14C Arbitrary Substitution Principle

What is an arbitrary substitution?

- When you have a choice of writing two *different* programs by substituting one variable (or argument) for another…
    - … then the arbitrary substitution principle (ASP) applies.
- You should always be on your guard in such circumstances: in my opinion, you should annotate such code with a comment or annotation such as @asp.
- Failure so to do may result in you getting bitten!

What is a non-arbitrary substitution?

- Suppose you write:
    - *int x = a + b*

- What would happen if you wrote instead:
    - *int x = b + a*

- You'd be writing the exact same program because the + operator *commutes*.

- The alternative substitution here is immaterial so cannot be said to be chosen arbitrarily.

What is an arbitrary substitution?

- Suppose you write:
  - *boolean y = a(x) && b(x)*
  - where *a(x)* is a boolean function and *b(x)* is a different boolean function.
- What would happen if you wrote instead:
  - *boolean y = b(x) && a(x)*
- You'd be writing a *different* program because the && operator *does not commute*. [That's because it is a "short-circuit" operator.]
- To another programmer reading this code, it is not at all obvious why you chose the particular form that you did. I think that programmer is entitled to an explanation, or at least a comment to the effect that you think it doesn't matter.

Does it really matter?

- If *a(x)* and *b(x)* are pure functions (i.e. no side-effects), then it might not matter.
  - But suppose that *a(x)* takes 10 µsec while *b(x)* takes 10 msec, it might matter quite a lot.
- Choosing one option arbitrarily over another should always be considered a code smell.
- Not only that, but such a situation can lead to an insight that improves performance:
  - Example: Weighted Quick Union.