Linked list

↓

Data structure.

Pseudo code

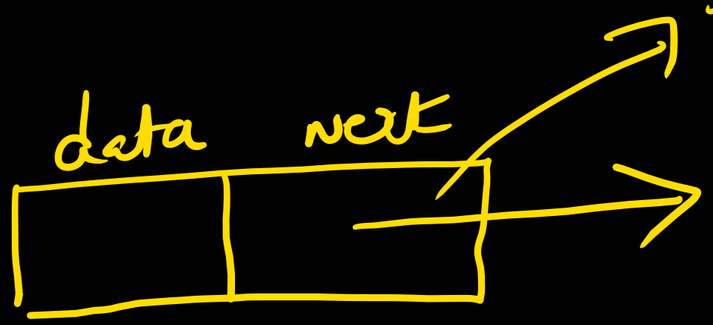
↓

C like language ~~C++~~ no programming language is required

But we will follow

C-like program style

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



it refers to data structure of same time. It is called self referential structure.

next is a pointer to the node structure.

Linked list

↳ C

↳ C++

↳ Python

C - like language.

Data structures are not related to programming language.

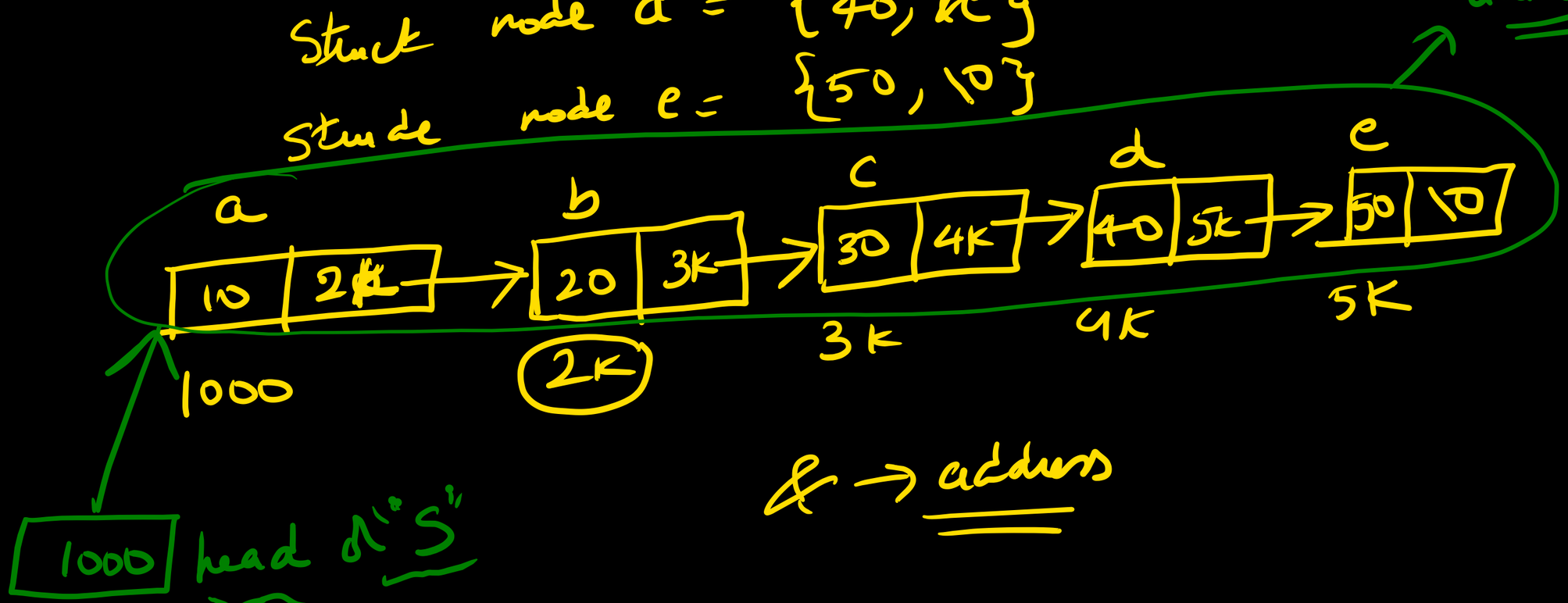
Struct node *head;

head = &a;

```
Struct node a = {10, &b}
Struct node b = {20, &c}
Struct node c = {30, &d}
Struct node d = {40, &e}
Struct node e = {50, 10}
```

```
Struct node
{ int data;
  Struct node * next
};
```

linked list.



& → address

a

data next

10

1000

struct node *b = &a.

b = 1000.

"b"

*b

((*b).data)

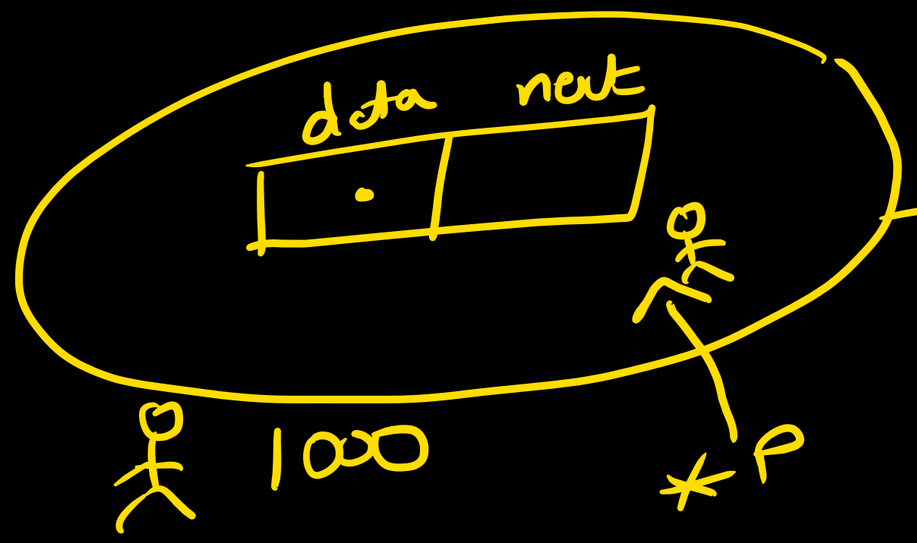
10

1000

"b"

*b'

p 1000



Fol your understanding.

$(*p).data \equiv p \rightarrow data$ ✓

"p"

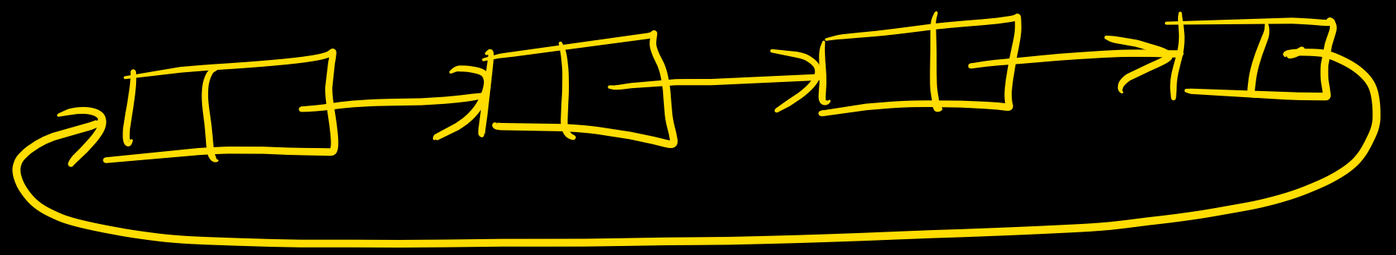
p = 1000

$p \rightarrow data$

$p \rightarrow next$

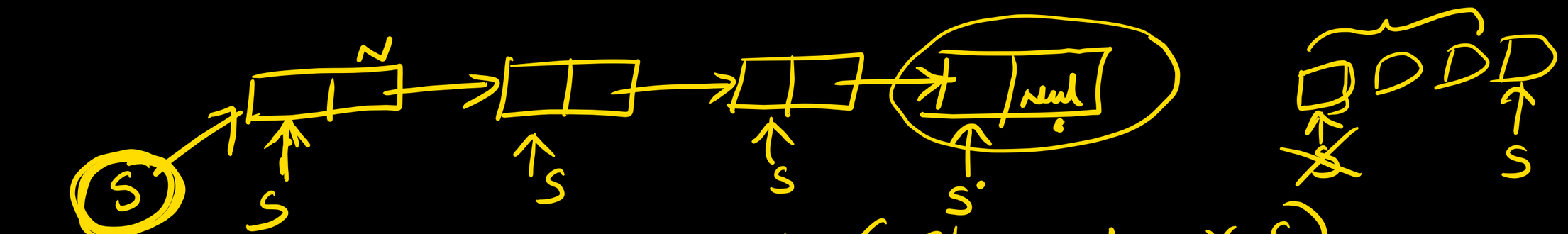
Write an algo to find last node of a Single LL?

SLL



Circular SLL.

Double linked list.

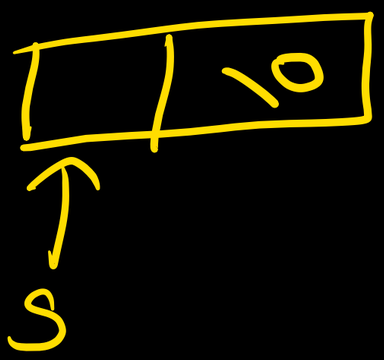



```
(struct node *)Find last node add ( struct node * s)
{   if (s==NULL)(pf(LL empty
                   return NULL)
    s1 = s;
    while( s1 -> next != NULL)
        s1 = s1 -> next;
    return s1;
}
```

Write an algo to find add of 2nd last node in a given SLL

① if (S == null) return null ✓ → LL is empty

② if (S → next == null) return null ✓



⟹ only one element is present.

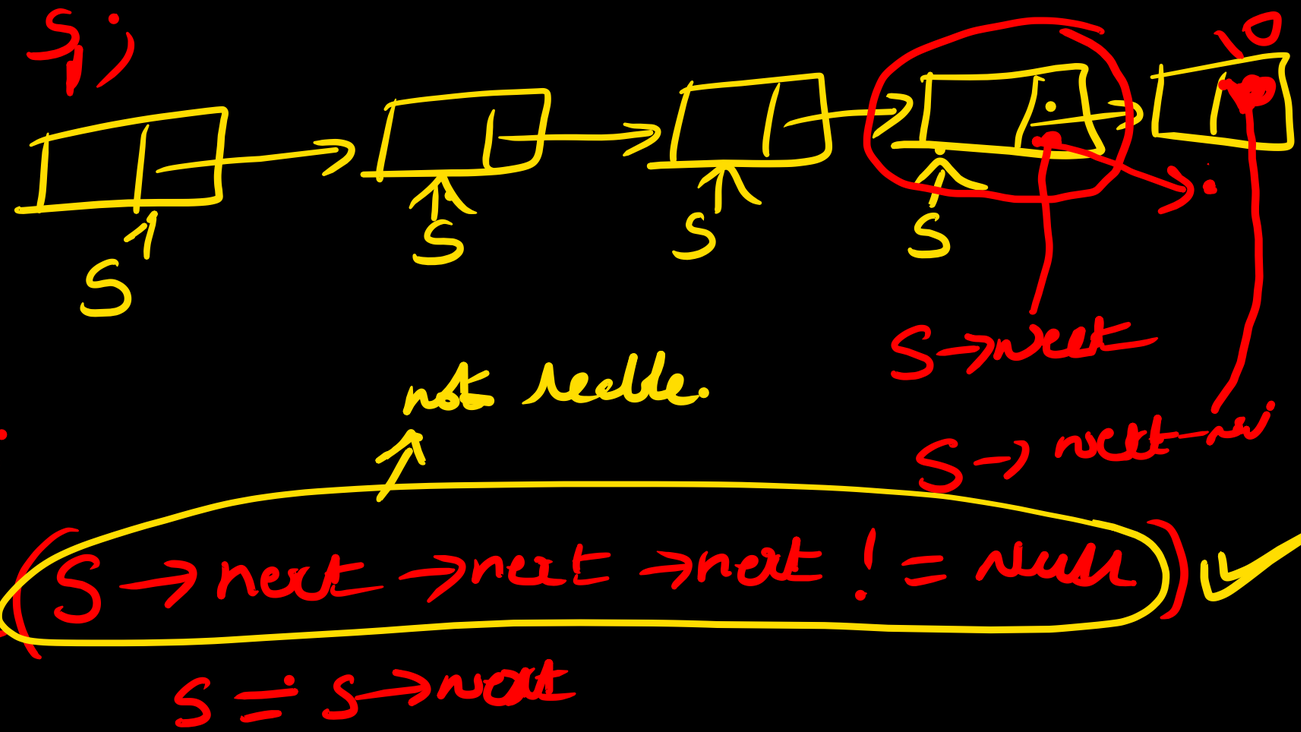
We should have at least 2 nodes

$s_1 = s$

while ($s_1 \rightarrow next \rightarrow next$! null)

$s_1 = s_1 \rightarrow next;$

return s_1;



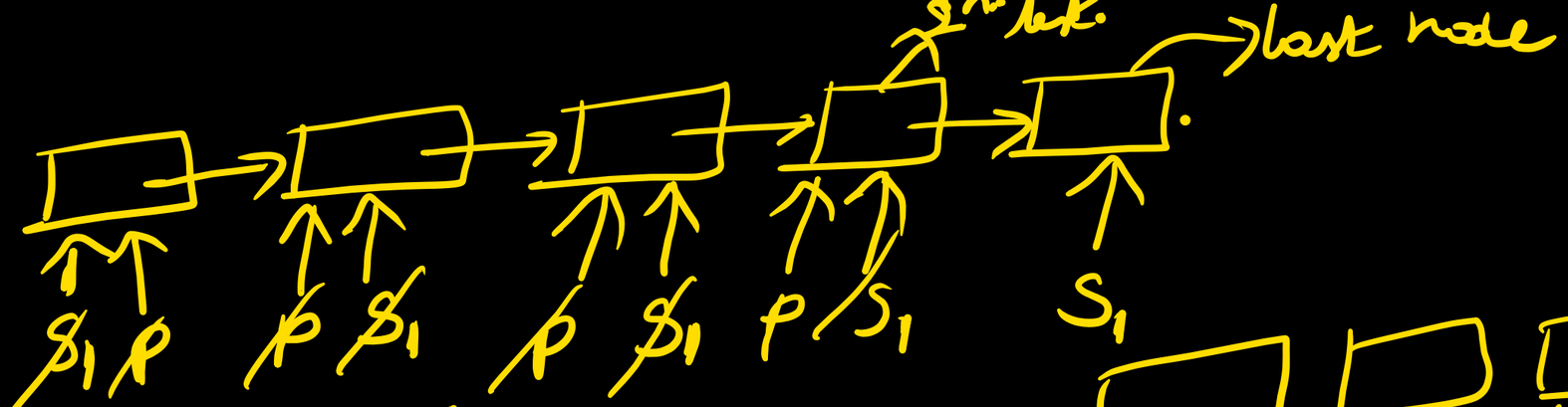
3rd last:

not needed.

while (s -> next -> next -> next != null)

s = s -> next

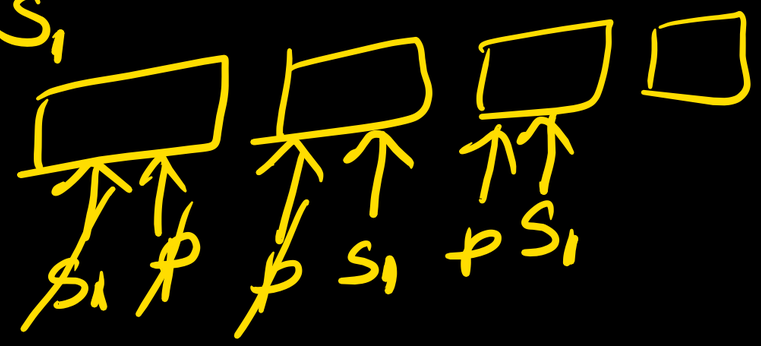
"S" is the head.

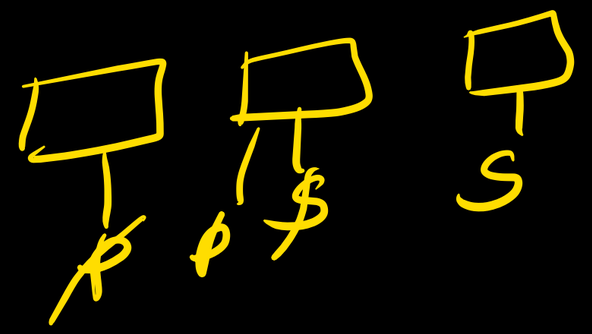


$S_1 = S$ ✔

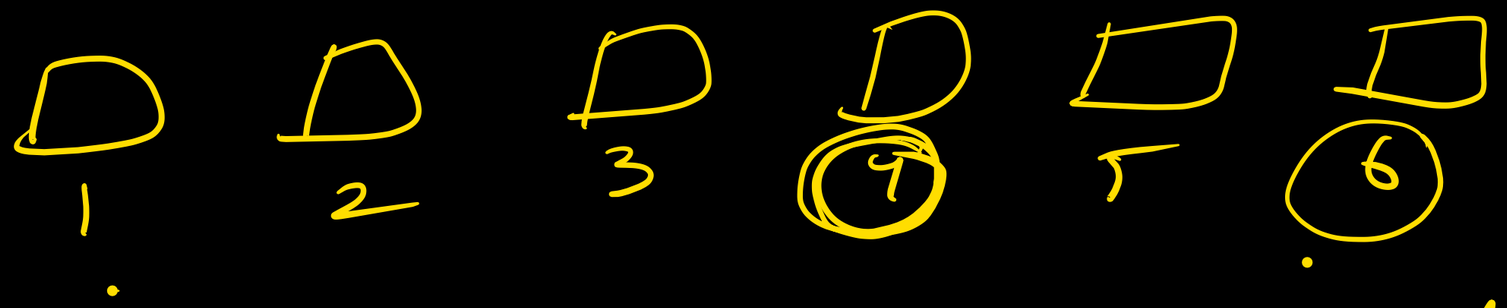
O(n)

```
while( S1 -> next != null)
{
    p = S1
    S1 = S1 -> next
}
return p;
```





Write an algo to find address of K^{th} node from last.



I pass - count

Count = 6

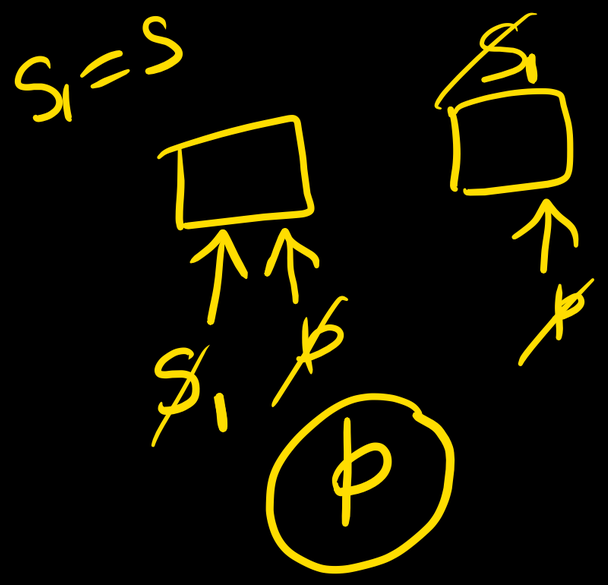
II pass 3rd node = '4'

then print '4'.

3rd node from the last.

I pass ⇒ count (7)

II pass ⇒ get the element (5) ✓



3rd element from the last.

move s and p one step each till p reaches last element

$k = 3$

$k = 5$

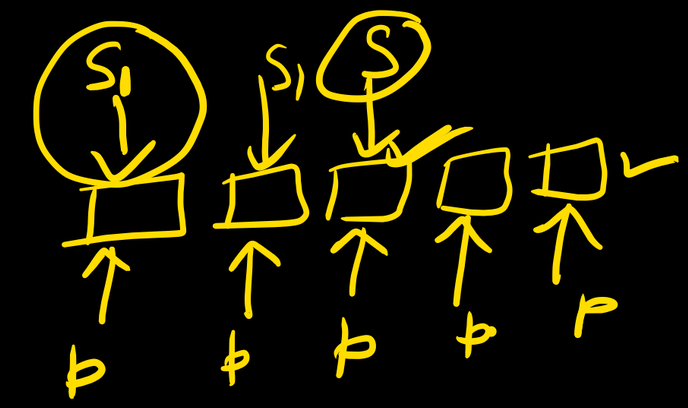
$k = 10$

```
if (S==null) return S;
    p=S
    while (p->next!=null && k!=1)
    {   p = p->next;
        k = k-1;
    }
if ( p->next == null)
```

False

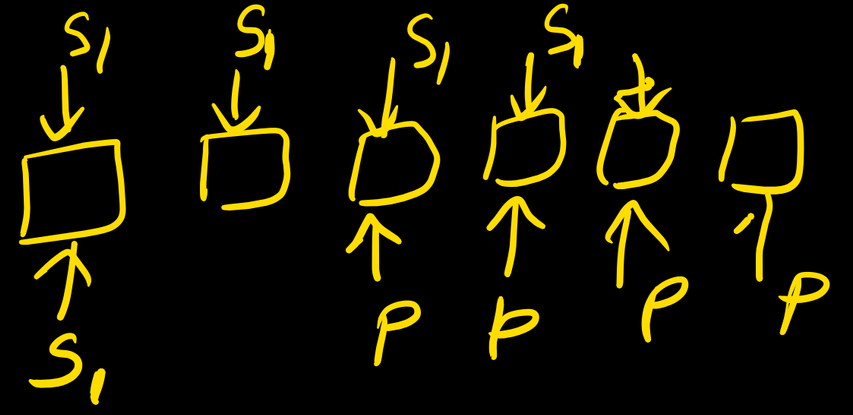
fals.

This means that size of LL is less than K.

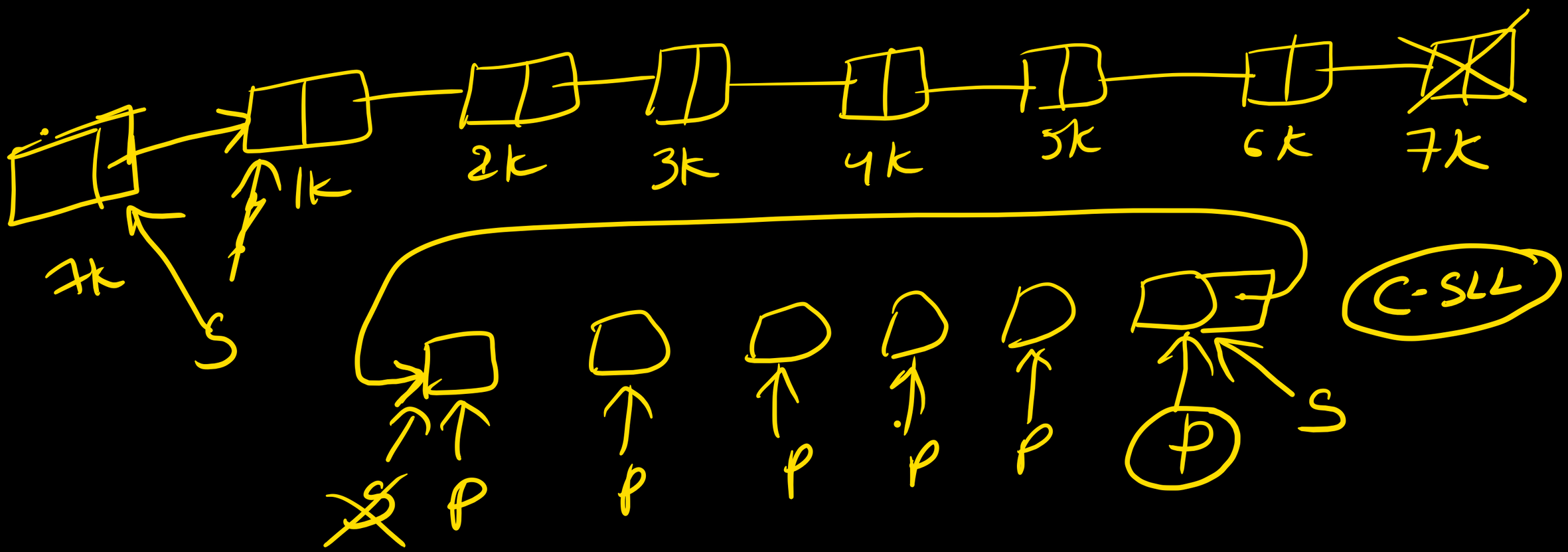


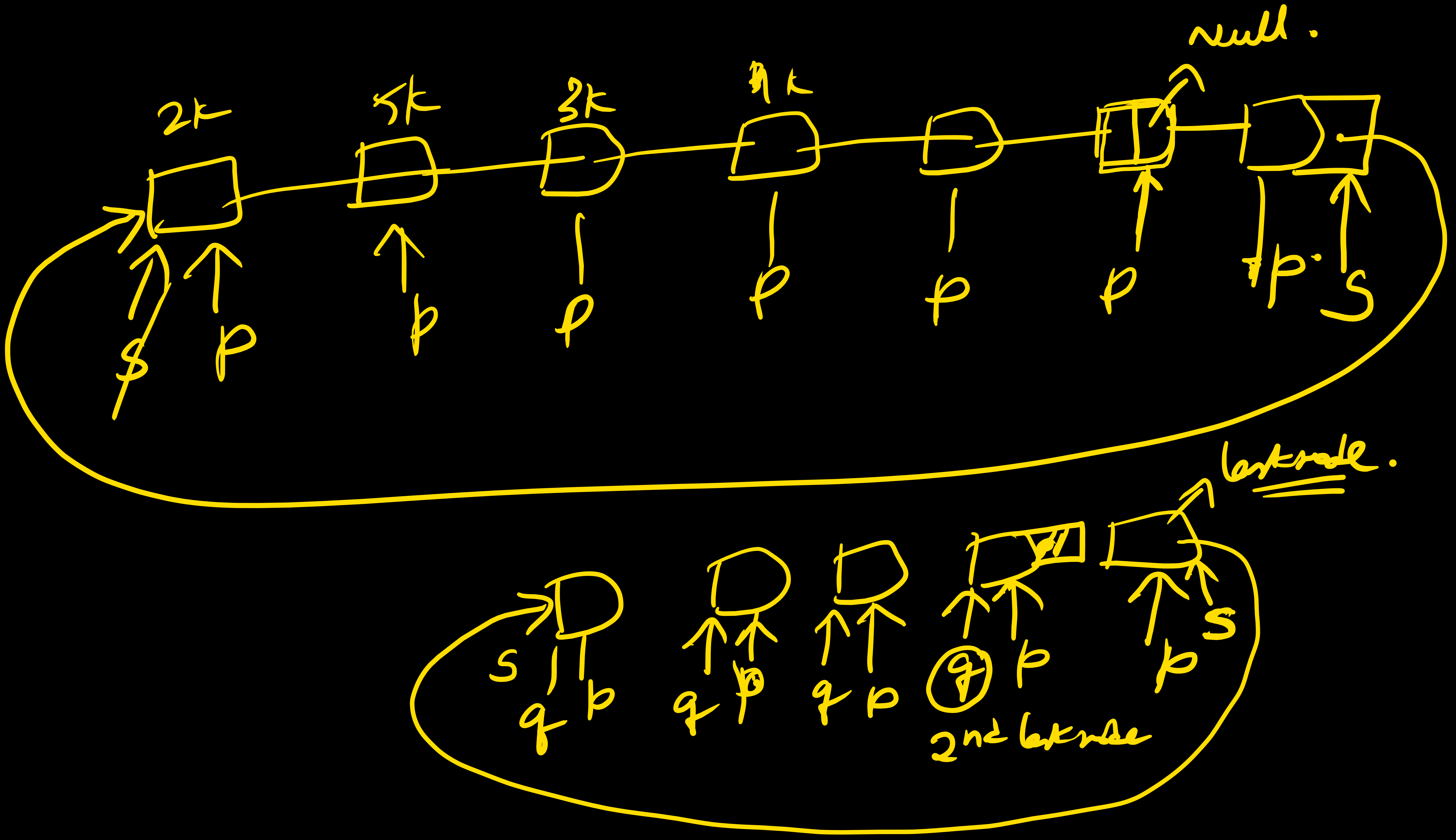
if K=3
=
p will move 2 time.

```
s1 = s
while (p->next != null)
{
    p = p->next;
    s1 = s1->next;
}
return s1;
}
```

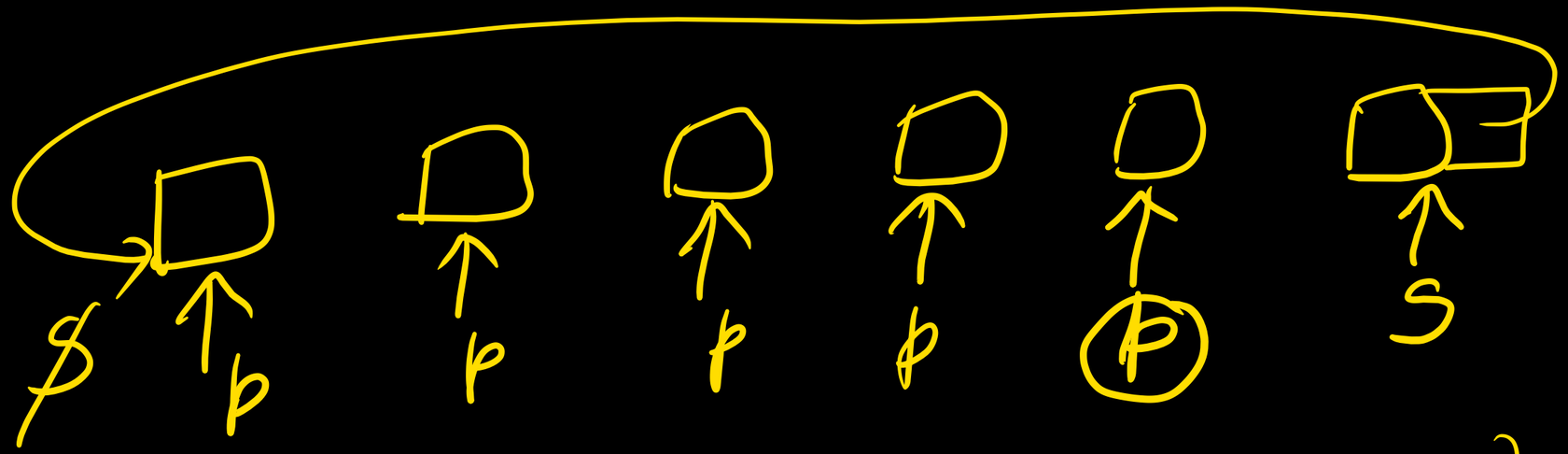


Write an algo to move last node of SLL to the first position





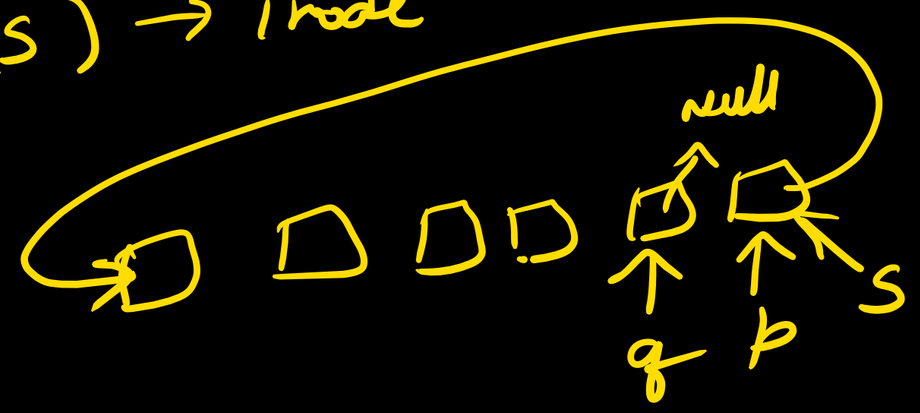
Null.
2k
5k
3k
s
p
lastnode.
2nd lastnode



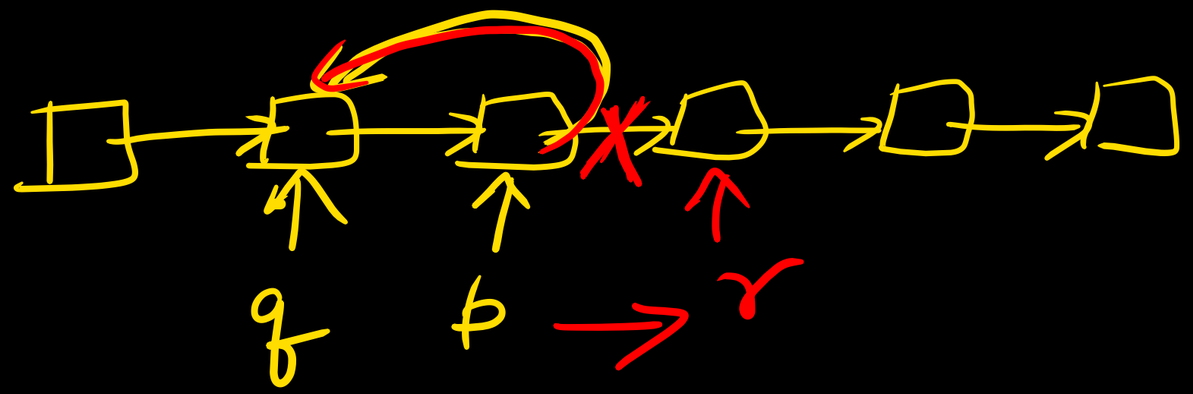
Single pointer. ⟸

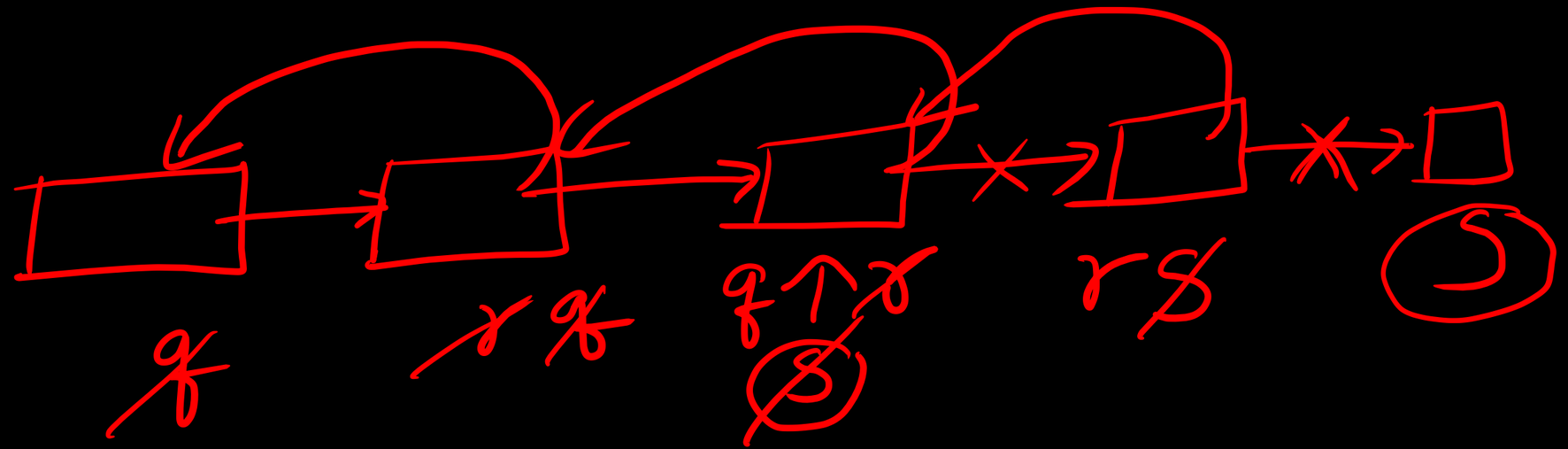
- $p \rightarrow next \rightarrow next = S$
- $S = p \rightarrow next$
- $p \rightarrow next = null$

```
if (S == null) return null  → 0 nodes
if (S->next == null) return (S) → 1 node

p = q = S;
while (p->next != null)
{  q = p;
   p = p->next
}
q->next = null; p->next = S; S = p; return(S)
```

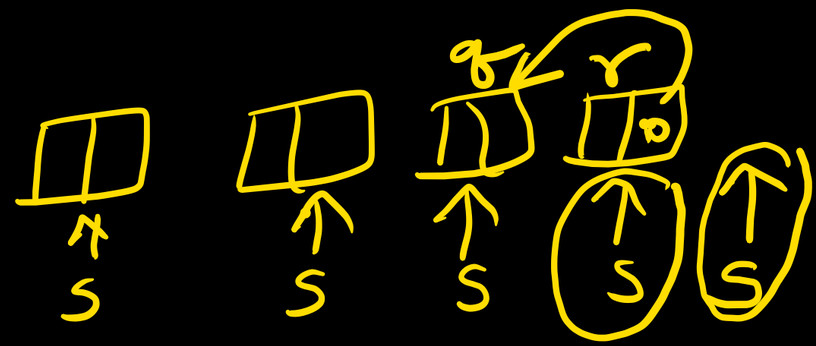


write an algo to reverse a given SLL



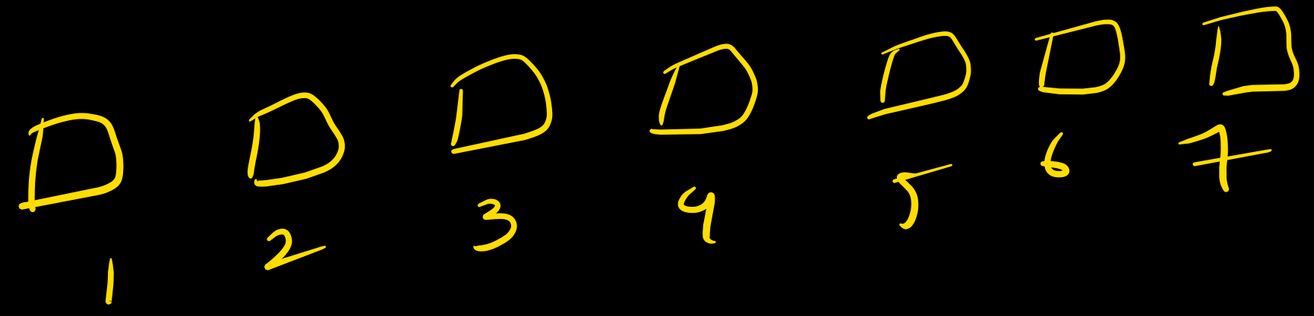


```
q = r = null
while ( s != null)
{   q = r;   → q is following r
    r = s;   → sr is following s
    s = s -> next;
    r -> next = q;
}
    s = r;
    return(s);
```



q → r → s

Write an algo to find address of middle node in SLL.

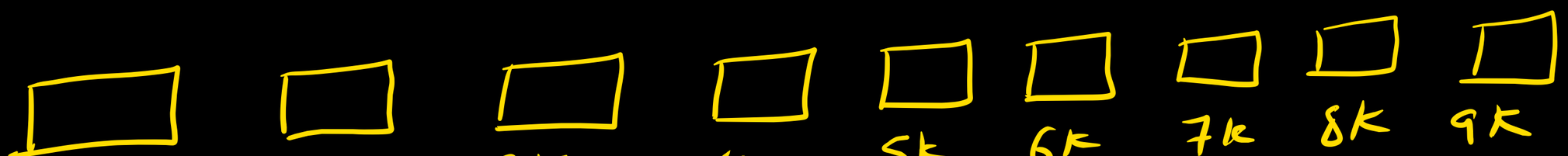


(7) → number.

then 4 is middle

1 pass ← 2 passes: I pass – count
II pass – get middle

Assum that LL Contains odd no of nodes



move by 1 → p

q → moves 2

p	q
1k	1k
2k	3k
3k	5k
4k	7k
5k	9k.

By the time 'q' reaches end, p will be at the middle.

~~Bf~~

```
p = q = s
while ( q != null && q->next != null && q->next->next != null)
{   p = p->next;
    q = q->next->next;
}
return(p);
```

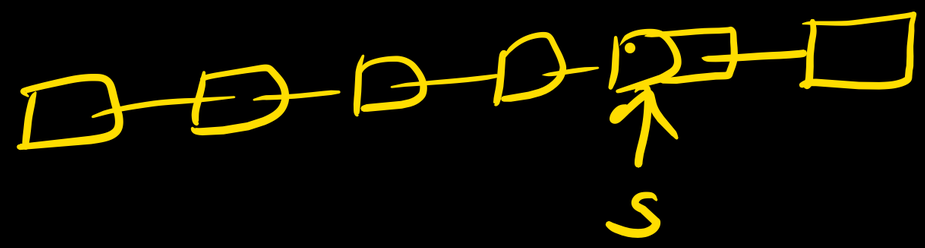
Write an algo to insert a node at the end of the SLL

p = Create a node with data x and give its pointer

p → data = x

p → next = null

if (S == null) S = p return 'S';



$S_1 = S$

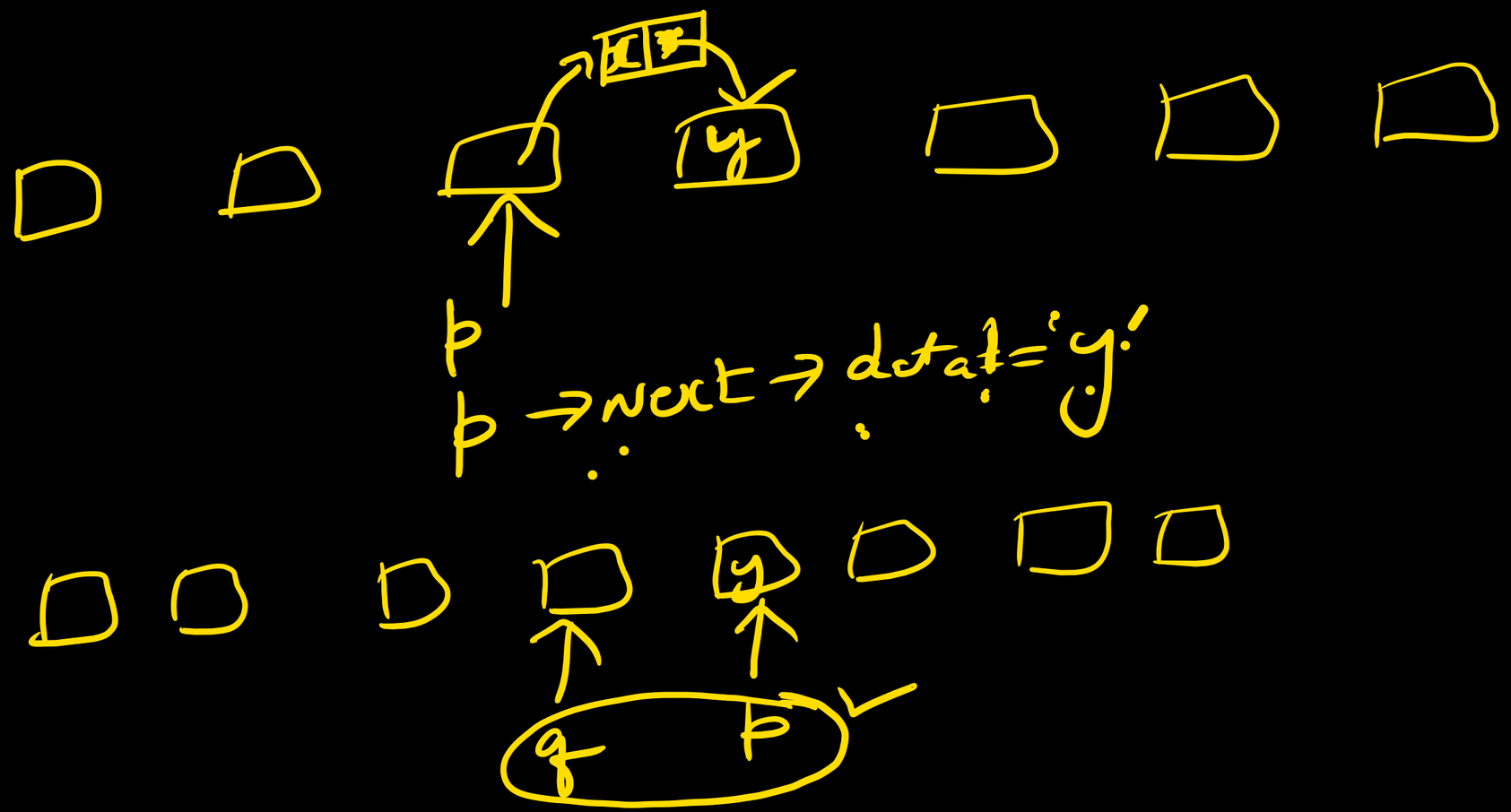
while (S_1 → next != null)

$S_1 = S_1$ → next;

S_1 → next = p;

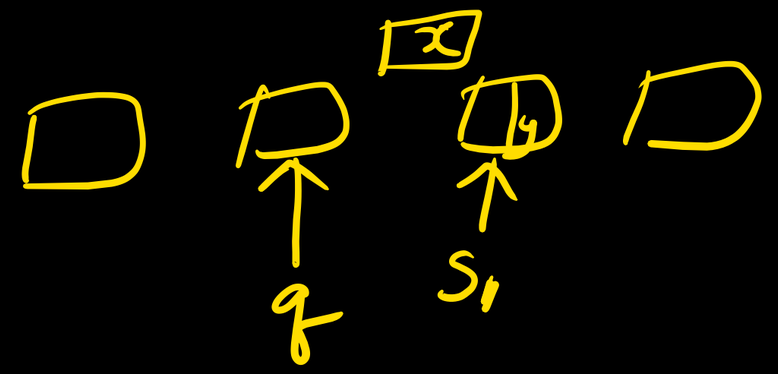
return 'S' ⟶ returning the first node address.

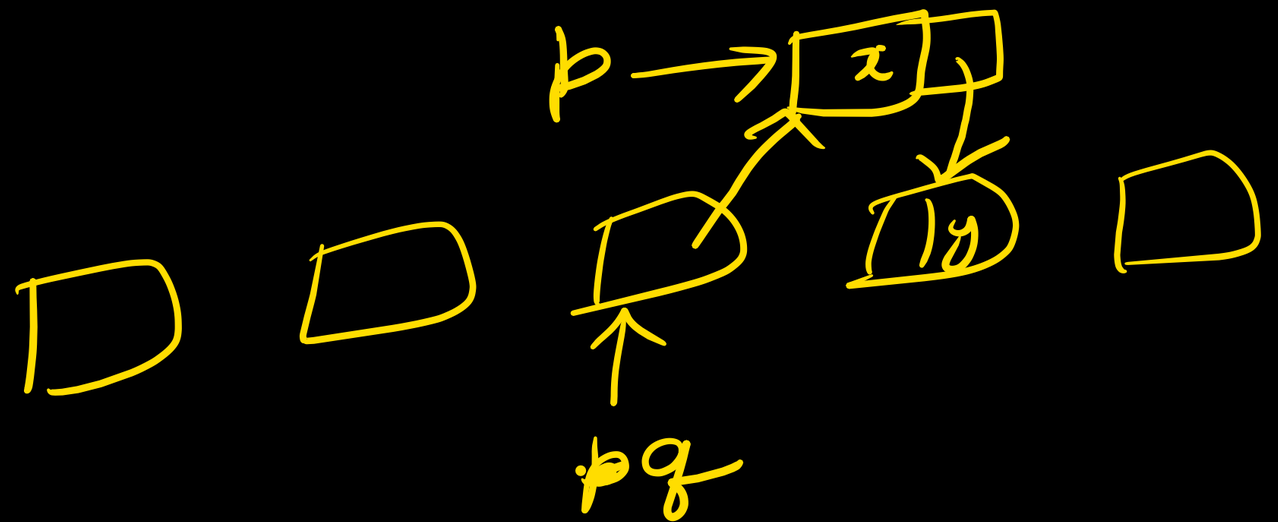
Write an algo to insert a node with data x before node with data y in SLL.



```
if (S==null) return 'S'
struct node *p = (struct node *) malloc (size of (struct node))
p->data = x;
S1 = S;
while (S1->data!=y && S->next!=null)
{
    q = S1
    S1 = S1->next
}
```

```
if (S1 -> data == y)
{
    q -> next = p
    p -> next = S1
    return (S);
}
Pf("no y in list")
return 'S'
```

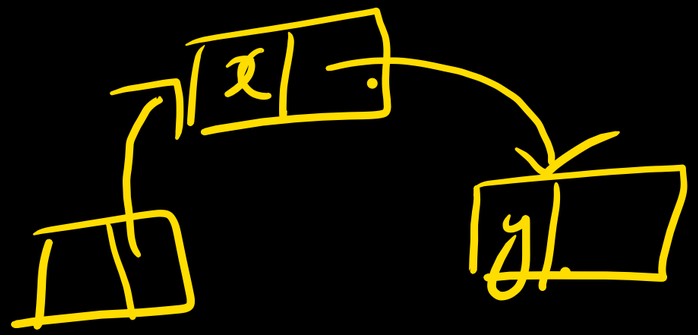




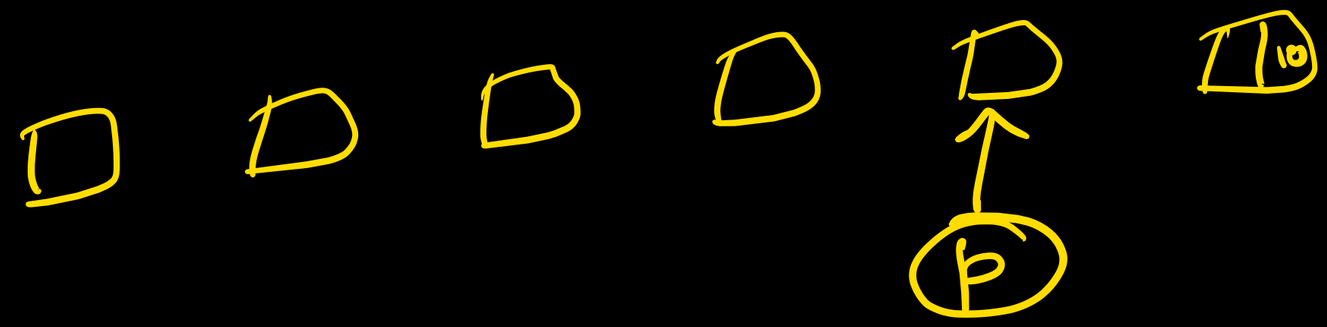
while(p → next → data != y)

p → next = q → next

q → next = p


x
y

Write an algo to delete a node at the end of SLL.



while(p → next → next != 0)

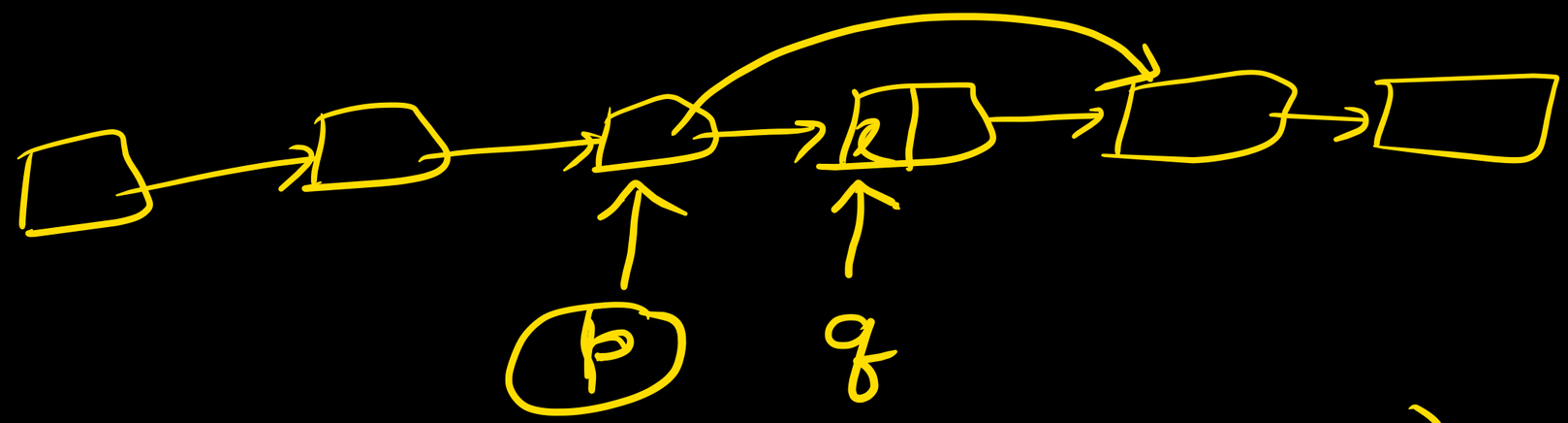
free up the space ← free(p → next).

p → next = null;

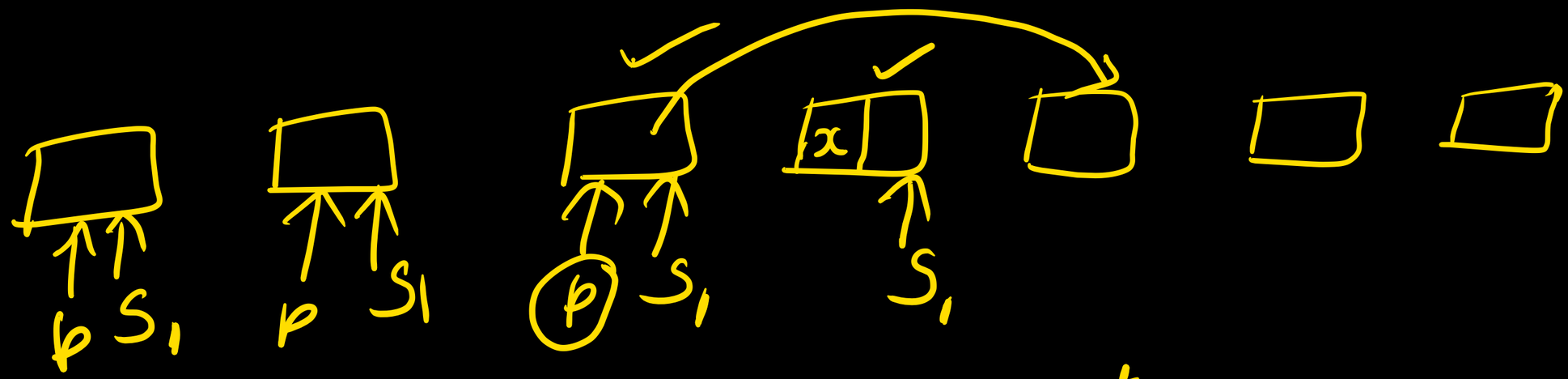
```
if (S == null) return S;

S1 = S
while ( S1 -> next != null)
  {  p = S1
     S1 = S1 -> next
  }
p -> next = null;
   free (S1)
return(S);
```

free means deallocate the memory allocated.

Write an algo to delete a node with data x.




```
while (p -> next -> data != x)
    q = p -> next;
p -> next = q -> next.
free(q).
```



$p \rightarrow next = x \rightarrow next$

$free(S_1)$.