Slide 1

Administrivia

- Reminder: Homework 5 due today. Will be considered on time if sent before midnight.
- Office hours today — 2pm to 2:50pm, 3:30pm to 5pm. (How many are still having trouble with the second problem?)
- Next homework to be on Web later today / early tomorrow. Probably due next Thursday.
- Course next semester (shameless self-promotion?):
  - CSCI 3294 (Unix Power Tools) — similar to course two years ago, syllabus and notes on the Web, linked from my home page under "Old course materials".

Slide 2

Minute Essay From Last Lecture

- (I asked about what you thought was the biggest limiting factor in your ability to write code, what you wanted to be able to do. Many thoughtful answers!)
- "Hard to remember everything" — yes, but practice will help. Same thing applies to being able to decipher error messages.
- Cut/copy/paste in `vim` — `:help visual.txt` explains the method you'll probably like best, and/or the tutorial (`vimtutor` from the command line).
- Programs that do something more exciting(?) than simple text input/output — well, you can't really "get there from here" in standard C. Other programming languages (e.g., Java) make it easier.

Why Arrays?

Slide 3

- 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?

Why Arrays?, Continued

Slide 4

- 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 5

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 6

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 7

Arrays in C, Continued

- Declaring that a function parameter is an array — put brackets after parameter, usually don't specify array size as part of array parameter. Example:

  ```
  double array_sum(int size, double nums[]);
  ```

  (The newer C99 standard supports a slightly different syntax. More about this another time.)

- Passing an array to a function — just give its name. Example:

  ```
  double stuff[100];
  printf("%f\n", array_sum(100, stuff));
  ```

Slide 8

Arrays in C, Continued

- Using an array parameter within the function — as in previous examples.
- A difference between array parameters and other parameters — array elements can be changed.
- If your function isn't supposed to change the array, declare the parameter `const`, e.g.

  ```
  double array_sum(int size, const double
  nums[]);
  ```

  (Helps people using your function understand its effects, allows compiler to enforce that no changes are made.)

Arrays in C, Continued

Slide 9

- 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? $2n$?

Arrays in C, Continued

Slide 10

- 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 11

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 12

Example(s)

- (Next time — instead look at second Homework 5 problem.)

Minute Essay

- None — sign in.