Generalizing Constant Expressions in C++

Jens Maurer

What?
Why?
How?

1
What is a constant expression?

The compiler needs to know its value.

array bound  case label
bitfield length  enumerator initialiser
static member initialiser  template argument
null pointer constant  static_assert

static initialization — ROM-able data
Motivation for improvement

• template tricks for template value arguments
• surprising dynamic initialisation
• not enough ROM-able data
• macros vs. generic programming
• underused numeric_limits and bitmap operators

Goal: Create a general, fully-typed mechanism for compile-time evaluations.
Macros vs. generic programming

```cpp
static_assert(INT_MAX > 10000, "tiny");
```

Replace `INT_MAX` with

```cpp
std::numeric_limits<int>::max();
```

What if `int` is a template type parameter `T`?
Surprising dynamic initialisation

```c
struct S {
    static const int c;
};

const int d = 10 * S::c;
const int S::c = 5;
```
Type-safe bitmask types

Create a type with

- type-safe overloaded operator&, operator|
- ill-formed operator+, operator-
- usable as a compile-time constant
Constant-expression functions and data

```cpp
constexpr bool is_even(int number) {
    return number % 2 == 0;
}

constexpr int array_size =
    is_even(n) ? n : n + 1;

int my_array[array_size];
```
Literal types

```cpp
struct Complex {
    constexpr Complex(double r, double i)
        : re(r), im(i) { }
    constexpr double real() { return re; }
    double re, im;
};

constexpr double value = Complex(1.0, 2.0).real();
```
Thanks

WG21 Document N2116, “Generalized Constant Expressions — Revision 4” by Gabriel Dos Reis, Bjarne Stroustrup, Jens Maurer

WG21 Document N2219 “Constant Expressions in the Standard Library” by Gabriel Dos Reis, Bjarne Stroustrup

Questions?