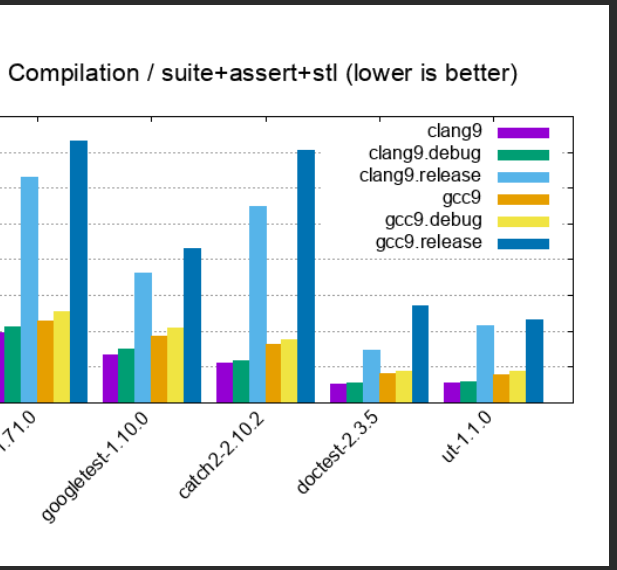
Framework	Version	Standard	License	Linkage	Test configuration
Boost.Test	1.71.0	C++03	Boost 1.0	single header/library	`static library`
GoogleTest	1.10.0	C++11	BSD-3	library	`static library`
Catch	2.10.2	C++11	Boost 1.0	single header	`CATCH_CONFIG_FAST_COMPILE`
Doctest	2.3.5	C++11	MIT	single header	`DOCTEST_CONFIG_SUPER_FAST_ASSERTS`
UT	1.1.0	C++17	Boost 1.0	single header/module	

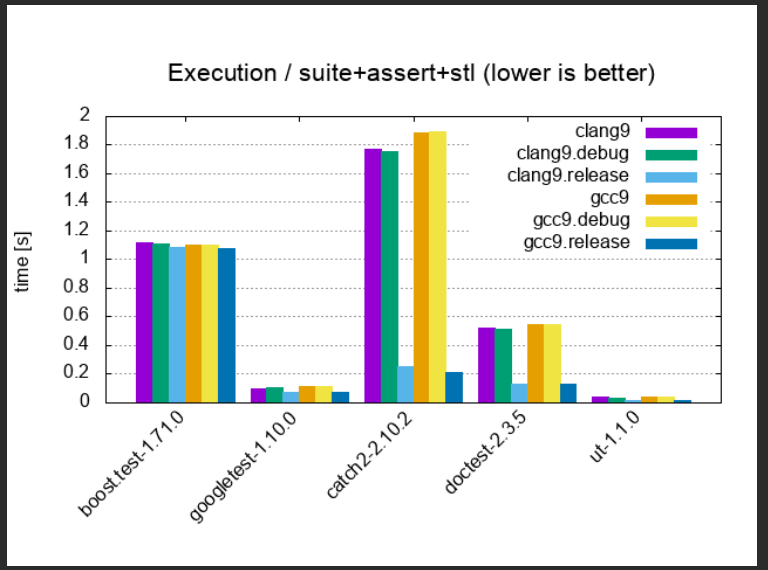
Benchmarks - Include

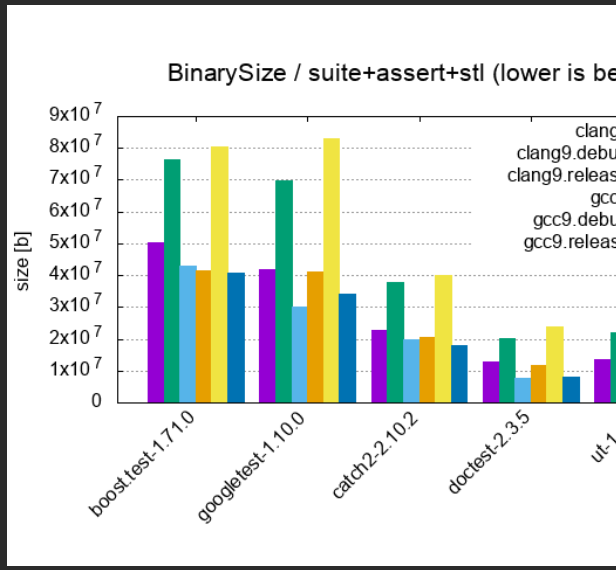


0 tests, 0 asserts, 1 cpp file

Benchmarks - Suite+Assert+STL

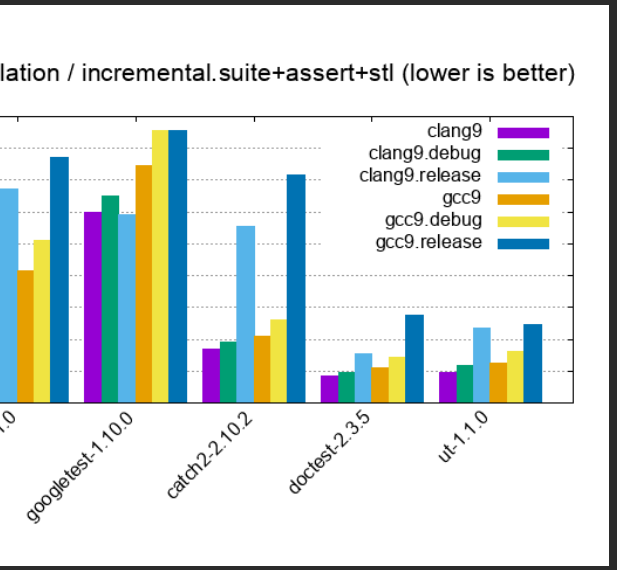


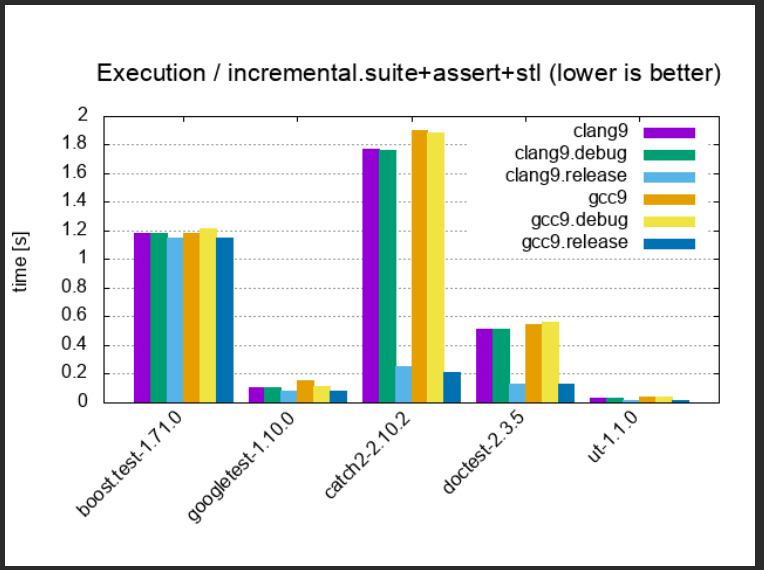


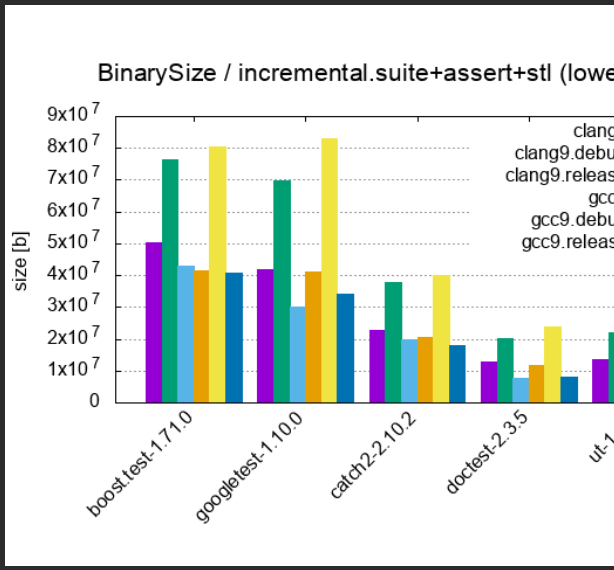


10'000 tests, 20'000 asserts, 100 cpp files

Benchmarks - Incremental Build

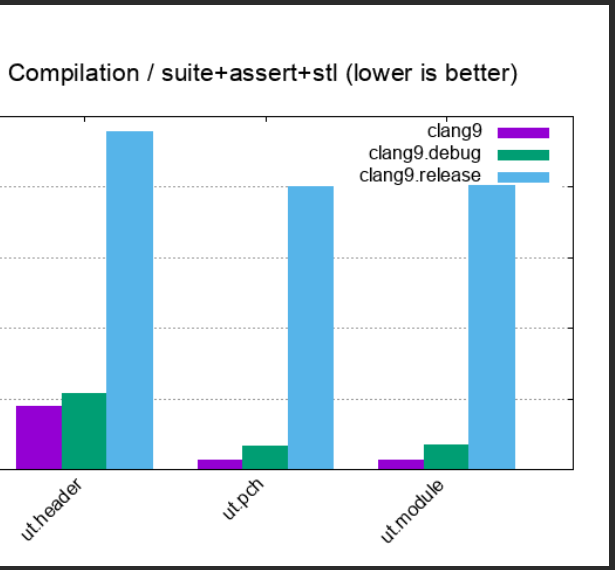


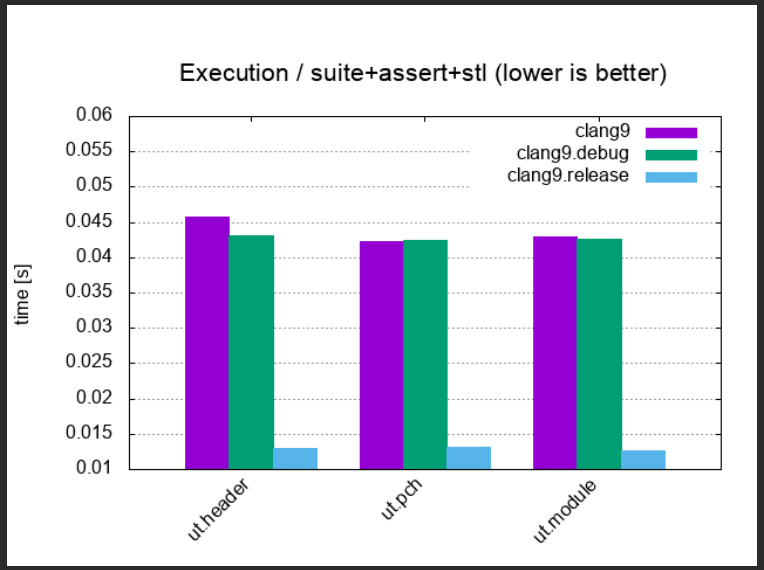


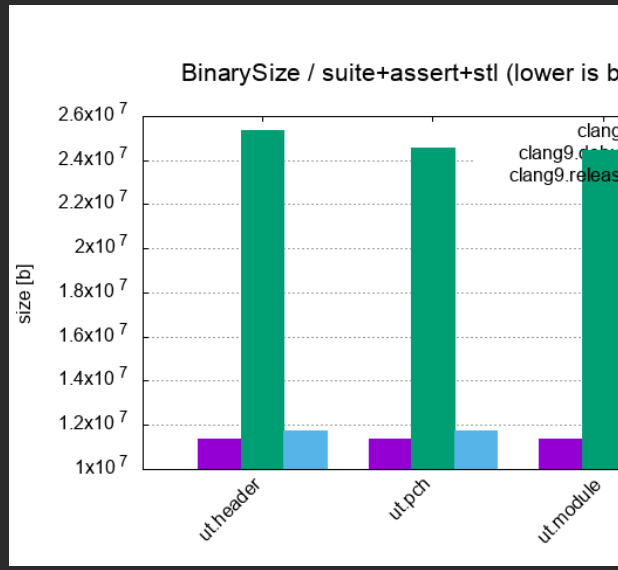


1 cpp file change (1'000 tests, 20'000 asserts, 100 cpp files)

Benchmarks - Headers vs Precompiled headers vs C++20 Modules







10'000 tests, 20'000 asserts, 100 cpp files

Summary

Summary

- C++20 enables a cleaner design and implementation

Summary

- C++20 enables a cleaner design and implementation
- UT is an example of cutting edge Unit Testing Framework

Summary

- C++20 enables a cleaner design and implementation
- UT is an example of cutting edge Unit Testing Framework
- Possible standardization of Unit Testing primitives?

Summary

- C++20 enables a cleaner design and implementation
- UT is an example of cutting edge Unit Testing Framework
- Possible standardization of Unit Testing primitives?
- Macro-based Frameworks can be built on top of Unit Testing primitives

Catch2 - https://godbolt.org/z/jfb7jK

Catch2 - https://godbolt.org/z/jfb7jK

```
#define REQUIRE(...)    ut::expect(that % __VA_ARGS__)
#define TEST_CASE(...) ut::test{"test", __VA_ARGS__} = [=]() mutable
#define SECTION(name)  ut::test{"section", name} = [=]() mutable
```

Catch2 - https://godbolt.org/z/jfb7jK

```
#define REQUIRE(...)    ut::expect(that % __VA_ARGS__)
#define TEST_CASE(...) ut::test{"test", __VA_ARGS__} = [=]() mutable
#define SECTION(name)  ut::test{"section", name} = [=]() mutable
```

```
TEST_CASE("vectors can be sized and resized", "[vector]") {
  std::vector<int> v(5);

  SECTION("resize bigger") {
    v.resize(10);
    REQUIRE(10 == std::size(v));
  };
};
```

Catch2 - https://godbolt.org/z/jfb7jK

```
#define REQUIRE(...)   ut::expect(that % __VA_ARGS__)
#define TEST_CASE(...) ut::test{"test", __VA_ARGS__} = [=]() mutable
#define SECTION(name)  ut::test{"section", name} = [=]() mutable
```

```
TEST_CASE("vectors can be sized and resized", "[vector]") {
  std::vector<int> v(5);

  SECTION("resize bigger") {
    v.resize(10);
    REQUIRE(10 == std::size(v));
  };
};
```

```
./catch2
 -> All tests passed (1 asserts in 1 tests)
```

If you liked it then you *"should have put a"_ test* on it!

Beyonce rule

https://www.quantlab.com/careers