Slide 1

Administrivia

- Reminder: Quiz 6 Thursday. Likely topics are pointers and strings.
- Homework 7 on Web; due next Tuesday.

Slide 2

Pointers — Review

- Most/many programming languages provide a way to “point to” something in memory (such as a variable). In C, these are called pointers, and you declare them by putting a `*` after the type of the thing pointed to. (Notice that this means you can have pointers to pointers!)
- You can get the address of a variable with &. You “dereference” a pointer (access what it points to) with `*`.
- One important use of pointers is to allow returning more than one thing from a function, as `scanf` can.

Slide 3

Pointers, Arrays, and Pointer Arithmetic in C

- C treats pointers and arrays as interchangeable in most respects. (This is why it works that many functions whose parameters are supposed to be strings — arrays of characters — declare them as pointers. `fopen` is an example.)
- C also permits doing some arithmetic operations on pointers (addition and subtraction). Adding n to a pointer that points to *type* advances it n times the size of *type*.

  Example: If `a` is an array of `int`s, `a[2]` and `*(a+2)` are equivalent. (This means we could write loops over arrays using pointers. Once upon a time that was sometimes more efficient. With current compilers, probably not so, so use whatever is most readable.)

Slide 4

Text Strings in C, Revisited

- As mentioned briefly last time: C represents text strings as arrays of characters, with the end of the string indicated by a special “null” character.
- There are many library functions useful for working with strings. But as practice working with arrays and pointers and dynamic memory, we could write some of our own . . .

Slide 5

Text Strings in C, A Little More

- A significant problem in working with strings is that there's no natural maximum size, so you have to decide how big to make the array of characters you will use to hold one — and then be sure you don't try to put in too many characters.
- Some library functions let you say how big the array is; some don't. *Always* be as careful as you can when working with strings; trying to store a string in an array not big enough is a source of "buffer overflows", which can lead to program crashes and more subtle problems, including security risks.
- Example — revisit the "change case" example, but prompt for filenames.

Slide 6

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.

Minute Essay

- About the textbook — what did you like? what did you not like?