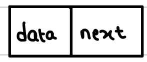
LINKED LIST

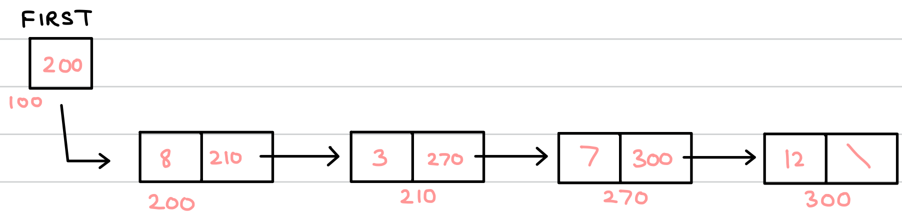
1. **Problem with arrays** : Fixed Size

Q What is a linked list?

Ans Linked list is a collection of nodes where each node contain data and pointer to next node.

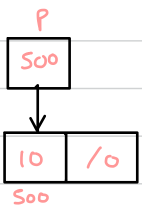


NODE




```
struct Node                SELF REFERENTIAL STRUCTURE
{
      int data;            ——— 2
      struct Node * next;  ——— 2   (same as datatype)
};                             4 bytes
```

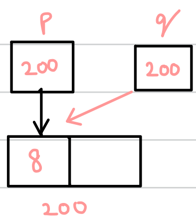
```
struct Node *p;
p= (struct Node *)malloc (sizeof(struct Node));
p = new Node;      C++

p → data =10;
p → next = 0;
```

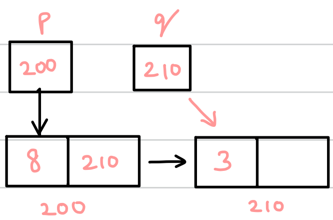


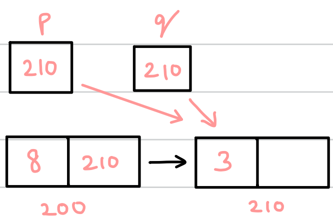
---

struct Node *p, *

1. ~~q~~ q=p 2. q = p → next 3. p = p → next

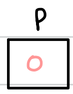






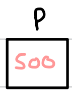
```
struct Node *p = NULL;
```

```
if ( p == NULL)
if ( p == 0)
if ( !p)
if (p->next == NULL)
```



To check if pointer is not pointing anywhere

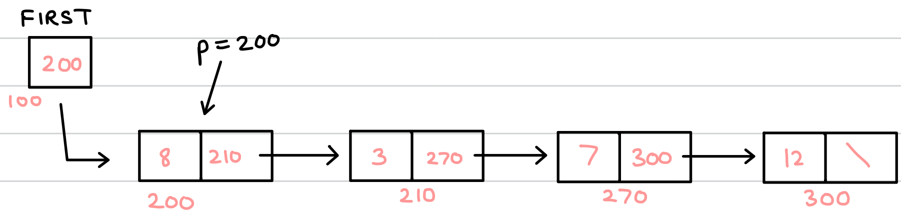
```
if (p!= NULL)
if (p ! =0)
if (p)
if (p -> next != NULL)
```



To check if pointer is not NULL.

TRAVERSING THROUGH LINKED LIST

```
struct Node * p = first;
while ( p ! = 0)
{
        p = p ->next;
}
```



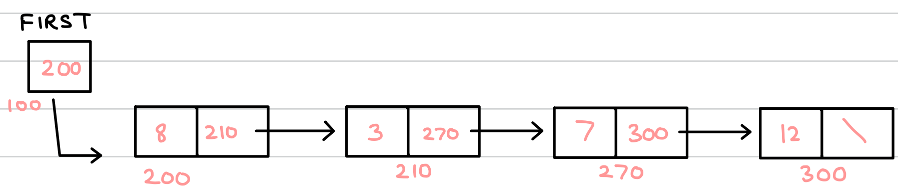

```
#include<stdio.h>
#include <stlib.h>

struct Node
{
      int data;
      struct Node *next;
} * first = NULL;
```

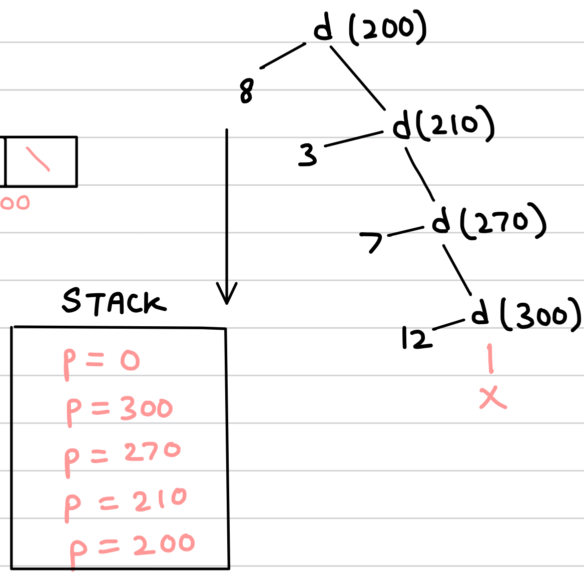
Check struct Node * first= NULL in main

```
void create (int A[], int n)
{
      int i;
      struct Node *t , *last ;
      first = ( struct Node *)malloc (sizeof ( struct Node));
      first ->data= A[0]
      first -> next = NULL;
      last = first;
```

```
    for( i=1; i<n; i++)
    {
        t = (struct node *)malloc( sizeof ( struct Node));
        t -> data = A[i];
        t -> next = NULL;
        last -> next = t;
        last = t;
    }
}

void Display ( struct Node *p)
{
    while (p != NULL)
    {
        printf ("%d ", p -> data);
        p = p -> next;
    }
}

void main()
{
    struct Node *temp;
    int A[] = { 3, 5, 7, 10, 25, 8, 32, 2};
    create (A, 8);
    Display (first);
}
```

RECURSIVE DISPLAY OF LINKED LIST

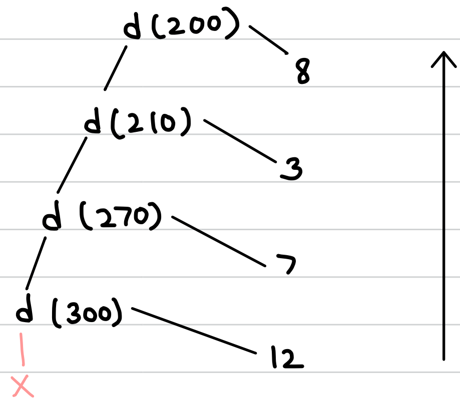





```
void Display (struct Node *p)
{
    if (p != NULL)
    {
        printf(" %d", p -> data);
        Display ( p -> next);
    }
}            Display ( first);
```

```c
void Display (struct Node *p)
{
    if (p != NULL)
    {
        Display (p->next);
        printf(" %d", p->data);
    }
}
```

O(n)



COUNTING NODES IN LINKED LIST

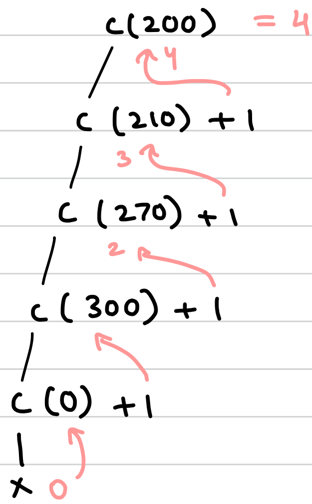
```c
int count (struct Node *p)
{
    int c=0;
    while (p!=0)
    {
        c++;
        p=p->next;
    }
    return(c);
}
```

O(n)

RECURSIVE FUNCTION FOR COUNTING NUMBER OF NODES

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

O(n)



SUM OF ALL ELEMENTS IN A LINKED LIST

```c
int add (Struct Node *)
{
    int sum=0;
    while (p)
    {
        sum = sum + p->data;
        p = p->next;
    }
    return sum;
}
```

USING RECURSION

```c
int Add(struct Node *p)
{
    if (p==0)
        return 0;
    else
        return Add(p->next)+ p->data;
}
```

O(n)

MAXIMUM ELEMENT IN A LINKED LIST

```
int max(struct Node *p)
{
    int m = -32768;
    while(p)
    {
        if(p->data>m)
            m = p->data;
        p = p->next;
    }
    return(m);
}
```

RECURSION

```
int max(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;
    }
}
```

SEARCHING IN A LINKED LIST

Binary search is not suitable for linked list as we cannot go directly in the middle of the list

LINEAR SEARCH

```
Node* Search(Struct 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);
}
```

IMPROVING SEARCHING

```
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

Before first Node

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

Pos == 4

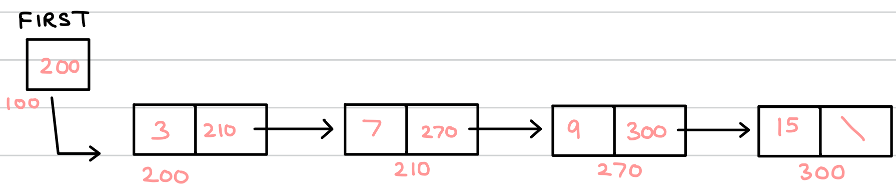
```
Node *t = new Node;
t->data = x;
p = first;
for (i=0 ; i < pos-1 ; i++)
        p = p->next;
t->next = p->next;
p -> next = t;
```

```
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;
        }
    }
}
```

INSERTING AT 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;
    }
}
```

INSERTING IN A SORTED LIST




```
p = first;
q = NULL        ( Tailing Pointer)

while ( p && p → data < x)
{
    q = p;
    p = p → next;
}
t = new Node;
t → data = x;
t → next = q → next;
q → next = t ;
```

DELETION FROM LINKED LIST

(1) Deletion of first Node

```
Node *p = first;          O(1)
first = first → next;
x = p → data;
free (p);
```

(2) Deletion from a given position

```
Node *p = first;          min O(1)
Node *q = NULL;           max O(n)

for (i = 0 ; i < pos -1 ; i++)
{
        q = p;
        p = p → next;
}

q → next = p → next;
x = p → data;
free (p);
```

CHECK IF LIST IS SORTED

```
int x = -32768;           O(n)
Node *p = first;
while ( p != NULL)
{
        if (p → data < x)
                return -1;
        x = p → data;
        p = p → next;
}
return 0;
```

REMOVE DUPLICATES FROM LIST

```
Node *p = first;                O(n)
Node *q = first->next;

while ( q != NULL)
{
    if ( p->data != q->data)
    {
        p = q;
        q = q->next;
    }
    else
    {
        p->next = q->next;
        free (q);
        q = p->next;
    }
}
```

REVERSING A LINKED LIST

(1) Reversing Elements
(2) Reversing Links

Reversing links is preferred over reversing elements because maybe there are lots of values in a single node.

(1) Reversing Elements

First create an array A equal to size of length of linked list.

```
p = first;                 O(n)
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;
}
```

(2) *Reversing Links   TRACE IT!

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

sliding pointers

REVERSING LINKED LIST USING RECURSION

```
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;
second = NULL;
```

O(n)

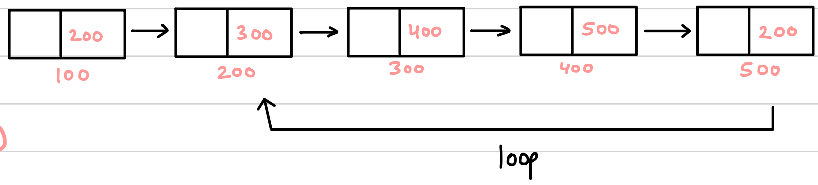
MERGING TWO LINKED LIST

We have two sorted linked list and we want to combine it into a single list.

$\Theta(m+n)$

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

CHECK FOR LOOP IN LINKED LIST

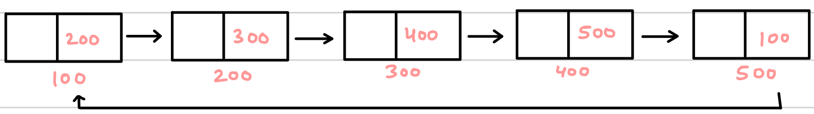


O(n)

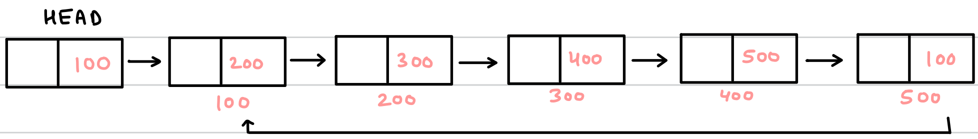
```
int isloop (Struct Node *f)
{
    Struct Node *p, *q;
    p=q=f;

    do
    {
            p = p→next;
            q = q →next;
            q = q ? q →next : q;
    } while ( p && q && p!=q);

    if (p==q)
            return 1;
    else
            return 0;
}
```

CIRCULAR LINKED LIST

1



2



DISPLAY CIRCULAR LINKED LIST

```
void Display (Node *p)
{
    do
    {
            printf(" %d", p → data);
            p=p→next;
    } while(p != Head);
}
```

Display (Head);

DISPLAY CIRCULAR LINKED LIST USING RECURSION

```
void Display (Node *p)
{
        static int flag = 0 ;
        if ( p! = Hea  || flag = 0)
        {
                flag = 1;
                printf(" %d ", p → data);
                Display ( p → next);
        }
}
```

CREATION OF CIRCULAR LINKED LIST

```
void create (int A[], int n)
{
        int i;
        struct Node *t, * last ;
        Head = (struct Node *) malloc (sizeof (struct Node));
        Head → data = A[0];
        Head → next = Head;
        last = Head ;

        for (i= 1; i <n; i++)
        {
                t= (struct Node *)malloc (sizeof (struct Node));
                t → data = A[i];
                t → next = last → next;
                last → next = t;
                last = t;
        }
}

int main()
{
        int A[] = { 2,3,4,5,6};
        create (A , 5);
}
```

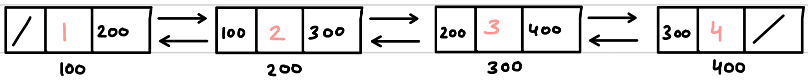
INSERTING IN A CIRCULAR LINKED LIST

```
void Insert( struct Node *p, int index , int x)
{
    struct Node *t;
    int i;

    if (index <0 || index > length())
        return;

    if (index ==0)
    {
        t= (struct Node *)malloc ( sizeof (struct Node));
        t -> data = x;

        if (Head = = NULL)
        {
            Head = t;
            Head -> next = Head;
        }
        else
        {
            while( p -> next != Head)
                    p = p -> next;
            p -> next = t;
            Head = t;
            t -> next= Head;
        }
    }

    else
    {
        for (i =0; i< index-1 ; i++)
            p= p -> next;
        t= (struct Node *)malloc ( sizeof (struct Node));
        t -> data = x;
        t -> next = p ->next;
        p -> next =t;
    }
}
```

DELETION IN CIRCULAR LINKED LIST

```
int Delete (struct Node *p, int index)
{
    struct Node *q;
    int i, x;

    if (index<0 || index > length(Head))
        return -1;

    if (index == 1)
    {
        while (p->next! = Head)
            p=p->next;
        x= Head->data;

        if (Head == p)      // If there is only one node
        {
            free (Head);
            Head = NULL;
        }

        else
        {
            p->next = Head->next;
            free (Head);
            Head = p->next;
        }
    }

    else
    {
        for (i=0 ; i< index-2; i++)
            p = p->next;
        q= p->next;
        p->next = q->next;
        x= q->data;
        free (q);
    }
}
```

INSERT IN A DOUBLY LINKED LIST DOUBLY LINKED LIST




```
struct Node
{
        struct Node * prev;
        int data;
        struct Node * next;
} * first = NULL;


void  create (int A[], int n)
{
        struct Node *t, * last;
        int i;

        first = (struct Node *) malloc ( sizeof ( struct Node));
        first -> data = A[0];
        first -> prev = first -> next = NULL;
        last = first;

        for (i=1 ; i<n ; i++)
        {
                t = (struct Node *) malloc ( sizeof ( struct Node));
                t -> data = A[i];
                t -> next = last -> next
                t -> prev= last ;
                last -> next = t;
                last = t
        }
}
```

INSERT IN A DOUBLY LINKED LIST

(1) Insert at first node

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

(2) Insert at any 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;
```

// To check if a node is available after p or not

DELETE FROM A DOUBLY LINKED LIST

(1) Deleting first node

```
p = first;
first = first -> next;
x = p -> data;
delete p;
if (first)        // If first is not NULL
    first -> prev = NULL;
```

(2) Deleting Node from given index

```
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;
```

if p -> next is not NULL

REVERSING A DOUBLY LINKED LIST

```
p= first;

while (p)
{
        temp = p→next;
        p→next = p→prev;
        p→prev = temp;
        p = p→prev

        if ( p! = NULL && p→next = NULL)
                first = p;
}
```

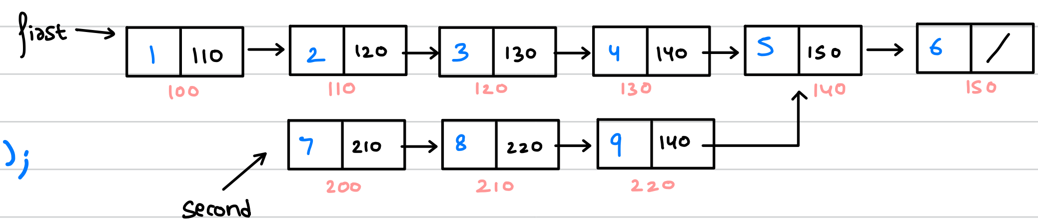
FINDING MIDDLE NODE IN A LINKED LIST

```
p=q= first;

while(q)
{
        q = q→next;
        if (q)
            q = q→next;
        if (p)
            p = p→next;
}
```

Two pointers p and q will move at the same time.

q will move 2 nodes and p will move 1 node.

FINDING INTERSECTION POINT OF TWO LINKED LIST




```
p= first
while ( p!=NULL)
        push (&stk1, p);
p= second;
while ( p! NULL)
        push (&stk2, p);
while ( stacktop (stk1) == stacktop (stk2))
{
        p = pop(&stk1);
        pop (&stk2);
}
```