Abstract data types

May 14, 2004

Abstract data types

A generic mechanism for explaining the format in which data is stored and operated on

Independent of a particular implementation (programming language, computer program, etc)

Examples:
- lists (array and linked)
- stacks and queues
- trees

List

- Three properties:
  - homogeneity (all items are of the same type)
  - order (all items have a notion of “previous item” and “next item”)
  - variable length

List actions

- Add item
- Remove item
- Search for item
- Sort
- Calculate length
List implementations

- linked
- array

Array

- Items are placed sequentially in memory
- Refer to item by its *index* (position in list)
  - first item is at position 0
- Length is an important concept to an array
  - defines the “limit” of our list
  - stored with the list, so we don't have to compute it

Array actions

- Add item to sorted list:
  - loop through the existing items on the list until you find the “slot” into which the new item should go
  - starting with the last item in the list, move all items down by one until this “slot” opens up
  - put the new item into this “slot”
- Delete item from sorted (or unsorted) list:
  - search the list until you find the item to delete
  - delete the item
  - move every item after the deleted item up by one “slot”, until there are no holes in memory.

Linked list

- *Node*:
  - one part contains the item
  - the other part contains a “pointer” to the next node in the list
- Draw as a “boxes and arrows” diagram
- Items do not have to be next to each other in memory! (can be anywhere)
  - the “pointer” indicates the memory address at which we can find the next item
  - pointer also to the start (“head”) of the list
Linked list actions

- **Add item to sorted list:**
  - Create the new node anywhere in memory. Leave the pointer blank, for now.
  - Starting at the head of the list, follow the pointers and look at each node until you find the position at which to place the new item.
  - Go back to the previous node.
  - Copy the previous node's pointer into the new node's pointer field. The new node now points to this next node.
  - Update the previous node's pointer so that it points to the new node's address.

Linked list actions (cont.)

- **Delete node from (sorted or unsorted) list:**
  - Starting at the head, search through the list until you find the node to delete. Make a note of to which node it points.
  - Go back to the previous node.
  - Update the previous node's pointer so that it points to the same node as the node that's being deleted.
  - The node still exists, but it's out of the list.

Linked list actions (cont.)

- **Calculate the length of the list:**
  - Start at the head.
  - Follow the pointers.
  - Each time you encounter a node, add one to the counter.