CSE 105:
Data Structures and Algorithms-I
(Part 2)

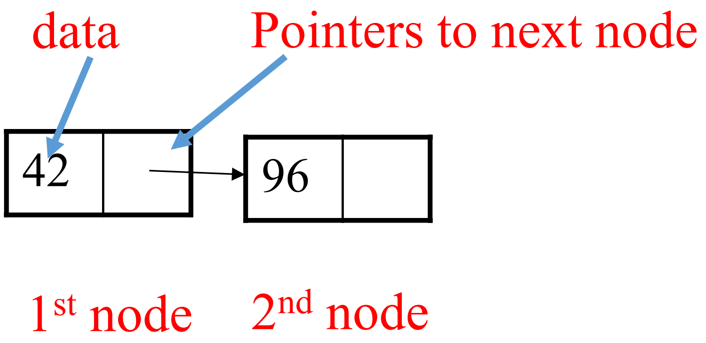
Instructor

Dr Md Monirul Islam

Linked List Based Implementation

A *linked list* is a data structure where each object is stored in a *node*

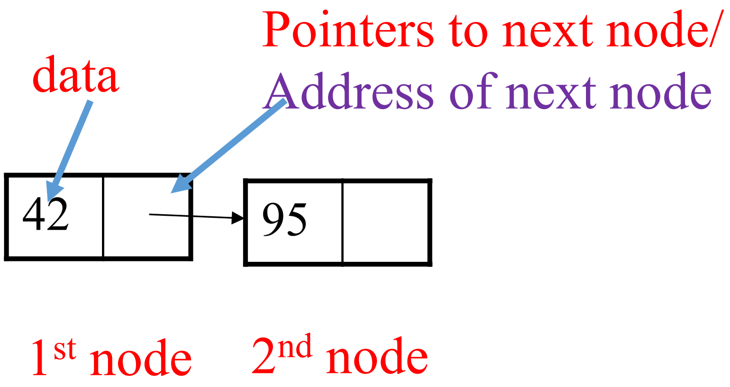
As well as storing data, the node must also contains a reference/pointer to the node containing the next item of data



Linked List Based Implementation

- Each node is a dynamically created structure/class
- Each node is divided into 2 parts:
  - 1st part contains the information of the element.
  - 2nd part is called the link field or next pointer field which contains the address of the next node in the list.

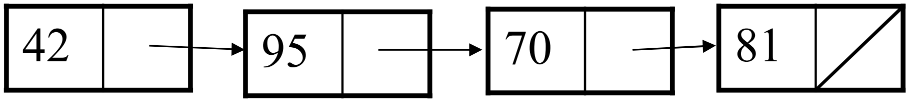
```
struct node {
    int element;
    struct node *next_node;
}
```



Linked List

- A linked list is a linear collection of nodes
- The next pointer of the last node is NULL/NILL

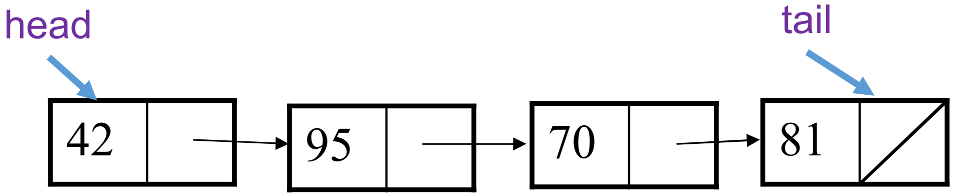
```
struct node {
     int element;
     struct node *next_node;
}
```



Linked List

- A linked list has a head and a tail (sometimes not explicitly mentioned)

```
struct node {
    int element;
    struct node *next_node;
}
 struct node  *head,*tail;
```

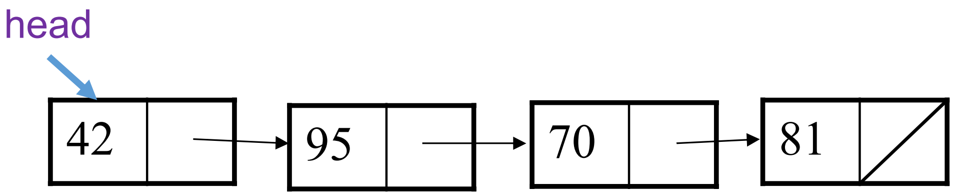


Linked List

A linked list uses dynamic allocation, and therefore each node may appear anywhere in memory

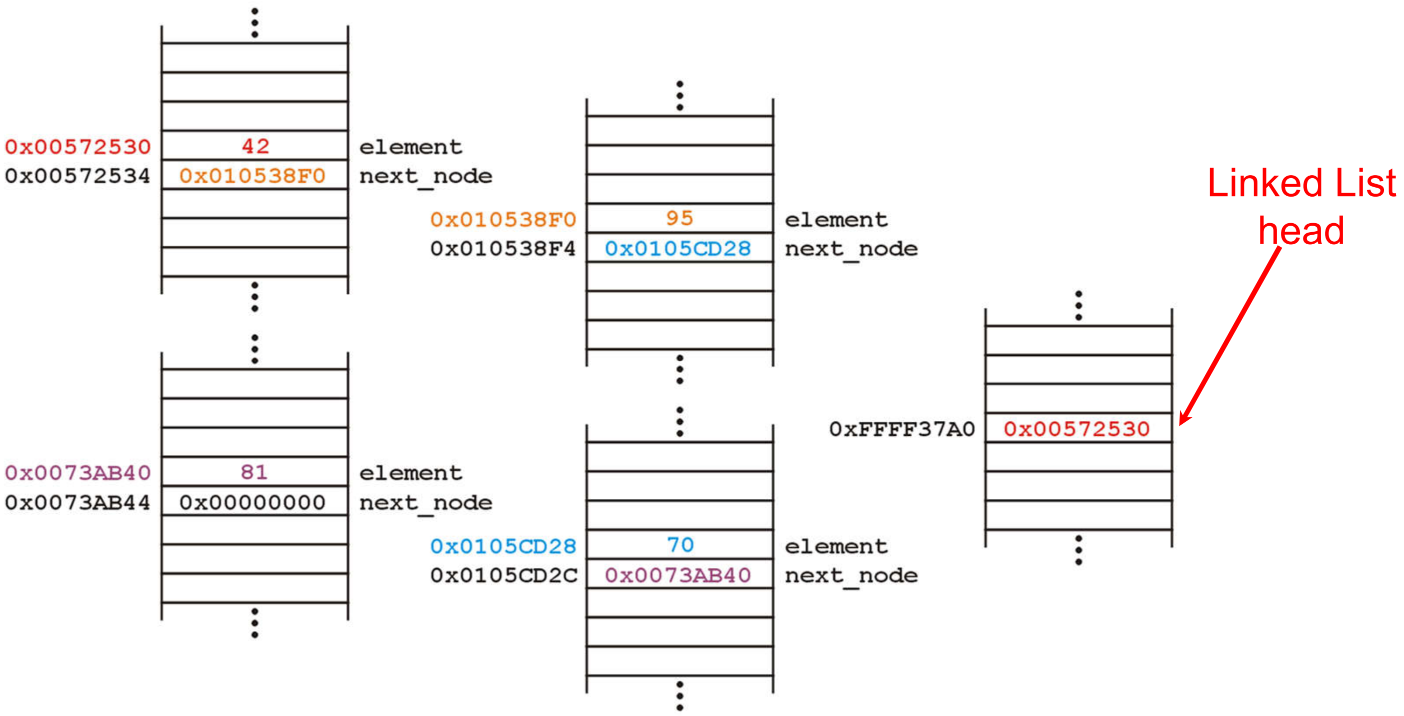
Also the memory required for each node equals the memory required by the member variables

- 4 bytes for the linked list head (a pointer)
- 4 + 4 = 8 bytes for each node (an **int** and a pointer)
  - We are assuming a 32-bit machine



Linked List in Memory

The previous list could occupy memory as follows:



0x00572530 42 element
0x00572534 0x010538F0 next_node

0x010538F0 95 element
0x010538F4 0x0105CD28 next_node

0x0073AB40 81 element
0x0073AB44 0x00000000 next_node

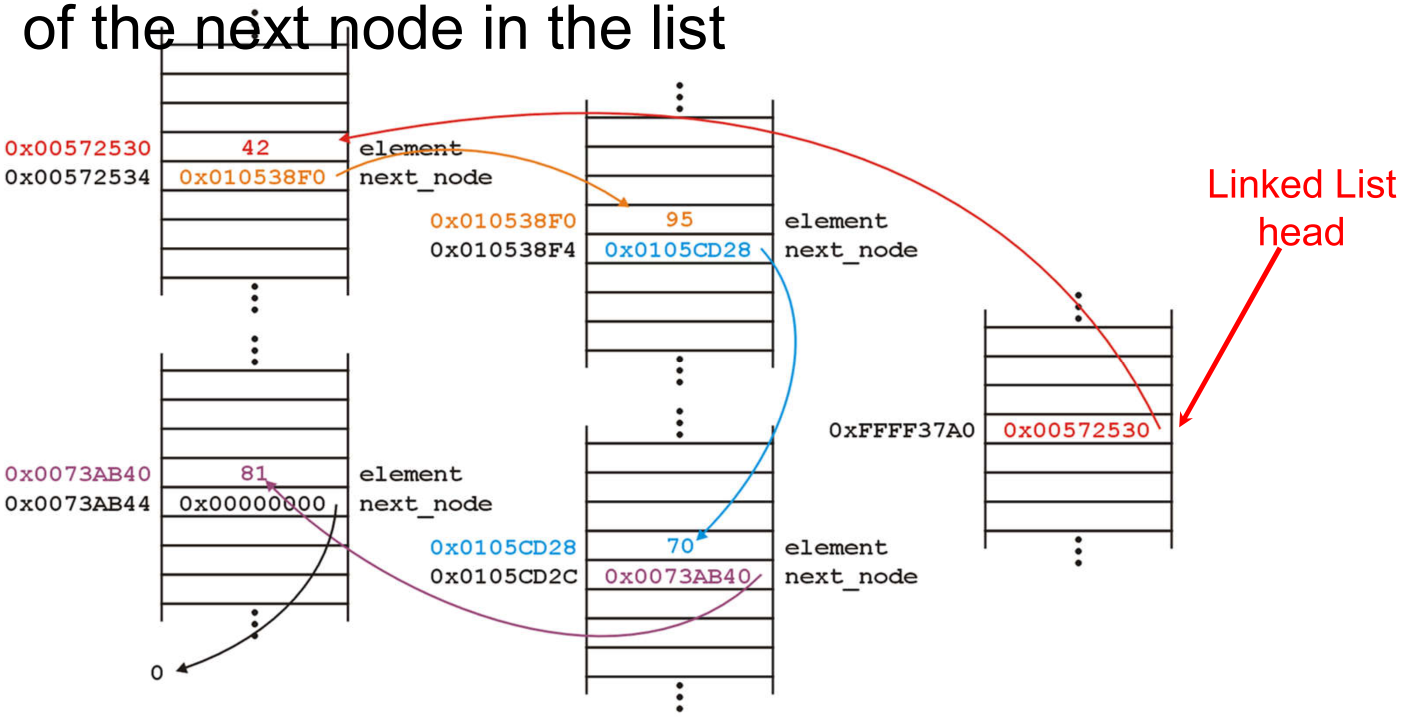
0x0105CD28 70 element
0x0105CD2C 0x0073AB40 next_node

0xFFFF37A0 0x00572530

Linked List head

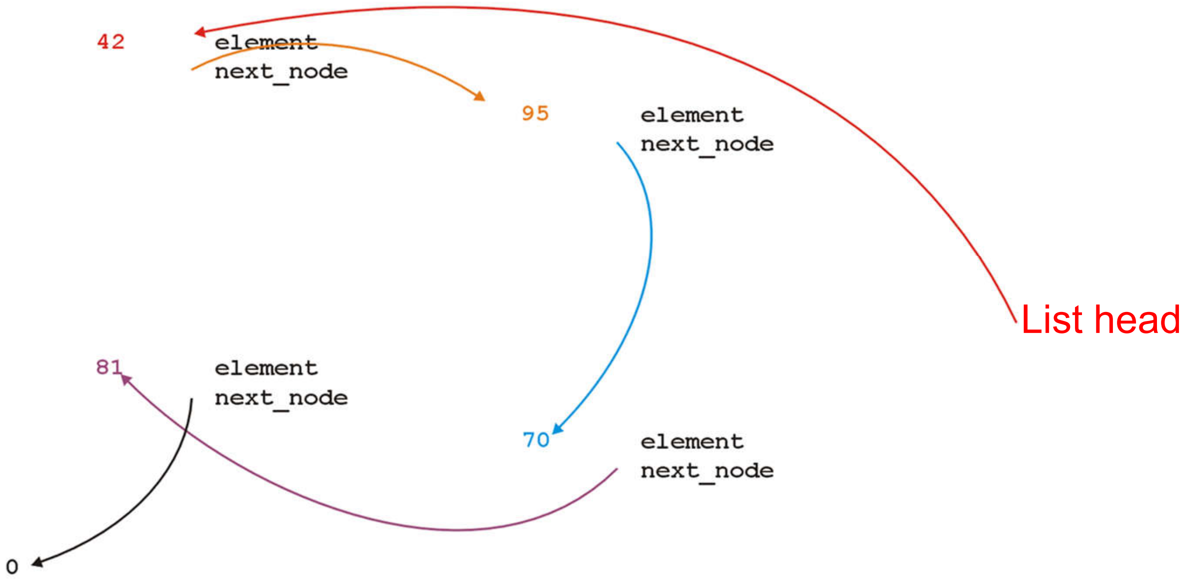
Linked List in Memory

The **next_node** pointers store the addresses of the next node in the list



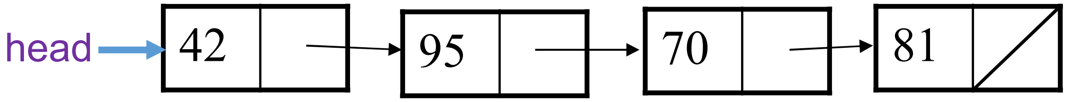
Linked List in Memory

Because the addresses are arbitrary, we can remove that information:



Linked List in Memory

We will clean up the representation as follows:



We do not specify the addresses because they are arbitrary

Linked List Creation and Operations

First, we want to create a linked list

We also want to be able to:

- insert into,
- Access/search/print, and
- Delete from

the values stored in the linked list

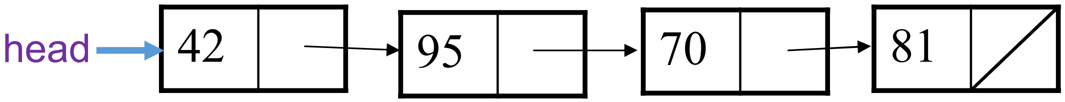
Linked List Creation

```
// linked list node in C

struct node {

    int element;

    struct node *next;

}
```

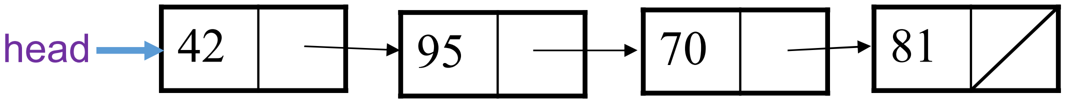
```
//linked list node in C++ template

template <typename E> class Link {
public:
E element; // Value for this node
Link *next; // Pointer to next node in list
// Constructors
Link(const E& elemval, Link* nextval =NULL)
    { element = elemval; next = nextval; }
Link(Link* nextval =NULL) { next = nextval; }


}; //class end
```

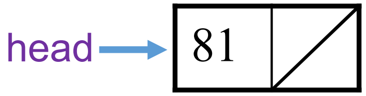
Linked List Creation

```
// linked list node in C

struct node {

    int data;

    struct node *next;

}
```



Linked List Creation



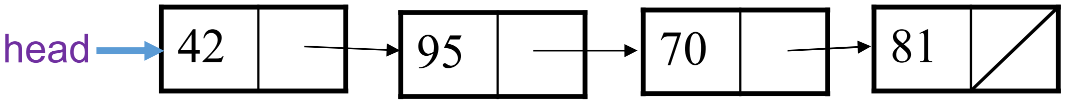


```
// linked list node in C
struct node {
    int data;
    struct node *next;
}
```

```
struct node *head, *temp;
temp = (struct node*) malloc (sizeof (struct node));
if (temp==NULL)  //error handling code
temp->data=81;
temp->next=NULL;
head=temp;
```

Linked List Creation

```
// linked list node in C

struct node {

    int data;

    struct node *next;

}
```



head

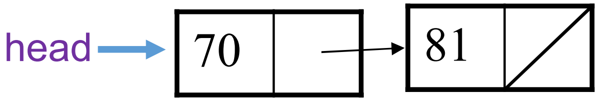
42

95

70

81

```
struct node *head, *temp;
temp = (struct node*) malloc (sizeof (struct node));
if (temp==NULL)  //error handling code
temp->data=81;
temp->next=NULL;
head=temp;
```



head

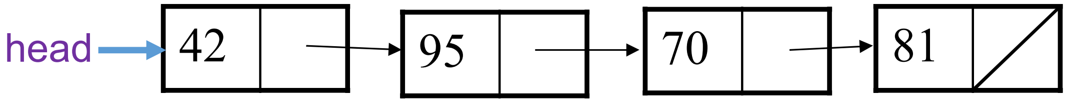
70

81

```
temp = (struct node*) malloc (sizeof (struct node));
if (temp==NULL)  //error handling code
temp->data=70;
temp->next=head;
head=temp;
```

Linked List Creation

- List created so far is a singly-linked list (SLL)



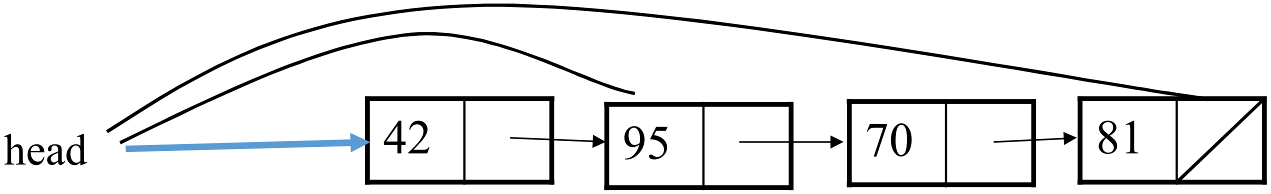
- Each node contains a value and a link to its successor
- The last node has no successor
- The header points to the first node in the list

Linked List Traversal

- One (bad) way to process every value in the list:

```
while (head != NULL) {
    do_something(head->data);
    head = head->next;    // move to next node
}
```

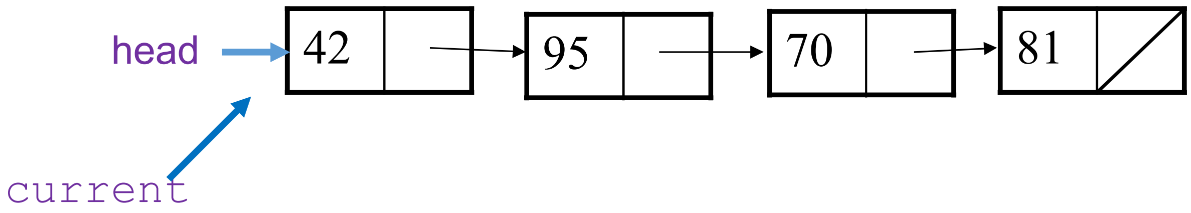
  - What's wrong with this approach?
    - (It loses the linked list as it progress!)



Linked List Traversal with a Current Reference

- Don't change head. Make another variable, and change it.

```
struct node *current = head;
```



- What happens to the picture above when we write:

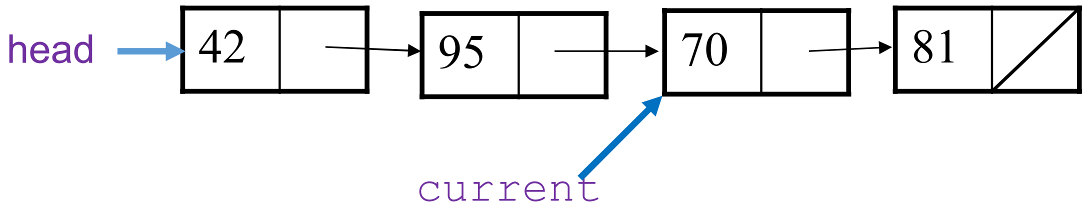
```
current = current->next;
```

Traversing the List Correctly

- The correct way to process every value in the list:

```
struct node *current = head;
while (current != NULL) {
    do_something (current->data);
    current = current->next;  // move to next node
}
```

  - Changing current does not damage the list.

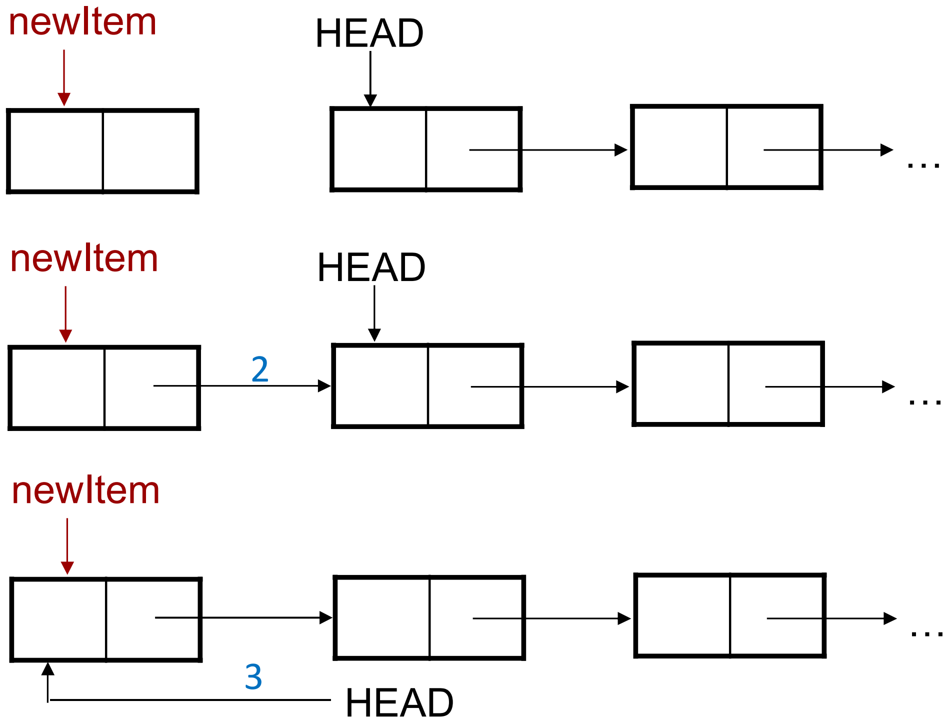


Insertion a new node into an SLL

- many ways to insert:
  - As the new first node
  - As the new last node
  - Before a given node (specified by a *reference*)
  - After a given node
  - Before a given value
  - After a given value
- All are possible, but differ in difficulty

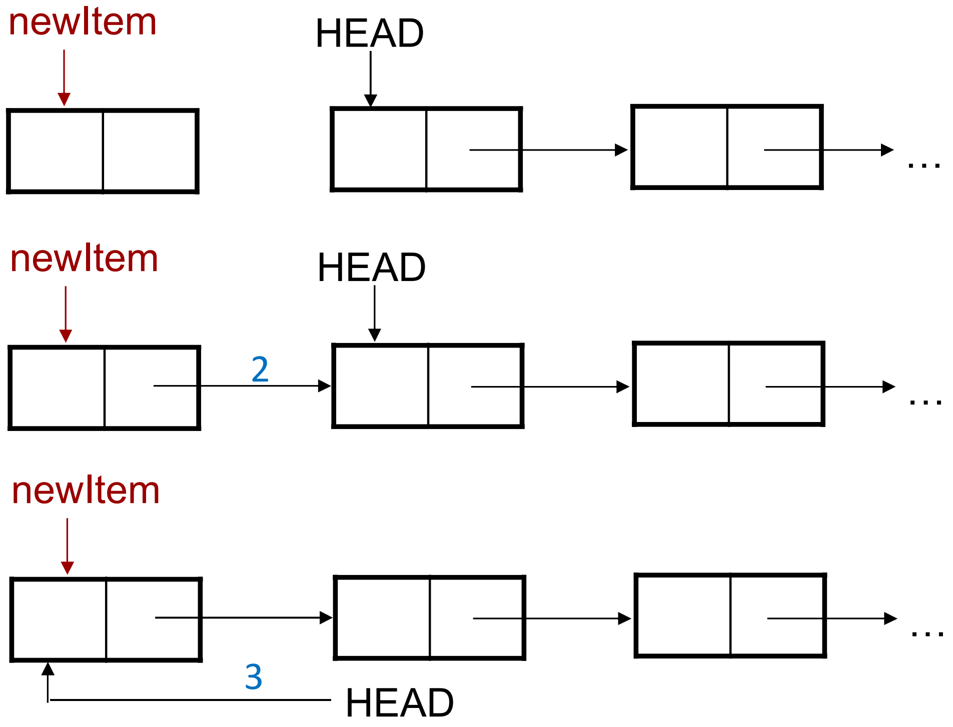
Insert First

- Step 1. Create a new node that is pointed by pointer *newItem*.
- Step 2. Link the new node to the first node of the linked list.
- Step 3. Set the pointer *head* to the new node.



Insert First

- Step 1. Create a new node that is pointed by pointer *newItem*.
- Step 2. Link the new node to the first node of the linked list.
- Step 3. Set the pointer *head* to the new node.

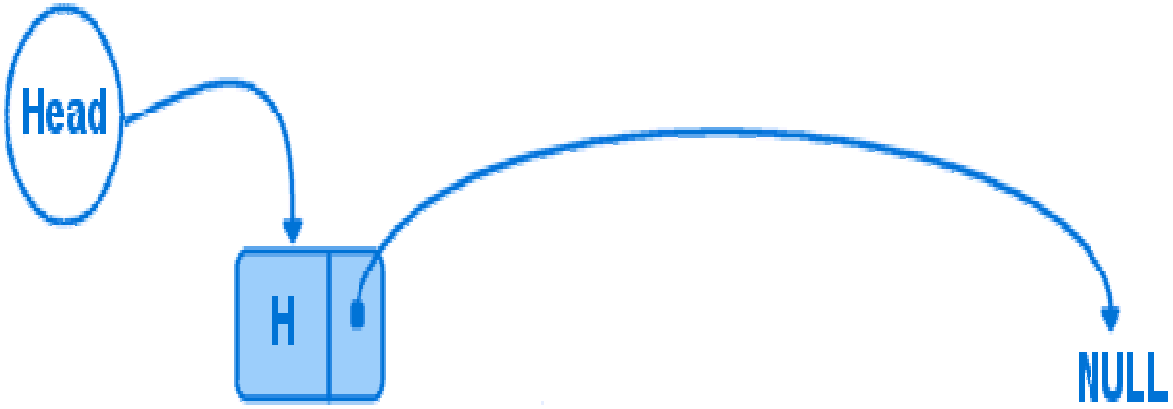


```
newItem = //create it as before
if (newItem ==NULL)  //error handling
newItem ->data= //assign it ;
newItem ->next=head;
head= newItem;
```

Complexity?

Insert Last

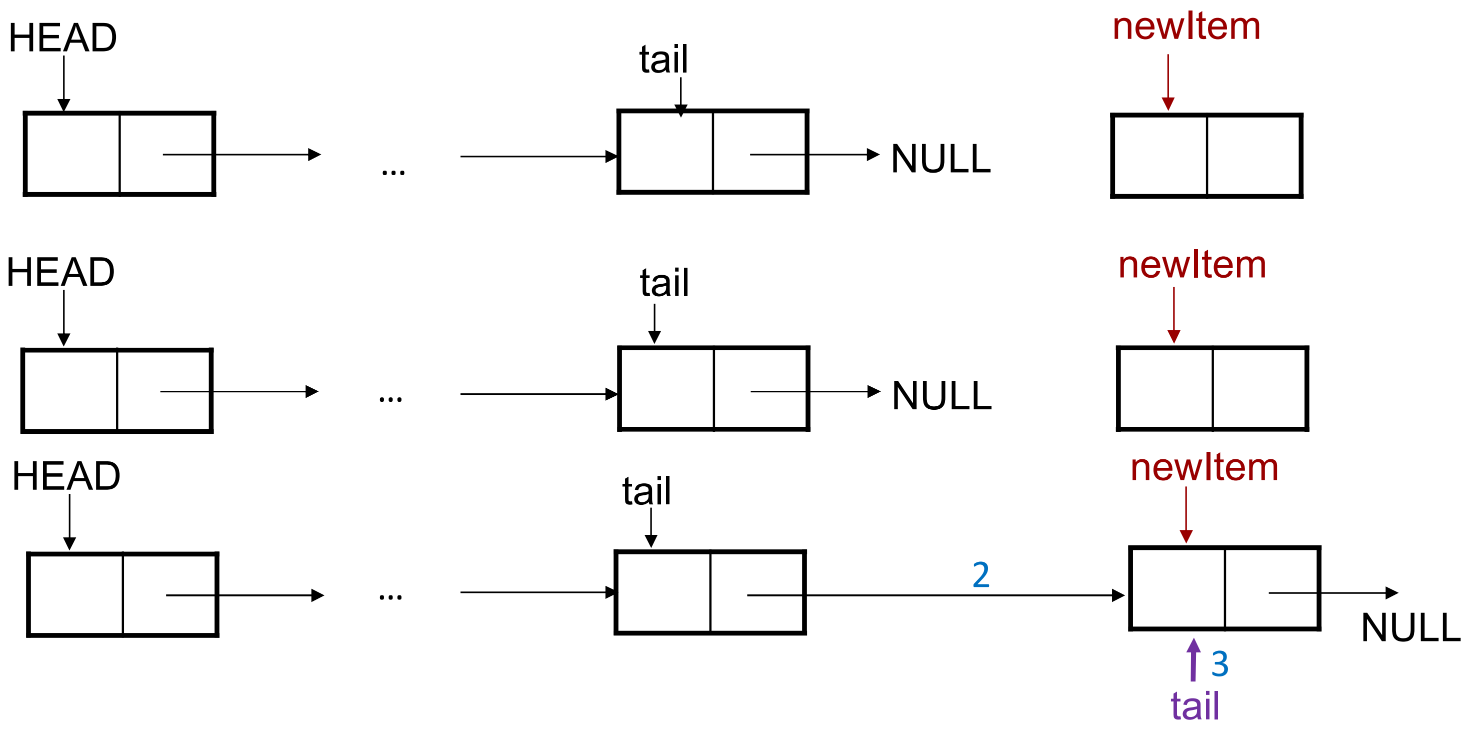
- To add a new node to the tail, we need to construct a new node with next field = NULL.
- Assume the list is not empty, locate the last node and change it's next field to point to the new node.



New node

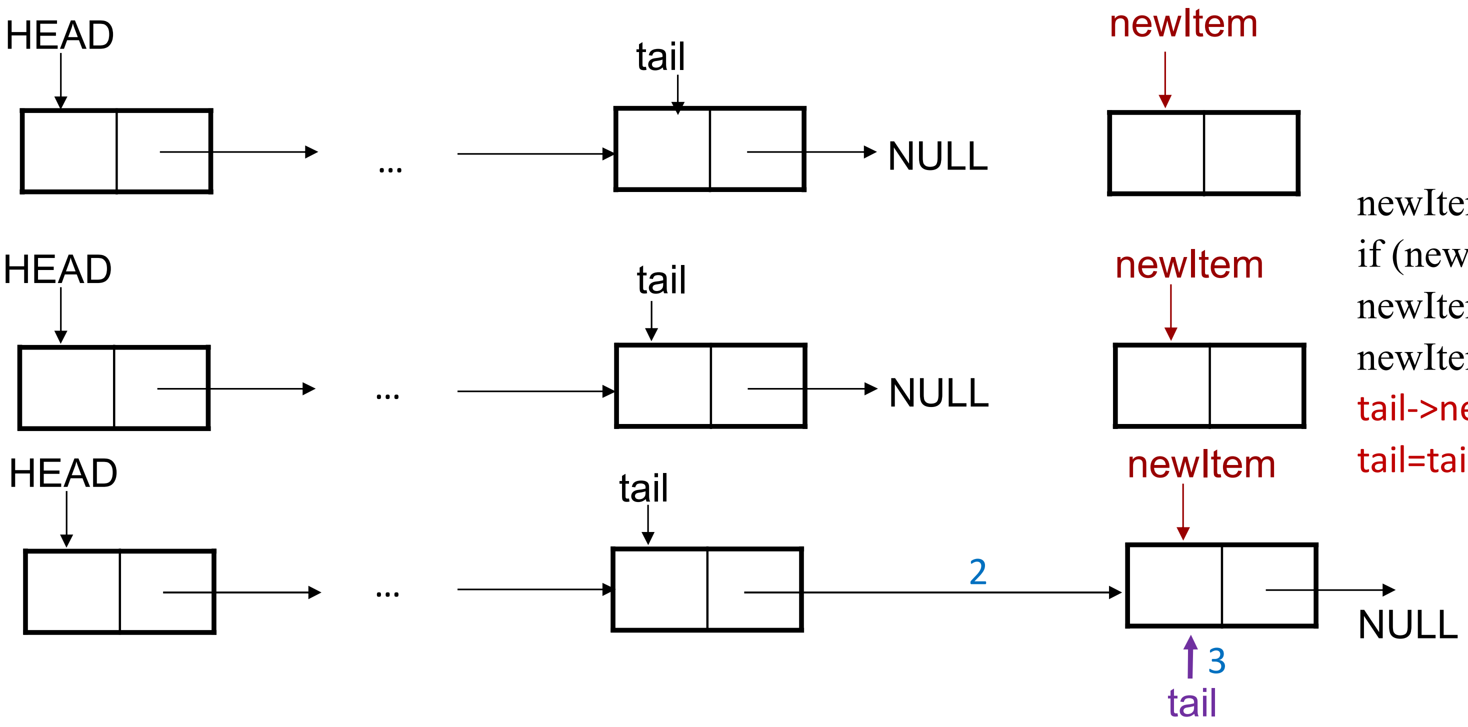
Insert Last

- Step1. Create the new node.
- Step2. Set a next field of temporary pointer tail to point to the new node.
- Step3. Set tail to point to tail->next.



Insert Last

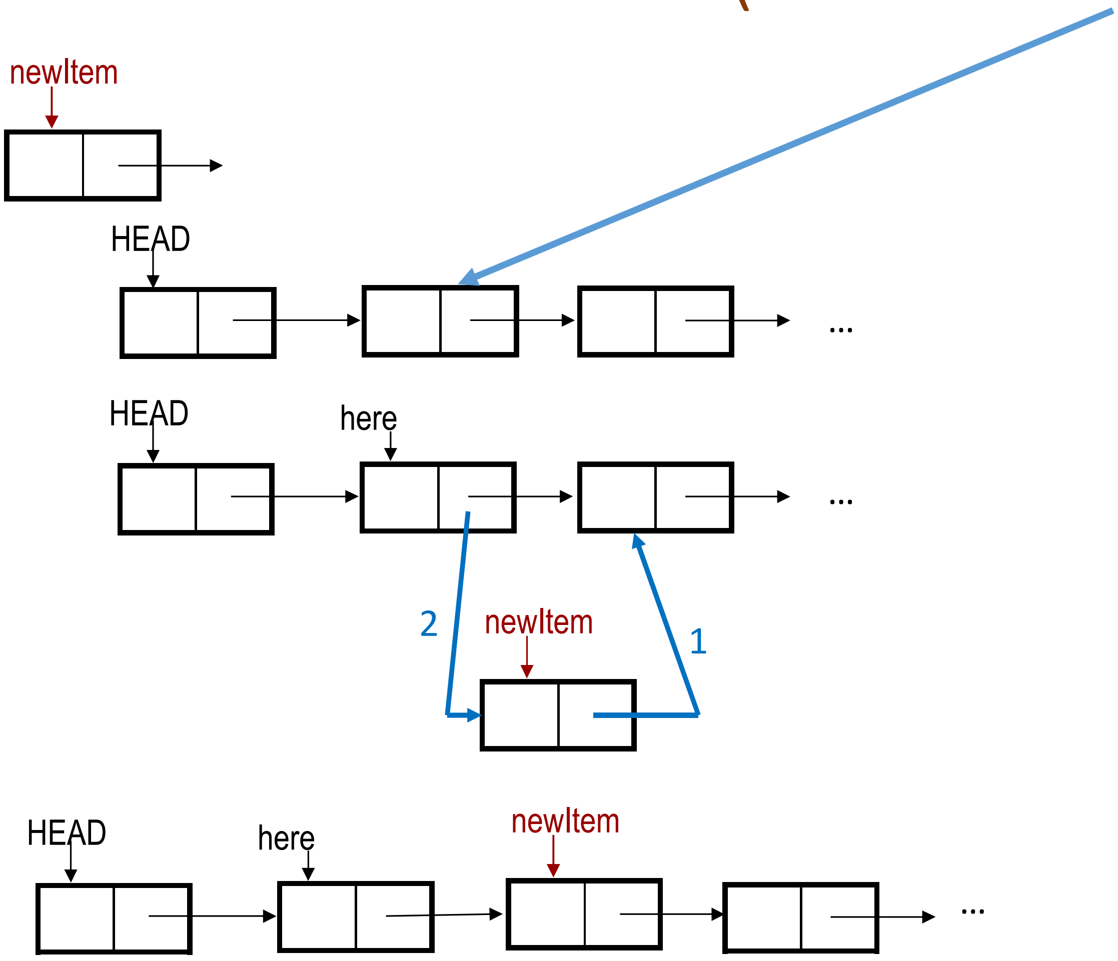
- Step1. Create the new node.
- Step2. Set a next field of temporary pointer tail to point to the new node.
- Step3. Set tail to point to tail->next.



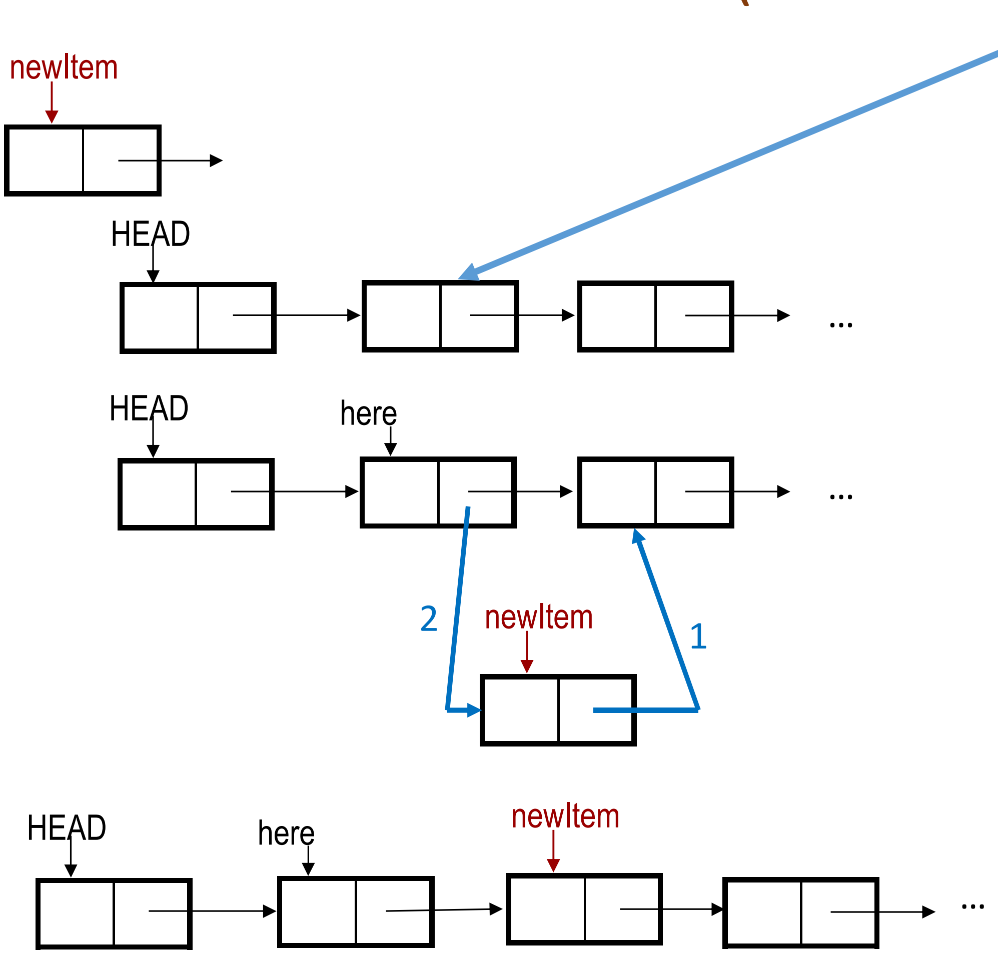
```
newItem = //create it as before
if (newItem ==NULL)  //error handling
newItem ->data= //assign it ;
newItem ->next=NULL;
tail->next = newItem;
tail=tail->next;
```

Complexity?

Insert at Middle (after a desired node)



Insert at Middle (after a desired node)



```
newItem = //create it as before
if (newItem ==NULL)  //error handling
newItem ->data= //assign it ;
newItem ->next=here ->next;
here ->next= newItem;
```

Complexity?

Insert a node after a given value

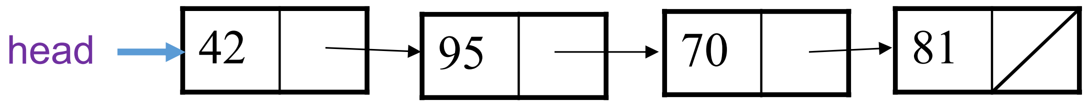
```
//head is the start of list
//value is the given value
//newItem is the node to be inserted

  for (struct node *here = head; here != null; here = here->next {
    if (here->data==value) {
          newItem ->next=here ->next;
          here ->next= newItem;
          exit loop;  //done
    } // if
  } // for
  // Couldn't insert--do something reasonable!
}
```

Deleting a node from an SLL

- In order to delete a node from a SLL, you have to change the link in its *predecessor*
- This is slightly tricky, because you can’t follow a pointer backwards

Deleting a node from an SLL

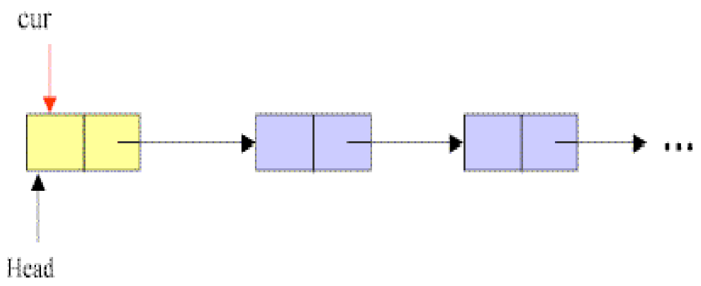
- Deletion can be done
  - At the first node of linked list.
  - At the end of a linked list.
  - Within the linked list.

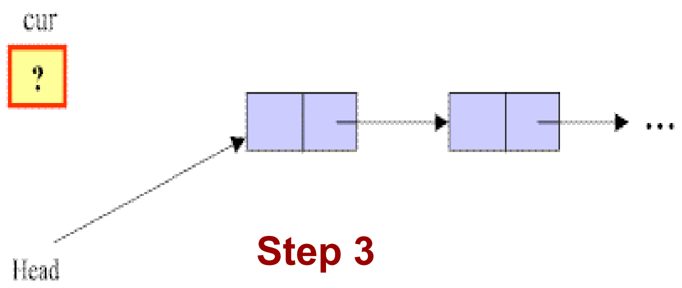


Delete First

- Step1. Initialize the pointer *cur* point to the first node of the list.
- Step2. Move the pointer *head* to the second node of the list.
- Step3. Release the memory of the node that is pointed by the pointer *cur*.

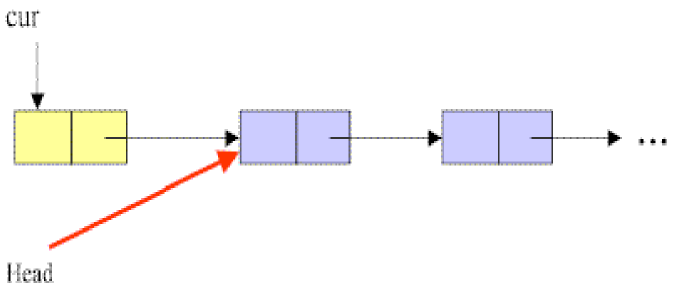
Step 1





Step 3

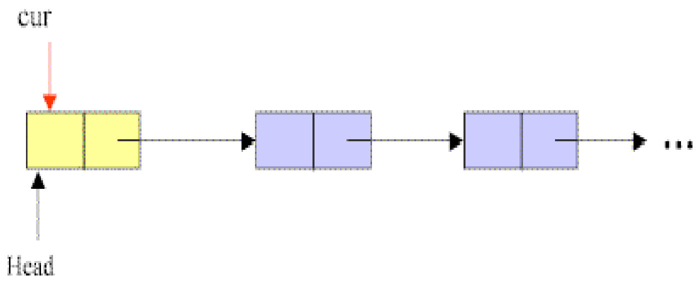
Step 2

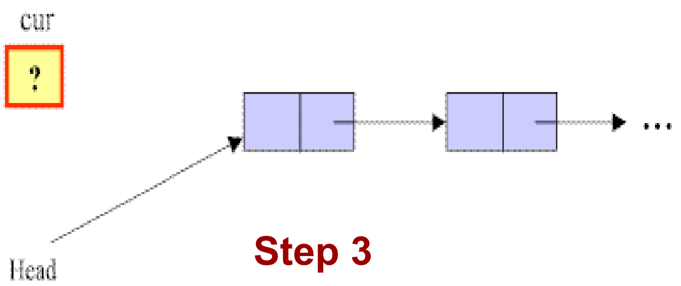


Delete First

- Step1. Initialize the pointer *cur* point to the first node of the list.
- Step2. Move the pointer *head* to the second node of the list.
- Step3. Release the memory of the node that is pointed by the pointer *cur*.

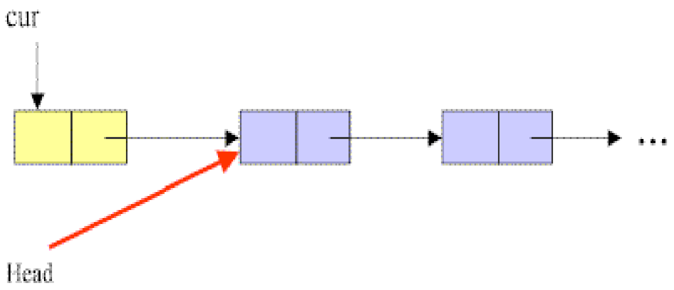
Step 1





Step 3

Step 2

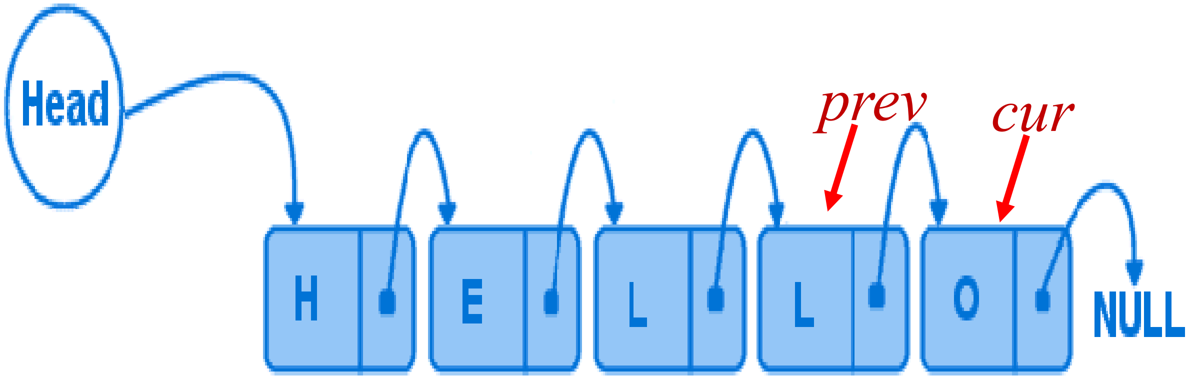


```
struct node* curr;
if (head ==NULL)  //error handling
curr =head;
head=head=->next;
free (curr);
```

Complexity?

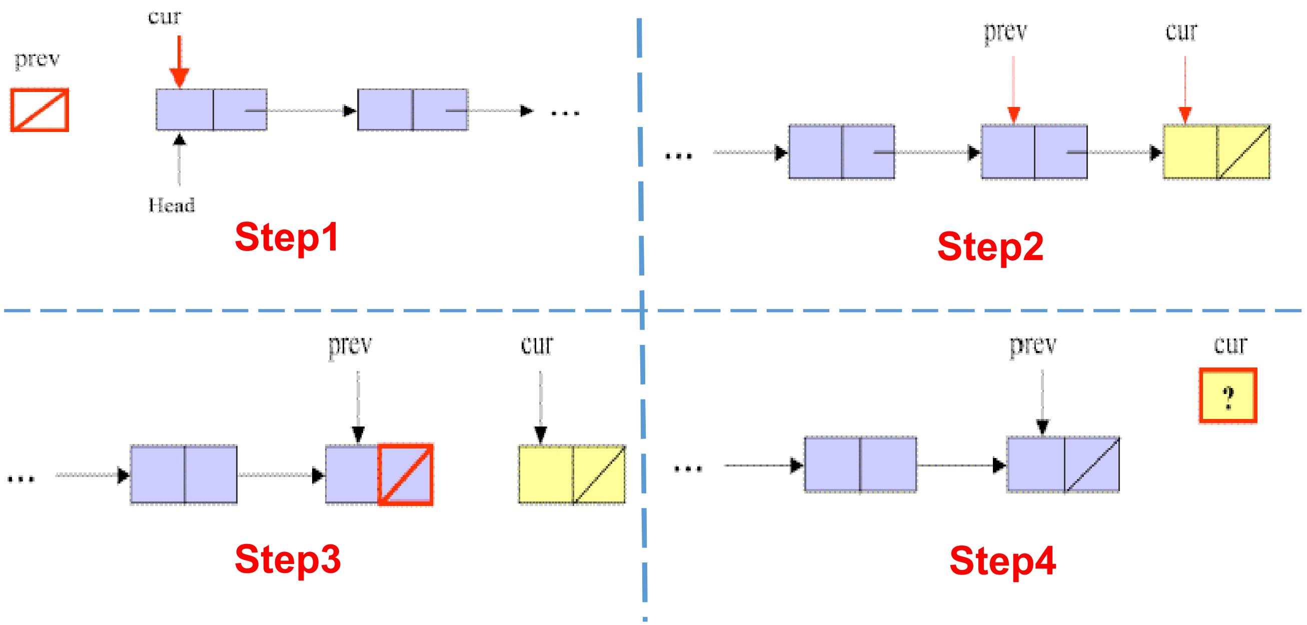
Delete Last

- To **delete** the last node in a linked list, we use a local variable, *cur*, to point to the last node. We also use another variable, *prev*, to point to the second last node in the linked list.



Delete Last

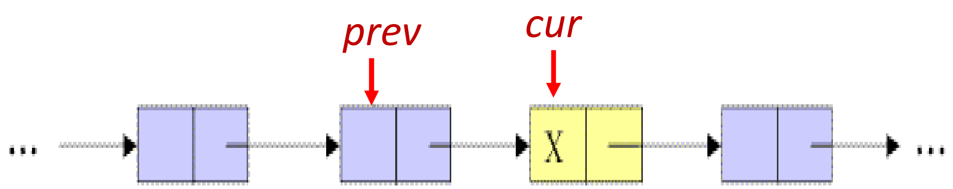
- Step1. Initialize *cur* = first node of the list, while the pointer *prev* = NULL.
- Step2. Traverse the entire list until the pointer *cur* points to the last node of the list: assign cur to prev and then advance cur
- Step3. Set prev->next = NULL
- Step4. Release the memory of curr



```
If head==NULL
      //error handling and return

cur =head;
prev=NULL;
while cur->next !=NULL
      prev =cur;
      cur =cur->next;

if (prev) prev ->next=NULL;
else head= NULL;

free (cur);
```

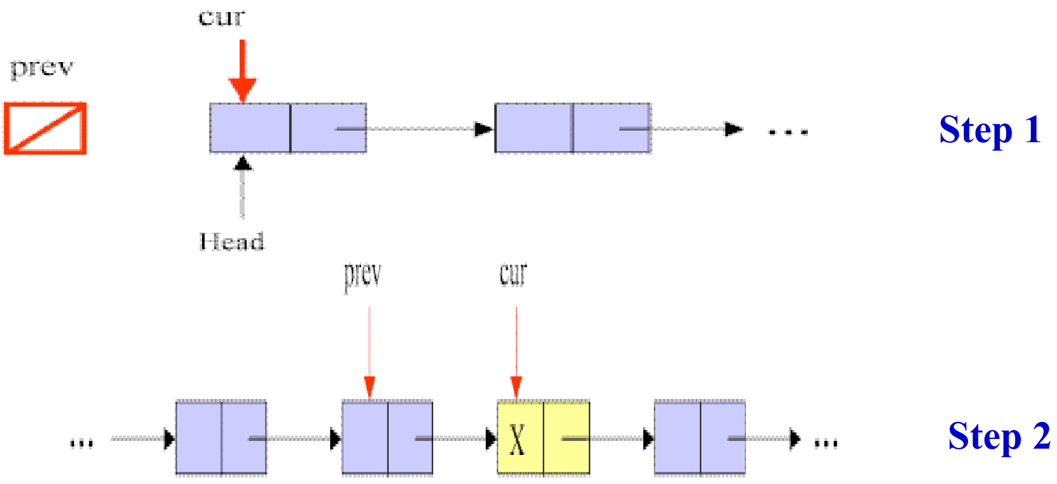
Delete Any

- To **delete a node** that contains a particular value *x*
  - use *cur* to point to the node with value *x*, and *prev* to point to the previous node



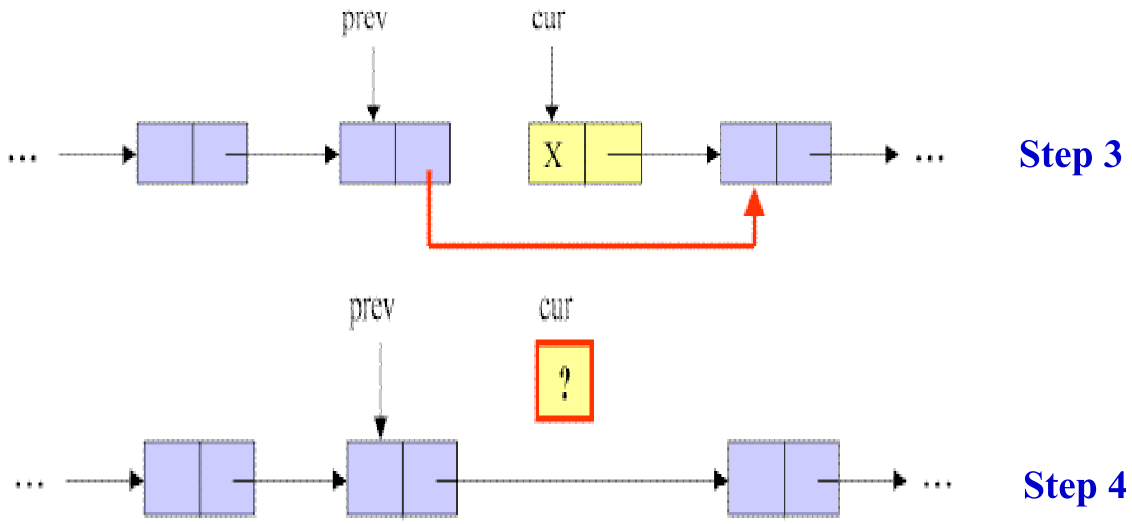
Delete Any

- Step1. Initialize *cur* = the first node of the list, while the pointer *prev* =NULL
- Step2. Traverse the entire list until *cur->data=x* and *prev* points to the previous node: assign cur to prev and then advance cur
- ……..

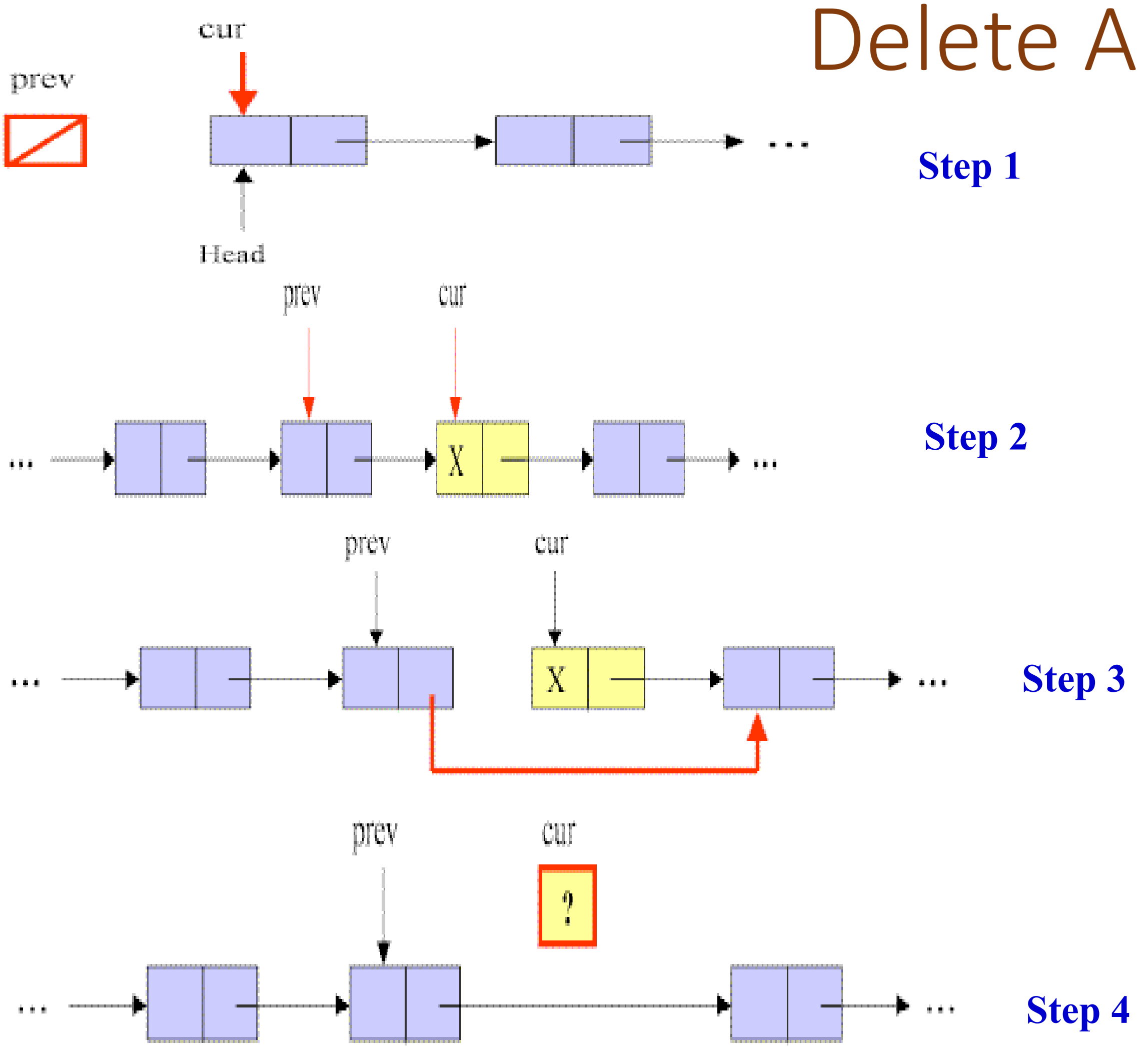


Delete Any

- …….
- Step3. Link the node pointed by pointer *prev* to the node after the *cur*'s node.
- Step4. Release the memory of the node pointed by *cur*.

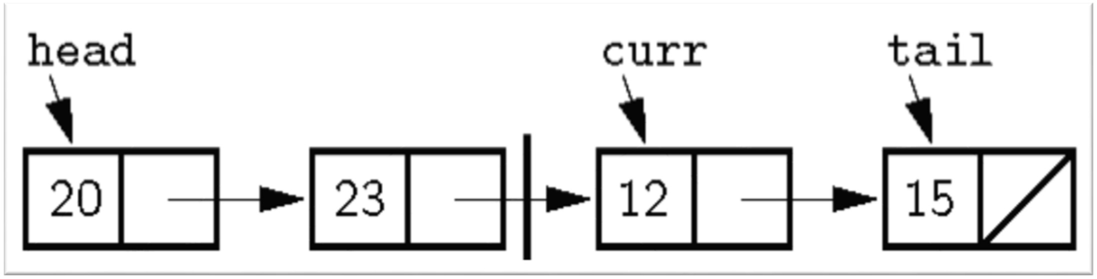


Delete Any

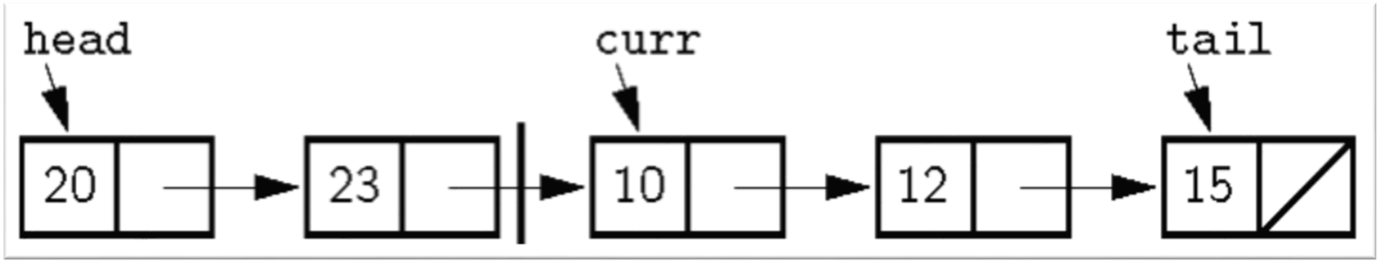


```
If head==NULL
     //error handling and return

cur =head;
prev=NULL;
while (cur)
     if curr->value ==x
        break;
     prev =cur;
     cur =cur->next;

if (cur==NULL) //NOT Found, Return

If (prev) prev ->next=curr->next;
else head= head->next;

free (cur);
```

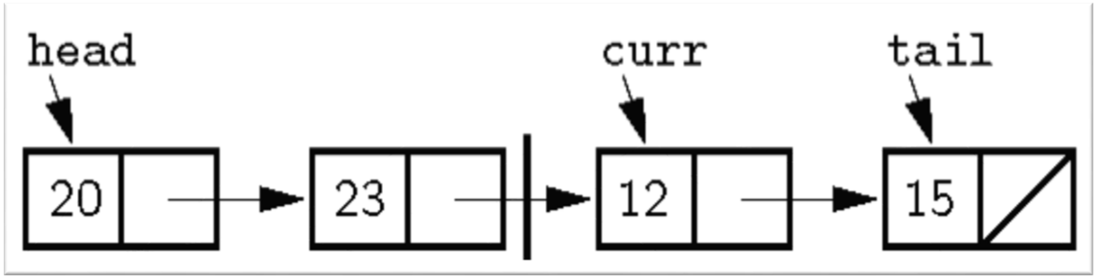
Designated head, tail, current nodes



Difficult to insert at curr, we don’t have access to previous node



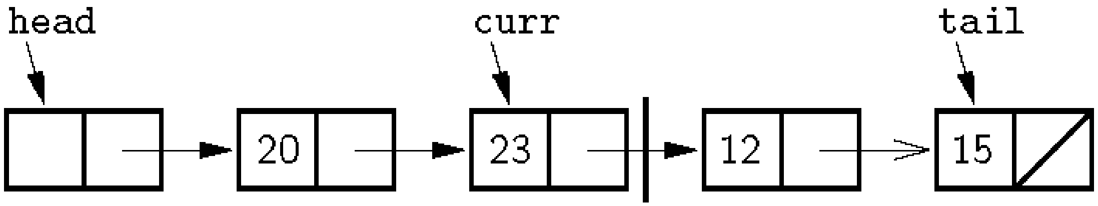
Designated head, tail, current nodes

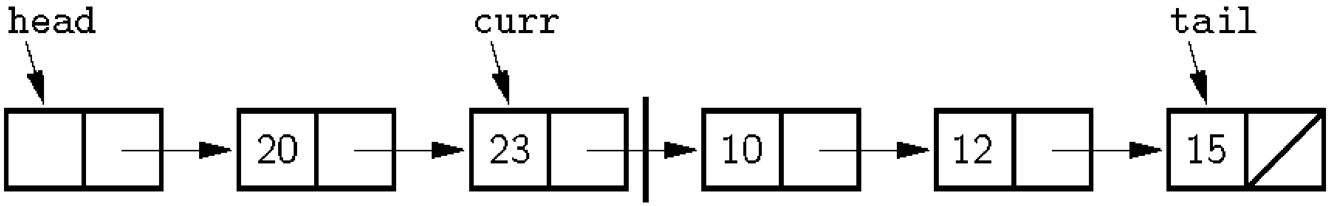


Other special case:

Empty list: no head, tail, curr

Designated head, tail, current nodes

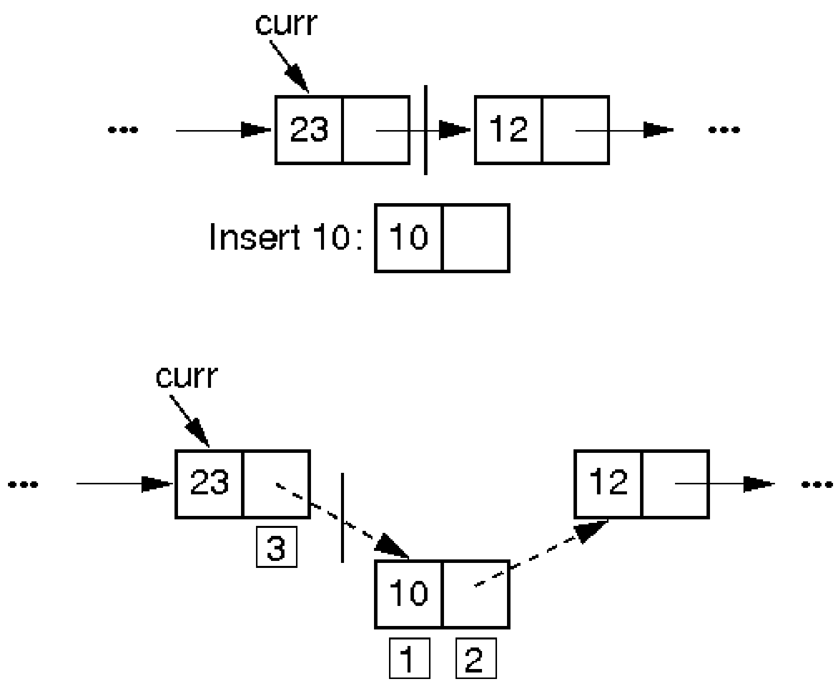




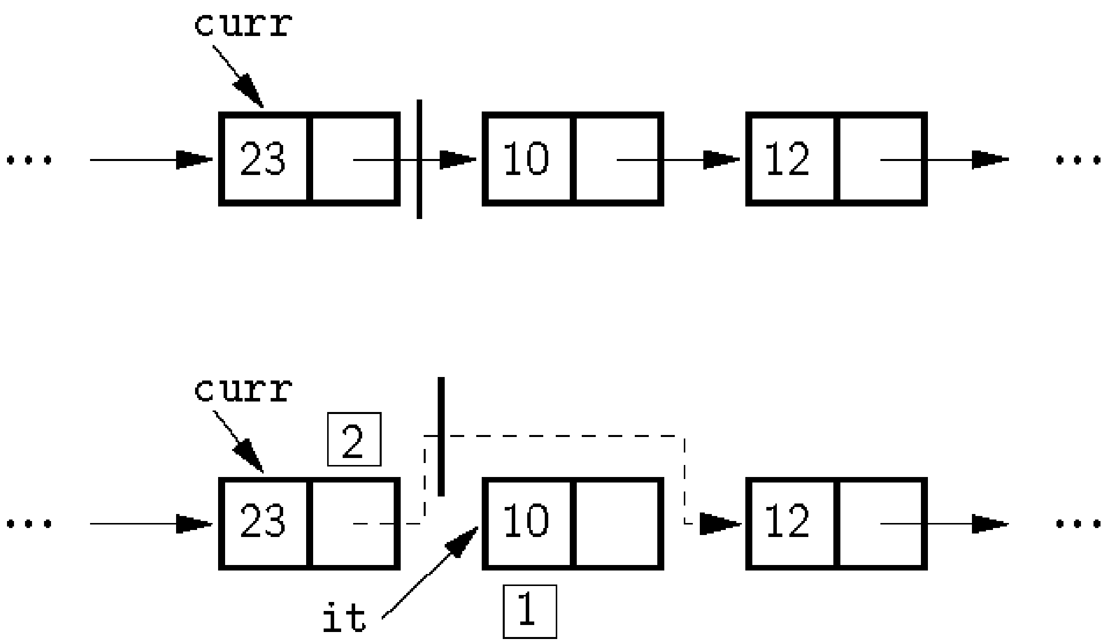
Two modification:

1. a dummy header node
2. Curr point to the previous of the desired node

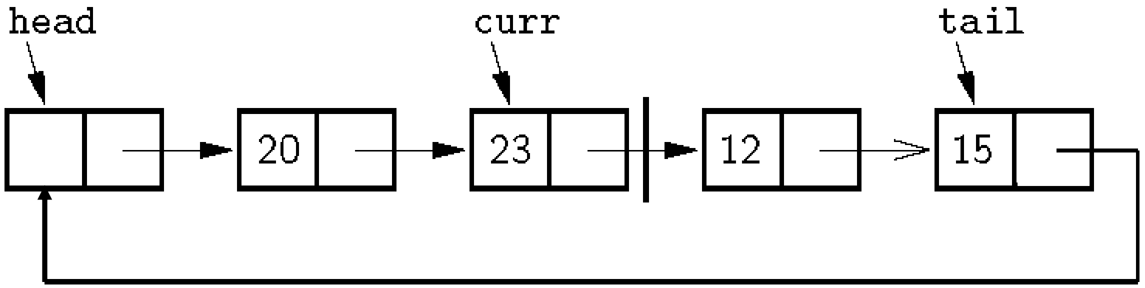
Insertion



Removal



Circular linked list


head
curr
tail
20
23
12
15