rules.sonarsource.com

C++ static code analysis: Function pointers should not be used as function parameters

2 minutes

When you want to define a function that can accept a function pointer as an argument, there are three ways in C++ to declare the parameter type:

- A function pointer: `void f(void (*callback)());`
- A `std::function`: `void f(std::function<void()> callback);`
- A template argument: `template<class Callback> void f(Callback callback);`

Using a function pointer is an inferior solution, for the following reasons:

- Only a function pointer can be passed as an argument, while the other options offer the caller more flexibility because they can take more advanced functors, such as lambdas with some captured state
- The syntax is obscure
- It typically has worse performance than the template parameter solution.

See {rule:cpp:S5213} for a discussion of how to choose between `std::function` and a template parameter.

Noncompliant Code Example

```
using Criterion = bool (*)(DataPoint const&);
void filter(DataSet* data, Criterion criterion); // Noncompliant

using Callback = void (*)(EventInfo const&);
class Button {
public:
  void addOnClick(Callback c) {myOnClickHandler = c;} // Noncompliant
private:
  Callback myOnClickHandler;
};
```

Compliant Solution

```
template<class Criterion>
void filter(DataSet* data, Criterion criterion); // Compliant, uses the more efficient template argument

using Callback = std::function<void(EventInfo const&)>;
class Button {
public:
  void addOnClick(Callback c) {myOnClickHandler = c;} // Compliant, uses the more flexible std::function
private:
  Callback myOnClickHandler;
};
```

See

- C++ Core Guidelines T.40 - Use function objects to pass operations to algorithms