Slide 1

Administrivia

- Reminder: Homework 2 due today at 5pm. Send me your program source code (two text files) by e-mail.
- Homework 3 will be on the Web later today / early tomorrow. Due next Thursday.
- Quiz solutions will be online shortly after class. I will also usually bring one printed solution you can look at after you turn in your paper.

Slide 2

Encouraging(?) Words

- Writing programs is not easy. If you've finished at least one of the Homework 2 problems, you've done something most people don't know how to do (!). And you're doing it with tools that aren't particularly easy either.
- Quote of the day/week/something, from a key figure in the early days of computing:

  "As soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent finding mistakes in my own programs." (Maurice Wilkes: 1948)

Slide 3

Tips for Using the Command-Line Environment

- Someone asked about commands for moving and copying files, etc. Summary available here, linked from "Lecture topics and assignments" and "Useful links" pages. Also mentions shortcuts such as arrow keys to repeat previous command, tab to fill in filename.
- Unix fan's response to a claim that Unix isn't user-friendly: "Sure it is; it's just choosy about its friends."

Slide 4

Tips for Dealing with Compiler Errors

- `gcc`'s error messages aren't always helpful to beginners. As you get experience they will mean more. Some advice for now:
- Fix the first error mentioned, then try again to compile. Sometimes this will resolve many reported errors at once.
- Most error messages reference a line number. Look at that line and the line above; you can often spot the problem even if you don't completely understand the error message.

Slide 5

Tips for Dealing With Runtime Errors

- You may have observed that even after the program compiles without error, it may not work — it may crash (e.g., give a cryptic error message and stop), or it may run but give wrong answers.
- A very typical error message is `Segmentation fault`. Most likely cause right now is not putting that `&` in front of a variable name in a `scanf` call.
- For dealing with wrong answers . . . (next slide).

Slide 6

Tracing Code

- A valuable skill to have is working through what the computer will do when it executes your program — "tracing code" (also known as "playing computer").
- Idea is to write down names of variables, their values; when one changes, cross out old value and put in new one.
- Let's do an example using code from last week.

Slide 7

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*

Slide 8

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 9

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 10

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).

Slide 11

Conditional Execution — `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

Minute Essay

- None — quiz.