C++ static code analysis: The global namespace should only contain "main", namespace declarations, and "extern" C declarations

2-3 minutes

Declaring names in appropriate namespaces reduces the number of names found during lookup, decreasing the risk of name clash and of surprising name lookup.

This rule raises an issue when a name in the global namespace has external linkage and therefore can be accessed from outside a particular translation unit.

Some names have to be declared in the global namespace, and are excluded from this rule:

- `main` (or its variants)
- Overloads of the global `new` & `delete` operators

In addition, no issue is raised for function definitions because they can only be accessed from different translation units through a forward declaration that will be flagged.

Noncompliant Code Example

```
int a; // Noncompliant
int b = 1; // Noncompliant
extern int  c = 1; // Noncompliant
extern const int d = 1; // Noncompliant

void f(); // Noncompliant

class A { // Noncompliant
};
```

Compliant Solution

```
namespace MY_API { // Compliant
int a;
int b = 1;
extern int  c = 1;
extern const int d = 1;

void  f();

class A {
};
} // namespace MY_API

namespace { // Compliant, anonymous namespace
  int  a = 1;
  void m2() {
  }
}

int main() { // Compliant, exception for main
}

static int a; // Compliant, internal linkage
static void m1(); // Compliant, internal linkage

const int a = 1; // Compliant, a global constant is implicitly static

extern "C" int a = 1; // Compliant

extern "C" void f1(); // Compliant
void f2() {} // Compliant

typedef int a; // Compliant, we don't detect aliases

void *operator new(size_t bytes, const X::Y& context) { return
X::malloc(bytes,context); } // Compliant by exception
```

void operator delete(void* ptr, const X::Y& context) {
X::free(bytes,context); } // Compliant by exception

See

- MISRA C++:2008, 7-3-1 - The global namespace shall only contain main, namespace declarations and extern "C" declarations.