Slide 1

Administrivia

- As of writing this, it looks like only four people have turned in reasonably complete versions of Homework 8. I'm willing to accept this assignment through end of the day Monday, so if you haven't finished yet, I say don't give up.
- Monday and Wednesday I showed code for a recursive implementation of a sorted linked list. I've also written an iterative (loops) version and put it on the sample programs page, if you're interested.

Slide 2

More Administrivia

- About grades, remember from the syllabus how I do this — every assignment (incluing exams and quizzes) has a "perfect score", and I add those up to give an overall perfect score, add up your scores, and divide your score by the perfect score.
- Extra-credit problems still in work, but I will try to post them by Monday. You can't lose anything (except time?) working on them — any points you get are added to your score but don't affect the perfect-score number.
- Final next Friday. Review sheet to be posted by Monday. A little more when we do the minute essay.

Slide 3

“Collections” Types Review/Revisited

- Can think of a “collection type” as a way of representing a collection of data items. Arrays are one; linked (linear) lists are another.
- There are many others, all beyond what we can cover in this course; more-advanced courses in programming and computer science are apt to discuss many or all of these . . .

Slide 4

Sidebar: Abstract Data Types

- In computer science we talk often about “abstract data types”.
- Somewhat formally, an abstract data type is a set of values and some operations on them.
- Simple example: fixed-size integers, with obvious(?) values and operations including arithmetic and bit manipulation.
- Simple example: Boolean values, also with obvious values and operations including logical “and”, logical “or”, etc.
- Less simple example: Linear list, with operations including insert, delete, traverse, indexed access. Could be implemented with an array or with a linked-list structure.

Slide 5

Stacks

- A “stack” is a list of values of a particular type (integers, say, or something more complicated), with a restricted set of operations, often just “enqueue” (add to the “tail” of the queue), “pop” (remove and return the top element), and “is empty?” (I.e, this is a “last-in, first-out” linear list.)
- Turns out to be widely useful. This idea is how functions (including recursive ones) are typically implemented, with a stack each element of which contain values passed to the function, local variables, and a “return address”.
- Could be implemented using an array or as linked list.

Slide 6

Queues

- A “queue” is a list of values of a particular type (integers, say, or something more complicated), with a restricted set of operations, often just “enqueue” (add to the “tail” of the queue), “dequeue” (remove and return the first element), and “is empty?” (I.e, this is a “first-in, first-out” linear list.)
- Also widely useful any time you need to maintain something in first-in first-out order.
- Could be implemented using an array (“circular queue” idea) or as linked list.

Trees

Slide 7

- A "tree" in computer science is a way of representing data organized in some hierarchical way. Each is a collection of "nodes" that store a value and pointers to "child nodes".
- In the same way as a linked list is represented by a pointer to the first node, a tree is represented by a pointer to its "root node".
- Useful any time you want to represent a hierarchical structure (directories and files, e.g.).

Trees, Continued

Slide 8

- "Binary trees" (in which each node has at most two children) are simpler to represent and effective in many situations.
- "Binary search tree" is a binary tree where everything in the "left subtree" of a node has smaller values and everything in the "right subtree". Allows faster lookup, sort of like binary search in an array.
- "Heap" is a binary tree where everything in *both* subtrees of a node has larger values. Useful for maintaining a "priority queue" (with operations including "remove and return smallest element" and "insert element").

Graphs

Slide 9

- In some mathematical contexts, “graph” means a collection of nodes and edges connecting them. Edges can be uni- or bi-directional. Nodes can store values, and associated with each edge there can also be a value (a “weight”).
- Also turns out to be widely useful as a way of reprenting all kinds of things — e.g., the classic traveling-salesperson problem.
- Can implement used a linked data structure or with various types of 2D arrays.

Hash Table

Slide 10

- A “hash table” is a meant-to-be-efficient way of storing (key, value) pairs, such that looking up a value using the key is reasonably fast.
- Basic idea is to define a reasonably-sized array and some way to map from a key to an index into this array (“hash function”). Each element of the array points to a list of (key, value) pairs, and to look for a particular key, you use the hash function to map into the array and then search the list.

  If the hash function and the table size are well-chosen, these lists will be short, perhaps in many cases of length 1, making lookup fast.
- Also widely useful in the many circumstances in which fast lookup is desirable. (As an example — in a minute essay recently someone asked about fast access to items in “a database”? this idea would probably work for that.)

Slide 11

"Ragged Arrays"

- (This isn't really an abstract data type, but it's another kind of collection, so mention it here.)
- Sometimes useful to be able to define a "ragged array" — an array in which rows can have different sizes. Not difficult in C, if you represent each row as an array and have some way of remembering sizes of rows.
- An obvious(?) example is the second argument to `main` — it's a ragged array of characters, with no need to explicitly save how many columns in each row since strings are null-terminated.

Slide 12

Sorted Linked List Revisited

- (Compare and contrast recursive and iterative version of code. Perhaps interesting to note that some things seem easier in one version and some in the other.)

Minute Essay

- About the final, like the midterm and quizzes it will be open-book etc. Questions will be similar to those on quizzes and midterm, *except* that I'm considering one change:

  People seem to have a lot of trouble with the "write some code" questions. I would be willing to relax the rule about not using computers so you could type in your answers and try them, but that would mean no questions of the form "what does this code do?" Would you prefer that I do that (relax the rule), or not?

- There's so much we just haven't been able to cover in this course. Any questions you'd like me to try to answer Monday? (I have some examples of full-screen text-mode programs in C that I could show, or one that does something graphical. Both use "third-party" libraries but interesting??)