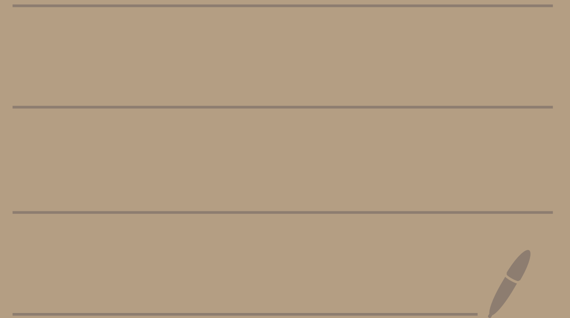
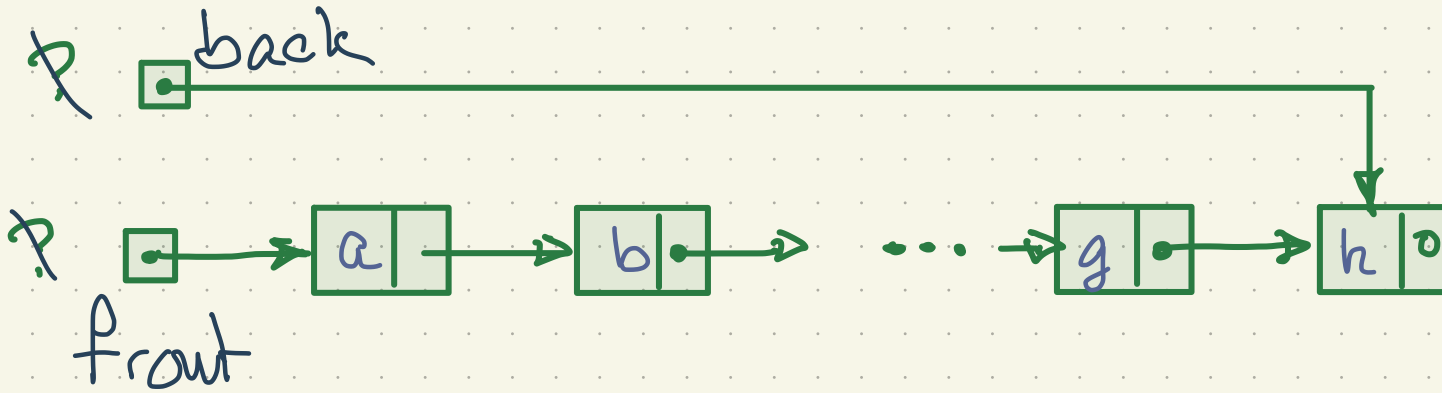
L.L. Queues & L.L. Traversal



Basic L.L. Queue

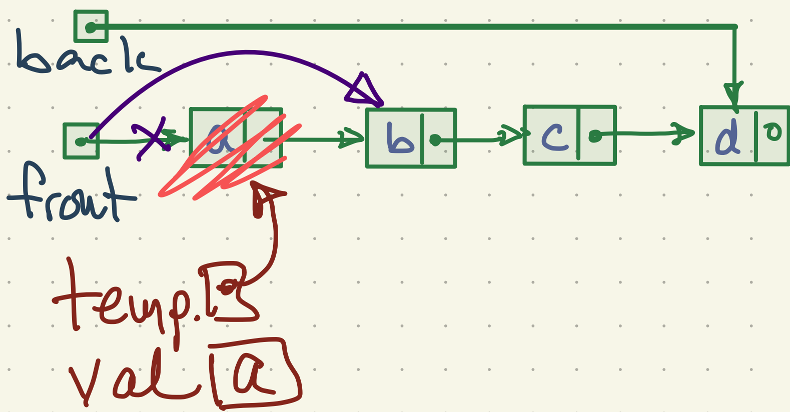


class Node {
  Type data
  Node * next
}

· as a stack, this L.L. would be:

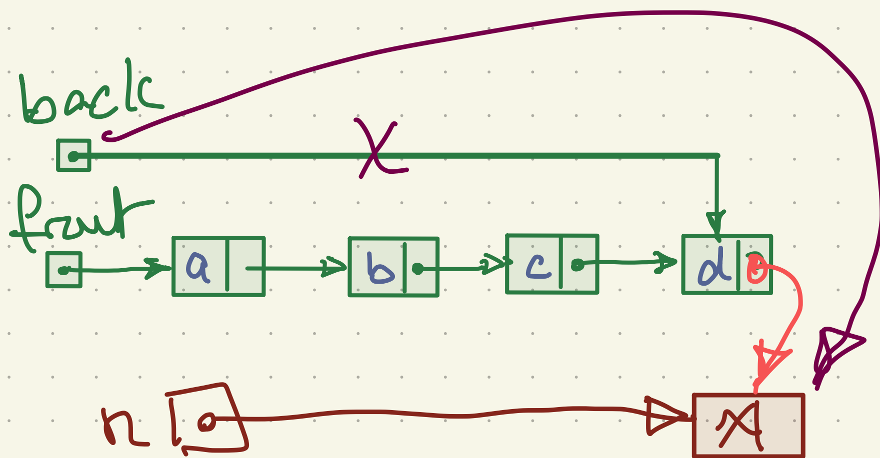
a
b
⋮
h

Enqueue + Deque – First Versions



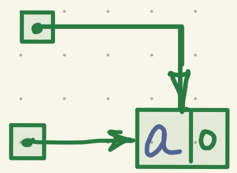
Variables: front } pointers to nodes.
back }
size : # elements in queue

```
dequeue(){
    temp = front
    val = front -> data
    front = front -> next
    delete = temp
    size = size - 1
    return val
}
```



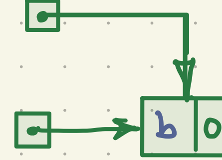

```
enqueue(x){
    n = new node containg x
    back -> next = n
    back = back -> next
    size = size + 1.
}
```

Empty Queue



dequeue ⟹

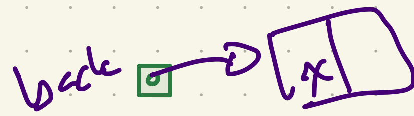
enque b ⟹

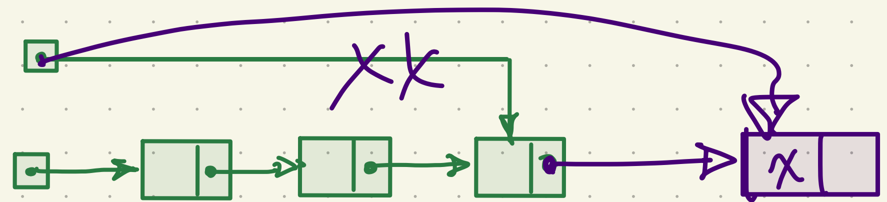


$\langle a \rangle$

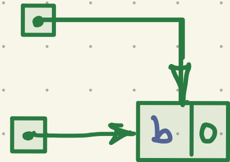
- Enqueue & Dequeue are different for empty/non-empty queues.

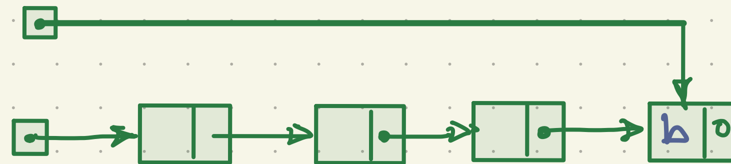
Enqueue



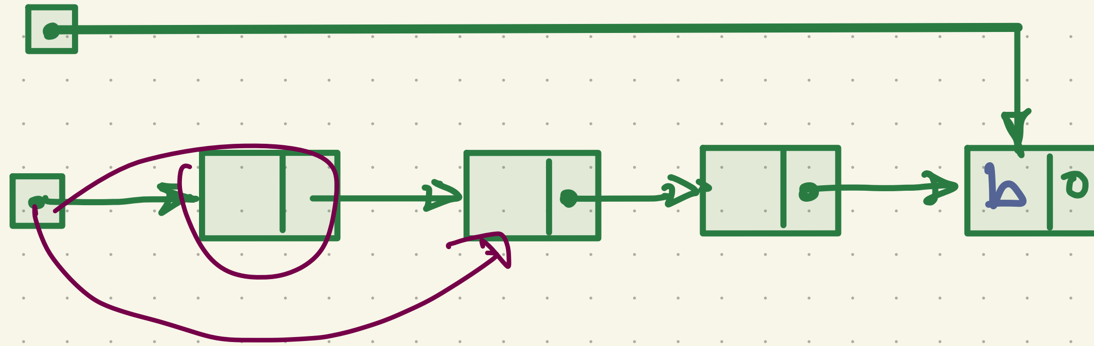


enque b:



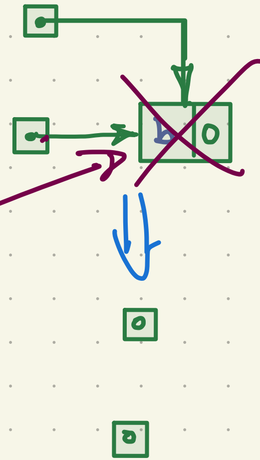


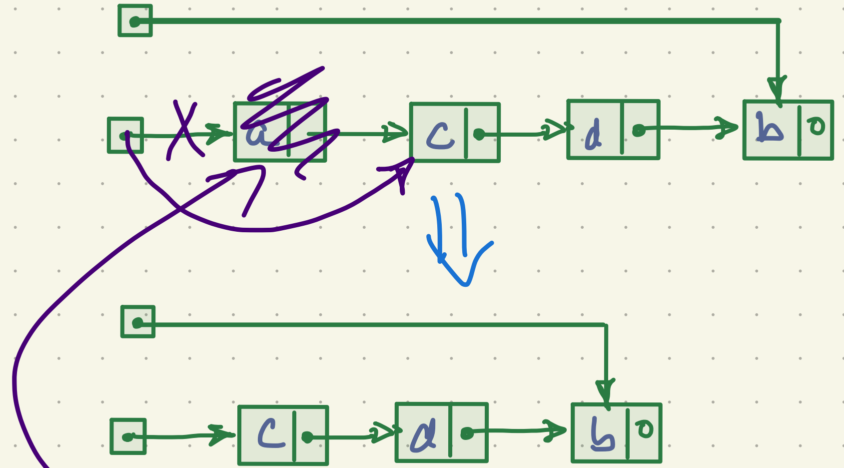

```
enqueue(x){
  if size > 0 {
    • back -> next = new Node(x)
      back = back -> next
  }else{
    back = new Node(x)
    front = back
  }
  size = size + 1
}
```



front = front → next
return value.

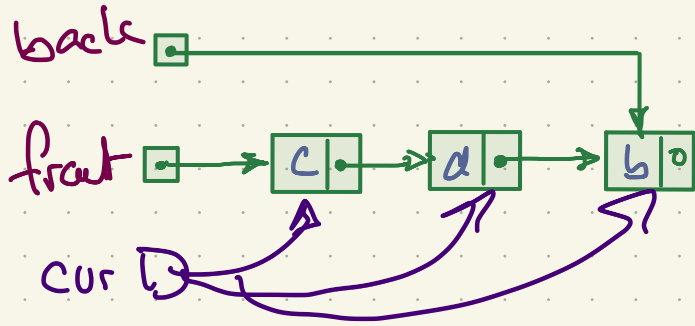
Dequeue

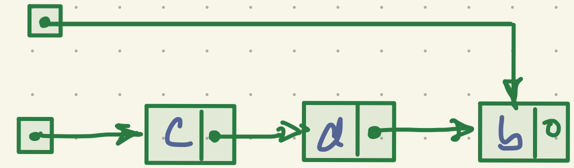





```
dequeue(){
  • temp  = front
  • val   = front -> data
  • front = front -> next
    delete = temp
    size = size - 1
    return val
}
```

Traversing the list

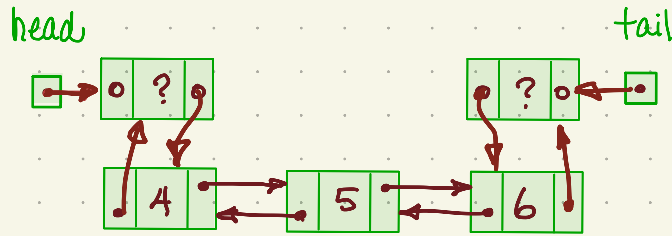




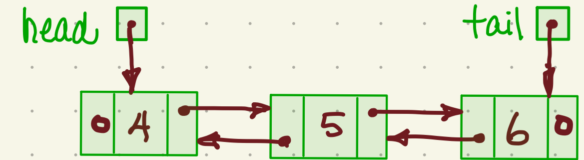

```
displayList(){
    Node * cur = front ;
    while (cur != nullptr ) {
        output cur -> data
        cur = cur -> next
    }
```

Linked List Ends

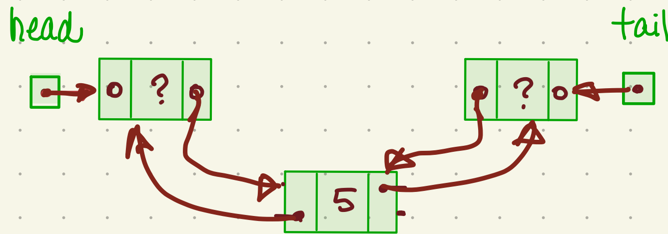
⟨4,5,6⟩:



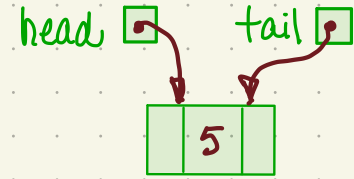
VS.



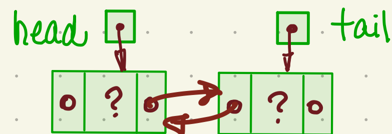
⟨5⟩ :



VS.

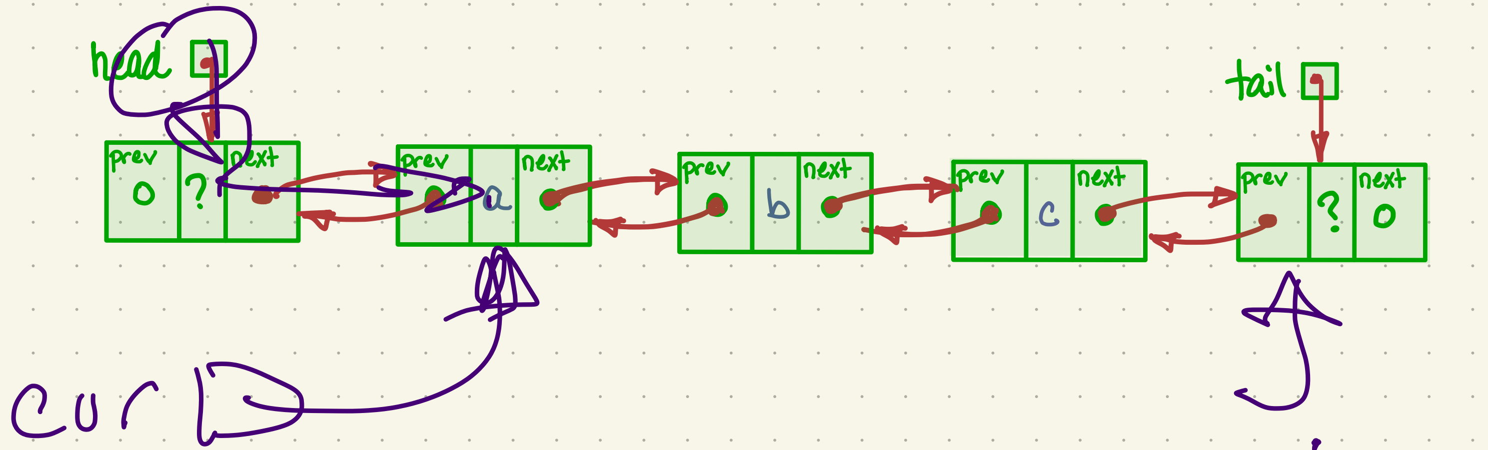


⟨⟩ :



VS.

List Class: Traversing the List




```
displayList(){
    Node * cur = front;        // "front" crossed out -> head -> next
    while (cur != nullptr) {   // "nullptr" crossed out -> tail
        output cur -> data
        cur = cur -> next
    }
```

End