Slide 1

Administrivia

- Reminder: Homework 4 due Friday.
- Quiz 3 next Wednesday. Midterm the following week(!).
- Quiz 2 solution online.

Slide 2

Homework 3 Essays

- Not much really stood out.
- Several people found this assignment more difficult than the previous one. No surprise! (But most people did well — good!)
- A few people found the problems interesting to solve. I try!

Slide 3

Repetition Via Loops

- Recursion provides one way to repeat something. Often not efficient (every call to a function requires space for local variables, and at some point you can run out of room), nor is it always convenient (writing a function every time you want to repeat something).
- Hence C, like most procedural languages, offers constructs called *loops*. All have four basic elements (sometimes implicit).

Slide 4

Loop Elements

- Initializer — something that sets initial values for variables involved in the repetition (iteration).
- Condition — something that determines whether repetition continues. Can be tested at the start of each iteration (*pre-test* loop) or at the end (*post-test* loop).
- Body — the code to repeat.
- Iterator — something that moves on to the next iteration.

Slide 5

`while` Loops

- Probably the simplest kind of loop. You decide where to put initializer and iterator. Test happens at start of each iteration.
- Example — print numbers from 1 to 10:

```
int n = 1;              /* initializer */
while (n <= 10) {       /* condition */
    printf("%d\n", n);  /* body */
    n = n + 1;          /* iterator */
}
```

- Various short ways to write `n = n + 1`:

```
n += 1;
n++;
++n;
```

What do you think happens if we leave out this line?

Slide 6

`for` Loops

- Probably the most common type of loop. Particularly useful for anything involving counting, but can be more general. Syntax has explicit places for initializer, condition, iterator (so it's less likely you'll forget one of them).
- Example — print numbers from 1 to 10:

```
int n;
for (n = 1; n <= 10; ++n) {
    printf("%d\n", n);
}
```

- Initializer happens once (at start); condition is evaluated at the start of each iteration; iterator is executed at the end of each iteration.

Slide 7

`do while` Loops

- Very similar to `while` loop, except that test happens at end of each iteration.
- Example — print numbers from 1 to 10:

```
int n = 1;                          /* initializer */
do {
    printf("%d\n", n);              /* body */
    n = n + 1;                      /* iterator */
} while (n <= 10);                  /* condition */
```

Slide 8

Loops — Simple Examples

- We could do a loop version of the program to sum integers from stdin, as an example of using a `while` loop.
- We could do loop versions of the factorial and Fibonacci programs, as examples of using `for` loops.

Slide 9

Minute Essay

- Have you seen loops in another context? (Matlab, etc.)?