C++ static code analysis: Member variables should not be "protected"

3 minutes

Protected member variables are similar to global variables; any derived class can modify them. When protected member variables are used, invariants cannot be enforced. Also, protected member variables are hard to maintain since they can be manipulated through multiple classes in different files.

If a class is just a data store without logic, it can safely contain only `public` member variables and no member functions. Otherwise, data members are tightly coupled to the class' logic, and encapsulation must be used. In this case, having only private member variables enforces invariants for data and ensures that logic is defined only in the member functions of the class. Structuring it this way makes It easier to guarantee integrity and easier for maintainers to understand the code.

But when an object provides encapsulation by using `protected` member variables, data integrity logic can be spread through the class and all its derived class, becoming a source of complexity and that will be error-prone for maintainers and extenders.

That’s why `protected` member variables should be changed to `private` and manipulated exclusively through `public` or `protected` member functions of the base class.

This rule raises an issue when a `class` or `struct` contains `protected` member variables.

Noncompliant Code Example

```
class Stat {
public:
  long int getCount() {
    return count;
  }
protected:
  long int count = 0; // Noncompliant; expose a protected member
variable.
                      // By just looking at "Stat" class, it's not possible to be
sure that "count"
                      // is modified properly, we also need to check all
derived classes
};

class EventStat : public Stat {
public:
  void onEvent() {
    if (count < LONG_MAX) {
      count++;
    }
  }
};
```

Compliant Solution

```
class Stat {
public:
  long int getCount() {
    return count;
  }
protected:
  void increment() { // Compliant; expose a protected member
function
    if (count < LONG_MAX) {
      count++;
    }
  }
private:
  long int count = 0; // member variable is private
};

class EventStat : public Stat {
public:
  void onEvent() {
    increment();
  }
};
```

Exceptions

Const member variables and reference member variables are ignored since they don’t break invariants.

See

- MISRA C++:2008, 11-0-1 - Member data in non-POD class types shall be private.
- C++ Core Guidelines C.133 - Avoid protected data