Structured binding declaration as a condition

Zhihao Yuan
2024/6/26
Example 1

```cpp
if (auto [first, last] = parse(begin(), end()))
{
    // interpret [first, last) into a value
}
```
The return type implements two protocols

```cpp
struct parse_window
{
    char const *first, *last;

    explicit operator bool() const noexcept
    {
        return first != last;
    }
};
```

```cpp```
parse_window parse(char const*, char const*);
```cpp```

Structured binding protocol

condition protocol
if (auto [first, last] = parse(begin(), end()); first != last)
{
    // interpret [first, last) into a value
}
Problem solved in example 1

```c++
if (auto [first, last] = parse(begin(), end()); first != last) {

```

Information about the condition is spread across the components

How do I reconstruct the condition?
if (auto [first, last] = parse(begin(), end()))
{
    // interpret [first, last) into a value
}
auto e = parse(begin(), end());
bool t(e.operator bool());
if (t)
{
    condition
}
auto e = parse(begin(), end());
using T1 = decltype((e.first));
using T2 = decltype((e.last));
T1 first = e.first;
T2 last = e.last;
auto e = parse(begin(), end());
using T1 = decltype((e.first));
using T2 = decltype((e.last));
bool t(e.operator bool());
T1 first = e.first;
T2 last = e.last;
if (t)
{

decision variable

structured binding as a condition
The formulation allows moving get()

```cpp
std::generator<int> f()
{
    co_yield 1;
    co_yield 2;
}

if (auto g = f(); auto [a, b] = std::ranges::subrange{g})
{
    // ok
}
```
Move-only ranges

```cpp
template<std::size_t N, class I, class S, std::ranges::subrange_kind K>
requires (N < 2)
constexpr auto get(std::ranges::subrange<I, S, K>&& r)
{
    if constexpr (N == 0)
        return r.begin(); // may perform move construction
    else
        return r.end();
}
```
Operator bool usable only if range isn’t moved-out

```cpp
std::ranges::view_interface<D>::operator bool

constexpr explicit operator bool() requires /* see below */; (1) (since C++20)
constexpr explicit operator bool() const requires /* see below */; (2) (since C++20)
```

The default implementation of `operator bool` member function checks whether the view is non-empty. It makes the derived type contextually convertible to `bool`.

1) Let derived be `static_cast<D>(*this)`. The expression in the requires-clause is equal to
   ```cpp
   requires { ranges::empty(derived); } ,
   return !ranges::empty(derived); ;
   ```

2) Same as (1), except that derived is `static_cast<const D>(*this)`. 
auto e = parse(begin(), end());
using T1 = decltype((e.first));
using T2 = decltype((e.last));
bool t(e.operator bool());
T1 first = e.first;
T2 last = e.last;
if (t)
{

}
auto e = std::ranges::subrange{g};
using E = decltype(e);
using T1 = std::tuple_element<0, E>::type;
using T2 = std::tuple_element<1, E>::type;
bool t(e.operator bool());
T1&& a = get<0>(std::move(e));
T2&& b = get<1>(std::move(e));
if (t)
{
}
Where can I use this?

- if constexpr opt (init-statement opt condition ) statement else statement
- switch (init-statement opt condition ) statement
- while (condition ) statement
- for (init-statement condition opt ; expression opt ) statement
Example 2

```cpp
if (auto [ptr, ec] = std::to_chars(p, last, 42))
{
    // okay to proceed
}
else
{
    // handle errors
}
```
Without this paper in C++26

```cpp
if (auto result = std::to_chars(p, last, 42))
{
    auto [ptr, _] = result;
    // okay to proceed
}
else
{
    auto [ptr, ec] = result;
    // handle errors
}
```
...or this

```cpp
if (auto [ptr, ec] = std::to_chars(p, last, 42); ec != std::errc{})
{
    // okay to proceed
}
else
{
    // handle errors
}
```
Problem solved in example 2

```cpp
if (auto [ptr, ec] = std::to_chars(p, last, 42); ec != std::errc{}) {
```

A single component contains information about the condition

Still want to channel the information via the complete object
Example 3

iterative solver

```c
struct tableau {
    matrix Ap;
    vector bp, x, y;
};
```
Example 3
iterative solver

class tableau
{
    matrix Ap;
    vector bp, x, y;
    bool reached_optimal_;
};

// more code required

class Solver
{
    public:
        tableau solve();
        bool is_optimal(vector const&);
};
With this paper

```cpp
for (;;;;)
{
    // ...
    if (auto [Ap, bp, x, y] = solve())
        break;
}
```
Problem solved in example 3

for (;;)
{
    // ...
    auto [Ap, bp, x, y] = solve();

    if (is_optimal(x))
        break;
}

Reconstructing information from the components is not zero-cost
Example 4

```cpp
if (auto [city, state, zip] = ctre2::match<"(\w+), (\w+) (\d+)"(s))
{
    return location{ city, state, zip };}
```
Without this paper

```cpp
if (auto [all, city, state, zip] = ctre::match<"(\w+), (\w+) (\d+)">(s); all)
{
    return location{ city, state, zip };
}
```
Problem solved in example 4

```cpp
if (auto [all, city, state, zip] = 
    ctre::match<"(\w+), (\w+) (\d+)">(s); all)
{

All components but one have similar roles
```
Alternative solution with this paper in C++26

```cpp
if (auto [_, city, state, zip] =
    ctre::match<"(\w+), (\w+) (\d+)"">(s))
{
```
Implementation Experience
R1 is implemented in Clang

```
struct format_status
{
    format_errc ec;
    char *bp;
    explicit operator bool() const noexcept
    {
        return ec == format_errc::no_error;
    }
};

int main()
{
    if (auto [ok, ptr] = readint())
    {
        printf("stopped at %p\n", ptr);
    }
}
```
R2 semantics is not in the extension

```cpp
#include <generator>
#include <ranges>

std::generator<int> f() {
    co_yield 1;
    co_yield 2;
}

int main() {
    if (auto g = f();
        auto [b, e] = std::ranges::subrange(g)) {
            ^^^^^^^^  
        auto [b, e] = std::ranges::subrange(g)) {
            ^^^^^^^^  
    return 0;
}

Program terminated with signal: SIGSEGV
```
Thank you