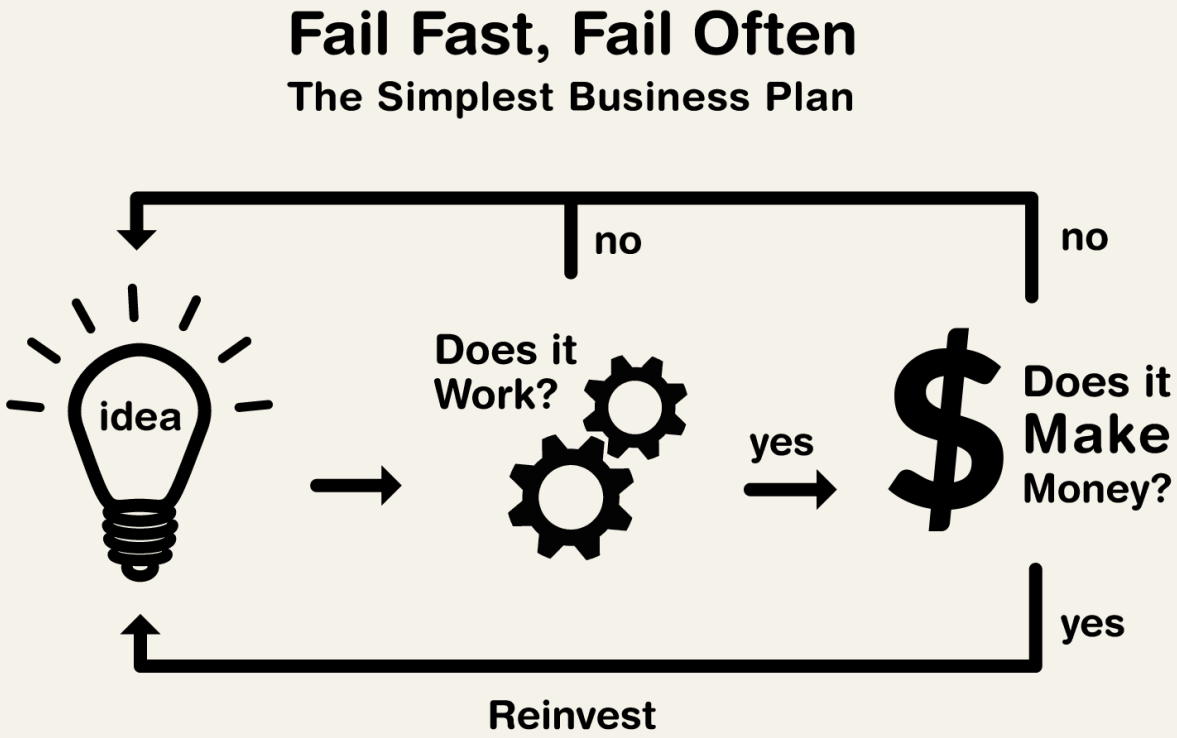
sonar::expected

Toward zero-exception

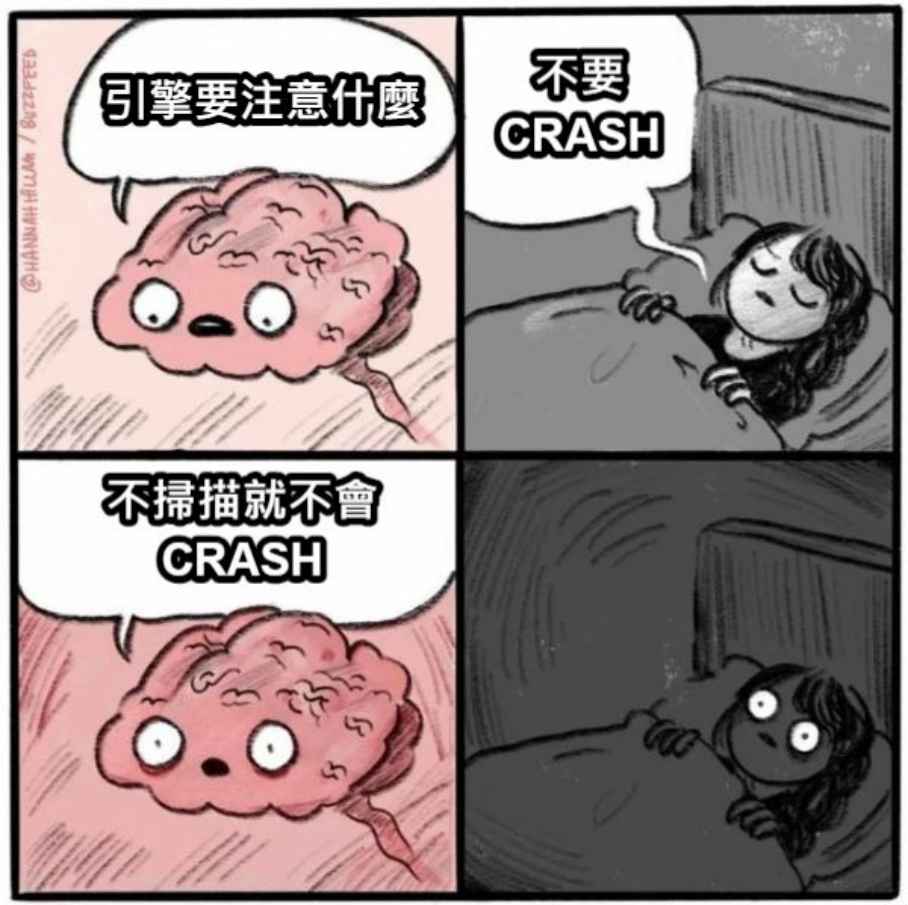
scc

TEAMT5
杜 浦 數 位 安 全
Persistent Cyber Threat Hunters

Product stability



Product stability



Zero Exception

Recall: Better main

TEAMT5 杜浦數位安全

```cpp
template <typename T, typename E>
class Expected
{
    T data;
    E error;

public:
    Expected(T data, E error) : data(data), error(error) {}
    Expected(T data) : data(data), error(E()) {}
    Expected(E error) : data(T()), error(error) {}
    [[nodiscard]] bool operator()() const noexcept { return !error; }
    [[nodiscard]] bool operator!() const noexcept { return error; }
    [[nodiscard]] bool is_success() const noexcept { return !error; }
    [[nodiscard]] bool is_error() const noexcept { return error; }
    [[nodiscard]] T get() const noexcept { return data; }
    [[nodiscard]] E get_error() const noexcept { return error; }
};
```

TEAMT5 杜浦數位安全

std::expected since C++23

TEAMT5
杜 浦 數 位 安 全

std::expected

Defined in header <expected>

```
template< class T, class E >
class expected;
```

(since C++23)

TL;DR How to use?

TEAMT5 杜浦數位安全

```
enum class sonar_error_code
{
    invalid_input = 1,
    overflow,
    nan,
};
```

```
class sonar_error : public std::error_code {
public:
    sonar_error() noexcept = default;
    sonar_error(sonar_error_code err) noexcept
    : std::error_code(static_cast<int>(err), sonar_error_category_impl{}) {}
    sonar_error(std::error_code ec) noexcept : std::error_code(ec) {}
private:
    ...
};
```

```
auto make_sonar_error(sonar_error_code sec) ->
expected<int, sonar_error> {
    return sonar_error(sec);
}
```

https://godbolt.org/z/7K1zqGdGn

TL;DR How to use?

TEAMT5 杜浦數位安全

```
auto parse_number(std::string_view input) -> sonar::expected<double, sonar_error> {
    const char* begin = input.data(), *end;
    double retval = std::strtod(begin, &end);
    using namespace sonar_error_code;

    if (begin == end)
        return make_sonar_error(invalid_input);
    else if (std::isinf(retval))
        return make_sonar_error(overflow);
    else if (std::isnan(retval))
        return make_sonar_error(nan);

    return retval;
}
```

https://godbolt.org/z/7K1zqGdGn

TL;DR How to use?

TEAMT5 杜浦數位安全

```
auto process = [](std::string_view str) {
    std::cout << "str: " << std::quoted(str) << ", ";
    if (const auto num = parse_number(str); num.has_value())
        std::cout << "value: " << *num << '\n';
    else if (num.error().value() == static_cast<int>(sonar_error_code::invalid_input))
        std::cout << "error: invalid input\n";
    else if (num.error().value() == static_cast<int>(sonar_error_code::overflow))
        std::cout << "error: overflow\n";
    else if (num.error().value() == static_cast<int>(sonar_error_code::nan))
        std::cout << "error: NaN\n";
    else
        std::cout << "unexpected!\n";
};
```

```
Program returned: 0
    str: "42", value: 42
    str: "42abc", value: 42
    str: "meow", error: invalid input
    str: "inf", error: overflow
    str: "nan", error: NaN
```

```
int main() {
    for (const auto& src :
        {"42", "42abc", "meow", "inf", "nan"})
    process(src);
}
```

https://godbolt.org/z/7K1zqGdGn

Extra features:

TEAMT5 杜浦數位安全

```
auto demo() -> sonar::rebind<int> {
    return sonar::rebind<int>(1, sonar::make_sonar_error(1));
}

int main() {
    const auto &[result, err] = demo();
    std::cout << "result: " << result << ", err: " <<
    err.message() << std::endl;
    return 0;
}
```

Learned from Golang.

Extra features:

TEAMT5 杜浦數位安全

```
auto demo(int v) -> sonar::rebind<int> {
    return sonar::rebind<int>(1, sonar::make_sonar_error(v));
}

int main() {
    auto e1 = demo(1), e2 = demo(2), e3 = demo(0), e4 = demo(0);
    std::cout << (e1 && e2) << "\n"
              << (e1 || e2) << "\n"
              << (e3 && e4) << "\n"
              << (e3 || e4) << "\n";
    return 0;
}
```

Learned from Rust.

Growth zone

P0323r5 std::expected

errno

- + General
- - Minimize soft errors
- + Centralized handling
- + Local handling
- - Arbitrary amount of error info
- + Little cost on the normal path
- - Make correct code easy to write
  - Error handling entirely optional
  - Threading issues

Special Value

- - General (won't work with surjective functions)
- - Minimize soft errors
- - Centralized handling
- + Local handling
- - Arbitrary amount of error info
- ? Little cost on the normal path
- - Make correct code easy to write
  - Error handling often optional
  - Error handling code intertwined with normal code

https://cppeurope.com/wp-content/uploads/2018/02/Andrei_Alexandrescu_Expect_the_expected_slides.pdf

https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0323r5.html

errno

- + G
- - Mi
- + C
- + L
- - Ar
- + L
- - Ma
  -
  -

Value of Separate Type

- + General
- ? Minimize soft errors
- - Centralized handling
- + Local handling
- + Arbitrary amount of error info
- + Little cost on the normal path
- - Make correct code easy to write
  - Error handling requires extra code & data

```
long strtol(const char*s, const char**e, int r);
```


https://cppeur les.pdf

https://www.o

In Rust?

TEAMT5
杜 浦 數 位 安 全

```rust
enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

```rust
let x: Result<i32, &str> = Ok(-3);
assert_eq!(x.is_ok(), true);

let x: Result<i32, &str> =
Err("Some error message");
 assert_eq!(x.is_ok(), false);
```

https://doc.rust-lang.org/std/result/index.html

In Rust?

TEAMT5
杜 浦 數 位 安 全

Results must be used

A common problem with using return values to indicate errors is that it is easy to ignore the return value, thus failing to handle the error. `Result` is annotated with the `#[must_use]` attribute, which will cause the compiler to issue a warning when a Result value is ignored. This makes `Result` especially useful with functions that may encounter errors but don't otherwise return a useful value.

The question mark operator, `?`

When writing code that calls many functions that return the `Result` type, the error handling can be tedious. The question mark operator, `?`, hides some of the boilerplate of propagating errors up the call stack.

https://doc.rust-lang.org/std/result/index.html

In Rust?

TEAMT5 杜浦數位安全

Boolean operators

These methods treat the `Result` as a boolean value, where `Ok` acts like `true` and `Err` acts like `false`. There are two categories of these methods: ones that take a `Result` as input, and ones that take a function as input (to be lazily evaluated).

The `and` and `or` methods take another `Result` as input, and produce a `Result` as output. The `and` method can produce a `Result<U, E>` value having a different inner type `U` than `Result<T, E>`. The `or` method can produce a `Result<T, F>` value having a different error type `F` than `Result<T, E>`.

method	self	input	output
`and`	`Err(e)`	(ignored)	`Err(e)`
`and`	`Ok(x)`	`Err(d)`	`Err(d)`
`and`	`Ok(x)`	`Ok(y)`	`Ok(y)`
`or`	`Err(e)`	`Err(d)`	`Err(d)`
`or`	`Err(e)`	`Ok(y)`	`Ok(y)`
`or`	`Ok(x)`	(ignored)	`Ok(x)`

The `and_then` and `or_else` methods take a function as input, and only evaluate the function when they need to produce a new value. The `and_then` method can produce a `Result<U, E>` value having a different inner type `U` than `Result<T, E>`. The `or_else` method can produce a `Result<T, F>` value having a different error type `F` than `Result<T, E>`.

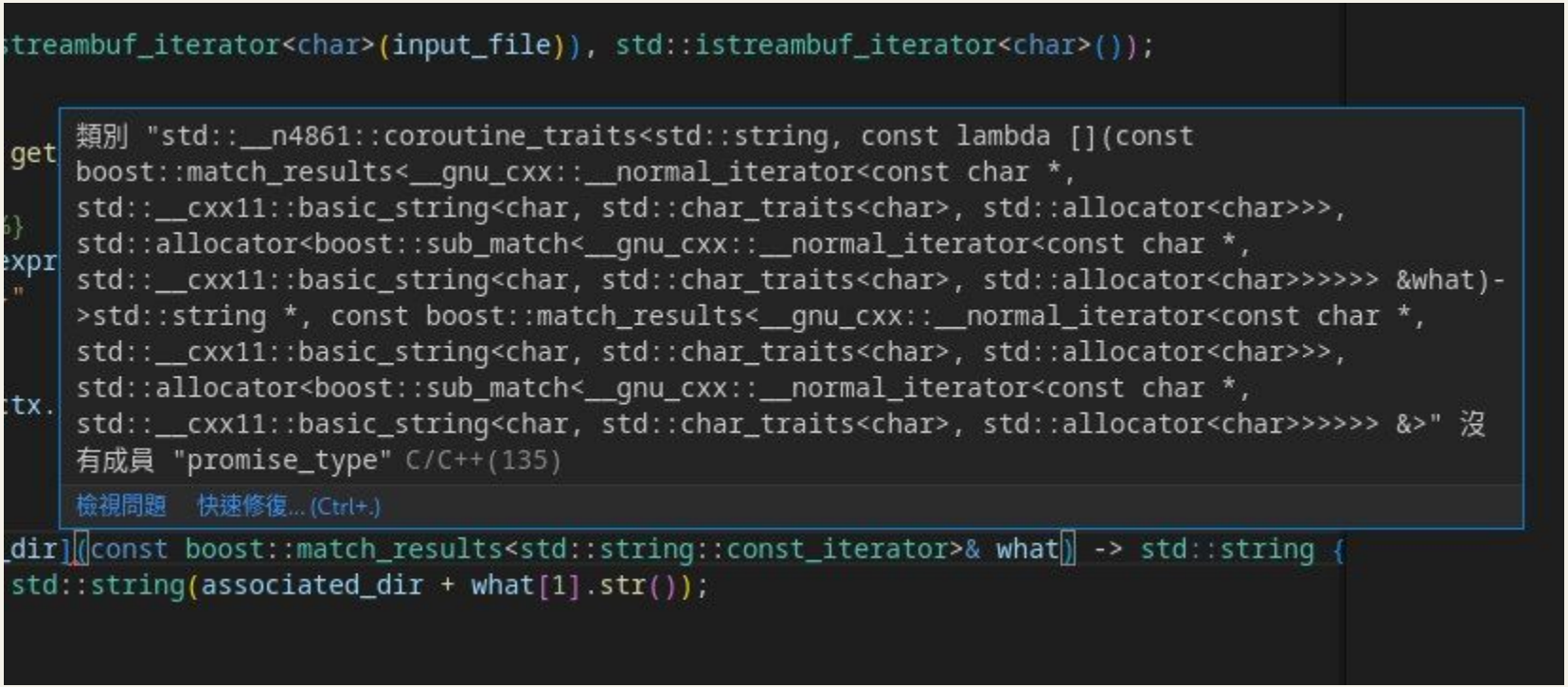
method	self	function input	function result	output
`and_then`	`Err(e)`	(not provided)	(not evaluated)	`Err(e)`
`and_then`	`Ok(x)`	`x`	`Err(d)`	`Err(d)`
`and_then`	`Ok(x)`	`x`	`Ok(y)`	`Ok(y)`
`or_else`	`Err(e)`	`e`	`Err(d)`	`Err(d)`
`or_else`	`Err(e)`	`e`	`Ok(y)`	`Ok(y)`
`or_else`	`Ok(x)`	(not provided)	(not evaluated)	`Ok(x)`

https://doc.rust-lang.org/std/result/index.html

concepts and constraints

concepts and constraints

TEAMT5 杜浦數位安全



concepts and constraints

TEAMT5
杜 浦 數 位 安 全

```cpp
template<typename T>
concept Hashable = requires(T a) {
{ std::hash<T>{}(a) } -> std::convertible_to<std::size_t>;
};
template<Hashable T>
void f(T) {
    std::unordered_map<T, int> table{};
}

int main() {
    struct meow {};
    f(meow{}); // Error: meow does not satisfy Hashable
}
```

https://godbolt.org/z/e6Kzvxdv4

```
template<typename T>

concept Hashable = requires(T a) {
```


```
<source>: In function 'int main()':
<source>:16:6: error: no matching function for call to 'f(main()::meow)'
   16 |     f(meow{}); // Error: meow does not satisfy Hashable
      |     ~^~~~~~~~
<source>:10:6: note: candidate: 'template<class T>  requires  Hashable<T> void f(T)'
   10 | void f(T) {
      |      ^
<source>:10:6: note:   template argument deduction/substitution failed:
<source>:10:6: note: constraints not satisfied
<source>: In substitution of 'template<class T>  requires  Hashable<T> void f(T) [with T = main()::meow]':
<source>:16:6:   required from here
<source>:5:9:   required for the satisfaction of 'Hashable<T>' [with T = main::meow]
<source>:5:20:   in requirements with 'T a' [with T = main::meow]
<source>:6:21: note: the required expression 'std::hash<_Tp>{}(a)' is invalid
    6 |     { std::hash<T>{}(a) } -> std::convertible_to<std::size_t>;
      |       ~~~~~~~~~~~~~~~~^~~
cc1plus: note: set '-fconcepts-diagnostics-depth=' to at least 2 for more detail
Compiler returned: 1
```

```
    f(meow{}); // Error: meow does not satisfy Hashable
}
```

https://godbolt.org/z/e6Kzvxdv4

```
template<typename T>
void f(T) {
std::unordered_map<T, int> table{};
}
int main() {
    struct meow {};
    f(meow{}); // Error: meow does not satisfy Hashable
}
```

TEAMT5
杜浦數位安全

https://godbolt.org/z/e6Kzvxdv4

x86-64 gcc 13.1 (Editor #2) | Output of x86-64 gcc 13.1 (Compiler #2)

A | Wrap lines | Select all

```
<source>: In instantiation of 'void f(T) [with T = main()::meow]':
<source>:10:6:   required from here
<source>:5:32: error: use of deleted function 'std::unordered_map<_Key, _Tp, _Hash, _Pred, _Alloc>::unordered_map() [with _Key = main()::meow; _Tp =
    5 |     std::unordered_map<T, int> table{};
      |                                ^~~~~
In file included from /opt/compiler-explorer/gcc-13.1.0/include/c++/13.1.0/unordered_map:41,
                 from <source>:1:
/opt/compiler-explorer/gcc-13.1.0/include/c++/13.1.0/bits/unordered_map.h:146:7: note: 'std::unordered_map<_Key, _Tp, _Hash, _Pred, _Alloc>::unorder
  146 |       unordered_map() = default;
      |       ^~~~~~~~~~~~~
/opt/compiler-explorer/gcc-13.1.0/include/c++/13.1.0/bits/unordered_map.h:146:7: error: use of deleted function 'std::_Hashtable<_Key, _Value, _Allo
In file included from /opt/compiler-explorer/gcc-13.1.0/include/c++/13.1.0/bits/unordered_map.h:33:
/opt/compiler-explorer/gcc-13.1.0/include/c++/13.1.0/bits/hashtable.h:530:7: note: 'std::_Hashtable<_Key, _Value, _Alloc, _ExtractKey, _Equal, _Hash
  530 |       _Hashtable() = default;
      |       ^~~~~~~~~~
/opt/compiler-explorer/gcc-13.1.0/include/c++/13.1.0/bits/hashtable.h:530:7: error: use of deleted function 'std::__detail::_Hashtable_base<_Key, _V
In file included from /opt/compiler-explorer/gcc-13.1.0/include/c++/13.1.0/bits/hashtable.h:35:
/opt/compiler-explorer/gcc-13.1.0/include/c++/13.1.0/bits/hashtable_policy.h:1710:7: note: 'std::__detail::_Hashtable_base<_Key, _Value, _ExtractKey
 1710 |       _Hashtable_base() = default;
      |       ^~~~~~~~~~~~~~~
/opt/compiler-explorer/gcc-13.1.0/include/c++/13.1.0/bits/hashtable_policy.h:1710:7: error: use of deleted function 'std::__detail::_Hash_code_base<
/opt/compiler-explorer/gcc-13.1.0/include/c++/13.1.0/bits/hashtable_policy.h:1297:7: note: 'std::__detail::_Hash_code_base<_Key, _Value, _ExtractKey
 1297 |       _Hash_code_base() = default;
      |       ^~~~~~~~~~~~~~~
/opt/compiler-explorer/gcc-13.1.0/include/c++/13.1.0/bits/hashtable_policy.h:1297:7: error: use of deleted function 'std::__detail::_Hashtable_ebo_h
/opt/compiler-explorer/gcc-13.1.0/include/c++/13.1.0/bits/hashtable_policy.h:1211:12: note: 'std::__detail::_Hashtable_ebo_helper<1, std::hash<main(
 1211 |     struct _Hashtable_ebo_helper<_Nm, _Tp, true>
```

https://godbolt.org/z/e6Kzvxdv4

```
template<typename T>

concept Hashable = requires(T a) {
```

```
<source>: In function 'int main()':
<source>:16:6: error: no matching function for call to 'f(main()::meow)'
   16 |     f(meow{}); // Error: meow does not satisfy Hashable
      |     ~^~~~~~~~
<source>:10:6: note: candidate: 'template<class T>  requires  Hashable<T> void f(T)'
   10 | void f(T) {
      |      ^
<source>:10:6: note:   template argument deduction/substitution failed:
<source>:10:6: note: constraints not satisfied
<source>: In substitution of 'template<class T>  requires  Hashable<T> void f(T) [with T = main()::meow]':
<source>:16:6:   required from here
<source>:5:9:   required for the satisfaction of 'Hashable<T>' [with T = main::meow]
<source>:5:20:   in requirements with 'T a' [with T = main::meow]
<source>:6:21: note: the required expression 'std::hash<_Tp>{}(a)' is invalid
    6 |     { std::hash<T>{}(a) } -> std::convertible_to<std::size_t>;
      |       ~~~~~~~~~~~~~~~~^~~
cc1plus: note: set '-fconcepts-diagnostics-depth=' to at least 2 for more detail
Compiler returned: 1
```

```
f(meow{}); // Error: meow does not satisfy Hashable

}
```

https://godbolt.org/z/e6Kzvxdv4

sonar::expected

```
61
62  int main() {
63      expected<int, std::error_code> e;
64      expected<int, bool> eee;
65  }
```

Output of x86-64 clang 8.0.0 (Compiler #1)

A ▾ | Wrap lines | Select all

```
<source>:20:5: error: static_assert failed due to requirement
'std::is_base_of<std::error_code, bool>::value' "E must be
inherited from std::error_code"
    static_assert(std::is_base_of<std::error_code, E>::value,
"E must be inherited from std::error_code");
    ^              ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
<source>:64:25: note: in instantiation of template class
'detail::expected_impl_<int, bool, void>' requested here
    expected<int, bool> eee;
                        ^

1 error generated.
Compiler returned: 1
```

https://godbolt.org/z/4j8Kd5hhe

```cpp
int main() {
    expected<int, std::error_code> e;
    expected<int, bool> eee;
    std::error_code real_error = eee.error();
}
```

std::enable_if

We use std::enable_if to simulate the **concept** since C++20.

Possible implementation

TEAMT5
杜 浦 數 位 安 全

```
template<bool B, class T = void>
struct enable_if {};

template<class T>
struct enable_if<true, T> { typedef T type; };
```

```
template <typename T, typename E>
using expected = detail::expected_impl_<T, E>;
```

```
template <typename T, typename E, typename = void>
class expected_impl_;

template <typename T, typename E>
class expected_impl_<T, E,
                     typename std::enable_if<!can_be_expected<T, E>>
                     ::type>
{};

template <typename T, typename E>
class expected_impl_<T, E,
                     typename std::enable_if<can_be_expected<T, E>>
                     ::type>
{}
```

```
template <typename T, typename E, typename = void>
class expected_impl_;
```

擴增參數

```
template <typename T, typename E>
class expected_impl_<T, E,
                    typename std::enable_if<!can_be_expected<T, E>>
                    ::type>
{};

template <typename T, typename E>
class expected_impl_<T, E,
                    typename std::enable_if<can_be_expected<T, E>>
                    ::type>
{}
```

```
template <typename T, typename E, typename = void>
class expected_impl_;

template <typename T, typename E>
class expected_impl_<T, E,
                     typename std::enable_if<!can_be_expected<T, E>>
                     ::type>
{};

template <typename T, typename E>
class expected_impl_<T, E,
                     typename std::enable_if<can_be_expected<T, E>>
                     ::type>
{}
```

類似 concept 的東西

```
template <typename T, typename E>
class expected_impl_<T, E,
                     typename std::enable_if<!can_be_expected<T, E>>
                     ::type>
{};
```

類似 concept 的東西

```
template <typename T, typename E>
constexpr bool can_be_expected =
                     !std::is_void_v<T> &&
                     std::is_default_constructible<T>::value &&
                     std::is_base_of<std::error_code, E>::value;
```

```
template <typename T, typename E>
class expected_impl_<T, E,
                     typename std::enable_if<!can_be_expected<T, E>>
                     ::type>
{};
```

類似 concept 的東西

```
template <typename T, typename E>
constexpr bool can_be_expected =
                     !std::is_void_v<T> &&
                     std::is_default_constructible<T>::value &&
                     std::is_base_of<std::error_code, E>::value;
```

```
template <typename T, typename E, typename = void>
class expected_impl_;

template <typename T, typename E>
class expected_impl_<T, E,
                     typename std::enable_if<!can_be_expected<T, E>>
                     ::type>
{};
```

類似 concept 的東西

```
template <typename T, typename E>
constexpr bool can_be_expected =
                     !std::is_void_v<T> &&
                     std::is_default_constructible<T>::value &&
                     std::is_base_of<std::error_code, E>::value;
```

```
template <typename T, typename E>
class expected_impl_<T, E,
                     typename std::enable_if<!can_be_expected<T, E>>
                     ::type>
{};
```

類似 concept 的東西

```
template <typename T, typename E>
constexpr bool can_be_expected =
                     !std::is_void_v<T> &&
                     std::is_default_constructible<T>::value &&
                     std::is_base_of<std::error_code, E>::value;
```

```
template <typename T, typename E, typename = void>
class expected_impl_;

template <typename T, typename E>
class expected_impl_<T, E,
                     typename std::enable_if<!can_be_expected<T, E>>
                     ::type>
{};

template <typename T, typename E>
class expected_impl_<T, E,
                     typename std::enable_if<can_be_expected<T, E>>
                     ::type>
{}
```

```
template <typename T, typename E, typename = void>
class expected_impl_;

template <typename T, typename E>
class expected_impl_<T, E,
                     typename std::enable_if<!can_be_expected<T, E>>
                     ::type>
{};

template <typename T, typename E>
class expected_impl_<T, E, true>
{}
```

```
template <typename T, typename E, typename = void>
class expected_impl_;
```

```
template <typename T, typename E>
class expected_impl_<T, E, void::type>
{};

template <typename T, typename E>
class expected_impl_<T, E, true>
{}
```

TEAMT5
杜 浦 數 位 安 全

```
template <typename T, typename E, typename = void>
class expected_impl_;
```

SFINAE

```
template <typename T, typename E>
class expected_impl_<T, E, void::type>
{};

template <typename T, typename E>
class expected_impl_<T, E, true>
{}
```

SFINAE

Substitution Failure Is Not An Error.

When substituting the explicitly specified or deduced type
for the template parameter fails,
the specialization is discarded from the overload set
instead of causing a compile error.

TEAMT5
杜 浦 數 位 安 全

```
template <typename T, typename E, typename = void>
class expected_impl_;
```

~~template <typename T, typename E>~~
~~class expected_impl_<T, E, void::type>~~
~~{};~~

SFINAE

```
template <typename T, typename E>
class expected_impl_<T, E, true>
{}
```

```
template <typename T, typename E, typename = void>
class expected_impl_;

template <typename T, typename E>
class expected_impl_<T, E,
                     typename std::enable_if<!can_be_expected<T, E>>
                     ::type>
{};

template <typename T, typename E>
class expected_impl_<T, E,
                     typename std::enable_if<can_be_expected<T, E>>
                     ::type>
{}
```

So, our supportion:

TEAMT5 杜浦數位安全

Member types

Member type	Definition
value_type (C++23)	T
error_type (C++23)	E
unexpected_type (C++23)	std::unexpected<E>
rebind (C++23)	`template< class U > using rebind = expected<U, error_type>;`

Member functions

(constructor) (C++23)	constructs the expected object (public member function)
(destructor) (C++23)	destroys the expected object, along with its contained value (public member function)
operator= (C++23)	assigns contents (public member function)

Observers

operator-> operator* (C++23)	accesses the expected value (public member function)
operator bool has_value (C++23)	checks whether the object contains an expected value (public member function)
value (C++23)	returns the expected value (public member function)
error (C++23)	returns the unexpected value (public member function)
value_or (C++23)	returns the expected value if present, another value otherwise (public member function)

Monadic operations

and_then (C++23)	returns the result of the given function on the expected value if it exists; otherwise, returns the expected itself (public member function)
transform (C++23)	returns an expected containing the transformed expected value if it exists; otherwise, returns the expected itself (public member function)
or_else (C++23)	returns the expected itself if it contains an expected value; otherwise, returns the result of the given function on the unexpected value (public member function)
transform_error (C++23)	returns the expected itself if it contains an expected value; otherwise, returns an expected containing the transformed unexpected value (public member function)

Modifiers

emplace (C++23)	constructs the expected value in-place (public member function)
swap (C++23)	exchanges the contents (public member function)

Non-member functions

operator== (C++23)	compares expected objects (function template)
swap(std::expected) (C++23)	specializes the std::swap algorithm (function)

Helper classes

unexpected (C++23)	represented as an unexpected value (class template)
bad_expected_access (C++23)	exception indicating checked access to an expected that contains an unexpected value (class template)
unexpect_t unexpect (C++23)	in-place construction tag for unexpected value in expected (class) (constant)

- Structured binding
- Logic operators && || & |

std::expected<T,E>::expected

`constexpr expected() noexcept(/* see below */);`	(1)	(since C++23)
`constexpr expected( const expected& other );`	(2)	(since C++23)
`constexpr expected( expected&& other ) noexcept(/* see below */);`	(3)	(since C++23)
`template< class U, class G >` `constexpr explicit(/* see below */) expected( const expected<U, G>& other );`	(4)	(since C++23)
`template< class U, class G >` `constexpr explicit(/* see below */) expected( expected<U, G>&& other );`	(5)	(since C++23)
`template< class U = T >` `constexpr explicit(!std::is_convertible_v<U, T>) expected( U&& v );`	(6)	(since C++23) (T is not *cv* void)

`template< class G >`
`constexpr explicit(!std::is_convertible_v<const G&, E>)`
`  expected( const std::unexpected<G>& e );`
	(7)	(since C++23)

`template< class G >`
`constexpr explicit(!std::is_convertible_v<G, E>)`
`  expected( std::unexpected<G>&& e );`
	(8)	(since C++23)

`template< class... Args >` `constexpr explicit expected( std::in_place_t, Args&&... args );`	(9)	(since C++23) (T is not *cv* void)

`template< class U, class... Args >`
`constexpr explicit expected( std::in_place_t,`
`  std::initializer_list<U> il, Args&&... args );`
	(10)	(since C++23)
		(T is not *cv* void)

`template< class... Args >` `constexpr explicit expected( std::in_place_t ) noexcept;`	(11)	(since C++23) (T is *cv* void)
`template< class... Args >` `constexpr explicit expected( std::unexpect_t, Args&&... args );`	(12)	(since C++23)

`template< class U, class... Args >`
`constexpr explicit expected( std::unexpect_t,`
`  std::initializer_list<U> il, Args&&... args );`
	(13)	(since C++23)

std::expected<T,E>::expected

```
constexpr expected() noexcept(/* see below */);                                   (1)  (since C++23)
constexpr expected( const expected& other );                                      (2)  (since C++23)
constexpr expected( expected&& other ) noexcept(/* see below */);                 (3)  (since C++23)
template< class U, class G >
constexpr explicit(/* see below */) expected( const expected<U, G>& other );     (4)  (since C++23)
template< class U, class G >
constexpr explicit(/* see below */) expected( expected<U, G>&& other );          (5)  (since C++23)
template< class U = T >
constexpr explicit(!std::is_convertible_v<U, T>) expected( U&& v );              (6)  (since C++23)
template< class G >
constexpr explicit(!std::is_convertible_v<const G&, E>) expected( const std::unexpected<G>& e );   (7)  (since C++23)
template< class G >
constexpr explicit(!std::is_convertible_v<G, E>) expected( std::unexpected<G>&& e );               (8)  (since C++23)
template< class... Args >
constexpr explicit expected( std::in_place_t, Args&&... args );                   (9)  (since C++23)
                                                                                       (T is not cv void)
template< class U, class... Args >
constexpr explicit expected( std::in_place_t, std::initializer_list<U> il, Args&&... args );   (10)  (since C++23)
                                                                                                    (T is not cv void)
constexpr expected( std::in_place_t ) noexcept;                                   (11)  (since C++23)
                                                                                       (T is cv void)
template< class... Args >
constexpr explicit expected( std::unexpect_t, Args&&... args );                   (12)  (since C++23)
template< class U, class... Args >
constexpr explicit expected( std::unexpect_t,
                             std::initializer_list<U> il, Args&&... args );       (13)  (since C++23)
```

constexpr ?!

constexpr

This is a big issue, it might need another demo meeting to introduce.

constexpr

- C++11, 14, 17, 20 differences
- constinit, consteval, constexpr
- cv-qualifiers

Thank you

scc@teamt5.org