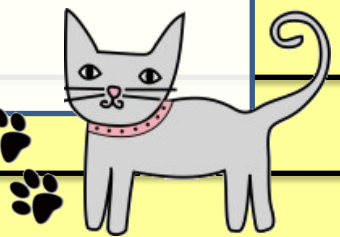
CMSC 21
Funadmentals of Programming

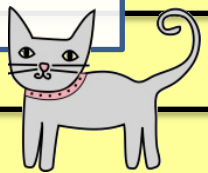
2nd Semester 2011-2012

Doubly Linked List

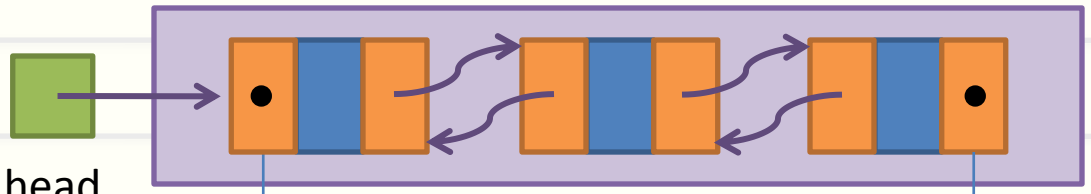


Doubly Linked List

- Consists of two pointers, one for the previous node and one for the next node
- Can be traversed either forward or backward
- The self-referential structure for a doubly linked list must have two pointers as fields
- The previous pointer of the first node is set to NULL
- The next pointer of the last node is set to NULL

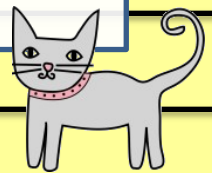


Doubly Linked List



The previous pointer of the first node is set to NULL

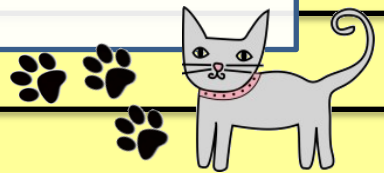
The next pointer of the last node is set to NULL



Doubly Linked List

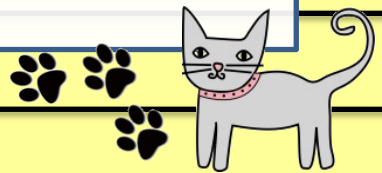
- The self-referential structure for doubly linked list is:

```
struct node {
  int x;
  struct node *next; //pointer to the next
                     //node
  struct node *prev; //pointer to the
                     //previous node
};
```



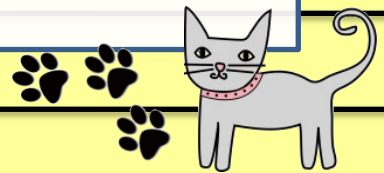
Operations on Doubly Linked List

- Insert
- Delete
- Search
- Output the contents of the list

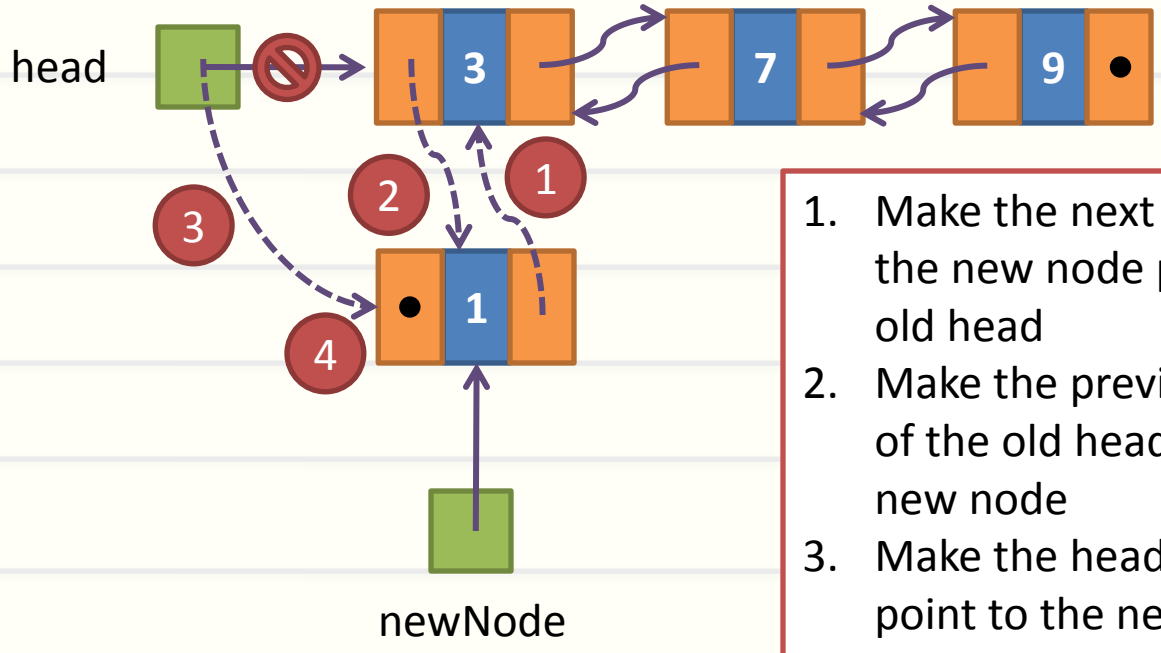


Insert

- Just like a singly linked list, we can insert nodes to a doubly linked list either at:
  - Head
  - Middle
  - Tail

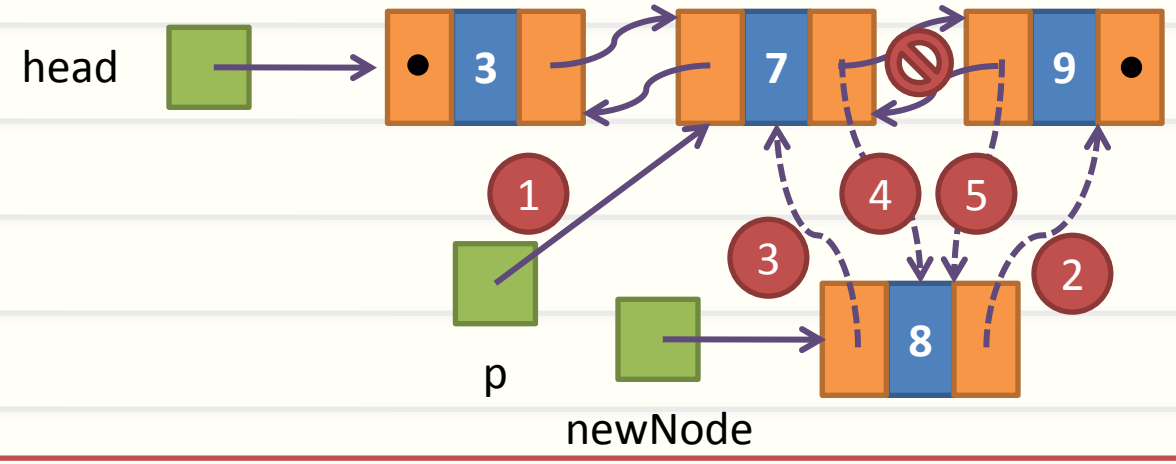


Insert at head



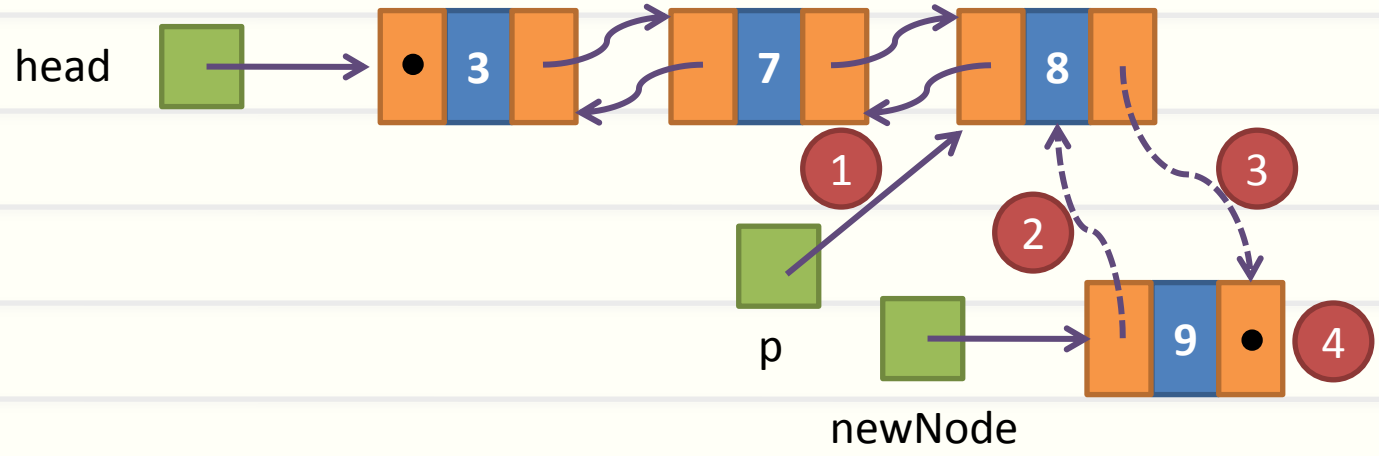
1. Make the next pointer of the new node point to the old head
2. Make the previous pointer of the old head point to the new node
3. Make the head pointer point to the new node
4. Make the previous pointer of the new node equal to NULL

Insert at the middle



1. Find the position where the node is to be inserted
2. Make the next pointer of the new node point to the node next to the one selected in step 1
3. Make the previous pointer of the new node point to the node selected in step 1
4. Make the next pointer of the node selected in step 1 point to the new node
5. Make the previous pointer of the node next to the one selected in step 1 refer to the new node

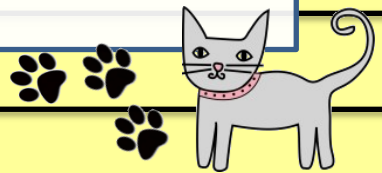
Insert at tail



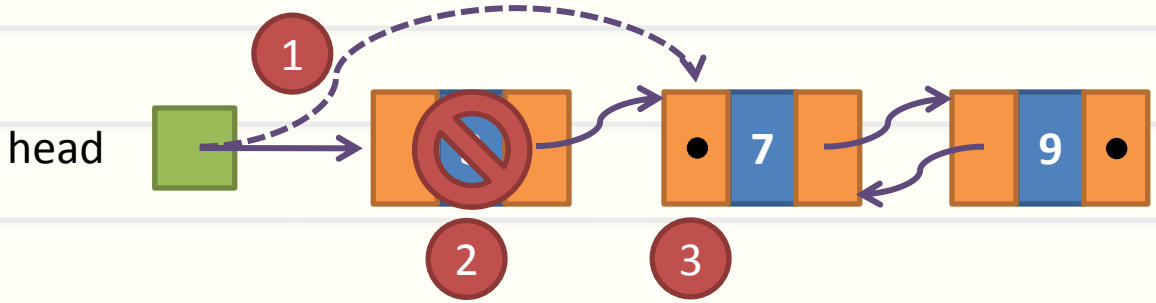
1. Check if the new node is to be inserted at the end of the list
2. Make the previous pointer of the new node point to the last node
3. Make the next pointer of the last node point to the new node
4. Make the next pointer of the new node NULL

Delete

- Delete at:
  - Head
  - Middle
  - Tail

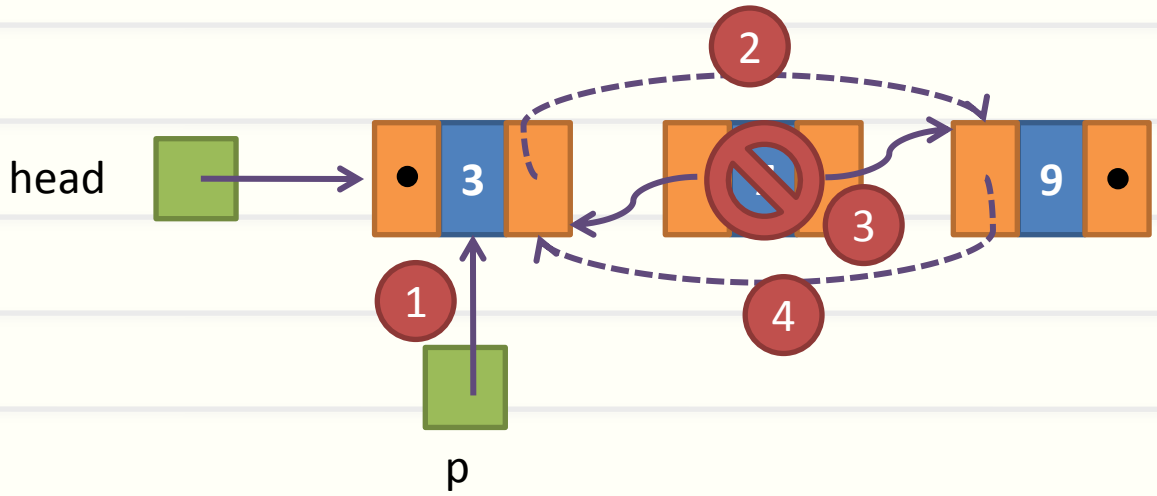


Delete at head



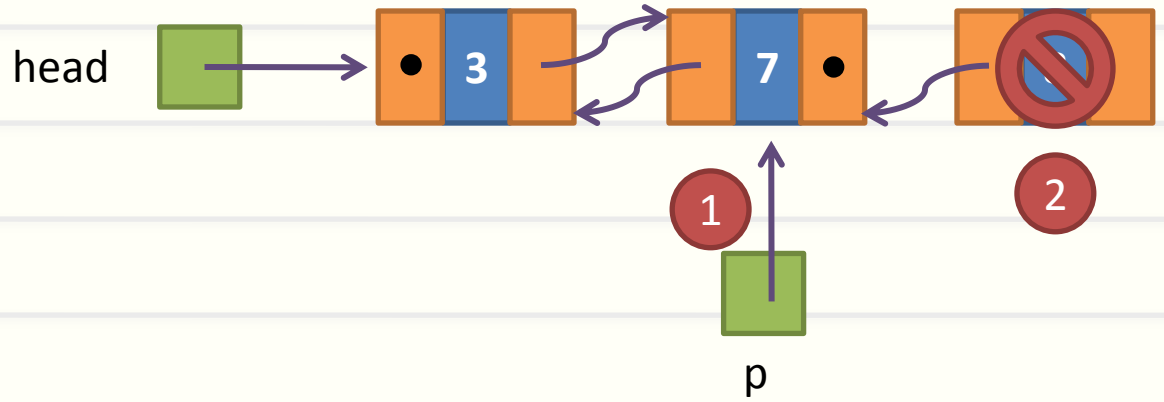
1. Make the head pointer refer to the second node (this becomes the new first node)
2. Free the first node
3. Make the previous pointer of the new first node NULL

Delete at middle



1. Find the position of the node to be deleted. Make a pointer refer to the node before the node to be deleted
2. Make the next pointer of the node selected in step 1 point to the node next to the node to be deleted
3. Free the node next to the node selected in step 1
4. Make the previous pointer point to the node selected in step 1

Delete at tail



1. Find the position of the node to be deleted. Have another pointer refer to the node before the node to be deleted. Check if the last node is to be deleted
2. Make the next pointer of the node selected in step 1 equal to NULL