Slide 1

Administrivia

- Reminder: Homework 5 due next week.

Slide 2

Minute Essay From Last Lecture

- Advantages of automatic garbage collection? easier, more convenient, more efficient (I'm skeptical about that but maybe).
- Disadvantages of automatic garbage collection? less control, could be delete something you want (well, no).
- (Also review my answer(s) from slides.)

Slide 3

Homework 5 Hints — Recommended Approach

- First read whole file into (dynamically allocated) array of `char`. (How do you find out how big it should be?) `fgetc` will work to read a character at a time. Or you can use `fread` to read the whole file.
- Build an array of pointers-to-lines (i.e., an array of `char*`). (How do you find out how big it should be?) Could do this with two `for` loops.
- Use `qsort` to actually sort *the array of pointers*.
- Print the (sorted) result using the array of pointers.

Slide 4

A Little More About `gcc`

- Many, many compiler options for `gcc`. One of the most useful is `-Wall`.
- To automate using them every time, you can use the UNIX utility `make` ...

Slide 5

A Little About `make`

- Motivation: Most programming languages allow you to compile programs in pieces ("separate compilation"). This makes sense when working on a large program — when you change something, just recompile parts that are affected.
- Idea behind `make` — have computer figure out what needs to be recompiled and issue right commands to recompile it.

Slide 6

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 7

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
```

- 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 8

Minute Essay

- None really — sign in — unless you have questions?