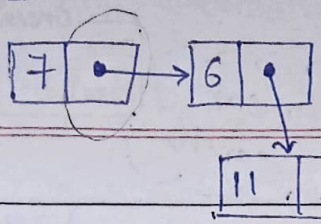
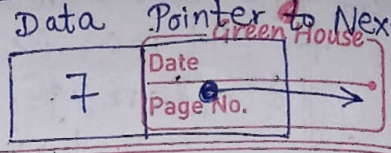
Lec 13 Introduction to Linked List in Data Structures:

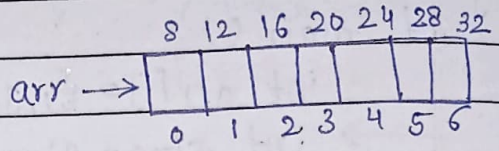




Node

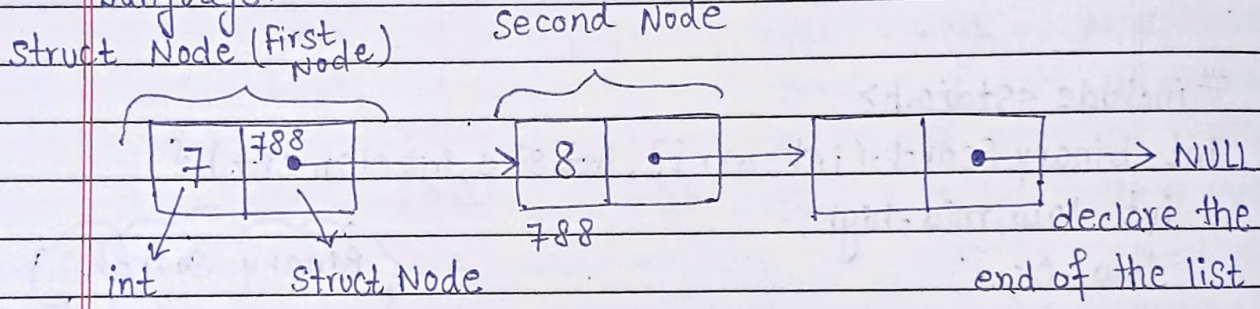
```
Struct Node {
    int data;
    Struct Node * next;
};
```

↓ Self refrencing Structure



(8)+0×4
8+1×4
(8)+n×4 =

Lec-14 Linked list Data Structure: Creation and Traversal in C Language:



head = (struct Node*) malloc (sizeof (StructNode))

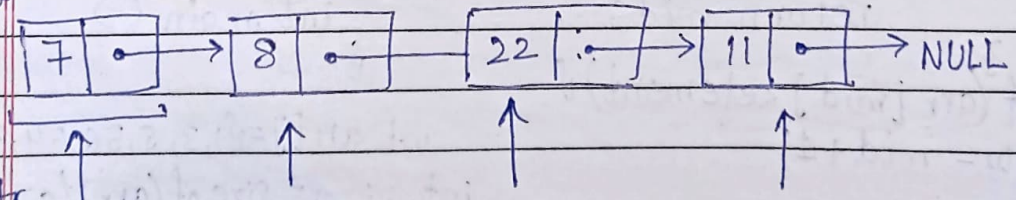
↓

Struct pointer

head → data = 7

head → next = Second Node

O(n) Traversal of linked list



```
#include <stdio.h>
#include <stdlib.h>
struct Node.
{ int data;
  struct Node *next;
};
void linkedlistTraversal(struct Node *ptr)
{   while (ptr != NULL)
    {
      printf("Element: %d\n", ptr->data);
      ptr = ptr->next;
    }
}
int main ()
{
    struct Node *head;
    struct Node *second;
    struct Node *third;
// Allocate memory for nodes in the linked list in Heap
    head = (struct Node*) malloc(sizeof(struct Node));
    second = (struct Node*) malloc(sizeof(struct Node));
    third = (struct Node*) malloc(sizeof(struct Node));
// Link first and second Nodes
    head->data = 7;
    head->next = second;
// Link Second and third Nodes
    second->data = 11;
    second->next = third;
// Terminate the list at the third node
    third->data = 66;
    third->next = NULL;

    linkedlistTraversal(head);
    return 0;
}
```

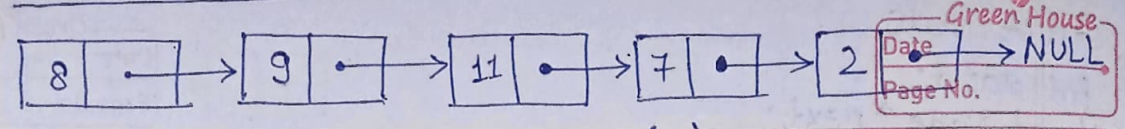
Code For Traversal

- Declaration

Initialization.

```
Element: 7
Element: 11
Element: 66
```

Lec-15 Insertion of a Node in a Linked List Data Structure:

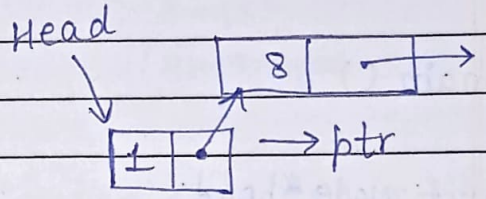


Green House
Date
Page No.

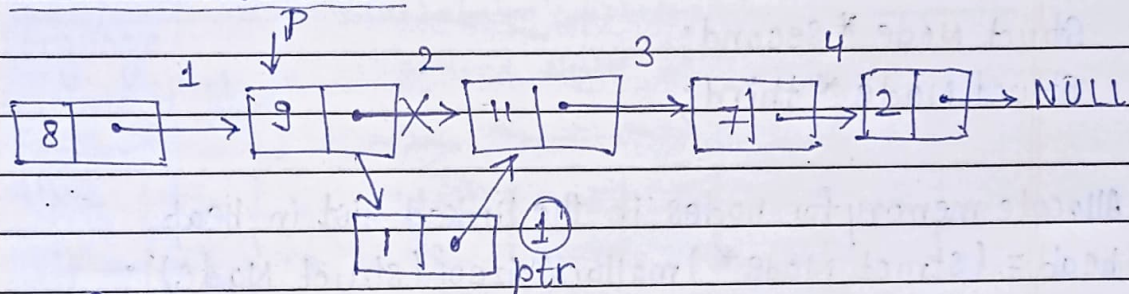
→ Case 1: Insert at the beginning ~ O(1)
→ Case 2: Insert in between ~ O(n) → WC
→ Case 3: Insert at the end ~ O(n)
→ Case 4: Insert after a Node ~ O(1)

Case: 1

```
struct Node *ptr = (struct Node*) malloc (sizeof(struct Node
ptr → next = head;
head = ptr;
return head;
```

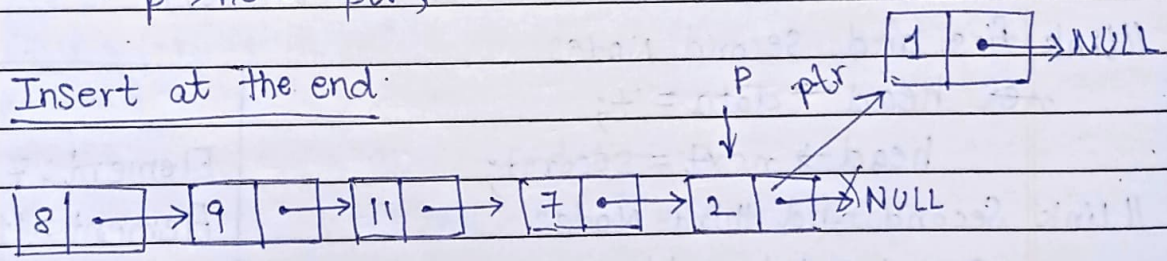


Case: 2 Insert inbetween



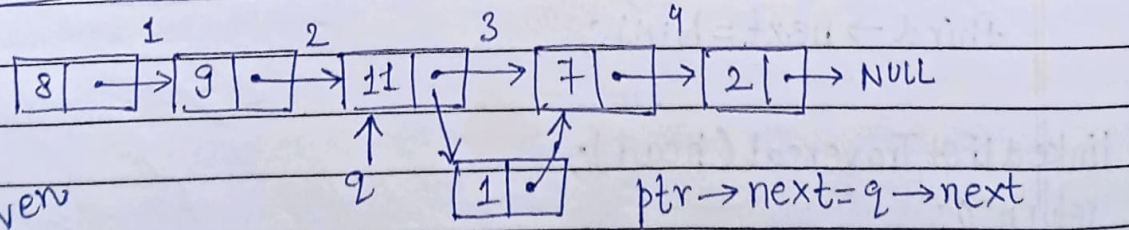
① ptr → next = p → next;
p → next = ptr;

Case: 3. Insert at the end



p → next = ptr
ptr → next = NULL;

Case: 4. Insert after a Node:



q → Given

ptr → next = q → next
q → next = ptr

Lec-16. Insertion in a Linked List in C Language:

Case: 1

Green House
Date
Page No.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};

void linkedListTraversal(struct Node *ptr){
    while(ptr != NULL)
    {
        printf("Element: %d \n", ptr->data);
        ptr = ptr->next;
    }
}

struct Node * insertAtfirst(struct Node *head, int data){
    struct Node *ptr = (struct Node*) malloc(sizeof(struct Node));
    ptr->next = head;
    pt->data = data;
    return ptr;
}

int main(){
    struct Node *head;
    struct Node *second;
    head = (struct Node*) malloc(sizeof(struct Node));
    second = (struct Node*) malloc(sizeof(struct Node));
    head->data = 7;
    head->next = second;
    second->data = 9;
    second->next = NULL;
    linkedlistTraversal(head);
    head = insertAtfirst(head, 56);
    linkedlistTraversal(head);
    return 0;
}
```

```
Element: 7
Element: 9

Element: 56
Element: 7
Element: 9
```

Case:2. Insert InBetween

```
struct Node *insertAtIndex(struct Node *head, int data,
                           int index){
struct Node *ptr=(struct Node*)malloc(sizeof(struct Node));
struct Node *p=head;
int i=0;
while(i!=index-1)
{
  p=p->next;
  i++;
}
ptr->data=data;
ptr->next=p->next;
p->next=ptr;
return head;
}

// Function Call
head=insertAtIndex(head,56,1);
```

Case:3. Insert at End:

```
struct Node *insertAtEnd(struct Node *head, int data)
{
struct Node *ptr=(struct Node*)malloc(sizeof(struct Node));
  ptr->data=data;
  struct Node *p=head;
while(p->next!=NULL)
{
  p=p->next;
}
  p->next=ptr;
ptr->next=NULL;
return head;
}

// Function Call
head=insertAtEnd(head,56);
```

Case: 4 Insert After a Node: special Case O(1)

Green House
Date
Page No.

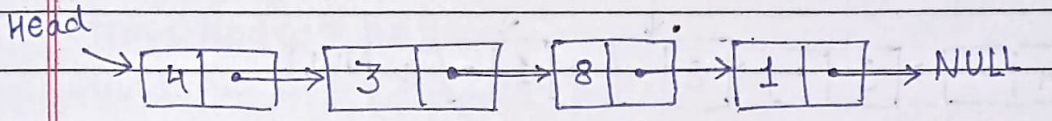
```
Struct Node *insertAfterNode(struct Node *head, Struct Node *prevNode,
                             int data)
{
    Struct Node *ptr = (struct Node*)malloc(sizeof(struct Node));
    ptr->data = data;

    ptr->next = prevNode->next;
    prevNode->next = ptr;

    return head;
}
```

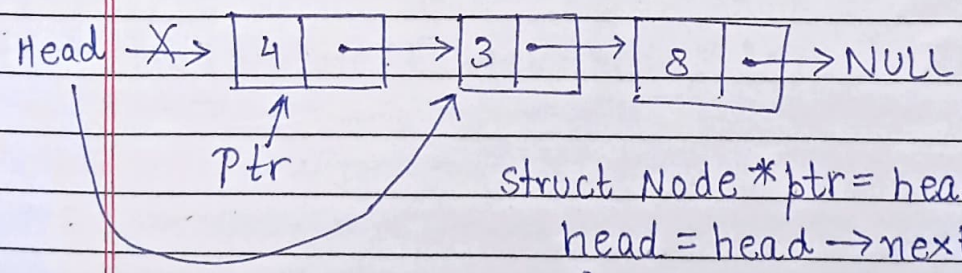
```
//Function Call
->head = insertAfterNode (head, second, 45);
```

Lec-17. Deletion in a Linked List:

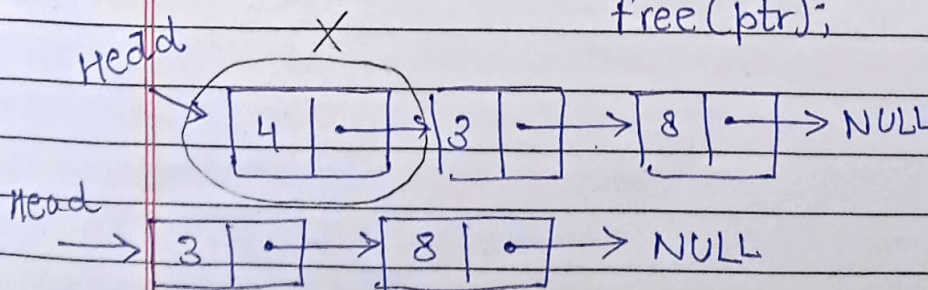


Case: 1 → Deleting the First Node. → O(1)
Case: 2 → Deleting a node inbetween
Case: 3 → Delete the last Node.
Case: 4 → Delete a node with a given value / key / Data.

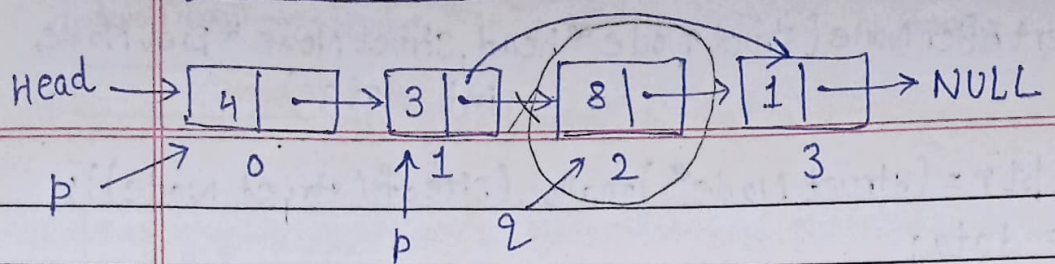
Case: 1: Deleting the first Node




```
Struct Node *ptr = head;
head = head->next;
free(ptr);
```

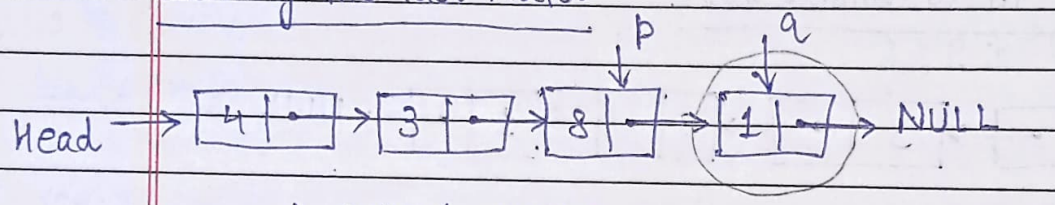


Case 2: Deleting a Node in between: ind=2;




```
Struct Node *p= head;
while(p!= index-1){
        p=p->next;
}
Struct Node *q=p->next;
p->next=q->next;
free(q);
```

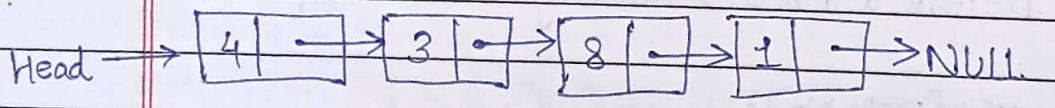
Case 3: Deleting the last Node:




```
p->next= NULL;
free(q);
```

Case 4:

Delete a Node with a given Value:



Lec-18. Delete a Node from Linked List (C code for deleting from beginning, End, Specific position and key).

Green House
Date
Page No.

Case1: Deleting the first Element from the linked list:

```
Struct Node * deletefirst (struct Node *head)
{
struct Node *ptr = head;
head = head -> next;
free (ptr);
return head;
}
```

→ head = deletefirst (head); // Function Call

Case2: Delete At Index:

```
Struct Node * deleteAtIndex( struct Node * head, int index)
{
struct Node * p = head;
struct Node * q = head -> next;
for (int i=0; i< index-1; i++)
 {   p = p -> next;
     q = q -> next;
 }
p -> next = q -> next;
free(q);
return head;
}
```

→ head = deleteAtIndex (head, 1); // Function Call

Case:3. Deleting the last Element:

```
Struct Node *deleteAtLast (struct Node * head)
{
  struct Node *p=head;
  Struct Node *q = head->next;
  while(q->next!=NULL)
  {
     p=p->next;
     q=q->next;
  }
  p->next=NULL;
  free(q);
  return head;
}
```

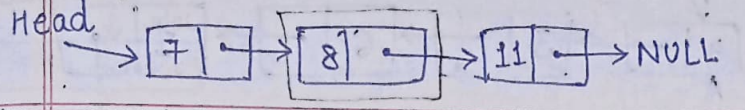
→ `head = deleteAtLast(head);` // function call

Case4: Deleting the Element with a given value from the linked list:

```
Struct Node *deleteAtvalue (struct Node * head, int value){
   Struct Node *p=head;
   Struct Node *q=head->next;
  while(q->data!=value && q->next!=NULL)
  {
     p=p->next;
     q=q->next;
  }
 if(q->data==value){
     p->next=q->next;
     free(q);
  }
 return head;
}
```

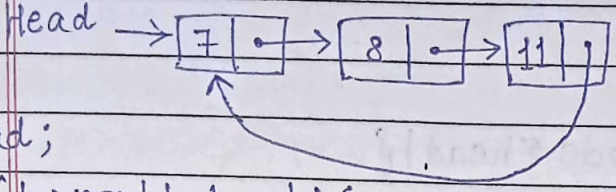
→ `head= deleteAtValue(head, 2);` // Function Call

Lec-19. Circular Linked List and Operations in Data Structure:



Head is a pointer

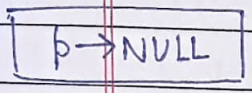
Circular Linked List:

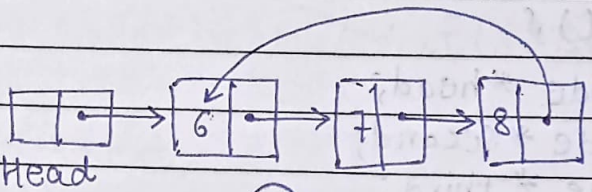



```
p=head;
while(p->next!=head){
  printf(p->data);
  p=p->next;
}
print(p->data);
```

```
do{
printf(p->data)
 p=p->next;
}while(p!=head)
```

Empty Singly Link List

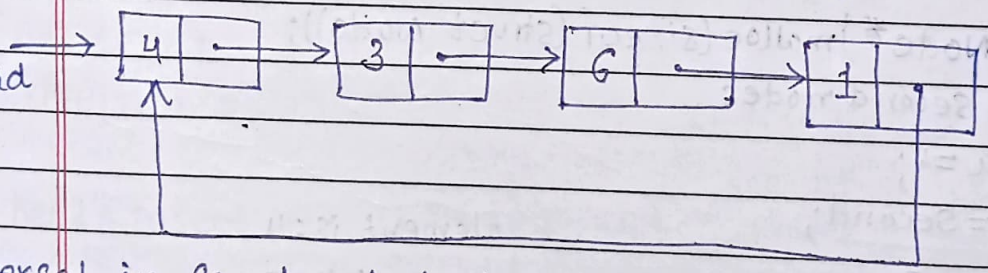




Head is a ~~Node~~

Empty Circular LL.

Lec-20 Circular Linked Lists: Operations in C Language:

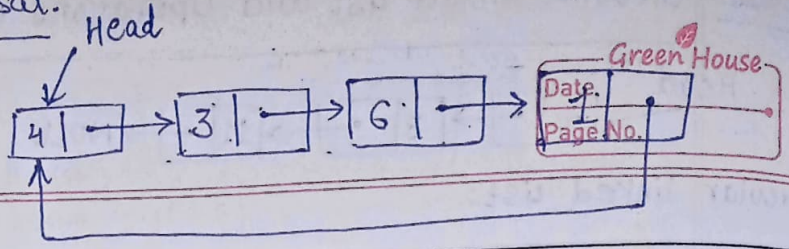


Traversal in Circular Linked List:

```
void LinkedListTraversal (Struct Node *head)
{  Struct Node * ptr =head;
printf(" Element %d ",ptr->data);
ptr = ptr->next;
while (ptr != head){
printf("Element %d ",ptr->data);
  ptr =ptr-> next;
}
}
```

```
void LinkedList Traversal (Struct Node *head
{  Struct Node * ptr=head;
 do {
   printf("Element %d \n",ptr->data);
   ptr= ptr ->next;
} while (ptr != head);
}
```

Circular Linked List Traversal:

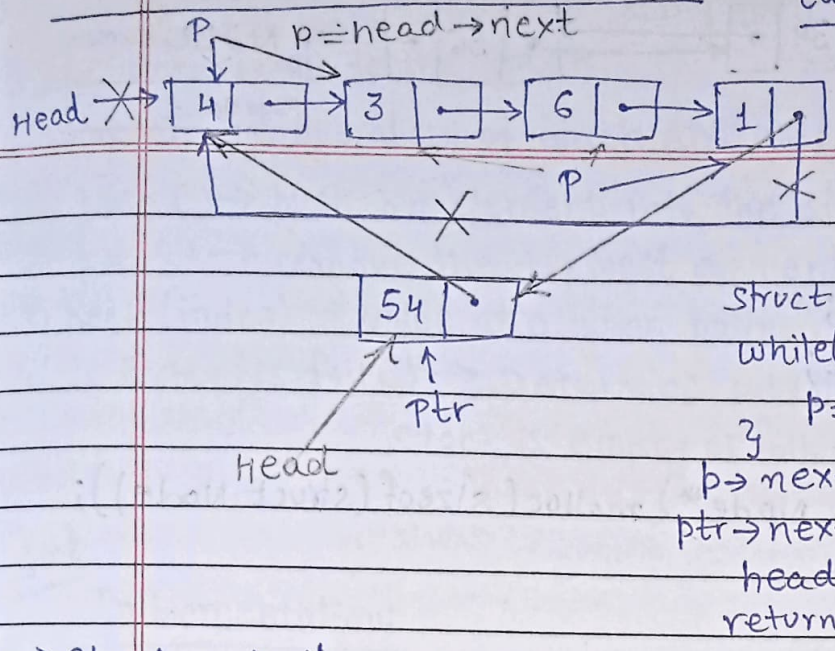



```
#include<stdio.h>
#include<stdlib.h>
struct Node {
    int data;
    struct Node *next;
};
void LinkedListTraversal(struct Node *head){
    struct Node* ptr=head;
    do{
        printf("Element is: %d", ptr->data);
        ptr = ptr->next;
    }while(ptr!=head);
}
int main(){
    struct Node *head;
    struct Node *second;
    struct Node *third;
    struct Node *fourth;
//Allocate memory for nodes in the linked list in Heap
head=(struct Node*)malloc(sizeof(struct Node));
second=(struct Node*)malloc(sizeof(struct Node));
third=(struct Node*)malloc(sizeof(struct Node));
fourth=(struct Node*)malloc(sizeof(struct Node));
//Link first and second nodes
    head->data=4;
    head->next=second;
    second->data=3;
    second->next=third;
    third->data=6;
    third->next=fourth;
    fourth->data=1;
    fourth->next=head;
    LinkedListTraversal(head);
    return 0;
}
```

```
Element is: 4
Element is: 3
Element is: 6
Element is: 1
```

Insertion in Circular Linked List:

Case:1 Insert at First

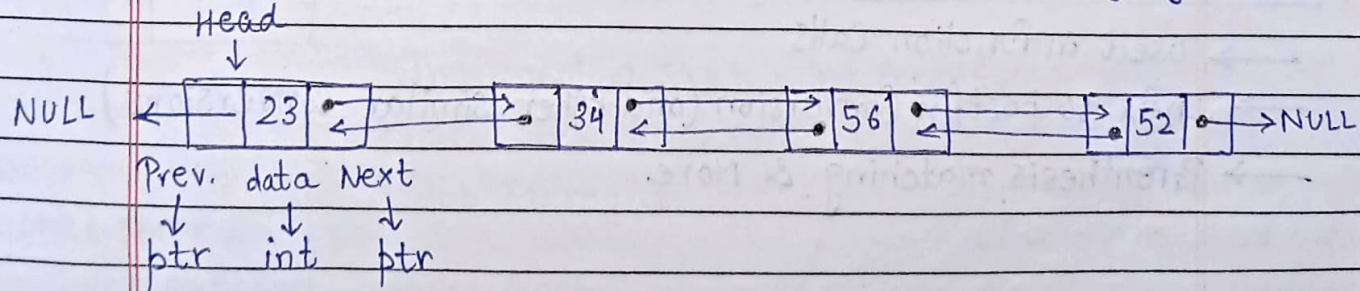


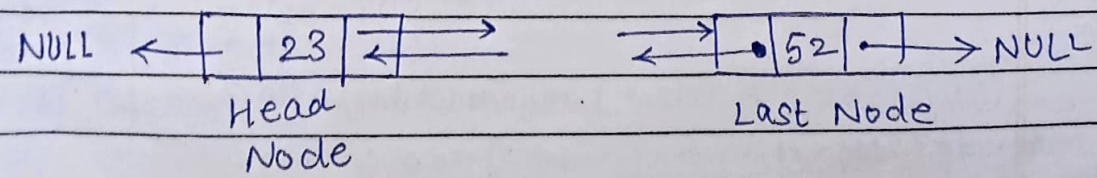

```
ptr→data=data;
Struct Node *p=head→next;
while(p→next!=head){
    p=p→next;
}
p→next=ptr;
ptr→next=head;
head=ptr;
return head;
```

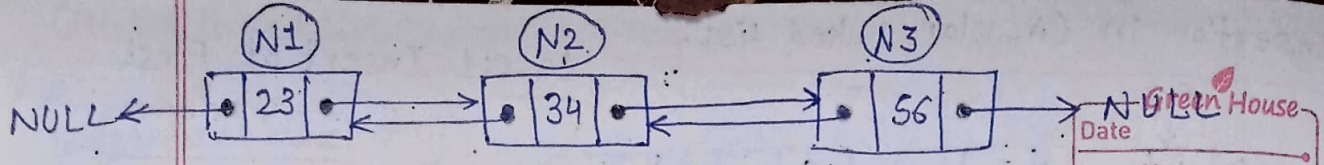
```
→ Struct Node *insertAtFirst(struct Node *head, int data){
    Struct Node*ptr=(struct Node*)malloc(sizeof(struct Node));
        ptr→data=data;
    struct Node *p=head→next;
    while(p→next!=head){
        p=p→next;
    } //At this point p points to the last Node of this Circular LL.
    p→next=ptr;
    ptr→next=head;
    head=ptr;
    return head;
}

→ head=insertAtFirst(head,54)
```

Lec-21. Doubly Linked Lists Explained with Code in C Language:







Green House
Date
Page No.

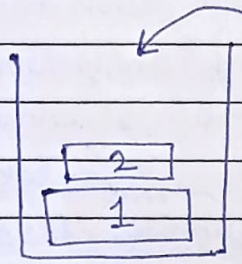
```
struct Node{
    int data;
struct Node *prev;
struct Node *next;
}
```

```
struct Node *N1 =(struct Node*) malloc( sizeof (struct Node));
        N2
        N3
```

```
//Link Nodes
N1 -> next=N2;
N1 -> prev= NULL;
```

Lec-22 Introduction to Stack in Data Structures:

Stack is a linear data structure. Operations on Stack are performed in LIFO (Last in First Out) order.



Insertion/deletion can happen on this end

⇒ Item 2 which entered the basket last, will be the first one to come out

LIFO (Last in First Out)

Applications of Stack:

→ used in function calls
→ Infix to postfix Conversion (and other similar conversions)
→ Parenthesis matching & More.