Slide 1

Administrivia

- Homework 1 being graded; should be returned next time.
- Remember that some (most?) code from class will be on the Web later that day, linked from the "Sample programs" page here.

Slide 2

C Built-In Types, Continued

- Last time we used only `int`, which can represent integers only, limited range (but it's a fairly big limit – about two million). Other integer types with smaller/larger ranges include `byte`, `short`, and `long`. All can be signed or unsigned (non-negative only).
- For numbers with a fractional part, `float` and `double`. More about them soon.
- For character data, `char`. For more than one character ("character string") we need to know about arrays — later in the course.
- C99 adds several types, including ones for booleans and complex numbers.

Slide 3

Expressions in C

- C (like many other programming languages) has a notion of an *expression*, which looks and works a little like a mathematical formula. Simple examples (assuming we've declared variables `x` and `y`):
  - `5`
  - `x`
  - `y + 5`
  - `(x + y) / 2`
- Every expression has a *value*, and computing this value is called *evaluating the expression*. Evaluate the above expressions, assuming `x` has value 10 and `y` has value 20 . . .

Slide 4

Expressions in C, Continued

- Sometimes evaluating an expression also produces changes to variables in the expression or other variables; these are called *side effects*. Examples:
  - `x = 10`
  - `printf("hello, world\n)`

  (Yes, really! Usually we don't care about much about the values of these expressions, just their side effects.)
- Many, many operators of different kinds. For now we'll look only at the ones for arithmetic.

Slide 5

Arithmetic Expressions — Operators

- Usual arithmetic operators `+`, `-`, `*` (multiplication), `/` (division). (`+` and `-` can be unary too.)

  Notice that division, applied to integers, discards any remainder. This is so the result will be an integer too, and can even be useful. What if you want a fraction? Later.
- Also `%` operator for getting remainder; e.g., `x % 2` is 0 if `x` is even, 1 if it's odd.
- Other useful arithmetic operators include pre/post increment/decrement, bit shifts.
- Expressions can be quite complex. How they're evaluated depends on rules of precedence and associativity. Full details in your textbooks, but my advice is — when in doubt, use parentheses! Example: `(x + y) / 2` versus `x + y / 2`.

Slide 6

Statements in C

- C programs are made up of *statements* (usually collected inside *functions* — more about them later).
- Statements come in several types:
  - Null (`;`).
  - Expression (*expression* `;`).
  - Return (`return` *expression* `;`).
  - Compound (more later).

Slide 7

A Simple Program — Making Change

- Goal — use what we know so far to develop a simple program for "making change".
- Input: An integer N representing number of pennies.
- Output: How many coins of different denominations are needed to make up N, assuming we use the minimum number of smaller coins (e.g., one dollar rather than four quarters).

  Example: For 237, we should output something like this:

```
Input:  237
2 dollars
1 quarters
1 dimes
0 nickels
2 pennies
```

Slide 8

Making Change, Continued

- First step in writing a program is — start up a text editor and start typing code? Well, no.
- Instead, first try to understand the problem — what kinds of values can be input, what's supposed to be output. Possibly work through some examples.
- *Now* start the text editor — but rather than starting to type code, type comments describing the input and output. This is good to have and also helps make sure you do understand.

Slide 9

Making Change, Continued

- Now think about how to get the output we want from the input we have.
- Once we have a plan, we can start turning it into code. For anything very complex, it can help to first type in the plan as comments, then fill in code.
- (Do this . . .)

Slide 10

Minute Essay

- None — quiz.