Slide 1

Administrivia

- Homework 1 grades sent by e-mail. This is how you will get grades and feedback for programming assignments.
- Reminder: Homework 2 due today.
- Homework 3 on the Web; due next week. Also not especially algorithmically challenging. Be advised that you should be able to complete this assignment with only the C constructs mentioned in class or reading, and that's what I intend.

Slide 2

More Administrivia

- When turning in homework, please do identify in the subject line both the course and the assignment. I ask for this so it's easy for me to drop it into the right "folder" for grading — I'll be getting assignments by e-mail for four courses this semester.
- As with the minute essays, if there's an urgent question in what you're sending, put "question" or "urgent" in the subject line.
- If you need/want to mail homework to me when you're logged in remotely (or if you just want to know one more thing you can do from the command line!), see the `mail-files` script on the "Sample programs" page.

Slide 3

Minute Essay From Last Lecture

- No clear consensus with regard to buying the textbook — several are doing so, many not. One person is getting a copy of the mentioned reference book! “FYI”.

Slide 4

A Few More Words About `vim`

- Most people seem to have learned something from the tutorial. Good! `vim` is painful to use if you know only the bare minimum but starts to seem reasonable when you know more.
- I have an introduction to `vim` linked from the class “Useful links” page. The first part you may know, but at the end there’s a section about behaviors that may have puzzled you.
- Two keys to know about if you don’t already: `u` to “undo” previous command, and `%` to find matching parenthesis/brace. Also you might like “visual mode”; `:help visual-mode` to learn more.

Slide 5

A Few Words About "Old C" Versus "New C"

- First ANSI standard for C — 1989 ("C89"). Widely adopted, but has some annoying limitations.
- Later standard — 1999 ("C99"). Many features are widely implemented, but few compilers support the full standard, and older programs (and some programmers concerned about maximum portability) don't use new features. Much of what we do in this class will focus on older standard for this reason. Some additions will work in `gcc` only with `-std=c99` option.
- Still-later standard (2011) exists but is not (yet?) widely implemented.

Slide 6

Sidebar — Compiler Options

- Earlier I showed the simplest way to use `gcc` to compile a program. But there are many variations — *options*. Specify on the command line, ahead of name of input file.
- Some of the most useful:
  - `-Wall` and `-pedantic` warn you about dangerous and non-standard things. `-Wall` *highly* recommended.
  - `-std=c99` allows you to use features new with C99.
  - `-o` allows you to name the output file (default `a.out`).
- Automate with `make` (more later).

Slide 7

Expressions in C (Review)

- C (like many other programming languages) has a notion of an *expression*.
- Every expression has a *value*, and computing this value is called *evaluating the expression*.
- Sometimes evaluating an expression also produces *side effects*. E.g., a call to `printf` is an expression; evaluating it produces a result (yes, really!) and a side effect.

  Assignment is also an expression(!), so you can write

  ```
  a = b = 0;
  ```
- Many, many operators of different kinds. Last time we talked about arithmetic operators, including pre/post increment/decrement.

Slide 8

Expressions — “Caveat Programmer”

- Expressions can be quite complex. How they’re evaluated depends on rules of precedence and associativity. My advice — when in doubt, use parentheses! Example: `(x + y) / 2` versus `x + y / 2`.
- C standard is somewhat imprecise about details of expression evaluation — e.g., in evaluating

  ```
  f() + g()
  ```

  two functions could be called in either order. (Why? To allow greater flexibility for implementers, possible allow for more-efficient programs.)
- C syntax allows programmers to write statements/expressions in which a variable’s value is changed more than once, e.g.,

  ```
  i = (i++) + (i--);
  ```

  Syntactically legal, but standard says that such expressions invoke “undefined behavior”. Best to avoid that!

Slide 9

Conditional Execution

- As in other procedural languages, C has syntax for saying that some code should be executed only if some condition holds.
- Syntax is

  `if (` *boolean-expression* `)`
  statement1
  `else`
  statement2

  where *statement1* and *statement2* can be single statements or blocks enclosed in curly braces.
- You can build up chains of conditions by making the statement after `else` another `if`, and you can omit the `else` and following statement. (The ideas here should be very familiar, and for most of you even the syntax should be pretty much what you know.)

Slide 10

Boolean Expressions and Values in C

- Early standards for C didn't include a Boolean type, but represented it with integers, with the convention that 0 is `false` and anything else `true`.
- Later standards include a `bool` type, but if you use it for variables you must be sure the compiler knows you want to compile with the right standard, and you must include

  ```
  #include <stdbool.h>
  ```
- Partly as a consequence of this, you can use an integer-valued expression where a Boolean expression is needed.

  (So you can write `if (a = b)`, but it won't do what you probably want!)
- Of course(?), C also includes the usual range of relational and Boolean operators.

Conditional Expressions

- Scala and Python both provide a way to include if/else idea within an expression.
- C does too, but it's not as obvious — "ternary operator", e.g.,

```
int sign = (x >= 0) ? 1 : -1;
```

Slide 11

Conditional Execution — One More Thing

- One other conditional-execution construct you may encounter — `switch`. Basically a short form of if/elseif/else. Somewhat like `match` in Scala but nowhere near as powerful. Example:

```
char c; /* code to set value omitted */
switch (c) {
    case 'a': printf("first case\n"); break;
    case 'b': printf("second case\n"); break;
    default:  printf("default\n");
}
```

Slide 12

Slide 13

Simple Input, Revisited

- As mentioned last time, there *is* a way to find out whether `scanf` was able to actually read something of the requested type(s).
- Considered as an expression, call to `scanf` has a value, namely the number of variables successfully read. C-idiomatic way to check for success is

```
if (scanf("%d %d", &var1, &var2) == 2) ....
```

- Even with this, `scanf` is not entirely satisfactory as a way of getting even numeric input, let alone text, but it's commonly used and will do for now.

Slide 14

Functions in C

- Functions in C are conceptually much like functions in other procedural programming languages. (Methods 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.
- Also, C functions are not supposed to be nested (though some compilers allow it).

Slide 15

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 — more later when we talk about them.

Slide 16

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 17

Library Functions in C

- C does include a library of standard functions, though it's nowhere near 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.

  (If when you type `man` *function* you get something other than a description of *function* — as you do for `printf`, for example — try `man 3` *function*). Explanation on request.)

Slide 18

Conditional Execution and Functions in C — Example(s)

- (Examples as time permits.)

Minute Essay

- What (if anything!) was interesting or difficult or otherwise noteworthy about Homework 2?