Class 30: Preconditions, Revisited

Held: Tuesday, March 16, 2010

Summary: We consider preconditions and programming techniques for ensuring that preconditions are met.

Related Pages:
- EBoard.
- Lab: Verifying Preconditions.
- Reading: Verifying Preconditions.

Notes:
- Reading for tomorrow: Local Procedure Bindings.
- It’s still Not National Pipe Cleaner Day Day.
- Questions on the exam (now due Friday)?
- EC for Thursday’s CS Extra.
- EC for Friday’s CS Table.

Overview:
- Verifying preconditions.
- The error procedure.
- Husk and Kernel programming.

Preconditions
- Remember: Documentation is a contract, be careful.
- Even if you don’t write documentation (or don’t write careful documentation), you should reflect on your expectations for parameters.
- We call those expectations the preconditions of the procedure.
- In some cases, you may want to explicitly test to ensure that the preconditions are met.
  - Testing lets you provide more useful error messages.
  - Testing prevents dangerous results from being returned.

Reporting Failure
- When a precondition fails, you want to stop computation immediately and report the error.
- In Scheme, you can do so with error or throw
  - The one to use depends on the version of Scheme you are using.
  - Our Scheme interpreter uses error.
Husk and Kernel Programming

- Particularly for recursive procedures, it is inefficient to check preconditions at every recursive call
  - If the preconditions were met for the first call, they should be met for every subsequent call.
- Hence, programmers tend to use what I refer to as “Iowa’s Great Contribution to Programming”: The Husk-and-Kernel approach
  - The husk checks the preconditions and, if all preconditions are met, calls the kernel.
  - The kernel does the real work.
- Corn serves as the metaphor: The husk protects the kernel, and the kernel is the valuable part.
  - And no, Husk-and-Kernel programming was not invented in Iowa.