6.1 Lists as an ADT
focus on what lists do
properties of data
homogeneous
finite length
sequential
operations
constructor - creates empty list
empty - check if list is empty
insert - add an item
delete - remove an item
traverse - various operations that go through a list sequentially
search - find an element
output - print list contents
copy - create a copy
sort - rearrange elements
various implementations possible
static arrays
dynamic arrays
linked lists w/ pointers (most common)
linked lists w/ 2D arrays

6.2 Static Array Implementation
list will have max capacity equal to the size of the array
list stored sequentially in memory
must be able to allocate mem chunk of appropriate size
head of list is slot 0
need to add a count of elements in list
tells if list is empty or full
operations
constructor - sets count of elements to 0
empty - check if count is 0
traverse - for (i=0; i<count; i++)
takes n for loops to traverse
search traversal averages half the list
sort traversal can take longer depending on the sort algorithm
insert - depends on type of insert
tail insert
if (count < max-capacity)
    array[count++] = element
else
    // no space left error
insert mid-list or head
have to move elements down a slot
have to validate given position
can be used for tail insert too
if (count < max-capacity) {
if (pos < 0 || pos > count)
    // issue "bad position" error
else {
    for(i = count; i > pos; i--)
        array[i] = array[i-1];
    array[pos] = element;
    count++;
}
else {
    // issue "no space" error
}
takes up to n for loops to shift current elements
    worst case is head insert
    average case is mid-list insert
    best case is tail insert
delete - also has to shift elements
    if (empty()) {
        // issue "empty list" error
    }
else if (pos < 0 || pos >= count) {
    // issue "illegal position" error
}
else {
    for (i = pos; i < count-1; i++)
        array[i] = array[i +1];
    count--;}
} best/worst/average same as for insert
implementation details
how to define element type
    #define macro
typedef (book)
template class
how to define max capacity
    #define macro
const int (book)
class variable - should be static
    only one copy of static vars across all class instances
    static const int capacity = 100;
pp 262-269 has book's implementation
6.3 Dynamic Amery Implementation
Operations similar to above
    default constructor should select some default capacity & allocate mem
    add constructor to take an int for capacity & allocate mem
    add destructor to deallocate mem
    copy has to allocate space for new list first
    add assignment operator to deal w/ memory allocation issue
    otherwise both lists point to same mem
    also could cause mem leaks by not deleting old var
Implementation changes
    add capacity var to member vars
add capacity var to member vars
change array var to a pointer

6.4 Linked Lists
use pointers to connect elements
arrays have implicit order
linked lists have explicit order
list nodes need to store data & point to the next element
create node as separate class
needs functions to retrieve/set data & retrieve/set pointer
list is a collection of nodes & operations on the nodes
needs a pointer the 1st (head) node
consider list w/ only head ptr now
list variants add other pointers
list operations
create empty list - set head to NULL
is empty? - does head equal NULL?
traversal - from head node, follow pointer to next element
repeat until pointer to next is NULL
pseudocode
set ptr to head
while ptr is not NULL
  do traversal operation
  set ptr to ptr's next node
insertion - add new node to list
several cases depending on where adding
head insert / 1st node insert
new node will become head
pseudocode
  set new's next to head
  set head to new
must set next before changing head pointer
  otherwise lose reference to old list
tail insert
new node will become end of list
pseudocode
  traverse list to find current tail
  set tail's next to new
  set new's next to NULL
mid-list insert
insert after some specific node
pseudocode
  traverse list to find previous node
  set new's next to prev's next
  set prev's next to new
can avoid traversal if pass ptr to insert
pseudocode
  if ptr is NULL, do head insert
  else, do tail/mid-list insert w/ ptr as tail/previous
traversal still has to be done somewhere for mid-list/tail
insert
to make insertion of 1st element or at tail easier, have list
node initialize next to NULL
deletion - remove node from list
must update pointers to reflect new order
two cases
head delete
  remove 1st element
  2nd element becomes new head
Pseudocode
  create tmp ptr that points to head
  set head to head's next
  deallocate tmp
mid-list & tail delete
  need to find nod before one being deleted
  previous node will "skip over" deleted node
pseudocode
  traverse list to find previous node
  set prev's next to node's next
  deallocate node

6.5 Linked List Implementation
NOTE: This differs from the book's implementation
Node class
  member vars
    an element (template type)
    a pointer to the next node
  member functions
    default constructor - sets next to NULL
    a constructor that takes an element & sets next to NULL
    a constructor that takes an element & node pointer
    setData to set the element
    getData to retrieve element
    setNext to set next pointer
    getNext to retrieve next pointer
    equality operator (for list search)
    output operator (for list output)
Linked List Class
  member vars
    a node pointer for head
  member functions
    default constructor - sets head to NULL
    destructor - deallocate list nodes
    copy constructor - create 2nd list that stores same elements
      has separate pointers & memory space
    assignment operator - also create copy
    bool empty() - check if head is NULL
    output operator - print list contents
    Node *search (T elem) - search list for element
      return pointer of node if found
      return NULL if not found
    Node *find_previous (Node *ptr)
      find the node before given node
      can be private helper function for delete
    void insert (T elem, Node *prev)
      create new node to store elem
      insert at head if prev is NULL
      otherwise insert after prev
    void delete (Node *node) - remove node from list
void delete (T elem) - alt form of delete
traverse list to find elem's node
call node delete function