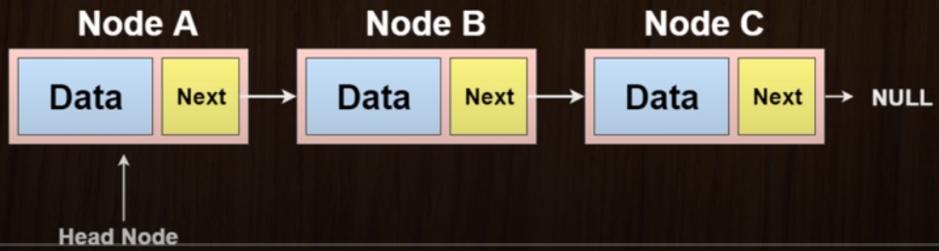


What is a linked list?

What is Linked List Data Structure ?

Definition : A *linked list* is a *linear data structure*, in which the elements are **not** stored at **contiguous** memory locations. The elements in a linked list are **linked** using **pointers**(entity that point to the next element)

In simple words, a linked list consists of **nodes** where each **node** contains a **data field** and a **reference(link)** to the next node in the list.



Linked List vs Arrays

Advantages of Linked List over Arrays -

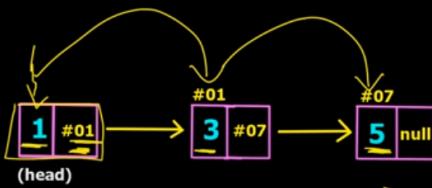
- 1) Dynamic size
- 2) Ease of insertion/deletion

Disadvantages of Linked List over Arrays -

- 1) Random access is not allowed. We have to access elements sequentially starting from the first node.
- 2) Extra memory space for a pointer is required with each element of the list.
- 3) Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

#01	#02	#03	#04	#05	#06
1 0	3 1	5 2	3	4	5

int arr[6] = {1,3,5} arr[2] = 5



Operations of Linked List - $N \rightarrow next$

1. Traversing a linked list.	2. Append a new node (to the end) of a list	3. Prepend a new node (to start) of list

4. Inserting a new node to a specific position in the list	5. Deleting a node from the list	6. Updating a node in the list

Types of Linked List -

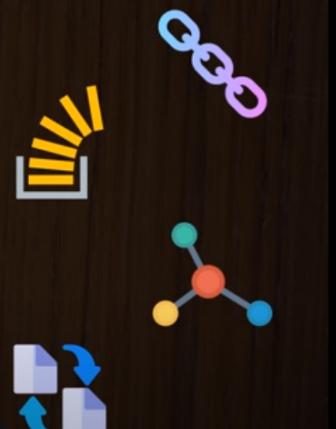
D = Data P = Previous N = Next

1) Singly Linked List	2) Doubly Linked List	3) Circular Linked List

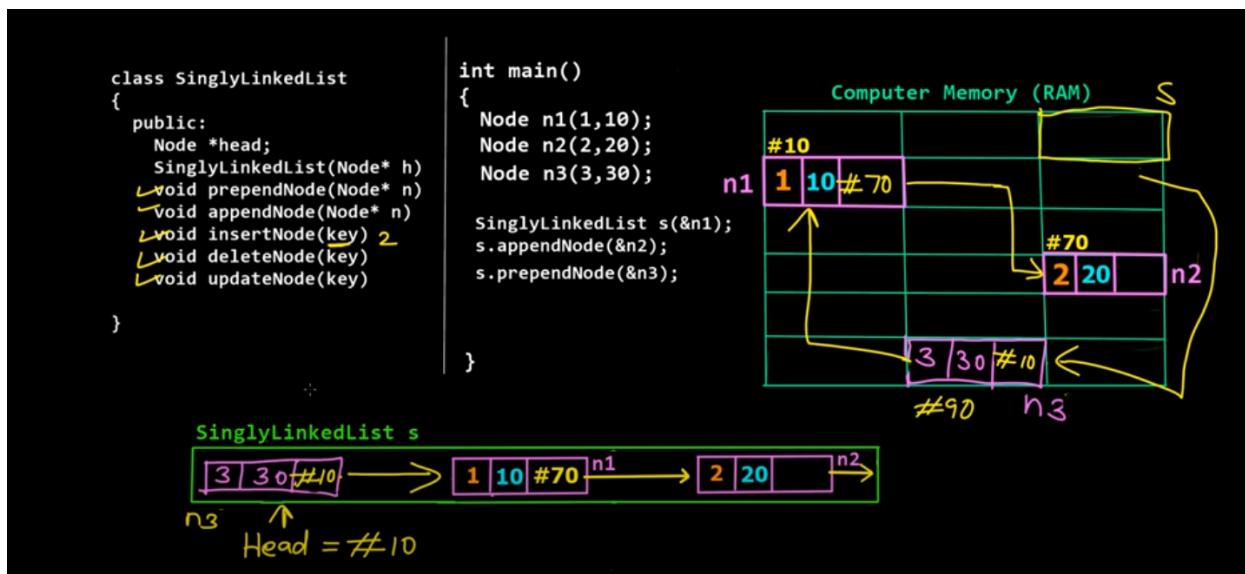
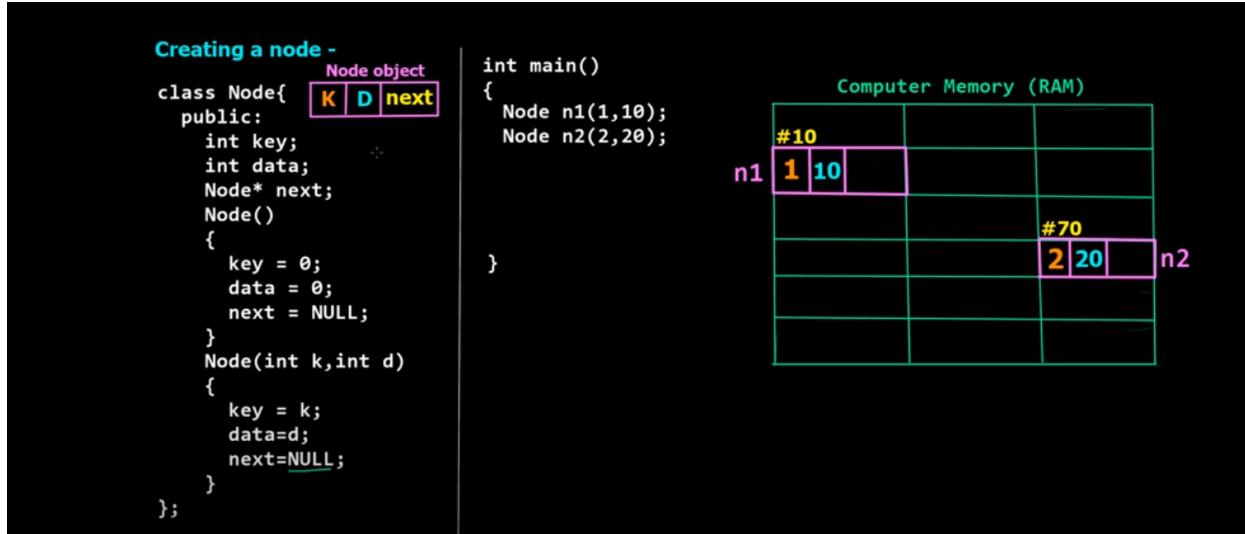
Some Applications of Linked List Data Structure

Some Applications of Linked List are as follows –

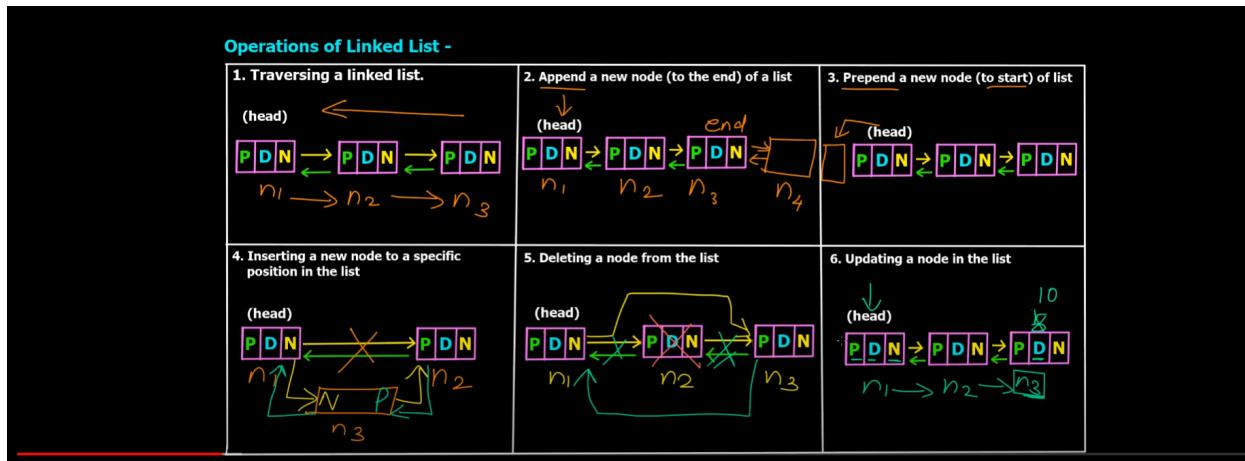
- Linked Lists can be used to implement **Stacks**, **Queues**.
- Linked Lists can also be used to implement **Graphs**. (Adjacency list representation of Graph).
- Implementing Hash Tables :- Each Bucket of the hash table can itself be a linked list. (Open chain hashing).
- **Undo** functionality in Photoshop or Word . Linked list of states



Node



Doubly Linked List



Creating a node -

prev	K	D	next
------	---	---	------

Append node algorithm/pseudo code

- Get the new node.
- Check if node exists with same key
- a. if true, abort
- b. if false continue
- If head == NULL then append at start
- else traverse to the end of the list
- Append new node at the end

```

int main()
{
    Node n1(1,10);
    Node n2(2,20);
    Node n3(3,30);
    Node n4(4,40);

    DoublyLinkedList d;
    d.appendNode(&n1);
    d.appendNode(&n2);
    d.prependNode(&n3);
    d.insertNodeAfter(1,&n4);
    d.deleteNode(3);
    d.updateNode(4,55);
}

```

Computer Memory (RAM)

#10		
null	1	10 null
n1		

#70		
null	2	20 null
n2		

#45		
null	3	30 null
n3		

#90		
null	4	40 null
n4		

$\downarrow \text{head}^* \text{ptr} = \#45 / 0$

DoublyLinkedList d n1

Creating a node -

prev	K	D	next
------	---	---	------

Prepend node algorithm/pseudo code

- Get the new Node.
- Check if node exists with same key
- a. if true, abort
- b. if false continue
- head->previous = new_node (n3)
- new_node->next = head
- head = n (n3)
- = new_node

```

int main()
{
    Node n1(1,10);
    Node n2(2,20);
    Node n3(3,30);
    Node n4(4,40);

    DoublyLinkedList d;
    d.appendNode(&n1);
    d.appendNode(&n2);
    d.prependNode(&n3);
    d.insertNodeAfter(1,&n4);
    d.deleteNode(3);
    d.updateNode(4,55);
}

```

Computer Memory (RAM)

#10		
null	1	10 null
n1		

#70		
null	2	20 null
n2		

#45		
null	3	30 null
n3		

#90		
null	4	40 null
n4		

$\downarrow \text{head}^* \text{ptr} = \#45$

DoublyLinkedList d n1 n2

Creating a node -

prev	K	D	next
------	---	---	------

Insert node node after algorithm/pseudo code

- Get the new Node & the key of the node in the list after which you want to link this new node.
- Check if node exists with same key as new node
- a. if true, abort
- b. if false continue
- Check if node exists with the key entered by user
- a. if false, abort
- b. if true continue
- access node N after which you want to append new node new_N
- a. if N is at the end then
 - N->next = new_N
 - new_N->previous = N;
- b. if N is in between.
 - new_N->next = N->next
 - N->next->previous = new_N
 - new_N->previous = N
 - N->next = new_N

```

int main()
{
    Node n1(1,10);
    Node n2(2,20);
    Node n3(3,30);
    Node n4(4,40);

    DoublyLinkedList d;
    d.appendNode(&n1);
    d.appendNode(&n2);
    d.prependNode(&n3);
    d.insertNodeAfter(1,&n4);
    d.deleteNode(3);
    d.updateNode(4,55);
}

```

Computer Memory (RAM)

#10		
null	1	10 null
n1		

#70		
null	2	20 null
n2		

#45		
null	3	30 null
n3		

#90		
null	4	40 null
n4		

$\downarrow \text{head}^* \text{ptr} = \#45$

DoublyLinkedList d n1 n2

Creating a node -

Node Obj



Delete node by key algorithm -

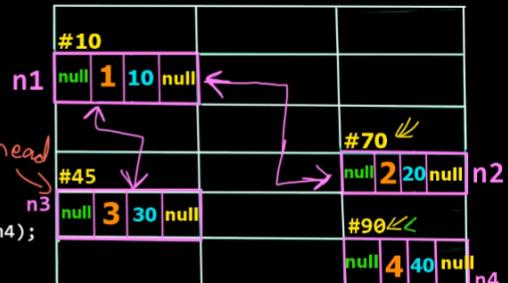
1. Get the key of node you want to delete.
2. Check if node exists with same key
- 2a. if false, abort
- 2b. if true continue ↴
- 3a. if head== NULL then list empty
- 3b. if head!=NULL & key matches head node then ↴
- head=head->next
- 3c. if head!=NULL & key doesn't matches head node then ↴
- Traverse to the node with the key.
- 4a. if node at the end then previousNode->next = null
- 4b. if node in between then ↴
- prevNode->next=nextNode;
- nextNode->previous=prevNode;

int main()

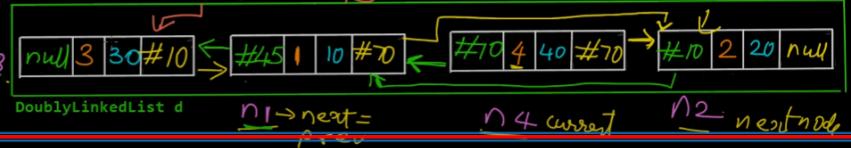
```
{
    Node n1(1,10);
    Node n2(2,20);
    Node n3(3,30);
    Node n4(4,40);

    DoublyLinkedList d
    ↴ d.appendNode(&n1);
    ↴ d.appendNode(&n2);
    ↴ d.prependNode(&n3);
    ↴ d.insertNodeAfter(1,&n4);
    ↴ d.deleteNode(3); 4
    ↴ d.updateNode(4,55);
}
```

Computer Memory (RAM)



head* ptr = #45



Creating a node -

Node Obj



Update node by key algorithm -

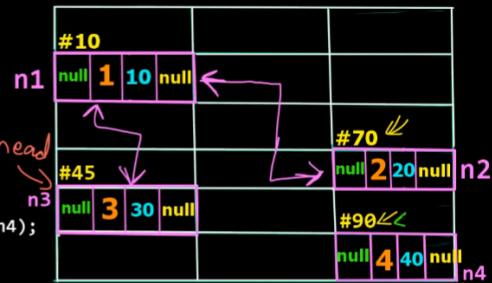
1. Get the key of node you want to update.
2. Check if node exists with same key
- 2a. if false, abort X
- 2b. if true continue ↴
3. traverse to that node.
4. Update the data value.

int main()

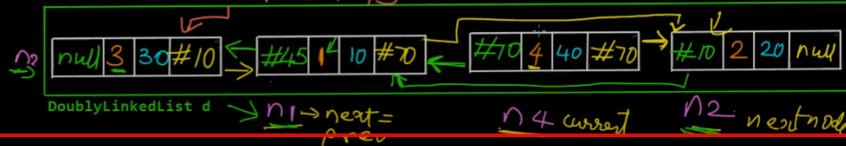
```
{
    Node n1(1,10);
    Node n2(2,20);
    Node n3(3,30);
    Node n4(4,40);

    DoublyLinkedList d
    ↴ d.appendNode(&n1);
    ↴ d.appendNode(&n2);
    ↴ d.prependNode(&n3);
    ↴ d.insertNodeAfter(1,&n4);
    ↴ d.deleteNode(3); 4
    ↴ d.updateNode(4,55);
}
```

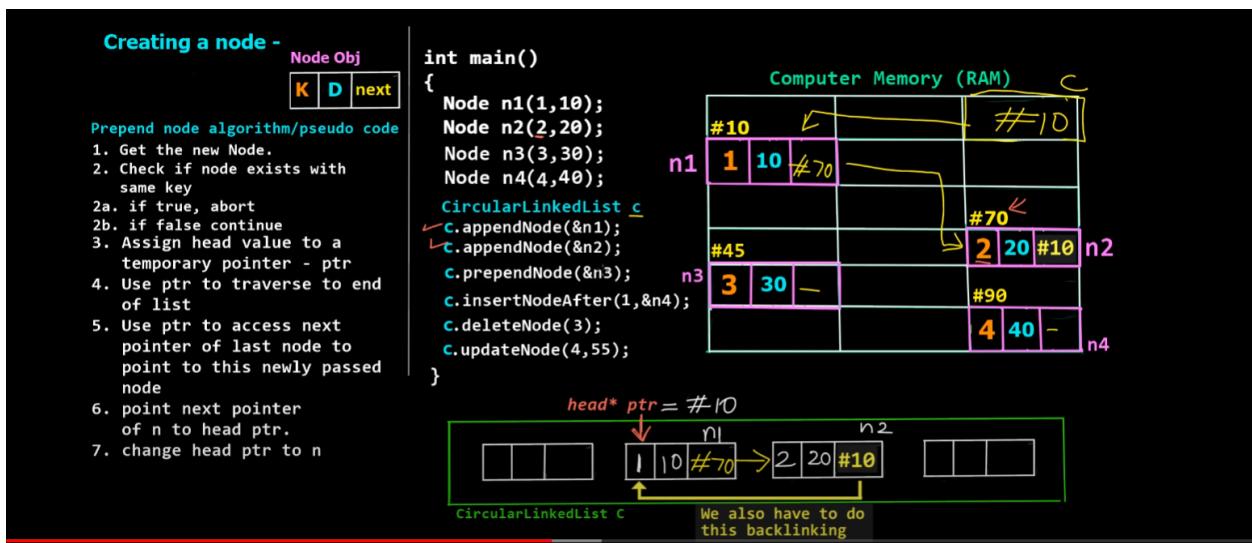
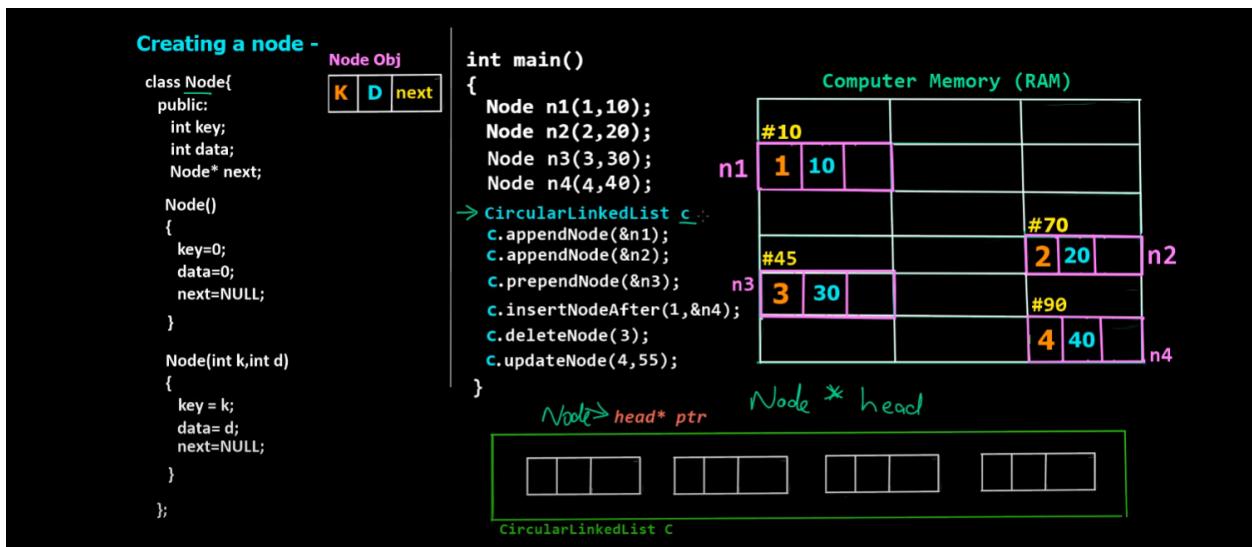
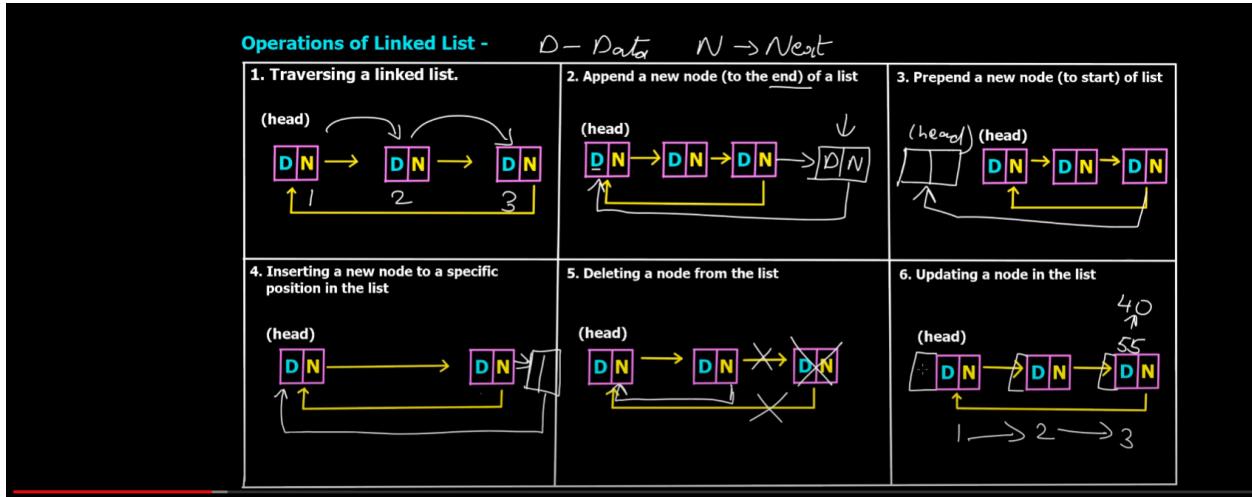
Computer Memory (RAM)



head* ptr = #45



Circular Linked List



Creating a node -

Node Obj

K	D	next
---	---	------

Insert node node after algorithm -

- Get the new Node & the key of the node in the list after which you want to link this new node.
- Check if node exists with same key as new node
- a. if true, abort
- b. if false continue
- Check if node exists with the key entered by user
- a. if false, abort
- b. if true continue
- access node N after which you want to append new node new_N
- If node to be inserted at the end then
new_node->next = head
previous_node->next = new_node
- if node to be inserted in between then
new_node->next = previous_node->next
previous_node->next = new_node

```

int main()
{
    Node n1(1,10);
    Node n2(2,20);
    Node n3(3,30);
    Node n4(4,40);
    CircularLinkedList c
    ↴c.appendNode(&n1);
    ↴c.appendNode(&n2);
    ↴c.prependNode(&n3);
    ↴c.insertNodeAfter(1,&n4);
    c.deleteNode(3);
    c.updateNode(4,55);
}

```

Computer Memory (RAM)

head* ptr = #45

CircularLinkedList C

Creating a node -

Node Obj

K	D	next
---	---	------

Delete node by key algorithm -

- Get the key of node you want to delete.
- Check if node exists with same key
- a. if false, abort
- b. if true continue
- if head== NULL then list empty
- if head!=NULL & key matches head node then
head=>next
last_node->next = head
- if head!=NULL & key doesn't matches head node then
Traverse to the node with the key.(ptr)
- if node at the end then
previousNode->next = head
- if node in between then
prevNode->next=ptr->next

```

int main()
{
    Node n1(1,10);
    Node n2(2,20);
    Node n3(3,30);
    Node n4(4,40);
    CircularLinkedList c
    ↴c.appendNode(&n1);
    ↴c.appendNode(&n2);
    ↴c.prependNode(&n3);
    ↴c.insertNodeAfter(1,&n4);
    ↴c.deleteNode(3);
    c.updateNode(4,55);
}

```

Computer Memory (RAM)

head* ptr = #45

CircularLinkedList C

Creating a node -

Node Obj

K	D	next
---	---	------

Update node by key algorithm -

- Get the key of node you want to update.
- Check if node exists with same key
- a. if false, abort
- b. if true continue
- traverse to that node.
- Update the data value.

```

int main()
{
    Node n1(1,10);
    Node n2(2,20);
    Node n3(3,30);
    Node n4(4,40);
    CircularLinkedList c
    ↴c.appendNode(&n1);
    ↴c.appendNode(&n2);
    ↴c.prependNode(&n3);
    ↴c.insertNodeAfter(1,&n4);
    ↴c.deleteNode(3);
    ↴c.updateNode(4,55);
}

```

Computer Memory (RAM)

head* ptr = #45

CircularLinkedList C