

Slide 1

Administrivia

- Reminder: Homework 4 due today.
- Homework 5 on the Web; due next Friday.
- Sample solution for Homework 3 posted.
- As I hope I've mentioned previously, but to say it maybe-again: Most code I/we write in class will be available via the course Web site "sample programs" page. So no need to worry too much about keeping up as I type in code.
Also note that these are tidied-up versions, with more comments than I take time to do in class — I don't take time in class to write many comments or to make error messages helpful, but you should in your homeworks, and the Web-site versions are examples of what's desired.

Slide 2

Minute Essay From Last Lecture

- Everyone had seen loops in some other context, though some said they weren't comfortable with them or didn't remember much. We'll use them a lot!
- One person asked about when to use `while` and when to use `for`. In C you can (almost?) always use either one, so it's kind of a matter of style, but ... A little more today.

Loops — Recap/Review

Slide 3

- Loops, like recursion, are a way to repeat some operation. Useful in applying the same operation to all elements of some collection or in repeating an operation until some condition is met.
- What all these ways of repetition have in common:
 - A starting point (initial condition, first element of a collection).
 - The operation to repeat.
 - How to move from one iteration to the next.
 - When to stop (though the syntax often is such that what you actually say is when *not* to stop).

Loops — Recap/Review, Continued

Slide 4

- Last time we looked at basic syntax for `for`, `while`, and `do while` loops.
- When to use which one? “it depends”, and sometimes a matter of style, but in general:
 - If you know how many times you want to repeat something, a `for` loop is probably more idiomatic.
 - If you don't, a `while` or `do while` is probably better. (I find that I only rarely use `do while`.)

Loops Versus Recursion

Slide 5

- As noted in class, recursive functions can be simple to write but potentially inefficient (though in some cases a sufficiently smart compiler can reduce or eliminate the inefficiency — look up “tail recursion” to find out more).
- For other problems, a loop is simpler to write — loop versions of many of the in-class examples of recursion are as simple or simpler, and that program to get an integer from input without using `scanf` would have been much simpler with loops. So it may seem that loops are better.
- But there are problems for which recursive solutions are much simpler to write and get right, while non-recursive solutions are decidedly not simple — anything involving “trees”, plus at least two widely-used algorithms for “sorting” (putting things in order).

Loops — More Examples

Slide 6

- First let's modify the “sum of integers” program to compute an average. Both programs are examples of what one might call a “running total” pattern.
- We could also rewrite that “get an integer” program to use a loop or loops . . .

Numerical Computation

Slide 7

- A big use of computers is in solving (exactly or approximately) mathematical problems — “numerical computation” or “numerical analysis”. Matlab is one tool for this, and/or you can write your own programs in a general-purpose programming language. Often (maybe always?) these involve various forms of repetition.
- An example is “numerical integration”, in which you approximate a definite integral (area under a curve) by computing areas of rectangles and adding them up. As an example . . . (next time).

Minute Essay

Slide 8

- What was noteworthy (interesting, difficult, etc.) about Homework 4?