CSC161 2011S *Imperative Problem Solving*

Class 24: Building Libraries

Held: Friday, 4 March 2011

Summary: We consider why and how to put C code in separate files.

Related Pages:

- EBoard.
- Lab: Simple Libraries in C.
- Reading: K&R 4.3-4.6.

Notes:

- I did not answer all of last night's questions. Let me know if things are still unclear after the presentation and lab.
- I will be unavailable for much of next week. Please get me questions on the homework before Tuesday's class.
- Assignment 6 is now ready.
- Reading for Sunday night: Unit Testing.
- No CS Extras or CS Table next week!

Overview:

- RPN and Stacks.
- Why We Need Code Libraries.
- Simple Code Libraries in C.
- Lab.

Reverse Polish Notation and Stacks

- We'll run a bit through the K&R example (or at least my variation of it).

Libraries

- We often write functions that we can reuse in other situations and other contexts.
- We could copy those files from program to program.
- But that makes it hard to propagate updates.
- So different languages provide different ways to build libraries or other collections of useful stuff.

- In C, you can break a program into separate files.
- You can compile those files together.
- You can compile those files separately and then link them together.

- There are also ways to build libraries in C.

- How does one compile a file separately?
  - One builds a `.o` file.
  - `make` knows about `.o` files,
- How does one join all the .o files together?
  - With `cc`.
  - Or with an appropriate make command.

- How does one file know about the code from another file?
  - It can rely on the standard C perspective of “If you don’t tell me anything about a function, I’ll assume that it’s available and returns an `int`.”
  - You can explicitly indicate that the function is available.
- Often, we group the information about a library file into a *header* which we can then include in any file that uses it.
- Headers have a suffix of `.h`.

Lab

- Do The lab.
- Be prepared to reflect.
