

## Class 18: Documentation, Preconditions and Postconditions

**Held:** Monday, 25 September 2006

**Summary:** Today we consider an important program design issue: the use of preconditions and postconditions to document our procedures. We also see how to use the `error` procedure to report errors and the husk-and-kernel technique to catch errors.

### Related Pages:

- EBoard.
- Lab: Preconditions and Postconditions.
- Reading: Preconditions and Postconditions.

### Notes:

- I'll leave time at the start of class to discuss questions on exam 1.
- Reminder: No CS extra tomorrow.
- I'll try to do less introductory lecture today.

### Overview:

- The need for documentation.
- Verifying preconditions.
- Husk and Kernel programming.
- Other uses of Husk and Kernel.

## The Need for Documentation

- While you're currently writing small programs, you will eventually write larger programs, including programs with other people.
- Other people will sometimes need to read or use your programs.
  - In this way, programming differs from many other disciplines, in which people don't so directly interact with the work of others (If you read someone else's essay, you'll be able to see the structure of their argument as part of the essay; you also wouldn't reuse an essay directly.)
- Experience shows that after about six months, you've forgotten enough of what you've done that you become "other people".
- Every collection of procedures will need documentation giving some overall structure (and a table of contents).
- Every procedure should be clearly documented as to what it does and what it needs in order to do what it does.
- I tend to speak of "the six P's". Whenever you write a procedure, you should consider (and often

document)

- Any *Parameters* the procedure takes. Typically, you name and give types to the parameters.
- The *Purpose* of the procedure.
- What the procedure *Produces*. Again, you name and give a type to the result.
- *Preconditions* that must hold for the procedure to work.
- *Postconditions* that will hold after the procedure finishes. Typically, these formalize the purpose and produced value.
- Potential *Problems*
- You should also document “tricky” parts of your procedures.
- Note that I recommend that you write the documentation *before* you write the procedure. The six P’s give you a way to think carefully about what you want the procedure to do before you think about how, and that thought can often guide you through both creation and debugging.

## Robustness; Testing Preconditions

- There are different perspectives on preconditions.
- Some people feel that preconditions are a contract. If the client of a procedure (the code that calls the procedure) doesn’t ensure that the preconditions are met, then the procedure is not required to meet the postconditions.
- Others feel that procedures should check their preconditions and respond appropriately.
- Robust programs deal appropriately with incorrect input. Sometimes, the appropriate response is to print an error message and crash.
  - You can use `error` to do just that.
- If your procedure explicitly tests a precondition, I suggest that you mark the precondition as *verified*. If your procedure does not test the precondition, please mark the precondition as *unverified* (even if another procedure will probably generate an error message).

## Husk and Kernel Programming

- One danger of careful precondition checking is that we do the same thing again and again and again ...
- Good programming style dictates that we try not to do redundant work. If we’ve verified that a long list contains only numbers, then we know that any sublist contains only numbers.
- One strategy is to break the procedure up into two parts (that is, two procedures). We call them the husk and the kernel.
  - A *husk* does the checking and then calls the kernel.
  - The kernel does the real work.
- Why these names?
  - The husk protects the kernel
  - The kernel is the good part
- Some programmers prefer *proc-safe* and *proc-unsafe* as procedure names that warn the client about the status of errors.

## Lab

- Do the lab.
  - Be prepared to reflect.
- 

Copyright © 2006 Samuel A. Rebelsky. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.