

Today - Lecture 14 - CS162

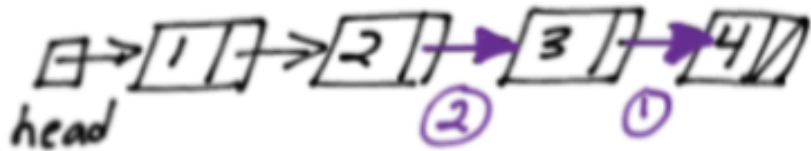
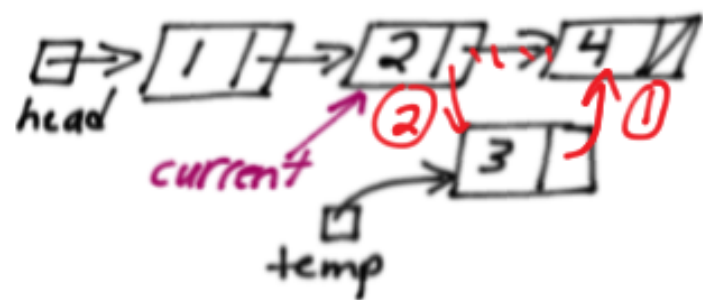
- 1) Insert in the midst
- 2) create the Algorithm for inserting in sorted order
- 3) Begin discussing removal
- 4) Answer Practice Questions!
- 5) Next time - Experience Recursion!

Announcements:

- * Online students - send me email if you want to pick up your exam (karlaf@cs.pdx.edu)
- * Important to begin practicing!

Inserting - in the midst

③ Add in the middle



- 1) First, make sure head is not NULL
- 2) Traverse to the right spot ...
- 3) Connect up the nodes (order is important)

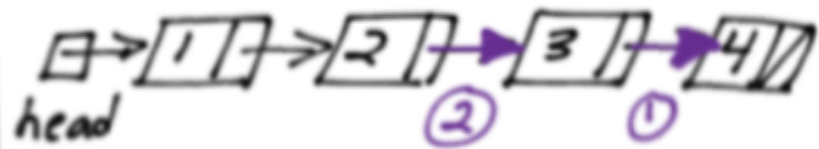
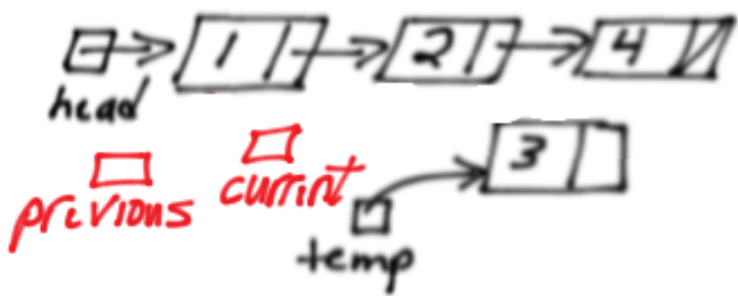
Question

How do we know that it is time to insert?

- a) data is between current and current \rightarrow next
"look ahead"
- b) or, drag a previous pointer one node behind

Inserting - in the midst (with a previous pointer)

③ Add in the middle



Traversal

make sure we don't dereference a NULL ptr

```
while (current && not time to stop)
{
  previous = current;
  current = current -> next;
}
```

Connect up

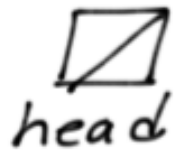
```
previous -> next = temp;
temp -> next = current;
```

Algorithm for inserting in sorted Order

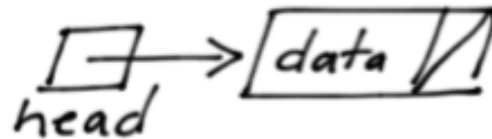
1. First Understand the problem & all special cases:

Before

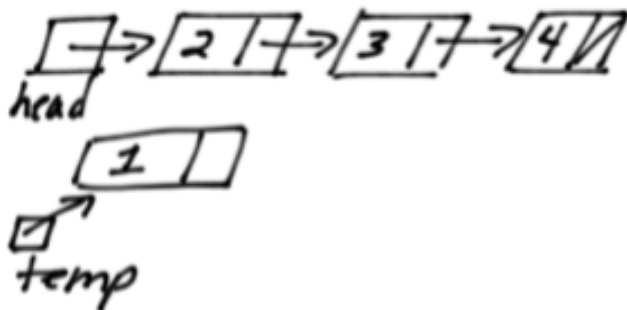
① Empty List



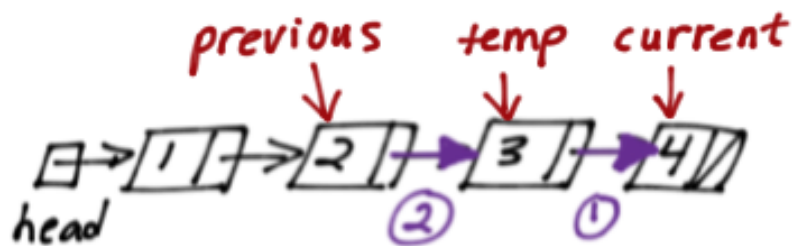
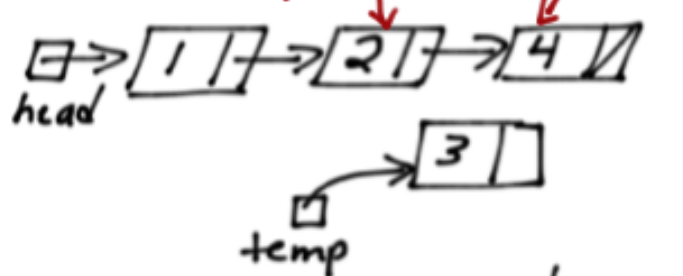
After



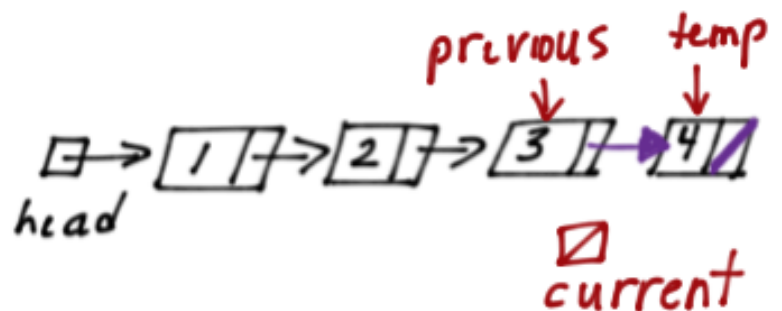
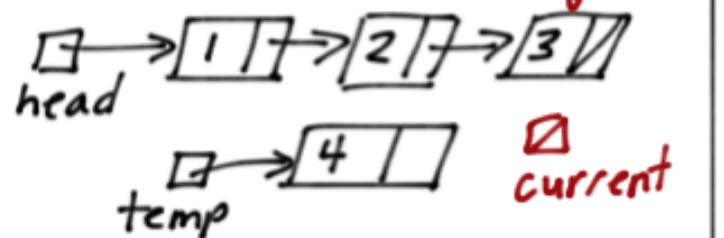
② Add at beginning



③ Add in the middle



④ Add at the end

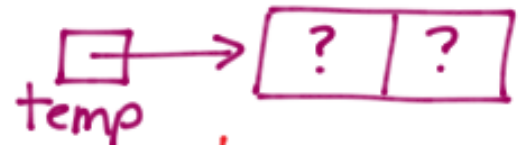


Algorithm to insert in sorted order

↑
alphabetical!
(if arrays of characters you must use strcmp)

1. Step 1 - Since we will insert somewhere in the list, let's allocate memory and store the data in the node. Initialize all data members

a) Allocate a node



node * temp = new node;

b) Store the data

c) Set the next pointer to NULL



2. Step 2 - Determine where to insert :

2a) Empty List *head is NULL*

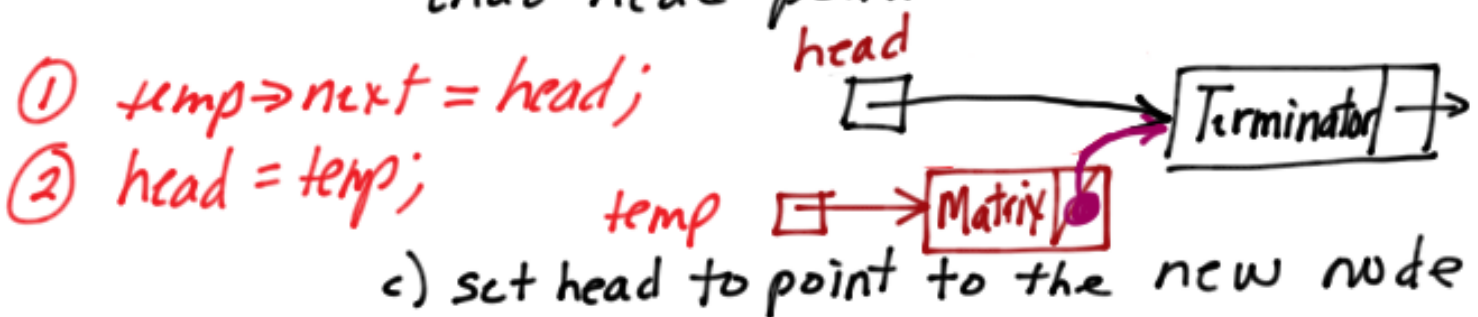
a) Set head to point where temp points



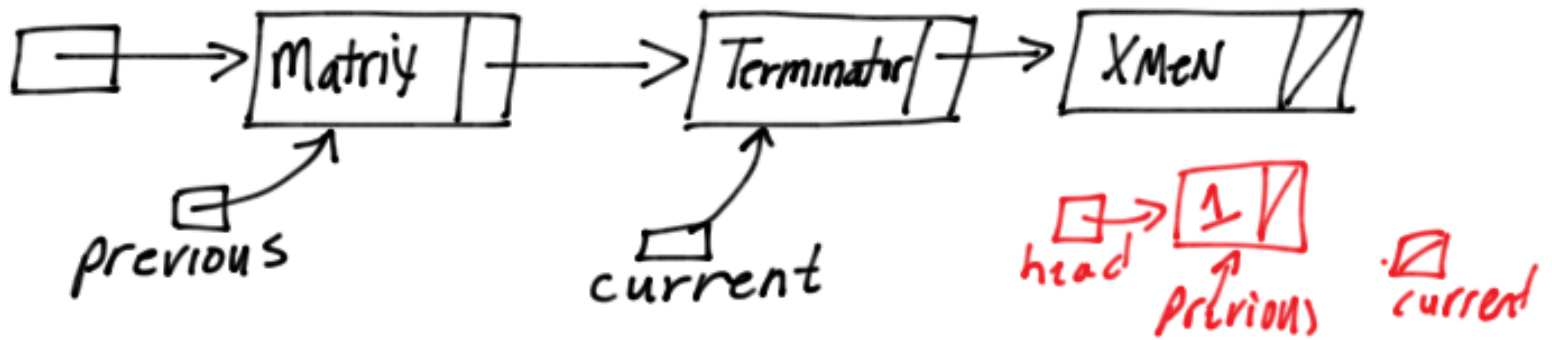
2b) Insert at the beginning (head gets changed)

a) Check if the data being inserted is "less" (alphabetically) than the data at the first node

b) if so, set the new node's next pointer to point to the same place that head points



Step 2c) Insert elsewhere (does not alter head)



a) set up temporary pointers

- we know head is not NULL

- we know the data won't be inserted as the first node

$node * previous = head;$

$node * current = head \rightarrow next;$

could this seg. fault? NO

b) Traverse as long as there are still nodes & it isn't time yet to insert

- stopping conditions

current is NULL

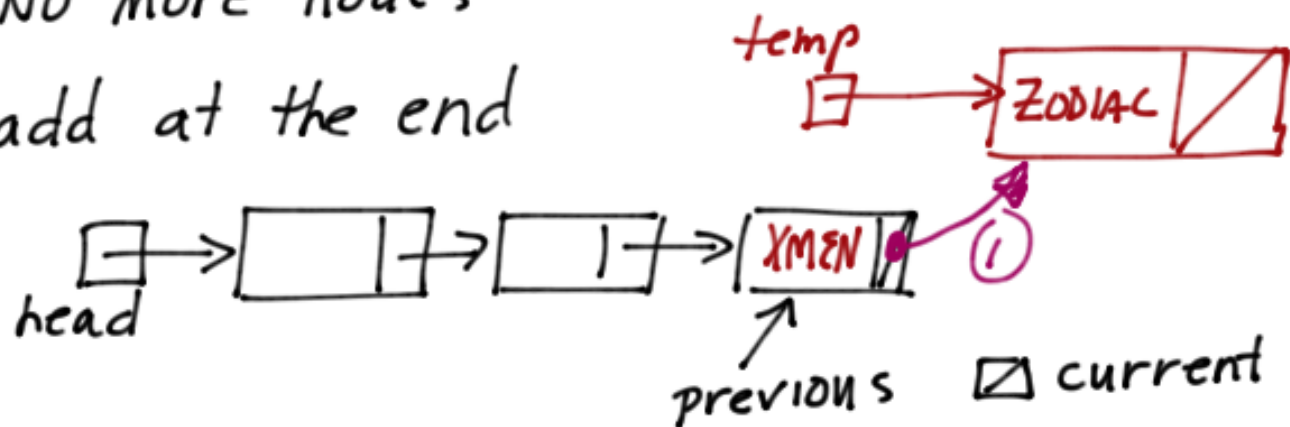
data less than current's data

} TIME to ADD!

Examine **BOTH** stopping conditions

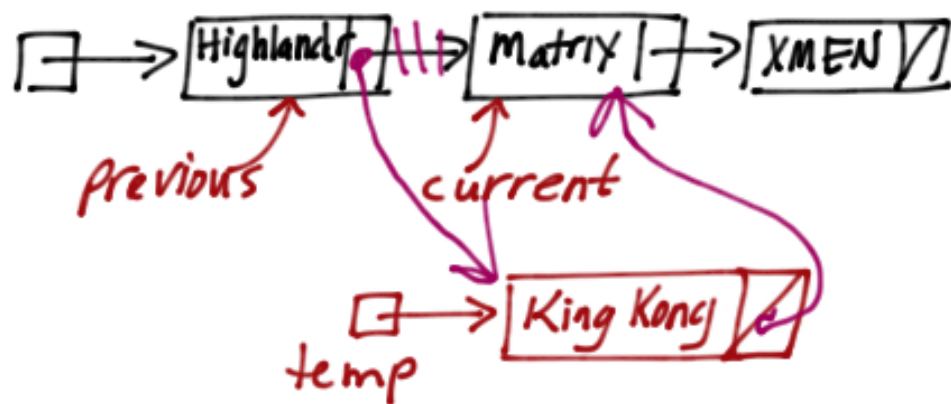
a) if current is NULL

- NO more nodes
- add at the end



① - set previous \rightarrow next = temp

b) Otherwise (the data being added is "less" than the data that current is pointing to:



- set previous \rightarrow next to temp, and temp \rightarrow next to current

But... Let's Review how to traverse?



Why not:

```
while (strcmp(current->data, newdata) < 0)
{
    assuming an array of characters
    previous = current;
    current = current->next;
}
```

Fix:



```
while (current && strcmp(current->data, newdata) < 0)
```


↑
interesting fact about
the order of evaluation

current != NULL

↑
Dereference
If current is
null it will
Seg fault

Removal from a LLL

Special Cases

1) Empty List 
head

2) Remove the first node, causing head to be changed



- can we just say: delete head?
NO!

3) Remove elsewhere - requiring traversal!



4) No Match found (ultimately current becomes NULL)

- Do Nothing!