Slide 1

Administrivia

- Homework 3 on Web, due next Monday.

Slide 2

Arrays in C, Briefly

- Syntax for creating arrays is somewhat different from Java's — no explicit `new`, but instead something like

  ```
  int x[10];
  ```

  to reserve space for 10 `int`s. In old-style C, sizes must be constants known at compile time. In new-style C, "variable-length arrays" (VLAs) are permitted as well.
- Syntax for array access is the same as Java, but there's no `length` variable, and no checks are made to ensure that the index is legit (between 0 and array size minus one). This can make for interesting bugs . . .
- Syntax for passing arrays as parameters to functions is somewhat like Java's, except brackets typically go after the parameter name, and and arrays and pointers (more soon) can be used more or less interchangeably.

Slide 3

Arrays in C — Example

- Let's write a sort program . . .
- (Where to get input? let's just generate random values, using library function `rand()`.)

Slide 4

Strings in C

- Java has a `String` class with many useful features and methods. In C that's not possible . . .
- Instead, in C, strings are arrays of `char`s, with the convention that the actual text of interest is followed by a null character (8-bit zero, represented in code as `'\0'`.
- You can operate on individual characters however you see fit; there are also standard library functions for some common operations (e.g., `strcmp` to compare two strings — similar to `compareTo` in Java).
- A significant source of potential trouble — most functions assume that strings are properly terminated, and (worse) many have no safety check to make sure you don't overflow a destination array.

Slide 5

Pointers in C

- Pointers in C are similar to, but not identical to, references in Java — with the key differences having to do with safety features and level of abstraction. (No surprise!)
- In C, pointers are just memory addresses — but they are declared to point to variables (or data) of a particular type. Example:

```
int * pointer_to_int;
double * pointer_to_double;
```

Slide 6

Pointers in C — Operators

- `&` gets the address of something in memory. So for example you could write

```
int x;
int * x_ptr = &x;
```

- `*` "dereferences" a pointer. So for example you could change `x` above by writing

```
*x_ptr = 10;
```

- You can also perform arithmetic on pointers (e.g., `++x_ptr`) — something not allowed in Java, and another example of the languages' different design goals.

Slide 7

Pointers Versus Arrays

- In C, pointers and arrays are in some sense(s) equivalent — not identical, but in many contexts interchangeable.
- This is reflected in the `man` pages for many functions (e.g., `printf`). It also means that when you pass an array to a function, what you're actually passing is a pointer — so the array is not copied.

Slide 8

Minute Essay

- None — sign in.