rules.sonarsource.com

C++ static code analysis: Local variables should be initialized immediately

3-4 minutes

Objects should be initialized as soon as they are declared. It will be implicitly the case if they have a default constructor, as this latter will be called, but otherwise the initialization must be explicit. Even when an object has a default constructor, it may be interesting to use another more relevant constructor to directly give the the object its right value.

Such direct initialization increases the readability of the code:

- seeing the initial value of a variable is usually a good indicator of its purpose,
- waiting until we know a good initial value before declaring a variable can lead to a reduced variable scope,
- it makes reasoning on the source code simpler: we remove the burden of having to know if a variable is initialized at a specific point in the code,
- it is a first step that can lead to the possibility of declaring the variable `const`, which further simplifies reasoning,
- it is also a first step toward declaring it `auto`, which could increase readability by shifting the focus away from the exact type.

Please note that the intent of the rule is not to initialize any variable with some semi-random value, but with the value that is meaningful for this variable.

This rule raises an issue when a variable of a non-array type with no constructor is declared without initial value.

The related rule {rule:cpp:S836} detects situations when a variable is actually read before being initialized, while this rule promotes the good practice of systematically initializing the variable.

Noncompliant Code Example

```
double init1();
double init2();
double init3();
double init4();

void f(bool b) {
  int i; // Noncompliant
  string s; // Compliant: default constructor will be called, but we could probably find a better value

  double d1; // Noncompliant
  double d2; // Noncompliant
  if (b) {
    d1 = init1();
    d2 = init2();
  } else {
    d1 = init3();
    d2 = init4();
  }
}
```

Compliant Solution

```
double init1();
double init2();
double init3();
double init4();

std::pair<double, double> init(bool b) {
    return b ? std::make_pair(init1(), init2()) : std::make_pair(init3(), init4());
}
```

```
void f(bool b) {
  int i = 0;
  string s;

  auto [d1, d2] = init(b);
}

// Or:
void f(bool b) {
  auto [d1, d2] = [b](){
    if (b) {
      return std::make_pair(init1(), init2());
    } else {
      return std::make_pair(init3(), init4());
    }
  }();
}
```

Exceptions

Buffers can be left uninitialized as long as they are written into immediately after their declarations.

```
  int buf[10]; // allowed but it should be initialized right after the declaration
```

See

- C++ Core Guidelines ES.20 - Always initialize an object