Slide 1

Administrivia

- (What were you doing on this date in 2001?)
- Homework 1 grades e-mailed Saturday. (This is how you will get feedback on programming assignments.)
- Reminder: Homework 2 due Wednesday. Please remember in the subject line to identify both the course and the assignment! (Most of you did for Homework 1, but not all.)
- First quiz next Monday. About 10 minutes, end of class, "open book / open notes" (meaning access to textbook, your notes, anything on the course Web site, nothing else). Topics include anything we cover up through Friday (so, C programming as covered so far, material about base 2 and how used to represent integers in computers). Meant to be not stressful and not something you need to study for, beyond a quick review.

Slide 2

Minute Essay From Last Lecture

- Off-topic but interesting questions:

  "I've heard of 64-bit encryption; anything stronger?" (Yes.)

  "What does it mean to say a processor is 32-bit? 64-bit? A friend also says that while addition and multiplication are both fast, division is not. Why?"
- Most had no questions about Friday's lecture, but a few were uncertain about whether any of it matters. So . . .

Slide 3

Binary Numbers — Recap

- General review of binary numbers and arithmetic on them is meant as background for understanding how computers represent integers internally. That in turn is meant to help you understand some of the limitations associated with C integer types.
- Discussion of how to convert among number systems is mostly I-think-useful background, but also because the two ways of converting from decimal to binary show how there can be more than one way to solve a problem.

Slide 4

C and Representing Numbers — Integers

- Computer hardware typically represents integers as a fixed number of binary digits, using "two's complement" idea to allow for representing negative numbers.
- C, like many (but not all!) programming languages bases its notion of integer data on this, but also has a notion of different types with different sizes.
- Unlike many more-recent languages, C defines for each type a minimum range rather than a definite size. 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 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. (Review slide from last time.) This allows representing not only fractional quantities but also allows representing larger numbers than would be possible with fixed-length integers. Notice 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`). As with integers, exact sizes not specified, only minimum criteria.

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

Sidebar(?): Type Conversions

- Implicit conversions: When you assign a value of one type to another (e.g., `float` to `int`), or write an expression that mixes types, C will perform an implicit conversion.
- Explicit conversions: Putting a type in parentheses before an expression means you want to convert to the indicated type. Example:

  ```
  (float) (1 / 2)
  ```

  versus

  ```
  (float) 1 / (float) 2
  ```

Slide 9

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 10

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 11

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 12

Boolean Expressions in C

- Although there are only two Boolean values, C represents them as `int`s, with 0 meaning true and anything else meaning false. (Usually you don't care about this, but it can be good to know.)
- This means that the compiler will accept both `x == y` and `x = y`, *but they mean different things*. Very common mistake (and not just for beginners!). Compiler will often warn you about this (though you may need to use that `-Wall` flag).

Slide 13

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 14

`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 15

`if/else`, Continued

- Several styles for where to put the curly braces. Which is best? Some people care; I say pick one that's readable (to humans) and stick with it.
- (Remember that you're writing for "two audiences" — compiler an humans.)

Slide 16

Conditional Execution, Continued

- What if more than two? We 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 17

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 18

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 */
```

Slide 19

Simple I/O, Revisited

- Doing a really good job with interactive input is surprisingly tricky — what constitutes an error, how do you 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, etc.).
- For this class it's usually best to just bail out on bad input, rather than retrying.

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?