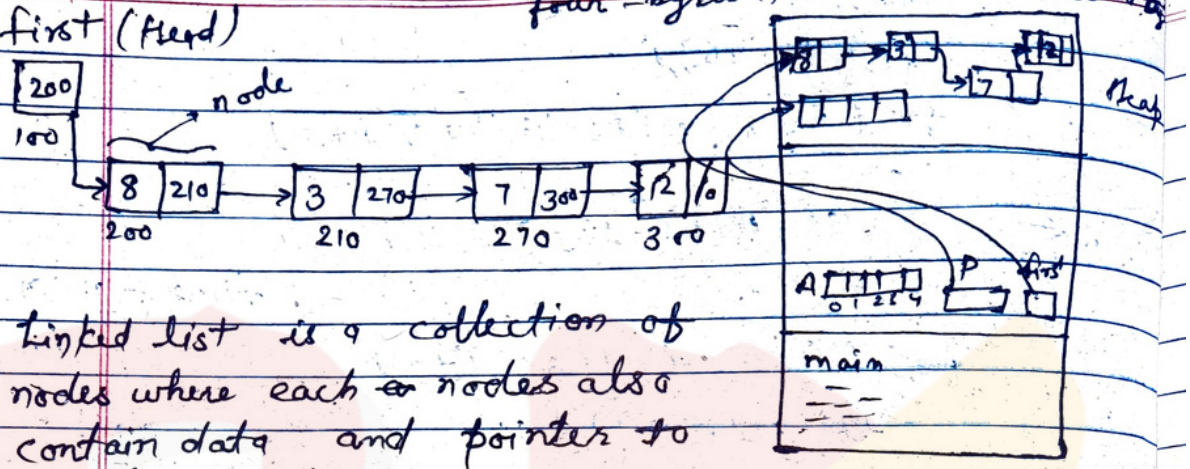
17/04/24

Linked list

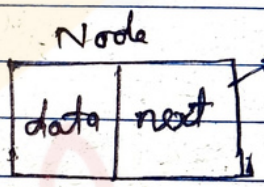
* In any compiler, if integer takes 2 bytes then pointer will also take 2 bytes similarly if integers four bytes then pointer also 4 by



(Linked list is a collection of nodes where each ~~or~~ nodes also contain data and pointer to next node.)

* In linked list, all nodes are not side by side.
* Its continuity is maintain through like links i.e addresses.

Node



→ Creating a node -

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

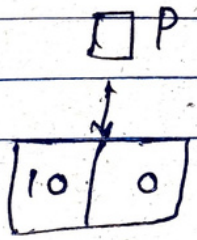
C

```
p = new node;
```

C++

```
p -> data = 10;
p -> next = 0;
```

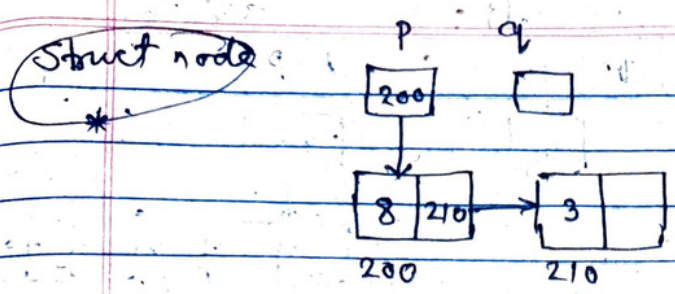
} accessing of node



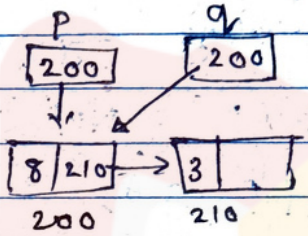

```
struct node
{
    int data;
    struct node * node;
}
```

Node having a pointer of its own type so such structure is known as self-Referential.

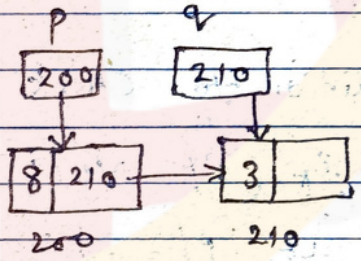
struct node *P, *q



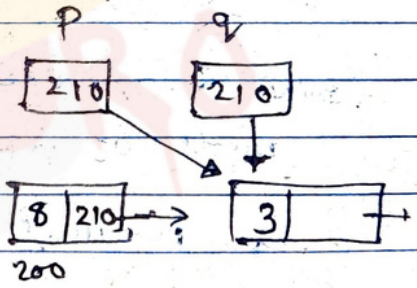
i) q = P; →



ii) q = P → next; →



iii) P = P → next; →



Not pointing any node

① p

0

Pointing a node

② p

200

Condition ~~for~~ to check a pointer, not pointing any node -

1) if (P = = NULL)
2) if (P = = 0)
3) if (!P)

Condition to check a pointer, pointing a node -

i) if (P != NULL)
ii) if (P != 0)
iii) if (P)

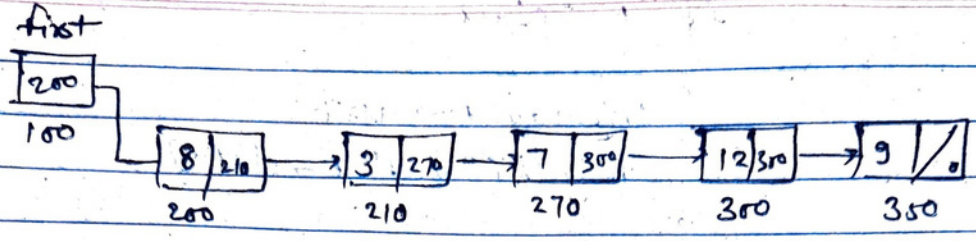
Condition to check a node, pointing another node or not.

i) if (P → next = = NULL) → It means this not pointing any node that is it's last node

ii) if (P → next != NULL) → Means that, this node also pointing a ~~pointer~~ another node.

Date
Page

* Display linked list

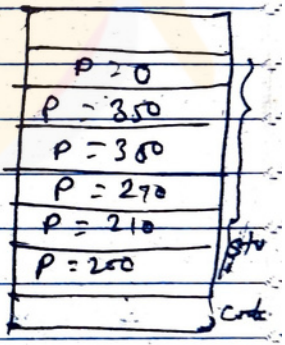


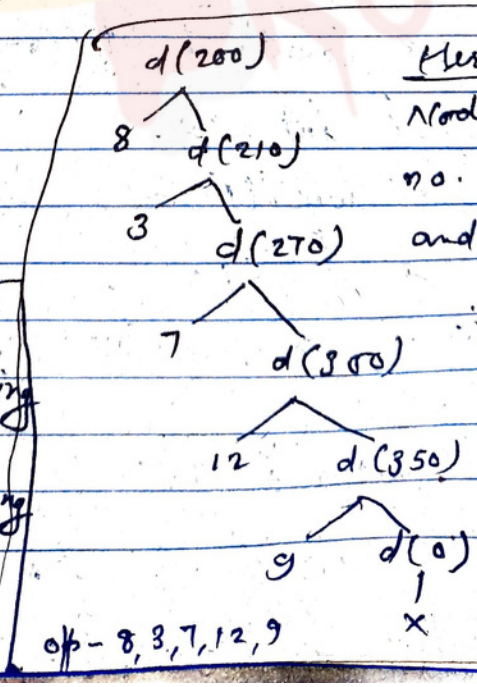

```
{ struct node *P = first;
  while (P != NULL)
  {  printf ("%d", P->data)
     P = P->next;
  }
```

18/04/22

* Recursive Display of linked list

```
void Display (struct node * P)
{  if (P != NULL)
   {  printf ("%d", P->data);
      Display (P->next);
   }
}
main()
Display (first);
```





Here
Node = 5
no. of calls = 6
and stack = 6
time - O(n)
space - O(n)

* If any recursive function as written for traversing a linked list just once then it will making (n+1) calls for the entire linked list and also stack size will (n+1)

* Void Display (struct node * P)

{ ~~if (P != Null)~~

{ Display(P → next)

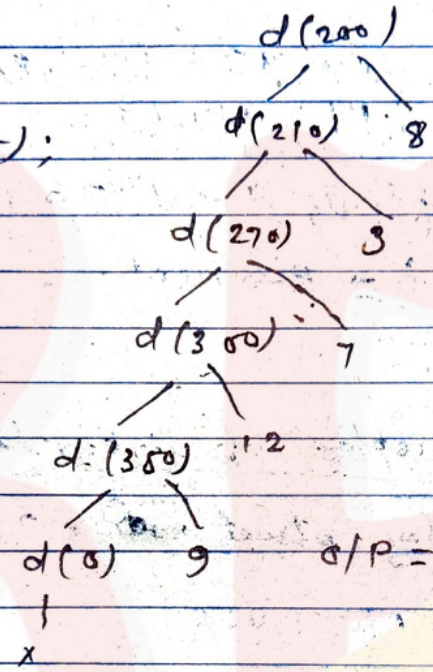
printf("%d", P → data)

}

}

main()

Display(first);



time - O(n)

space - O(n)

d/P = 9 12 7 3 8

* Counting Nodes in a linked list -

int count(struct Node * P)

{ int c=0;

while(P != 0)

{ c++;

P = P → next;

}

return(c);

}

Iterative

time - O(n)

space - O(1) → constant

↓

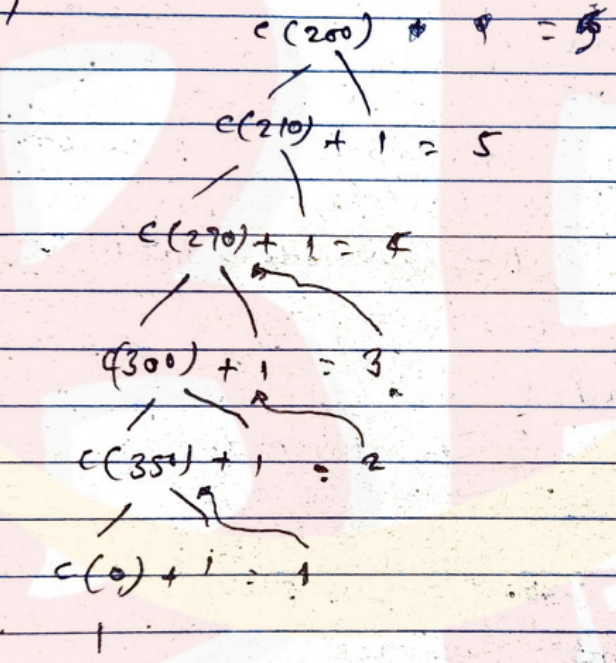
It has only an pointer and an integer.

(It is also depend upon data members and variables from

Date
Page

Recursive –

```
int count ( struct Node *P)
{
  if (P==0)
    return 0;
  else
    return count(P->next)+1;
}
```



time - o(n)

space - o(n)

Other way –

```
int count (struct Node * P)
{ int x=0;
  if (P)
  { x = count(P->next);
    return x+1;
  }
  else
    return x;
}
```

27/04/22 (creating) Implimentation of linked list

Date
Page

```
void main()
{
    struct node
    int data;
    struct node * next;
};
struct node *head, *newnode, * temp;
    head = 0;
    int choice;
while (choice)
{   newnode = (struct node *) malloc (sizeof (struct node));
    printf ("Enter data");
    scanf ("%d", &newnode -> data);
        newnode -> next = 0;
    if (head == 0);
    { head = temp = newnode; }
    else
    { temp -> next = newnode;
        temp = newnode;
    }
printf ("Do you want to continue press 0 or 1");
    scanf ("%d", &choice);
}
temp = head;
while (temp != 0)
    { printf ("%d", temp -> data);
    temp = temp -> next;
}
}
```

* Sum of all elements in a linked list

Iterative

```
int Add (struct node * P)
{  int sum = 0;
   while (P)
   {  sum = sum + P -> data;
      P = P -> next;
   }
```

Recursive

```
int Add (struct node * P)
{  if (P == 0)
      return (0)
   else
   return Add (P -> next) + P -> data
}
```

03/05/22

Max^m element in a linked list -

Iterative

```
int main (struct node * P)
{  int m = -32768 ; MIN_INT ;
   while (P)
   {  if (P -> data > m)
         m = P -> data;
      P = P -> next;
   }
   return (m);
}
```

Recursive

```
int max (struct node *P)
{ int x = 0;
  if (P == 0)
    return MIN_INT;
  else
  { x = Max(P->next);
    if (x > P->data)
      return x;
    else
      return P->data;
  }
}
```

(or)

```
int max (struct node *P)
{ int x = 0;
  if (P == 0)
    return MIN_INT;
  x = max(P->next);
  return x > P->data ? x : P->data;
}
```

- Searching in a linked list

① Linear search - ✓
② Binary " - ✗ (Not work in linked list)

1) Linear search -

Iterative

```
Node * Search (Node * P, int key)
    while (P != NULL)
    {
        if (key == P->data)
            return (P)
        P = P -> next;
    }
    return NULL;
}
```

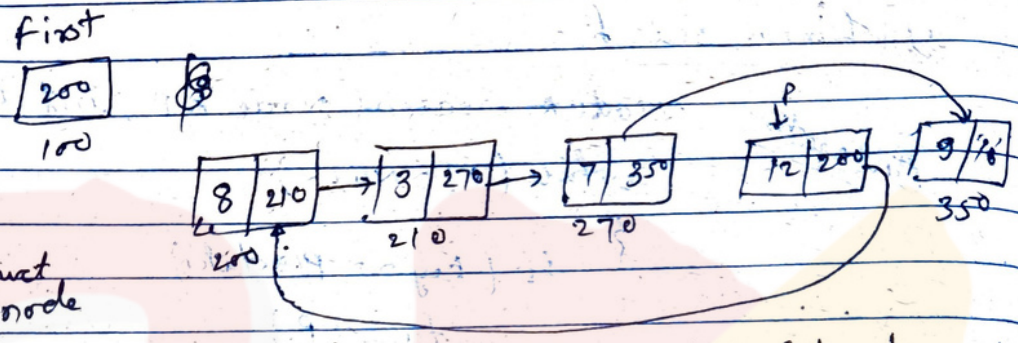
Recursive

```
Node * Search (Node * P, int key)
{   if (P == NULL)
        return NULL;
    if (key == P -> data)
        return (P)
    return Search (P -> next, key);
}
```

* Improve searching in linked list

1) Transposition - ✗

2) Move to head - ✓

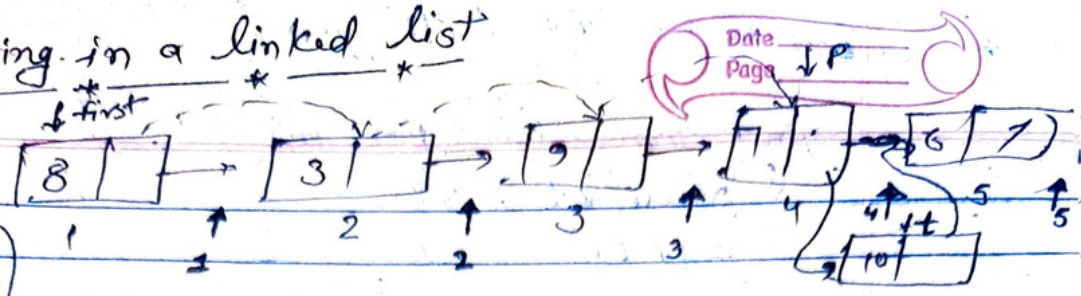


Node = struct node

let key = 12

```
Node * search(Node *p ; int key)
    Node *q = NULL;
    while(p != NULL)
        if (key == p->data)
        {  q->next = p->next;
           p->next = first;
           first = p
        }
        q = p;
        p = p->next;
    }
}
```

* Inserting in a linked list



These are the position where a node can be insert.

Two case - 1) Insert befor first
2) Inserting after given position

(1) Insert before first -

```
Node *t = new node ; // Creating a new node
t->data = x;  // Inserting the data
t->next = first // make point to first
first = t;  // Change first to t
```

time - O(1)

(2) Inserting after given position -

```
let : pos = 4
Node *t = new Node;
t->data = x;
p = first;  // 'p' pointer start from first
for(i=0; i<pos-1 && p; i++)  // Moving the 'p' pointer
{
    p = p->next;
}
t->next = p->next
p->next = t;
```

min - O(1)
max - O(n)

* Combine program for both cases -

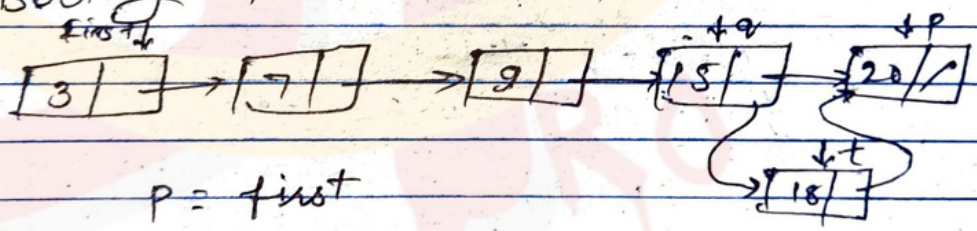
```
Void Insert (int pos, int x)
{ Node *t, *p;
  if(Pos == 0)
  { t = new Node;
    t->data = x;
    t->next = first;
      first = t;
  }
else if (Pos > 0)
  { p = first;
    for (i=0; i< pos-1 && p; i++)
      p = p->next;
    if(p)
    {
      t = new Node;
      t->data = x
      t->next = p->next;
        p->next = t;
    }
  }
}
```

* Creating a linked list by inserting a last -

```
void InsertLast(int x)
{
    Node *t = new Node;
    t->data = x;
    t->next = NULL;
    if(first == NULL)
    {
        first = last = t;
    }
    else
    {
        last->next = t;
        last = t;
        t = last;
    }
}
```

- It always add nodes at last.
- It also works when linked list is empty.

* Inserting in a sorted linked list -




```
p = first
q = NULL
while(p && p->data < x)
{
    q = p;
    p = p->next;
}
t = new node;
t->data = x;
t->next = q->next;
q->next = t;
```

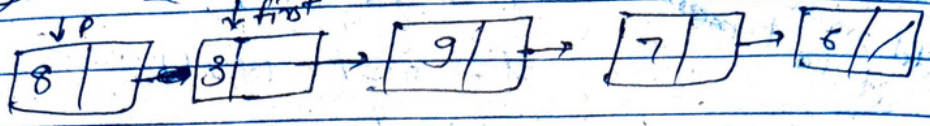
time
min - O(1)
max - O(n)

06/05/22

Delete a node in a linked list -

1) Delet first node
2) Delet a node at given position

Case -1
Delete first node -




```
Node *P = first;
first = first->next;
x = p->data;  (Deleted node's data is kept in x for further use)
delete P;
```

time - O(1)

Case-2 - Deleting at given postion - (two pointers are required for this)

let pos = 4

```
Node * P = first;
Node *q = NULL;
for(i=0; i< pos-1 ; i++)
{
    q = P;
    P = P->next;
}
q->next = P->next;
x = P->data;
delete P;
```

min - O(1)
max - O(n)

* Combine program -

```
int Delete (int pos)
    Node * P, *q;
    int x = -1, i;
    if (pos == 1)
    {
        x = first -> data;
        P = first;
        first = first -> next;
        delete P;
    }
else
    {
        P = first;
        q = NULL;
        for (i=0; i<pos-1 && P; i++)
        {
            q = P;
            P = P -> next;
        }
        if (P)
        {
            q -> next = P -> next
            x = P -> data
            delete P;
        }
    }
    return x;
```

* Check if a linked list is sorted

Date
Page

```
int x = -32768;
Node *p = first;
while (p != NULL)
{
    if (p->data < x)
        return false;
    x = p->data;
    p = p->next
}
return true;
```

min - O(1)
max - O(n)

* Removing a ~~link~~ duplicates from sorted ~~linked list~~

```
Node *p = first;
Node *q = first->next;
while (q != NULL)
{
    if (p->data != q->data)
    {   p = q;
        q = q->next;
    }
    else
    {p->next = q->next;
     delete q;
     q = p->next;
    }
}
```

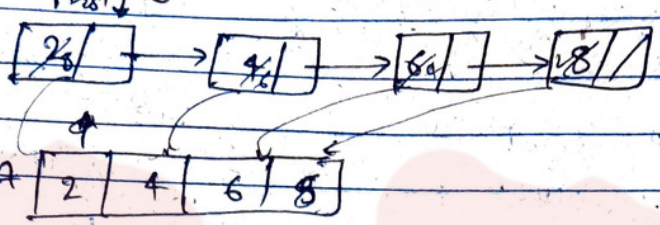
always
time - O(n)

* Reversing a linked list -

two methods - ① Reversing elements
② Reversing links

Date
Page

1) By Reversing the elements -



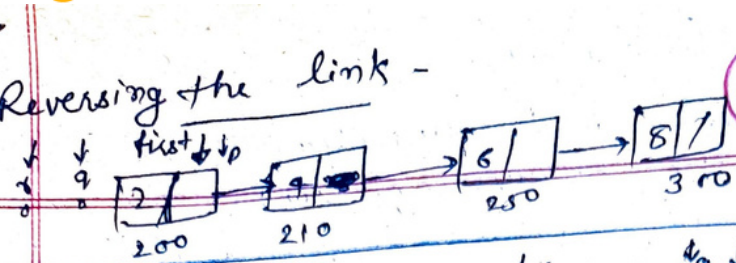
This array will be same size of linked list. With the help of this, we can copy the elements from linked list in array and after copying in array again copy the array elements to linked list but in reverse order.

```
P = first;
i = 0;
while (P != NULL)
{ A[i] = P->data;
  P = P->next;
  i++;
}
P = first, i--;
while (P != NULL)
  P->data = A[i--];
  P = P->next;
```

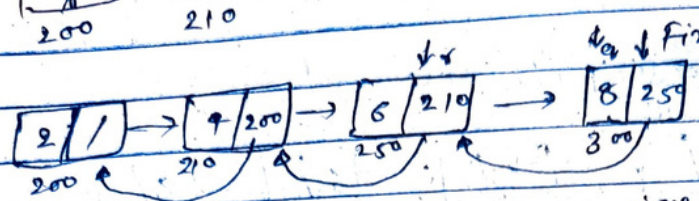
time - O(n)

07/09/22

ii) By Reversing the link -



Reverse the link

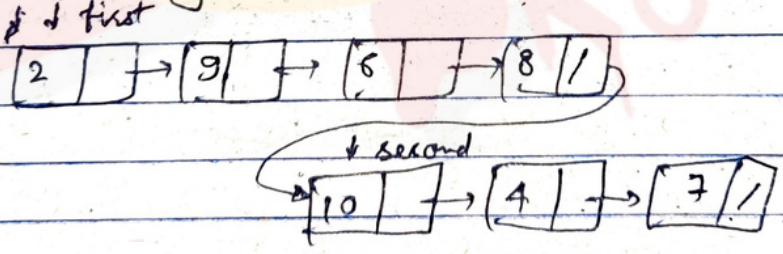



```
p = first
q = NULL
r = NULL
while (p != NULL)
  { r = q;
    q = p;
    p = p->next
    q->next = r;
  }
first = q;
```

Recursion (: it uses only two pointers)

```
void Reverse (Node *q, Node *p);
{
  if (p != NULL)
  {
    Reverse (p, p->next);
    p->next = q;
  }
  else
  {
    first = q;
  }
}
```

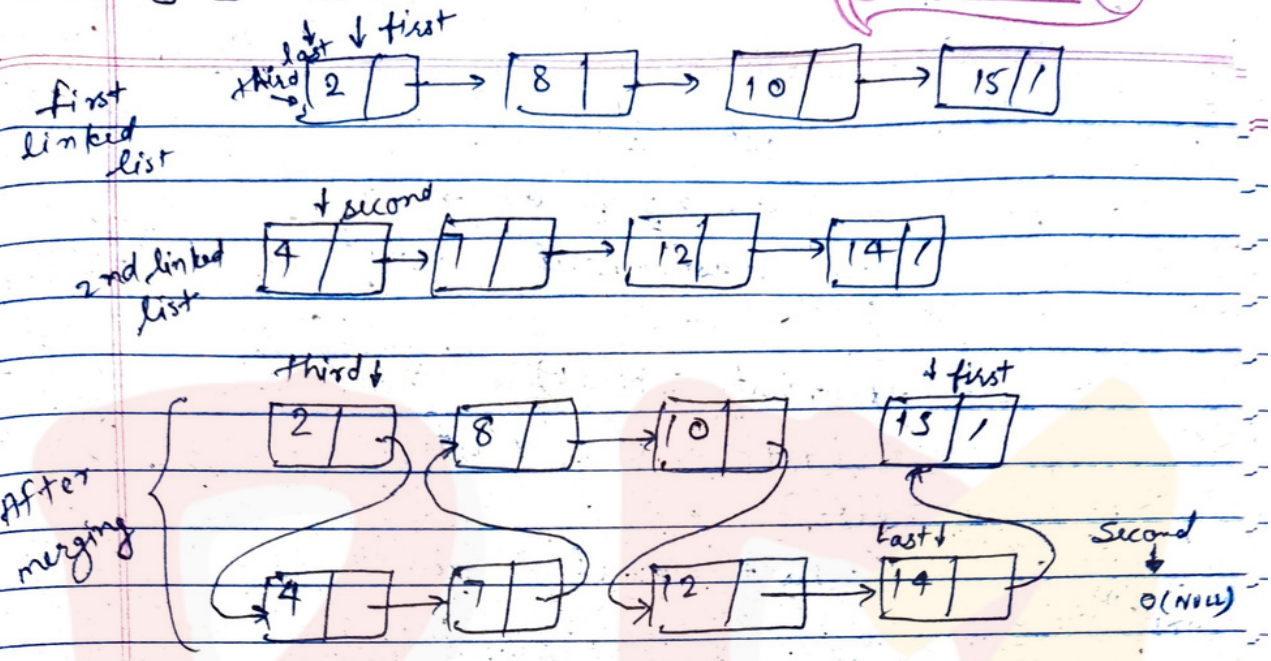
* Concatenating two linked list -




```
p = first;
while (p->next != NULL)
  { p = p->next;
  }
p->next = second;
```

time - $\sigma(n)$

* Merging of two linked list - (Always done on sorted linked list)




```
if (first->data < second->data)
  { third = last = first;
    first = first->next;
    last->next = NULL;
  }
else
  { third = last = second;
    second = second->next;
    last next = NULL;
  }
while (first != NULL && second != NULL)
  { if (first->data < second->data)
      { last->next = first;
        last = first;
        first = first->next
        last->next = NULL;
      }
```

that means it is started from first linked list

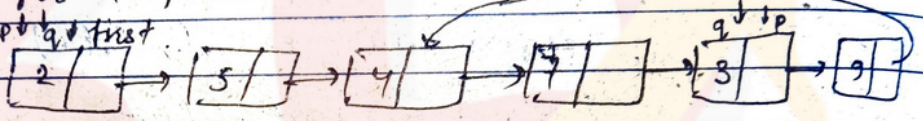
Date
Page

```
else
{
    last->next = second;
    last = second;
    second = second->next;
    last->next = NULL;
}
if (first != NULL)
    last->next = first;
else
    last->next = second;
```

time - $\Theta(m+n)$

08/05/22

* Check for loop in linked list -



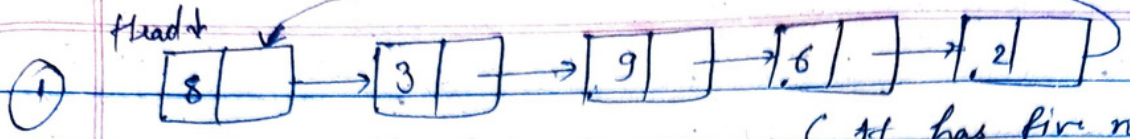

```
int isloop(Node *first)
{
    Node *p, *q;
    p = q = f;
    do
    {
        p = p->next;
        q = q->next;
        q = q != NULL ? q->next : NULL;
    } while (p && q && p != q);
    return p == q ? true : false;
}
```

time - $O(n)$

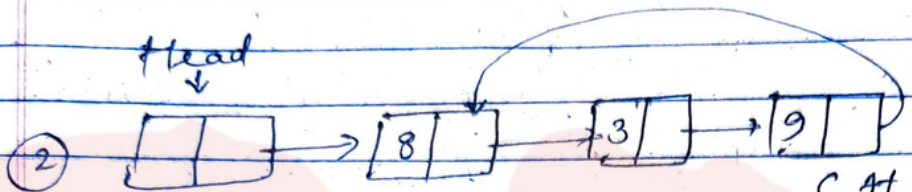
'p' pointer move one node at a time whereas 'q' pointer moves two nodes at time.

→ If p and q meet at a position then loop exist otherwise it is linear.

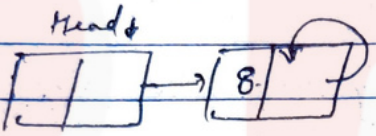
* Circular linked list - There are two representations of it -



(1) (It has five nodes)



(2) (It has three nodes)



(It has one node)

(Empty circular linked list)

* Display circular linked list -

Iterative

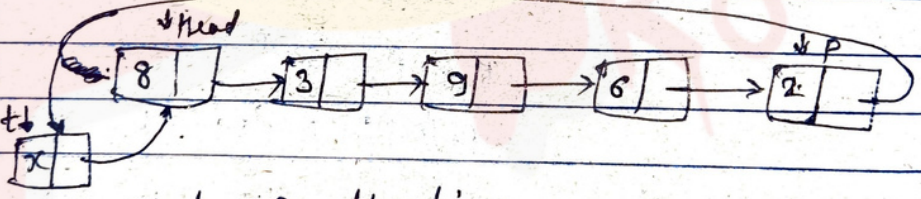
```
void Display (node *p)
{
    do
    {
        printf ("%d", p->data);
        p = p->next;
    } while (p != Head);
}
Main ()
{
    Display (Head);
}
```

Recursion

```
void display (Node *P)
{
  static int flag = 0;
  if (P != Head || flag == 0)
  {
    flag = 1;
    printf ("%d", P->data);
    Display (P->next);
  }
  flag = 0;
}
main()
Display (Head)
```

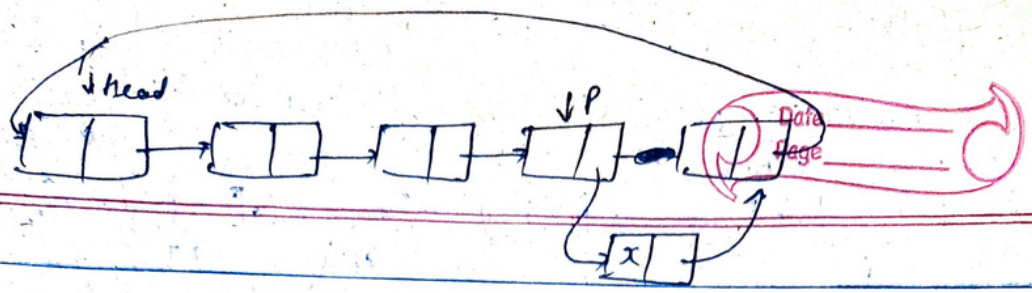
* Inserting in a circular linked list -
  i) Insert before first
  ii) Insert after given position

i)




```
Node *P = Head;
Node *t = New node;
t->data = x;
t->next = Head;
while (P->next != Head)
{
  P = P->next;
}
P->next = t;
Head = t; (optional).
```

(11)




```
Node *t;
Node *P = Head;
for(i=0; i<pos-1; i++)
    P = P->next;
t = new node;
t->data = x;
t->next = P->next;
P->next = t;
```

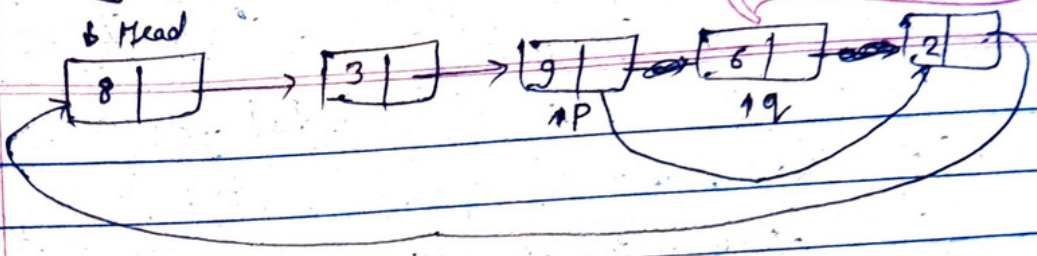
* Combine program -

```
void insert(int pos, int x)
{ Node *t, *P;
  int i;
  if(pos==0)
  { t = new Node;
    t->data = x;
    if(Head==NULL)
    { Head = t;
      Head->next = Head;
    }
    else
    { P = Head;
      while(P->next != Head)
      { P = P->next;
      }
      P->next = t;
      t->next = Head;
      Head = t;
    }
  }
  else
  { P = Head;
    for(i=0; i<pos-1; i++)
      P = P->next;
    t = new node;
    t->data = x;
    t->next = P->next;
    P->next = t;
  }
```

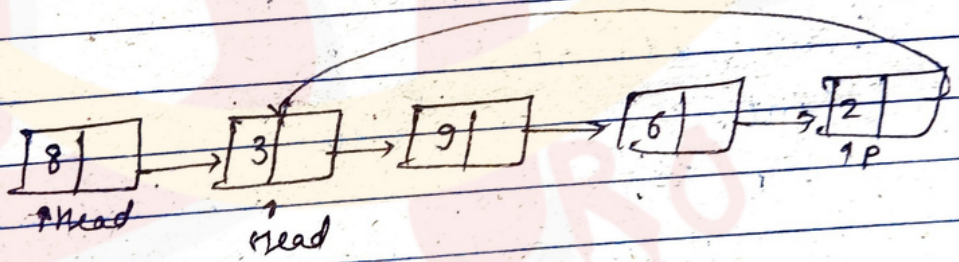
09/05/22

* Deleting from circular linked list

① Deleting node at given pos
② Deleting the head



①
```
P = Head;
for(i=0; i<Pos-2; i++)
{
    P = P->next;
}
q = P->next;
P->next = q->next;
x = q->data;
delete q;
```

② 


```
P = Head;
while(P->next != Head)
    P = P->next;
P->next = Head->next;
x = Head->data;
delete head;
Head = P->next;
```

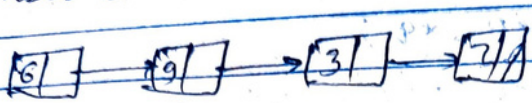
min = O(1)
max = O(n)

Combine program -

```
int Delete (int pos)
Node *P, *q;
if (Pos == 1)
{
    P = Head;
    while (P→next != Head)
        P = P→next;
    x = Head→data;
    if (P == Head)
    {
        delete Head;
        Head = NULL;
    }
    else
    {
        P→next = Head→next;
        delete Head;
        Head = P→next;
    }
}
else
{
    P = Head;
    for (i = 0; i < Pos - 2; i++)
    {
        P = P→next;
    }
    q = P→next;
    x = q→data;
    P→next = q→next;
    delete q;
}
return x
}
```

* Singly ~~Doubly~~ linked list -

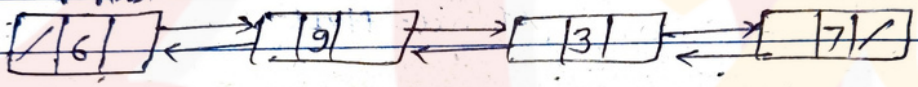
In singly linked list every node have a pointer to next node. ~~in~~ a



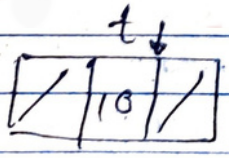
* Doubly linked list -

In this linked list every nodes have a pointer to next node as well as previously.

(~~it~~ means from node, I can go forward as ~~for~~ as I can come back so I can access this list ~~by~~ of elements bidirectionly.)




```
struct Node *t;
t = new node;
t -> prev = NULL
t -> data = 10
t -> next = NULL;
```

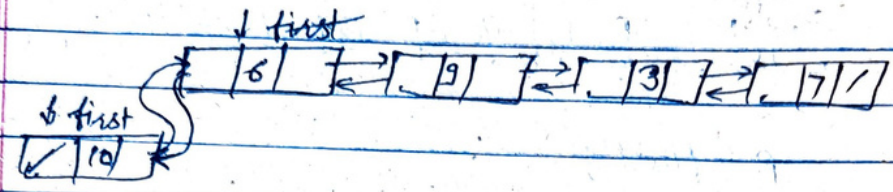


Node


```
struct Node
{
  struct node *Prev;
  int data;
  struct node * next;
};
```

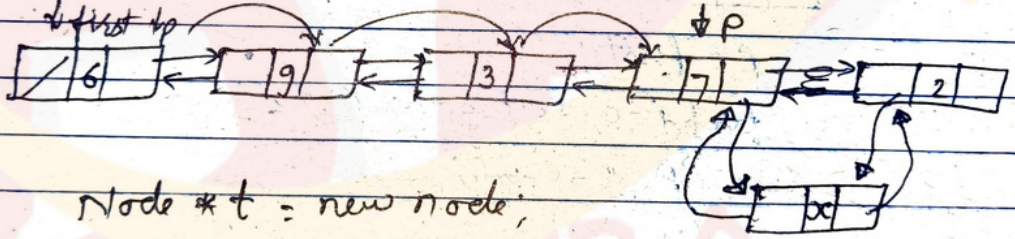
* Insert in a doubly linked list -

(i) Before first node




```
Node *t = new Node;
t->data = x;
t->prev = NULL;
t->next = first;
first->prev = t;
first = t
```

(ii) Inserting at given position -

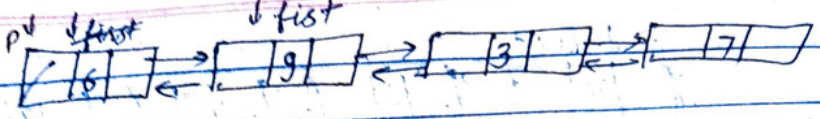



```
Node *t = new node;
t->data = x;
for(i=0; i<pos-1; i++)
{
  p = p->next;
}
t->next = p->next;
t->prev = p;
if(p->next)
{
  p->next->prev = t;
}
p->next = t;
```

min = o(1)
max = o(n)

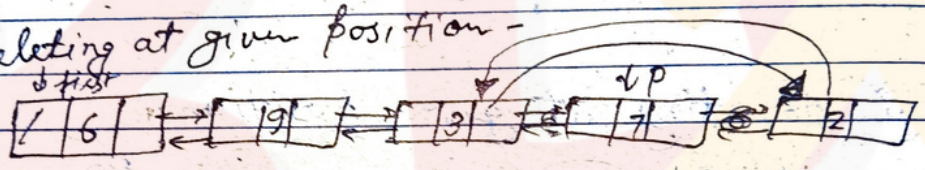
* Deleting from doubly linked list -

① Delete first node -




```
p = first;
first = first → next;
x = p → data;
delete p;
if (first)
{ first → prev = NULL
;
```

② Deleting at given position -




```
p = first;
for(i=0; i<pos-1; i++)
  {p = p → next;
  }.
p → prev → next = p → next;
  if (p → next)
    { p → next → prev = p → prev;
    }
    x = p → data;
    delete p;
```

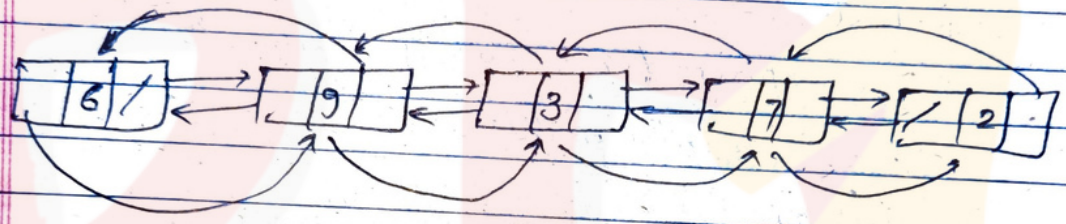
min - O(1)

max - O(n)

* Display doubly linked list


```
P = first;
while(P != NULL)
{
    printf("%d", P->data);
    P = P->next;
}
```

* Reverse




```
P = first;
while(P)
{
    temp = P->next;
    P->next = P->prev;
    P->prev = Temp;
    P = P->prev;
    if(P->next == NULL)
        first = P;
}
```