

CSE 143

Vector ADT as Linked List

[Chapter 4 p.170]

11/6/00 M-1

Linked List vs Vector ADT

- A linked list is a data structure
 - A programming technique for organizing data
- Earlier we defined a Vector ADT
 - Data encapsulated inside the class
 - Operations available only through the public interface
- Original implementation of Vector used arrays
 - Numerous drawbacks already pointed out
- Let's reimplement Vector using linked lists!
 - Will show a Vector of ints; Vectors of other types would be similar

11/6/00 M-2

Internal Data

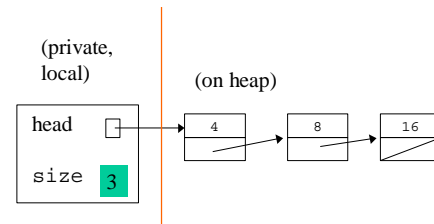
```
• Declare a struct to represent a node:
struct Node {
    int data;
    Node *next;
};

class Vector {
public:
    ...
private:
    int size; //number of items in the Vector
    Node *head; //ptr to linked list of items
...}
```

11/6/00 M-3

Portrait of a Vector

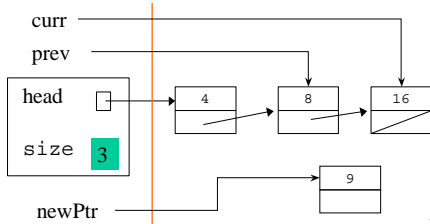
- Now a Vector variable might look like this:



11/6/00 M-4

Inserting at position X

- We have to find node X
 - Better yet, get a pointer to X (curr) and X-1 (prev)
- Example: X = 2



11/6/00 M-5

Finding Position X

- Write a function "PtrTo" to traverse the list, return a pointer to the Xth element (code: p.175)
- Should be a member function
 - But not part of the interface, so should be private
- Special cases: X outside the range of the list
 - return NULL
- Given this, curr and prev are easy to get:
 - curr = PtrTo(X); prev = PtrTo(X-1)
- Better yet:
 - prev = PtrTo(X-1); curr = prev->next;

11/6/00 M-6

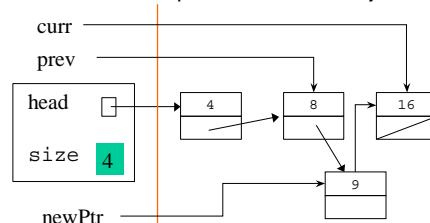
Relinking for Insert (p.176)

- Given prev and curr (via PtrTo function):
 - `newPtr->next = curr;`
 - `prev->next = newPtr;`
- Inserting at beginning is a special case (X=0)
 - `newPtr->next = head;`
 - `head = newPtr;`
- What about inserting at end of list?
 - How to recognize?
 - Is the code special?

11/6/00 M-7

Final Picture

- curr, prev, and newPtr are local variables that go away
- head and size persist inside the object



11/6/00 M-8

ListDelete

- Similar considerations
- PtrTo is helpful again
- The deleted node should have *delete* operator applied
 - or memory leak results
- Deleting from beginning of list a special case
 - changes head value
- Full code: textbook p. 177

11/6/00 M-9

Variations on a theme

- Doubly linked lists
 - Point backwards as well as forwards
 - Makes finding the previous pointer a breeze
 - Takes a little more space and complexity to manage the extra pointers
- Circular lists
 - Can remove some special cases
- Head and tail pointers.
 - Good for "queues" (always add at tail, always remove at head)
- Dummy nodes at front or rear
 - Can remove some special cases

11/6/00 M-10