Slide 1

Administrivia

- Homework 4 on Web; due a week after spring break.

Slide 2

Review — Strings and Pointers

- Strings in C are null-terminated arrays of `char`s.
- Pointers are in some ways a less abstract and less safe version of Java references. They're also in some respects interchangeable with arrays.

Slide 3

Pass By Reference (Sort Of)

- A significant potential limitation on functions is that a function can only return a single value. Pointers provide a way to get around this restriction: By passing a pointer to something, rather than the thing itself, we can in effect have a function return multiple things.
- To make this work, typically you declare the function's parameters as pointers, and pass addresses of variables rather than variables.
- The "sort of" of the title means that this isn't true pass by reference, as it exists in some other languages such as C++, but it can be used to more or less get the same effect. Notice also that Java can't do this, though again there are mechanisms that can more or less get the same effect. (What?)

Slide 4

Arrays of Text Strings and Command-Line Arguments

- If you can have arrays of `int` and `char` and so forth — can you have arrays of text strings? Sure! They look like two-dimensional arrays of `char`, or like arrays of `char *`.
- Further, this is how C programs get input "from the command line" (e.g., when you write `gcc myprogram.c`, `gcc` somehow gets `myprogram.c`, right?):

  `main` can also be defined as

  ```
  int main(int argc, char * argv[]) { .... }
  ```

  where `argc` is the number of arguments, plus one, and `argv` is an array of strings containing the arguments. Example — let's write a simple "echo" program.

Slide 5

Sidebar: I/O in C — Some Very Basic Functions

- `getchar` gets one character and returns it as an `int`. The special value `EOF` indicates end of input. ("End of input"? control-D from terminal, more in next sidebar.)
- `putchar` writes out one character.
- Use this to write a very simple program that simply copies its input to its output . . .

Slide 6

Sidebar: Input/Output Redirection in UNIX/Linux

- In programming classes I talk about "reading from standard input" rather than "reading from the keyboard", and "writing to standard output" (or "writing to standard error") rather than "writing to the screen".

  (In Java terms — `System.in`, `System.out`, and `System.err`. C has similar concepts but calls them `stdin`, `stdout`, and `stderr`.)
- What's the difference?

Slide 7

I/O Redirection, Continued

- `stdin` (standard input) can come from keyboard, file, or from another program or shell script.
- `stdout` and `stderr` (standard output, error) can go to terminal or file (overwrite or append), separately or together.

Slide 8

I/O Redirection, Continued

- For example — to redirect output of `ls` to `ls.out`, type

  `ls >ls.out`

  (Overwrites `ls.out`. To append, replace $>$ with $>>$.)

  To also redirect any error messages, append `2>&1`.
- To redirect input, use $<$`infile`.

Slide 9

I/O in C — Basics

- We talked already about single-character I/O (`getchar` and `putchar`).
- You already know about a function to write output to "standard output", `printf`. Many options, allowing a lot of control over what's printed.
- How about input? Counterpart of `printf` is `scanf` (skim man page). Simple to use, though error detection is somewhat crude, and reading text strings can be hazardous.
- One way to work with files is I/O redirection. Is there something more general? Yes . . . (next time).

Slide 10

Minute Essay

- None — sign in.