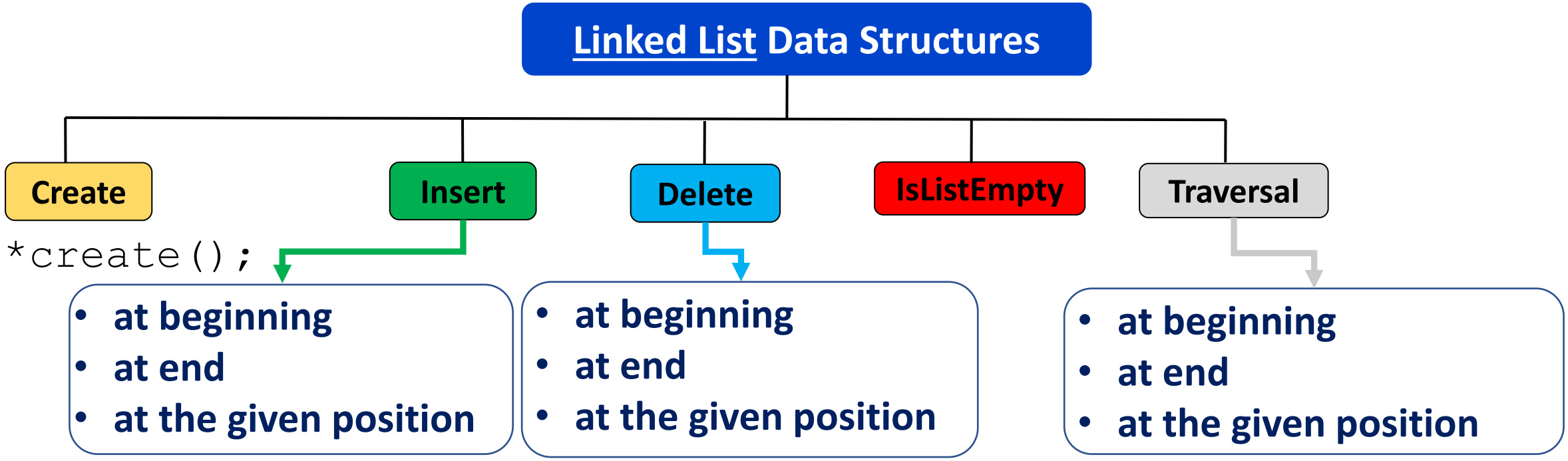
CS2x1:Data Structures and Algorithms

Koteswararao Kondepu

k.kondepu@iitdh.ac.in

Recap Linked List Operations

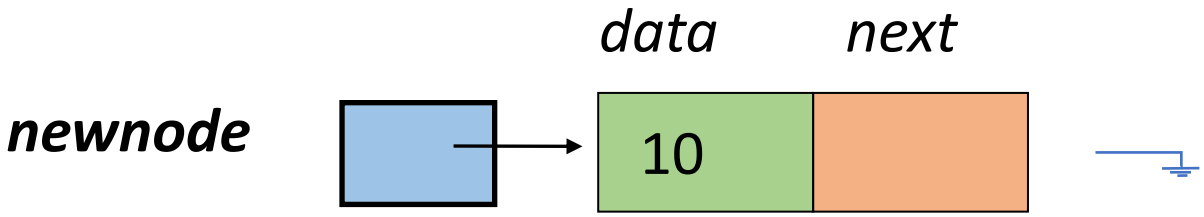



```
insert_begin();
insert_end();
insert_pos();
```

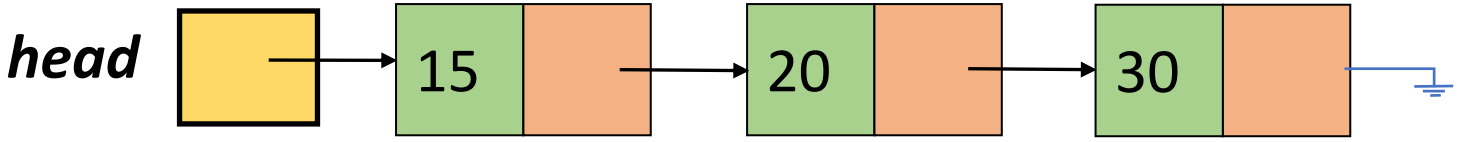
```
delete_begin();
delete_end();
delete_pos();
```

```
display();
get_begin();
get_end();
get_pos();
```

Exercise: Linked List (1)




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



Select the correct option to insert a node at the beginning of the linked list?

head = newnode
newnode → next = head

(A)

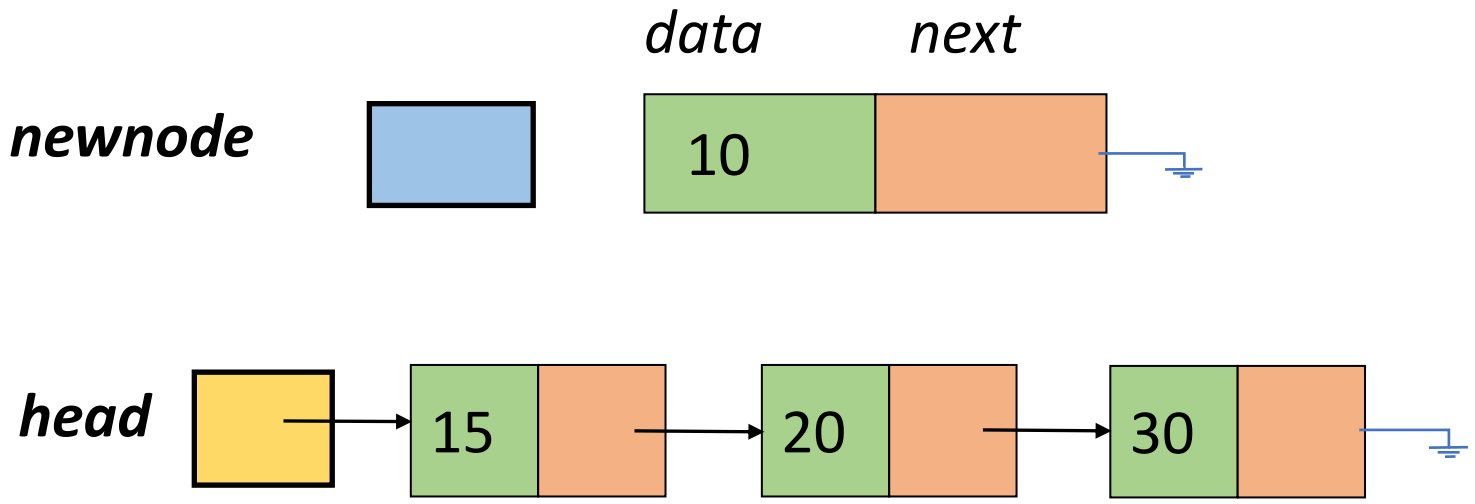
head → next = newnode
head = newnode

(B)

newnode → next = head
head = newnode

(C)

Exercise: Linked List (2)




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

Select the correct options to insert a node at the end of the linked list? Note:assume That the tail pointer pointing to the end of the given linked list

newnode → next = tail
tail = newnode

(A)

tail → next = newnode
tail = newnode

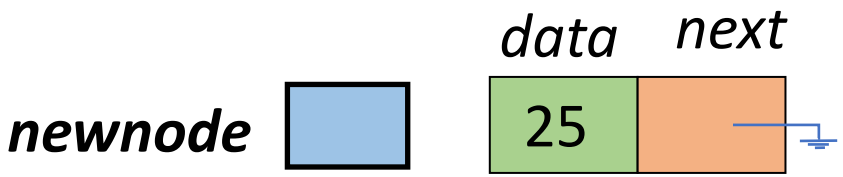
(B)

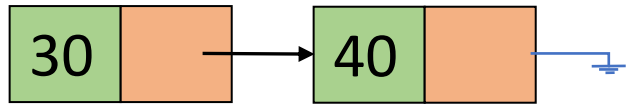
tail = newnode
tail → next = newnode

(C)

Linked List: Insert at the given position

Steps:

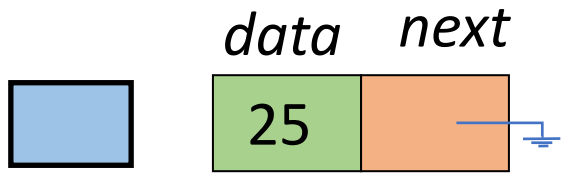




Linked List: Insert at the given position

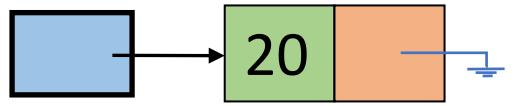
Steps:

(i) Creating a node with data




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

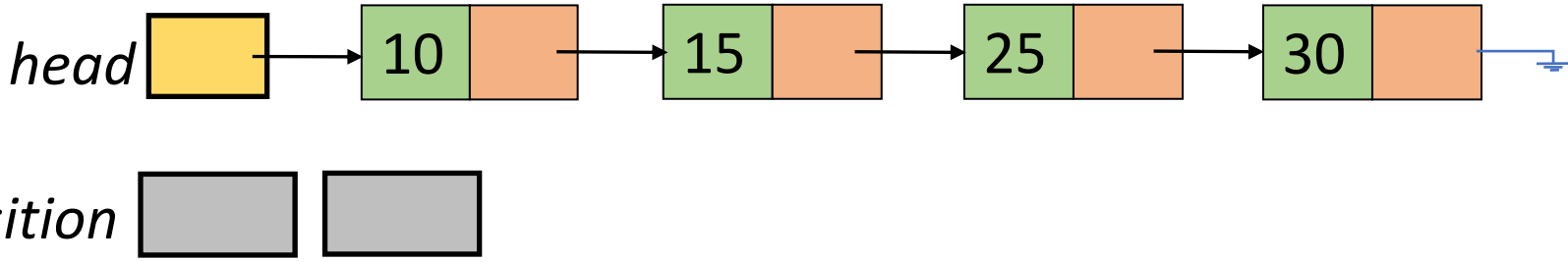
```
struct node *newnode = malloc(sizeof(struct node));
newnode → data = 10;
newnode → next = NULL;
```



(ii) Adding a node to at the given position

a) Traversal the list till the *position – 1*

```
struct node *position
position = head
i = 0
while (i<pos-1)
    position = position → next
    i++;
```



b) Point *newnode* → next to the position-node → next

```
newnode → next = position → next
```

c) Point position-node → next to the *newnode*

```
position → next = newnode
```

Exercise: Linked List (3)

Which of the following points is/are true about Linked List data structure when it is compared with array?

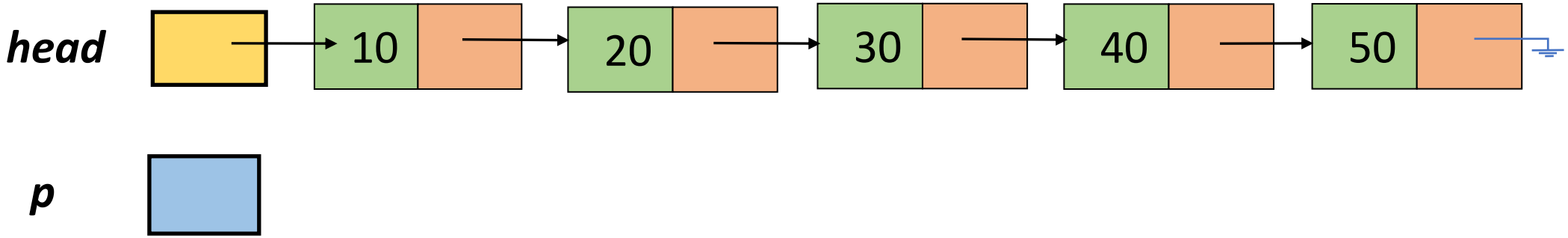
a) It is easy to insert and delete elements in Linked List

b) Random access is not allowed in a typical implementation of Linked Lists

c) The size of array has to be pre-decided, linked lists can change their size any time

d) All of the above

Exercise: Linked List (4)



*struct node *p;*

p=head;

(i) p=head → next → next ;

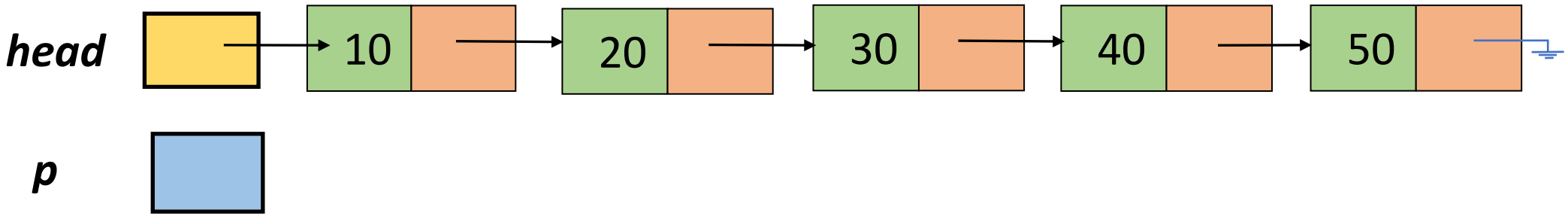
(ii) p → next → next = head;

(iii) print("%d", p → next → next → data);

What is the output if the above statements are executed in the same sequence

a) 10 b) 20 c) 40 d)50

Exercise: Linked List (5)



*struct node *p;*

p=head;

(i) p=head → next → next ;

(ii) p → next → next = head;

(iii) p = p → next

(iv) print("%d", p → next → next → data);

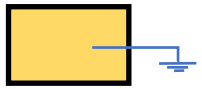
What is the output if the above statements are executed in the same sequence

a) 10 b) 20 c) 30 d)50

Linked List: delete at the beginning or delete at the head

Steps:

(i) Deleting a node from the empty list

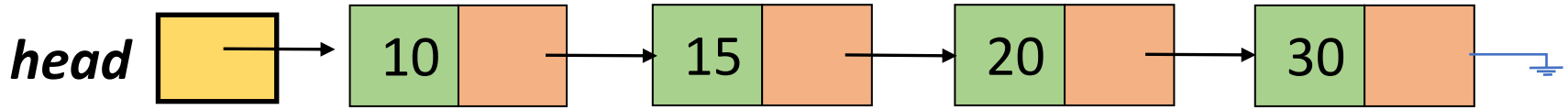


If (head == NULL)

printf("List is Empty\n")

(ii) Deleting a node at the beginning of linked list

*struct node *delete*

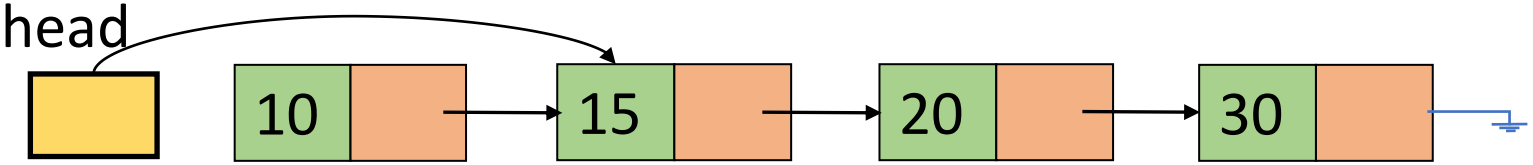


a) Point the head node to the delete pointer

delete = head

b) point head to next node of the head

head = head → next



c) physically deleting the node from the list (i.e., return the allocated node memory to head)

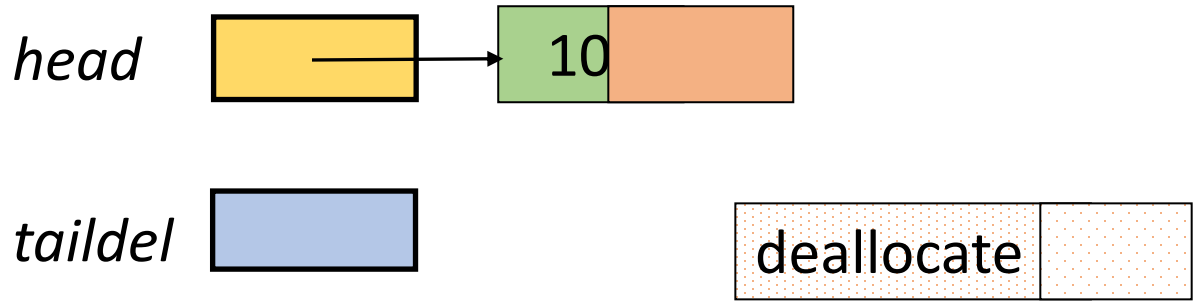
free (delete)

Linked List: delete at the end or delete at the tail

Steps:

(i) Deleting a node from the linked list with one node

```
struct node *taildel
taildel = head
If (head → next == NULL)
    head = NULL
    free (taildel)
```



(ii) Deleting a node at the end of linked list

Linked List: delete at the end or delete at the tail (1)

Steps:

(ii) Deleting a node at the end of linked list

a) Point the head to the delete pointer

taildel = head

b) Traversal to the tail node

*struct node *tailprevnode*

while (taildel → next != NULL)

tailprevnode=taildel

taildel = taildel → next

c) point tail previous node to NULL and free tail node

tailprevnode → next = NULL

free (taildel)

Linked List: delete at the end or delete at the tail (2)

Steps:

(ii) Deleting a node at the end of linked list

a) Point the head to the delete pointer

taildel = head

b) Traversal to the tail node

while (taildel → next → next != NULL)

taildel = taildel → next

c) point taildel next node to NULL and free tail node

taildel → next = NULL

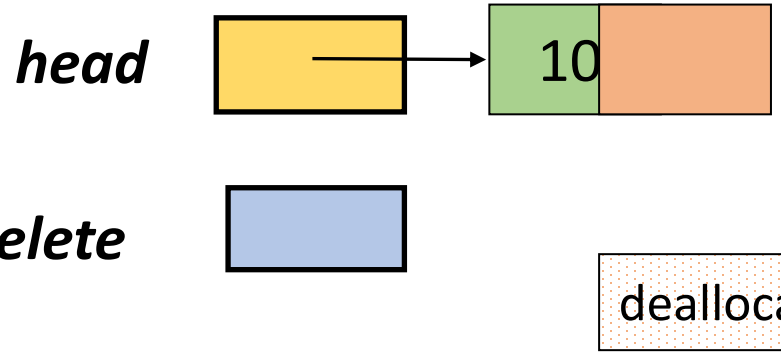
free (taildel → next → next)

Linked List: delete at the given position

Steps:

(i) Deleting a node if the linked list contains one node

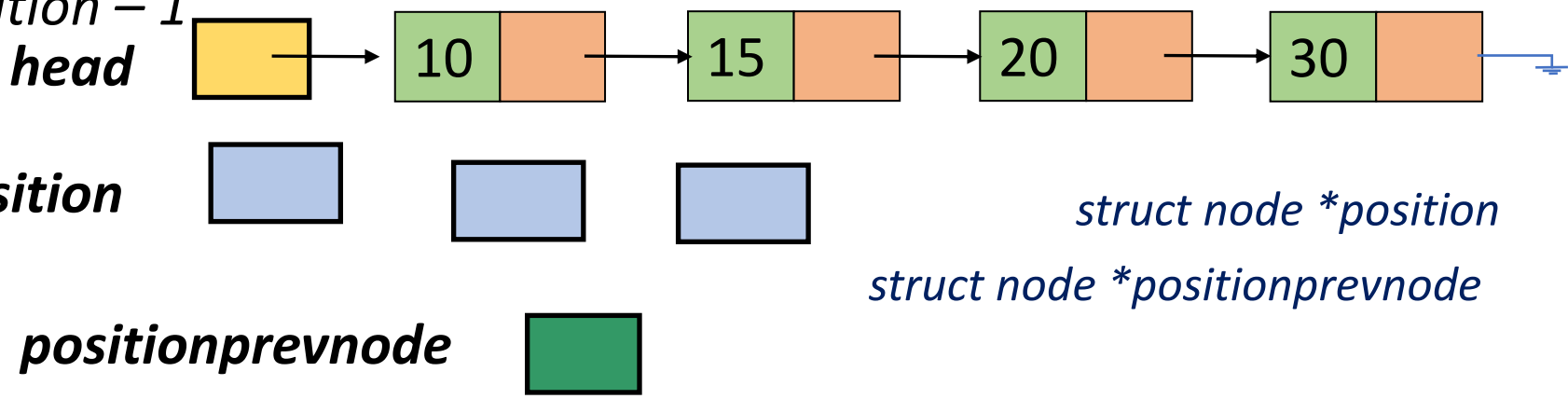
```
struct node *position
position = head
If (pos==0)
    head = head → next
    free (position)
```



(ii) Deleting a node at the given position

a) Traversal the list till the *position – 1*

```
position = head
i = 0
while (i<pos)
    positionprevnode = position
    position = position → next
    i++;
```

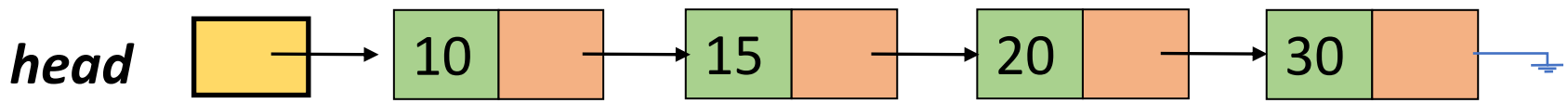


b) Change the position previous node next to position next node

```
positionprevnode → next = position → next
free (position)
```

Limitations: Singly Linked List

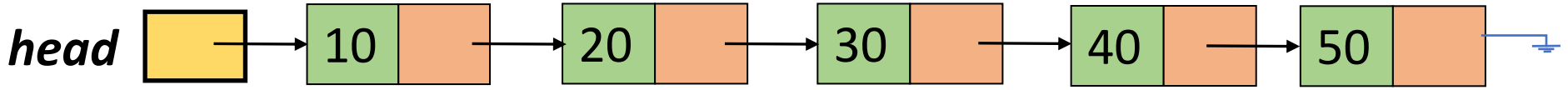
- Limitations:
  a) Nodes are stored in-contiguously, the time required to access individual elements greatly increased within the list
  b) Nodes in a linked-list are accessed in order from beginning, thus the linked lists are inherently sequential access
  c) Difficulties arises in linked-list when it comes to reverse traversing.
  d) It requires more space as pointers are also stored in Struct along with data field



Exercise: Singly Linked List (6)

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

What is the output of the function **find ()** for the following linked list?



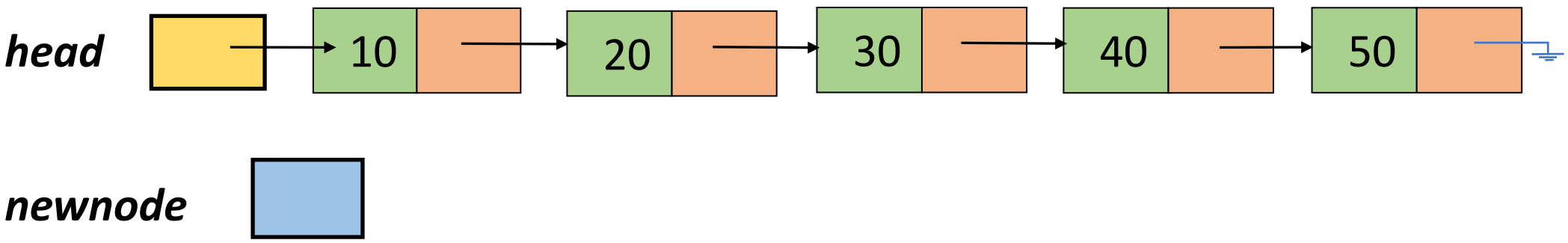

```
node *find (node *head)
{
node *P1=head, *P2=head;
while (P2)
        {
                P1 = P1 → next;
                P2 = (P2 → next!=NULL)? P2 → next → next : NULL;
        }
printf("%d", P1 → data);
}
```

(a) 20 *(b) 30* *(c) 40* *(d) 50*

Exercise: Singly Linked List (7)

Q: Fill the following table with the number of pointer operations need to be changed for each sinlge linked list operation ?

Operations	begin	end	Middle (pos)
Insert			
delete			



thank you!

email:
k.kondepu@iitdh.ac.in

NEXT Class: 03/05/2023