Slide 1

Administrivia

- (None.)

Slide 2

Files in C — Recap

- We talked last time about one way to deal with files in C — as “streams”.
- To do this — use `fopen` to associate a stream (`FILE *`) in your program with a filename, `fscanf` and `fprintf` to read and write (very much like `scanf` and `printf`), and `fclose` when done.

Slide 3

Examples

- Example — read integers from `numbers.txt`, write even ones to `evens.txt`, odd ones to `odds.txt`.
- Example — file-to-file copy, but turning uppercase into lowercase and vice versa. (This will also be practice using the character-oriented library functions described in the textbook.)
- (Other examples as time permits?)

Slide 4

Why Arrays?

- Suppose you wanted to write a program to count how many times each letter occurs in a text file. What would you do? Is there an obvious way to solve this with what we've discussed so far?

Slide 5

Why Arrays?, Continued

- You could have a variable for how many A's, one for how many B's, etc., and a huge `switch` construct. But how ugly . . .
- What seems to be needed is something similar to subscripted variables in math — an *array*.

Slide 6

Arrays

- Previously we've talked about how to reserve space for a single number/character and give it a name.
- Arrays extend that by allowing you to reserve space for many numbers/characters and give a common name to all. You can then reference an individual element via its *index* (similar to subscripts in math).

Slide 7

Arrays in C

- Declaring an array — give its type, name, and how many elements. Examples:

```
int nums[10];
double stuff[N];
```

(The second example assumes N is declared and given a value previously. In old C, it had to be a constant. In newer C, it can be a variable.)

- Referencing an array element — give the array name and an index (ranging from 0 to array size minus 1). Index can be a constant or a variable. Then use as you would any other variable. Examples:

```
nums[0] = 20;
printf("%d\n", nums[0]);
```

(Notice that the second example passes an array element to a function. AOK!)

Slide 8

Arrays in C, Continued

- We said if you declare an array to be of size n you can reference elements with indices 0 through $n - 1$. What happens if you reference element -1? n? $2n$?
- Well, the compiler won't complain. At runtime, the computer will happily compute a memory address based on the starting point of the array and the index. If the index is "in range", all is well. If it's not (i.e., it's "out of bounds) . . .

Slide 9

Arrays in C, Continued

- (What happens if you try to access an array with an index that's out of bounds?)
- "Results are unpredictable." Maybe it's outside the memory your program can access, in which case you probably get the infamous "Segmentation fault" error message.

  Almost worse is if it's not — then what's at the computed memory address might be some other variable in your program, which will then be accessed/changed. This is the essence of the *buffer overflows* you hear mentioned in connection with security problems.
- What to do? *Be careful.* (Probably worth noting here that some other languages, Java for example, protect you from such errors.)

Slide 10

Minute Essay

- One of our example programs copies a text file, changing lowercase letters to uppercase and vice versa. What would you have to do to this program to allow it to copy a non-text file without changing its contents?

Slide 11

Minute Essay Answer

- Not much — just open the input and output files with mode parameters `"rb"` and `"wb"` respectively, and copy characters without changing case.