CSC161 2011S *Imperative Problem Solving*

Class 23: Functions and Parameters

Held: Wednesday, 2 March 2011

Summary: We consider some issues with functions in C.

Related Pages:

- EBoard.
- Lab: Basic Functions in C.
- Reading: K&R 4.1-4.3.
- Due: Assignment 5.

Notes:

- I'll reserve time at the start of class for questions on Assignment 5.
- When you're in a group and email me a question, it's nice if you carbon-copy your colleagues so that I can just hit "reply-all".
- Assignment 6 should be ready some time on Thursday. I'll email you when it's ready.
- I've posted some answers to your questions on the control flow reading.
- Questions on reading missing from: DF.
- Next reading: K&R 4.3-4.6.
- EC: Thursday's CS Extra: FPGAs.
- EC: Friday's CS Table: GPU programming in CUDA, revisited.

Overview:

- Basics of functions in C.
- Compiling.
- RPN and Stacks.
- Lab.

Functions in C

- The form of a procedure definition

```
TYPE
NAME (PARAMETERS)
{
  BODY;
}
```

- Problem: How does the C compiler know if you're using a procedure in the "correct" way?
- Solution: Declarations, which appear at the "top" of your program
- Form of declarations:

```
TYPE NAME (PARAMETERS);
```

The Compilation Process

- The computer spends some effort going from human-readable programs (e.g., written in C) to computer-readable programs.
- For the C compiler, the process is as follows:
- First, *preprocess* the code (`cc -E`)
  - Load included files
  - Strip out comments
  - Resolve #defines
- Next, *translate* the code into assembly language (`cc -S`)
  - Assembly is a semi-human-readable version of the underlying computer code.
- Next, *assemble* the assembly code into object code (`cc -c`)
  - Object is computer-readable code that needs a few tweaks to be executed.
- Finally, *link* the various object files together into an executable.

Stacks and Reverse-Polish Notation

- RPN: An unambiguous way of computing that doesn't require parentheses.
  - Write the operator *after* the operands.
- Easy to implement
  - When you see a value, add it to the end of a list of things to process.
  - When you see an operator, grab the last two things on the list apply the operator, and add the result back to the end of the list.
- Stack: An ADT that supports those two operations (plus a few others).
  - Basically, stacks are collections of values that implement that "last in, first out" principle.
  - Add to the stack is called `push`
  - Remove from the stack is called `pop`

Lab

- Do The lab.
- Be prepared to reflect.
