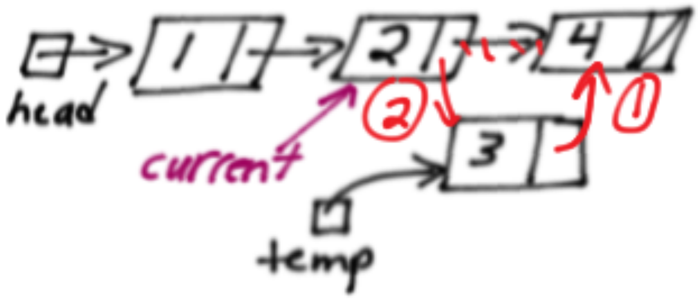
Today - Lecture 14 - CS162

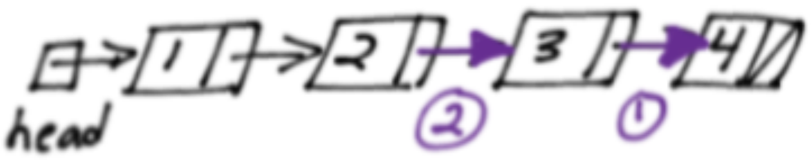
1) Insert in the midst
2) create the Algorithm for inserting in sorted order
3) Begin discussing removal
4) Answer Practice Questions!
5) Next time - Experience Recursion!

Announcements:

Inserting – in the midst

③ Add in the middle





1) First, make sure head is $\boxed{\text{not}}$ 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?

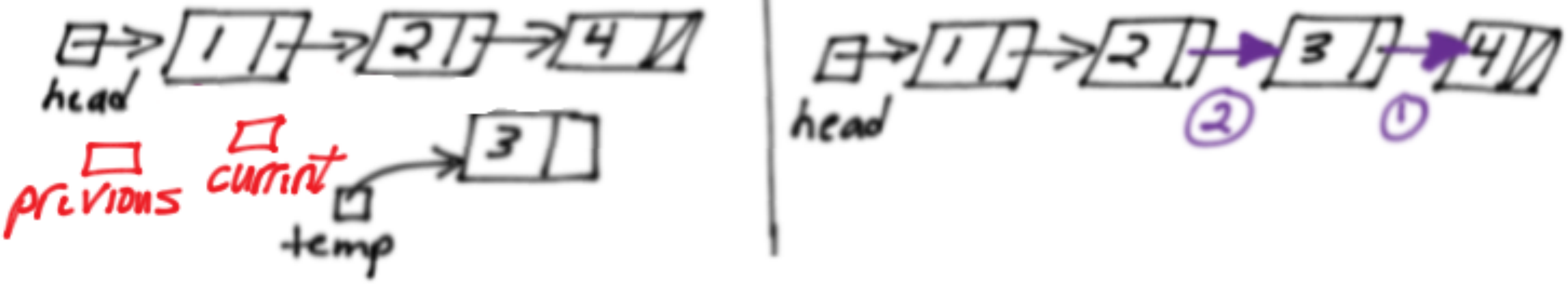
a) data is between current and current->next

"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

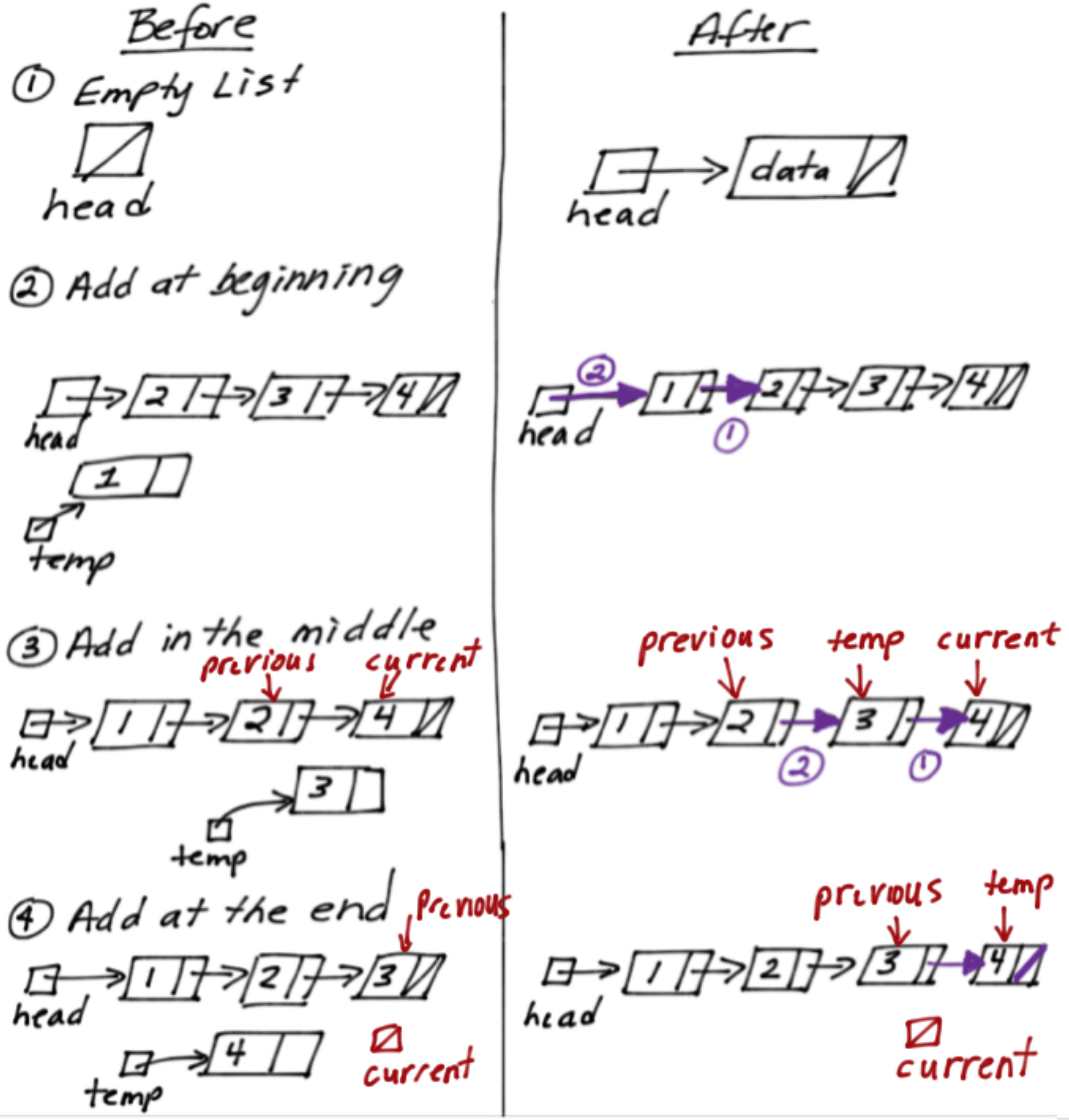
```
while (current && not time to stop)
{
        previous = current;
        current = current → next;
}
```

Connect up

```
previous → next = temp;
temp → next = current;
```

Algorithm for inserting in sorted Order

1. First Understand the problem & all special cases:

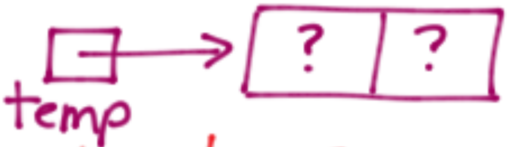


Algorithm to insert in sorted order

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

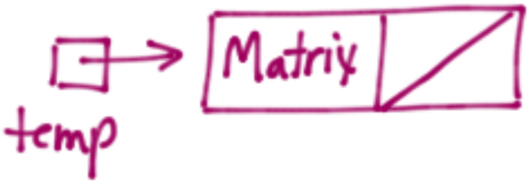
   a) Allocate a node



node* temp = new node;

   b) Store the data

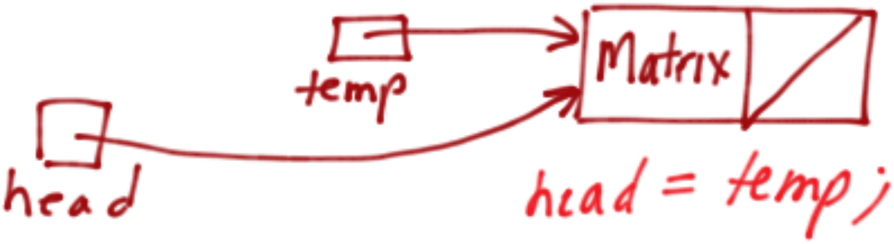
   c) Set the next pointer to NULL



2. Step 2 - Determine where to insert :

2a) Empty List head is NULL

a) Set head to point where temp points



head = temp;

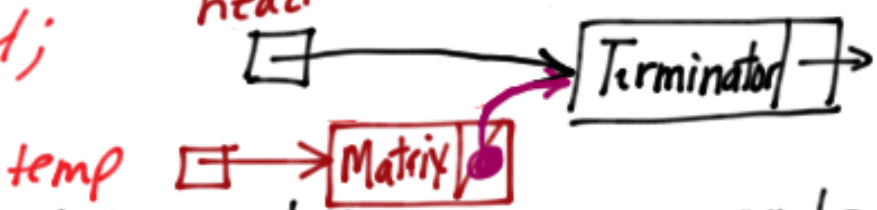
2b) Insert at the beginning (head gets changed)

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

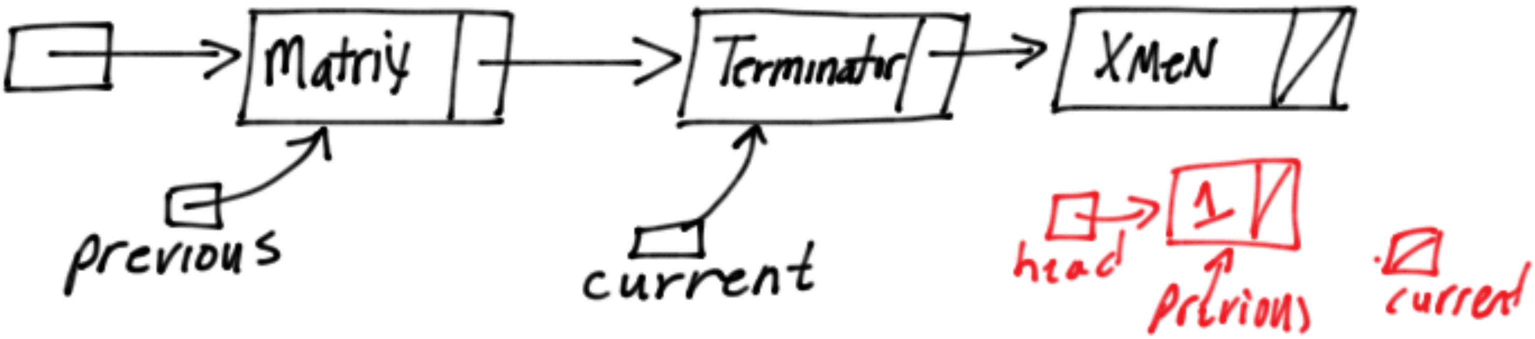
① temp->next = head;

② head = temp;



c) set head to point to the new node

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→next;

could this seg. fault? NO

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

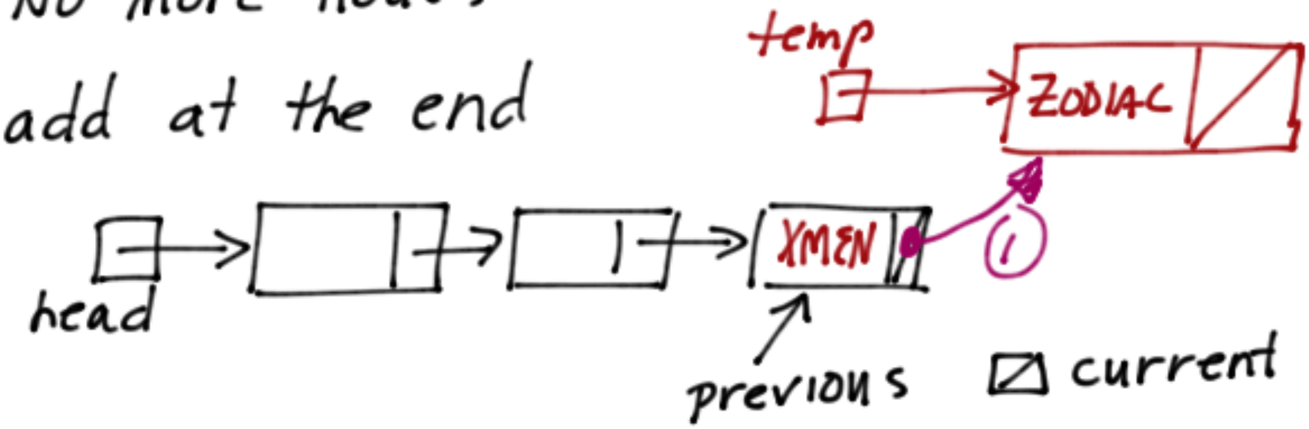
- stopping conditions

current is NULL
data less than current's data } TIME to ADD!

Examine **BOTH** stopping conditions

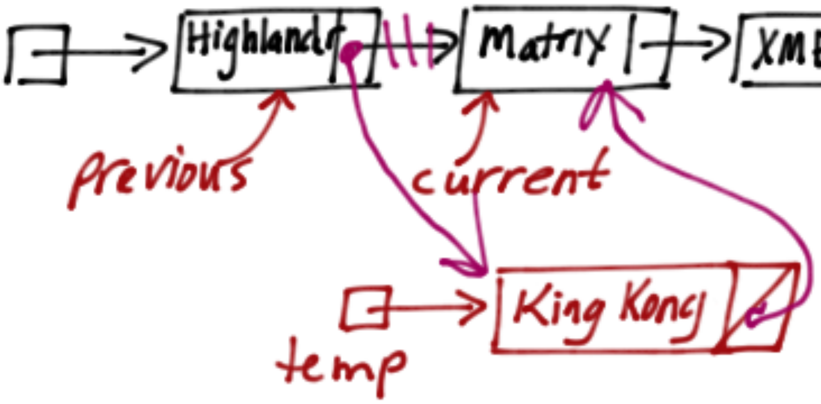
a) if current is NULL

- NO more nodes
- add at the end



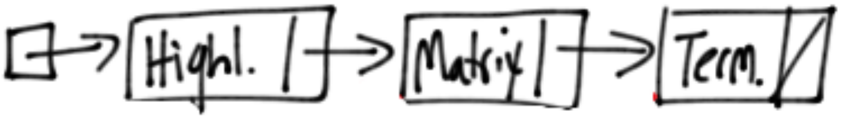
1 - set previous→next = temp

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



-set previous→next to temp, and temp→next to current

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



why not:

```
while (strcmp(current->data, newdata) < 0)
{
        previous = current;
        current = current->next;

}
```

assuming an array of characters

Fix:

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

current != NULL

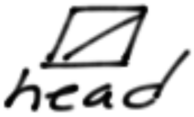
interesting fact about the order of evaluation

Dereference

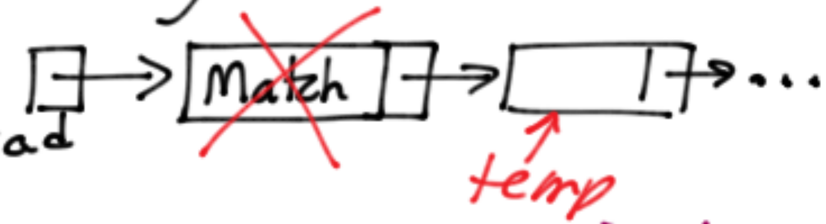
If current is null it will Seg fault

Removal from a LLL

Special Cases

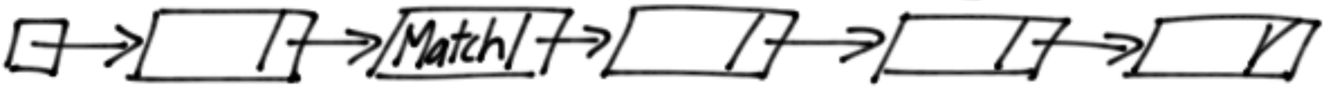
1) Empty List 

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!