Today - Lecture 15 — CS162

1) Deletion Algorithms for a LLL
2) Removing all nodes in a LLL
3) Experience Recursion
4) Next time: **Recursion**

Announcements:

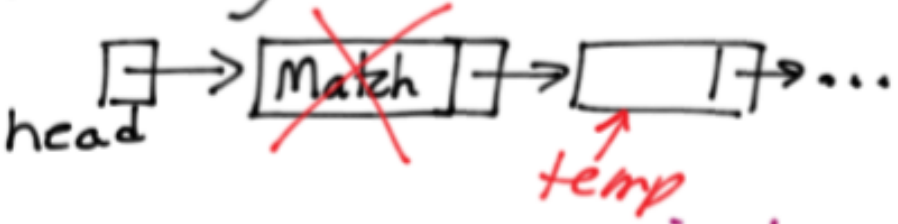
* PRACTICE LLL !

Removal from a LLL (Remember)

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!

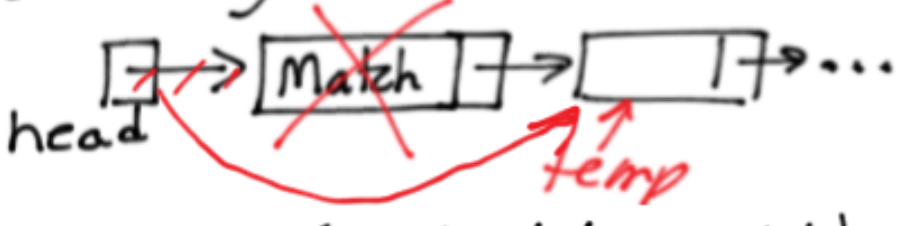
Special Cases

1) Empty List 


```
if (!head)  // if head is NULL
    return;  // nothing to delete!
```

Case #2:

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




```
if (strcmp(head->data, Match) == 0)  //Match!!
```

if our data, and match, are arrays of characters

```
{   temp = head->next;
}   delete head;  head = temp;
```

Case #3:

3) Remove elsewhere - requiring traversal!


```
current = head->next; // 2nd node
previous = head;
while (current && strcmp(current->data, match)!=0)
{
        previous = current;  //So we can "reconnect"
                             // around the node
                             // being deleted.

        current = current->next;
```

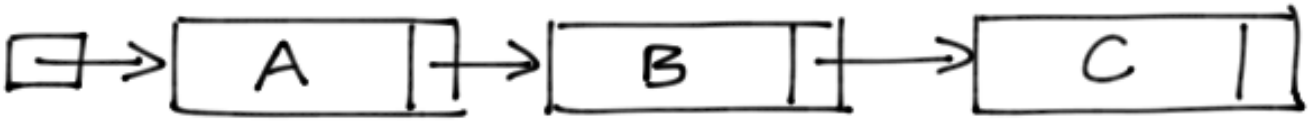
Traverse to the next node.

```
}

//so... could current be NULL now?
 if (current != NULL) // something to remove
 {
        previous->next = current->next;
        delete current;
 }
```

Removal All (when done with the LLL)

- Performed by the destructor when using classes



① why not: delete head?

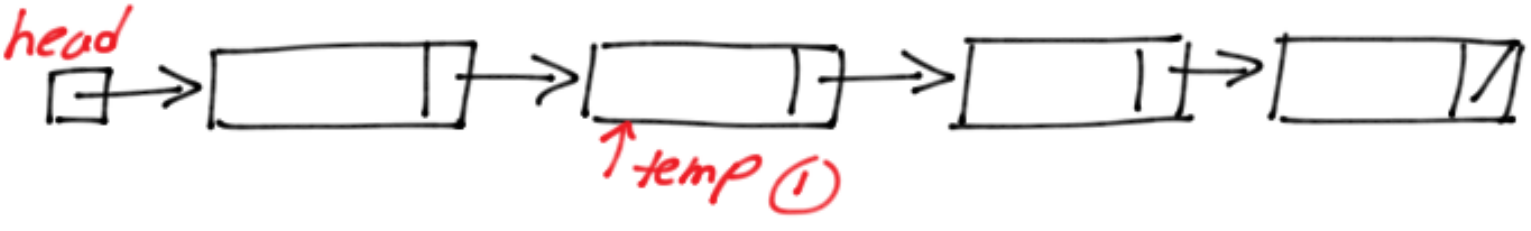
② why not:

```
while (head)
{
        delete head;
        head = head->next;

}
```

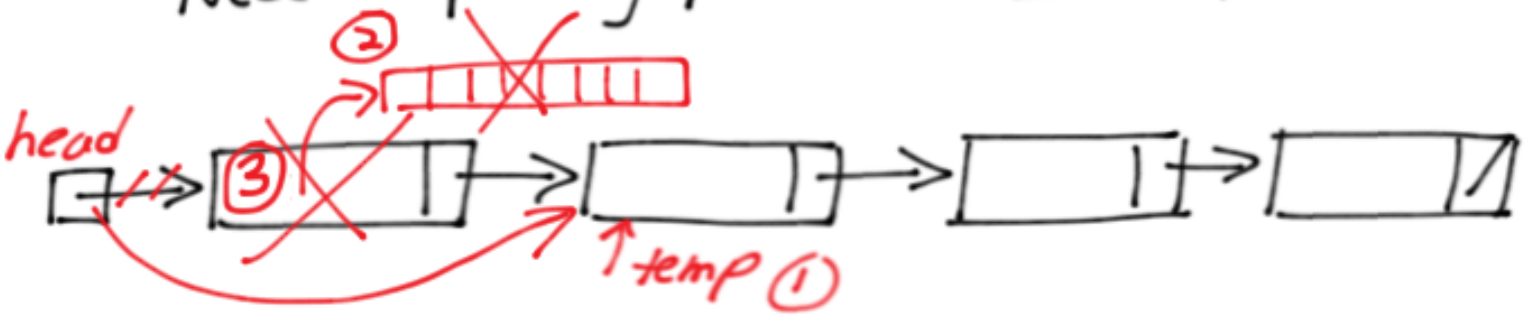
is this our memory?

③ Need temporary pointers (LOCAL VARIABLES)



Remove All:

Need temporary pointers (LOCAL VARIABLES)



IF There are Nodes —

1. set temp to point to the next node
2. Delete the dynamic memory managed by the node that head points to
3. Delete the node that head points to
4. Update head to point where temp is pointing

What would this do?

```
struct node
{     ~node();
      char * name;
      node * next;
};
```

```
node :: ~node()
{
      delete [] name;
      delete next;
      next = NULL;
}
```

```
delete head;
```