25. Take parameters appropriately by value, (smart) pointer, or reference.

Summary

Parameterize well: Distinguish among input, output, and input/output parameters, and between value and reference parameters. Take them appropriately.

Discussion

Choosing well among values, references, and pointers for parameters is good habit that maximizes both safety and efficiency.

Although efficiency should not be our primary up-front concern (see Item 8), neither should we write needlessly inefficient code when all other things, including clarity, are equal (see Item 9).

Prefer to follow these guidelines for choosing how to take parameters. For input-only parameters:

- Always **const**-qualify all pointers or references to input-only parameters.
- Prefer taking inputs of primitive types (e.g., **char**, **float**) and value objects that are cheap to copy (e.g., **Point**, **complex<float>**) by value.
- Prefer taking inputs of other user-defined types by reference to **const**.
- Consider pass-by-value instead of reference if the function requires a copy of its argument. This is conceptually identical to taking a reference to **const** plus doing a copy, and it can help compiler to better optimize away temporaries.

For output or input/output parameters:

- Prefer passing by (smart) pointer if the argument is optional (so callers can pass null as a "not available" or "don't care" value) or if the function stores a copy of the pointer or otherwise manipulates ownership of the argument.
- Prefer passing by reference if the argument is required and the function won't store a pointer to it or otherwise affect its ownership. This states that the argument is required and makes the caller responsible for providing a valid object.

Don't use C-style varargs (see Item 98).

References

[Alexandrescu03a] • [Cline99] §2.10-11, 14.02-12, 32.08 • [Dewhurst03] §57 • [Koenig97] §4 • [Lakos96] §9.1.11-12 • [McConnell93] §5.7 • [Meyers97] §21-22 • [Stroustrup94] §11.4.4 • [Stroustrup00] §5.5, §11.6, §16.3.4 • [Sutter00] §6, §46