rules.sonarsource.com

C++ static code analysis: "if constexpr" should be preferred to overloading for metaprogramming

2-3 minutes

C++17 version of the standards introduces `if constexpr`. If the `constexpr` keyword follows the `if` keyword in an if statement, then the `if` condition must be a constant and the `then` or `else` block is discarded at compile time, depending on the value of the constant.

More precisely, `if constexpr` branches that are discarded are not going to be instantiated. This behavior enables us to write some overloaded function templates in a more readable way: you don't need to use complex patterns (eg: by using `std::enable_if`) to make code compile.

This rule points out where a complex overloaded functions template could simply be replaced by `if constexpr`.

Noncompliant Code Example

```
template<typename Type>
typename std::enable_if_t<std::is_arithmetic_v<Type>>
process(Type&& type); // Noncompliant, this function can be combined with the one below

template<typename Type>
typename std::enable_if_t<!std::is_arithmetic_v<Type>>
process(Type&& type);

template <typename It, typename Distance>
void moveForward(It& it, Distance d, std::input_iterator_tag); // Noncompliant, this function can be combined with the one below

template <typename It, typename Distance, typename T>
void moveForward(It& it, Distance d, T);

template <typename It, typename Distance>
void moveForward(It& it, Distance d) { // Wrapper of the "moveForward" functions
    moveForward(it, d, typename std::iterator_traits<It>::iterator_category{} );
}
```

Compliant Solution

```
template<typename Type>
void process(Type&& type) {
    if constexpr(std::is_arithmetic_v<type>) {
        // implementation
    } else {
        // implementation
    }
}

template <typename It, typename Distance>
void moveForward(It& it, Distance d) { // Modifications have been directly done inside the wrapper
    if constexpr (std::iterator_traits<It>::input_iterator_tag) {
        // implementation
    } else {
        // implementation
    }
}
```