Slide 1

## Administrivia

- Reminder: Homework 2 was due last week, so if you haven't turned anything in (I think only one person is in this category), please do.
- Next homework will be on the Web later today or early tomorrow. To be due next week.

Slide 2

## Loops in C — Recap

- C has several constructs for repeating execution of a statement or block of statements — `while`, `do while`, and `for` loops. The first two will likely be familiar to Python and Scala programmers; the third, not so much.
- What C does not have, and Python and Scala do, is nice constructs for iterating through collections — in keeping with its being lower-level, maybe.

Slide 3

## Arrays in C — Recap

- Again in contrast to higher-level languages such as Python and Scala, C has only one construct for representing collections of similar data, namely the array.
- In some ways C's arrays are fairly similar to arrays in Python and Scala — basic idea of a collection of elements of the same type, fixed size, indexed starting at 0.
- A key difference is that with C's arrays the underlying implementation shows through more clearly — what you get is a sequence in memory of storage cells, all of the same size, with little in the way of safety checks that would keep you within the allowed bounds.

Slide 4

## Functions in C

- Functions in C are conceptually much like functions in other procedural programming languages. (Functions in object-oriented languages are similar but have some extra capabilities.)

  I.e., a function has a *name*, *parameters*, a *return type*, and a *body* (some code).
- One difference between C and higher-level languages: You aren't supposed to use a function before you tell the compiler about it, either by giving its full *definition* or by giving a *declaration* that specifies its name, parameters, and return type. The function body can be later in the same file or in some other file.

Slide 5

## Parameter Passing in C

- In C, all function parameters are passed “by value” — which means that the value provided by the caller is copied to a local storage area in the called function. The called function can change its copy, but changes aren’t passed back to the caller.
- An apparent exception is arrays — no copying is done, and if you pass an array to a function the function can change its contents (as we did in the sort program). Why “apparent exception”? because really what’s being passed to the function is not the array but a pointer! so the copying produces a second pointer to the same actual data. (More about pointers soon.)

Slide 6

## Functions, Local Variables, and Recursion

- Functions in C can contain local variables. Every time you call the function, you get a fresh copy of the variables.
- So yes, recursive functions work the way you (probably?) think they should.

Slide 7

## Library Functions in C

- C does include a library of standard functions, though it's not as extensive as that of some languages.
- At least on UNIX-like systems, for each library function there should be a `man` page that tells you about it, including information about `#include` files you need and link-time options (e.g., `-lm` for `sqrt`). For now, be advised that asterisks in types denote pointers, which we will talk about soon.

Slide 8

## Functions in C — Examples

- Examples as time permits — some simple recursive functions, and a partial array-sorting program.

## Minute Essay

- What was interesting about Homework 2? What was difficult?