Slide 1

Administrivia

- Reminder: Homework 4 due today.
- Homework 5 on the Web. Due next week.

  One problem asks you to fill in the body of a function to sort an array, which should be straightforward if you remember learning about sorting in CS1 (and if you don't, ask me about supplemental reading).

  The other problem involves strings and may be — if not algorithmically challenging, at least somewhat interesting?

Slide 2

Minor C Programming Tip

- In some of the sample programs, `main()` returns `EXIT_SUCCESS` or `EXIT_FAILURE`. These are constants defined in `stdlib.h` and somewhat more guaranteed to be more portable than 0 or 1 (not to mention that they make it clearer what's being done?).

Slide 3

Pointers — Review

- Pointers are, roughly speaking, memory addresses. Useful in many contexts. If they don't make sense to you yet, practice using them may help?

Slide 4

Sidebar: "Undefined Behavior" in C

- You may have noticed that if you try to input a really large value with `scanf` you don't get either the right value or any kind of error.
- You might also notice that strange things happen when you try to compute a fairly large number using an `int`. (This is easy to do with our factorial program.)
- Both examples of what C calls "undefined behavior". Means that the language doesn't say what's supposed to happen. Might be different depending on compiler and options!
- Out-of-bounds array accesses? Undefined behavior.
- Statements that modify the same variable twice (e.g., `i = i++)`)? Undefined behavior.

Slide 5

“Undefined Behavior” in C, Continued

- Since language makes no guarantees about what will happen, careful programmers do their best to avoid undefined behavior. (And non-careful programmers should probably avoid C.)
- Out-of-bounds array access should be easy if tedious to avoid.
- Arithmetic overflow may be harder to avoid, though with some extra code it’s (usually?) possible.

Slide 6

Characters and Strings in C

- C has a data type `char`, used for much the same purposes as characters in other language, *but* with a smaller minimum range (enough to represent 7-bit ASCII but not Unicode).
- C “strings” are arrays of characters, ending with “null character” (`\0`).

  Can be worked with as arrays or using pointers.

  There are standard library functions for doing (some) things with characters and strings.
- (Example — several ways to write a “string length” function.)

Characters and Strings in C — Library Functions

Slide 7

- C's standard library is pretty limited but does contain some useful functions for operating on character/string data.
- Some useful ones are `isdigit` etc. for characters and `strlen`, `strcmp`, `strchr`, and `strstr` for strings.

  (Notice that you need `strcmp` to compare strings for equality; == compares *pointers* so generally will not do what you want.)

Strings in C — Pitfalls

Slide 8

- Most functions assume that strings are properly terminated. (What do you think happens if they're not?)
- Many functions that store into a string have no way to check that it's big enough.

  So getting text input from standard input *safely* is surprisingly difficult! `scanf` can be made to check, but not (in my opinion) nicely, and it stops on whitespace anyway. `gets` gets a full line, but note what `gcc` says when you use it, and what's in its `man` page. `fgets` is maybe better but has its limitations too.

Minute Essay

- In C, strings are just arrays of characters, terminated with a special character. Other languages likely represent them as an array of characters plus a length field. What are the advantages and disadvantages of doing it C's way?

Minute Essay Answer

- Some things that occur to me:
  - Minus: Using a special character as a delimiter means you can't use that character within a string. That seems like an artificial restriction.
  - Minus: You need one extra character for the delimiter, and remembering to reserve space for it and ensure that it's present for all strings is a bit error-prone.
  - Minus: Determining string length takes longer.
  - Plus: You don't have to decide on a size for the length field, and short strings probably take less space if you don't need this field.
  - Plus: It's possible to work with substrings in ways that are more cumbersome if each string needs a length field.