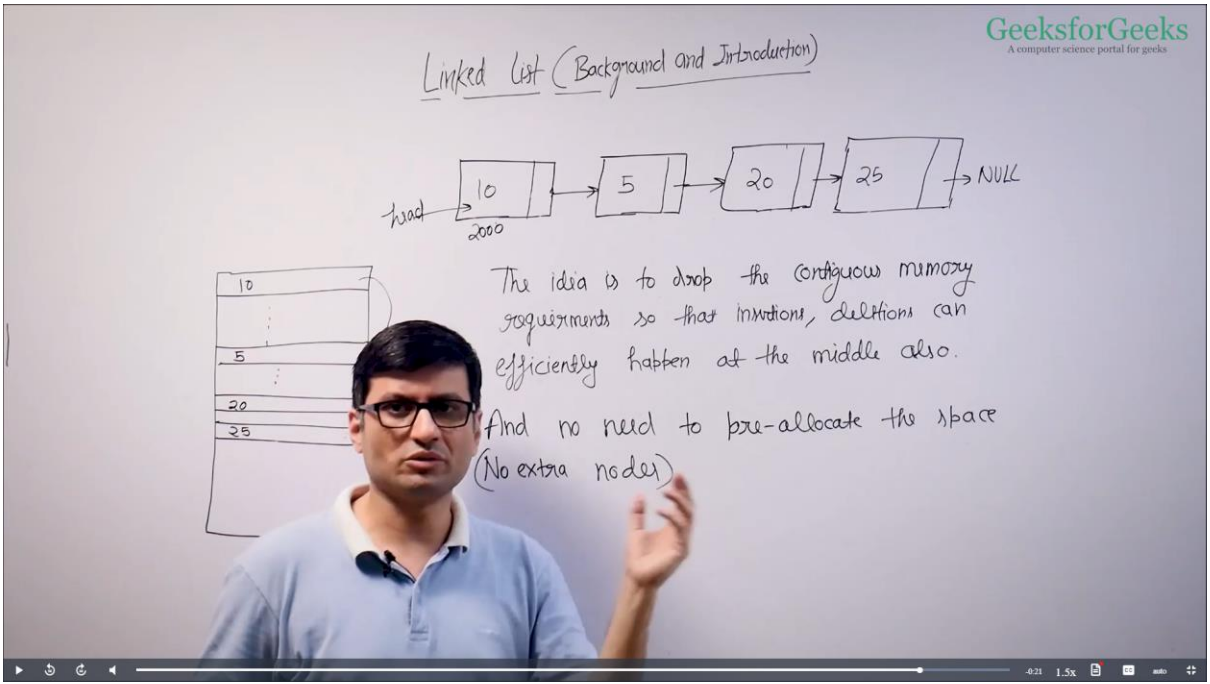


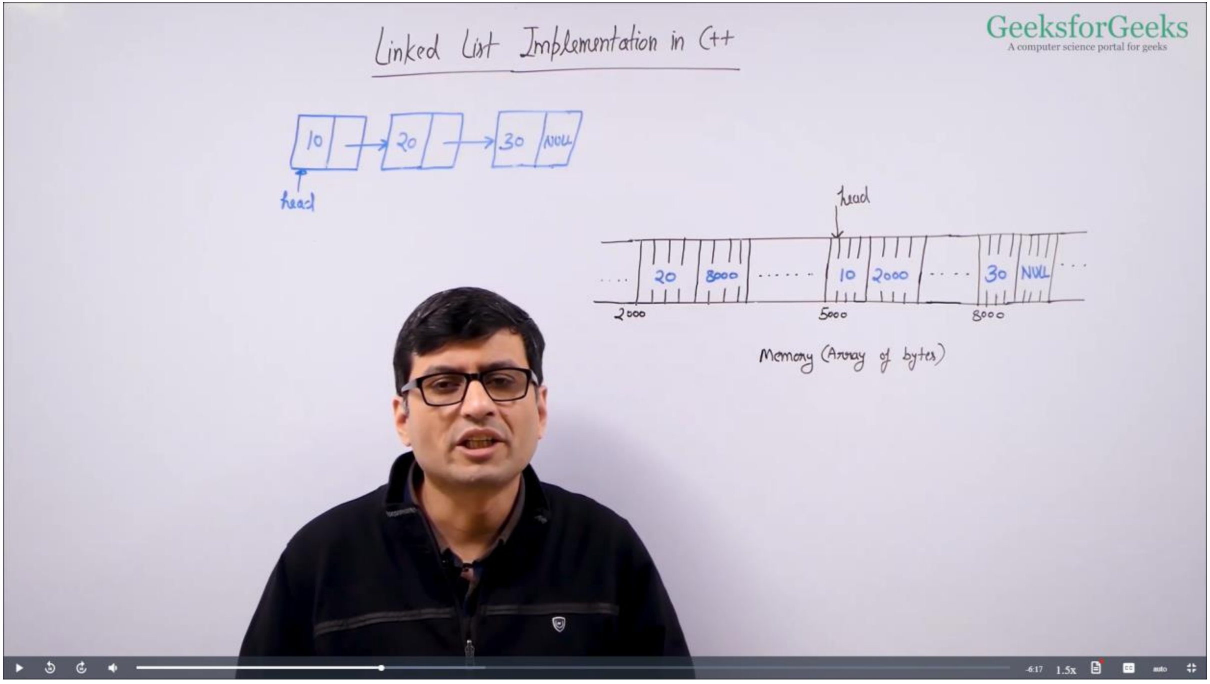
GeeksforGeeks

A computer science portal for geeks

Linked List (Background and Introduction)

The idea is to drop the contiguous memory requirments so that insertions, deletions can efficiently happen at the middle also.

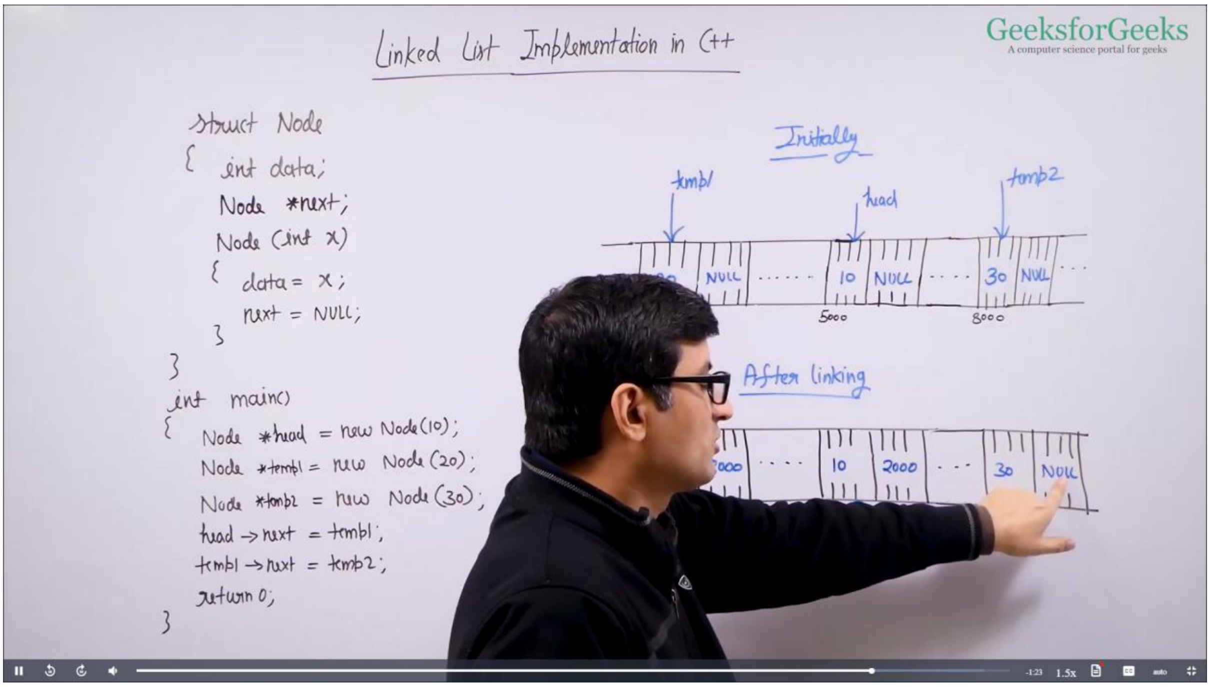
And no need to pre-allocate the space (No extra nodes)



GeeksforGeeks

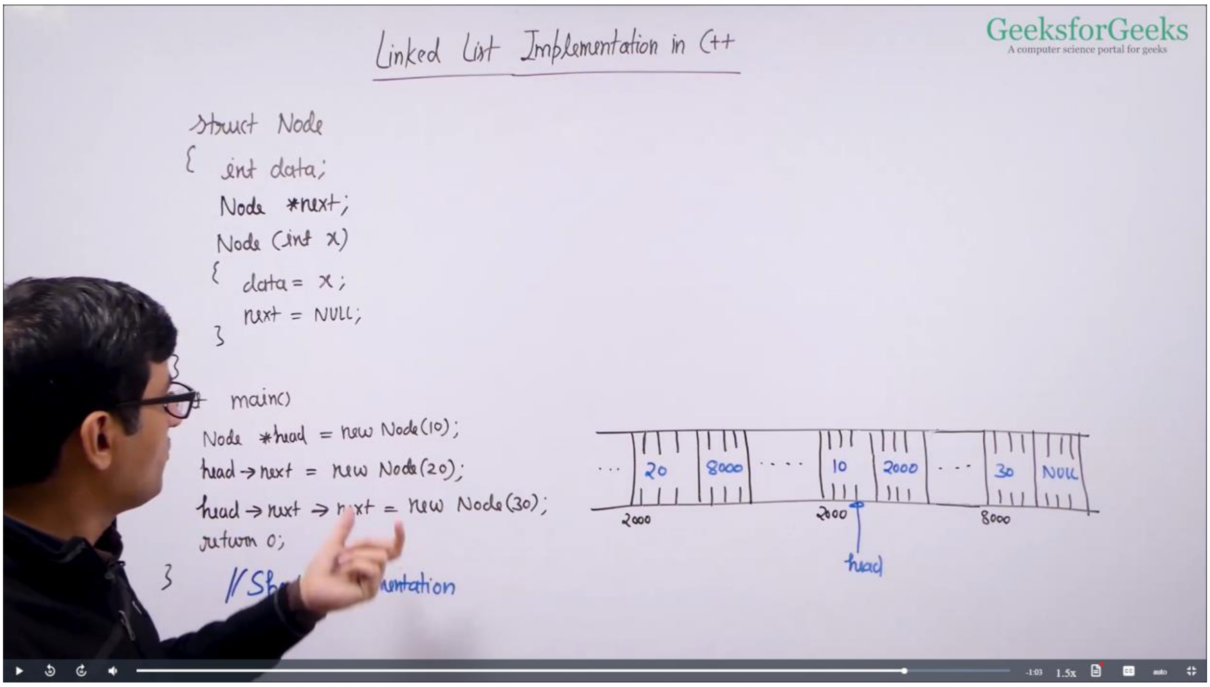
A computer science portal for geeks

Linked List Implementation in C++



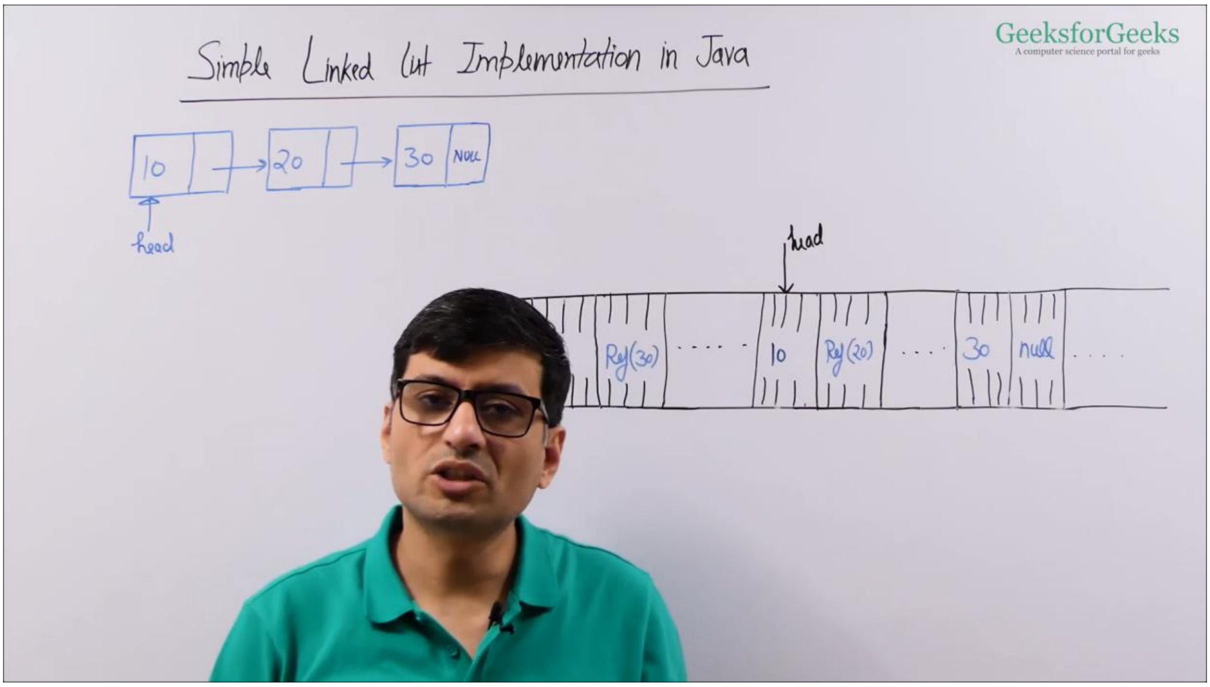
Linked List Implementation in C++

```
struct Node
{ int data;
  Node *next;
  Node (int x)
  { data = x;
    next = NULL;
  }
}
int main()
{ Node *head = new Node(10);
  Node *temp1 = new Node(20);
  Node *temp2 = new Node(30);
  head->next = temp1;
  temp1->next = temp2;
  return 0;
}
```

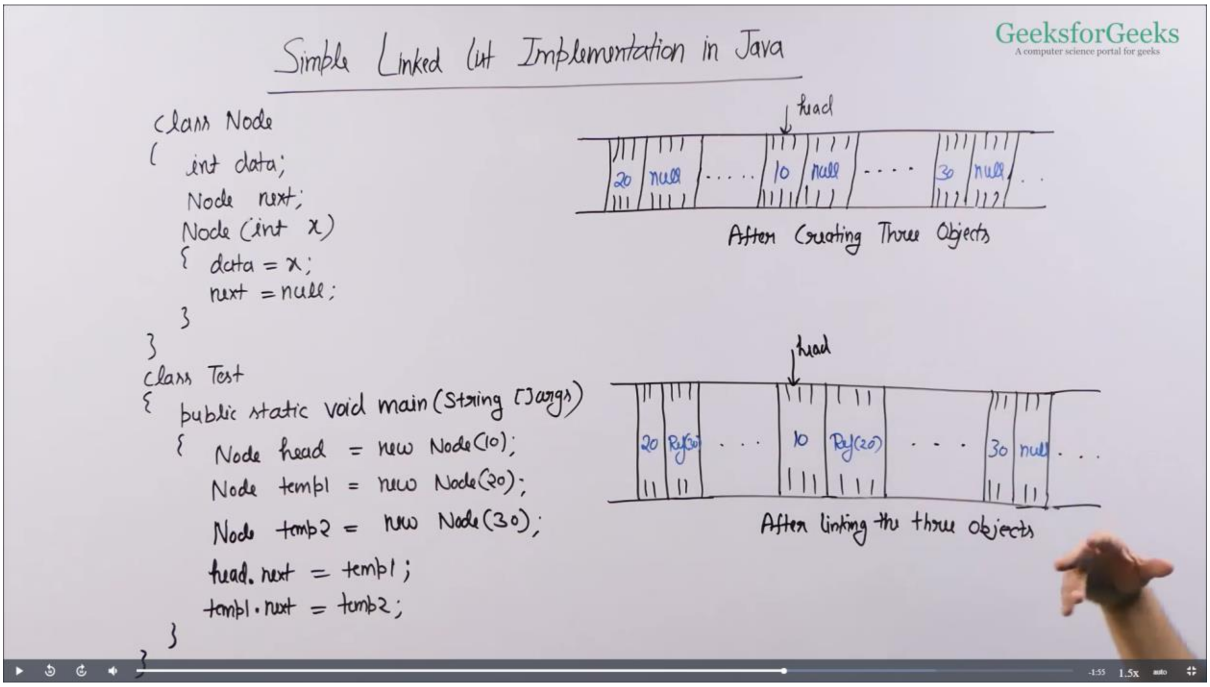


Linked List Implementation in C++

```
struct Node
{ int data;
  Node *next;
  Node (int x)
  { data = x;
    next = NULL;
  }
}
int main()
{ Node *head = new Node(10);
  head->next = new Node(20);
  head->next->next = new Node(30);
  return 0;
}  //Sh[illegible]mentation
```



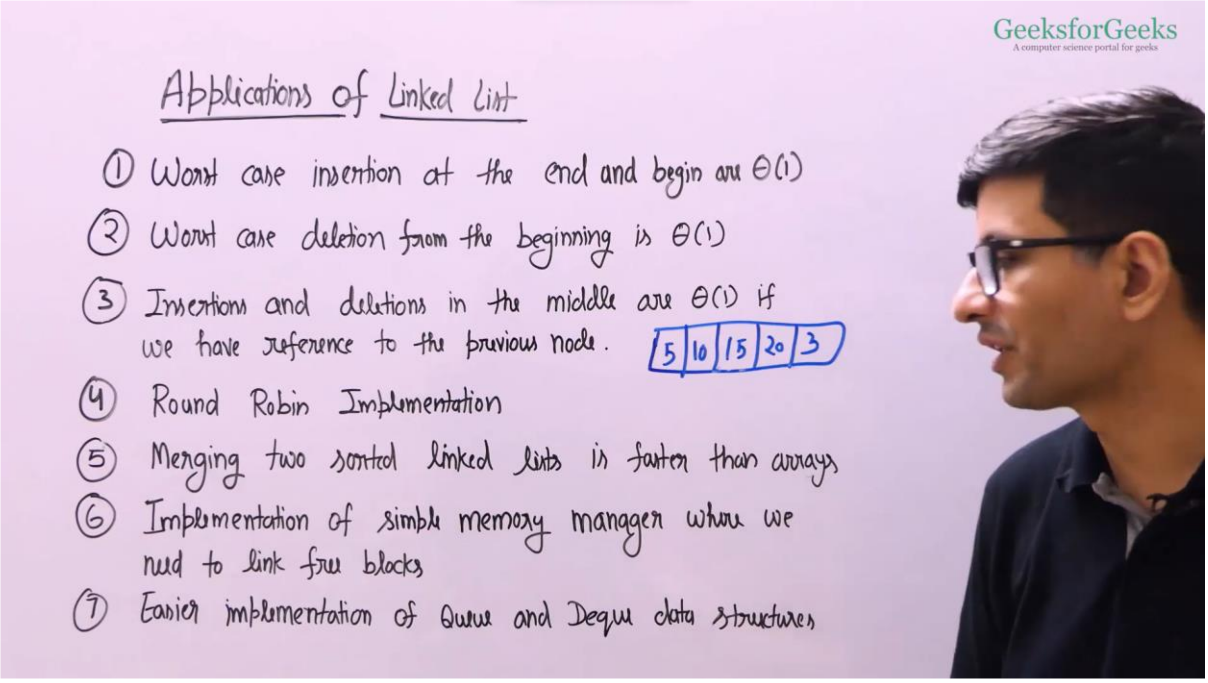
Simple Linked List Implementation in Java



Simple Linked List Implementation in Java

```
class Node
{
  int data;
  Node next;
  Node (int x)
  { data = x;
    next = null;
  }
}
class Test
{
  public static void main(String []args)
  {
    Node head = new Node(10);
    Node temp1 = new Node(20);
    Node temp2 = new Node(30);
    head.next = temp1;
    temp1.next = temp2;
  }
}
```

GeeksforGeeks
A computer science portal for geeks
head
20
null
10
null
30
null
After Creating Three Objects
head
20
Ref(30)
10
Ref(20)
30
null
After linking the three objects
-1:55
1.5x
auto

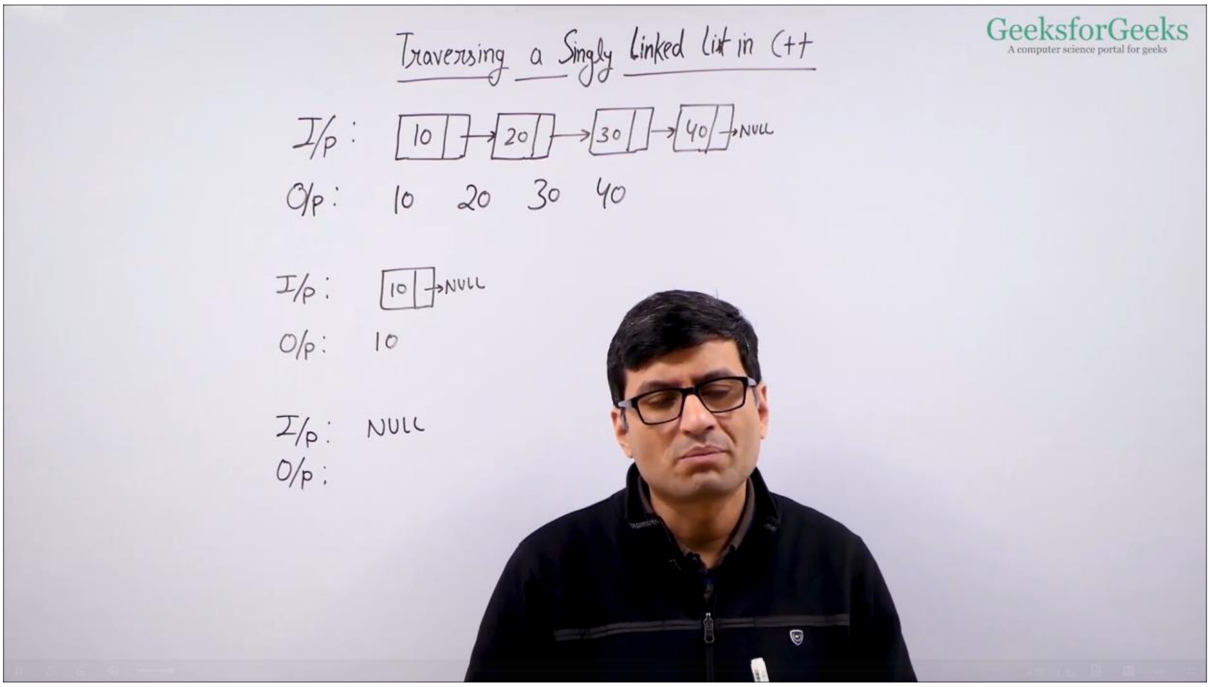


Applications of Linked List

1. Worst case insertion at the end and begin are Θ(1)
2. Worst case deletion from the beginning is Θ(1)
3. Insertions and deletions in the middle are Θ(1) if we have reference to the previous node.

5	10	15	20	3

4. Round Robin Implementation
5. Merging two sorted linked lists is faster than arrays
6. Implementation of simple memory manager where we need to link free blocks
7. Easier implementation of Queue and Deque data structures



Traversing a Singly Linked List in C++

I/p : 10 → 20 → 30 → 40 → NULL

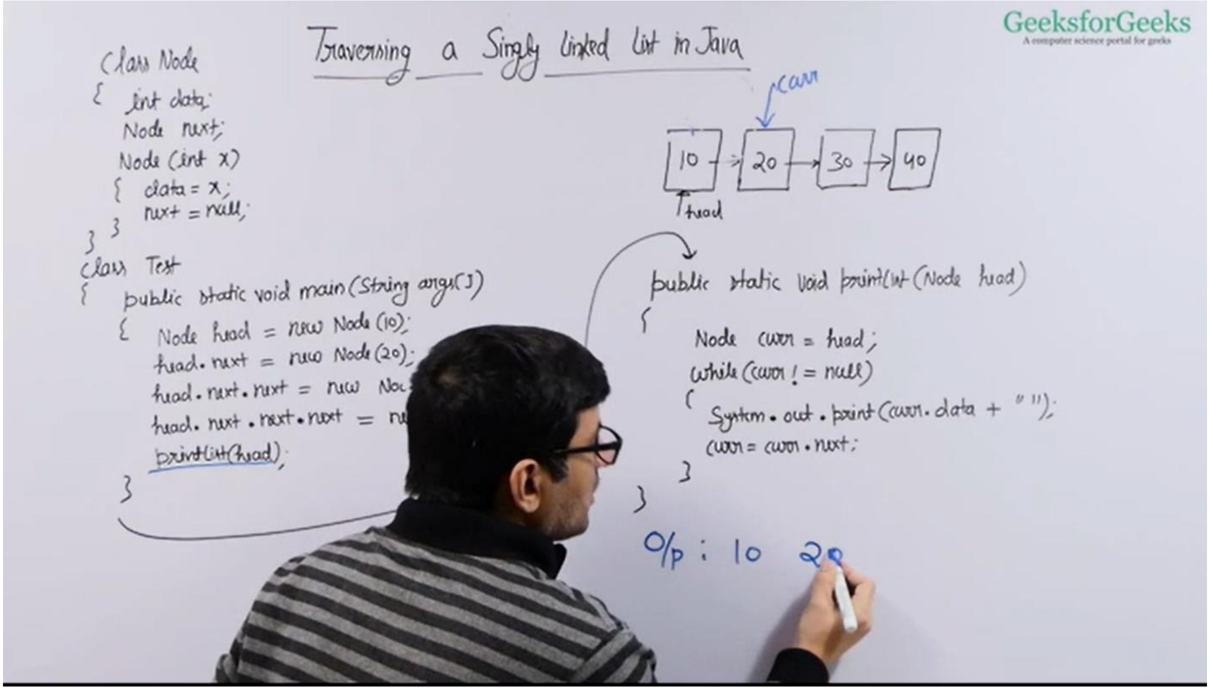
O/p : 10 20 30 40

I/p : 10 → NULL

O/p : 10

I/p : NULL

O/p :

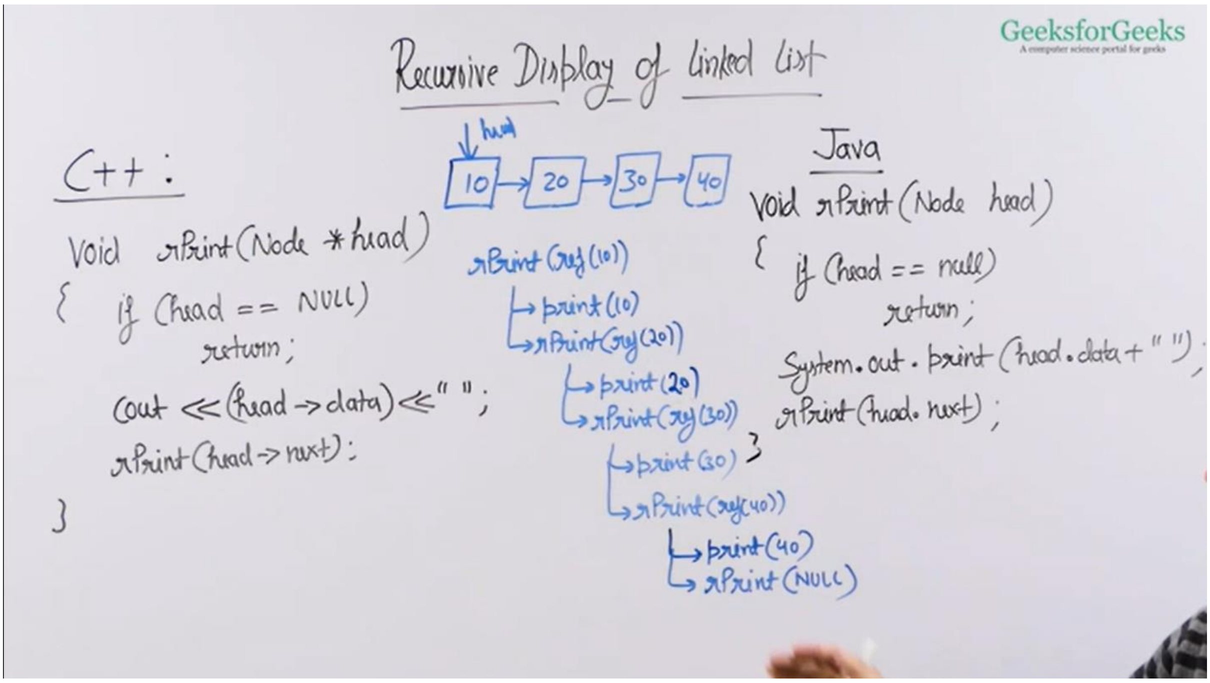


Traversing a Singly linked list in Java

```
class Node
{  int data;
   Node next;
   Node (int x)
   {  data = x;
      next = null;
   }
}
class Test
{  public static void main(String args[])
   {  Node head = new Node(10);
      head.next = new Node(20);
      head.next.next = new Noc
      head.next.next.next = n
      printList(head);
   }
```

```
public static void printList(Node head)
{
   Node curr = head;
   while(curr != null)
   {
     System.out.print(curr.data + " ");
     curr = curr.next;
   }
}
```

O/p : 10 20



Recursive Display of Linked List

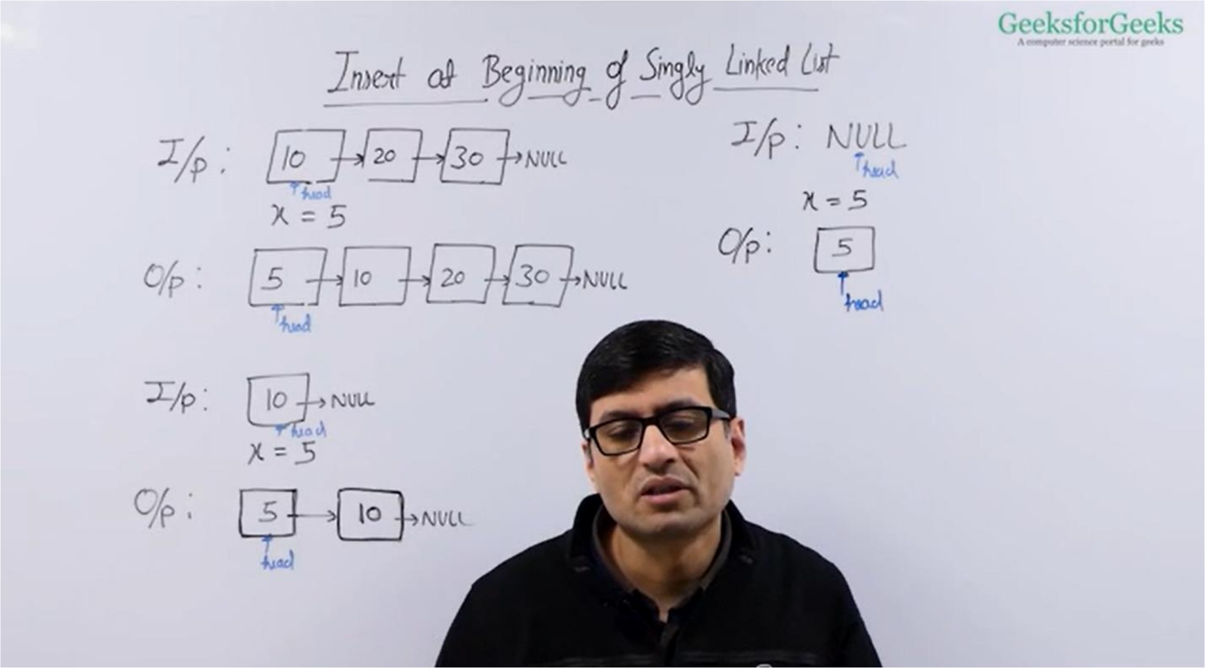
C++ :

```
Void rPrint(Node *head)
{  if (head == NULL)
        return;
   cout << (head->data) << " ";
   rPrint(head->next);
}
```

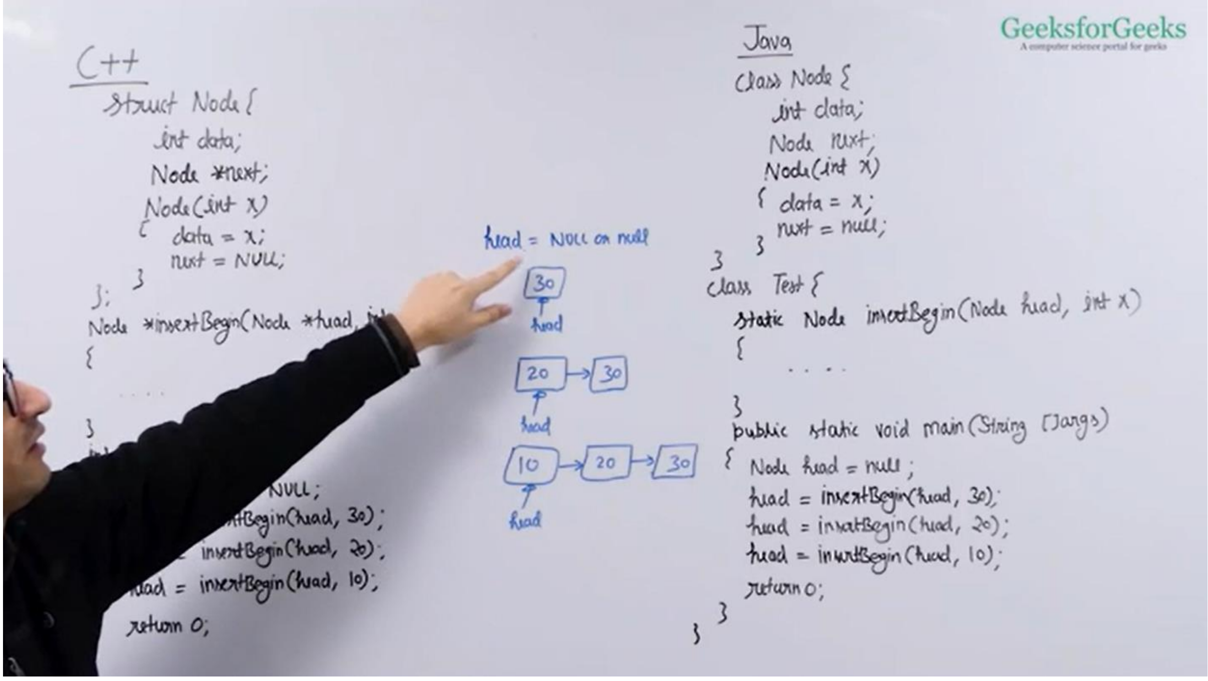
Java

```
Void rPrint(Node head)
{  if (head == null)
        return;
   System.out.print(head.data + " ");
   rPrint(head.next);
}
```

```
rPrint(ref(10))
  |-> print(10)
  |-> rPrint(ref(20))
        |-> print(20)
        |-> rPrint(ref(30))
              |-> print(30)
              |-> rPrint(ref(40))
                    |-> print(40)
                    |-> rPrint(NULL)
```



Insert at Beginning of Singly Linked List

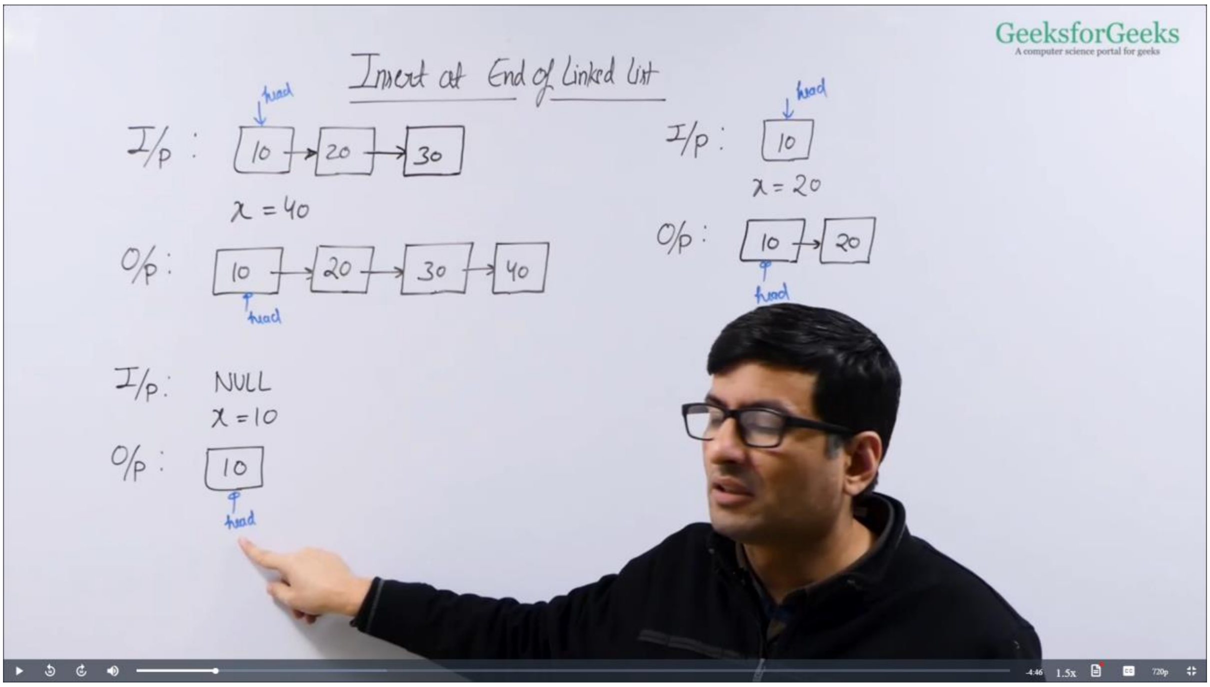


C++

```
struct Node {
    int data;
    Node *next;
    Node(int x)
    {  data = x;
       next = NULL;
    }
};
Node *insertBegin(Node *head, in
{
   . . . .
}
                NULL;
          rtBegin(head, 30);
          insertBegin(head, 20);
   head = insertBegin(head, 10);
   return 0;
```

Java

```
class Node {
    int data;
    Node next;
    Node(int x)
    { data = x;
      next = null;
    }
}
class Test {
  static Node insertBegin(Node head, int x)
  {
     . . . .
  }
  public static void main(String []args)
  { Node head = null;
    head = insertBegin(head, 30);
    head = insertBegin(head, 20);
    head = insertBegin(head, 10);
    return 0;
  }
}
```



GeeksforGeeks

A computer science portal for geeks

Insert at End of Linked List

I/p : 10 → 20 → 30

x = 40

O/p : 10 → 20 → 30 → 40

I/p : 10

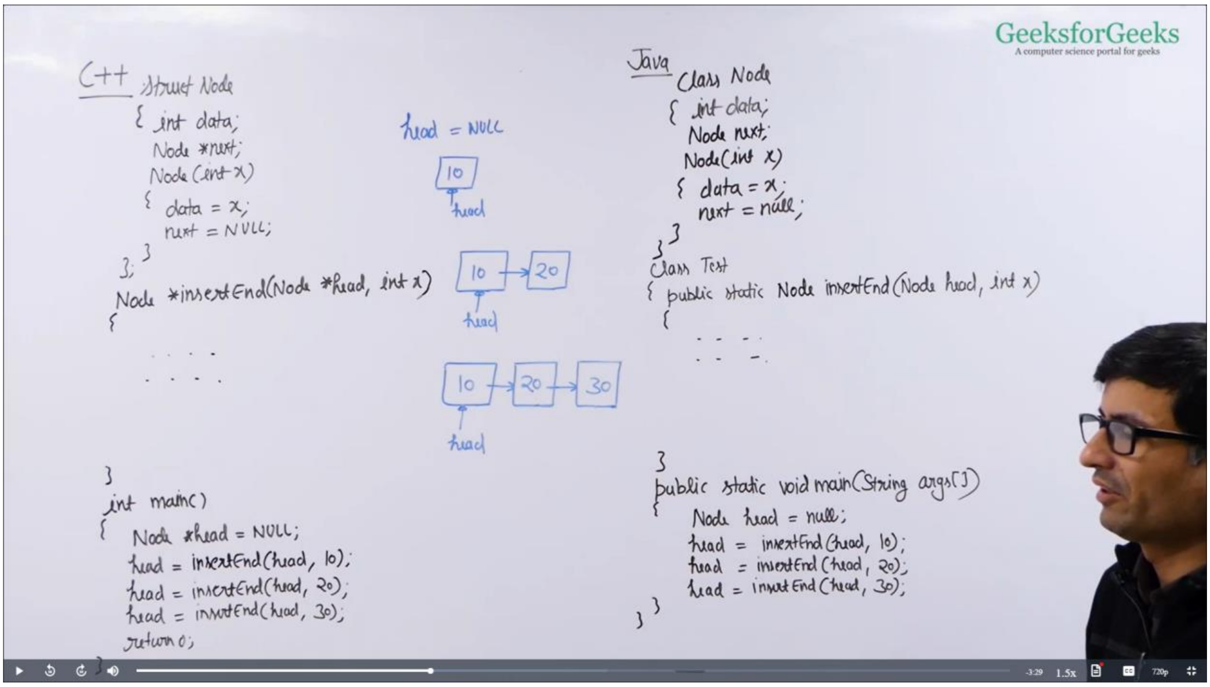
x = 20

O/p : 10 → 20

I/p : NULL

x = 10

O/p : 10



GeeksforGeeks

A computer science portal for geeks

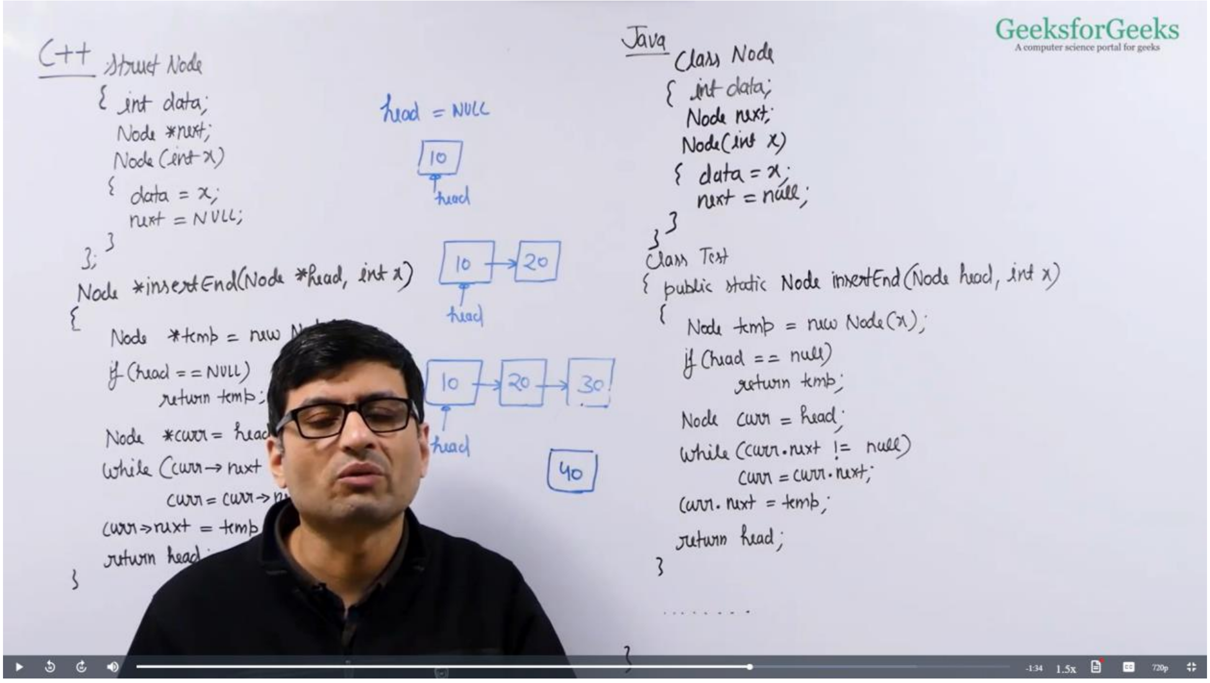
C++

```
struct Node
  { int data;
    Node *next;
    Node (int x)
    { data = x;
      next = NULL;
    }
  };
Node *insertEnd(Node *head, int x)
{
    . . . .
    . . . .

}
int main()
{   Node *head = NULL;
    head = insertEnd(head, 10);
    head = insertEnd(head, 20);
    head = insertEnd(head, 30);
    return 0;
}
```

head = NULL

Java

```
Class Node
  { int data;
    Node next;
    Node(int x)
    { data = x;
      next = null;
    }
  }
Class Test
{ public static Node insertEnd(Node head, int x)
  {
      - - -
      - - -

  }
public static void main(String args[])
{
     Node head = null;
     head = insertEnd(head, 10);
     head = insertEnd(head, 20);
     head = insertEnd(head, 30);
  }
}
```

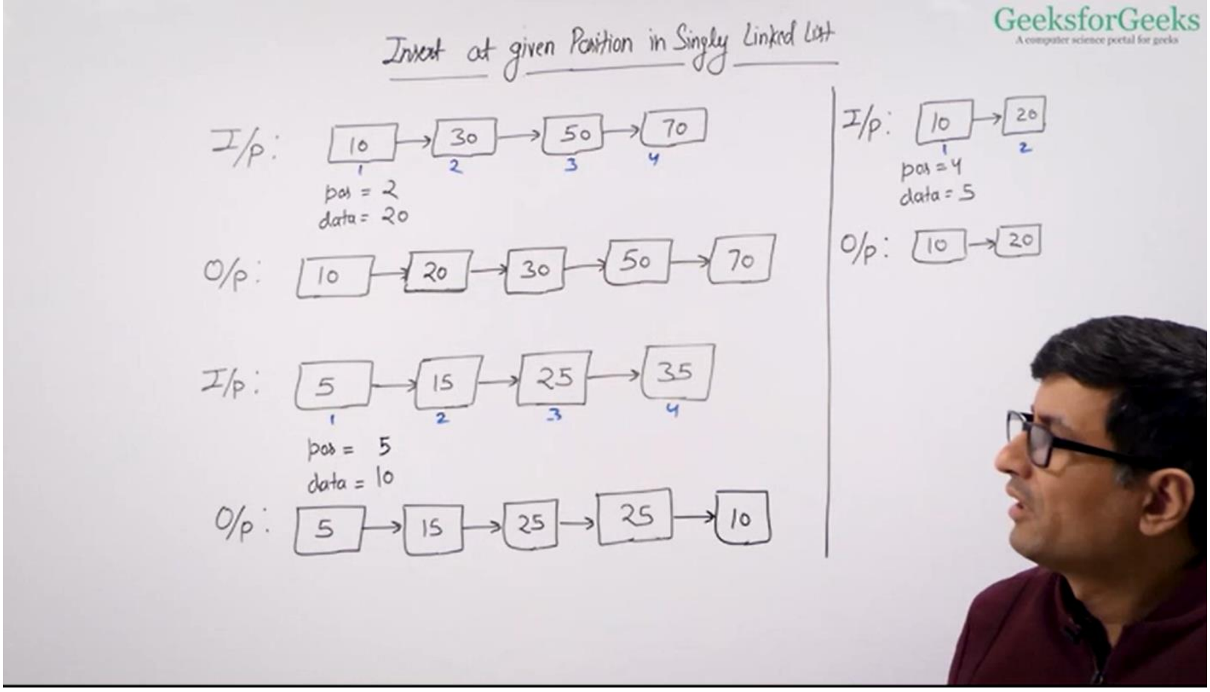


C++

```
Struct Node
{ int data;
  Node *next;
  Node (int x)
  { data = x;
    next = NVLL;
  }
};
Node *insertEnd(Node *head, int x)
{
   Node *temp = new N
   if (head == NULL)
        return temp;
   Node *curr = head
   while (curr→next
         curr = curr->n
   curr->next = temp
   return head
}
```

Java

```
Class Node
{ int data;
  Node next;
  Node(int x)
  { data = x;
    next = null;
  }
}
Class Test
{ public static Node insertEnd(Node head, int x)
  {
    Node temp = new Node(x);
    if (head == null)
         return temp;
    Node curr = head;
    while (curr.next != null)
         curr = curr.next;
    curr.next = temp;
    return head;
  }
  ........
}
```

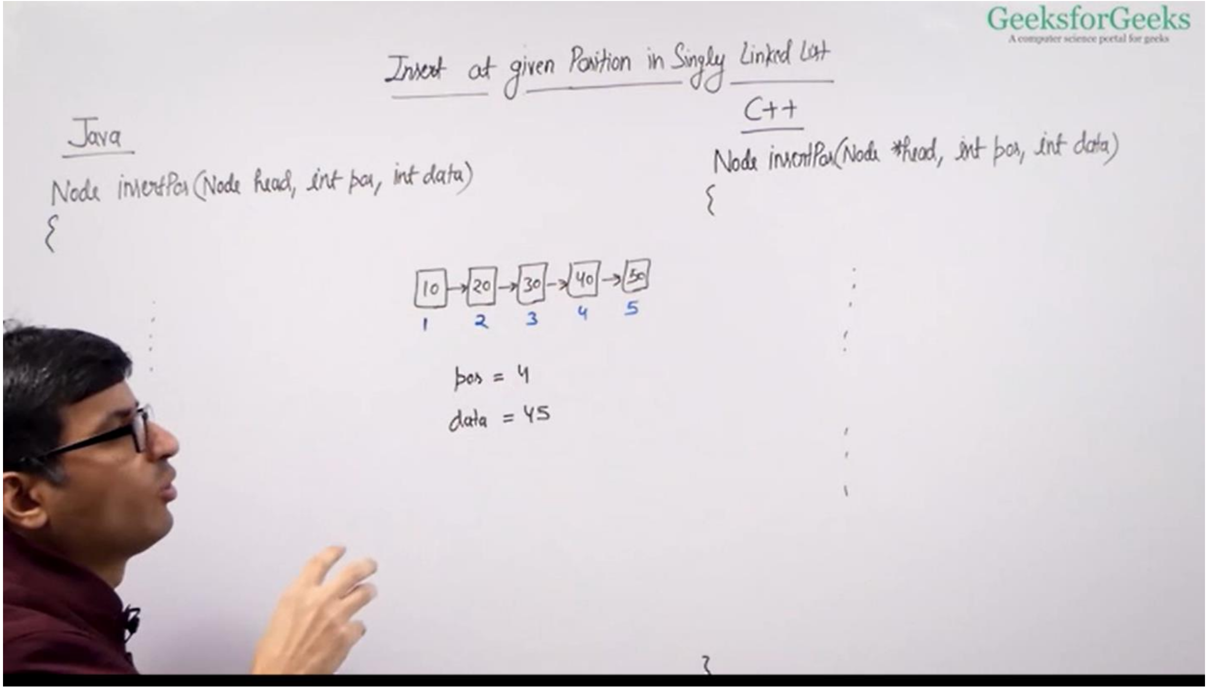


Insert at given Position in Singly Linked List

pos = 2
data = 20

pos = 5
data = 10

pos = 4
data = 5



Insert at given Position in Singly Linked List

Java

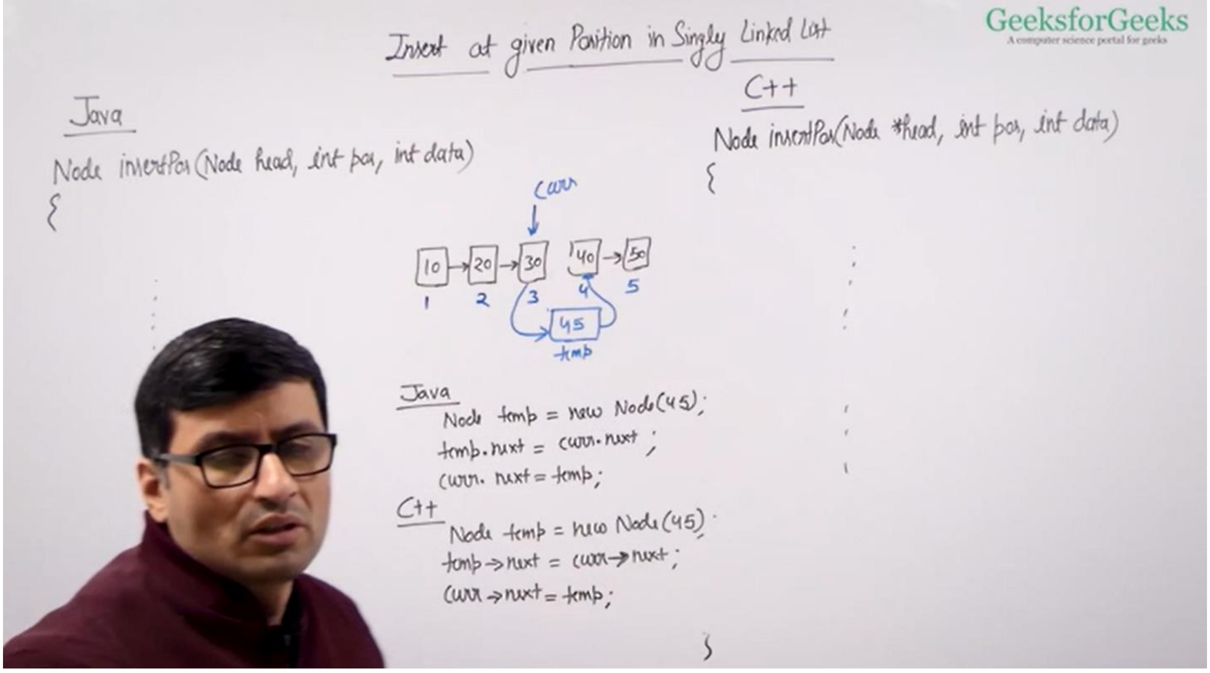
```
Node insertPos(Node head, int pos, int data)
{
```

C++

```
Node insertPos(Node *head, int pos, int data)
{


}
```

pos = 4

data = 45



Insert at given Position in Singly Linked List

Java

```
Node insertPos(Node head, int pos, int data)
{
```

C++

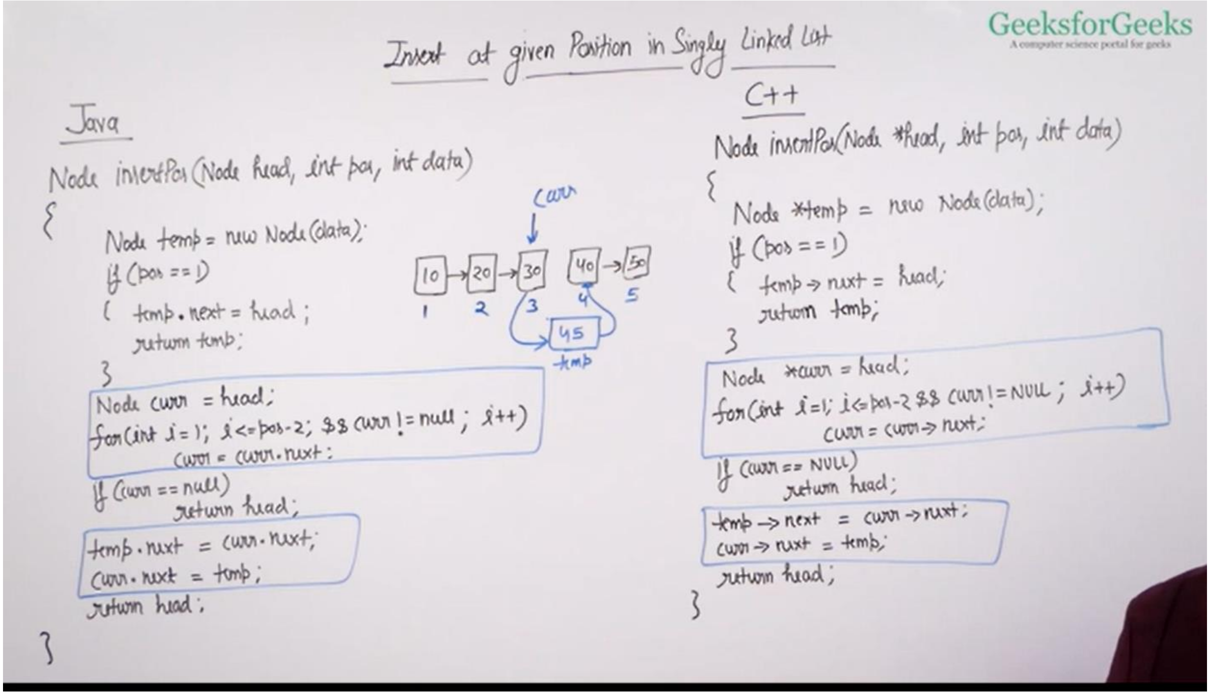
```
Node insertPos(Node *head, int pos, int data)
{
```

Java

```
Node temp = new Node(45);
temp.next = curr.next;
curr.next = temp;
```

C++

```
Node temp = new Node(45).
temp->next = curr->next;
curr->next = temp;
}
```



Insert at given Position in Singly Linked List

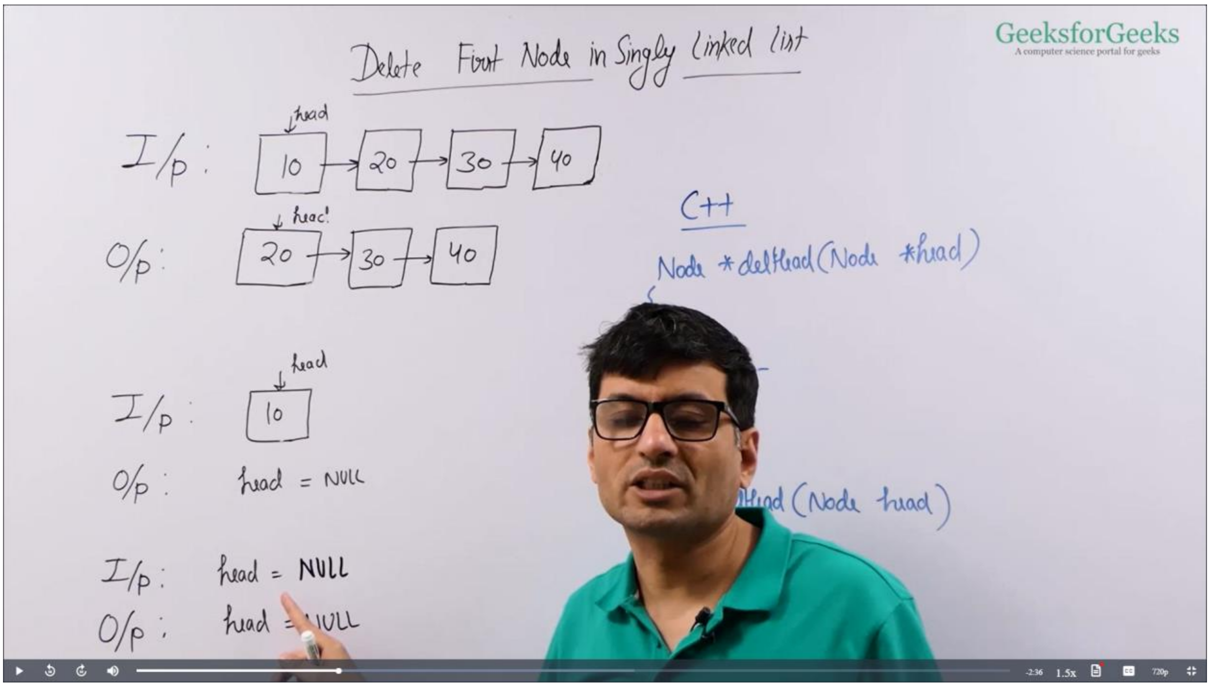
Java

```
Node insertPos(Node head, int pos, int data)
{
    Node temp = new Node(data);
    if (pos == 1)
    { temp.next = head;
      return temp;
    }
  Node curr = head;
  for(int i = 1; i<=pos-2; && curr != null ; i++)
        curr = curr.next;
  if (curr == null)
        return head;
  temp.next = curr.next;
  curr.next = temp;
  return head;
}
```

C++

```
Node insertPos(Node *head, int pos, int data)
{
  Node *temp = new Node(data);
  if (pos == 1)
  { temp -> next = head;
    return temp;
  }
 Node *curr = head;
 for(int i=1; i<=pos-2 && curr != NULL ; i++)
          curr = curr -> next;
 if (curr == NULL)
       return head;
 temp -> next = curr -> next;
 curr -> next = temp;
 return head;
}
```

GeeksforGeeks
A computer science portal for geeks



Delete First Node in Singly Linked List

I/p : head → 10 → 20 → 30 → 40

O/p : head → 20 → 30 → 40

I/p : head → 10

O/p : head = NULL

I/p : head = NULL

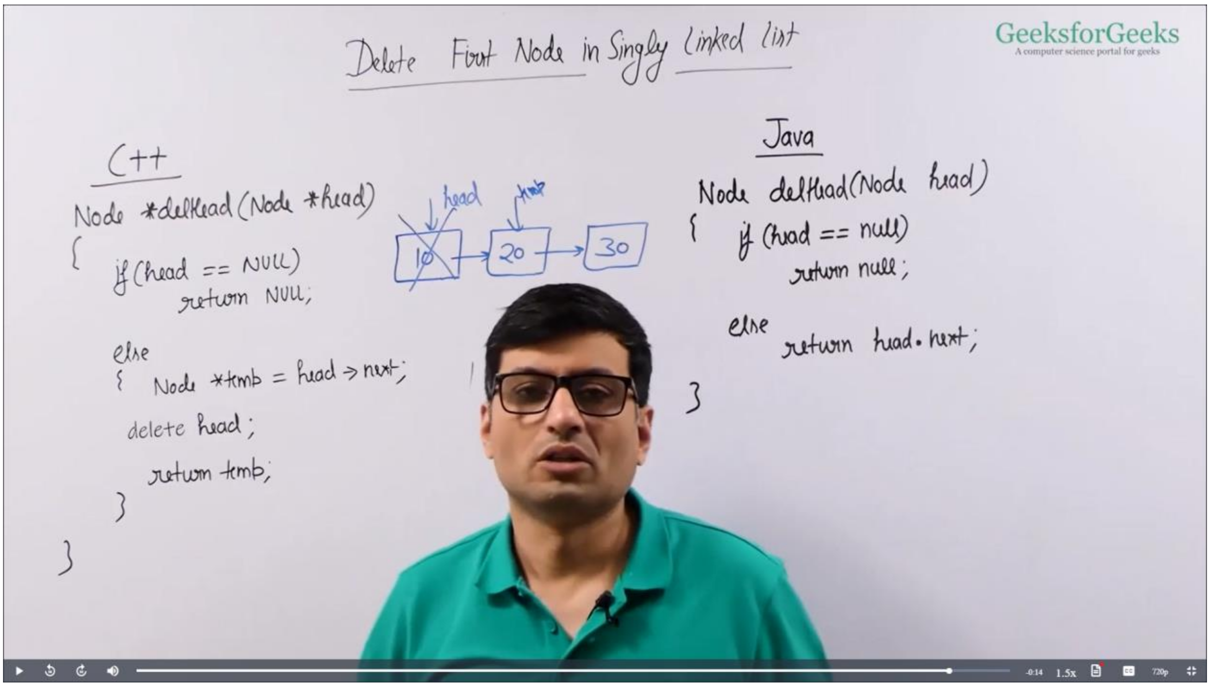
O/p : head = NULL

C++

```
Node *delHead(Node *head)
{
```

```
...lHead (Node head)
```

GeeksforGeeks
A computer science portal for geeks
-2:36
1.5x
720p



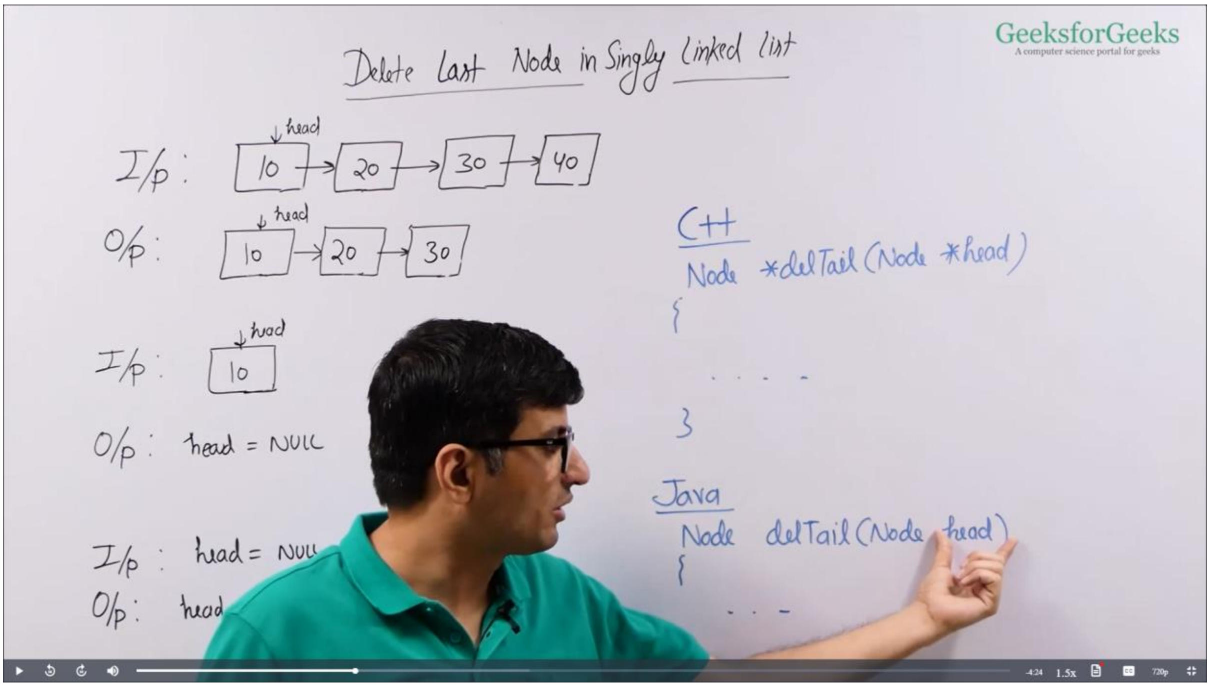
Delete First Node in Singly Linked list

C++

```
Node *delHead(Node *head)
{
    if(head == NULL)
        return NULL;

    else
    {   Node *temp = head->next;
     delete head;
      return temp;
    }
}
```

Java

```
Node delHead(Node head)
{
   if (head == null)
       return null;

  else  return head.next;

}
```



Delete Last Node in Singly Linked list

I/p :

O/p :

I/p :

O/p : head = NULL

I/p : head = NULL

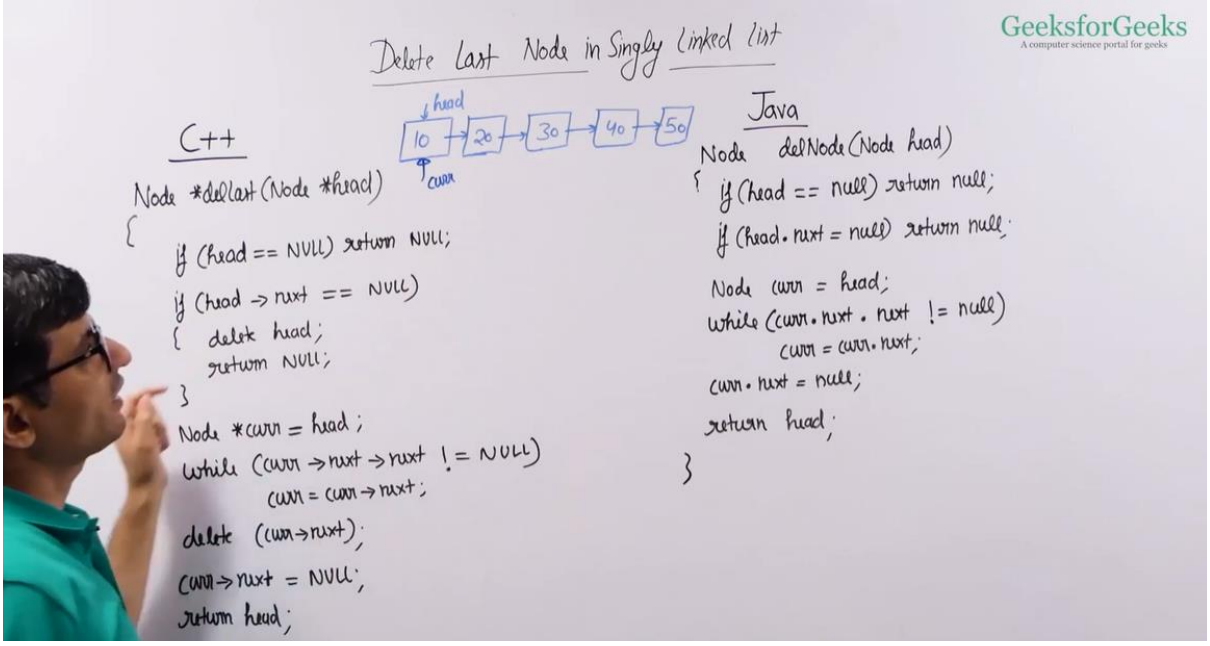
O/p : head

C++

```
Node *delTail(Node *head)
{

    . . . .

}
```

Java

```
Node  delTail(Node head)
{
   . . .
```



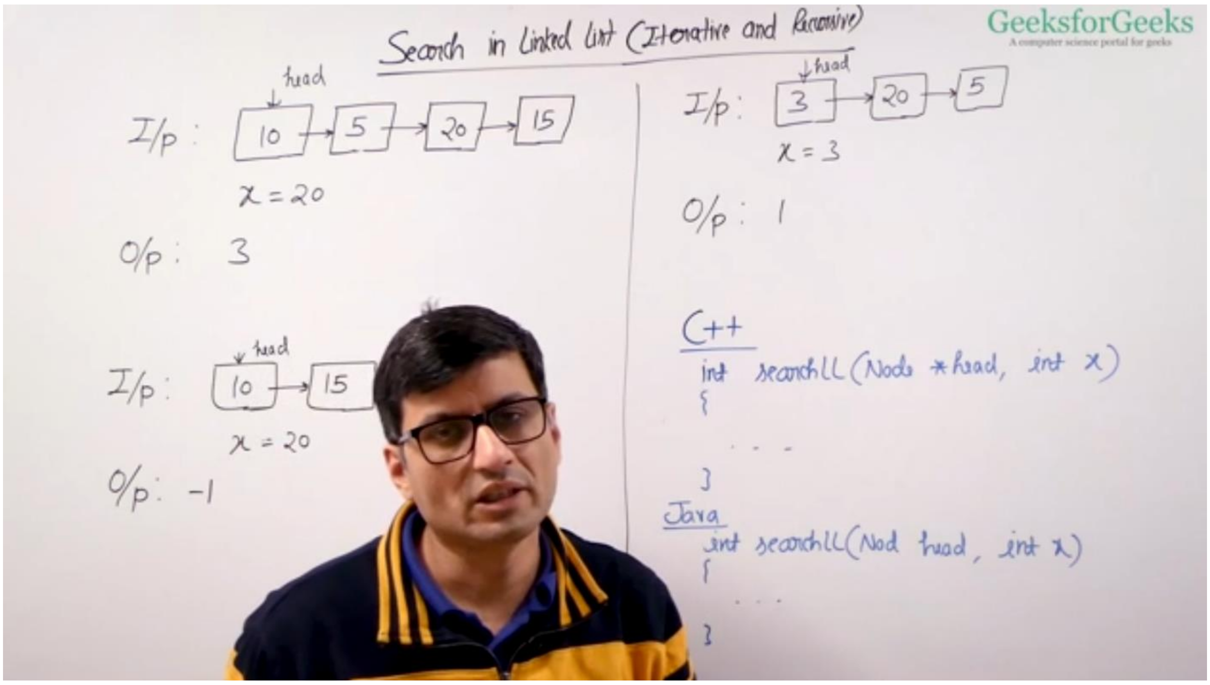
Delete Last Node in Singly Linked List

C++

```
Node *delLast(Node *head)
{
    if (head == NULL) return NULL;
    if (head -> next == NULL)
    {  delete head;
       return NULL;
    }
    Node *curr = head;
    while (curr -> next -> next != NULL)
            curr = curr -> next;
    delete (curr->next);
    curr->next = NULL;
    return head;
```

Java

```
Node  delNode(Node head)
{  if (head == null) return null;
   if (head.next = null) return null;
   Node curr = head;
   while (curr.next.next != null)
         curr = curr.next;
   curr.next = null;
   return head;
}
```



Search in Linked List (Iterative and Recursive)

I/p :

x = 20

O/p : 3

I/p :

x = 20

O/p : -1

I/p :

x = 3

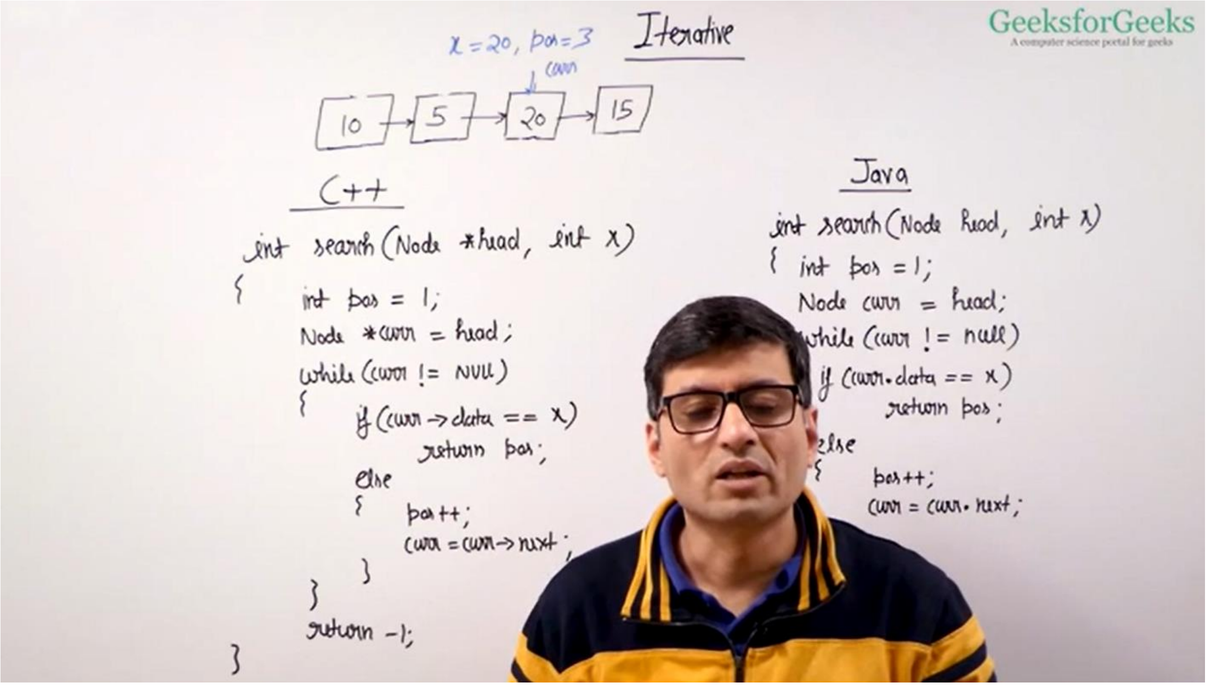
O/p : 1

C++

```
int searchLL(Node *head, int x)
{
    . . .
}
```

Java

```
int searchLL(Nod head, int x)
{
    . . .
}
```



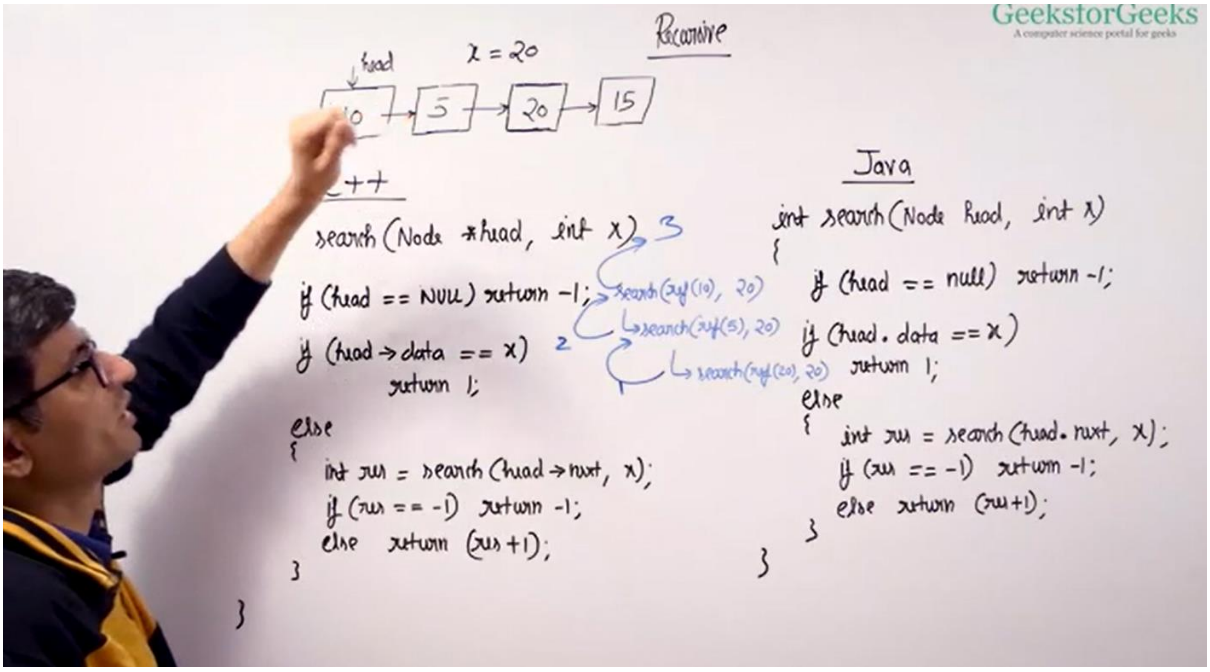
Iterative

C++

```
int search(Node *head, int x)
{
      int pos = 1;
      Node *curr = head;
      while (curr != NULL)
      {
            if (curr->data == x)
                  return pos;
            else
            {
                  pos++;
                  curr = curr->next;
            }
      }
      return -1;
}
```

Java

```
int search(Node head, int x)
{ int pos = 1;
  Node curr = head;
  while (curr != null)
    if (curr.data == x)
          return pos;
   else
   {
        pos++;
        curr = curr.next;
```



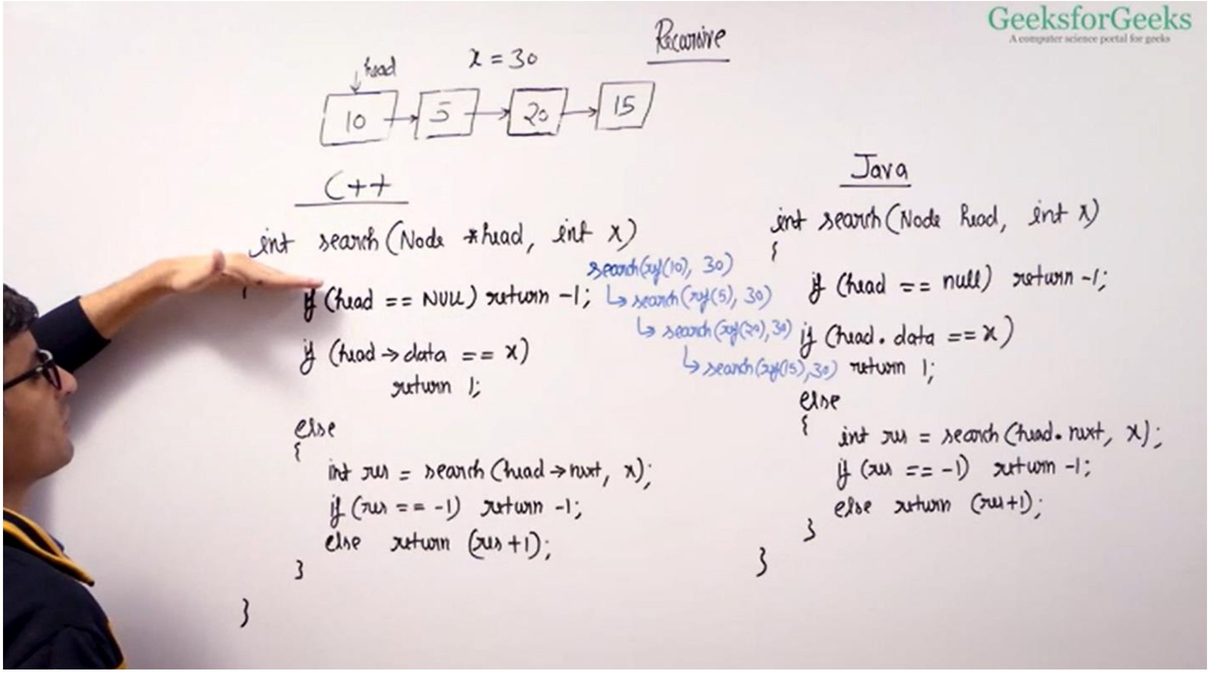
Recursive

C++

```
  search(Node *head, int x)

  if (head == NULL) return -1;

  if (head->data == x)
          return 1;
  else
  {
     int res = search(head->next, x);
     if (res == -1) return -1;
     else return (res+1);
  }
}
```

Java

```
int search(Node head, int x)
{
  if (head == null) return -1;

  if (head.data == x)
   return 1;
  else
  {
     int res = search(head.next, x);
     if (res == -1) return -1;
     else return (res+1);
  }
}
```



Recursive

x = 30

C++

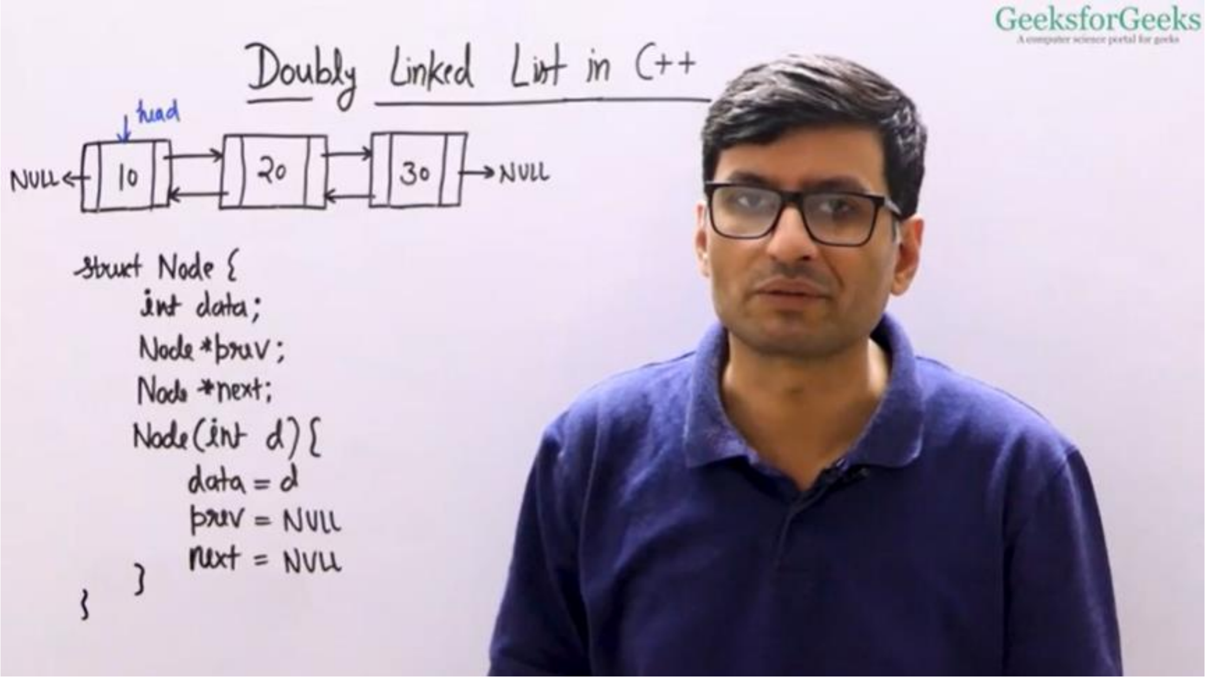
```
int search(Node *head, int x)
{
    if (head == NULL) return -1;
    if (head->data == x)
        return 1;
    else
    {
        int res = search(head->next, x);
        if (res == -1) return -1;
        else return (res+1);
    }
}
```

```
search(ref(10), 30)
 ↳ search(ref(5), 30)
    ↳ search(ref(20), 30)
       ↳ search(ref(15), 30)
```

Java

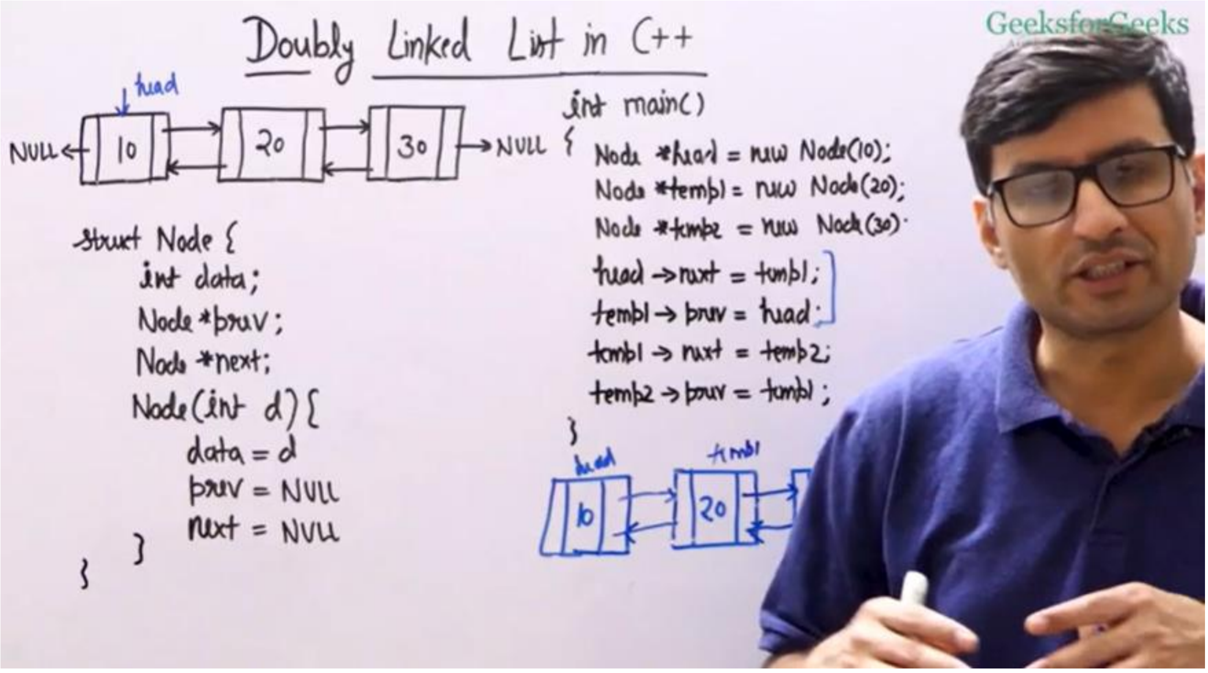
```
int search(Node head, int x)
{
    if (head == null) return -1;
    if (head.data == x)
        return 1;
    else
    {
        int res = search(head.next, x);
        if (res == -1) return -1;
        else return (res+1);
    }
}
```

\



Doubly Linked List in C++

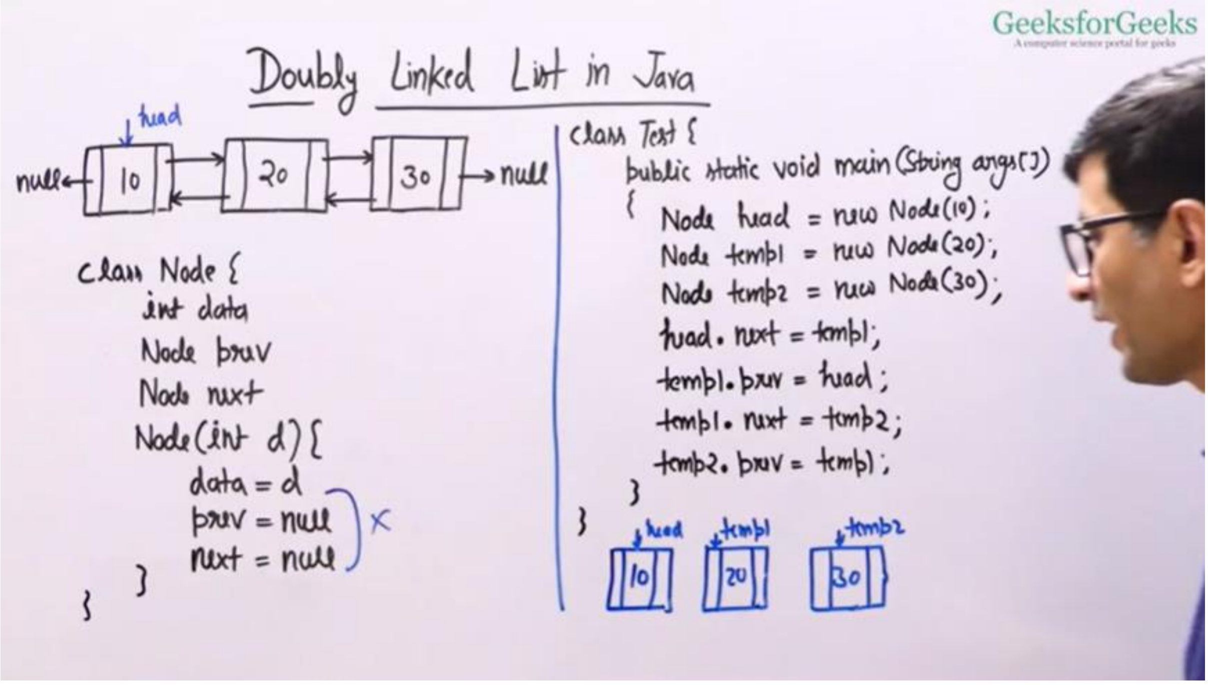
```
struct Node {
    int data;
    Node *prev;
    Node *next;
    Node(int d) {
        data = d
        prev = NULL
        next = NULL
    }
}
```



Doubly Linked List in C++

```
struct Node {
    int data;
    Node *prev;
    Node *next;
    Node(int d) {
        data = d
        prev = NULL
        next = NULL
    }
}
```

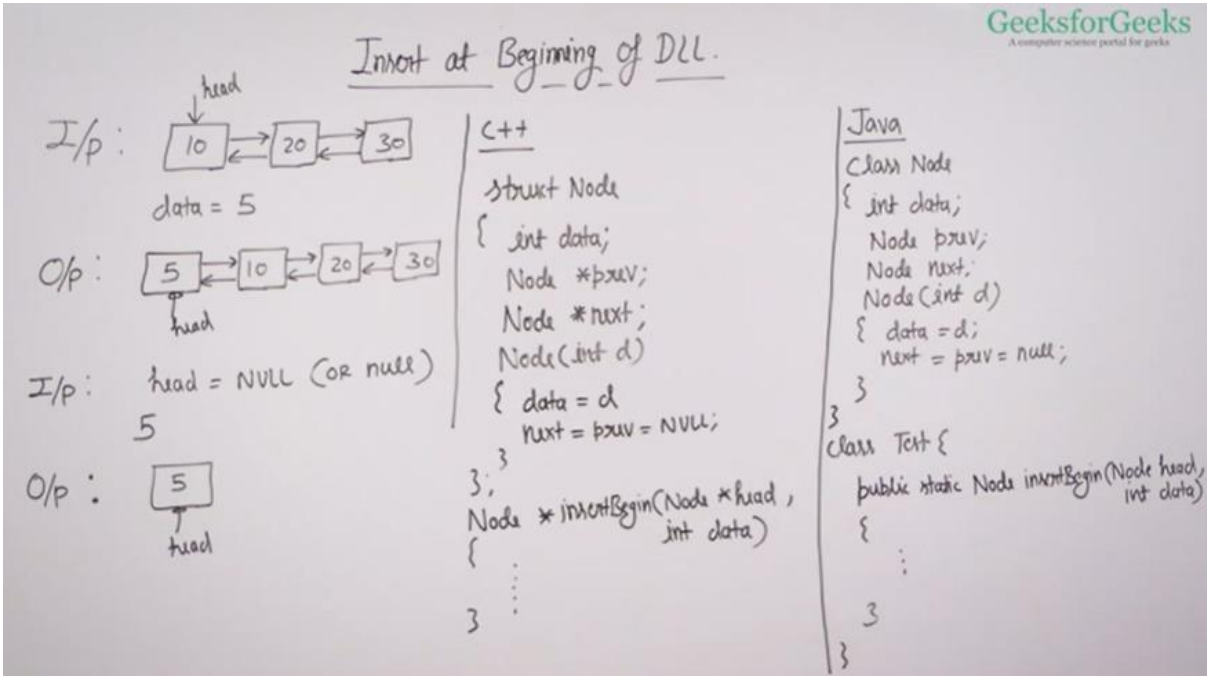
```
int main()
{
  Node *head = new Node(10);
  Node *temp1 = new Node(20);
  Node *temp2 = new Node(30)·
  head -> next = temp1;
  temp1 -> prev = head;
  temp1 -> next = temp2;
  temp2 -> prev = temp1;
}
```



Doubly Linked List in Java

```
class Node {
    int data
    Node prev
    Node next
    Node(int d) {
        data = d
        prev = null
        next = null
    }
}
```

```
class Test {
    public static void main(String args[])
    {
        Node head = new Node(10);
        Node temp1 = new Node(20);
        Node temp2 = new Node(30);
        head. next = temp1;
        temp1. prev = head;
        temp1. next = temp2;
        temp2. prev = temp1;
    }
}
```



Insert at Begining of DLL.

I/p : head → 10 ⇄ 20 ⇄ 30

data = 5

O/p : 5 ⇄ 10 ⇄ 20 ⇄ 30
head

I/p : head = NULL (OR null)
5

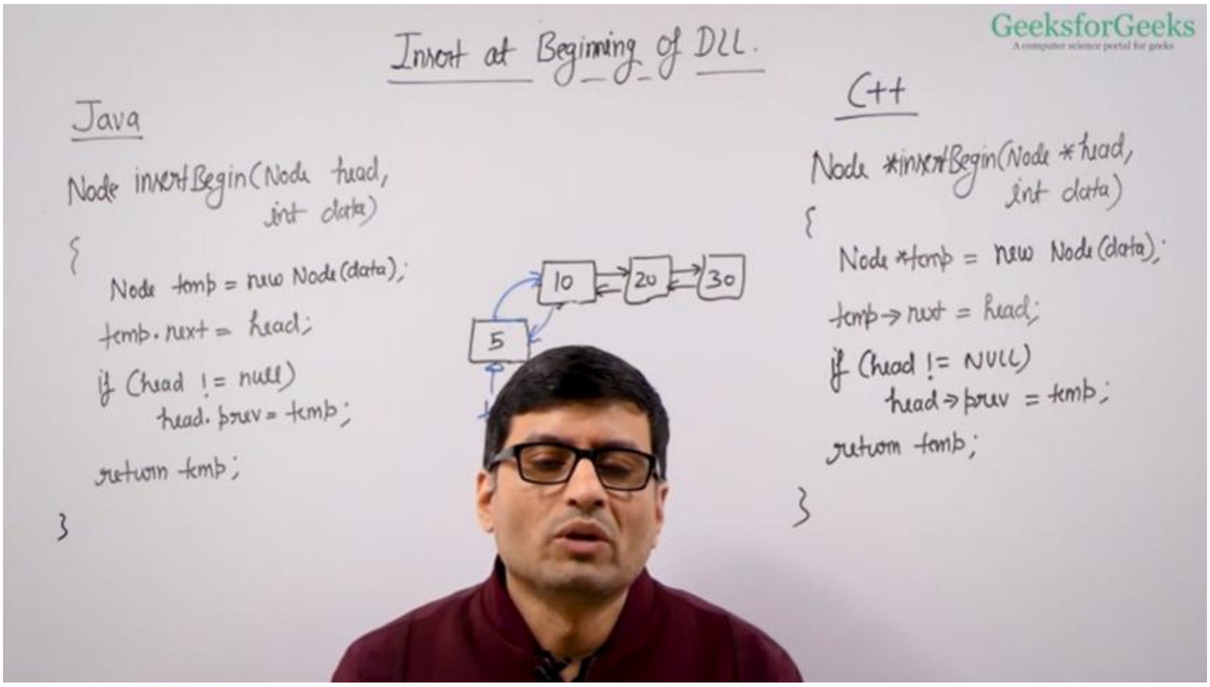
O/p : 5
head

C++

```
struct Node
{ int data;
  Node *prev;
  Node *next;
  Node(int d)
  { data = d
    next = prev = NULL;
  }
};
Node *insertBegin(Node *head,
                   int data)
{
   :
}
```

Java

```
Class Node
{ int data;
  Node prev;
  Node next;
  Node(int d)
  { data = d;
    next = prev = null;
  }
}
class Test {
  public static Node insertBegin(Node head,
                                  int data)
  {
    :

  }
}
```



Insert at Begining of DLL.

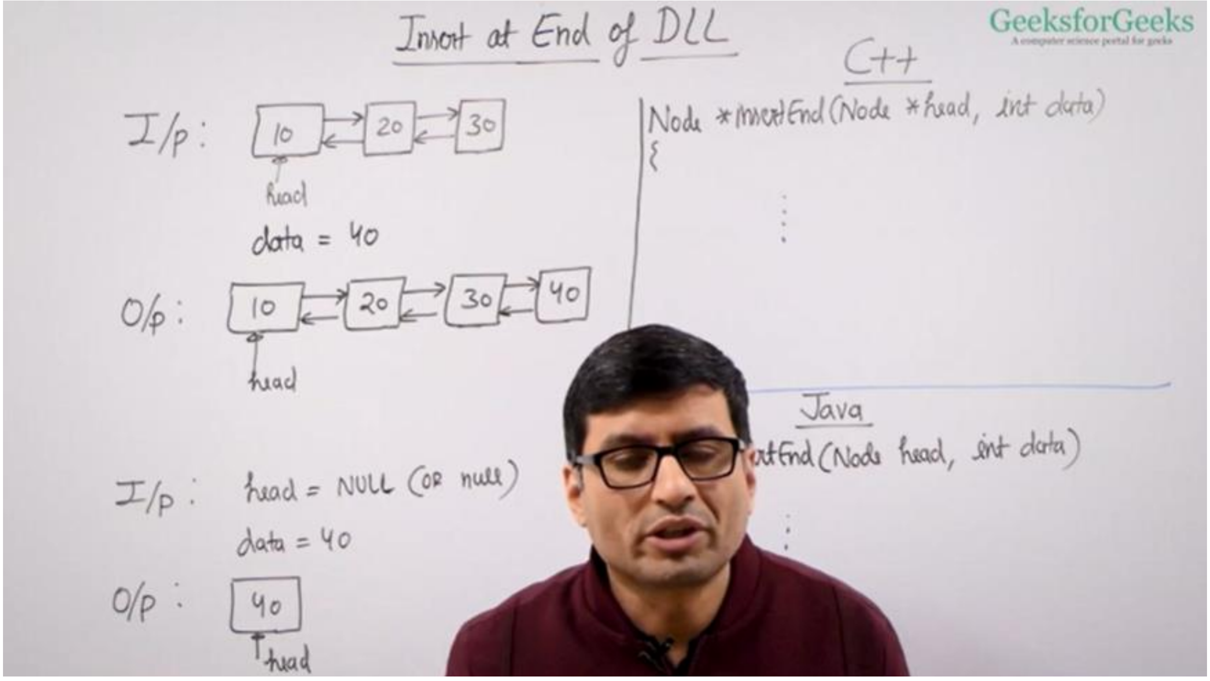
Java

```
Node insertBegin(Node head,
                  int data)
{
  Node temp = new Node(data);
  temp.next = head;
  if (head != null)
     head.prev = temp;
  return temp;
}
```

5 ⇄ 10 ⇄ 20 ⇄ 30

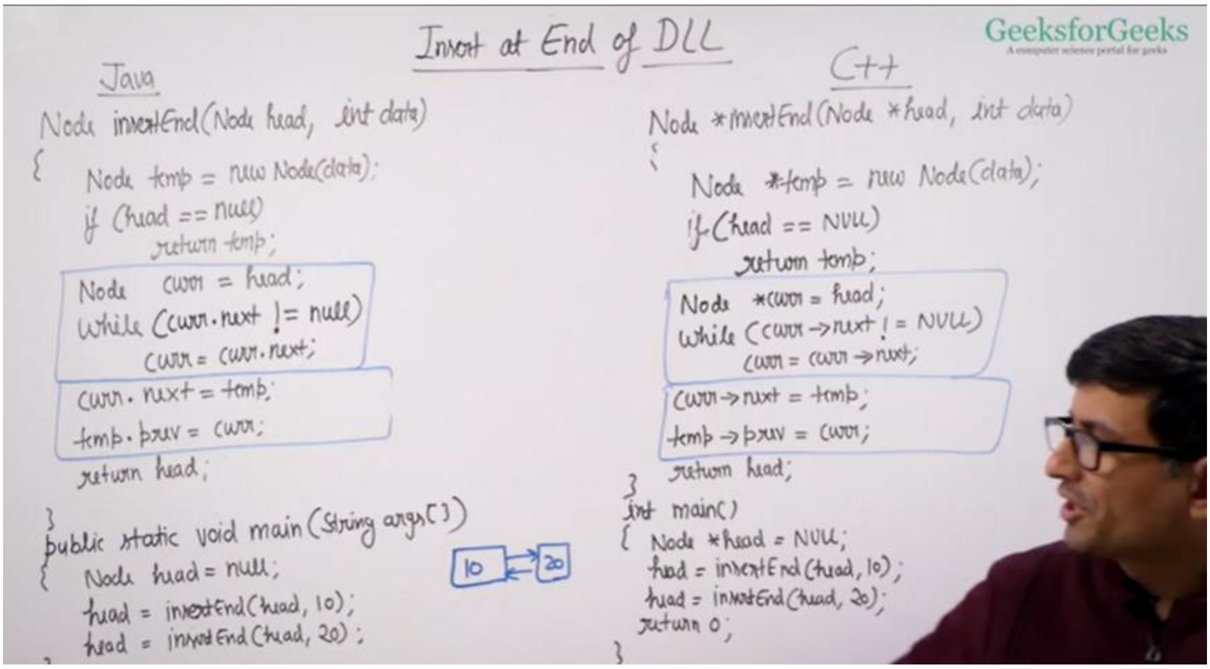
C++

```
Node *insertBegin(Node *head,
                   int data)
{
  Node *temp = new Node(data);
  temp->next = head;
  if (head != NULL)
     head->prev = temp;
  return temp;
}
```



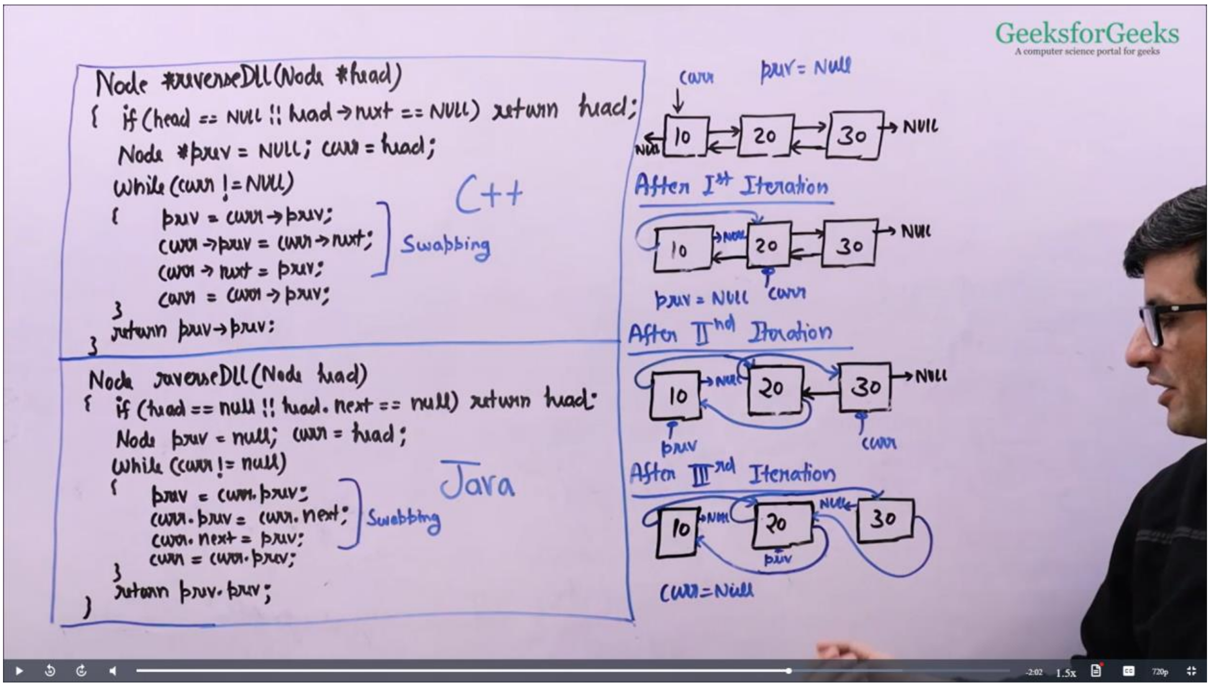
```
Node *insertEnd(Node *head, int data)
{
    ⋮
```

```
...rtEnd(Node head, int data)
    ⋮
```



```
Node insertEnd(Node head, int data)
{   Node temp = new Node(data);
    if (head == null)
        return temp;
    Node curr = head;
    while (curr.next != null)
        curr = curr.next;
    curr.next = temp;
    temp.prev = curr;
    return head;
}
public static void main(String args[])
{   Node head = null;
    head = insertEnd(head, 10);
    head = insertEnd(head, 20);
```

```
Node *insertEnd(Node *head, int data)
{
    Node *temp = new Node(data);
    if(head == NULL)
        return temp;
    Node *curr = head;
    while (curr->next != NULL)
        curr = curr->next;
    curr->next = temp;
    temp->prev = curr;
    return head;
}
int main()
{ Node *head = NULL;
  head = insertEnd(head, 10);
  head = insertEnd(head, 20);
  return 0;
}
```

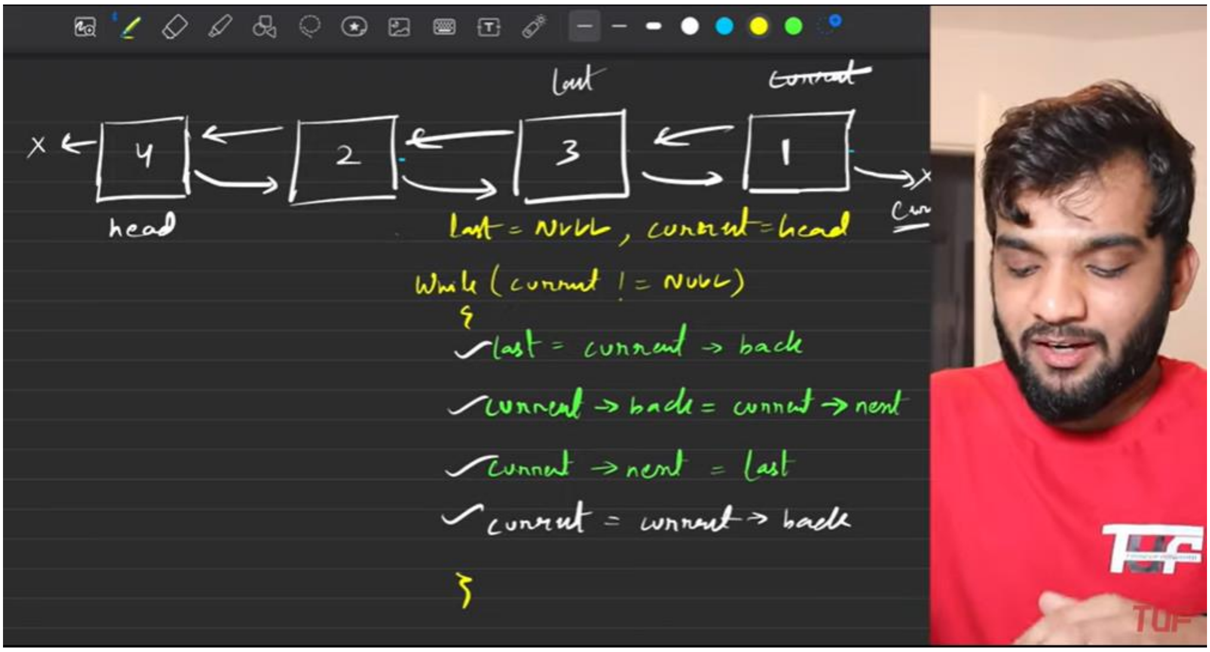


```
Node *reverseDLL(Node *head)
{  if(head == NULL || head->next == NULL) return head;
   Node *prev = NULL; curr = head;
   While(curr != NULL)
   {    prev = curr->prev;
        curr->prev = curr->next;
        curr->next = prev;
        curr = curr->prev;
   }
   return prev->prev;
}
```

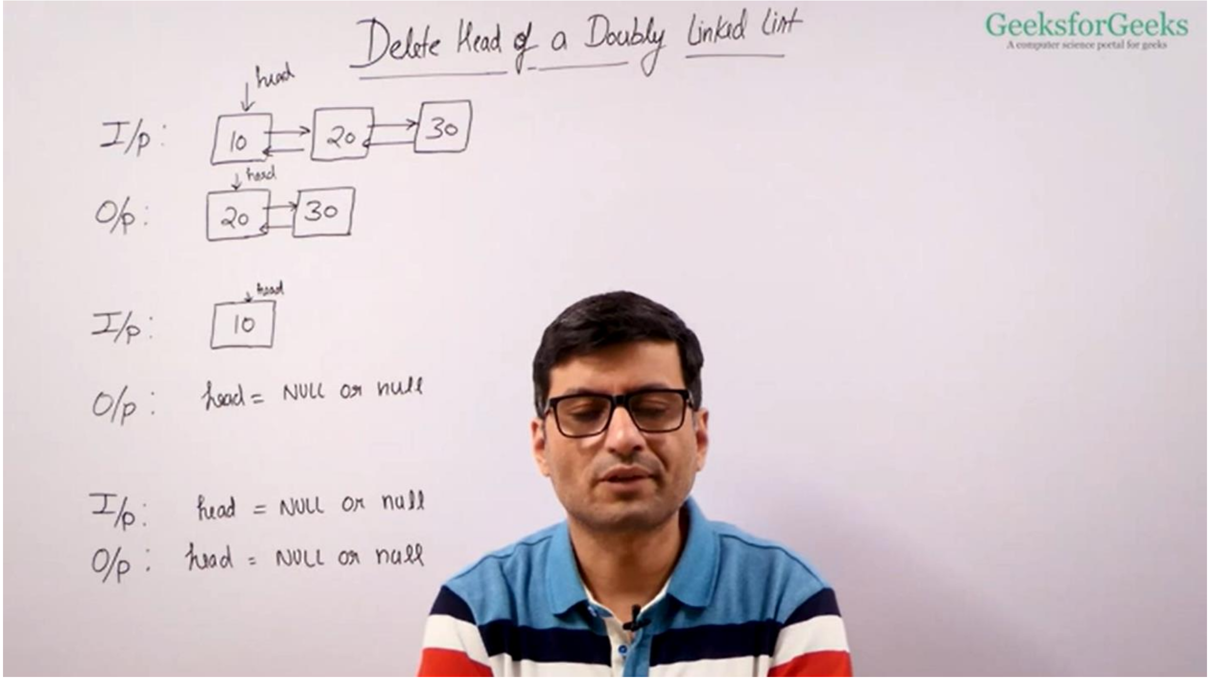
C++ Swapping

```
Node reverseDLL(Node head)
{  if (head == null || head.next == null) return head:
   Node prev = null; curr = head;
   While (curr != null)
   {   prev = curr.prev;
       curr.prev = curr.next;
       curr.next = prev;
       curr = curr.prev;
   }
   return prev.prev;
}
```

Java Swapping



```
last = NULL , current = head

While (current != NULL)
{
  ✓ last = current -> back
  ✓ current -> back = current -> next
  ✓ current -> next = last
  ✓ current = current -> back
}
```



Delete Head of a Doubly Linked List

GeeksforGeeks
A computer science portal for geeks

I/p :

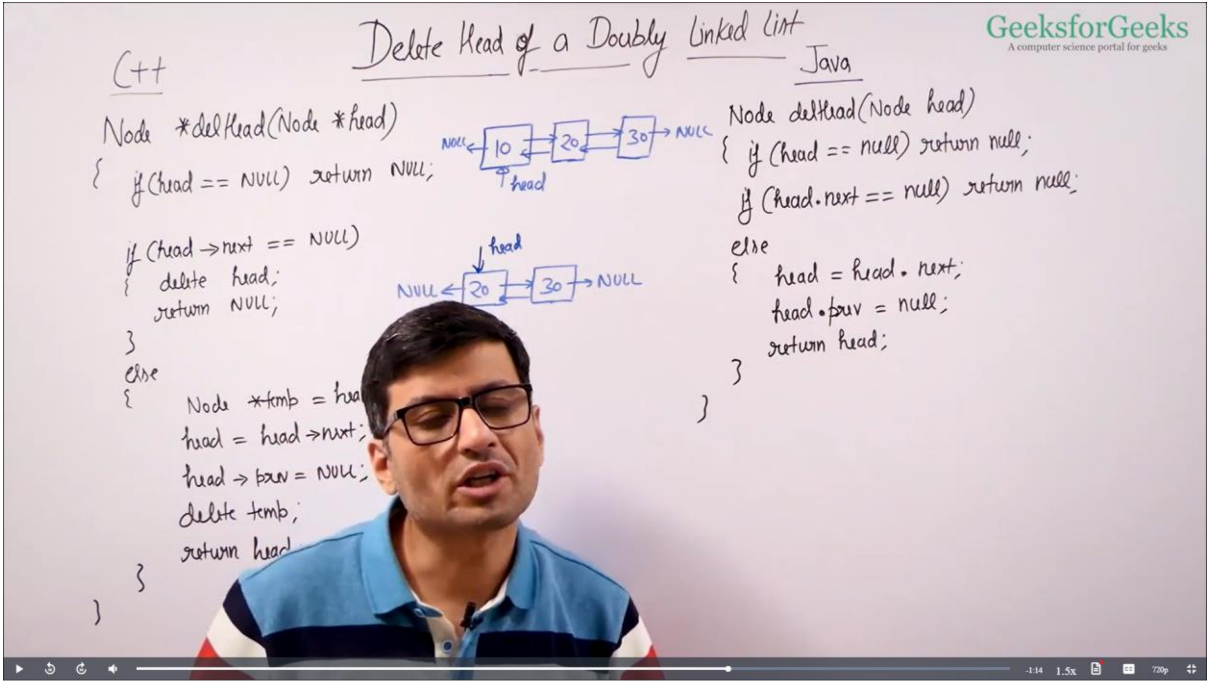
O/p :

I/p :

O/p : head = NULL or null

I/p : head = NULL or null

O/p : head = NULL or null



Delete Head of a Doubly Linked List

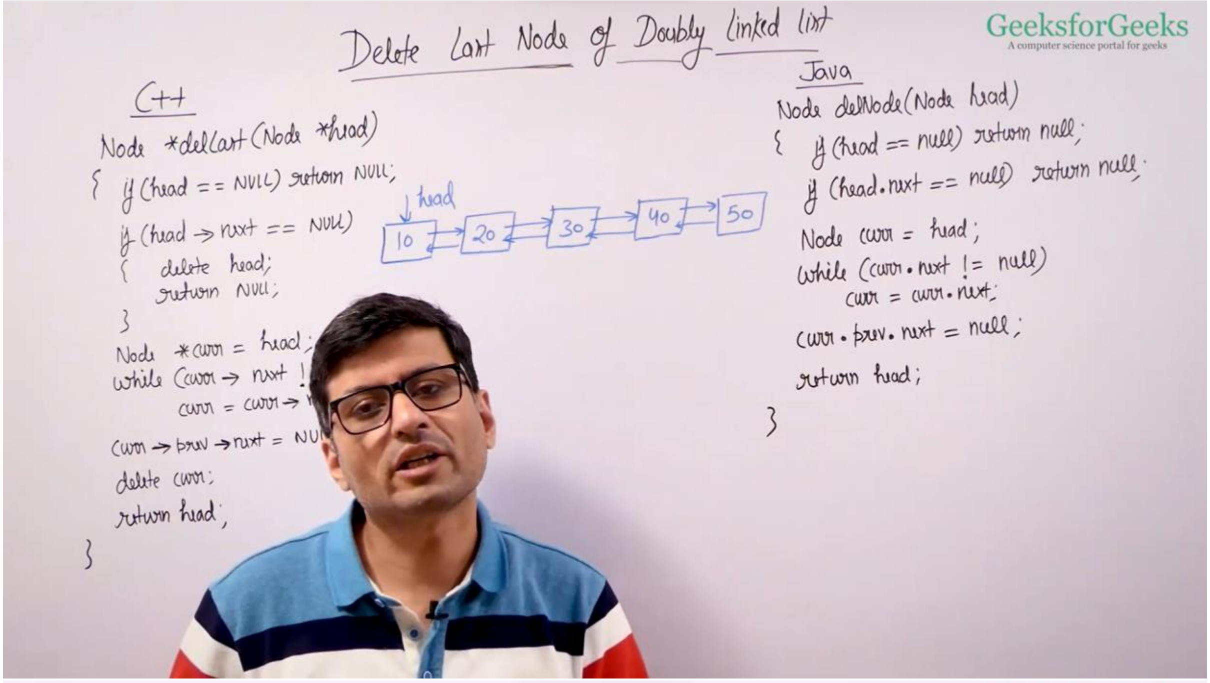
GeeksforGeeks
A computer science portal for geeks

C++

```
Node *delHead(Node *head)
{  if(head == NULL) return NULL;

   if (head->next == NULL)
   {  delete head;
      return NULL;
   }
   else
   {    Node *temp = hea
       head = head->next;
       head->prev = NULL;
       delete temp;
       return head
    }
}
```

Java

```
Node delHead(Node head)
{ if (head == null) return null;
  if (head.next == null) return null;
  else
  {  head = head. next;
     head.prev = null;
     return head;
  }
}
```



GeeksforGeeks
A computer science portal for geeks

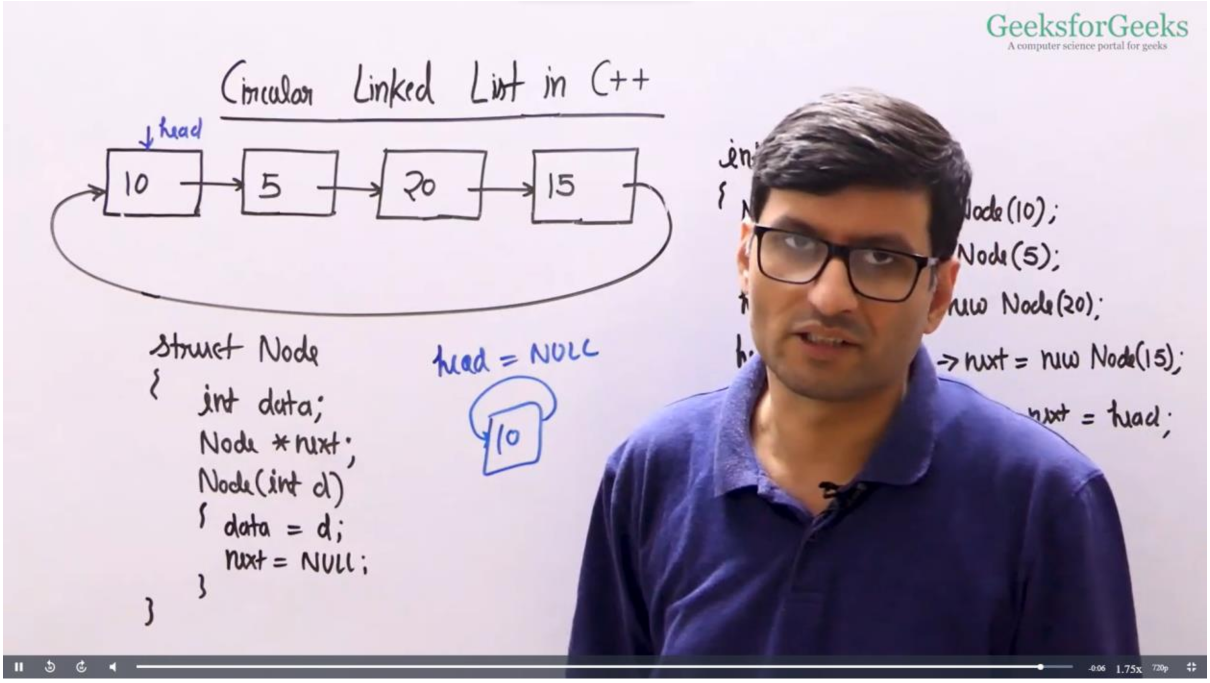
Delete Last Node of Doubly linked list

C++

```
Node *delLast(Node *head)
{ if(head == NULL) return NULL;
  if(head -> next == NULL)
  {  delete head;
     return NULL;
  }
  Node *curr = head;
  while (curr -> next !
        curr = curr -> 
  curr -> prev -> next = NU
  delete curr;
  return head;
}
```

Java

```
Node delNode(Node head)
{ if(head == null) return null;
  if (head.next == null) return null;
  Node curr = head;
  while (curr.next != null)
      curr = curr.next;
  curr.prev.next = null;
  return head;
}
```



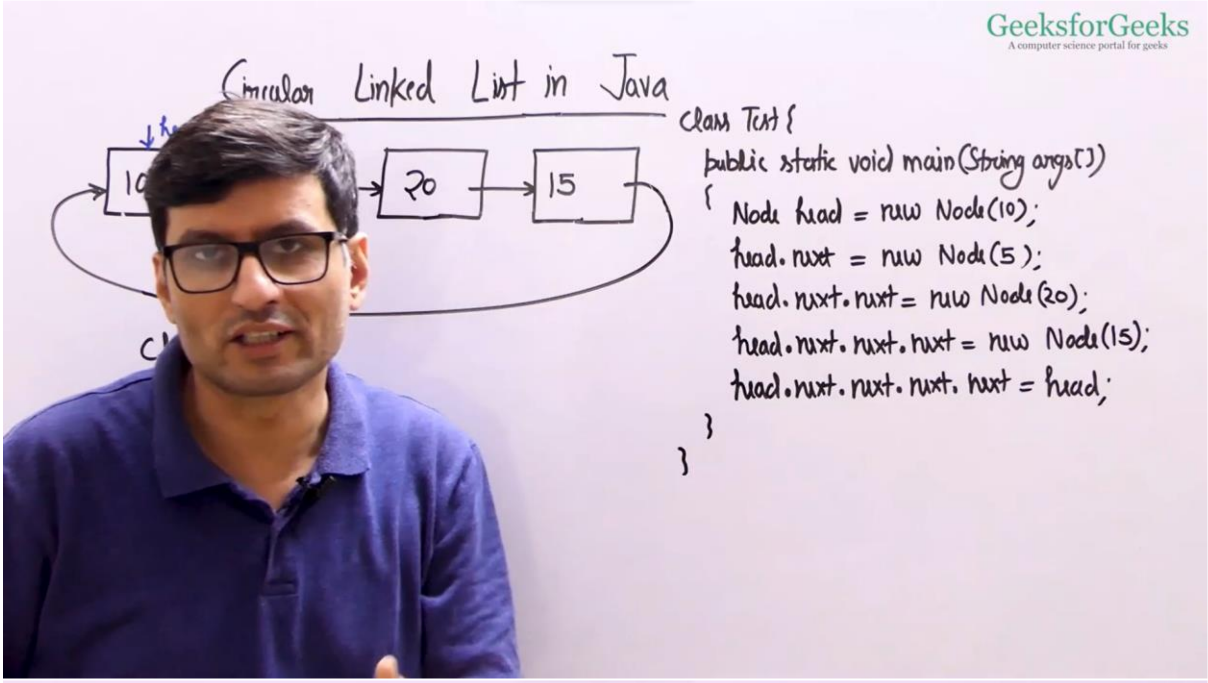
GeeksforGeeks
A computer science portal for geeks

Circular Linked List in C++

```
struct Node
{
   int data;
   Node *next;
   Node(int d)
   { data = d;
     next = NULL;
   }
}
```

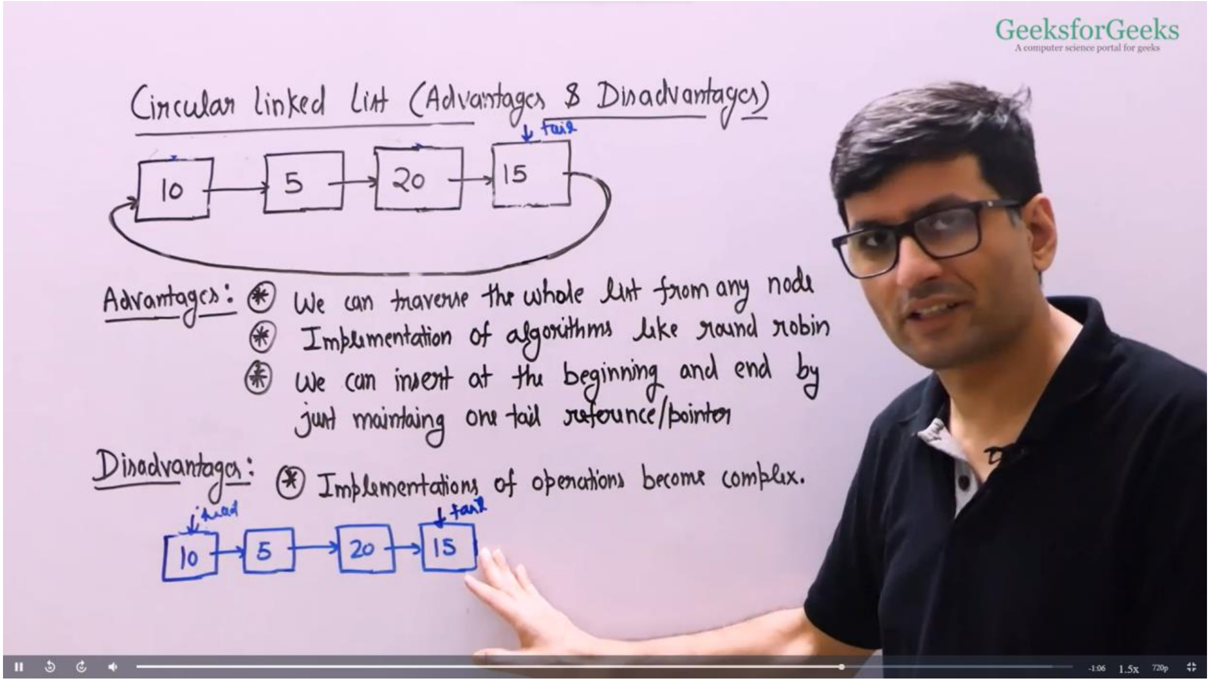
head = NULL

```
in
{ N              Node(10);
                 Node(5);
                 new Node(20);
  h         -> next = new Node(15);
                 next = head;
```



Circular Linked List in Java

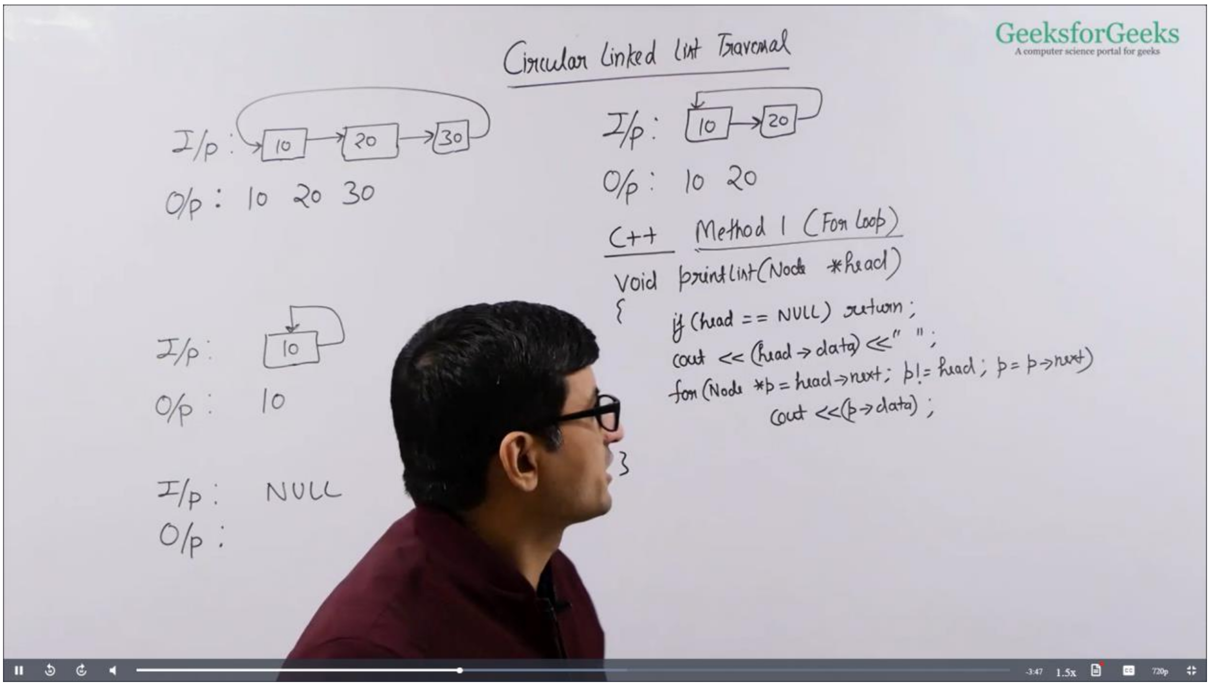
```
class Test {
  public static void main(String args[])
  {
    Node head = new Node(10);
    head.next = new Node(5);
    head.next.next = new Node(20);
    head.next.next.next = new Node(15);
    head.next.next.next.next = head;
  }
}
```



Circular Linked List (Advantages & Disadvantages)

Advantages:
- We can traverse the whole list from any node
- Implementation of algorithms like round robin
- We can insert at the beginning and end by just maintaing one tail reference/pointer

Disadvantages:
- Implementations of operations become complex.



Circular linked list Traversal

I/p:

O/p: 10 20 30

I/p:

O/p: 10

I/p: NULL

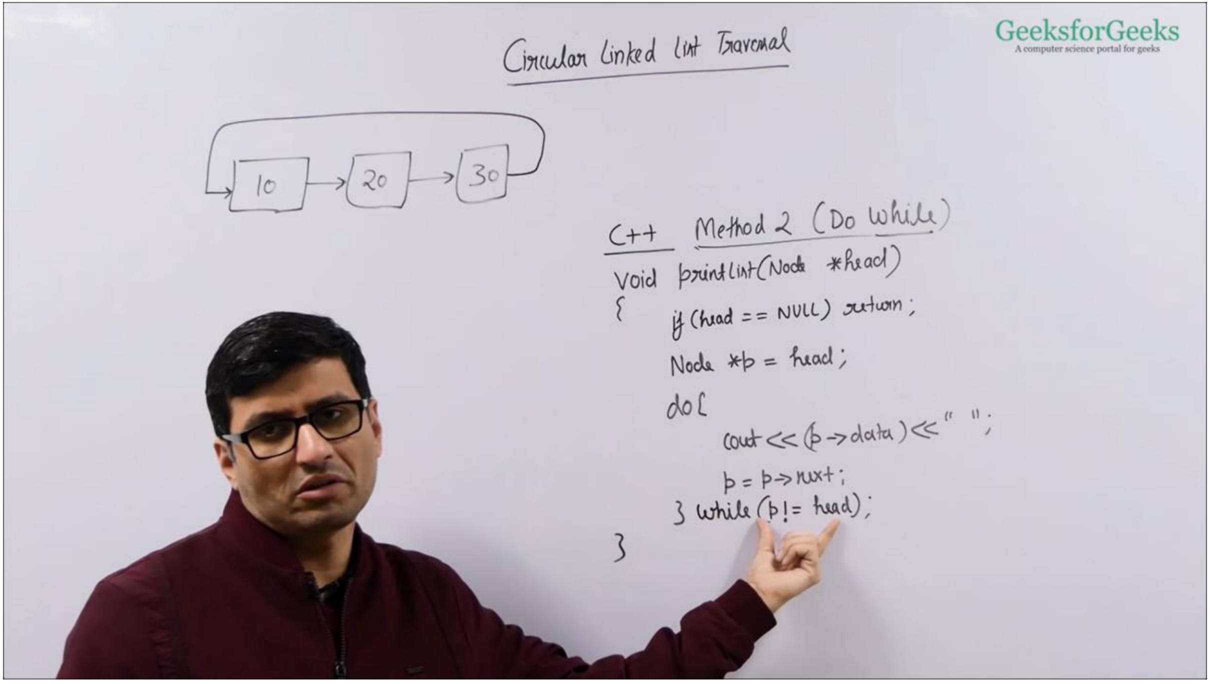
O/p:

I/p:

O/p: 10 20

C++ Method 1 (For Loop)

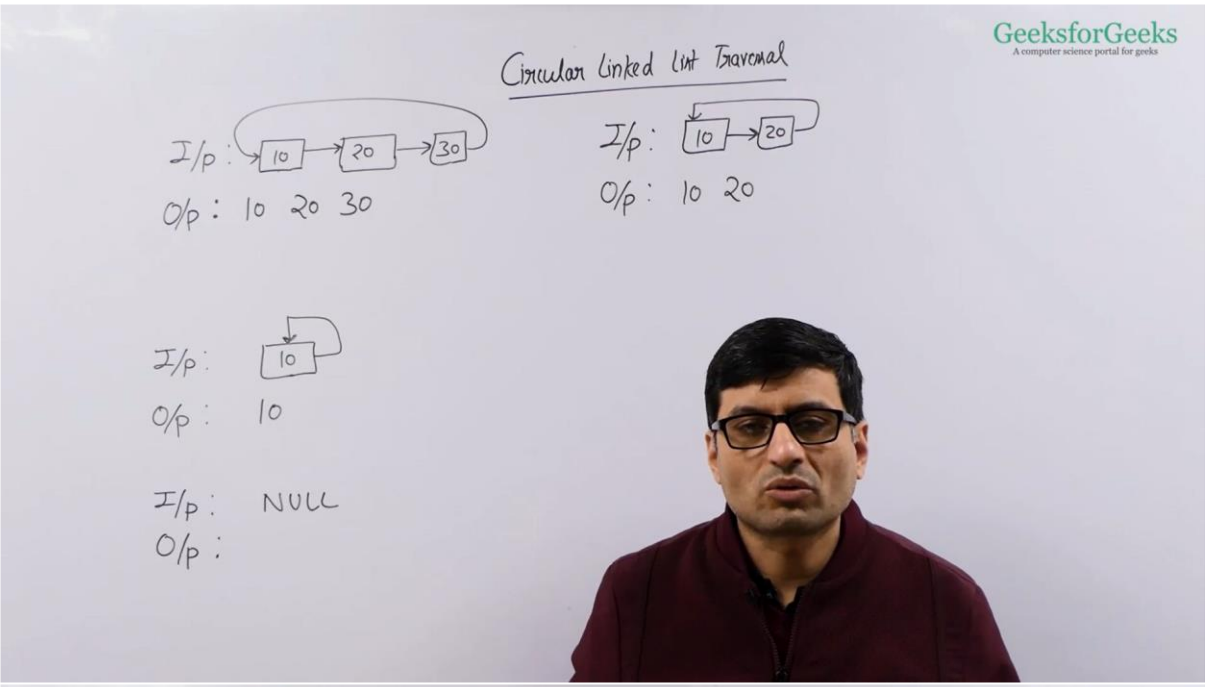
```
Void printList(Node *head)
{
    if (head == NULL) return;
    cout << (head -> data) << " ";
    for (Node *p = head->next; p!= head; p = p->next)
        cout <<(p->data);

}
```



Circular linked list Traversal

C++ Method 2 (Do while)

```
Void printList(Node *head)
{
    if (head == NULL) return;
    Node *p = head;
    do{
        cout << (p->data) << " ";
        p = p->next;
    } while (p!= head);
}
```



Circular Linked List Traversal

I/p :

O/p : 10 20 30

I/p :

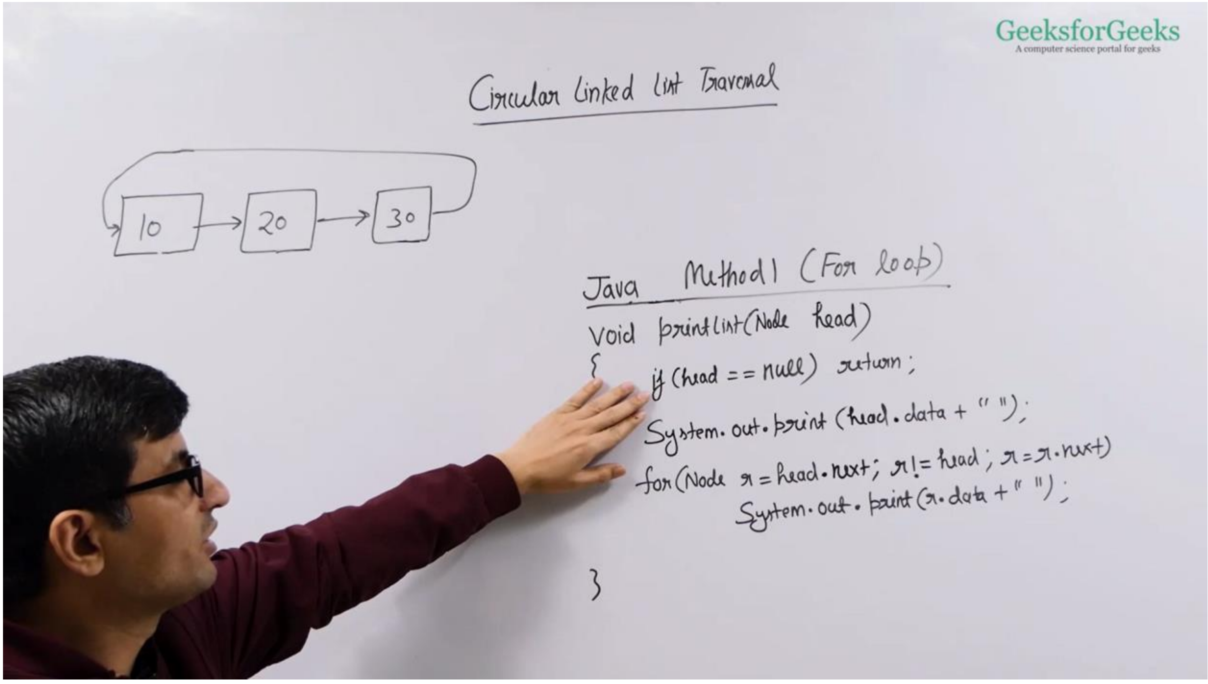
O/p : 10 20

I/p :

O/p : 10

I/p : NULL

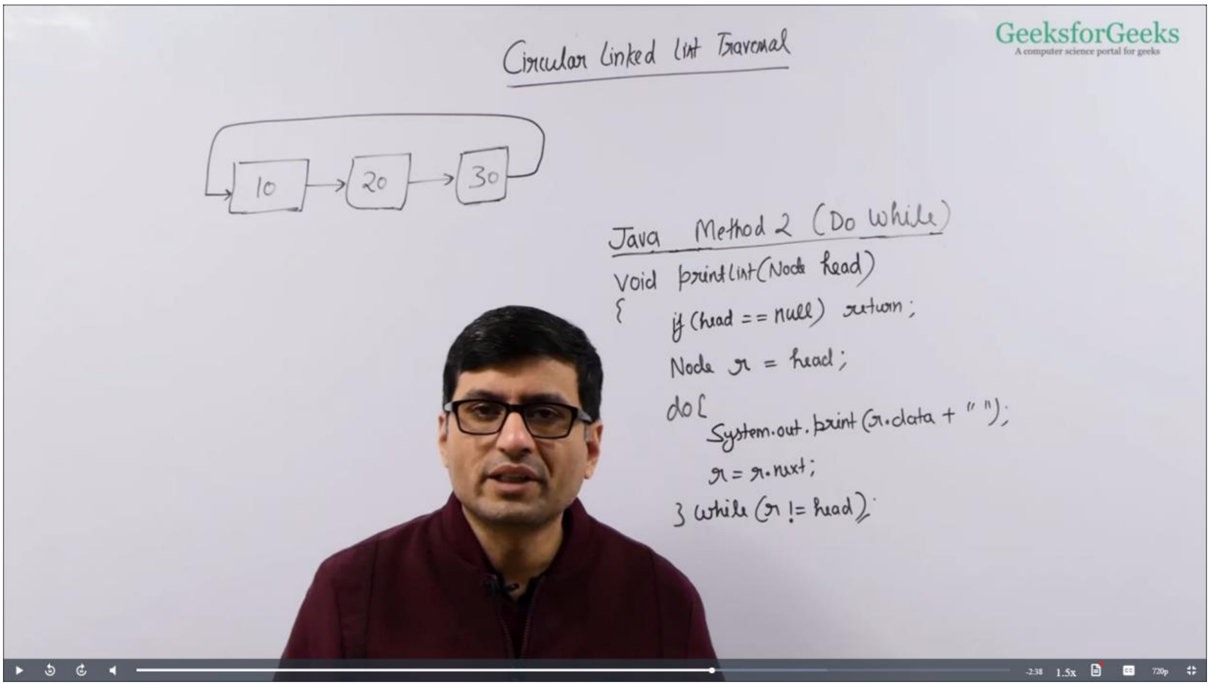
O/p :



Circular Linked List Traversal

Java Method1 (For loop)

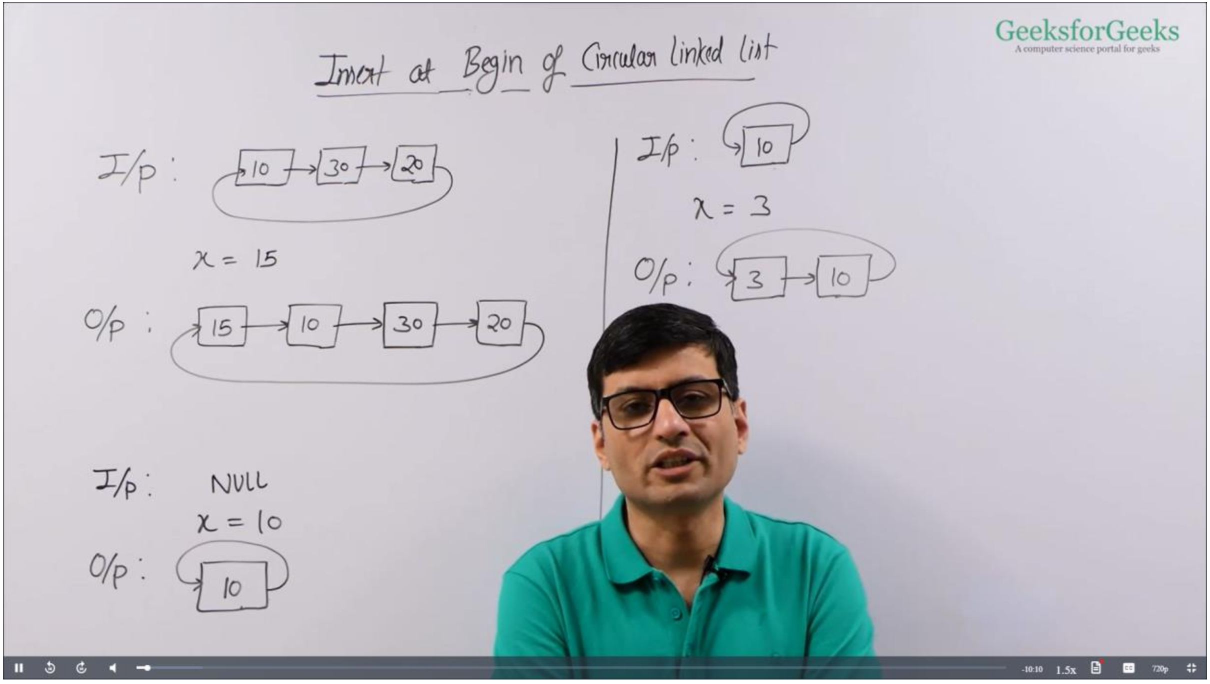
```
void printList(Node head)
{     if (head == null) return;
     System.out.print(head.data + " ");
    for(Node r = head.next; r != head; r = r.next)
           System.out.print(r.data + " ");


}
```



Circular linked list Traversal

Java Method 2 (Do While)

```
Void printList(Node head)
{
     if (head == null) return;
     Node r = head;
     do{
        System.out.print(r.data + " ");
        r = r.next;
     } while (r != head);
```



Insert at Begin of Circular linked list

I/p :

x = 15

O/p :

I/p : NULL

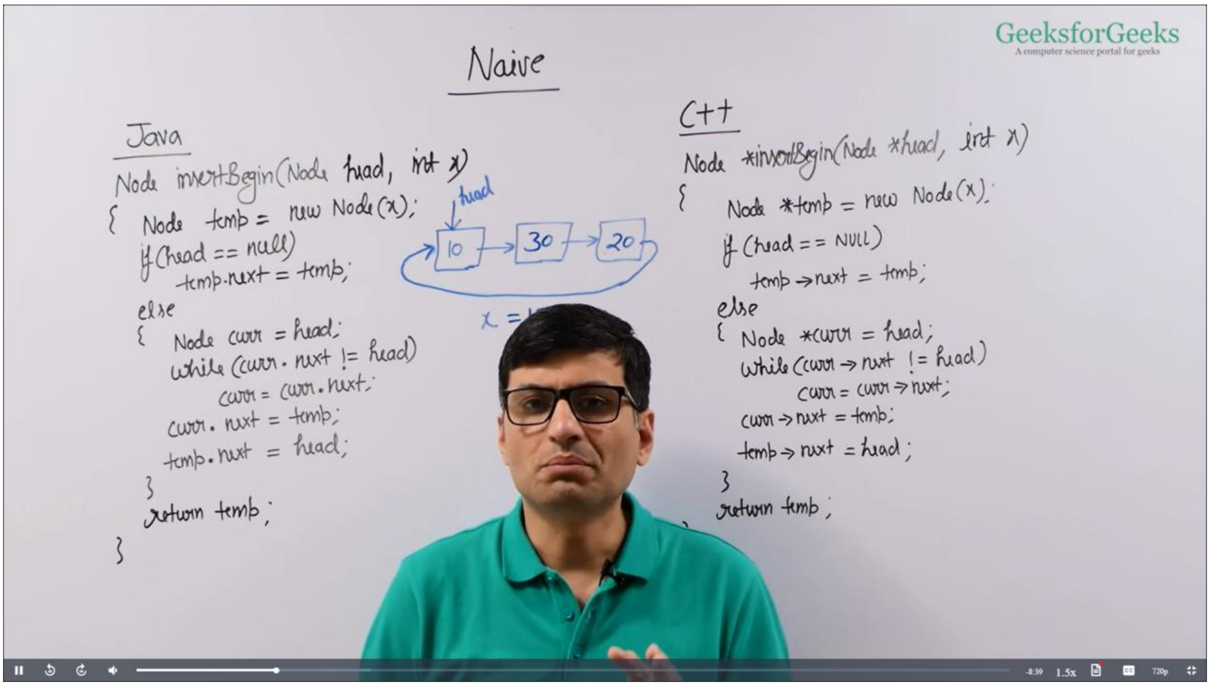
x = 10

O/p :

I/p :

x = 3

O/p :



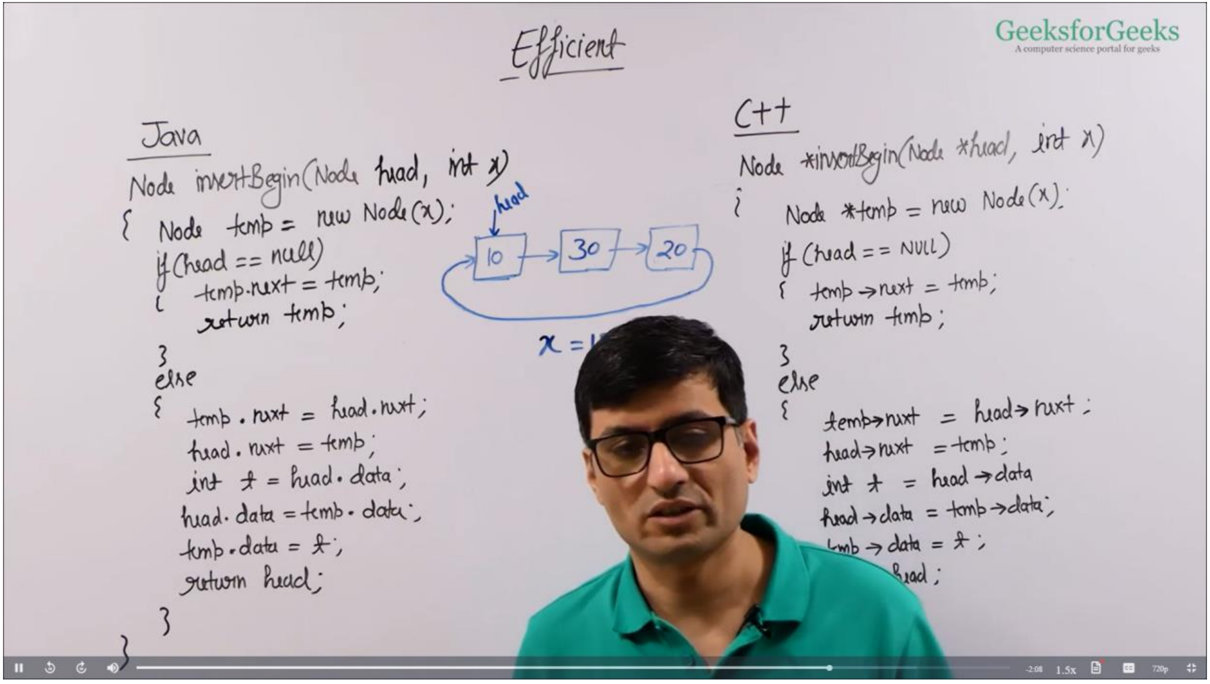
Naive

Java

```
Node insertBegin(Node head, int x)
{ Node temp = new Node(x);
  if(head == null)
     temp.next = temp;
  else
  { Node curr = head;
    while (curr.next != head)
         curr = curr.next;
    curr.next = temp;
    temp.next = head;
  }
  return temp;
}
```

C++

```
Node *insertBegin(Node *head, int x)
{   Node *temp = new Node(x);
    if (head == NULL)
       temp->next = temp;
    else
    { Node *curr = head;
      while (curr->next != head)
           curr = curr->next;
      curr->next = temp;
      temp->next = head;
    }
    return temp;
```



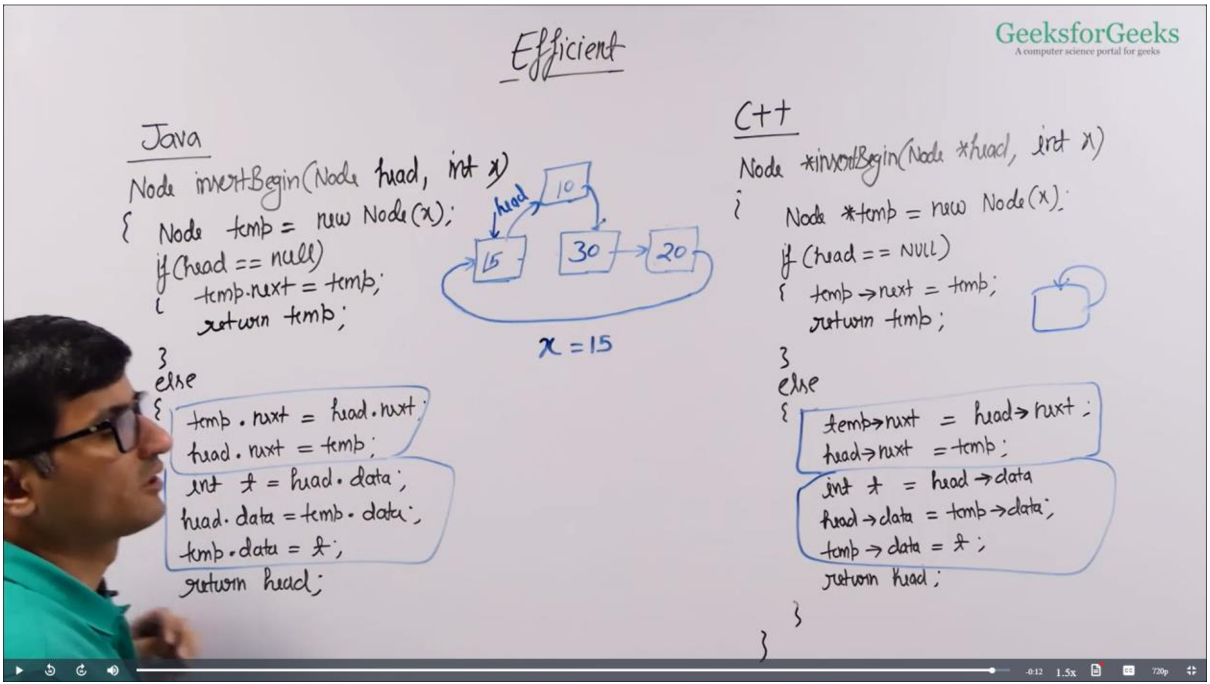
Efficient

Java

```
Node insertBegin(Node head, int x)
{ Node temp = new Node(x);
  if(head == null)
  {  temp.next = temp;
     return temp;
  }
  else
  {
     temp . next = head.next;
     head . next = temp;
     int t = head . data;
    head . data = temp . data;
    temp . data = t;
    return head;
  }
}
```

C++

```
Node *insertBegin(Node *head, int x)
{   Node *temp = new Node(x);
    if (head == NULL)
    { temp->next = temp;
      return temp;
    }
    else
    {
       temp->next  = head->next;
       head->next  = temp;
       int t  = head->data
       head->data = temp->data;
       temp->data = t;
       head;
```



Efficient

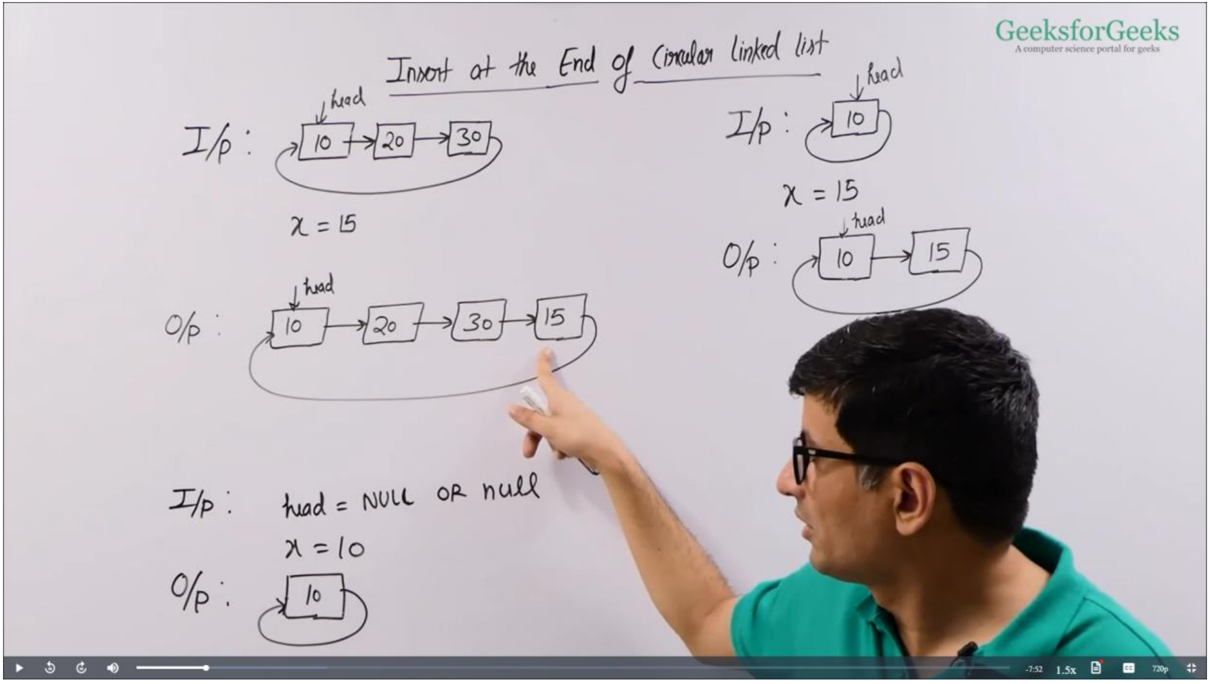
Java

```
Node insertBegin(Node head, int x)
{ Node temp = new Node(x);
  if(head == null)
  {  temp.next = temp;
     return temp;
  }
  else
  {
    temp . next = head . next;
    head . next = temp;
    int t = head . data;
    head . data = temp . data;
    temp . data = t;
    return head;
```

x = 15

C++

```
Node *insertBegin(Node *head, int x)
{
  Node *temp = new Node(x);
  if (head == NULL)
  { temp -> next = temp;
    return temp;
  }
  else
  {
    temp->next = head->next;
    head->next = temp;
    int t = head -> data
    head -> data = temp -> data;
    temp -> data = t;
    return head;
  }
}
```



Insert at the End of Circular linked list

I/p : head → 10 → 20 → 30

x = 15

O/p : head → 10 → 20 → 30 → 15

I/p : head = NULL OR null

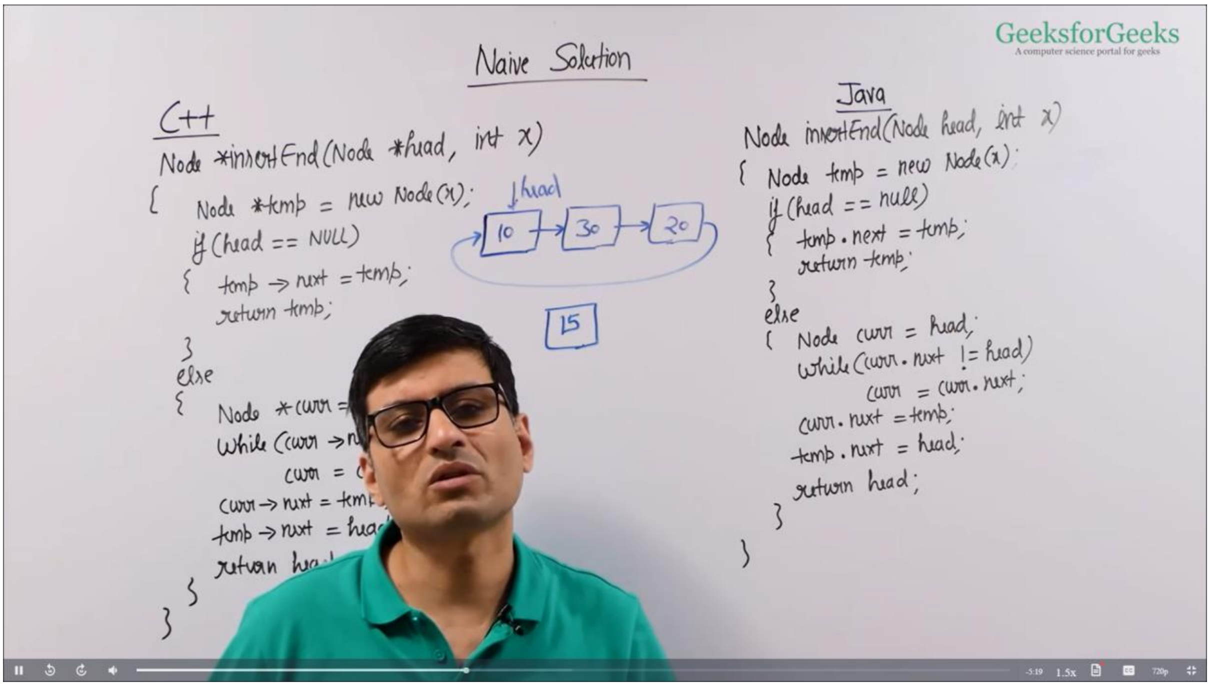
x = 10

O/p : 10

I/p : head → 10

x = 15

O/p : head → 10 → 15



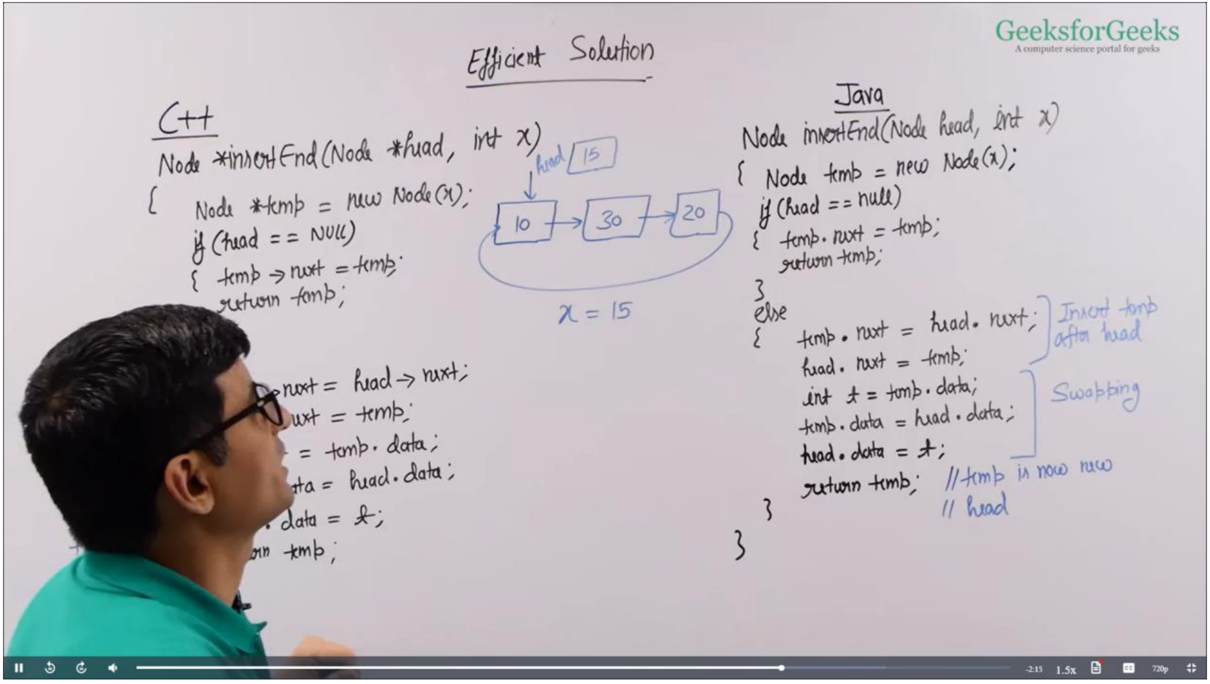
Naive Solution

C++

```
Node *insertEnd(Node *head, int x)
{   Node *temp = new Node(x);
    if(head == NULL)
    { temp -> next = temp;
      return temp;
    }
    else
    {   Node *curr =
        while(curr -> n
              curr = c
        curr -> next = temp
        temp -> next = hea
        return hea
    }
}
```

Java

```
Node insertEnd(Node head, int x)
{ Node temp = new Node(x);
  if(head == null)
  { temp.next = temp;
    return temp;
  }
  else
  ( Node curr = head;
    while(curr.next != head)
            curr = curr.next;
    curr.next = temp;
    temp.next = head;
    return head;
  }
}
```



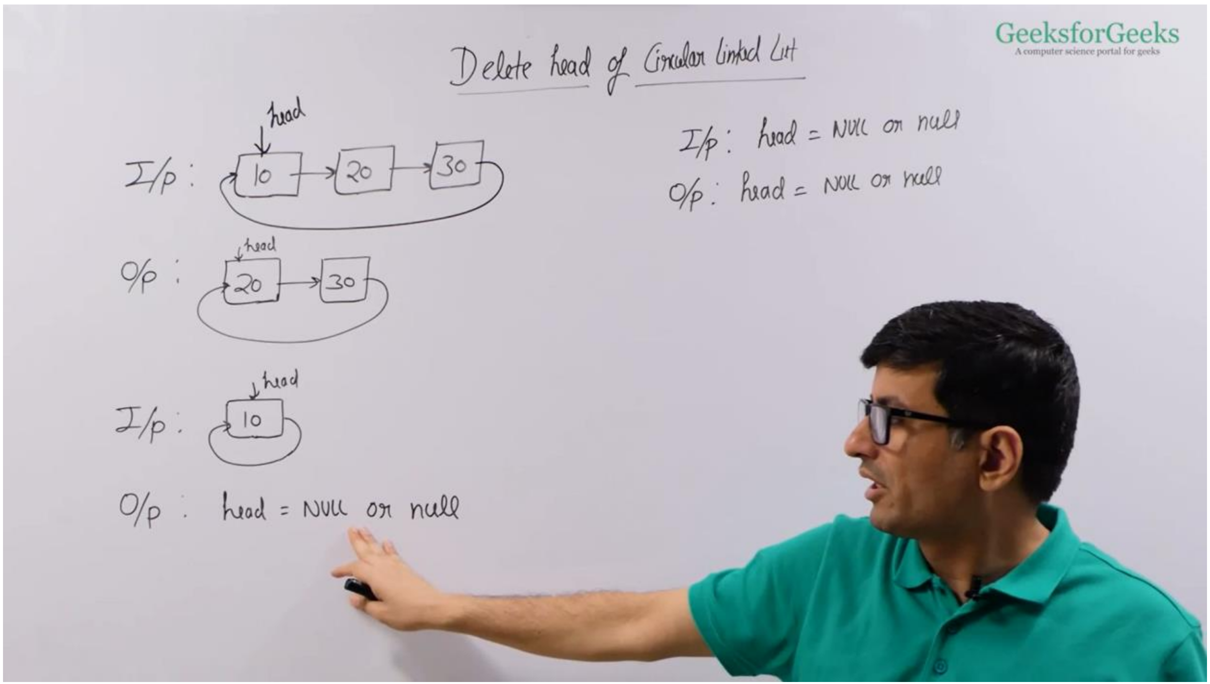
Efficient Solution

C++

```
Node *insertEnd(Node *head, int x)
{   Node *temp = new Node(x);
    if(head == NULL)
    { temp -> next = temp;
      return temp;

        next = head -> next;
        next = temp;
          = temp. data;
        ta = head. data;
        data = t;
        rn temp;
```

Java

```
Node insertEnd(Node head, int x)
{ Node temp = new Node(x);
  if(head == null)
  { temp.next = temp;
    return temp;
  }
  else
  {   temp.next = head.next;   ] Insert temp
      head.next = temp;        ] after head
      int t = temp.data;       ]
      temp.data = head.data;   ] Swapping
      head.data = t;           ]
      return temp;  // temp is now new
                    // head
  }
}
```

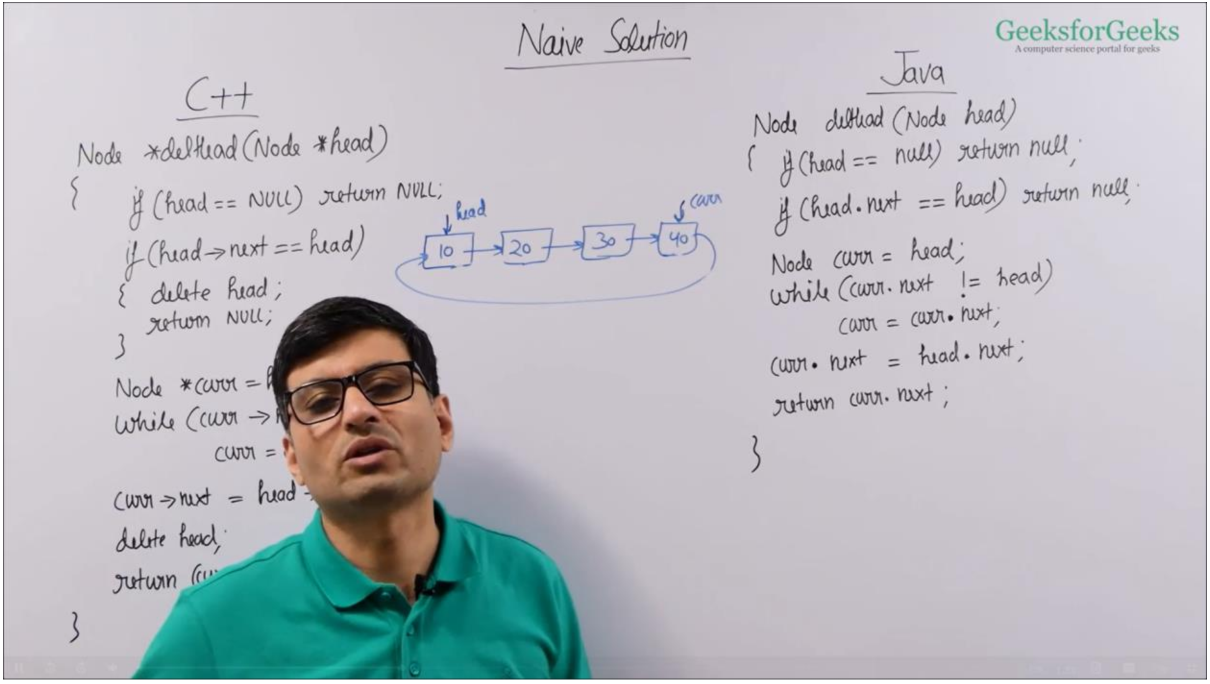


Delete head of Circular linked List

I/p : head = NULL or null

O/p : head = NULL or null

I/p :
head
10
20
30
O/p :
head
20
30
I/p :
head
10
O/p : head = NULL or null
GeeksforGeeks
A computer science portal for geeks



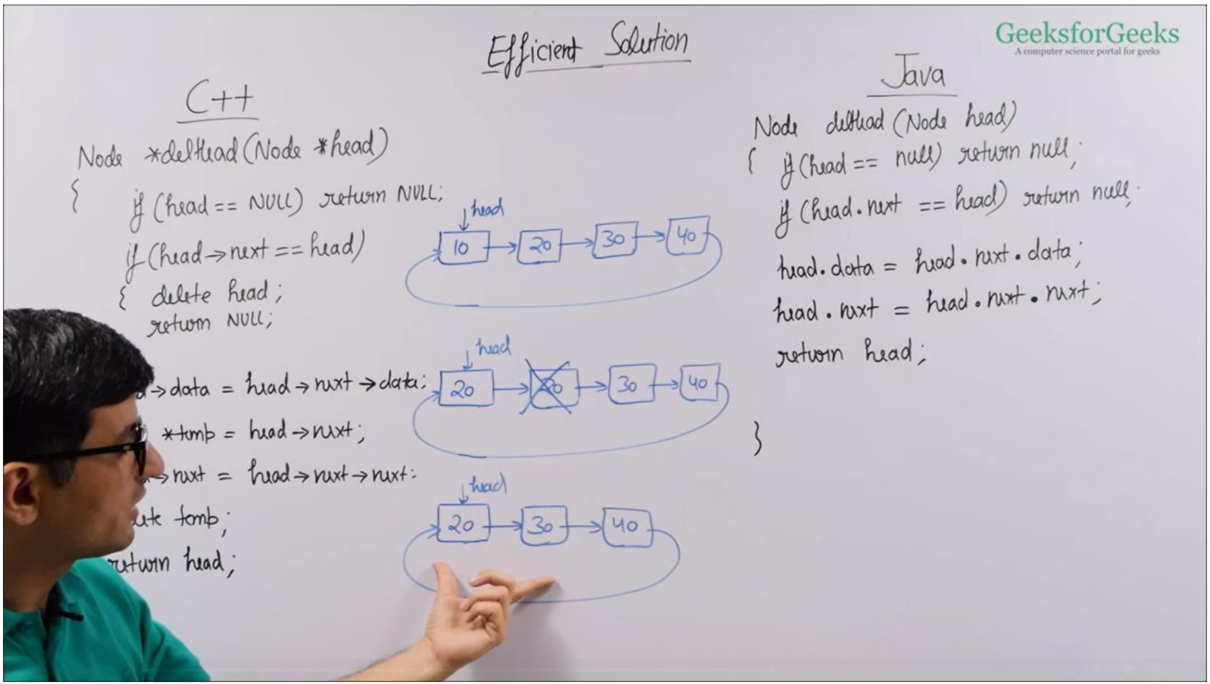
Naive Solution

C++

```
Node *delHead(Node *head)
{   if (head == NULL) return NULL;
    if (head->next == head)
    { delete head;
      return NULL;
    }
    Node *curr = h
    while (curr -> n
          curr =
    curr->next = head -
    delete head;
    return (cu
}
```

Java

```
Node delHead (Node head)
{ if (head == null) return null;
  if (head.next == head) return null;
  Node curr = head;
  while (curr.next != head)
        curr = curr.next;
  curr.next = head.next;
  return curr.next;
}
```



GeeksforGeeks
A computer science portal for geeks

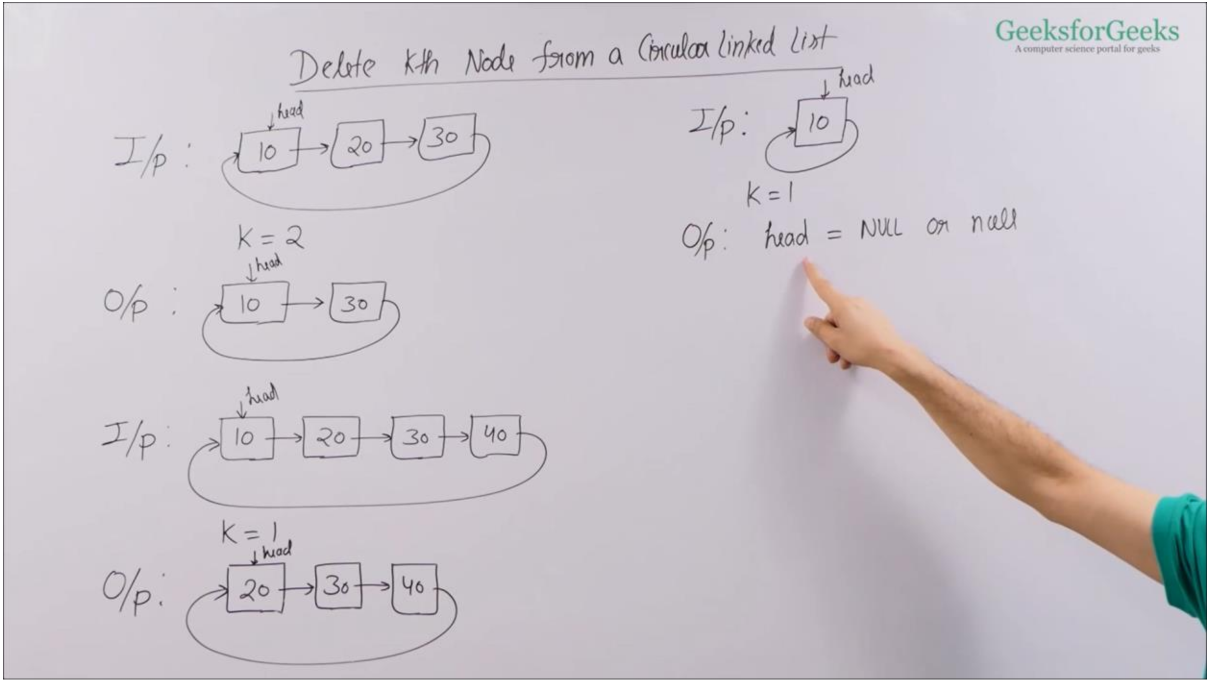
Efficient Solution

C++

```
Node *delHead(Node *head)
{    if (head == NULL) return NULL;
     if (head->next == head)
     {  delete head;
        return NULL;

       ->data = head->next->data;
        *temp = head->next;
        ->next = head->next->next;
       ete temp;
     return head;
```

Java

```
Node delHead(Node head)
{  if(head == null) return null;
   if(head.next == head) return null;
   head.data = head.next.data;
   head.next = head.next.next;
   return head;

}
```



GeeksforGeeks
A computer science portal for geeks

Delete Kth Node from a Circular Linked List

I/p :

K = 2

O/p :

I/p :

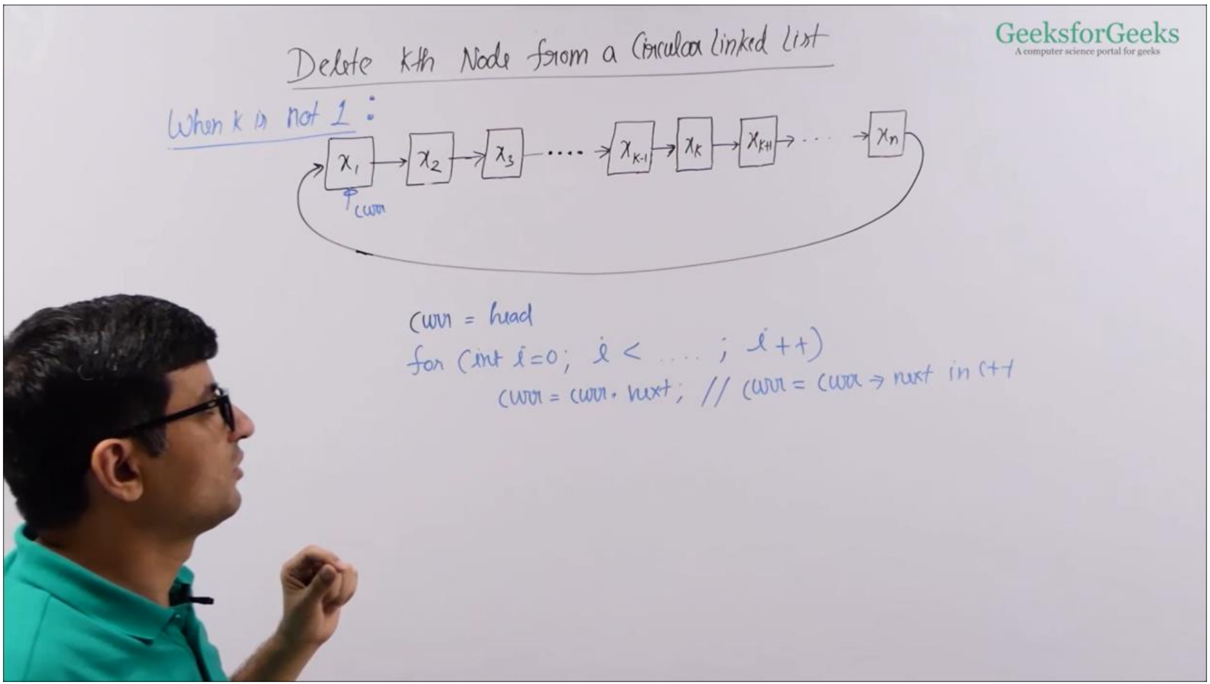
K = 1

O/p :

I/p :

K = 1

O/p : head = NULL or null

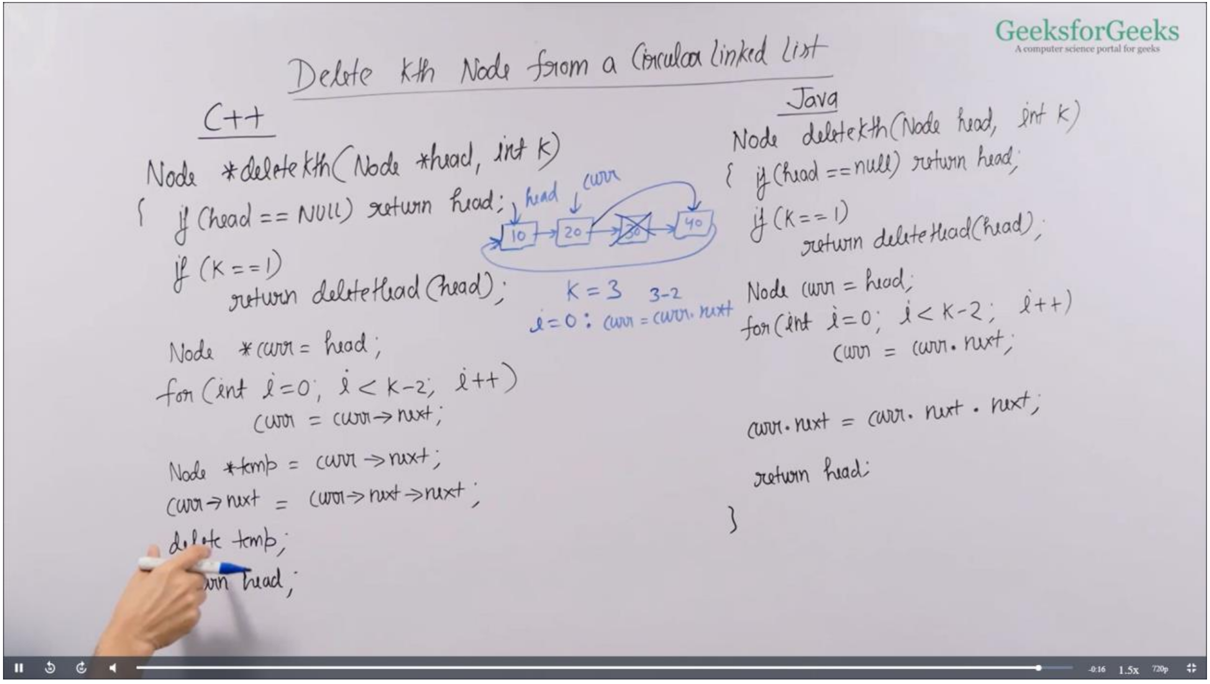


GeeksforGeeks
A computer science portal for geeks

Delete kth Node from a Circular linked List

When k is not 1 :

```
curr = head
for (int i=0;  i < .... ;  i++)
      curr = curr. next;  // curr = curr → next in C++
```



GeeksforGeeks
A computer science portal for geeks

Delete kth Node from a Circular linked List

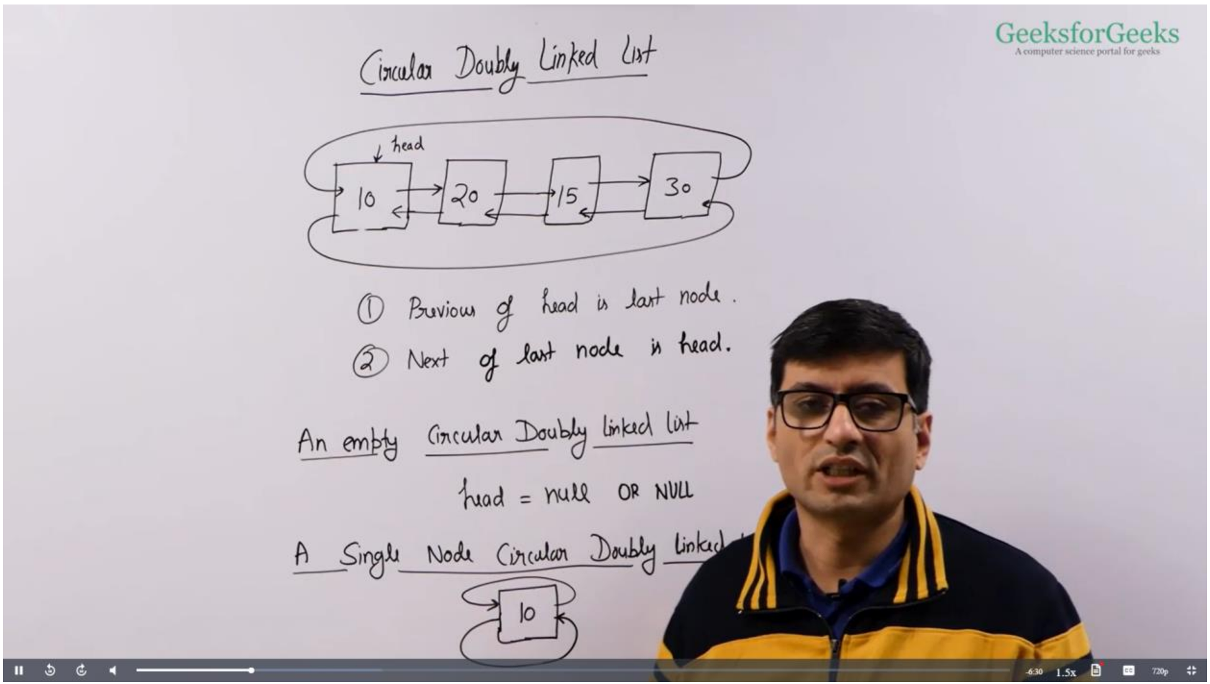
C++

```
Node *deletekth( Node *head, int k)
{  if (head == NULL) return head;
   if (K==1)
       return deleteHead(head);

   Node *curr = head;
  for (int i=0; i < k-2; i++)
        curr = curr->next;

   Node *temp = curr->next;
   curr->next = curr->next->next;
   delete temp;
   urn head;
```

K = 3   3-2
i=0 : curr = curr. next

Java

```
Node deletekth(Node head, int k)
{  if (head == null) return head;
   if (K==1)
       return deleteHead(head);
   Node curr = head;
   for(int i=0;  i< K-2;  i++)
          curr = curr. next;


   curr. next = curr. next . next;

   return head;
}
```



Circular Doubly Linked List

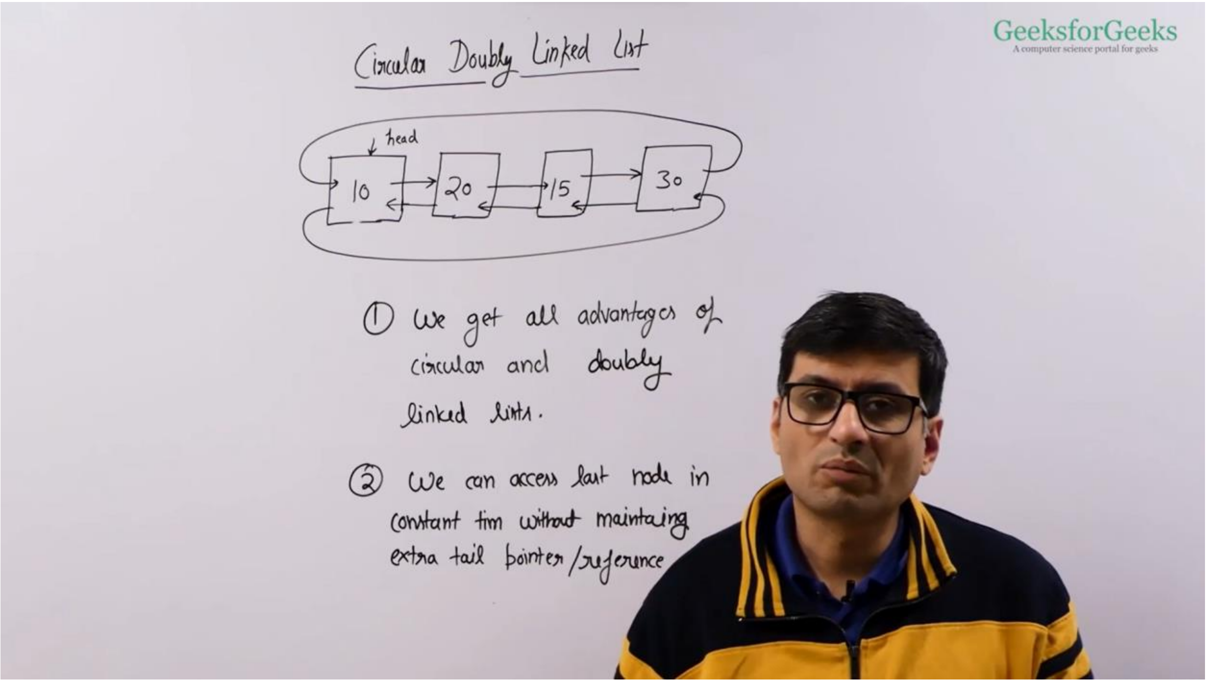
① Previous of head is last node.

② Next of last node is head.

An empty Circular Doubly linked list

head = null OR NULL

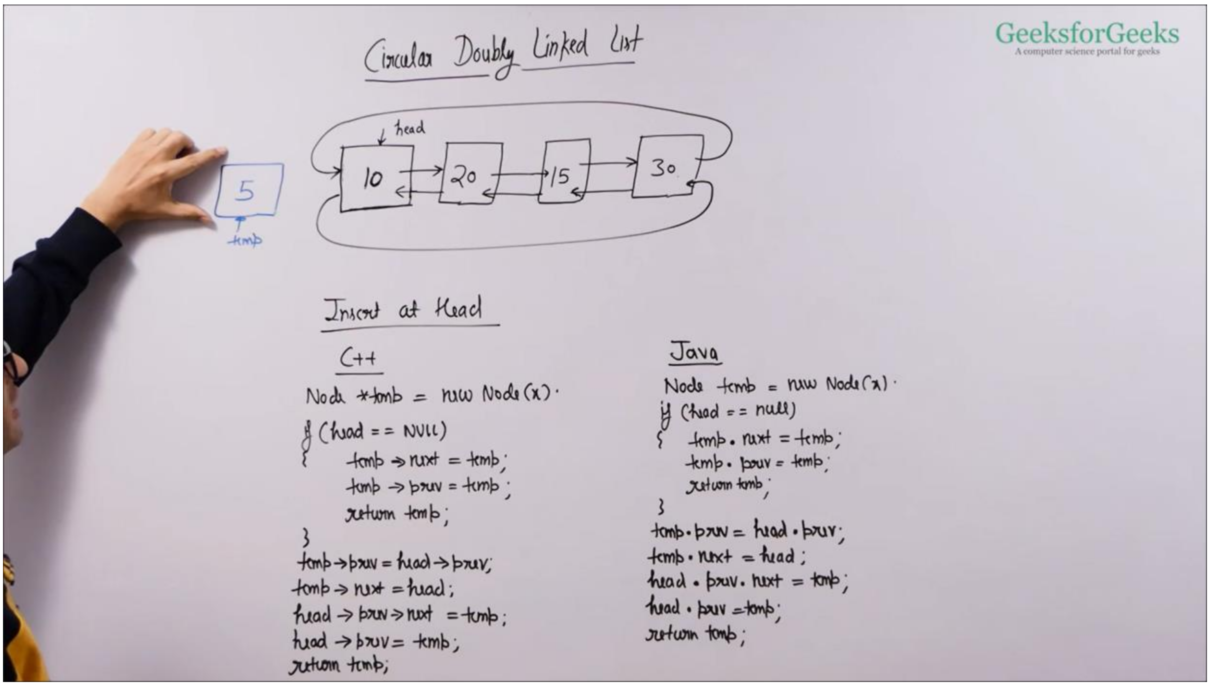
A Single Node Circular Doubly linked l



Circular Doubly Linked List

① We get all advantages of circular and doubly linked lists.

② We can access last node in constant tim without maintaing extra tail pointer/reference



Circular Doubly Linked List

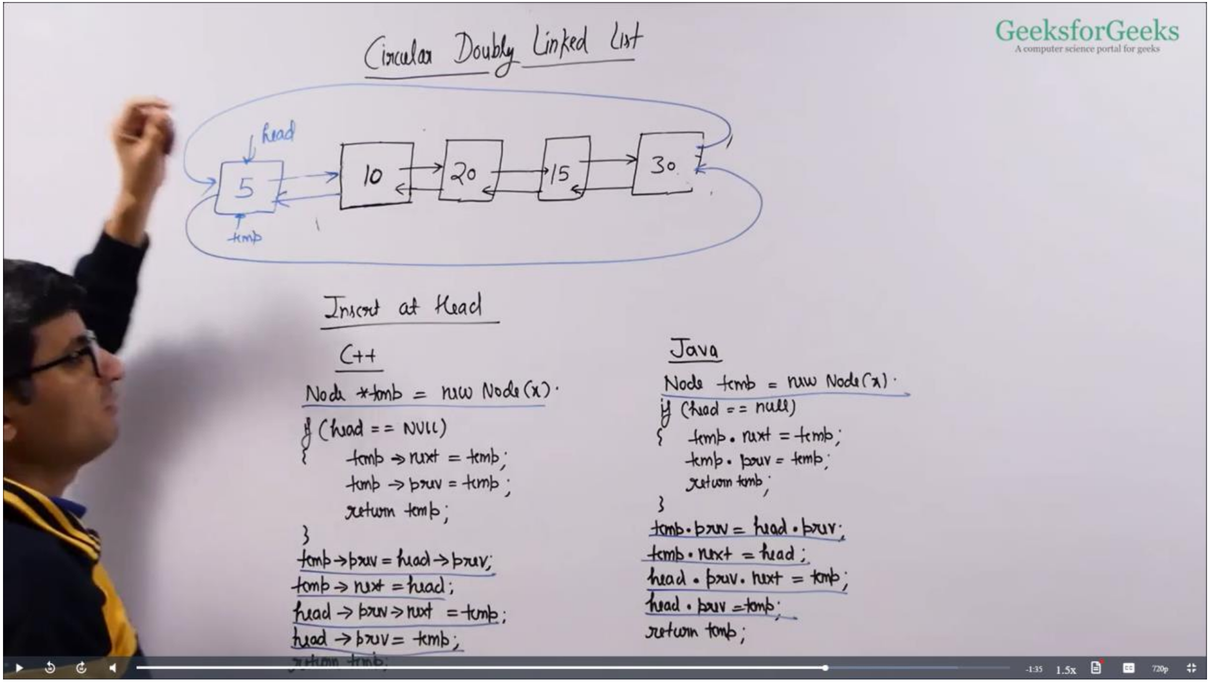
Insert at Head

C++

```
Node *temp = new Node(x).
if (head == NULL)
{
    temp -> next = temp;
    temp -> prev = temp;
    return temp;
}
temp->prev = head->prev;
temp -> next = head;
head -> prev -> next = temp;
head -> prev = temp;
return temp;
```

Java

```
Node temp = new Node(x).
if (head == null)
{
   temp. next = temp;
   temp. prev = temp;
   return temp;
}
temp.prev = head.prev;
temp.next = head;
head.prev.next = temp;
head.prev = temp;
return temp;
```



Circular Doubly Linked List

Insert at Head

C++

Node *temp = new Node(x).
if (head == NULL)
{
    temp -> next = temp;
    temp -> prev = temp;
    return temp;
}
temp->prev = head->prev;
temp -> next = head;
head -> prev -> next = temp;
head -> prev = temp;
return temp;

Java

Node temp = new Node(x).
if (head == null)
{
   temp. next = temp;
   temp. prev = temp;
   return temp;
}
temp.prev = head.prev;
temp.next = head;
head.prev.next = temp;
head.prev = temp;
return temp;