CSC161 2011S *Imperative Problem Solving*

# Class 27: Program Correctness and `assert`

**Held:** Wednesday, 9 March 2011

**Summary:** We consider issues of program correctness and ways in which C's `assert` macro can help.

**Related Pages:**

- EBoard.
- Lab: C's `assert` Macro.
- Reading: Design by Contract .
- Due: Assignment 6.

**Notes:**

- While I appreciate the 2/3 of you who submitted questions on the reading, I wish that all of you had. I feel like a broken record.
- Reading for Thursday night: K&R 4.11.
- I'll be checking email off and on today, so feel free to email me questions on your code. (And feel free to say "Boy Sam, that answer was too brief.")
- Your EBoard will be a bit brief today. Sorry.

**Overview:**

- Programming by Contract.
- `assert`.
- Lab.

# Programming by Contract

Because we focus on preconditions and postconditions in 151, you may not have found much new in the reading.

Still, it is useful to keep a number of things in mind.

- First, it is valuable to understand the *preconditions* of any function your write: What must hold in order for the function to work.
- Second, it is equally valuable to understand the *postconditions* of any function you write: What you can absolutely guarantee about the results of your function (provided the preconditions are met).
  - If you can't guarantee much, your function is not very useful.

One then has the question of how to deal with preconditions and postconditions.

- In the extreme, a formal analysis tool can step through the program, checking preconditions before each statement and updating its knowledge of the state of the program based on the postconditions of that statement.a
  - Such programs are typically hard to use and require that you spend a lot of time formally specifiying preconditions and postconditions.
- A procedure/statement can check its preconditions before it starts to execute.
- A procedure/statement can check its postconditions before it returns.
- Why would you choose one over another?

What should you do if a precondition is not met?

- Nothing special: If the precondition is not met, it's the caller's fault, and the caller has to accept whatever we decide to return.
- Return a special value to indicate "Whoops! Preconditions not met."
- Set a global variable to indicate "Whoops! Precodnitions not met."
- In object-oriented languages, we can invoke an alternate control flow on the program, telling the caller "You screwed up. Now deal with it."
- You can shut down the whole program.

Additional notes

- My assessment is not quite fair. Often, preconditions are difficult, expensive, or even impossible for the caller to check.
  - "The file system is stable."
  - "The file system contains room for this file."
  - "This operation will not overflow."
- We can treat almost any statement as having preconditions and postconditions.
  - For example, x = y / z;

# `assert`

- C provides the `assert` statement to help you deal with many of these issues.
- `assert (`*`expression`*`)` checks if *expression* fails. If so, `assert` prints an error message and terminates the program.
- `assert` helps us catch errors early in a program. (One great thing about C is that many errors pass through, screw something else up, and it's really hard to find out the first place that something went wrong.)
- Because assertion checking can be expensive, you can also turn it off.

# Lab

- Do the lab.
- Be prepared to reflect.