Data Abstraction

C&P, Chapter 3

Wednesday, February 5, 2003

MAT 102 - Data Structures
Abstract Data Types

• A collection of data together with a set of operations on that data is called an abstract data type (ADT).

• Each operation should be specified as a "contract" -- if you use the operation in this way, this is exactly what it will do for you.

• An ADT should be independent of any particular programming language.
Example: The ADT List

Data:

- An ordered collection of items of type ListItemType

Operations:

- Create a List
- Destroy a List
- Get the length of list
- Check to see if the list is empty
- Add an item to the list
- Remove an item from the list
- Retrieve an item from the list
Pseudocode for the ADT List operations

+createList() // Creates an empty list.

+isEmpty():boolean // Determines whether a list is empty.

+getLength():integer // Returns the number of items in the list.

+destroyList() // Destroys a list.

ListItemType is the type of the items stored in the list.

Pseudocode for the ADT List operations
`insert(index:integer, in newItem:ListItemType, out success:boolean)` // Inserts newItem at position index of a list that was at
// index+1 is now at position index and the item that
// items are renumbered so that the item that was at
// index+2 is now at index+1, etc.  The success
// flag indicates whether the insertion was successful.

// is successful or not.

The success flag indicates whether the insertion
the item that was at index+1 is now at index+2, etc.
that the item that was at index+1 is now at index+2,
items are renumbered so
if index > getlength() { // if i > i++;

removeAt(index:integer, out success:boolean)
retrieve(in index:integer, out dataItem:ListItemType, out sucess:boolean) {query}// Copies the item at position index of a list into dataItem, where 1 <= index <= getLength(). The list is left unchanged. The success flag indicates whether the retrieval is successful.

{query} (query) out success:boolean, out dataItem: ListItemType, index: integer

+retrieve( in index: integer, out dataItem: ListItemType, out sucess:boolean) {query};
We could use the following operations on our ADT:

\[
\begin{align*}
\text{alist} &: \text{createList()} \\
\text{alist} &: \text{insert}(4, \text{apples}, \text{success}) \\
\text{alist} &: \text{insert}(3, \text{butter}, \text{success}) \\
\text{alist} &: \text{insert}(2, \text{eggs}, \text{success}) \\
\text{alist} &: \text{insert}(1, \text{milk}, \text{success}) \\
\text{alist} &: \text{createList()} \\
\end{align*}
\]

Suppose we want to create a list with the items:

- milk
- eggs
- butter
- apples
If we then perform the operation, `aList.remove(3, success)` the list will become: milk, bagels, eggs, apples.

If we next perform the operation, `aList.insert(2, bagels)` the list will become: milk, bagels, eggs, apples.

If we then perform the operation, `aList.remove(3, success)` the list will become: milk, eggs, apples.
Implementing an ADT in C++

We can use classes to implement ADTs in C++.

As an example, consider the implementation of the ADT List using an array.

Homework (Exercise 7)

- Continue reading Chapter 3 of C&P.
- Work #8, p. 158, from C&P. Note that you should not be writing C++ code, rather, you should work everything out by hand using the ADT operations given in the problem.