ALSO NOTICE THAT MANY OF THE SAME OPERATIONS ARE PERFORMED REPEATEDLY WHEN NAVIGATING A LIST. THIS SUGGESTS THAT A LINKED LIST ADT SHOULD PROVIDE METHODS WHICH AUTOMATE THESE HIGHER LEVEL OPERATIONS.

Ex: in some class that uses Node

```
Public void insertFirst(int x ){
    Node N = new Node( x );
     N.next = head;
       head = N;
}
```

NOTICE THE ABOVE METHOD AUTOMATICALLY DOES THE RIGHT THING WHEN THE LIST IS EMPTY (i.e. head == null.) THE FOLLOWING METHOD MUST TREAT THIS AS A SPECIAL CASE.

```
Public void insertLast(int x ){
    if (head == null){
        head = new Node( x );
    }else{
        Node N = head;
        for( ; N.next != null; N = N.next);
        N.next = new Node(item);
    }
}
```

IT is VERY NATURAL TO PROCESS THE ITEMS IN A list IN THE FORWARD DIRECTION:

EX. IN SOME class OTHER THAN Node:

```
Public void printForward(Node head) {
    for(Node N=head; N!=null; N=N.next ){
        System.out.Println(N.next+" " );
    }
}
```

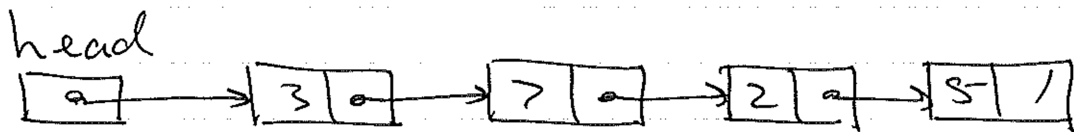
IT WOULD SEEM THAT PROCESSING THE list ITEMS IN BACKWARDS ORDER IS IMPOSSIBLE. IN FACT IT IS POSSIBLE BY USING RECURSION.

EX IN SOME class OTHER THAN Node;

```
Public void printBackward(Node head){
    if ( head != null) {
        PrintBackward(head.next );
        System.out.Println(head.item );
    }
}
```

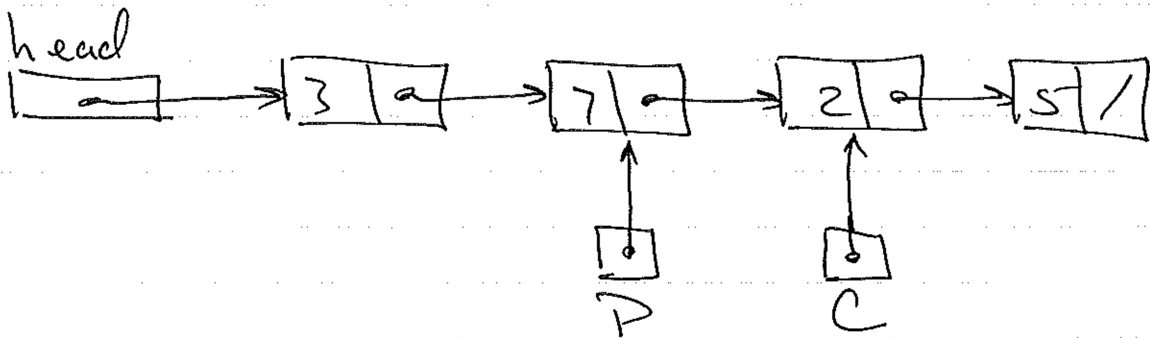
THE Println() OPERATION IN THE LAST TWO EXAMPLES CAN BE REPLACED BY ANY OTHER "PROCESS" TO BE PERFORMED ON THE list IN SOME ORDER.

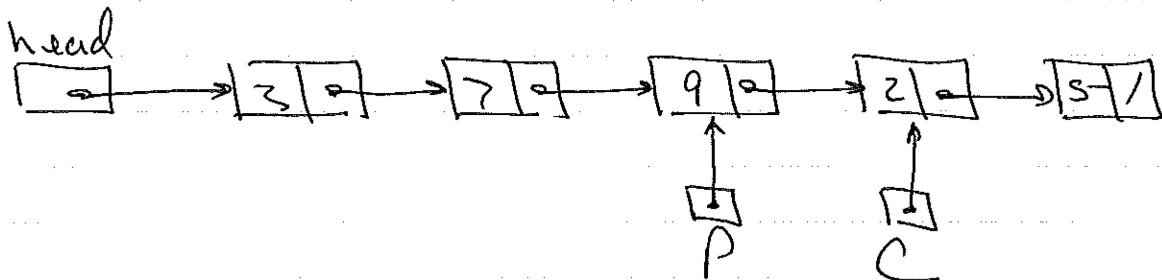
NOW SUPPOSE THAT GIVEN THE FOLLOWING PICTURE



WE WISH TO INSERT THE ITEM 9 BETWEEN 7 AND 2.

```
Node P=null;
Node C=head;
P=C; C=C.next;
P=C; C=C.next;
P.next = new Node(9);
P=P.next;
P.next=C;
```

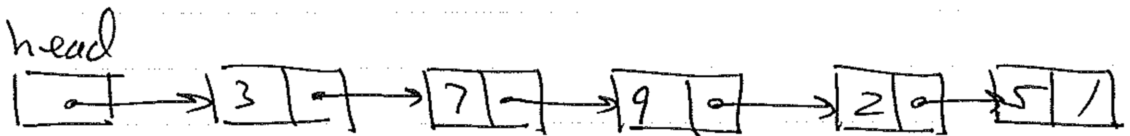




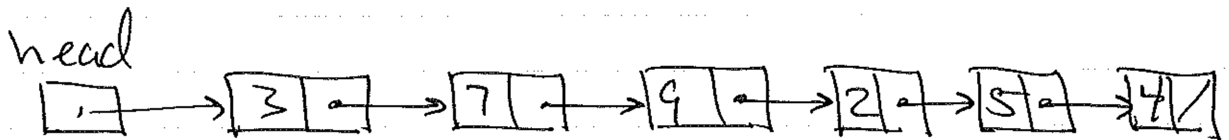
THIS PROCESS CAN BE GENERALIZED TO INSERT A NEW ITEM INTO ANY LIST POSITION.

```
// insert: FIRST DRAFT
Public void insert (int index, int item) {
    Node P=null;
    Node C = head;
    for (int i=1; i<index; i++) {
        P=C;
        C=C.next;
    }
    P.next = new Node (item);
    P = P.next;
    P.next = C;
}
```

THE CALL insert(3, 9) ON THE PREVIOUS STATE RESULTS IN:



CALLING insert(6, 4) ON THIS STATE YIELDS:



UNFORTUNATELY THIS method will FAIL (WITH NullPointerException) IF CALLED ON AN EMPTY LIST OR WITH index=1. (WHY?)

THE FOLLOWING METHOD TREATS BOTH SITUATIONS AS A SINGLE SPECIAL CASE:

```
// insert: THIS ONE WORKS
Public void insert (int index, int item){
    if (index == 1) {
        Node N = new Node (item);
        N.next = head;
        head = N;
    } else {
        Node P = null;
        Node C = head;
        for (int i = 1; i < index; i++) {
            P = C;
            C = C.next;
        }
        P.next = new Node(item);
        P = P.next;
        P.next = C;
    }
}
```

EXERCISE: WRITE A METHOD

Public void delete (int index)

WHICH DELETES THE item AT POSITION index IN THE LIST.

THE PRECEDING EXAMPLE COULD BE IMPROVED BY CREATING A HELPER FUNCTION TO FIND A SPECIFIED NODE IN THE LIST.

```
Public Node find (int index) {

    Node N = head;
    for (int i=1; i<index; i++) {
        N = N.next;
    }
    return N;
}
```

THEN insert Can BE REWRITTEN

```
// insert: FINAL DRAFT
Public void insert (int index, int item) {
    If (index == 1) {
        Node N = new Node (item);
        N.next = head;
        head = N;
    } else {
        Node P = find (index - 1) //index≥2 HERE
        Node C = P.next
        P.next = new Node (item);
        P = P.next
        P.next = C;
    }
}
```