Rules for C++ Programming

The following rules are derived from the *A First Look at C++ Program Analyzers* paper by Scott Meyers and Martin Klaus. The **[Exx]** and **[Mxx]** notes are references to the *Effective C++* and *More Effective C++* books.

Table of Rule Categories

General

1. **[E1]** Use `const` instead of `#define` at global and file scope.
2. **[M2]** Use new-style casts instead of C-style casts.
3. **[M3]** Don't treat a pointer to `Derived[]` as a pointer to `Base[]`.

Use of `new` and `delete`

4. **[E5]** Use the same form for calls to `new` and `delete`. (In general, this calls for dynamic analysis, but static analysis can catch some special cases, e.g., calls to `new` in constructors and `delete` in destructors).
5. **[E6]** When the result of a `new` expression in a constructor is stored in a dumb pointer class member, make sure `delete` is called on that member in the destructor.
6. **[E9]** Avoid hiding the default signature for `operator new` and `operator delete`.

Constructors/Destructors/Assignment

7. Copy-constructor and assignment operator.
    a. **[E11]** Declare a copy constructor for each class declaring a pointer data member.
    b. **[E11]** Declare an assignment operator for each class declaring a pointer data member.
8. **[E12]** Initialize each class data member via the member initialization list.
9. **[E13]** List members in a member initialization list in an order consistent with the order in which they are actually initialized.
10. **[E14]** Make destructors `virtual` in base classes.
11. **[E15]** Have the definition of `operator=` return a reference to `*this`. (Note: this says nothing about declarations).
12. Assignment operators and copy-constructors.
    a. **[E16]** Assign to every local data member inside `operator=`.
    b. **[E1]** Call a base class `operator=` from a derived class `operator=`.
    c. Use the member initialization list to ensure that a base class copy-constructor is called from a derived class copy-constructor.
13. Don't call `virtual` functions in constructors or destructors.

Design

14. **[E19]** Use non-member functions for binary operations like `+-/*` when a class has a converting constructor.
15. **[E20]** Avoid `public` data members.
16. **[E22]** Use pass-by-ref-to-`const` instead of pass-by-value where both are valid and the former is likely to be more efficient.
17. Operators.
    a. **[E23]** Have operators like `+-/*` return an object, not a reference.
    b. **[M6]** And make those return values `const`.
18. **[E25]** Don't overload on a pointer and an `int`.
19. **[M33]** Make non-leaf classes abstract.
20. **[M24]** Avoid gratuitous use of `virtual` inheritance, i.e., make sure there are at least two inheritance paths to each `virtual` base class.

Implementation

21. **[E29/E30]** Don't return pointers/references to internal data structures unless they are pointers/references-to-`const`.
22. **[M26]** Never define a `static` variable inside a non-member `inline` function unless the function is declared `extern`. [Footnote: In July 1996, changes to the nascent standard for ANSI/ISO C++ obviated the need for this rule, at least on paper. However, the need still exists in practice, because many compilers continue to heed the older rules that can lead to duplicated variables in `inline` non-member functions].
23. Avoid use of "`...`" in function parameter lists.

Inheritance

24. **[E37]** Don't redefine an inherited non-`virtual` function.
25. **[E38]** Don't redefine an inherited default parameter value.

Operators

26. **[M5]** Avoid use of user-defined conversion operators (i.e., non-`explicit` single-argument constructors and implicit type conversion operators).
27. **[M7]** Don't overload `&&`, `||`, or `,`.
28. **[M6]** Make sure `operator++` and `operator--` have the correct return type.
29. **[M6]** Use prefix `++` and `--` when the result of the increment or decrement expression is unused.
30. **[M22]** Declare `op=` if you declare binary `op` (e.g., declare `+=` if you declare `+`, declare `-=` if you declare `-`, etc.). One way to satisfy this constraint is by providing a template that yields the appropriate function.

Exceptions

31. **[M11]** Prevent exceptions from leaving destructors.
32. **[M13]** Catch exceptions by reference.

Last updated: Sun. Apr 14 11:55:52 PST 2002