

CSE 143

List (Vector) Implementation [Chapter 3]

06/28/01 H-1

Textbook example: List ADT

- A list... names, groceries, numbers, etc.
- What do you need to do?
 - Create and destroy a list
 - Find out how long it is
 - Add (insert) new items to it
 - Delete items
 - Look at (retrieve) items
- ADT: specify the "what" without giving away the "how"
- Build a solid wall around the object
 - The defined operations are the only ways through the wall

06/28/01 H-2

Steps to Turn This Into C++

- Let's call it Vector
 - we'll allow indexing by position
 - textbook calls it listClass
 - 1. Identify and clarify the operations
 - by studying the application(s) that will use the class
 - 2. Map the ADT operations to public class methods
 - 3. Decide on the data representation
 - internal variables and their structure
 - 4. Implement the methods in a .cpp file
- Why don't we just tell the client to use an array, by the way?*

06/28/01 H-3

Vector ADT Operations

- Original analysis of a list suggests these abstract operations:
 - CreateVector()
 - DestroyVector()
 - VectorIsEmpty()
 - VectorLength()
 - VectorInsert(NewPosition, NewItem)
 - VectorDelete(Position)
 - VectorRetrieve(Position)
- Question: what is a "position"?
 - integer index for Vector (usually beginning/end for list)

06/28/01 H-4

Map To Class Methods

- Make some adjustments and turn these into public methods
 - CreateVector() //use a constructor for this
 - DestroyVector() //use a "destructor"
(not currently needed)
 - VectorIsEmpty() //return a bool
 - VectorLength() //return an int
 - VectorInsert(NewPosition, NewItem)
//need to clarify the argument types, especially NewItem
 - VectorDelete(Position) // return the item deleted
 - VectorRetrieve(Position) // return the item retrieved

06/28/01 H-5

Public Member Functions

```
class Vector {
public:
    // construct empty vector
    Vector ( );
    // = "this Vector is empty"
    bool  isEmpty ( );
    // = # of items in this Vector
    int   length ( );
    ...
}
```

06/28/01 H-6

Public Member Functions

```
...
// Insert newItem in this Vector at newPosition
void vectorInsert (int newPosition, Item newItem);
// Delete item at specified position and return a copy of it
Item vectorDelete (int position);
// Return a copy of the item at the specified position
Item vectorRetrieve (int position);
...
}
```

06/28/01 H-7

Decide on Data Representation

- "Data representation"
 - Choose variables, data structures appropriate
 - Usually are many possible choices
 - We'll learn more and more useful data structures
- Issue for the vector application
 - need to store multiple list items
 - need some notion of "position"
 - need way to report how many items are in the list

06/28/01 H-8

Decide on Private Data

- How about: keeping the vector as a private array?
 - Items are packed in the array
 - Array indexes correspond to "positions"
 - Internal variable keeps track of number of items stored
- Complications to watch for
 - not all positions are valid
 - inserting/deletion requires shifting items

06/28/01 H-9

Declaring the Data

```
class Vector {
public:
    // constructors and other methods
    ...
private:
    Item items[MAX_ELEMENTS]; // Vector contents are in
                             // items[0..size-1]
    ...
}
• May have to declare some const values
```

06/28/01 H-10

Last Step: Implementing the Methods

- In the .cpp file:

```
Vector::Vector ( ) { ... }
bool Vector::isEmpty ( ) { ... }
etc. etc.
```
- insert and delete will have the trickiest programming
- See textbook 136-139 for full details

06/28/01 H-11

Vector Constructor

```
class Vector {
public:
    Vector ( );
private:
    Item items[MAX_ELEMENTS]; // ...
    int size;                 // ...
};

Vector::Vector ( ) {
    size = 0;
    do we need to initialize items?
}
```

06/28/01 H-12

Exercise

- Write client code which prints out all the elements in the vector

06/28/01 H-13

Vector Equality

- Remember, == is not defined on two classes instances
Let's define an *equals* function to compare two vectors

```
bool Vector::equals(Vector other) {  
    if (size != other.size)  
        return false;  
    for (int i=0; i < size; ++i) {  
        if (items[i] != other.items[i])  
            return false;  
    }  
    return true;  
}
```

Footnote: this implementation assumes the items can be compared with the != operator. What if that's not true??

06/28/01 H-14

Vectors: Above and Beyond

- In many real-world lists, the items need to be kept in order.
 - Appointments: in chronological order (date and time)
 - Students: by ID or by name
 - Books: by ISBN, Title, author, subject, etc.
- One approach: Sort the list when needed
- Another approach
 - *Keep the list sorted, as part of its invariant*
 - Consider a new ADT, "SortedVector" with this property
very similar to original vector ADT
see textbook p.118

06/28/01 H-15

One Class May Suggest Another

- Vector -> SortedVector
 - Would be nice to reuse code somehow (more later)
- Items inside one class may themselves represent an ADT
 - Example: a BookVector (Bookshelf) might require a Book class
Maybe author, publisher, etc. as well
 - Some of the additional classes might be visible to client, some might not be

06/28/01 H-16

Collection ADTs

- Vectors are an example of a "collection" ADT: something which holds multiple instances of entities of interest.
- Arrays can be thought of as a primitive collection ADT.
- Later we'll see Stacks, Queues, Trees, and other collection ADTs
- We'll also see more and more advanced programming techniques for implementing them.
 - What's wrong with what we have??

06/28/01 H-17