Linked Lists

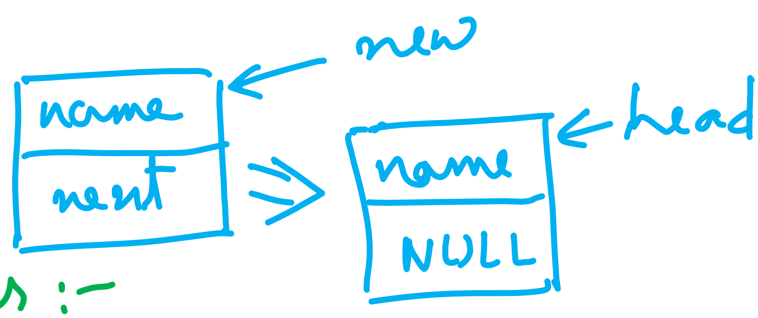
03/10/2021

```
struct person
{
    char name[20];
    struct person * next;
};
struct person * new;
struct person * head;
head = NULL;
```

node
=> a self-referential structure

Adding the first element of a linked list

```
new = (struct person *) malloc(sizeof(struct person));
new -> next = head;
head = new;
```



Insertion cases :-

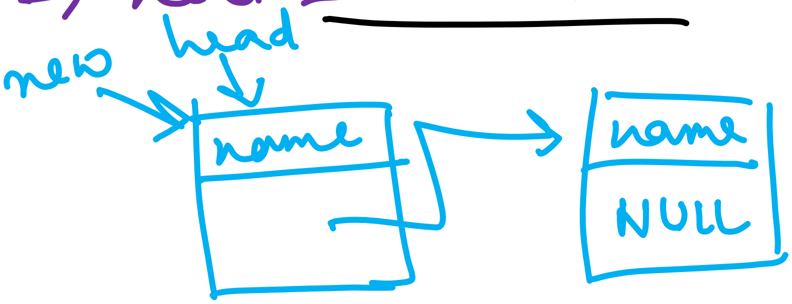
(a) Adding up node at the begining of the list

(b) Adding up node in an intermediate position.

(c) Adding up node at the last end of the linked list.

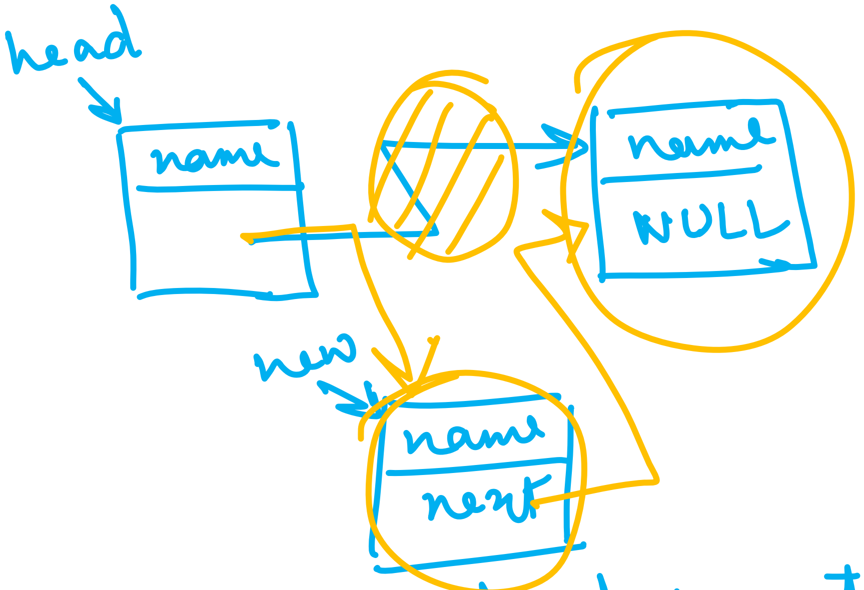
new = (struct person *) malloc(sizeof(struct person));

new → next = head;



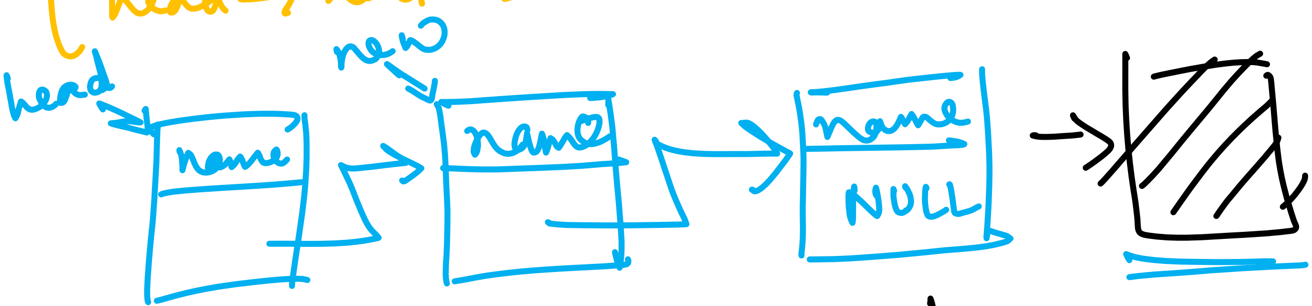
head = new;

new = (struct person *) malloc(sizeof(struct person));



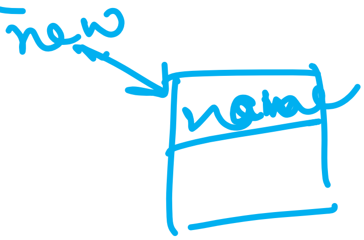
new → next = head → next;

head → next = new;



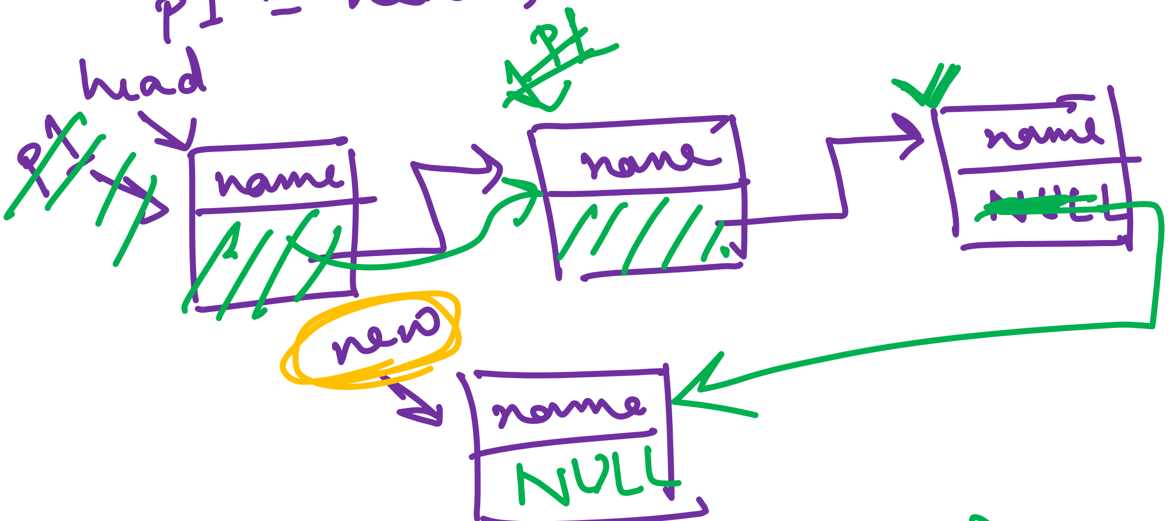
pseudo code : similar to program statements but not a complete program.

new = (struct person *) malloc(sizeof(struct person));



struct person * p1;

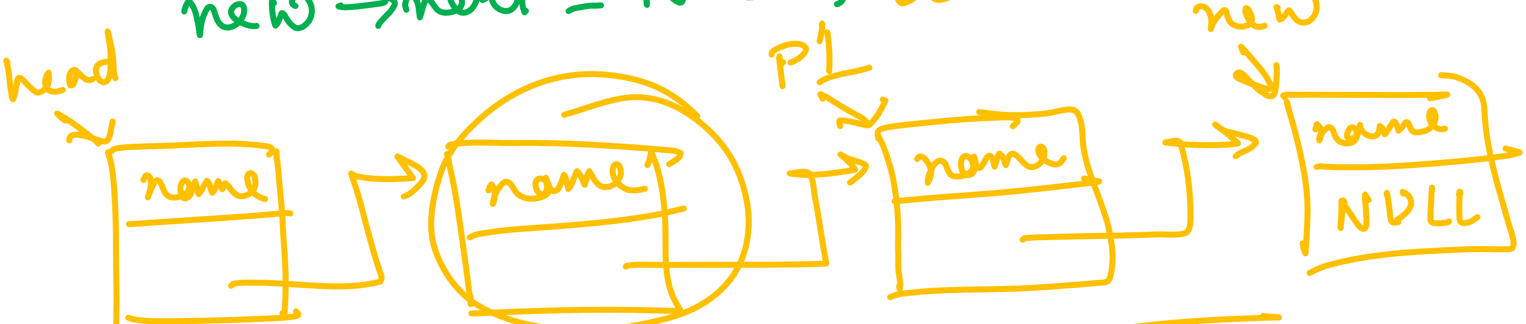
p1 = head;



while (p1 -> next != NULL)
{ p1 = p1 -> next; }

p1 -> next = new;

new -> next = NULL;



We have created a linked list with 4-nodes with insertions at the

(a) begining
(b) intermediate position
(c) at the end

a single-linked list / singly linked list.

⇓

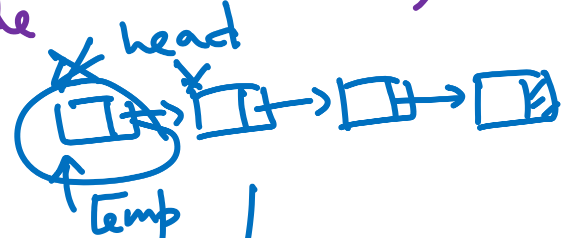
uni-directional linked list.

— x —

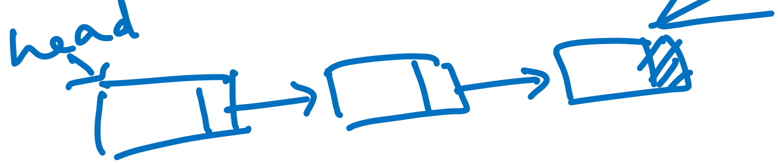
Deletion of a (element) node from a linked list

07/10/2021

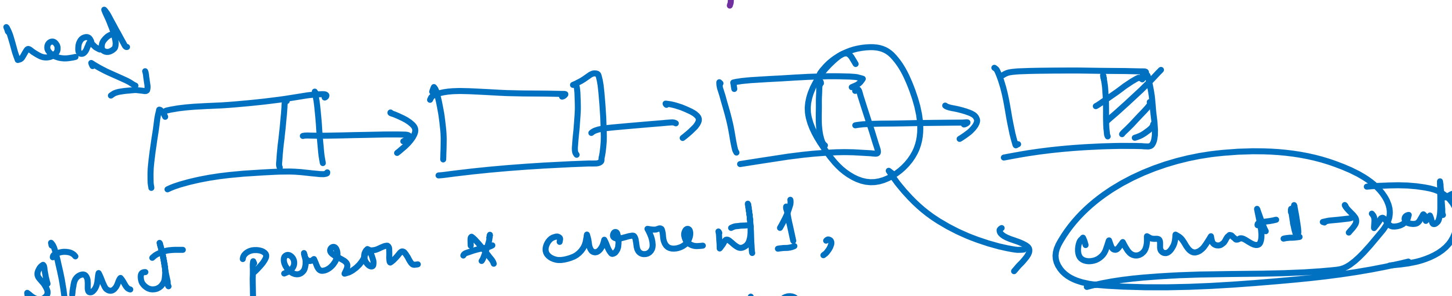
Case 1 :- Deletion of the 1st element / starting node



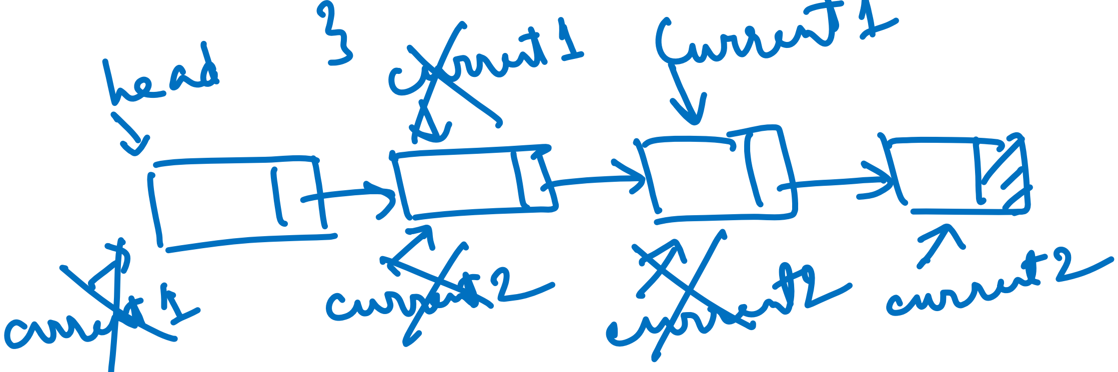

```
struct person * temp;
temp = head;
head = head->next;
free(temp); // frees the dynamically allocated memory/node pointed by temp
```



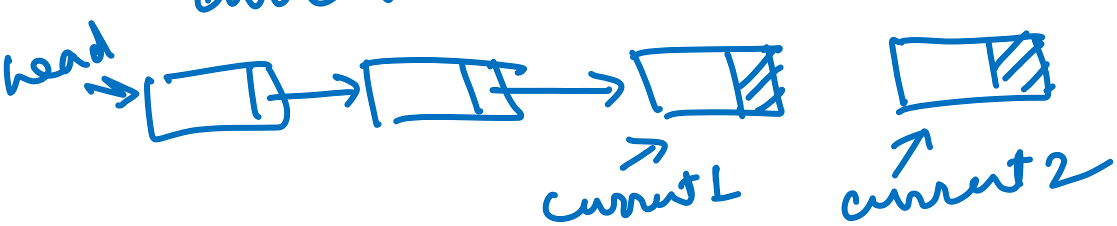
Case 2:— Deletion of the last node/element



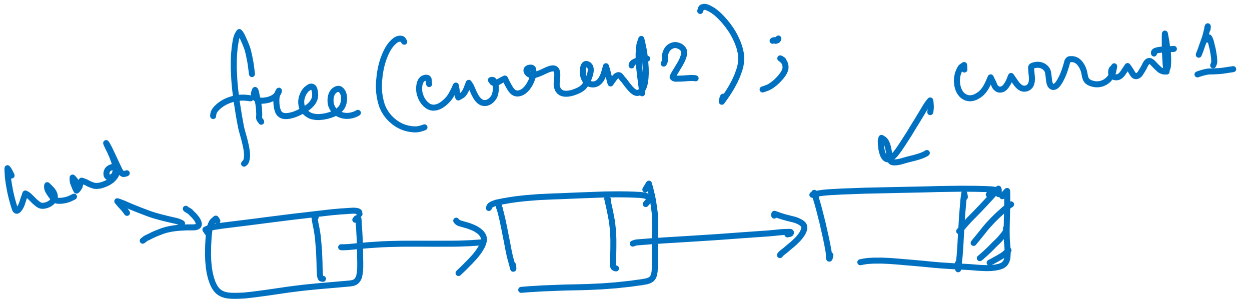

```
struct person * current1,
struct person * current2;
current1 = head;
current2 = current1->next;
while (current2->next != NULL)
   {
      current1 = current2;
      current2 = current1->next;
   }
```



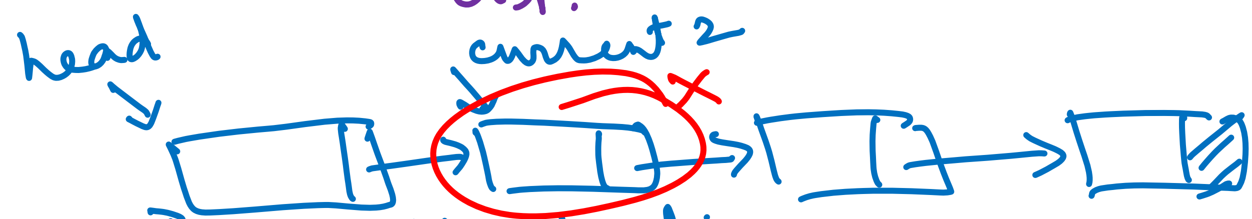
current1->next = NULL;



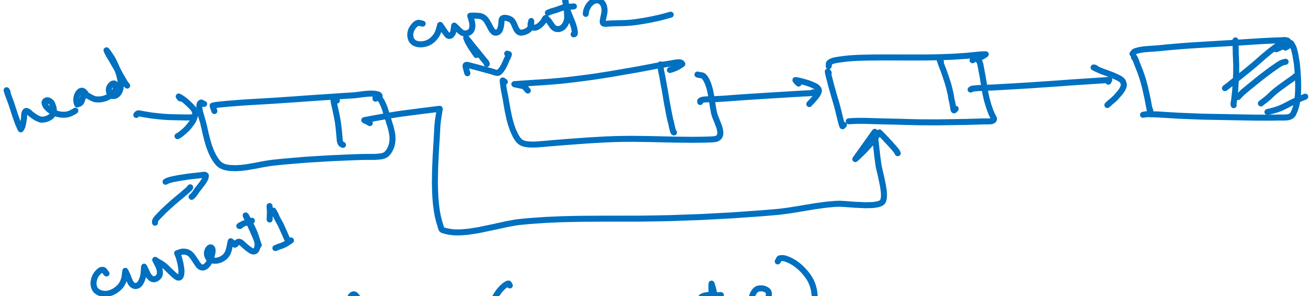
free(current2);



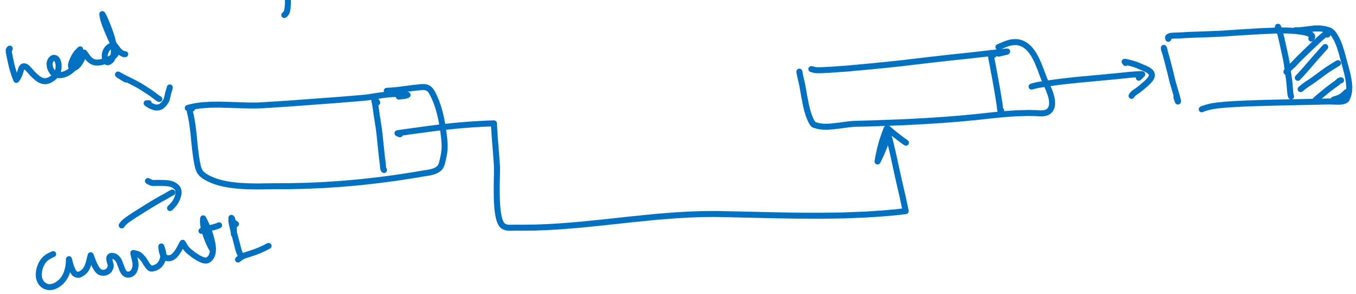
Case 3 :- Deletion of an intermediate node/element from a linked list.



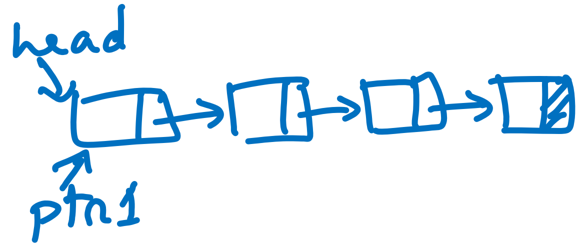

```
current1 = head;
current2 = current1->next;

current1->next = current2->next;
```




```
free(current2);
```

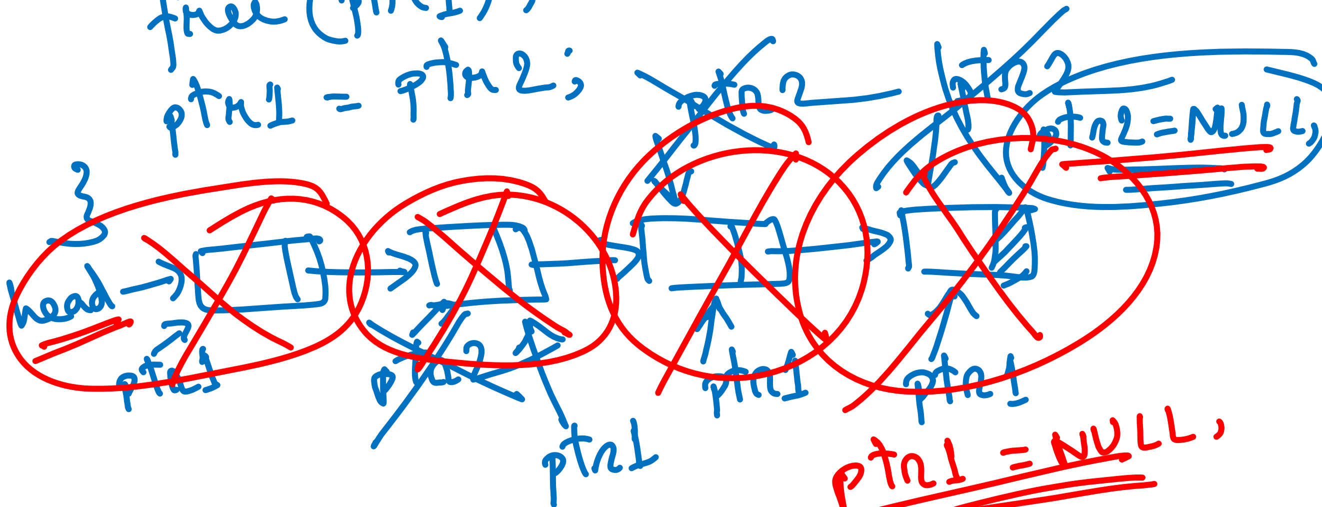


Case 4 :- Deletion of the whole linkedlist




```
struct person * ptr1;
struct person * ptr2;
ptr1 = head;
while (ptr1 != NULL)
```

{
ptr2 = ptr1 → next;
free(ptr1);
ptr1 = ptr2;
}

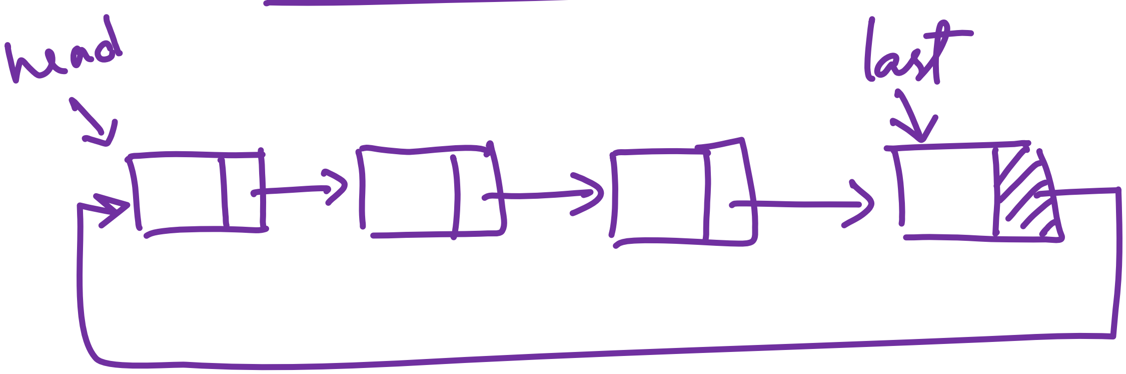


head = NULL;

— x —

Circular Linked Lists

26/10/2021



last → next = head;

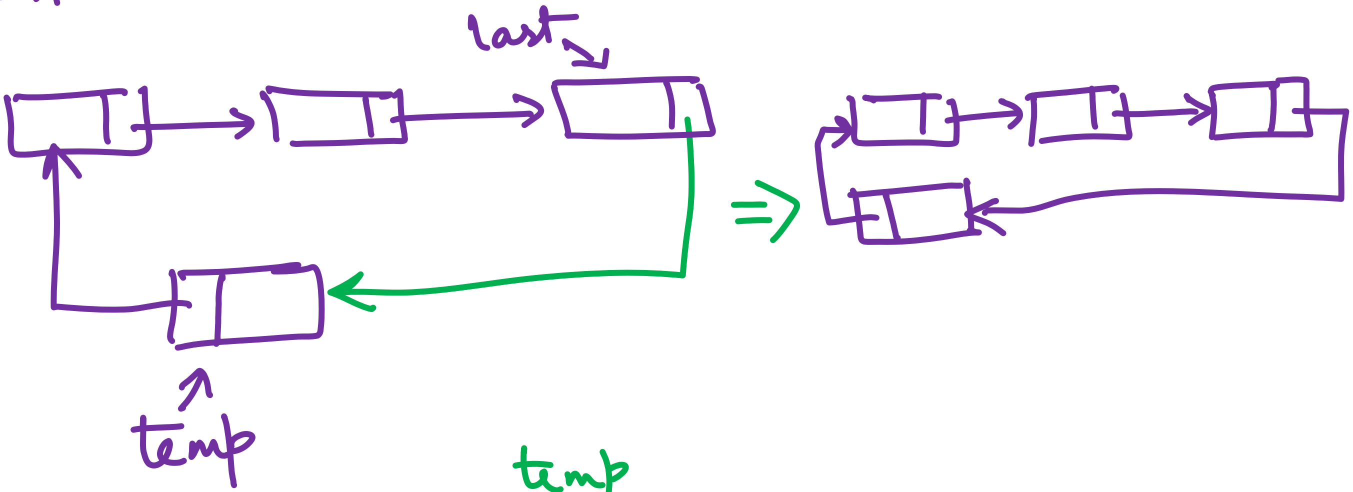
Insertion Cases:

(1) Insertion at begining
(2) Insertion in between

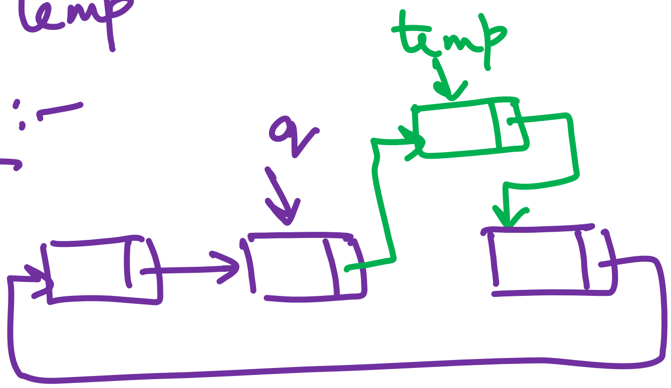
Case-1 :-

temp = (struct node *) malloc(sizeof(struct node));

```
temp -> next = last -> next;
Last -> next = temp;
```



Case 2 :-




```
temp = (struct node *) malloc(sizeof(struct node));

struct node
{
    int data;
    struct node * next;
};

temp -> next = q -> next
q -> next = temp;
```
✓

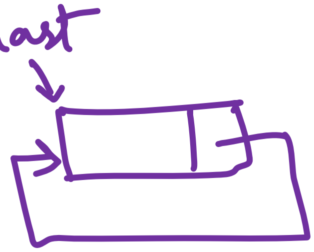
Deletion within Circular Linked List

1) If the list has only one node
2) Node to be deleted is the 1st node.

3) Deletion in-between

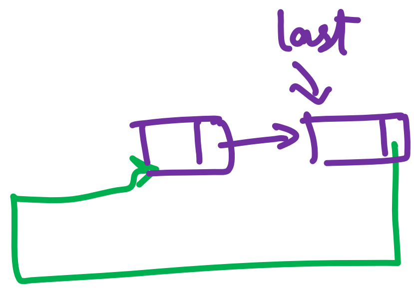
4) Node to be deleted is the last node

<u>Case - 1</u> :



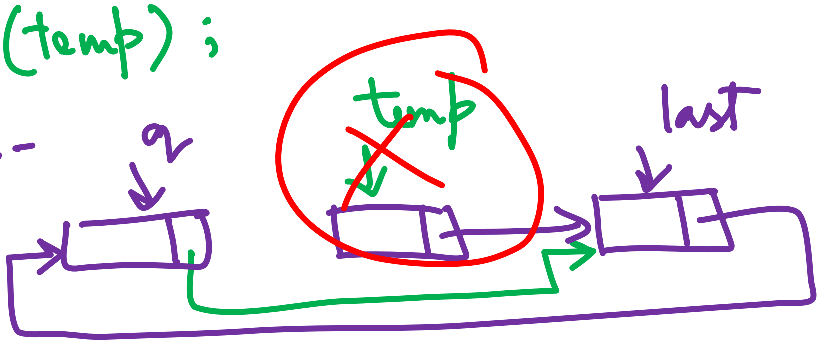

```
if (last->next = last)
{   temp = last;
    last->next = NULL;
    free(temp);
}
```

<u>Case - 2</u> :-

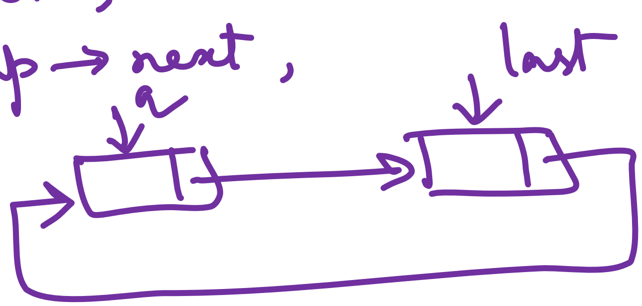



```
q = last->next;
temp = q;
last->next = q->next;
free(temp);
```

<u>Case - 3</u> :-

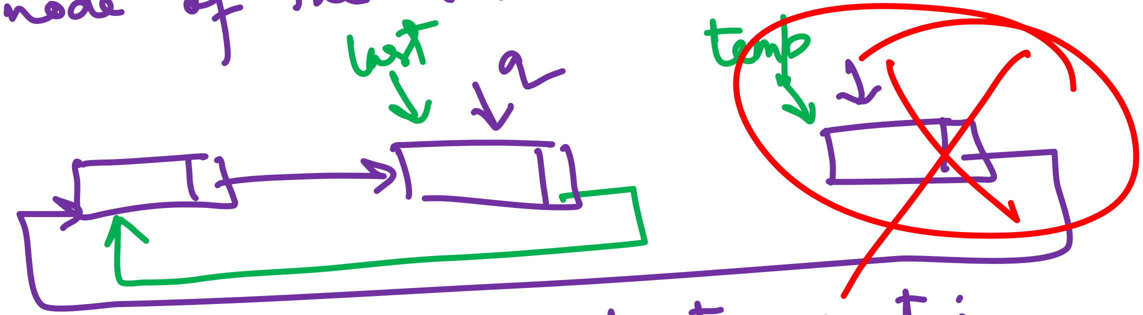



```
temp = q->next;
q->next = temp->next,
free(temp);
```

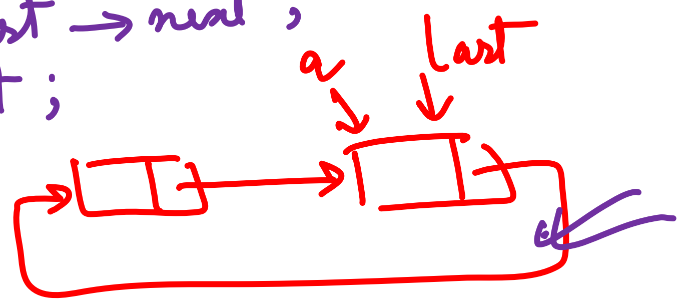


Case 4 :-

Let us consider q points to the previous node of the last node




```
q -> next = last -> next ;
temp = last ;
last = q ;
free (temp) ;
```

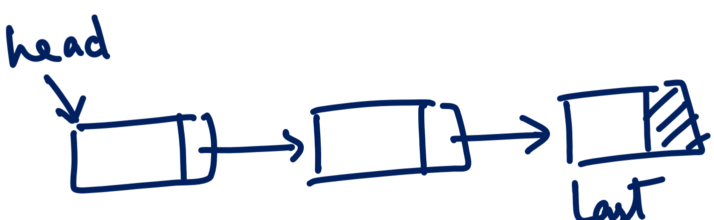


— X —

Double Linked List

28/10/2021

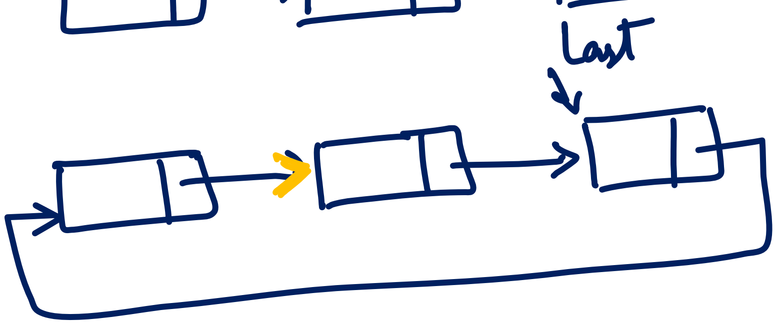
Type 1 :



"Singly" Linked List

↳ Single direction

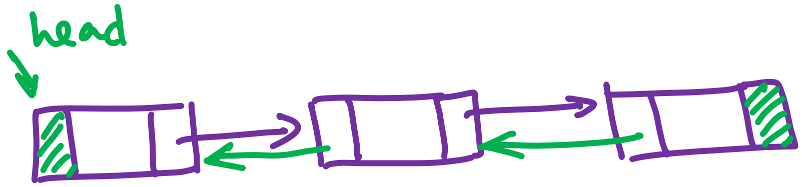
Type 2 :



Circular Linked List (also a type of Singly Linked List)

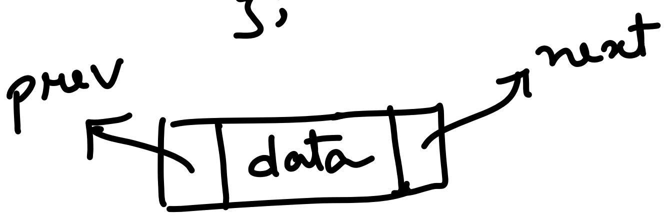
Double linked list
or
Doubly linked list

} Bi-directional Traversal within a linked list




```
struct node
    {
        struct node * prev,
        int data;
        struct node * next;
    };
```

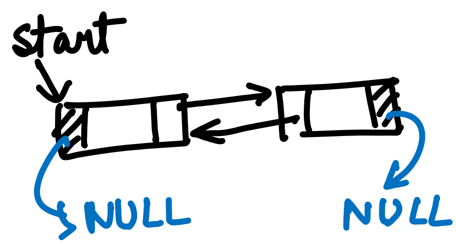
Node becomes bulkier
More bytes needed.



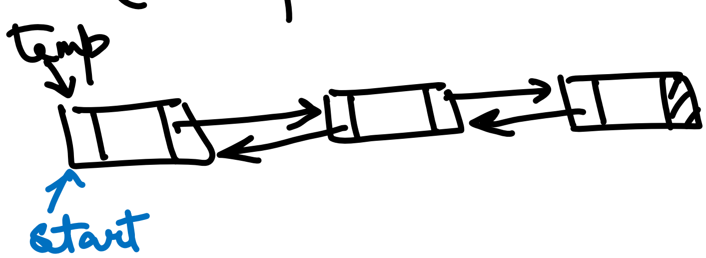
A. Insertion within a doubly linked list

(a) Insertion at begining
(b) Insertion in between
(c) Insertion at the end

Case - (a) :

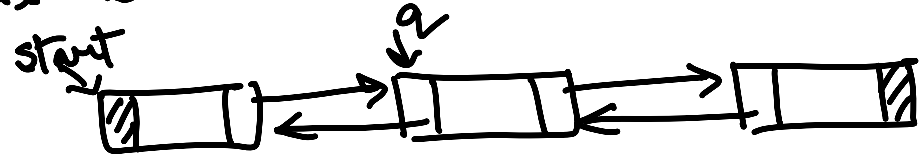



```
struct node * start;
struct node * temp;
temp = (struct node *) malloc (sizeof(struct node ));
temp -> next = start;
start -> prev = temp,
start = temp;
start -> prev = NULL,
```



Case - (b) :

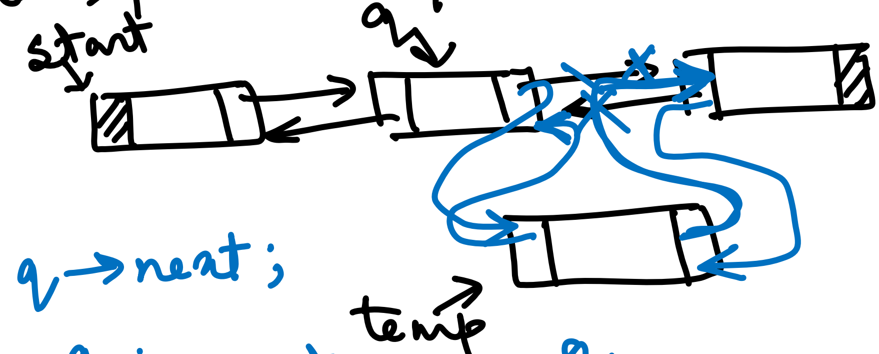
Let there be a pointer 'q' after which a new node shall be inserted then,



struct node * temp ;

temp = (struct node *)malloc(sizeof(struct node));

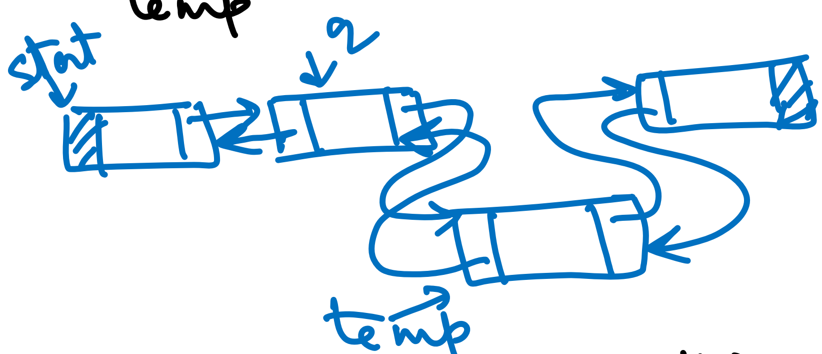
q → next → prev = temp ;



temp → next = q → next ;

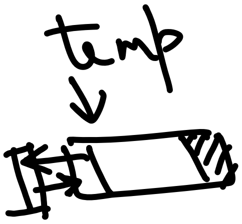
temp → prev = q ;

q → next = temp ;



Case - (c) : it is a sub-case of case - (b).

if (q → next == NULL)
{
temp → next = q → next ;
temp → prev = q ;
q → next = temp ;
}

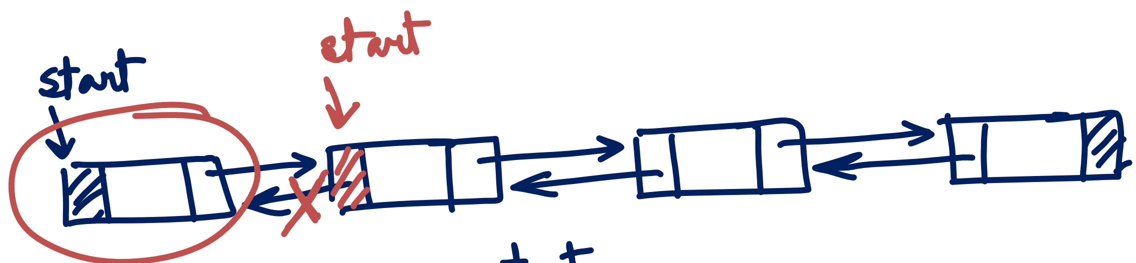


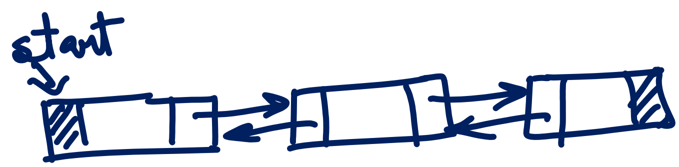
B. Deletion within a Doubly linked list

dated - 29/10/2021

(a) Deletion at the begining
(b) Deletion in-between
(c) Deletion at the end

Case-(a) :

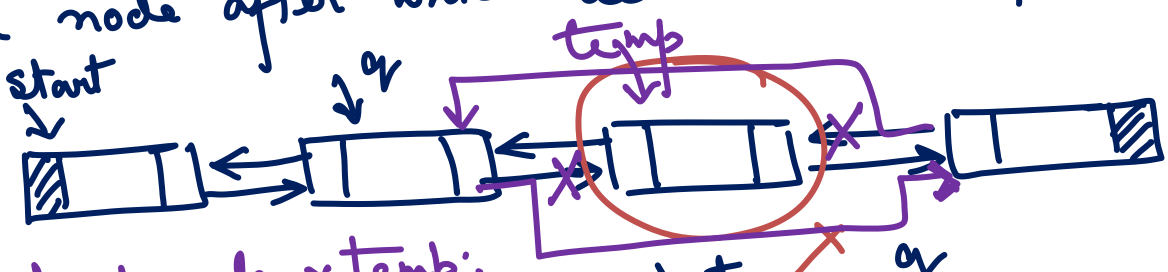


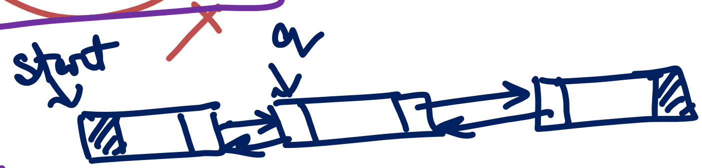



```
struct node * temp;
temp = start;
start = start->next;
start->prev = NULL;
free(temp);
```

Case-(b) :

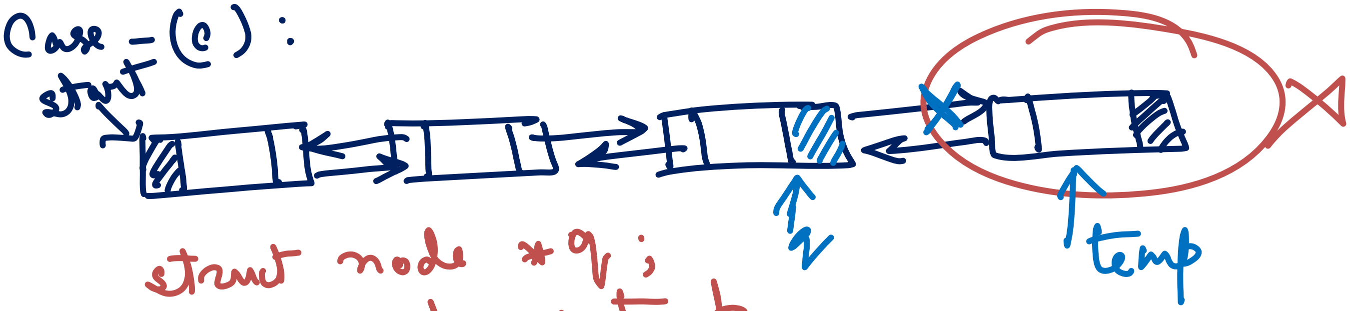
Let us assume, a struct node * q points to the node after which deletion takes place.






```
struct node *temp;
temp = q->next;
temp->next->prev = q;
q->next = temp->next;
free(temp);
```

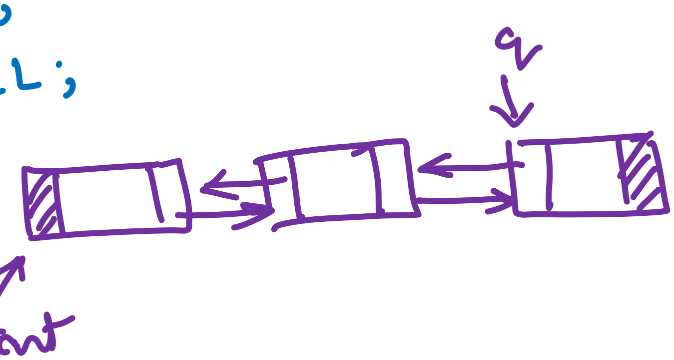
Case-(c):



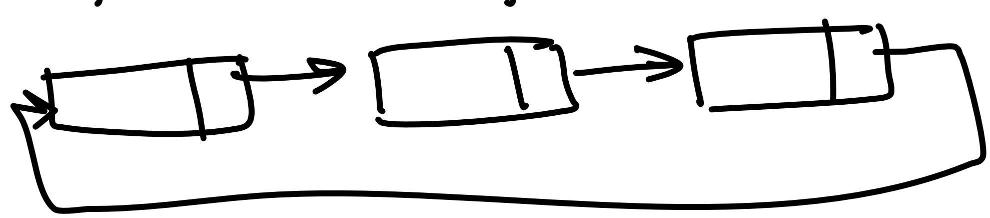
struct node *q;
struct node *temp;

Let us assume, that q points to the node after which the deletion takes place i.e. q points to the second-last node.

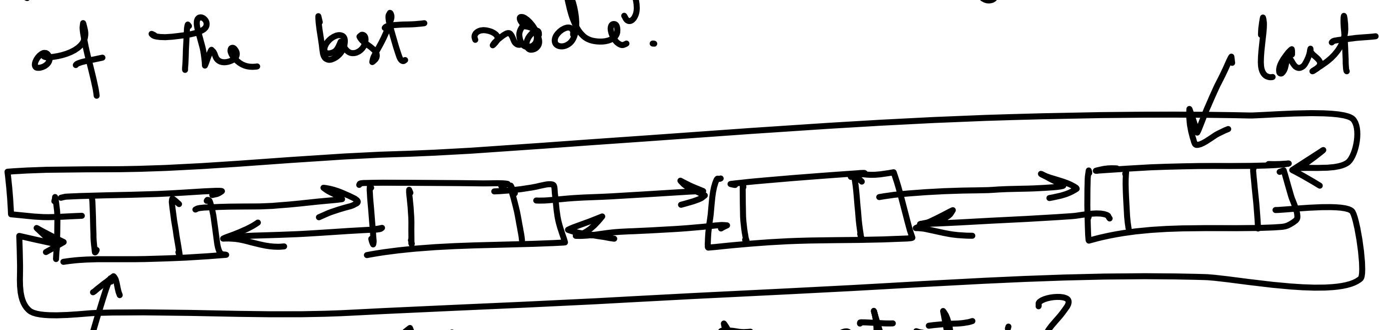
```
q = start;
while (q->next->next != NULL)
    q = q->next;

temp = q->next;
q->next = NULL;
free(temp);
```



Singly linked list can be circular if the last node points to the first node.



Similarly, Doubly linked list can also be converted into a Circular-Doubly linked list by making the last node's next field holding the address of the first node and the first node's prev field holding the address of the last node.



{ last → next = start ;
start → prev = last ; }

— x —

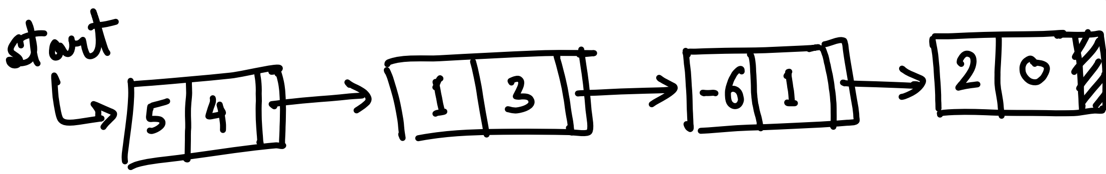
Polynomial Arithmetic With Linked List

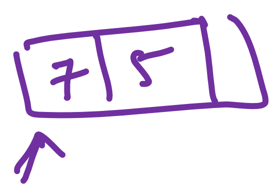
Dated on 31/10/2021

$5x^4 + x^3 - 6x + 2$

$7x^5$

$4x^2$





temp

```
struct node
{
    int coeff;
    int exp,
    struct node * link;
};
```

```
if (start == NULL || start → exp < temp → exp)
{
struct node *temp;
temp = (struct node *) malloc (sizeof(struct node));
scanf(" ... ", temp → exp);
}
```

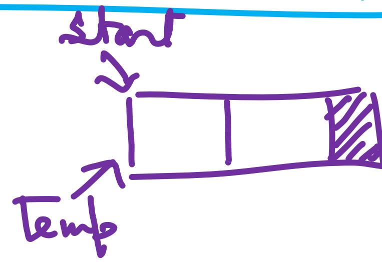
scanf(" - - - - - ., temp → coeff); }

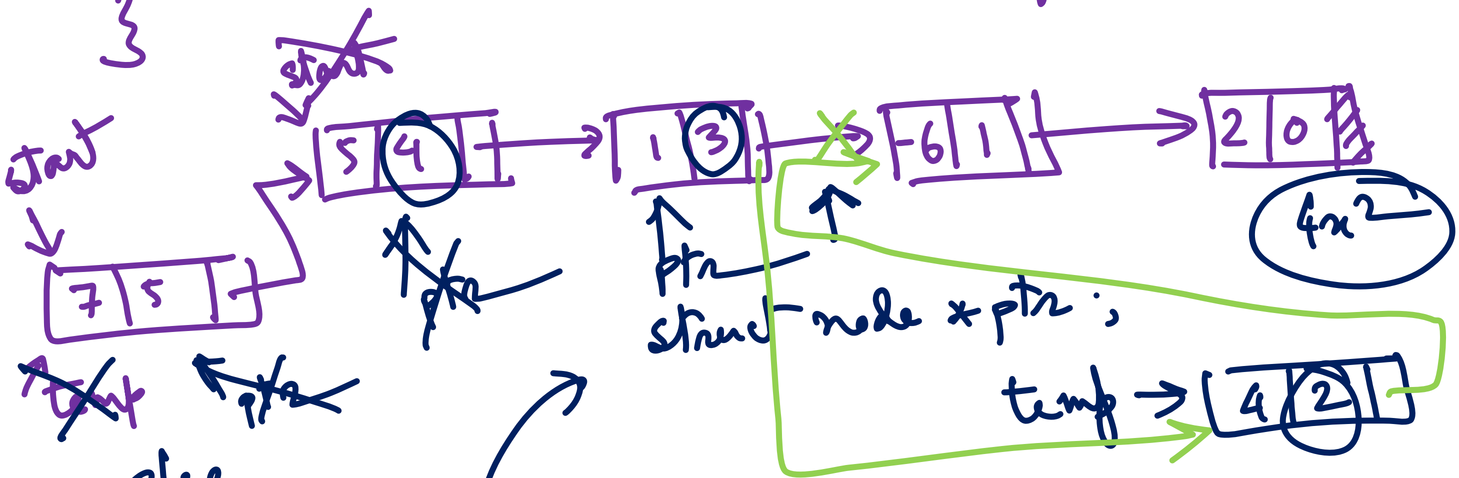
{

temp → link = start;

start = temp;

}





else

{

ptr = start;

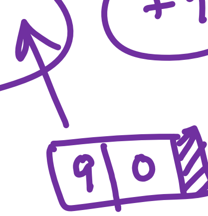
while (ptr → link != NULL && ptr → link → exp > temp → exp)

ptr = ptr → link;

temp → link = ptr → link;

ptr → link = temp;

$7x^3 + 2x^2 - 3x$ $+9$



if (ptr → link == NULL)

temp → link = NULL;

}