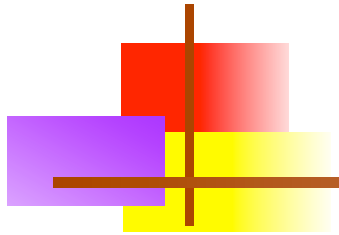
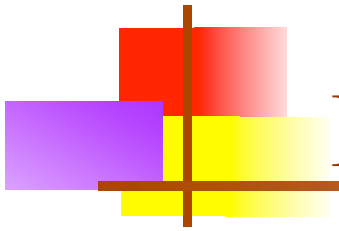


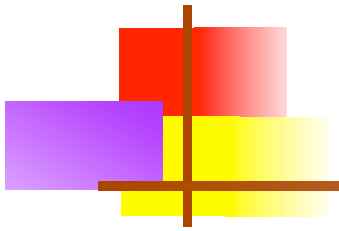
LINKED LISTS

Lecture 10



Recap

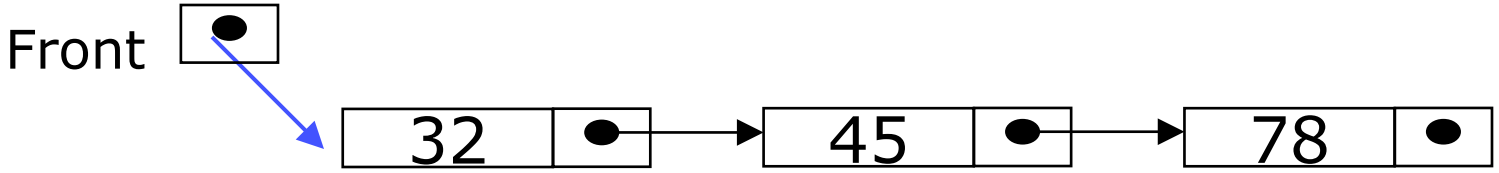
- Creating a node of the linked list
- Adding the first node of the list
- Insertion – front, end
- Deletion – front, end
- Linked list traversal
  - Insertion at a specific place
  - Deletion of a specific node
  - Counting the nodes
  - Printing the list
- Recursion and Linked lists



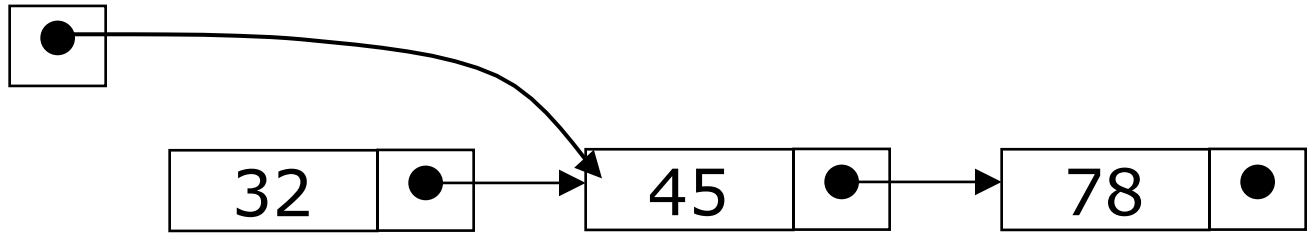
Deletion in Linked Lists

Deleting the first node

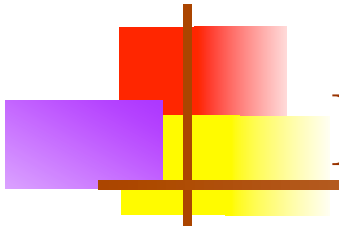
- To delete the first element, change the link in the header



-



- Node ptr = Front.getLink();
  Front = ptr;



Deleting the last node

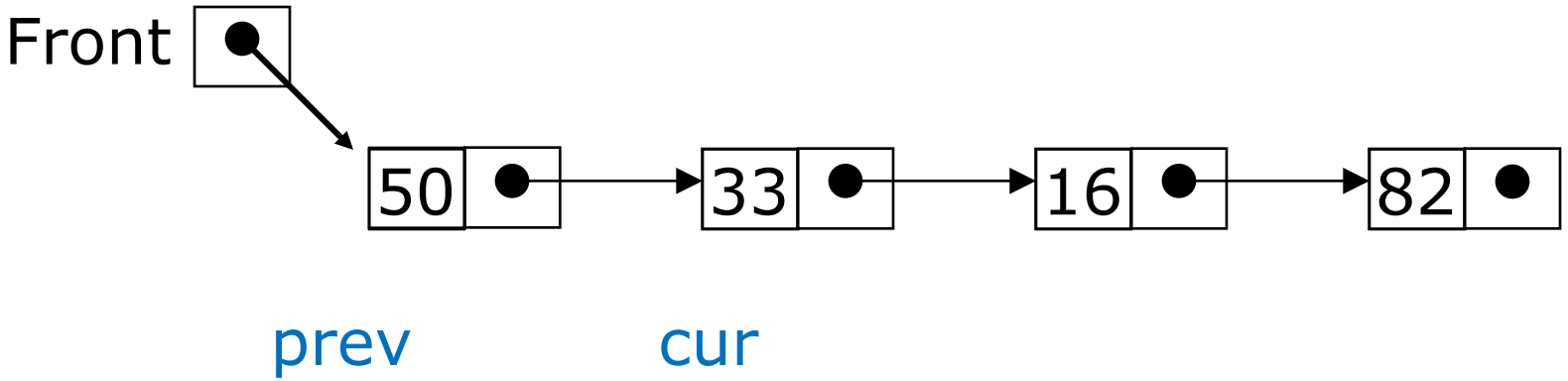
- Set pointer prev (previous node) to first node of the list.
- Set pointer cur (current node) to second node of the list.
- Traverse the list till cur reaches the last node
- set link for prev to null, so that last node is not reachable.

```
Node prev = Front;
Node cur = Front.getLink();
while ( cur.getLink() != null) {
        prev = cur;
        cur = cur.getLink();
 }
 prev.setLink(null);
```

Add code to handle single node in the list.

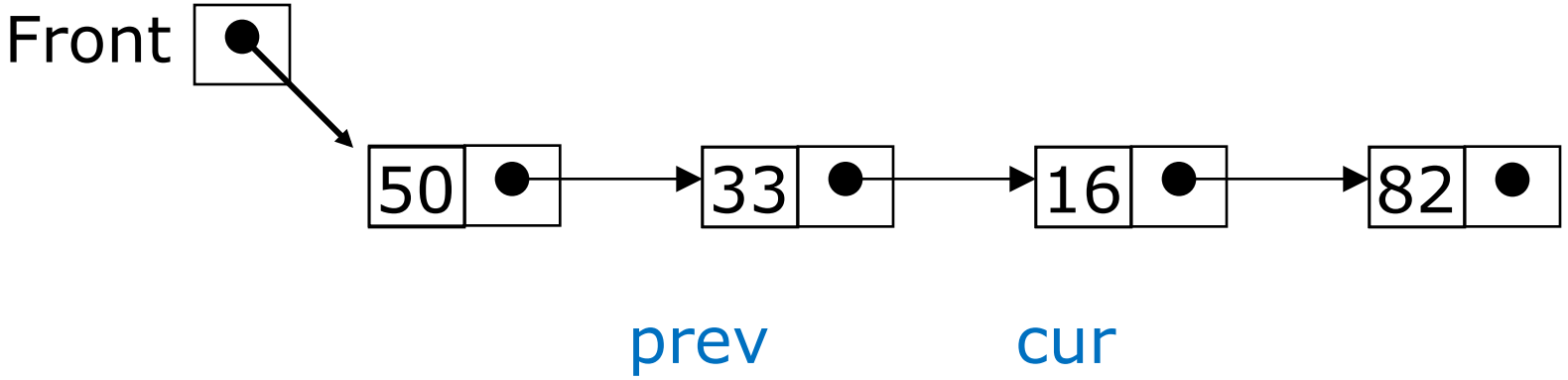
Traversing list with 2 pointers

- :



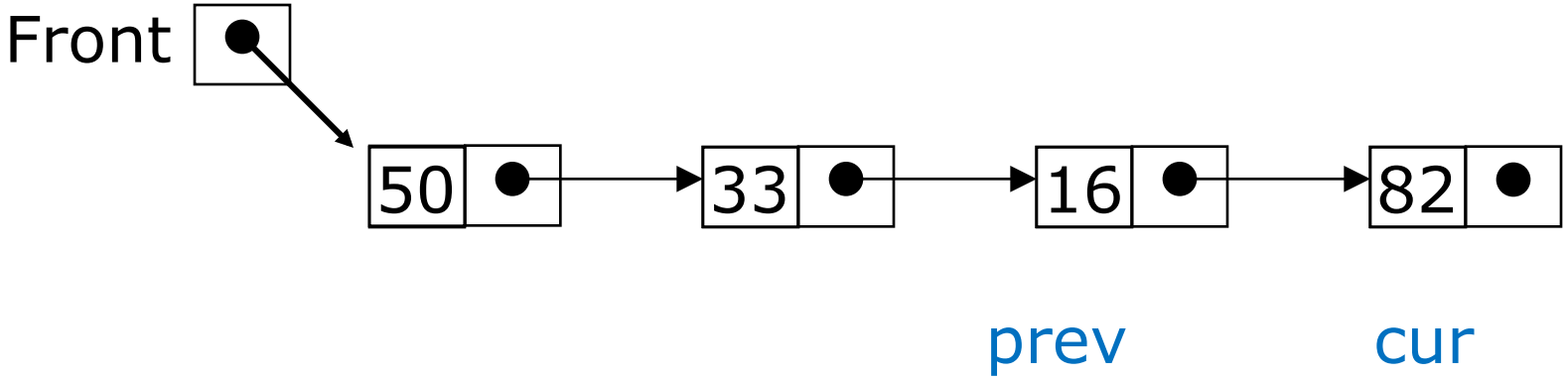
Traversing list with 2 pointers

- :



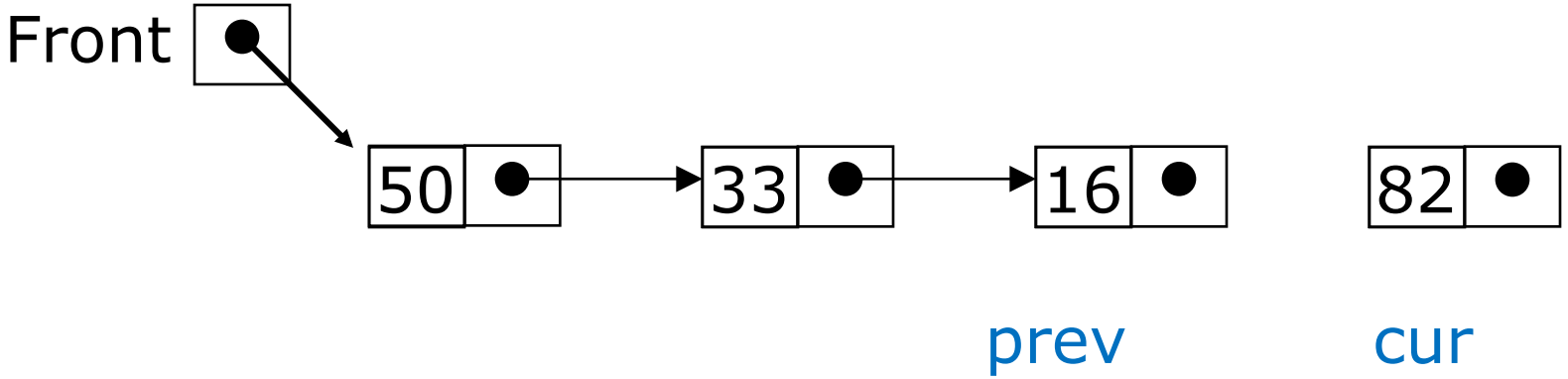
Traversing list with 2 pointers

- now cur points to last node and prev points to last but one node



Traversing list with 2 pointers

- set link of last but one node to null, so that the last node is not reachable (gets deleted)



Deleting a node containing data d

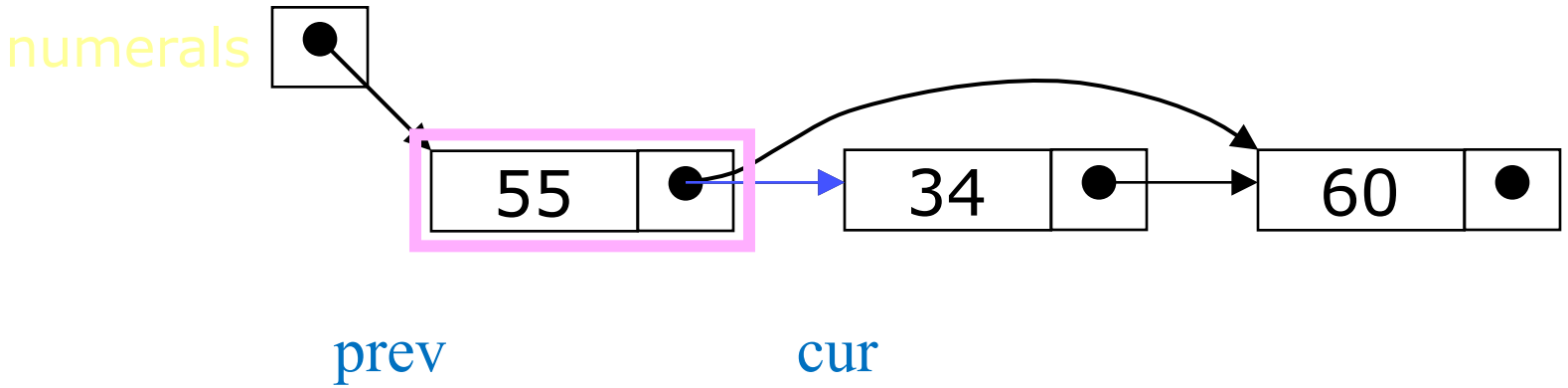
- Set pointer prev (previous node) to first node of the list.
- Set pointer cur (current node) to second node of the list.
- Traverse the list till cur reaches the node to be deleted.
- Set link for prev to next node of cur so that current node is not reachable.

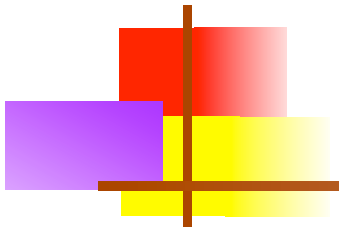
```
if (Front.getData == d) Front = Front.getLink();
else {
      Node prev = Front;
      Node cur = Front.getLink();
      while ( cur.getData() != d) {
prev = cur;
            cur = cur.getLink();
      }
      prev.setLink(cur.getLink());
 }
```

Deleting a specific element

```
prev.setLink(cur.getLink());
```

- To delete a node, change the link in its previous node

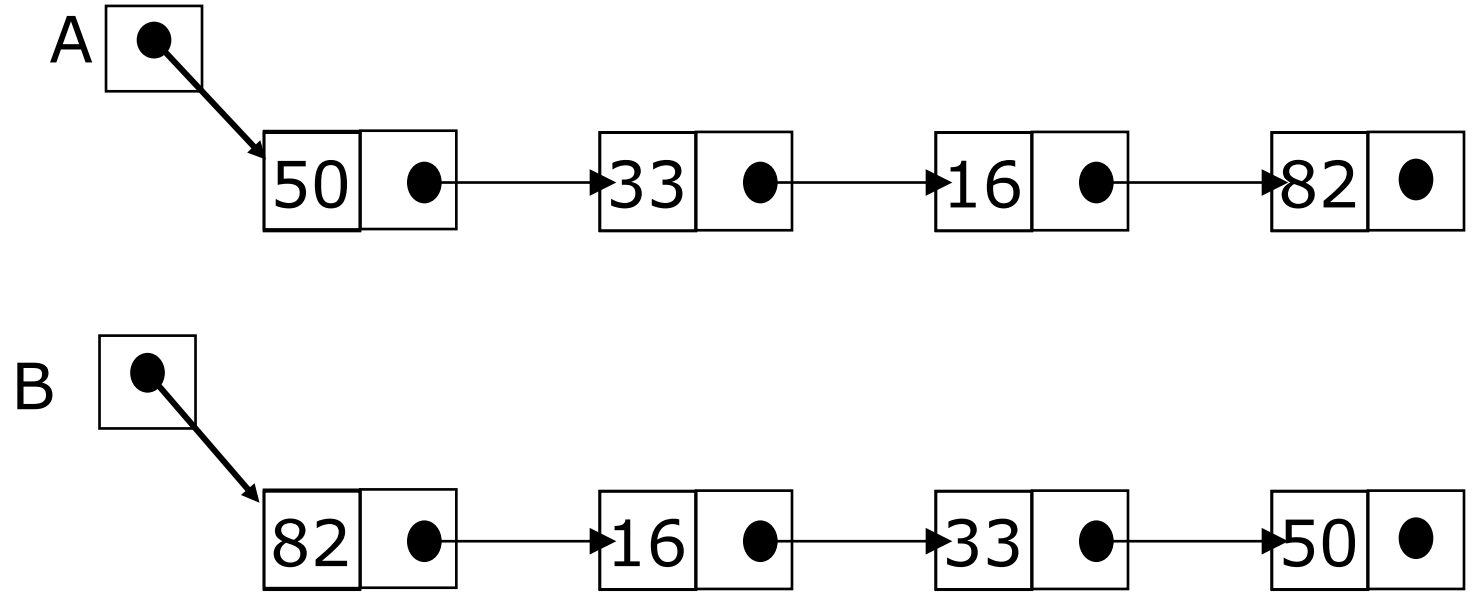


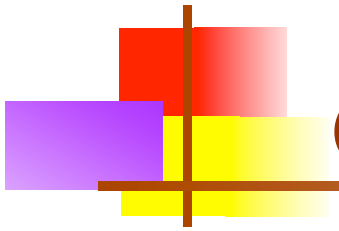


Reversing a Linked List

Creating another list with reversed contents

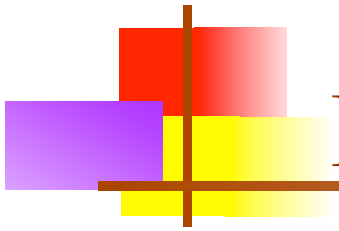
- Read contents from list A, and add them to front of list B.





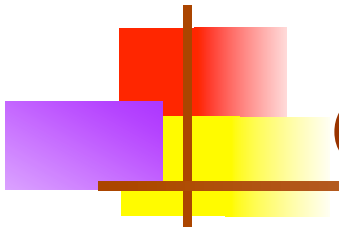
Create a list B which is reverse of list A

```
ptr = A;
B = null;
while (ptr  != null)
{
    int d = ptr.getData();
    Node nptr = new Node ( d, null);
    nptr.setLink(B);
    B = nptr;
    ptr = ptr.getLink();
 }
```



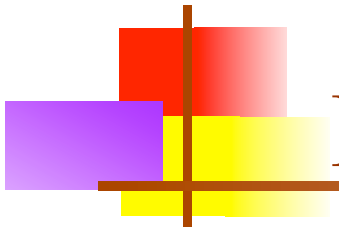
Reverse a Linked list in place

- You need to use 3 pointers: prev, cur, next
- Start with pointing cur to first node, and prev = null.
- Traverse the list in a while loop
  1. Set next to cur.link
  2. Set cur link to address of prev node
  3. Set prev to cur
  4. Set cur to next
- When loop is over, return prev node



Code for List Reversal

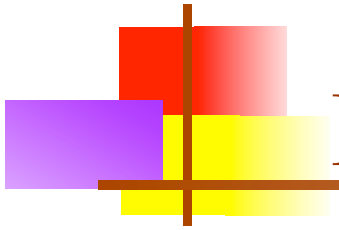
```
public static ReverseList (Node cur){
    Node prev = null;
    Node next = null;
    while (cur != null) {
        next = cur.getLink();
        cur.setLink(prev);
        prev = cur;
        cur = next;
    }
    return prev;
}
```



Recursive Function to Reverse a Linked List

- The base case is the empty list, or having a single node , when the Front (head) can be returned. The continuation case calls the function with node next to the head.

```
public static Node Reverse (Node head); {
    if (head == null || head.getLink == null)
        return head;
    Node next = head.getLink;
    head.getLink = null;
    Node rest = Reverse(next);
    next.setLink(head);
    return rest;
}
```



Exercises

- Create a list with each node containing name of student and his CGPA.
- Given the above list, find the name of the student having the highest CGPA.
- Given a list, create two lists with alternate elements of first list.
- Append a list at end of another list.
- Check if two lists are identical
- Given two lists, create a new lists taking corresponding elements from the two lists.