Administrivia

- Reminder: Homework 2 due Wednesday.
- First quiz next Monday. (More on next slide.) Topics include anything we cover up through Friday (so, C programming as covered so far, material about base 2 and how it's used to represent integers in computers).

Slide 1

Quizzes

- About 10 minutes.
- "Open book / open notes": access to textbook, anything on the course Web site, your notes and graded or ungraded work, nothing else.
- Can use computer only to view allowed material (so, no use of `gcc` or calculator).
- Meant to be not stressful and not something you need to study for, beyond a quick review.

Slide 2

Slide 3

Minute Essay From Last Lecture

- Pretty much everyone got the point about different schemes for floating point — more bits for the exponent means a larger range of values, while more bits for significant figures means more precision.

Slide 4

C and Representing Numbers — Integers

- Computer hardware typically represents integers as a fixed number of binary digits. Most hardware uses “two’s complement” idea to allow for representing negative numbers.
- C, like many (but not all!) programming languages largely bases its notion of integer data on this, but also has a notion of different types with different sizes (`short`, `int`, `long`, `long long`). Note that unlike many more-recent languages, C defines for each type a minimum range rather than a definite size. (C99 does define some fixed-size types. Later maybe.)

  Intent is to allow efficient implementation on a wide range of platforms, but means some care must be taken if you want portability.

Slide 5

C and Representing Numbers — Integers, Continued

- Because data is fixed in size, “overflow” is possible. Some hardware supports detecting that, but C doesn’t assume that’s possible, so no easy way to check. Programmers should check that each variable is of a type big enough to hold all anticipated values.
- (Why oh why . . . ? My guess is that it’s in keeping with the goals of “possible to implement on many diverse platforms” and “efficient code”.)

Slide 6

C and Representing Numbers — Real Numbers

- Hardware also typically supports “floating-point” numbers, with a representation based on a base-2 version of scientific notation. This allows representing not only fractional quantities but also allows representing larger numbers than would be possible with fixed-length integers. Note that only fractions that can be written with a denominator that’s a power of two can be represented exactly.
- Again C goes along with this and provides different “sizes” (`float` and `double`).

Slide 7

Text Data

- Remember that computers represent everything using ones and zeros. How do we then get text? well, we have to come up with some way of "encoding" text characters as fixed-length sequences of ones and zeros — i.e., as small(ish) numbers.
- (To be continued later in the semester.)

Slide 8

Conditional Execution

- So far all our programs have executed the same statements every time, just maybe with different numbers.
- Often, though, we want to be able to do different things in different circumstances — for example, print an error message and stop if the input values don't make sense (such as a negative number for the program to make change).
- So, C (like most languages) provides some constructs for *conditional execution*. Before we talk about them, we need . . .

Slide 9

Boolean Expressions

- A *Boolean value* is either *true* or *false*; a *Boolean expression* is something that evaluates to true or false.
- We can make simple examples in C using familiar math comparison operators. Examples:
  - `x > 10`
  - `y <= 5`
  - `x == y` (*Note* the use of == and not =!)

Slide 10

Boolean Expressions, Continued

- *Boolean algebra* defines some operators on these values; the most important for us are written in C as
  - `!` — "not", true if the operand is false.
  - `&&` — "and", true if both operands are true.
  - `||` — "or", true if either operand is true (or both are).
- Can use these to build up complex expressions. As with arithmetic expressions, use parentheses when in doubt. Examples:
  - `(x >= 0) && (x <= 10)`
  - `!(x == y)` (though we could also just write `x != y`).

Slide 11

Conditional Execution in C — `if/else`

- To execute a statement if an expression evaluates to true, use `if`:

```
if (x > 0)
    printf("greater than zero\n");
```

- To execute one statement if an expression is true, another if it's false, use `if` and `else`:

```
if (x > 0)
    printf("greater than zero\n");
else
    printf("not greater than zero\n");
```

Slide 12

`if/else`, Continued

- To execute a group ("block") of statements rather than just a single statement, use curly braces for grouping:

```
if (x > 0) {
    printf("greater than zero\n");
    printf("and that is good\n");
}
else {
    printf("not greater than zero\n");
    printf("and that is bad\n");
}
```

- What happens if you forget the braces? The program may still compile and run, *but it probably won't do what you meant.*

Slide 13

`if/else`, Continued

- Several styles for where to put the curly braces and how to indent. Which is best? Opinions differ. Some people insist on One True Way; I say pick one that's readable (to humans) and stick with it.
- (Remember that you're writing for "two audiences" — compiler and humans.)
- `vim` should help you with this — it has built-in indenting styles for many programming languages. If indentation gets out of synch with code because of editing, can reindent:

  `==` to reindent current line.

  `gg=G` to reindent whole file (`gg` to move to start of file, `=` to reindent, `G` to continue to end).

Slide 14

Conditional Execution, Continued

- What if more than two conditions we want to check for? Could "nest" `if/else` constructs, e.g.,

```
if (x < 0) {
    printf("less than\n");
}
else {
    if (x > 0) {
        printf("greater than\n");
    }
    else {
        printf("equal\n");
    }
}
```

- But this gets ugly fairly quickly. So . . .

Slide 15

Conditional Execution, Continued

- Better:

```
if (x < 0) {
    printf("less than\n");
}
else if (x > 0) {
    printf("greater than\n");
}
else {
    printf("equal\n");
}
```

- Can have as many cases as we need; can omit `else` if not needed.

Slide 16

Boolean Expressions in C

- Although there are only two Boolean values, C represents them as `int`s, with 0 meaning false and anything else meaning true.
- One consequence: Integer expressions can be used in place of Boolean expressions.

  So for example

  ```
  if (x == y)
  ```

  and

  ```
  if (x = y)
  ```

  are both valid C, but they mean different things. (The second one assigns the value of `y` to `x` and is considered true if the result is nonzero. Almost never what you want! `gcc` will warn you, at least with `-Wall`.)

Slide 17

Simple I/O, Revisited

- We can now do simple error-checking that `scanf` did what we asked. C-idiomatic way looks like this simple example:

```
if (scanf("%d", &x) == 1)
     /* okay */
else
     /* error */
```

- (More about what this means when we talk about functions, soon.)

Slide 18

Simple I/O, Revisited

- Doing a really good job with interactive input is surprisingly tricky — what constitutes an error, how/whether to prompt user to try again.
- So for this class we'll focus on some simple safety checks: if input should be numeric it is, values make sense for the program (e.g., input to “count change” program is not negative).
- For this class it's usually best to just bail out on bad input, rather than retrying.

Slide 19

Example — Finding Roots of a Quadratic Equation

- As a rather math-y example, let's write a program to compute and print the roots of a quadratic equation

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

- We'll use the formula

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

and try to account for as many cases as we can . . . (To be continued.)

Slide 20

Minute Essay

- Have you previously used something that supports conditional execution (Matlab?), and if so how does C's version compare to it?
- I should have asked last time, but belatedly: How much of the material about binary numbers was new to you and how much was review?