Slide 1

Administrivia

- Reminder: Homework 5 due today. If you're having trouble, let me know, and if my usual office hours don't work for you maybe we can arrange something . . .
- Homework 6 to be on the Web soon; due either end of next week or during finals. Some type of linked data structure.

Slide 2

Minute Essay From Last Lecture

- Several people asked about uses for C, or about its origins and why it has been influential. A bit about that next time.
- Others asked about C versus C++. In a few words, C++ is object-oriented and has an extensive standard library (like Scala and Java) but (at least traditionally) requires coping a bit more with low-level details.
- Others asked for more about pointers. I say if you do Homeworks 5 and 6 . . .

Separate Compilation — Overview

- C (like many languages) lets you split large programs into multiple source-code files. Typical to put function and other declarations in files ending `.h`, function definition in files ending `.c`. Compilation process can be separated into "compile" (convert source to object code) and "link" (combine object and library code to make executable) steps.
- UNIX utility `make` can help manage compilation process. Can also be useful as a convenient way to always compile with preferred options. Idea behind `make` — have computer figure out what needs to be recompiled and issue right commands to recompile it. (Similar to what many/some IDEs do behind the scenes.)

Makefiles

- First step in using `make` is to set up "makefile" describing how files that make up your program (source, object, executable, etc.) depend on each other and how to update the ones that are generated from others. Normally call this file `Makefile` or `makefile`.

  Simple example on sample programs page.
- When you type `make`, `make` figures out (based on files' timestamps) which files need to be recreated and how to recreate them.

Slide 5

Predefined Implicit Rules

- `make` already knows how to "make" some things — e.g., `foo` or `foo.o` from `foo.c`.
- In applying these rules, it makes use of some variables, which you can override.
- A simple but useful makefile might just contain:

  ```
  CFLAGS = -Wall -pedantic -O -std=c99
  ```

  and then you type `make foo` to compile `foo.c` to get executable `foo`.
- Or you could use

  ```
  OPT = -O
  CFLAGS = -Wall -pedantic -std=c99 $(OPT)
  ```

  and then optionally override the `-O` by saying, e.g., `make OPT=-g foo`.

Slide 6

Example — Sorted Singly-Linked List, Continued

- Last time we started writing code for a sorted list, as an example of implementing an ADT in C. Continue . . .
- (Code on sample programs page. Somewhat elaborate but I wanted to try to do a nice job of providing a test framework.)

Slide 7

Minute Essay

- Anything interesting to tell me about Homework 5?