

Administrivia

- Homework 6 on the Web. Due next Wednesday. One problem involves something I will talk about Wednesday, but you should be able to start both based on today's class.

Slide 1

Homework 5 Essays

- Many people found this assignment more difficult or time-consuming than earlier ones. No surprise! The good news is that many also say they learned from it.

Slide 2

Files and C

Slide 3

- Why files? You probably already know: Things stored in memory vanish when you turn the computer off; to preserve them, usually save them as *files*.
- We know one way for a C program to get its input from a file, or write its output to a file — I/O (input/output) redirection. But this makes it difficult to get input from more than one source, or save output in more than one place.
- So C (like many other programming languages) provides ways to work more generally with files.

Streams

Slide 4

- C's notion of file I/O is based on the notion of a *stream* — a sequence of characters/bytes. Streams can be *text* (characters arranged into lines separated by something platform-dependent) or *binary* (any kind of bytes). UNIX doesn't make a distinction, but other operating systems do.
- An input stream is a sequence of characters/bytes coming into your program (think of characters being typed at the console).
- An output stream is a sequence of characters/bytes produced by your program (think of characters being printed to the screen, including special characters such as the one for going to the next line).

Streams in C

Slide 5

- In C, streams are represented by the type `FILE *`. `FILE` is something defined in `stdio.h`. (As usual, the `*` means pointer — discussed a bit already, more later.)
(`FILE` is an example of an “opaque data type” — something defined in a library, the details of which might vary among implementations and which should not matter to users.)
- A few streams are predefined — `stdin` for standard input, `stdout` for standard output, `stderr` for standard error (also output, but distinct from `stdout` so you can separate normal output from error messages if you want to).
- To create other streams — next slide.

Creating Streams in C

Slide 6

- To create a stream connected with a file — `fopen`.
- Parameters, from its `man` page:
 - First parameter is the name of the file.
 - Second parameter is how we want to access the file – read or write, overwrite or append — plus a `b` for binary files.
 - Return value is a `FILE *` — a somewhat mysterious thing, but one we can pass to other functions. If `NULL`, the open did not succeed. (Can you think of reasons this might happen?)

Working With Streams in C

Slide 7

- To read from an input stream — `fscanf` or `fgetc`, almost identical to `scanf` and `getchar`. To write to an output stream — `fprintf` or `fputc`, almost identical to `printf` and `putchar`.
- When done with a stream, `fclose` to tidy up. (Particularly important for output files, which otherwise may not be completely written out.)
- (Simple examples.)
- How to get names of files from user? Well ...

Text Input in C

Slide 8

- We've seen how to read text a character at a time. Many programming languages provide nice ways to get a whole line at a time. C isn't really one of them (and why that is may become clearer after we talk about arrays a week or two from now).
- Many books show various not-perfect approaches; what I prefer instead for filenames is to "pass them as command-line arguments".

Slide 9

Another Way to Get Input — Command-Line Arguments

- (We can't completely discuss this until a bit later, but it's so useful for working with files that we'll do just a bit now.)
- You may have observed that most of the commands you use don't prompt you for input, but instead decide what to do based on what you type on the command line after the command name? so the program must be getting that information somehow, but — how? "command-line arguments" (e.g., for the command `gcc -Wall hello.c` there are two command-line arguments).
(And those commands? Many of them are C programs!)
- Most programming languages provide a way to access this text, often via some sort of argument to the main function/method.

Slide 10

Command-Line Arguments in C

- In C, command-line arguments are passed to `main` as an array of text strings (we'll talk soon about arrays and text strings). So if you define `main` as

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

`argc` is the number of arguments, plus one, and `argv` is an array of strings containing the arguments.
(“Plus one”? yes, one of the arguments is something system-dependent, often the path for the program's executable.)
- Reference individual arguments via `argv[0]`, `argv[1]`, `argv[2]`, etc.
- This turns out to be (I think!) a good way to pass text such as filenames to your program.

Files — Examples

- (Example of character-oriented I/O with files.)
- (Example of formatted I/O with files.)

Slide 11

Minute Essay

- Can you think of problems you might want to solve for which a program using files would be a help?

Slide 12