Slide 1

Administrivia

- Reminder: Homework 5 due today.
- Remaining quizzes . . . — I propose that we have five not six (I would still drop the lowest one) and that we schedule the first one for next Monday, the other the following week.

Slide 2

Text Data — Review

- We talked about how numbers (integer and real) are represented using ones and zeros. How about text? Have to represent it also in terms of ones and zeros. Usual way is to come up with some way of “encoding” single characters, then some way of assembling them into “strings”.
- So C, like most programming languages, has a data type for single characters, namely `char`. C’s is somewhat more limited than that of many more-recent languages — can represent ASCII values (Roman alphabet, digits, some punctuation, a few “special characters” such as newline). Recent versions also support “wide characters” (type `wchar_t`).
- C then represents “strings” as arrays of `char`.

Slide 3

Text Data — Single Characters

- `char` is considered an integer type and can be worked with as such. Notice that while these days ASCII is by far the most common encoding, standard doesn't require that and there are other possibilities.
- Many library functions for working with single characters (e.g., `isalpha`).
- Can read in / print single characters with `scanf` or `printf` using `%c`. Or can use `getchar`, `putchar`. Notice that `getchar` returns an `int`. Why? so it can return special value `EOF` when no more input.

Slide 4

Text Data — "Strings"

- Types for representing numbers (integer and floating-point) typically of fixed size. That doesn't work well for strings. What to do instead?
- C, as usual like most programming languages, provides support for varying-length "strings" of characters — but being C what is provided is somewhat primitive:
- A C "string" is a sequence of characters ending with special "null character" (`\0`), stored in an array of `char`.

Slide 5

Text Data — Strings, Continued

- C library contains functions for working with strings, but be aware that they must be used with caution — many do not check that requested operation does not overflow output array, and most assume that operands are properly “null-terminated”.
- Can read in / print strings with `scanf` or `printf` using `%s`. Notice however that `scanf` not only disallows embedded spaces but also can overflow array unless care is taken. Notice — with this type no need for & before variable name.
- Can also use `puts` to write a string, `fgets` to read “a line” of text. Latter is reasonably safe but puts end-of-line in array.

Slide 6

Command-Line Arguments

- Up to now any input from user has come from “standard input” (keyboard, or file via I/O redirection).
- Another option is “command-line arguments” — e.g.,

  ```
  a.out abcd 1234
  ```

  calls program `a.out` with two of them (`abcd` and `1234`).

  (Compare to calling, e.g., `gcc`.)
- Because getting text input in C is (in my opinion) so full of pitfalls, I think this is a good way to get some text information, such as filenames.

Slide 7

Command-Line Arguments, Continued

- Most programming languages provide some way to access command-line arguments, distinct from reading from standard input.
- In C, access requires a different declaration for `main`:

```
int main(int argc, char * argv[]) { .... }
```

where `argc` is the number of arguments, plus one, and `argv` is an array of strings containing the arguments. We could write a simple program to echo these arguments . . .

Slide 8

Minute Essay

- None — quiz.