Slide 1

Administrivia

- Homework 1 on the Web. Due a week from today. Two problems, algorithmically straightforward, so the main challenges will be syntax and tools.
- (Notice change in reading assignments.)

Slide 2

Minute Essay From Last Lecture

- How does C handle operators? some sort of stack?

  Up to the compiler, but evaluating expressions with operators might involve a stack.
- Why do you need that backslash-n in the call to `printf`?

  It "prints" an end-of-line character.
- Could `main` return something other than an `int`?

  Not in a program conforming to the C standard(s), at least in a so-called "hosted" environment (roughly, one in which the program runs under an O/S). Compilers are allowed to define other forms of `main`, and `void main(void)` is common in some environments (e.g., Windows). Almost a religious-war topic in some circles!

Slide 3

Preprocessor Directives — A Bit More

- Examples so far have started with `#include` directive to tell compiler where to find information about I/O library functions. This is input to the "preprocessor".
- Another useful directive is `#define`, to give meaningful names to constants, e.g.,

```
#define IMPRECISE_PI 3.14159
```

Slide 4

A Few Words About Syntax

- Python programmers should note that in C, unlike in Python, indentation is not generally syntactically significant. (But adopting a consistent style makes your code more readable to humans.)
- Scala programmers should note that in C, unlike in Scala, the compiler will not add semicolons to the ends of statements or guess about types.

Slide 5

Variables — Review/Recap

- In order to do anything useful we usually (though not always!) need some variables. In C, variables must be *declared* before being used. (Contrast with Python.) Declaration specifies name and type. (Contrast with Scala.)
- Once you have variables, you can assign values to them, using *expressions* that range from simple constants to complex math-like formulas involving constants and/or other variables.

Slide 6

Expressions — Recap/Review

- C provides support for the usual(?) arithmetic operators. Notice that operations on integer types produce integers.
- C also includes some operators (e.g., ++, +=) that produce side effects.

Slide 7

Expressions — "Caveat Programmer"

- 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 8

Simple Output, Revisited

- Simple/typical way to produce output (to "standard output" — terminal for now) is with library function `printf`.
- Parameters are "format string", which may include "conversion specifications", followed by zero or more expressions, one for each conversion specification.

Slide 9

Simple Input

- Simple way to get integer/float input (from "standard input" — keyboard for now) is with library function `scanf`. For now we will look only at simple forms:

  ```
  scanf("%d", &variable_name);
  scanf("%d %d", &var1, &var2);
  ```

  etc. Parameters similar to `printf`, except for that ampersand. (It generates a pointer. More about that later!)
- Considered as an expression, call to `scanf` has a value, namely the number of variables successfully read. (So you can check it to make sure valid input was entered.)

Slide 10

Conditional Execution

- Also 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 (and should probably be indented, for the convenience of human readers).
- 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; only the syntax should be new.)

Slide 11

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 12

Example — Finding Roots of a Quadratic Equation

- As an example of all of this, let's write a program that finds and prints the root(s) of a quadratic equation of the form

$$ax^2 + bx + c = 0$$

using the familiar(?) formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- (We'll also include in this program an example of getting input from standard input.)

Slide 13

Minute Essay

- None — sign in.