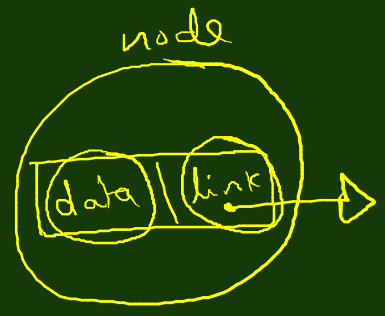
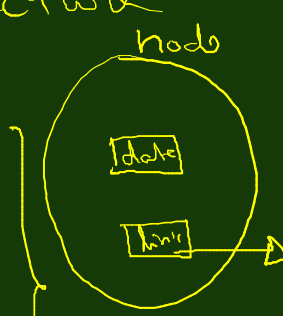


Self Referential Structure

```
struct node  
{  
    char data;  
    struct node * link;  
};
```



prototype
{ no memory is allocated }

struct node * NewNode()

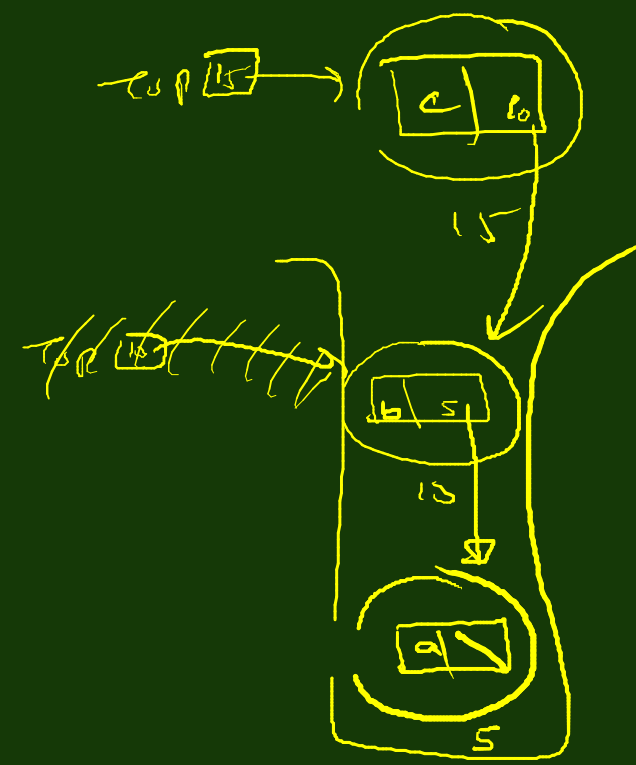
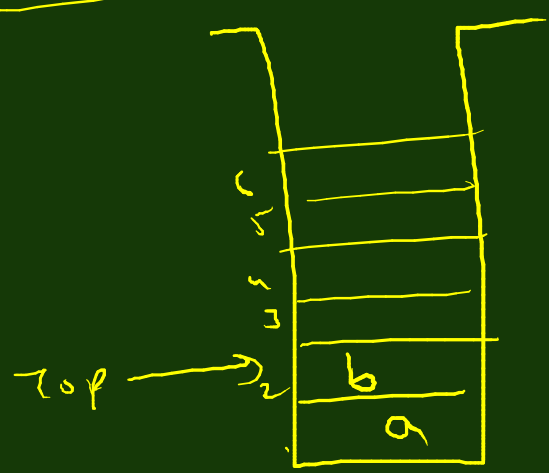
```
① { struct node * n;  
② n = (struct node *) malloc ( sizeof (struct node) )  
③ return n;  
}
```

struct Node * Top = NULL;

void main()

```
{  
    for ( int i = 0; i < 5; i++)  
        * push( i )  
}
```

Array Stack



Top = 0

push(a)
push(b)

Top = n

top → null

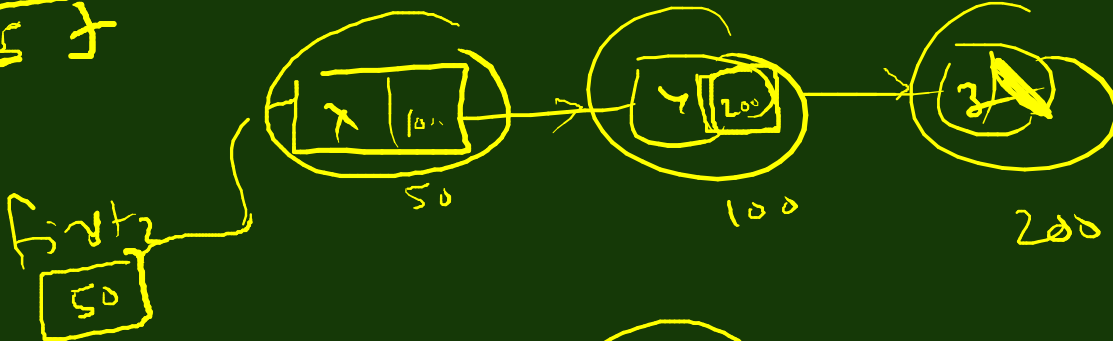
Access



$$\{ \begin{aligned} &(\neq n) \cdot \text{data} = 'a'; \\ &n \rightarrow \text{data} = 'a'; \\ &n \rightarrow \text{link} = \text{NULL} \end{aligned}$$
 OR

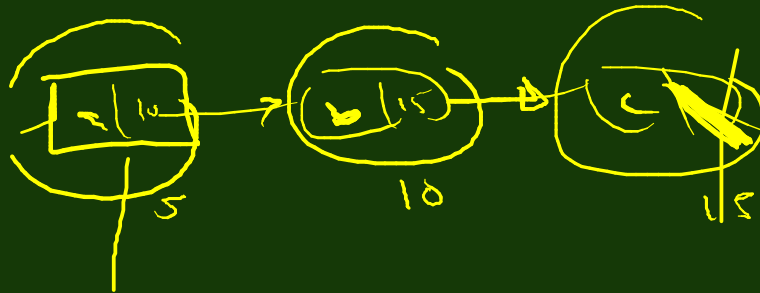


first1
[5]

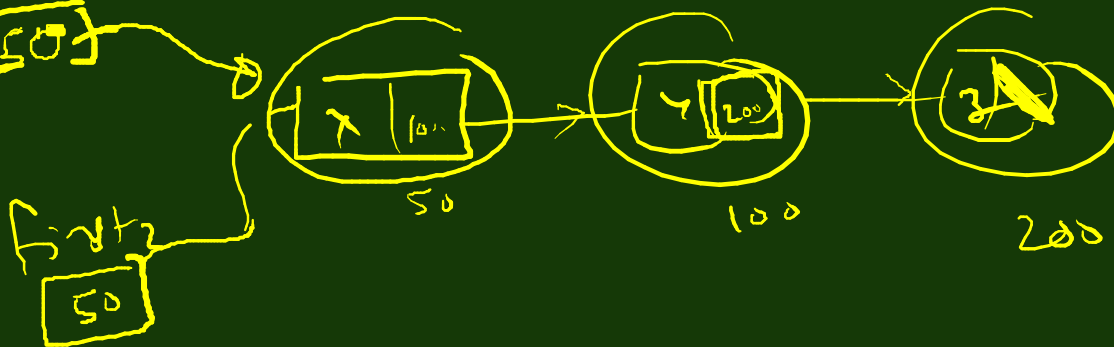


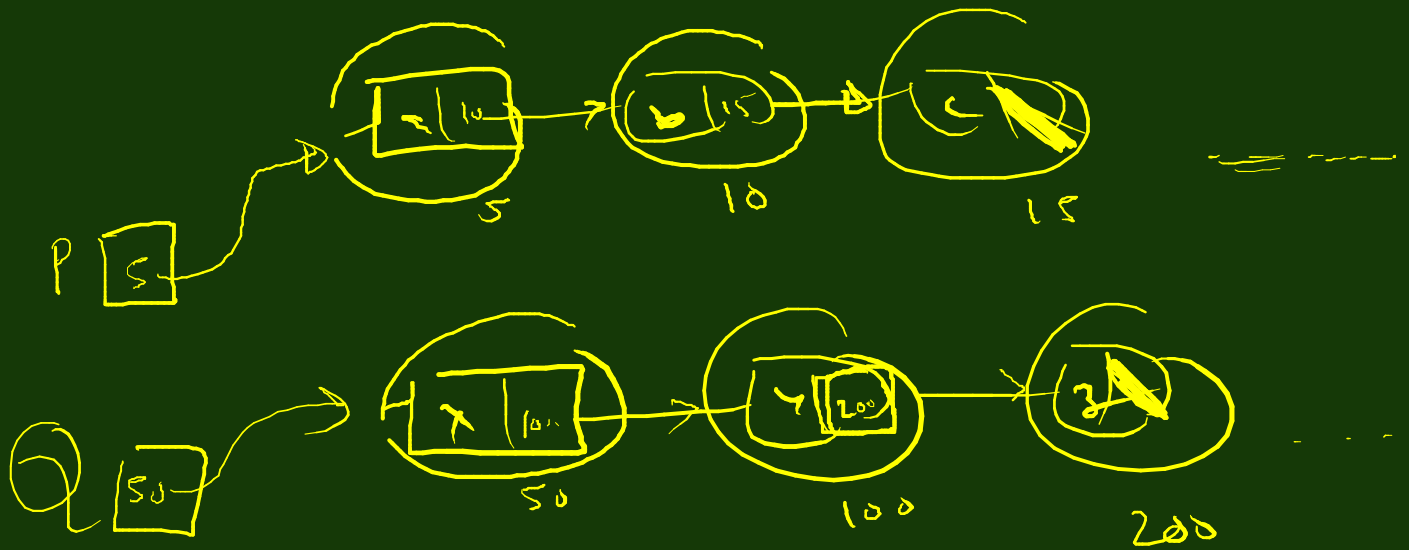
first2 get updated.

first2 goes to first 2

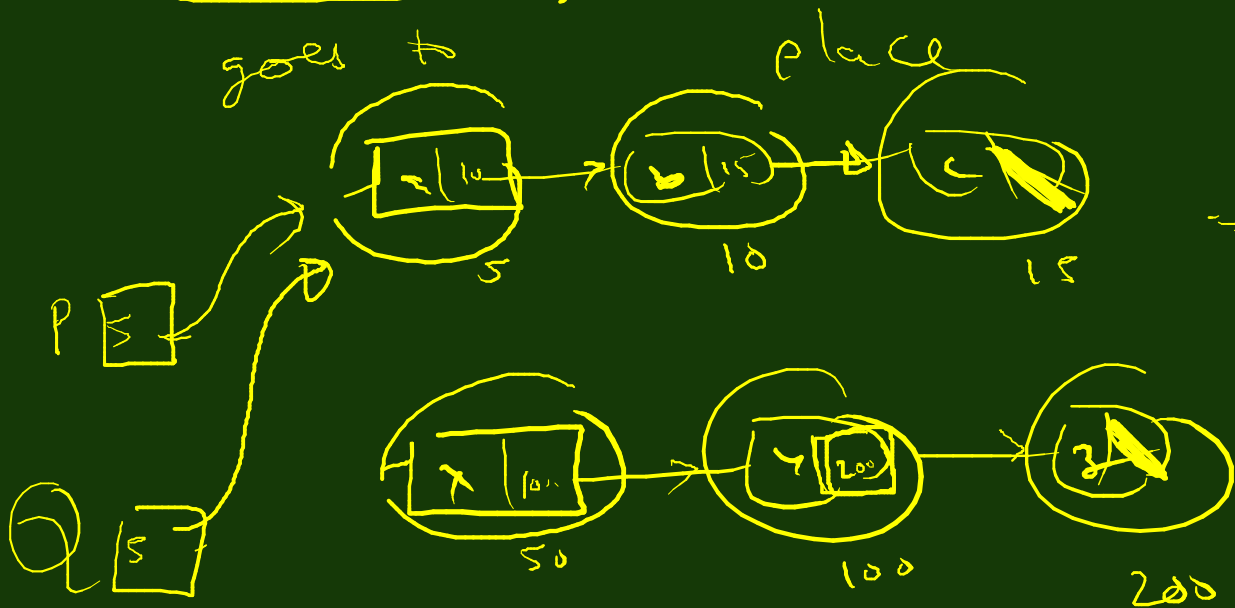


first1
[50]





Remember
Q goes to P's place



direct Selector

→ Indirect Selector

read

n → data

→ data part of my 'n'
→ n's data

push(char x)

{

1) ~~if not node~~ * n = new Node();

2) n → data = x;

3) n → link = Top;

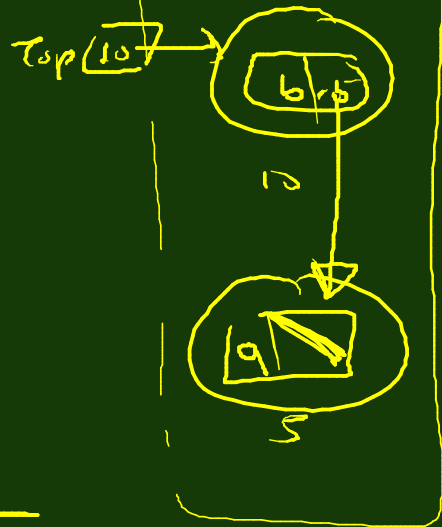
Top = n

4)

}

push(w)

push(u)



char

pop()

{ if (Top == NULL)
UNDERFLOW

// remember

→ char x;
struct Node *top;

Top → NULL

① temp = Top;

② x = Top → data;

3 Top = Top → link; // Forward

4 free(temp);

5 return x;



void push (char x)

{

1) struct node * n = newNode();

2) n → data = x;

3) n → link = Top

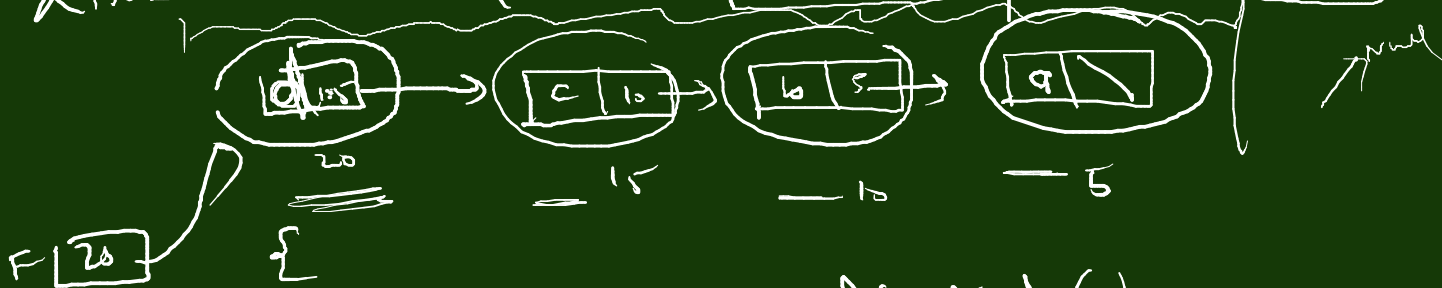
Top = n

4)

}

OB

Linked stack push() = Insert Front of SLL



① struct node * n = NewNode();

② n → data = x;

③ n → link = First;

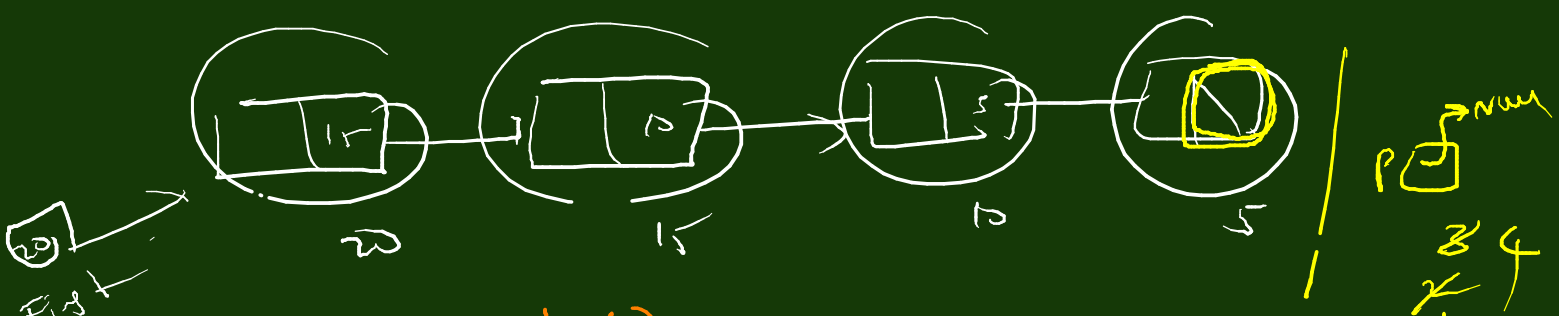
④ First = n;

Insert Front of SLL

= push() of Linked Stack.

Count

the number of nodes in SL



int CountNodes()

{ ① int counter = 0; ✓

② struct Node *p = First;

while (p) (or) while (p != null)

③ { 3a) counter = counter + 1
3b) p = p -> link; // Forward

④ return counter;



counter 0

9
Recursion

DO (on 0)

int { ① ~~int counter = 0;~~ ✓

② struct Node *p = First;

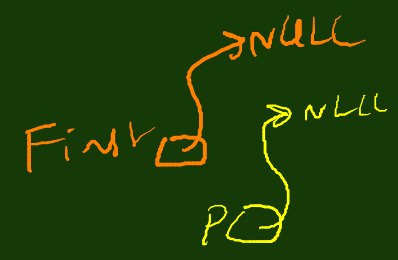
~~while (p) (or) while (p != null)~~

③ { 3a) ~~counter = counter + 1~~
3b) ~~p = p -> link;~~ // Forward

④ return counter;

Rough
p

DO (p) ④
DO (p -> link)
count 4



```

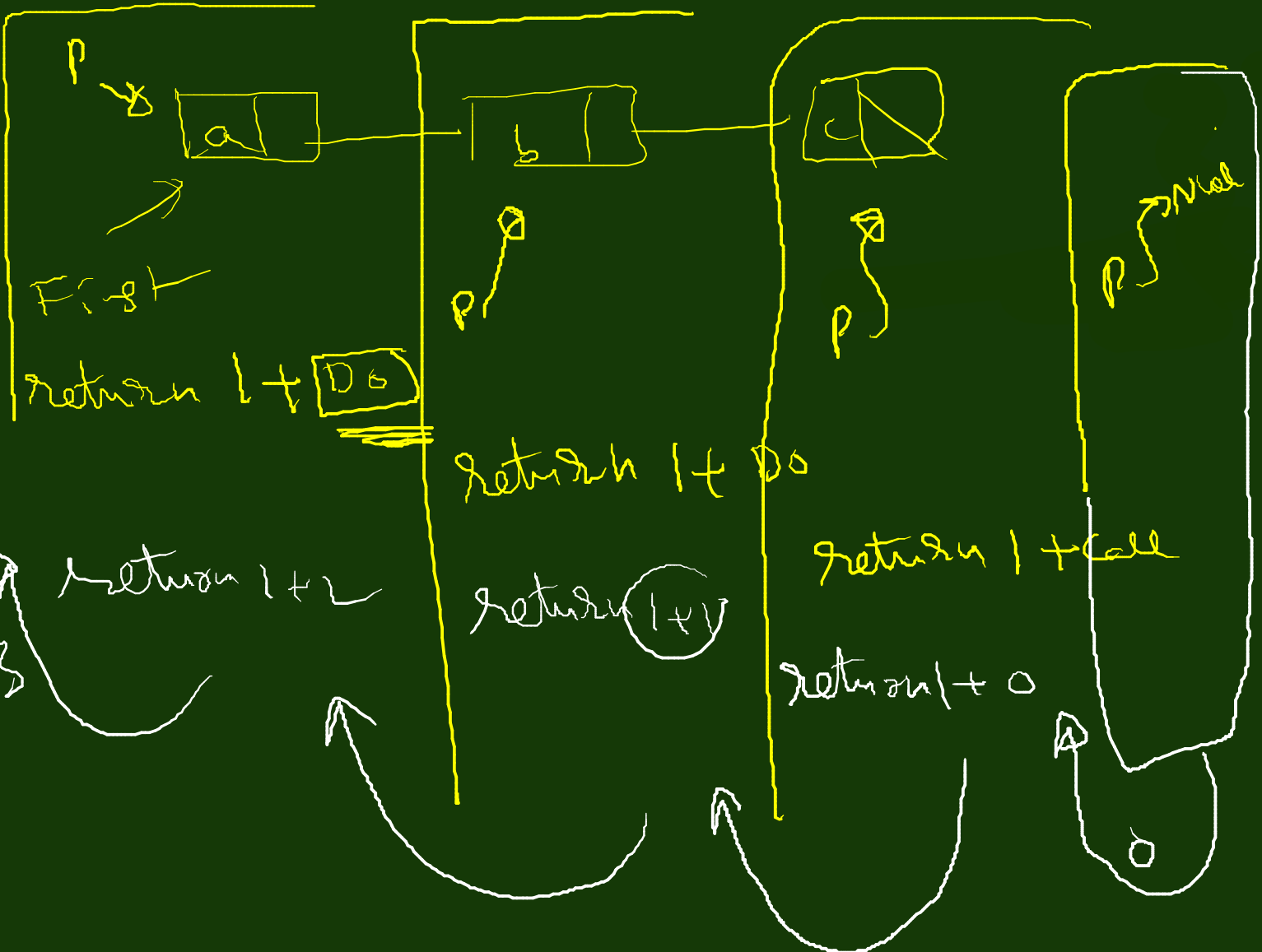
int DO ( struct Node * p )
{
    if ( p ) or if ( p != null )
        return 1 + DO ( p -> link );
    else
        return 0;
}

```

9 on the main()


Invoke

DO (First)



main
Invocation DO(First)


In the function DO(P)



P = First;

In DO(P)

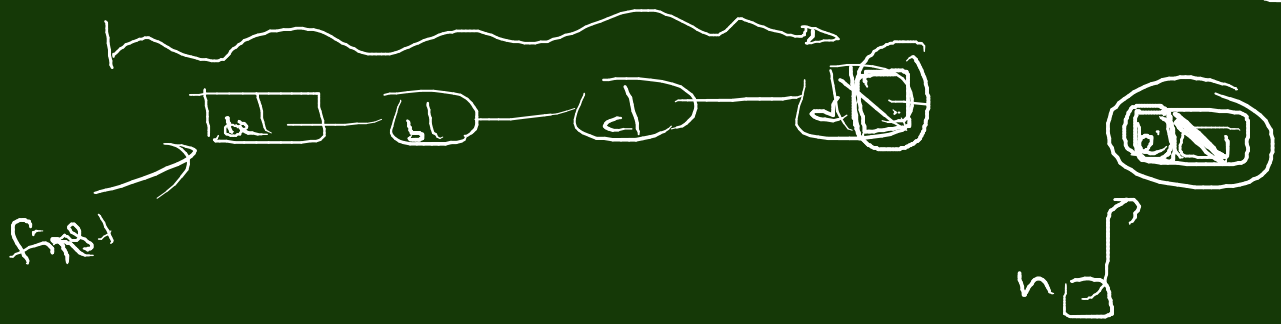
DO(P → link)



P = P → link

Forward

Write a routine to insert in the last

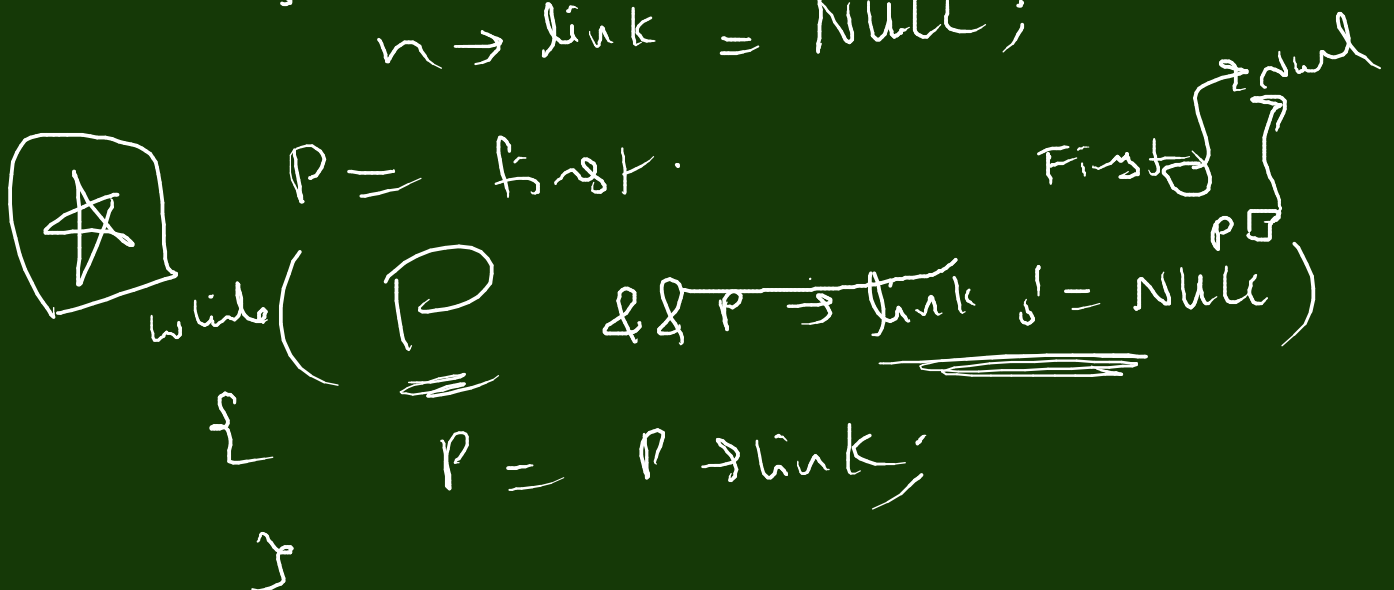


While ($P \neq \text{NULL}$)
 Entire list is traversed and
 in the end 'P' point to NULL

```
void InsertLast(char x)
{
    struct Node *p;
    struct node *n = New Node();
```

$n \rightarrow \text{data} = x;$

$n \rightarrow \text{link} = \text{NULL};$



If $P == \text{NULL}$

$\text{first} = P;$

else

$P \rightarrow \text{link} = n$

Linked Lists

Dynamic Memory allocation

(i) void * malloc (Size)

(ii) char * malloc (Size)

in C malloc() \equiv free()
in C++ new \times delete

{ p = malloc()

free(p)

} // p goes out of scope

After

free(p)

p is still alive

p = NewNode();

p = malloc()

if 'p' is to be deleted then

'p' should go out of scope

{
}