C++ static code analysis: Size argument of memory functions should be consistent

1-2 minutes

The memory functions `memset`, `memcpy`, `memmove`, and `memcmp` take as last argument the number of bytes they will work on. If this size argument is badly defined (eg it is greater than the size of the destination object), it can lead to undefined behavior.

This rule raises an issue when the size argument of a memory function seems inconsistent with the other arguments of the function.

Noncompliant Code Example

```
struct A {};

void f() {
  struct A dest;
  memset(&dest, 0, sizeof(&dest)); // Noncompliant; size is based on "A*" when the destination is of type "A"
  struct A src;
  memcpy(&dest, &src, sizeof(&dest)); // Noncompliant; size is based on "A*" when the source is of type "A"

  if (memset(&dest, 0, sizeof(dest) != 0)) { // Noncompliant; size argument is a comparison
    // ...
  }
}
```

Compliant Solution

```
struct A {};

void f() {
  struct A dest;
  memset(&dest, 0, sizeof(dest)); // Compliant
  struct A src;
  memcpy(&dest, &src, sizeof(dest)); // Compliant

  if (memset(&dest, 0, sizeof(dest)) != 0) { // Compliant
    // ...
  }
}
```