rules.sonarsource.com

C++ static code analysis: Pointer conversions should be restricted to a safe subset

2 minutes

Casting an object pointer can very easily lead to undefined behavior. Only a few cases are supported, for instance casting an object pointer to a large enough integral type (and back again), casting an object pointer to a pointer to void (and back again)… Using a pointer cast to access an object as if it was of another type than its real type is not supported in general.

This rule detects casts between object pointers and incompatible types.

Noncompliant Code Example

```
struct S1 *p1;
struct S2;
void f ()
{
  (float) p1; // Noncompliant, conversion to floating point type
  (int *) p1; // Noncompliant
  float f;
  int *i = (int *)&f; // Noncompliant, undefined behavior even if
sizeof(int) == sizeof(float)
  (int) p1; // Compliant, but might be undefined behavior if 'int' is not
large enough to hold the value of p1.
  (void *) p1; // Compliant, conversion to 'void *'
  (struct S2 *)p1; // Noncompliant, conversion to another type.
}
```

Exceptions

In C, it is allowed to cast an object pointer to a character pointer to access the byte representation of the object. This rule ignores this case.

Anything can be safely cast to `void` (since nothing can be done with a result of this cast), and doing so is a common pattern to silence compiler warnings about unused variables. This rule ignores such casts.

```
void f(int *p) {
  (void)p;
}
```

See

- MISRA C:2004, 11.2 - Conversions shall not be performed between a pointer to object and any type other than an integral type, another pointer to object type or a pointer to void.
- MISRA C:2012, 11.3 - A cast shall not be performed between a pointer to object type and a pointer to a different object type.