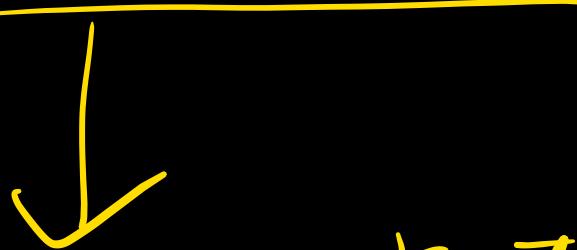


linked list



Data Structure.

Procedural code



C-like language



→ no programming language is required

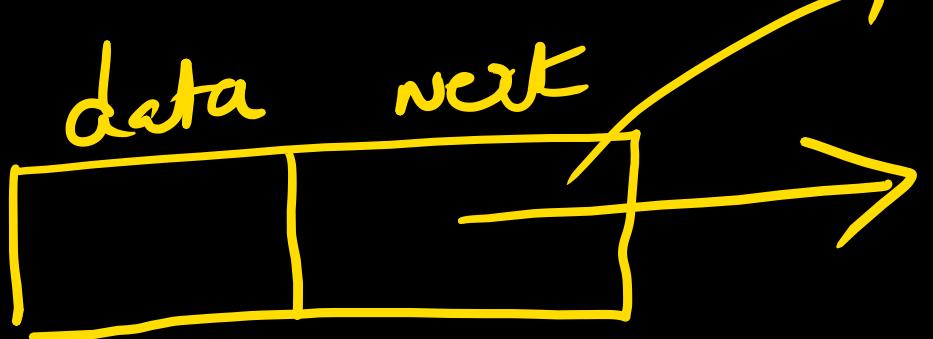
But we will follow

C-like program style

Struct node

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

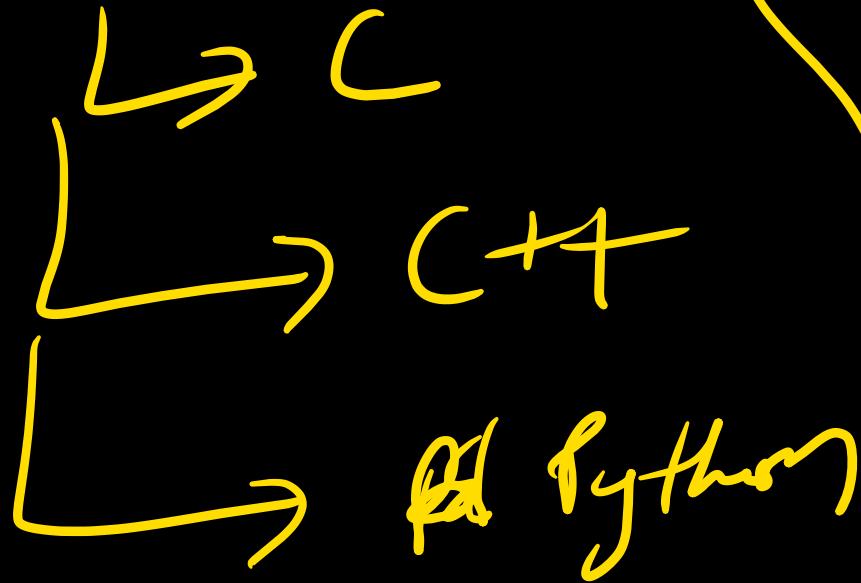
}



next is a pointer to a
node structure .

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

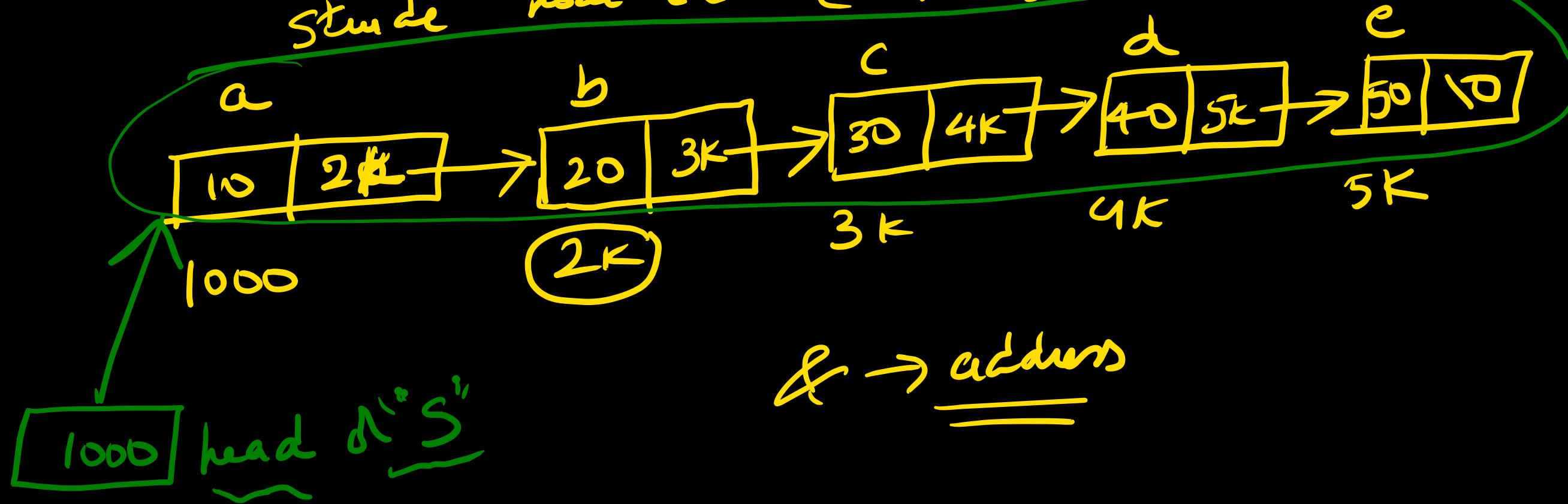
linked list



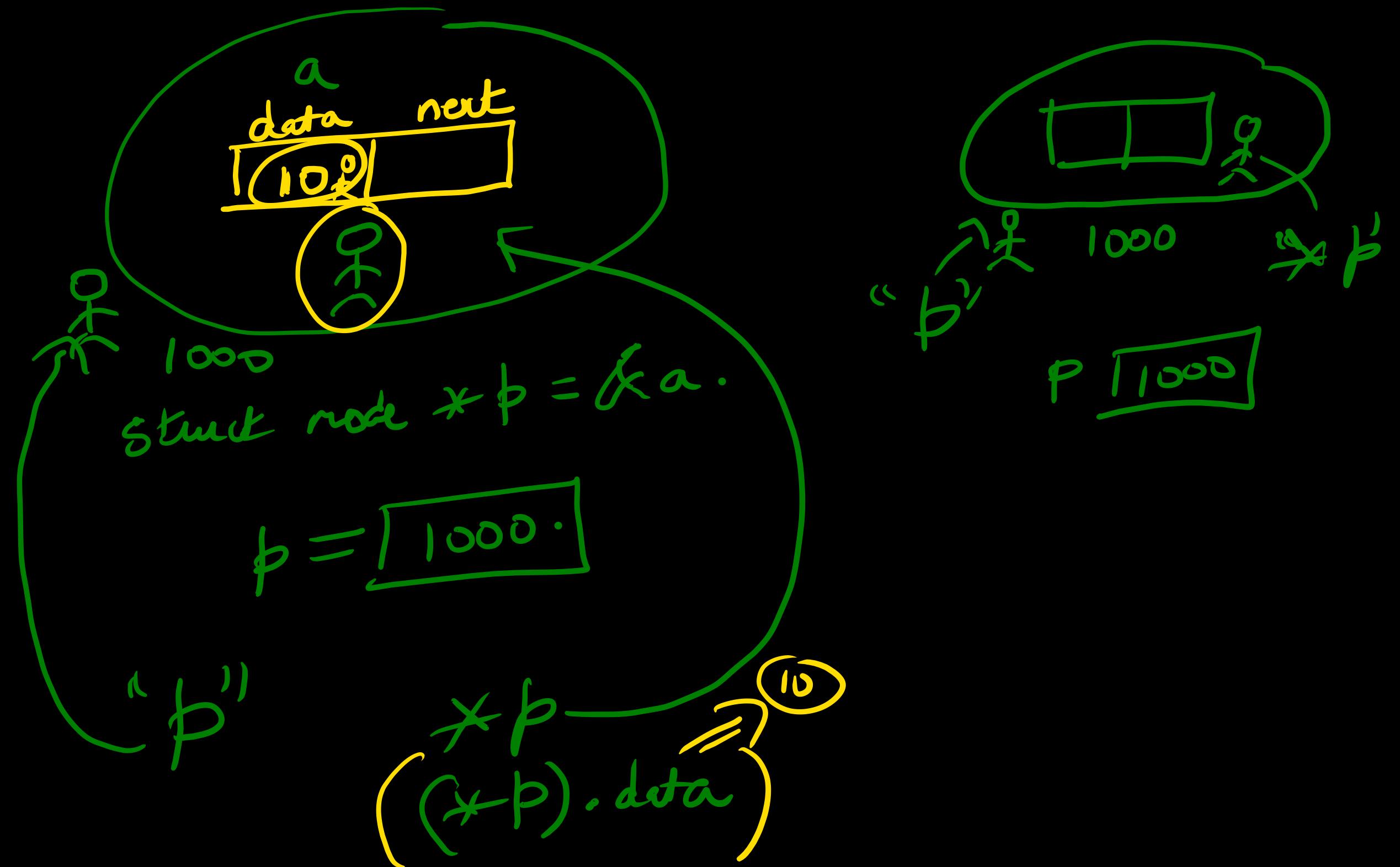
C-like lang.

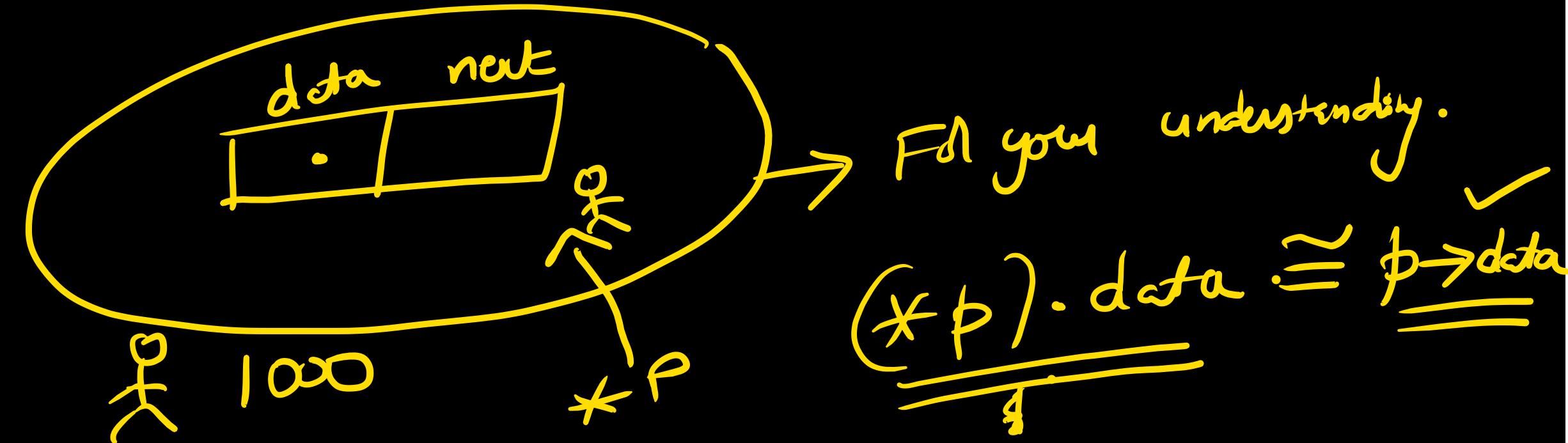
Data structures are not
related to programming
language.

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}



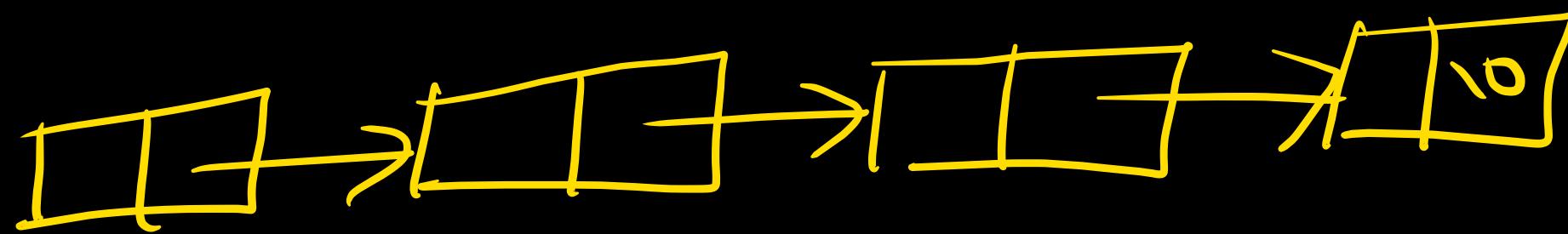
Struct node
 { int data;
 Struct node *next;



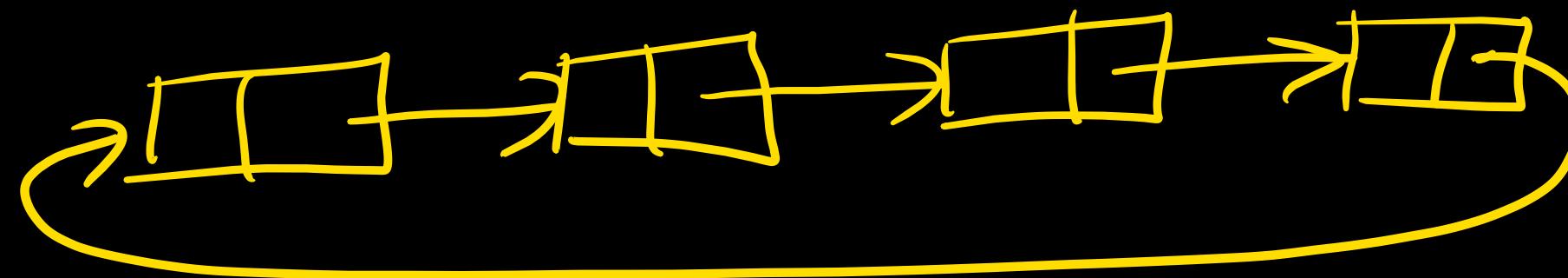


$p \rightarrow \text{data}$
 $p \rightarrow \text{next}$

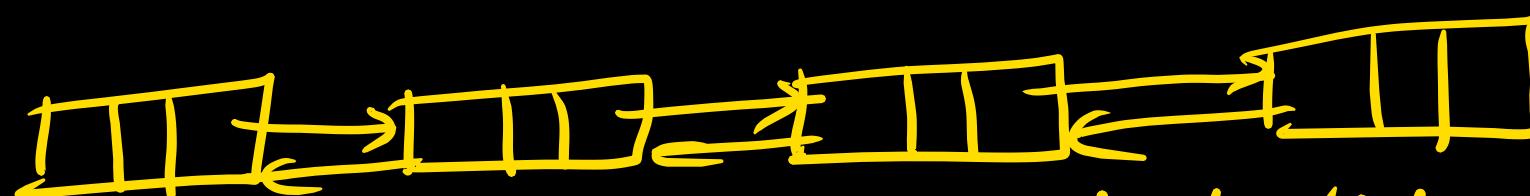
Write an algo to find last node of a Single LL?



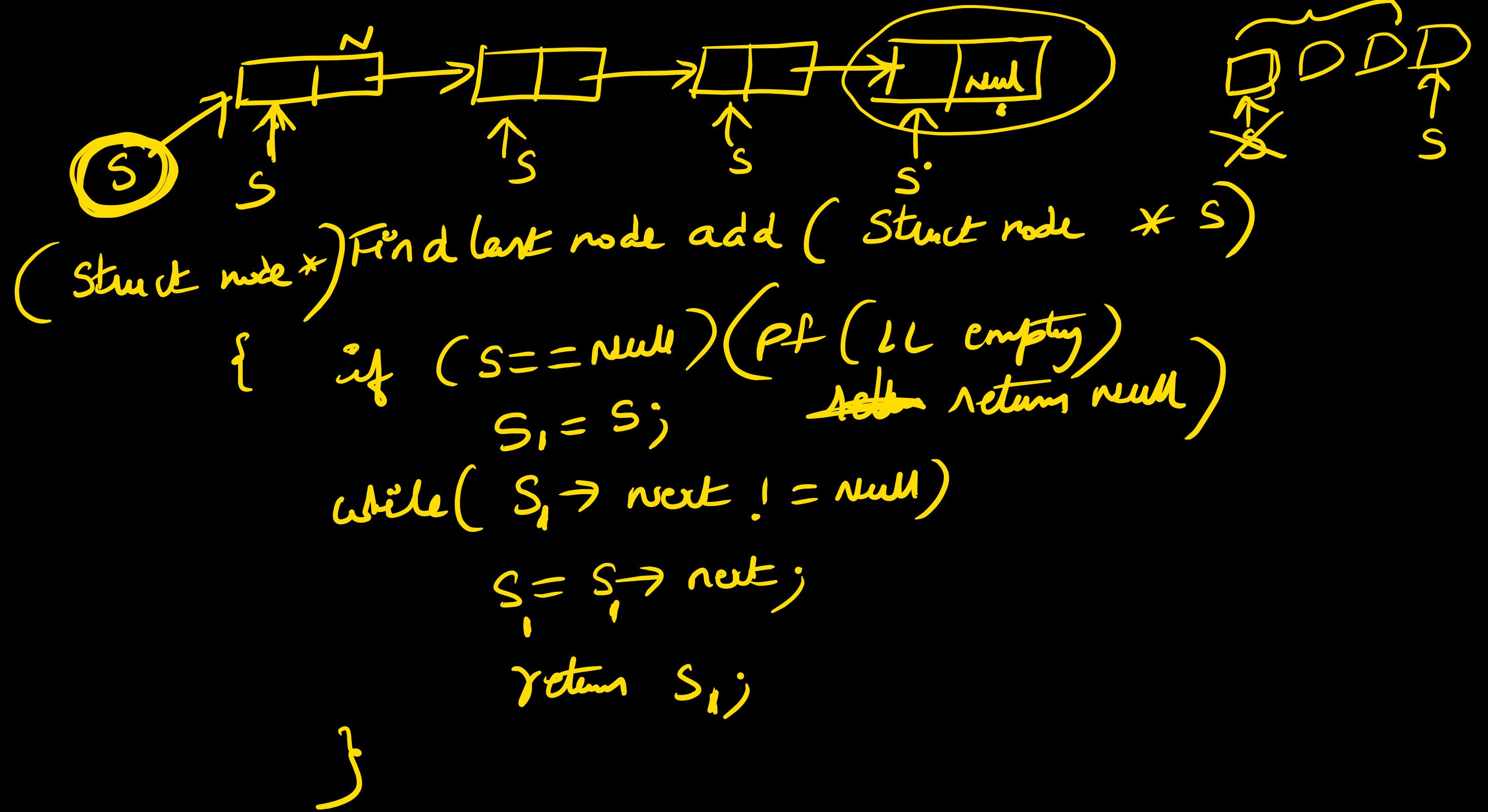
SLL



Circular SLL.



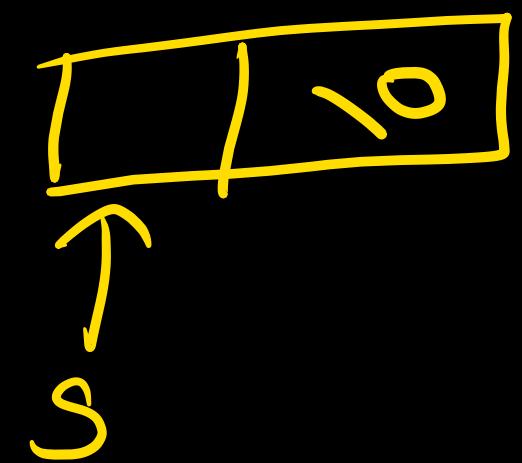
Double linked list.



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

① if ($s == \text{null}$) \rightarrow LL is empty
returns null ✓

② if ($s \rightarrow \text{next} == \text{null}$) returns null ✓
only one element is present.

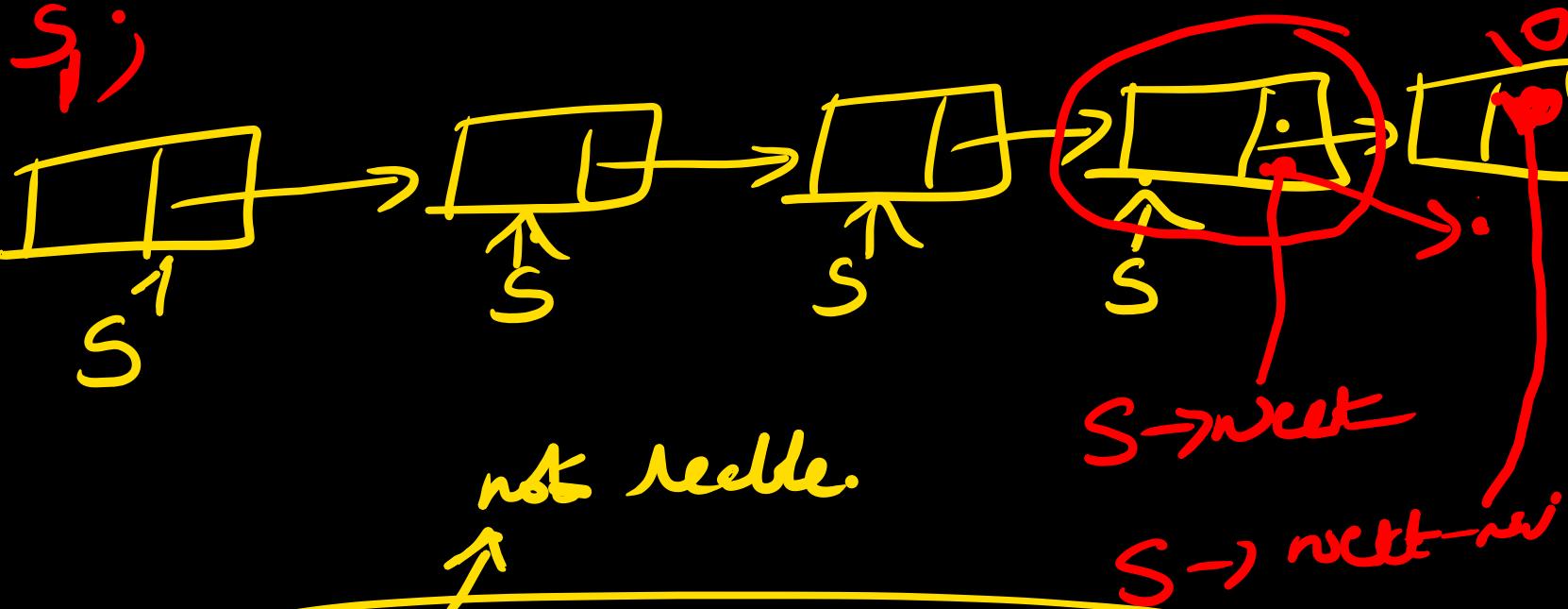


We should have at least 2 nodes

$s_1 = s$
while ($s_1 \rightarrow \text{next} \rightarrow \text{next} \neq \text{null}$)

$s_1 = \underline{s_1 \rightarrow \text{next}}$
 \equiv

return s_1 ;



3rd last:

while ($s \rightarrow \text{next} \rightarrow \text{next} \rightarrow \text{next} \neq \text{null}$)
 $s = s \rightarrow \text{next}$

“S” is the
node.

$O(n)$

while ($S_1 \rightarrow \text{next} \neq \text{null}$)

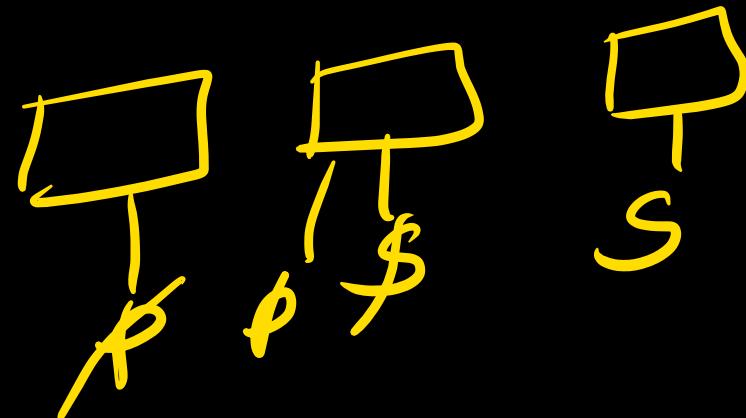
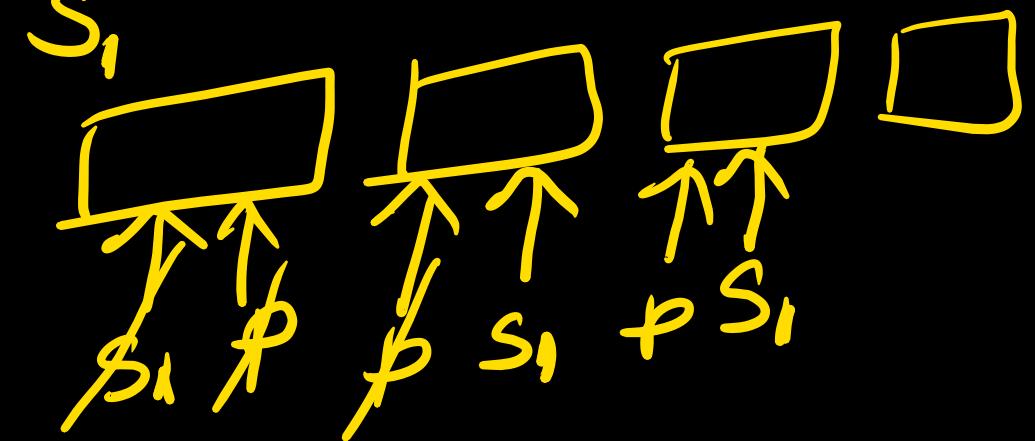
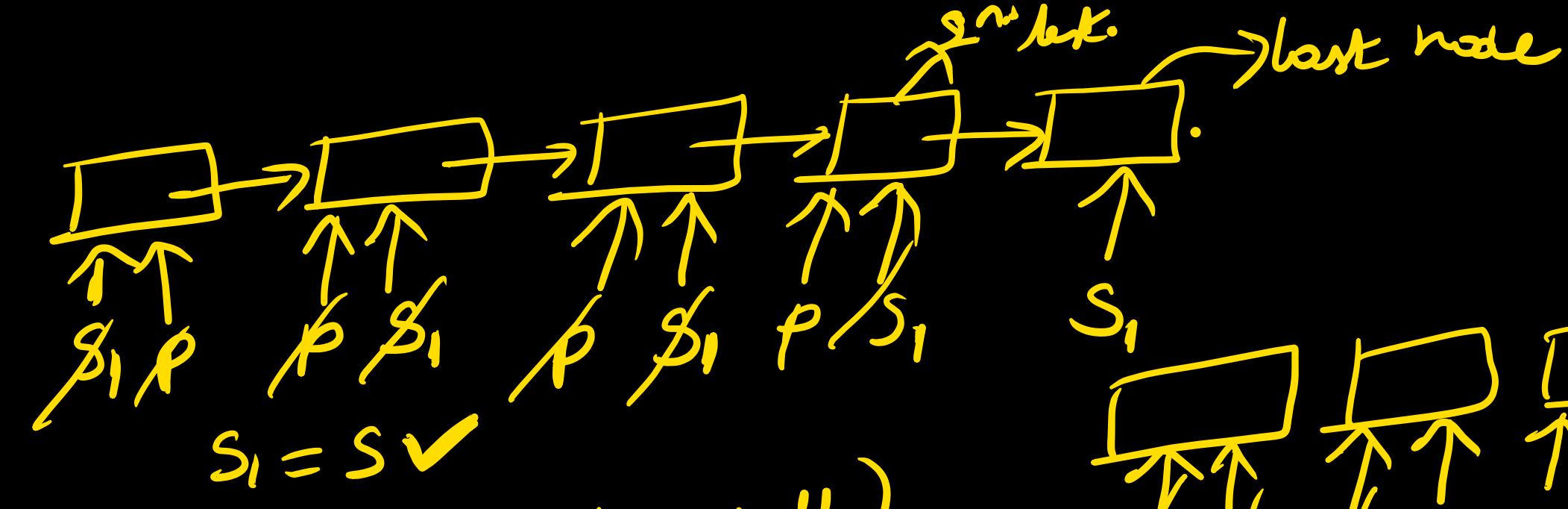
{

$p = S_1$

$S_1 = S_1 \rightarrow \text{next}$

}

return p ;



write an algo to find address of k^{th} node from last .

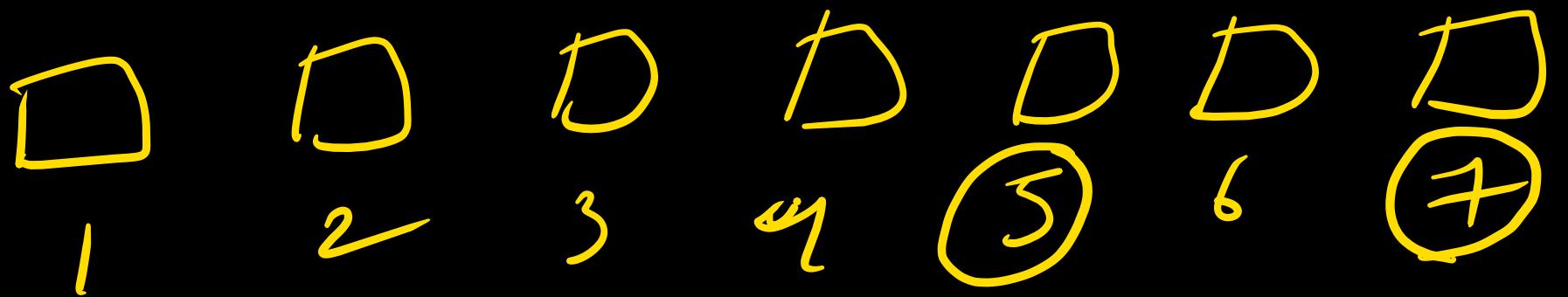


I pass - count

Count = 6

II pass
3rd node = (4)
then point '4'.

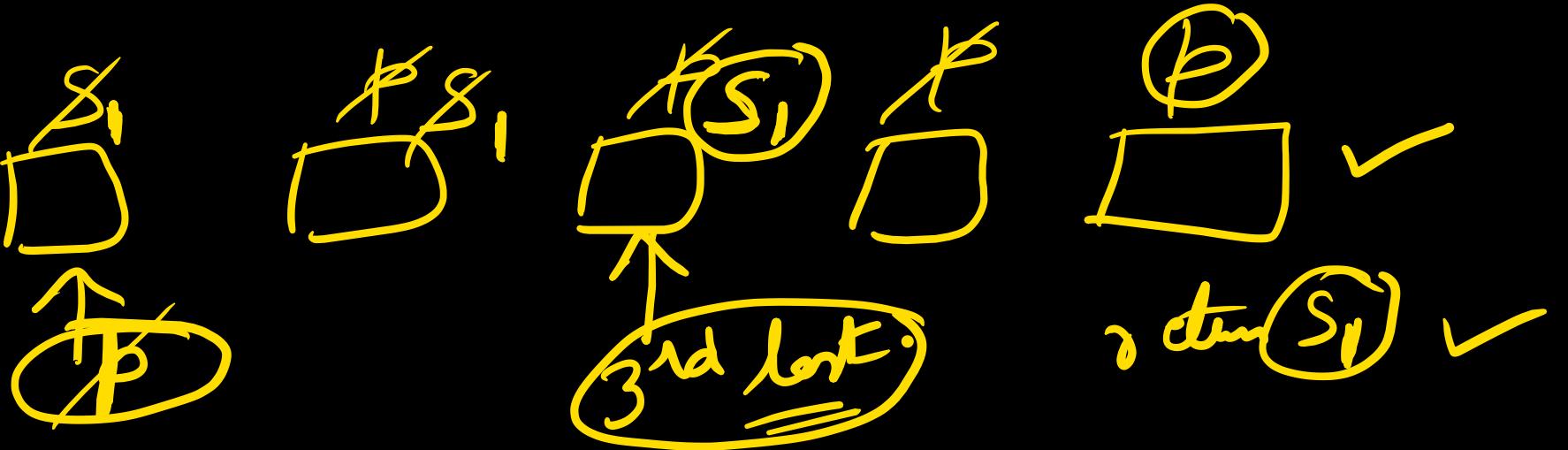
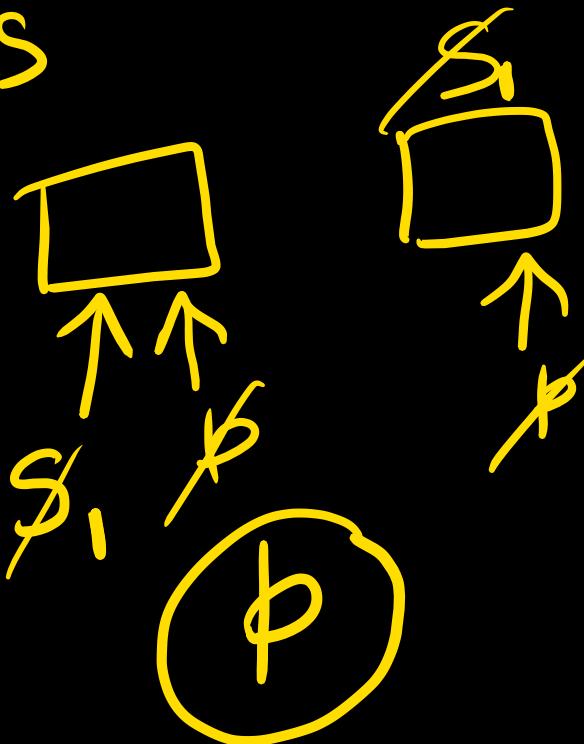
3rd node from the back.



I pass \rightarrow count(7)

II pass \rightarrow get the element(5) ✓

$s_i = s$



ϕ \equiv 3rd element from the last.
move s and ϕ one step each till
 left element $k = 3$

$$k=3$$

$$F = 5$$

$$k = \rho$$

if ($S == \text{null}$) return S ;

let $p = S$

while ($p \rightarrow \text{next} \neq \text{null}$) $\&$ $k \neq 0$

{ $p = p \rightarrow \text{next}$;

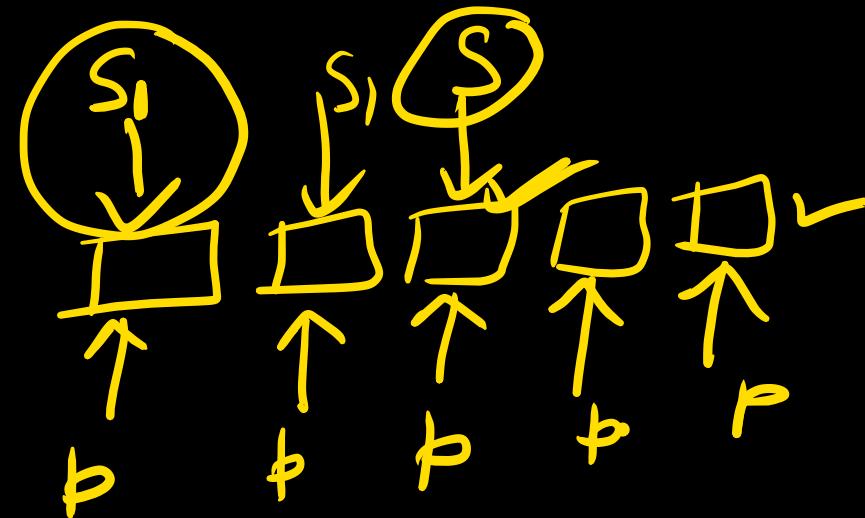
$k = k - 1$;

}

if ($p \rightarrow \text{next} == \text{null}$)

This means that size of
LL is less than ' k '.

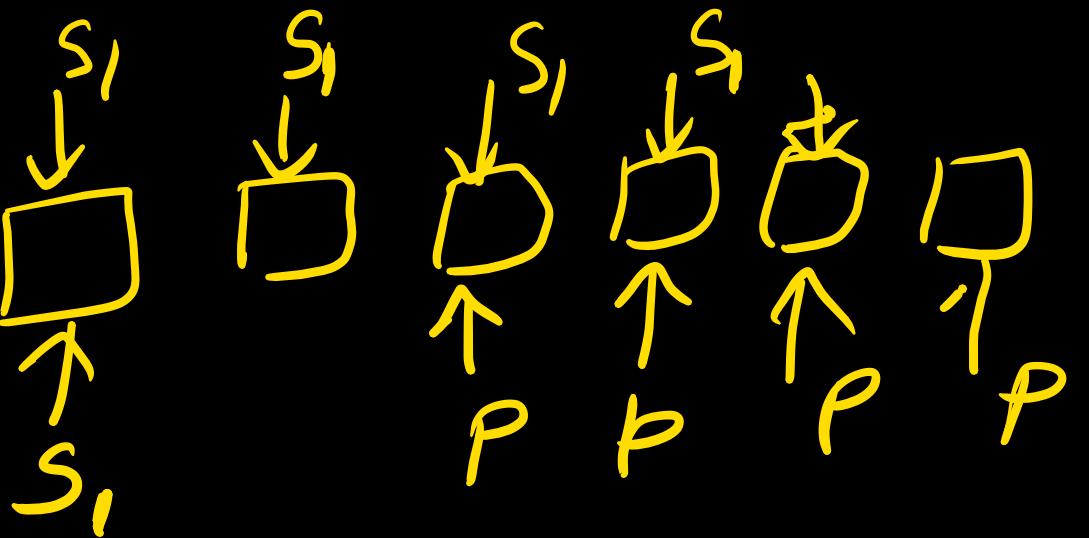
fake.



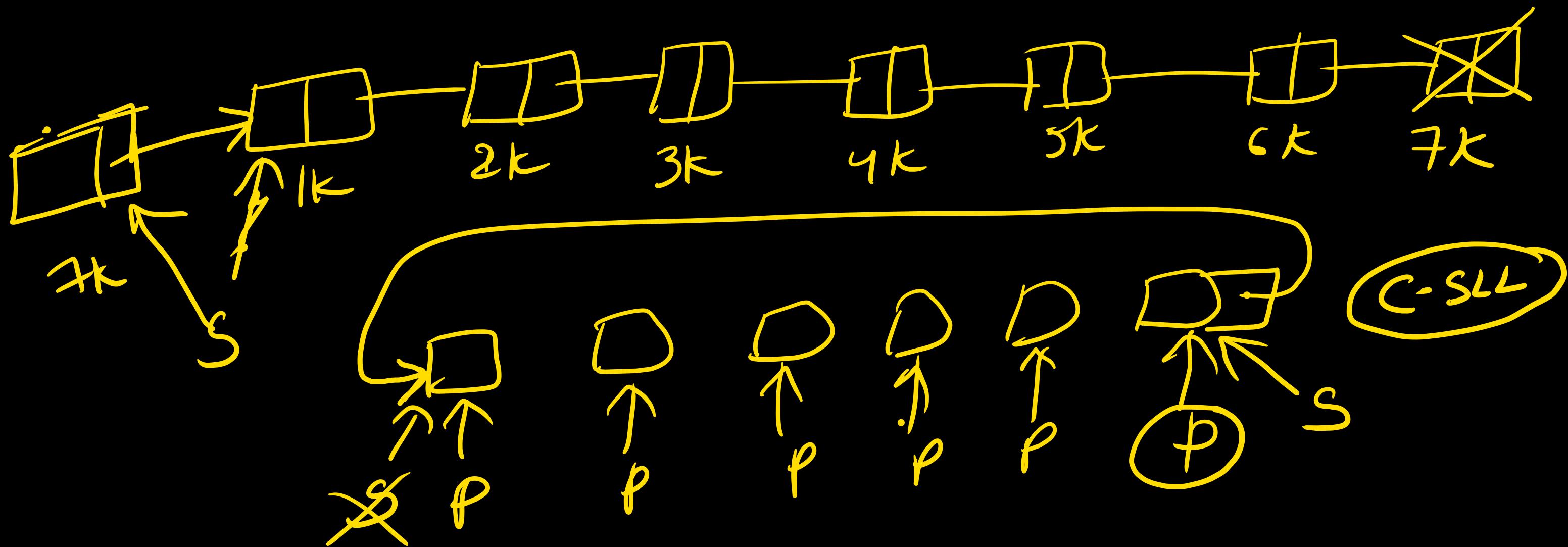
if $k = 3$
build more 2 times.

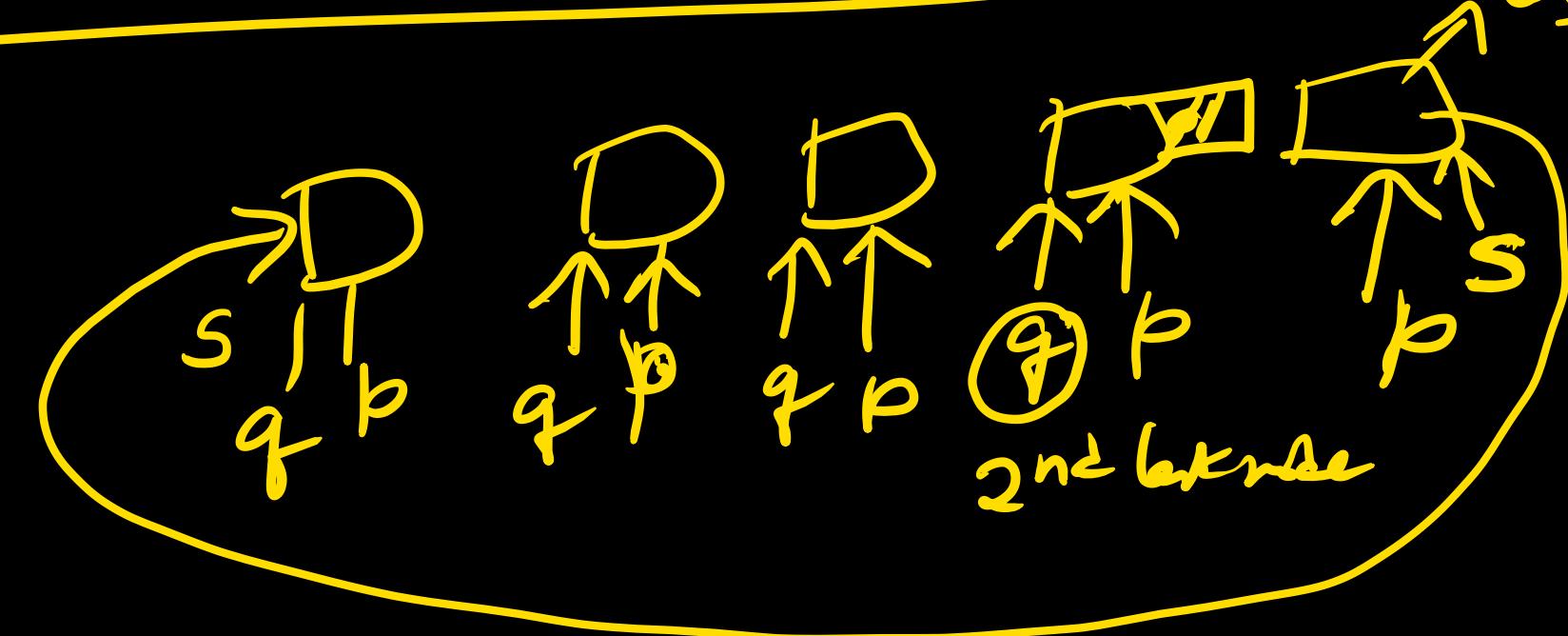
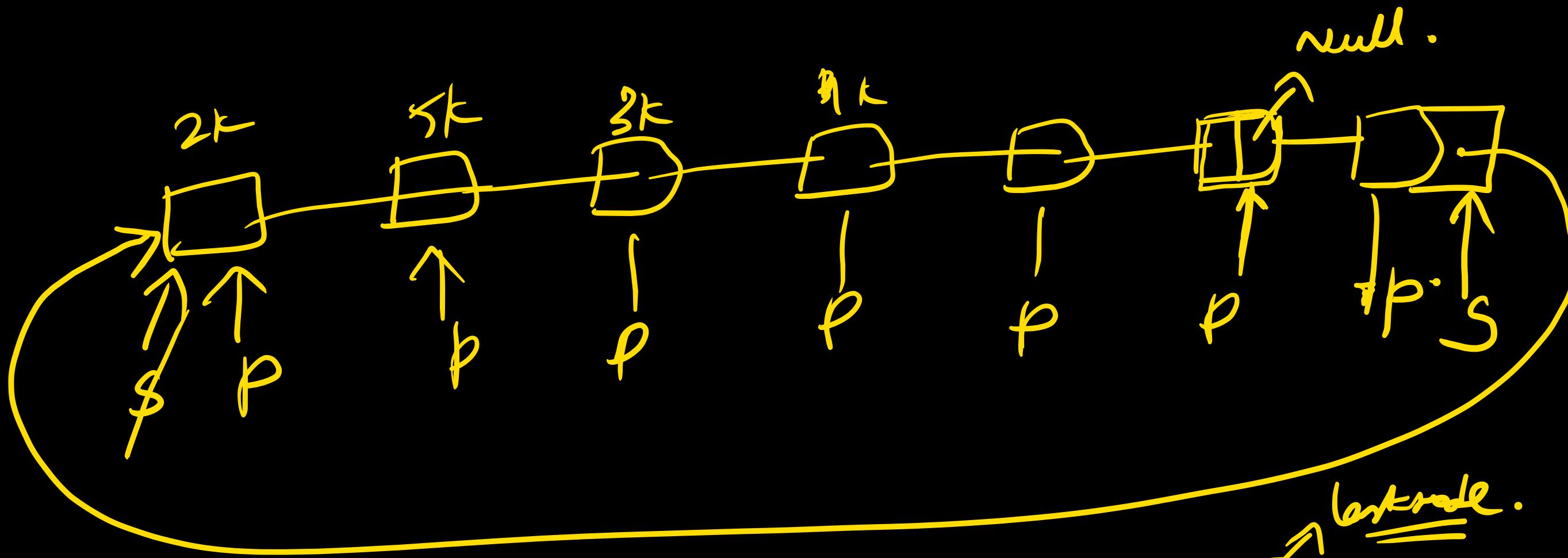
$s_1 = s$
 while ($p \rightarrow \text{next} \neq \text{null}$)
 {
 $p = p \rightarrow \text{next};$
 $s_1 = s_1 \rightarrow \text{next};$
 }
 return $s_1;$

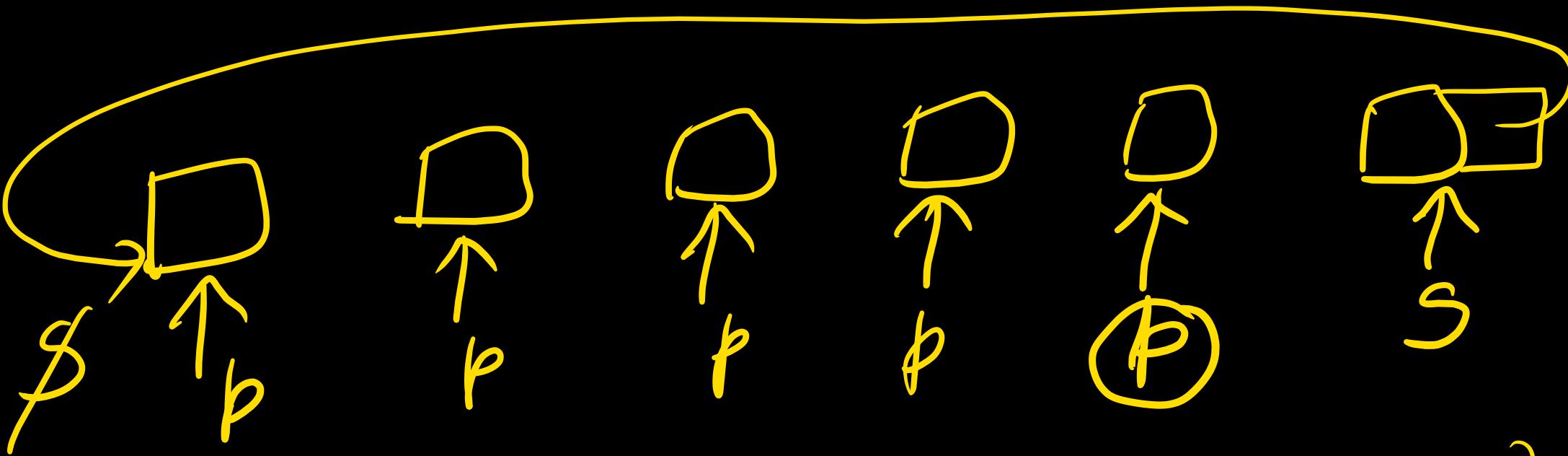
y



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







Single pointer $\leftarrow \left\{ \begin{array}{l} p \rightarrow \text{next} \rightarrow \text{next} = s \\ s = p \rightarrow \text{next} \\ p \rightarrow \text{next} = \text{null} \end{array} \right\}$

if ($s == \text{null}$) return null \rightarrow 0 nodes

if ($s \rightarrow \text{next} == \text{null}$) return (s) \rightarrow 1 node

$p = q = s ;$

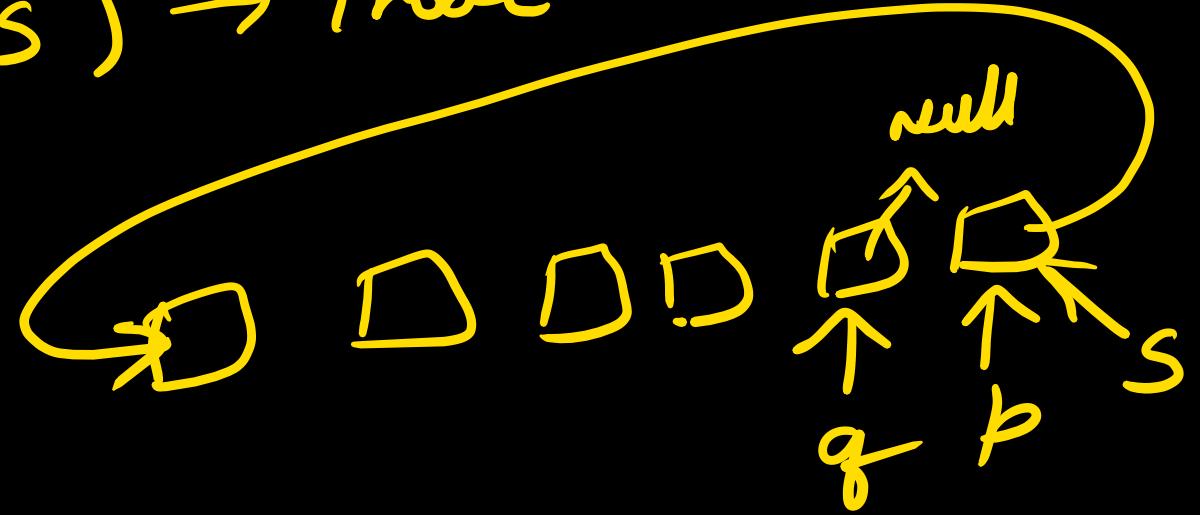
while ($p \rightarrow \text{next} != \text{null}$)

{ $q = p ;$

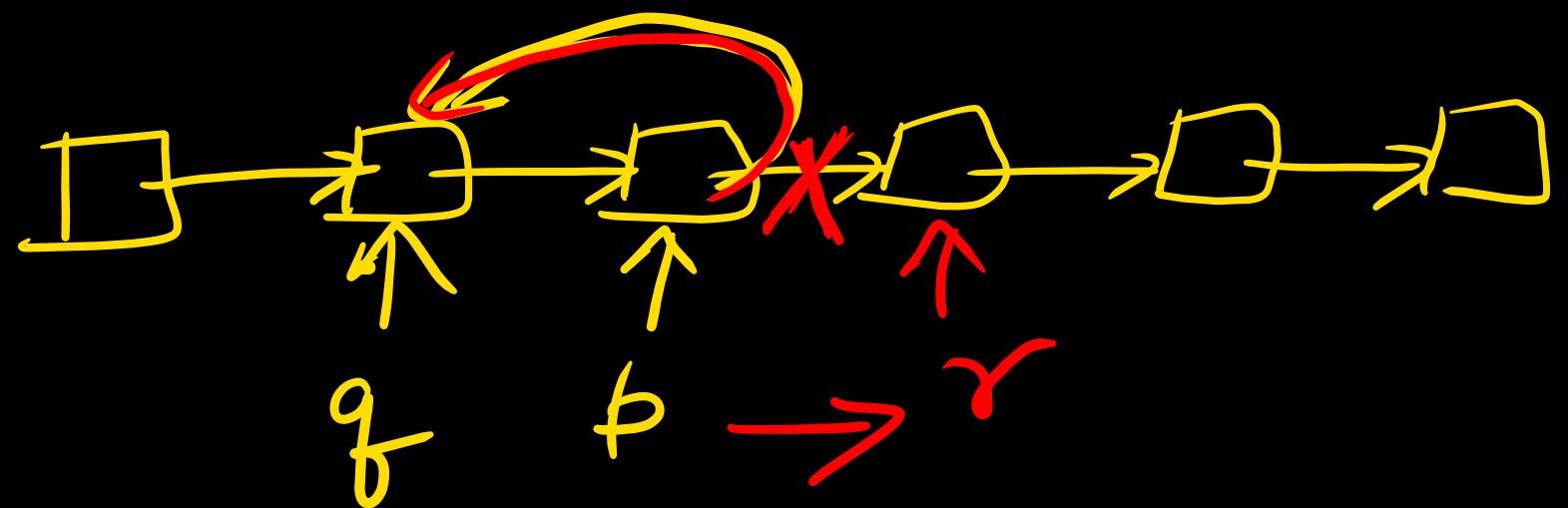
$p = p \rightarrow \text{next}$

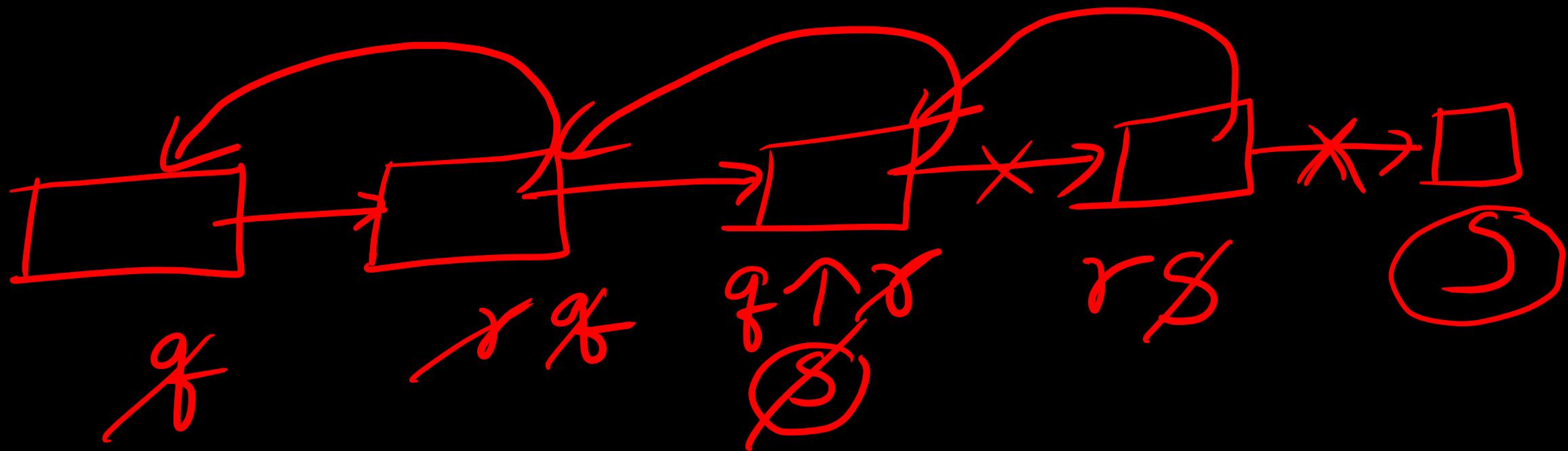
}

$q \rightarrow \text{next} = \text{null} ; p \rightarrow \text{next} = s ; s = p ; \text{return}(s)$



write an algo to reverse a given SLL





$q = r = \text{null}$

while ($S \neq \underline{\underline{\text{null}}})$

{ $q = r; \rightarrow q$ is following r

$r = S; \rightarrow S$ is following S

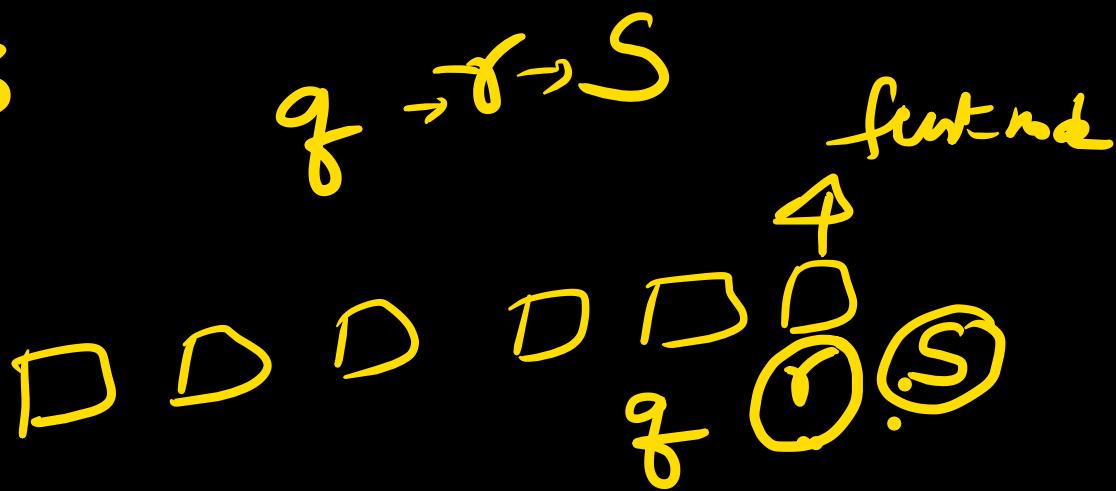
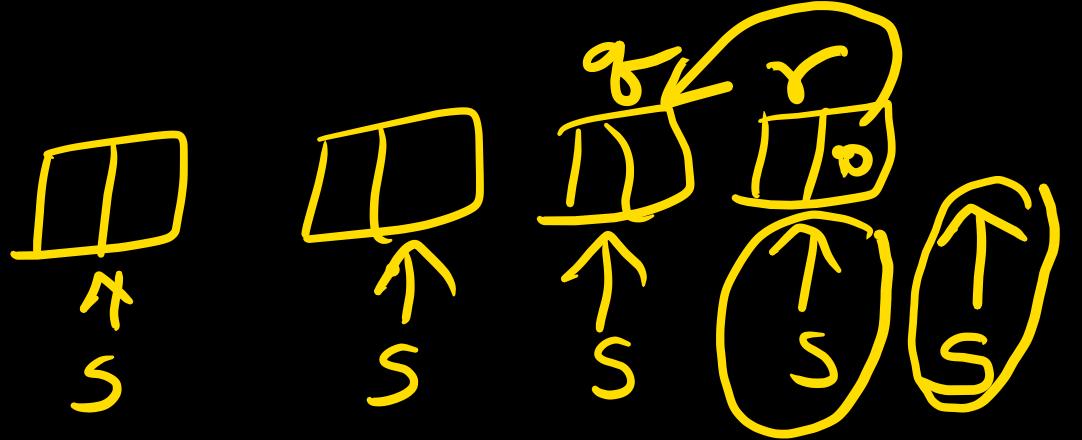
$S = S \rightarrow \text{next};$

$r \rightarrow \text{next} = q;$

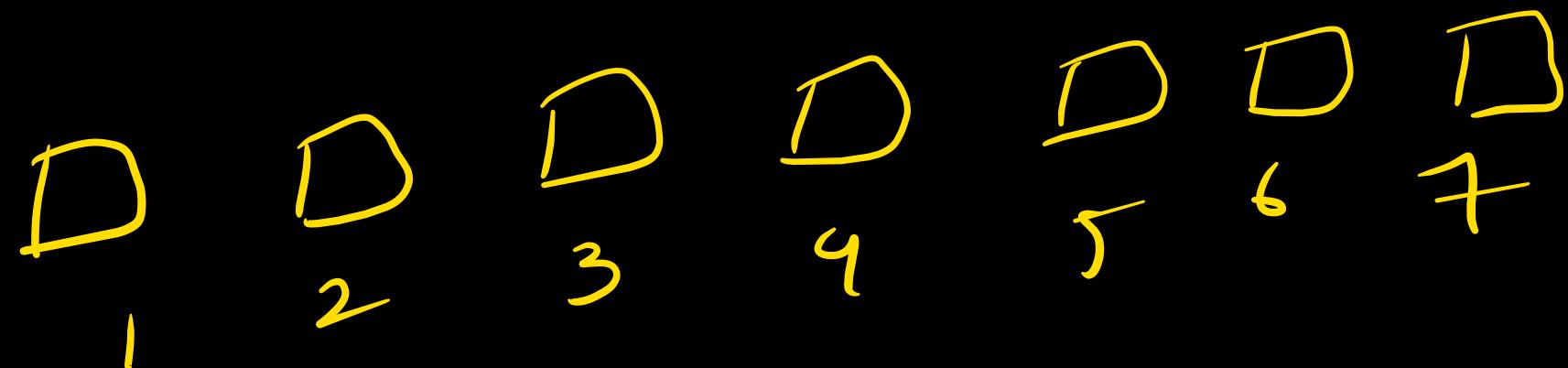
}

$S = \emptyset;$

$\text{extend}(S);$



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

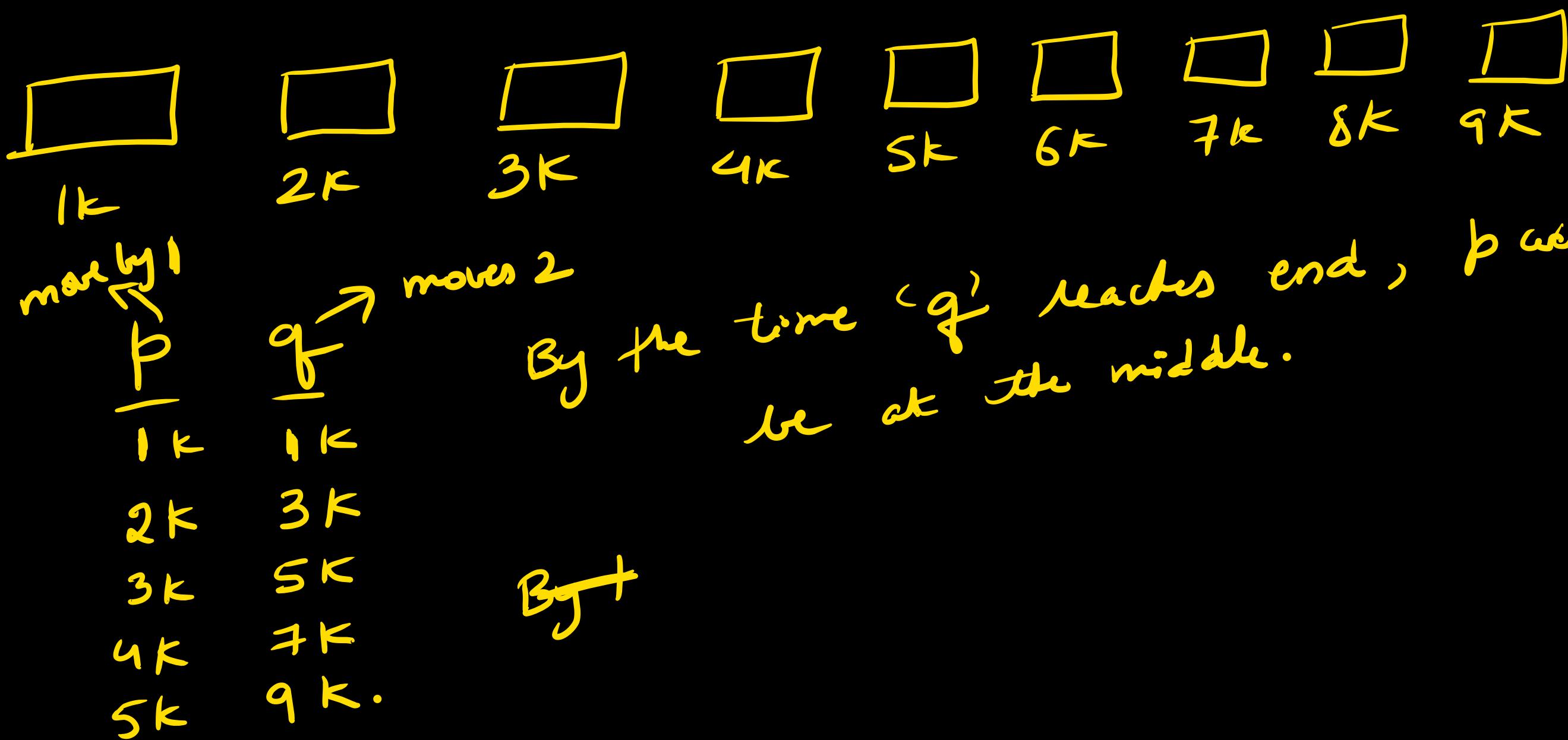


⑦ → number.
then 4 is middle

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

assume that LL Contains odd no of nodes

\equiv



$p = q = s$

while ($q \neq \text{null}$ && $q \rightarrow \text{next} \neq \text{null}$ && $q \rightarrow \text{next} \rightarrow \text{next} \neq \text{null}$)

{ $p = p \rightarrow \text{next};$

$q = q \rightarrow \text{next} \rightarrow \text{next};$

}

return (p);

With an **algo** to insert a node at the end of the SLL

$p \leftarrow$ Create a node with data x and give its pointer

$p \rightarrow \text{data} = x$

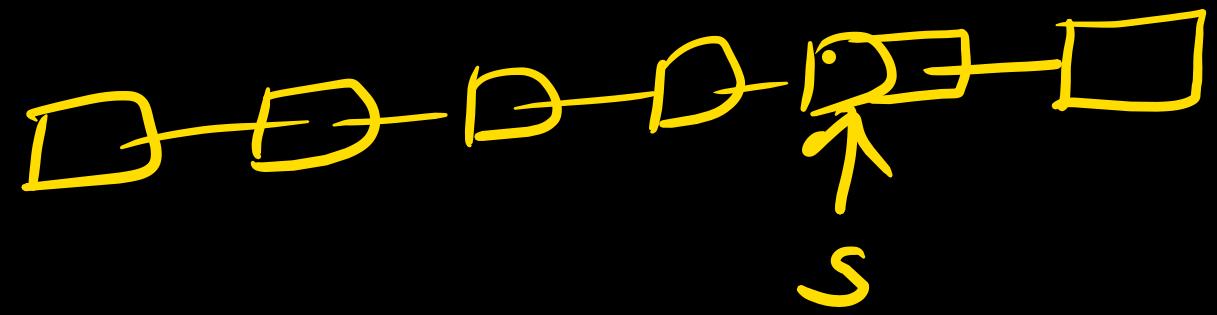
$p \rightarrow \text{next} = \text{null}$
if ($s = \text{null}$) $s = p$ return ' s ';

while ($s_1 \rightarrow \text{next} \neq \text{null}$)

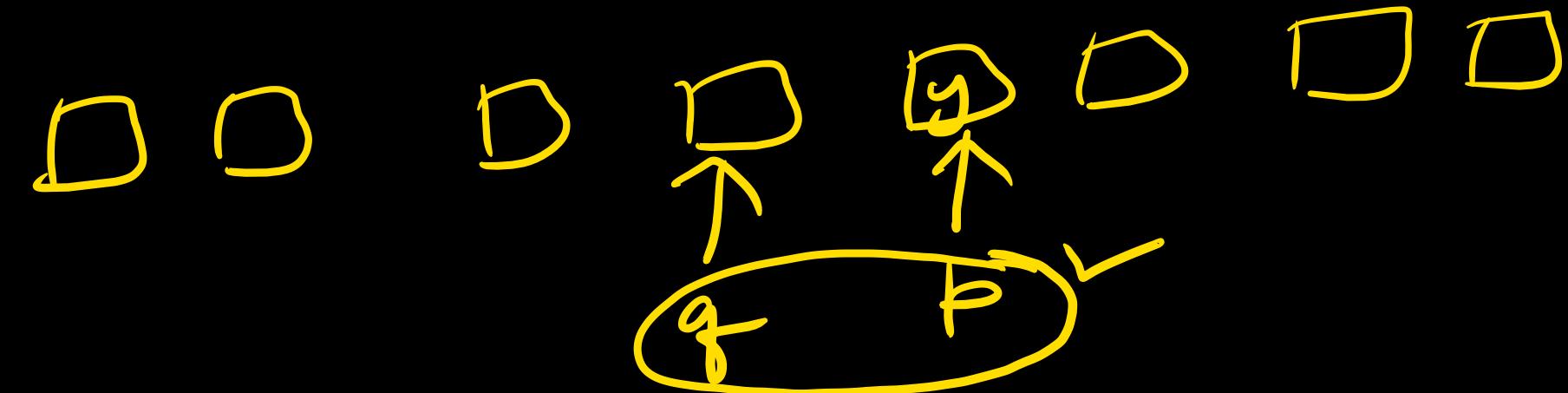
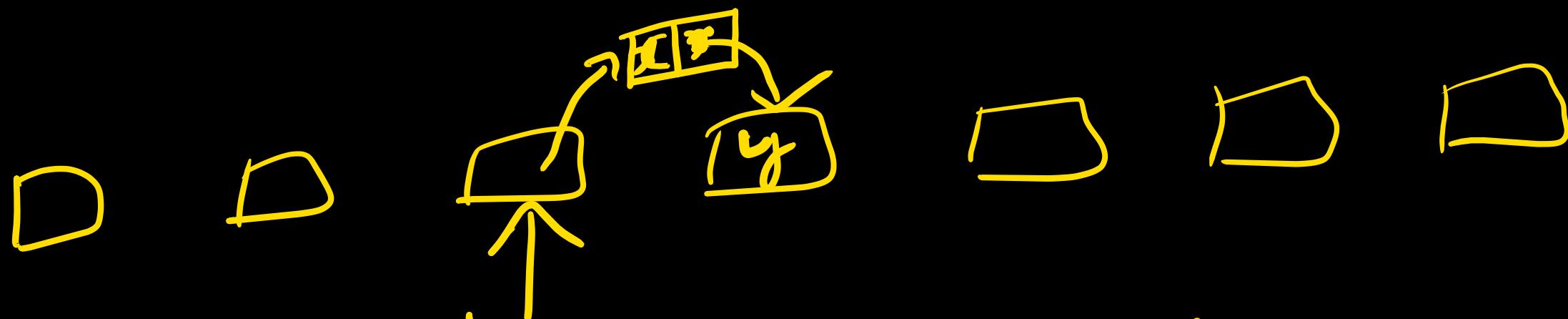
$s_1 = s_1 \rightarrow \text{next};$

$s_1 \rightarrow \text{next} = p;$

return ' s ' \rightarrow returning the first node on stack.



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



if ($S = \text{null}$) return 'S'
struct node * $p =$ ^(struct node *)
 $\text{malloc}(\text{size of (struct node)})$

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

$s_1 = S;$

while ($\underbrace{s_1 \rightarrow \text{data} \neq y}_{\text{or}} \text{ or } \underbrace{s \rightarrow \text{next} \neq \text{null}}$)

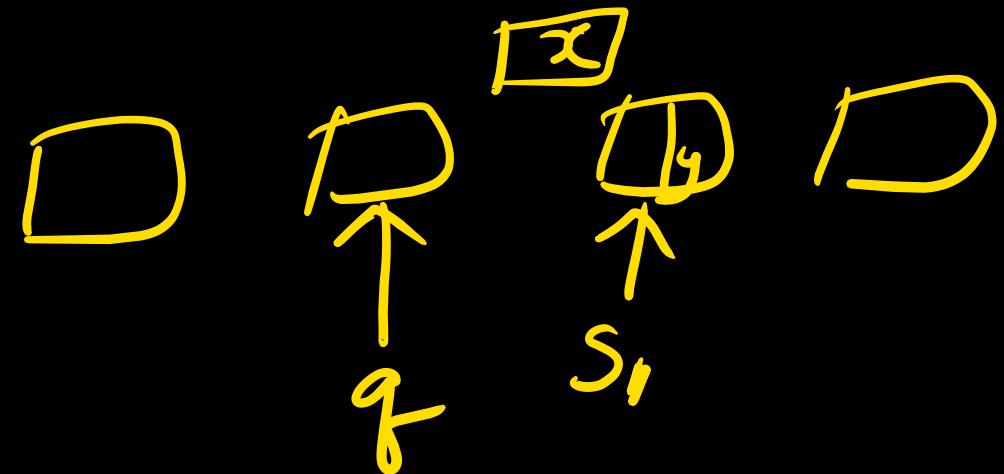
{ $q = s_1$
 $s_1 = s_1 \rightarrow \text{next}$
 y

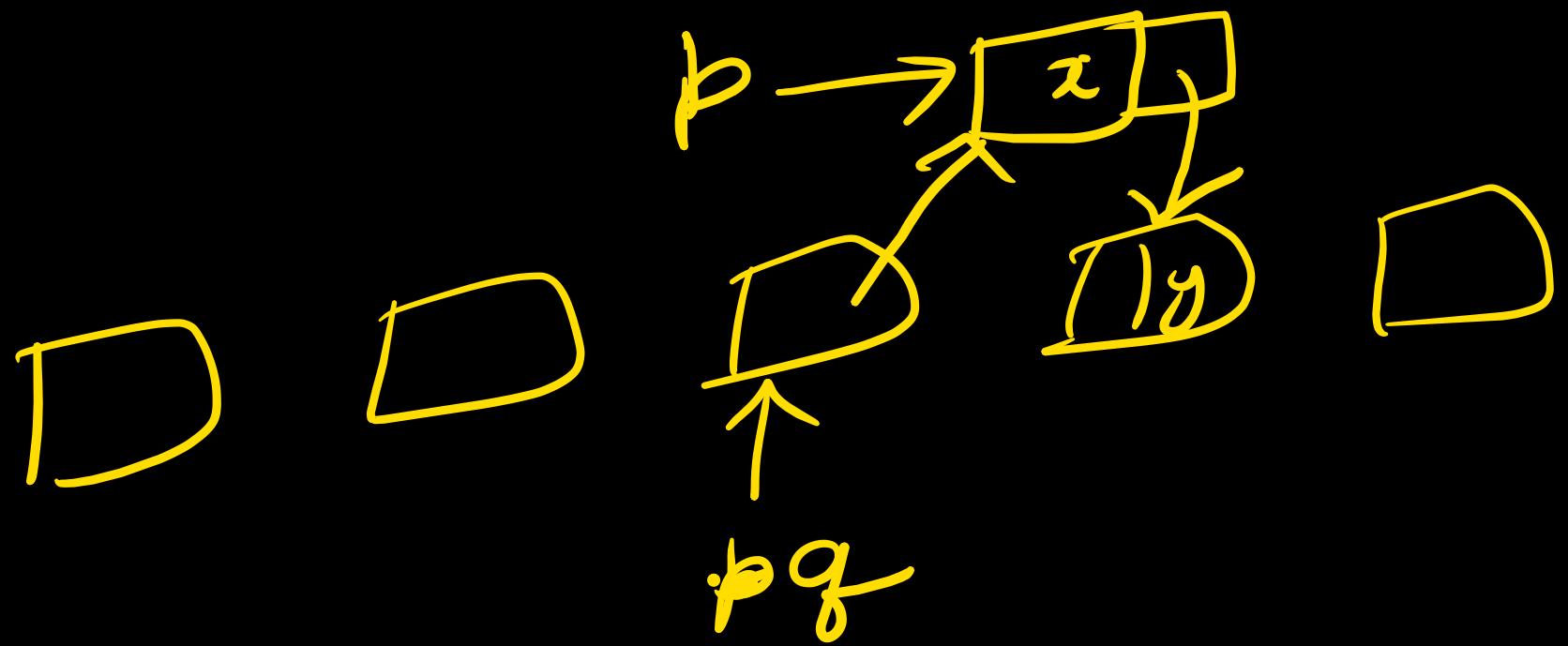


```

if (s->data == y)
{
    q->next = p
    p->next = s
    return(s);
}
printf("no y is present")
return 'S'

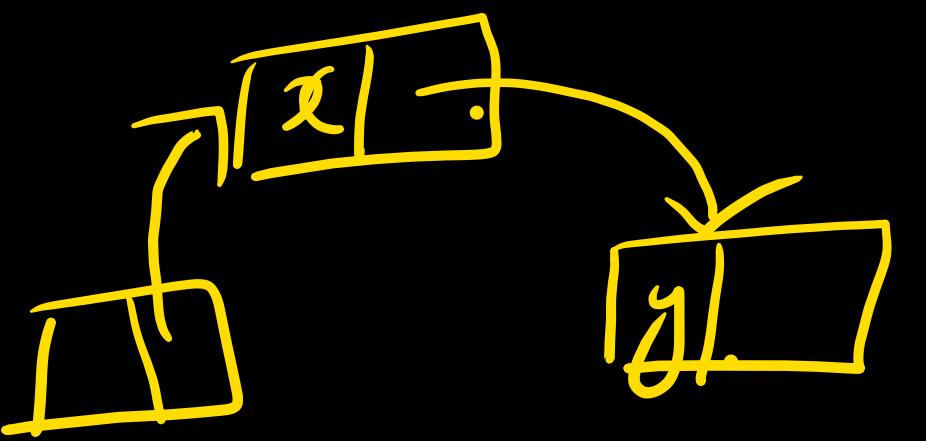
```



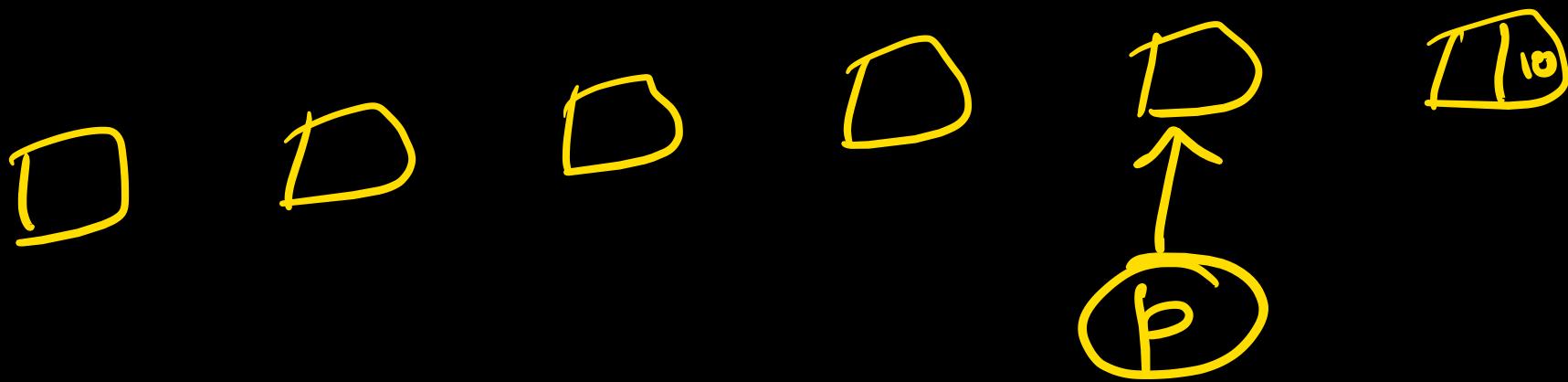


$\underbrace{\text{while}(\text{p} \rightarrow \text{next} \rightarrow \text{data} != y)$

$\text{p} \rightarrow \text{next} = q \rightarrow \text{next}$
 $q \rightarrow \text{next} = p$



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



while($p \rightarrow \text{next} \rightarrow \text{next} \neq \text{NULL}$)

free up the
space $\leftarrow \text{free}(p \rightarrow \text{next})$.
 $p \rightarrow \text{next} = \text{NULL};$

if ($S == \text{null}$) return S ;

~~$S_1 = S$~~

$S_1 = S$

while ($S_1 \rightarrow \text{next} != \text{null}$)

{ $p = S_1$

$S_1 = S_1 \rightarrow \text{next}$

}

$p \rightarrow \text{next} = \text{null};$

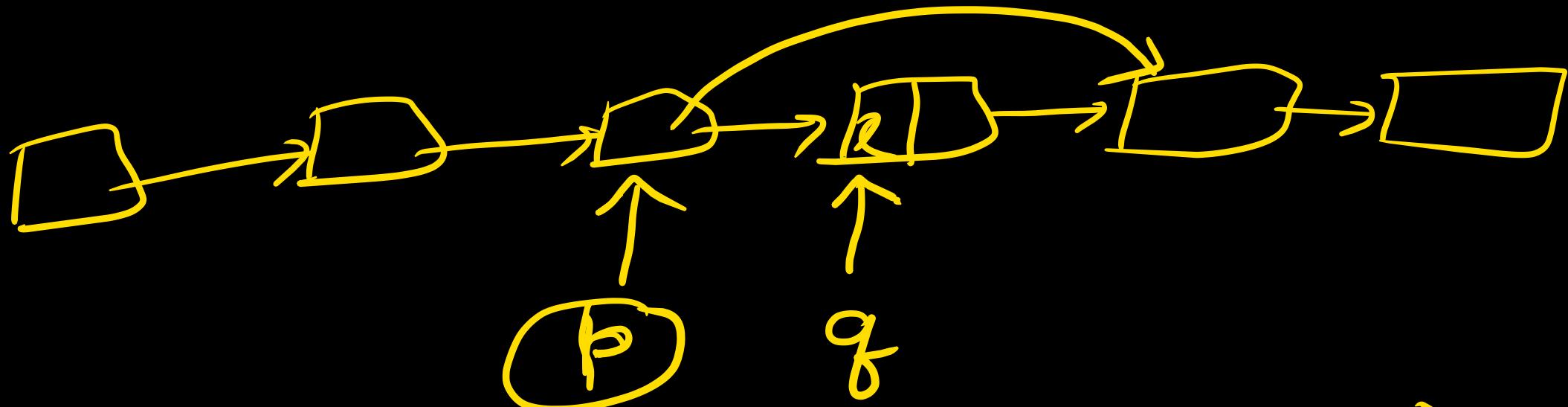
free ($\underline{\underline{S_1}}$)

return (S);



free means deallocate the
memory allocated.

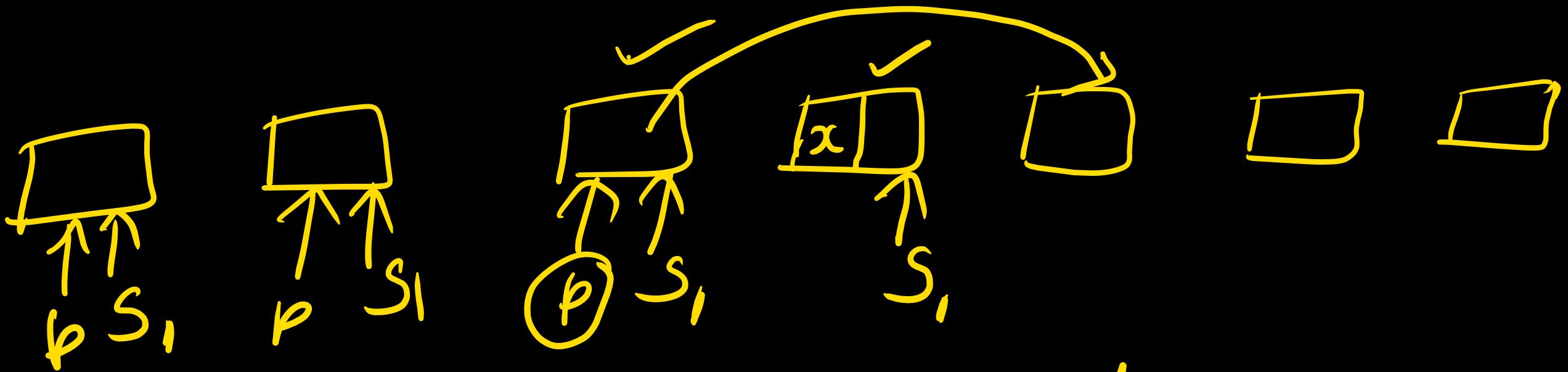
Write an algo to delete a node with data x .



while ($p \rightarrow \text{next} \rightarrow \text{data} \neq x$)

$q = p \rightarrow \text{next};$

$p \rightarrow \text{next} = q \rightarrow \text{next}.$
free(q).



$b \rightarrow \text{new} < x \rightarrow \text{new}$
 $\text{free}(S_1).$