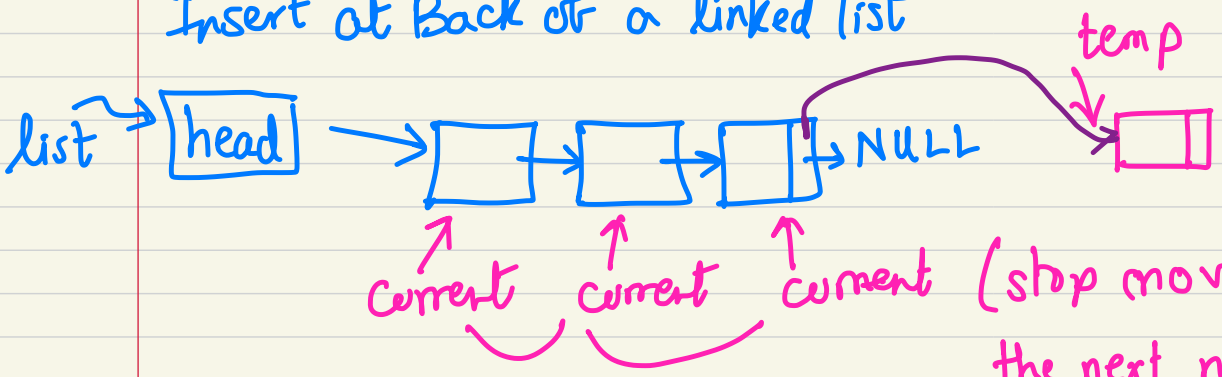
APS 105 Lecture 32 Notes

Recap: insertAtFront, findFirstNode, printList, insertAtBack functions on a linked list

Today: more on insertAtBack, insertIntoOrderedList, deleteFront, deleteAllNodes

Insert at Back of a linked list




```
bool insertAtBack(LinkedList *list, int data){
    Node * current = list -> head;
    while ( current -> next != NULL){
            current = current -> next;
    }
    → current pointing to last node
    current -> next = createNode (data);
    {if (current ->next != NULL) return true;
    else return false;
```

to know if we were capable of adding } a node

if list was empty

```
bool insertAtBack(LinkedList *list, int data){
    if (list->head == NULL){
        list->head = createNode(data);
        return (list->head != NULL);
    }
```

OR just `return insertAtFront(list, data);`

① if list is empty

② Work if there was only 1 node

```
    Node * current = list->head;
    while (current->next != NULL){
        current = current->next;
    }
```

→ current pointing to last node

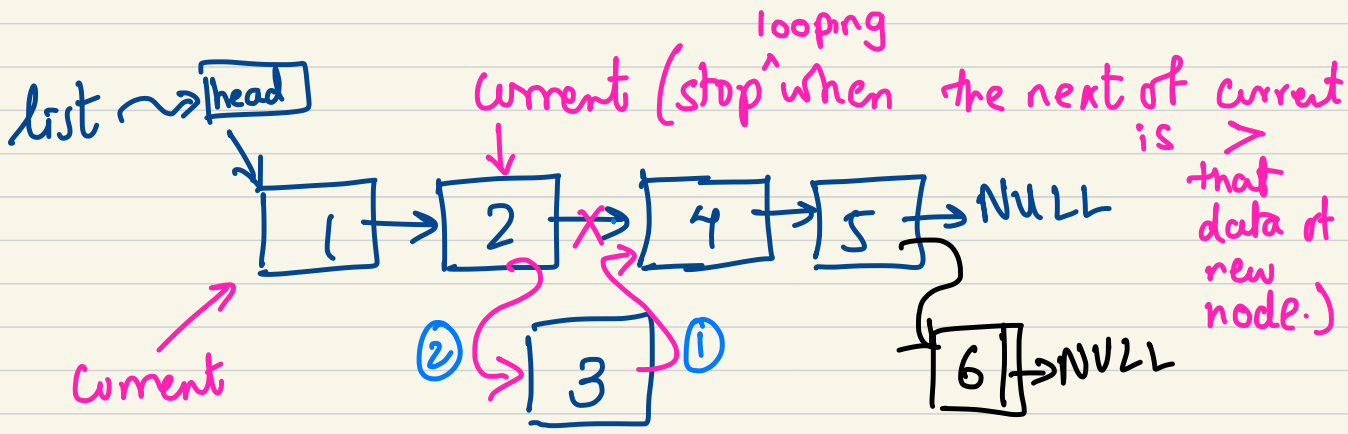
```
    current->next = createNode(data);
```

to know if we were capable of adding a node

```
    if (current->next != NULL) return true;
    else return false;
```

■ In black are special cases

To insert a node in an ordered list



seg. fault if current → next is NULL

```
while(current->next->data < data){
    current = current->next;
}
Node * newNode = createNode(data);
(1) newNode->next = current->next;
(2) current->next = newNode;
```

Renewed Code: (Recall lazy evaluation)

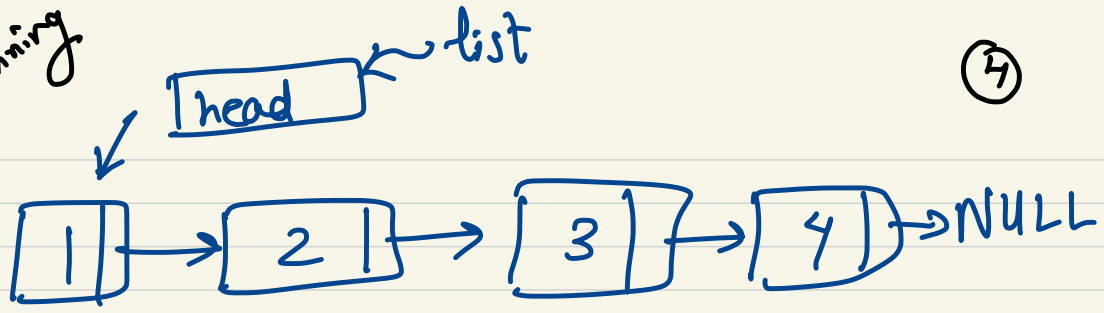
is false

won't be evaluated if

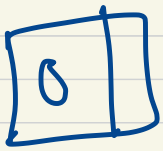
```
while((current->next != NULL) && current->next->data < data)
    current = current->next;
}
Node * newNode = createNode(data);
(1) newNode->next = current->next;
(2) current->next = newNode;
```

(1) If the node is inserted at the end

(2) If the node is inserted in the beginning



(4)




```
if (current -> data > data)
    return insertAtFront(list, data);
while((current ->next != NULL) &&
                    current ->next ->data < data)
    current = current ->next;
}
Node * newNode= createNode (data);
(1) newNode ->next = current ->next;
(2) current -> next = newNode;
```

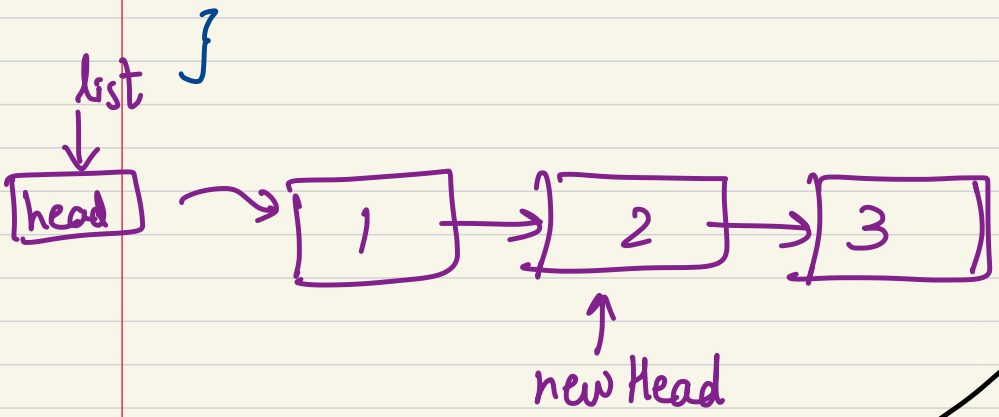
(3) If the list is empty :(

```
if ( list ->head ==NULL || data < current -> data)
    return insertAtFront(list, data);
while((current ->next != NULL) &&
                    current ->next ->data < data)
    current = current ->next;
}
Node * newNode= createNode (data);
(1) newNode ->next = current ->next;
(2) current -> next = newNode;
```

```
bool insertInOrderedList (linkedList * list, int data){
    Node * current = list -> head;
    if ( list ->head ==NULL || data < current -> data)
        return insertAtFront(list, data);
    while((current ->next != NULL) &&
          current ->next ->data < data)
        current = current ->next,
    }
    Node * newNode= createNode (data);
(1) newNode ->next = current ->next,
(2) current -> next = newNode;
    if (newNode ==NULL) return false;
    else    return true;
}
```

How to delete / shrink a linked list?

```
void deleteFront(LinkedList *list){
    Node *newHead = list->head->next;
    free(list->head);
    list->head = newHead;
}
```

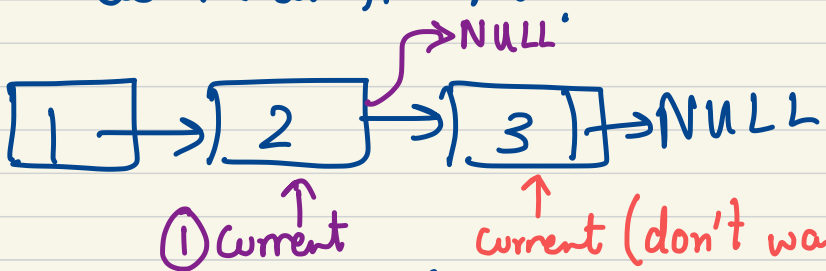


segmentation Fault :C

① What if list is empty?

```
void deleteFront(LinkedList *list){
    if(list->head == NULL)
        return;
    Node *newHead = list->head->next;
    free(list->head);
    list->head = newHead;
}
```

How to delete at the tail?




```
void deleteAtBack(LinkedList *list, int data){
```

do nothing if list is empty

```
    if (list->head == NULL) {
        return;
    }
```

If there is 1 node

```
    if (list->head->next == NULL)
        deleteFront(list);

    while(current->next->next != NULL) {
        current = current->next;
    }
    free(current->next);
    current->next = NULL;
}
```

Delete all nodes, return # of nodes you deleted

```
int deleteAllNodes(LinkedList *list){
    int numDeleted = 0;
    while(list->head != NULL){
        deleteFront(list);
        numDeleted++;
    }
    list->head = NULL;
    return numDeleted;
```

← unrequired as deleteFront does this