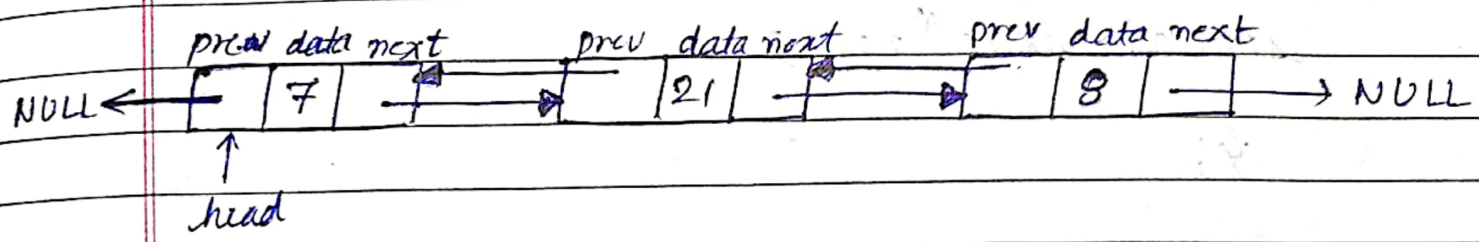
DOUBLY LINKED LISTS

Date
Page

→ Each node contains a data part & 2 pointers in a doubly-linked list, one for previous node & other for next node.
→ Both end pointers point to NULL



→ It allows traversal on both directions. So at any node, we'll have the freedom to choose "left" right as left.
→ Node has 3 parts, the data, a pointer to next node & pointer to previous node.
→ Head node has the pointer to prev. node pointing to NULL

• Operations on Doubly Linked List:

Insertion & deletion can be performed by recursing pointer connections, just like we saw in a singly linked list. Just we need to adjust 2 pointers instead of one.

Creating a node:

```
struct node{

struct node* prev;
int data;
struct node *next;

};

int main(){

struct node * head= malloc(sizeof(struct node));

head -> prev =NULL;
head -> data= 10;
head -> next =NULL;

}
```

INSERTION IN AN EMPTY

LIST

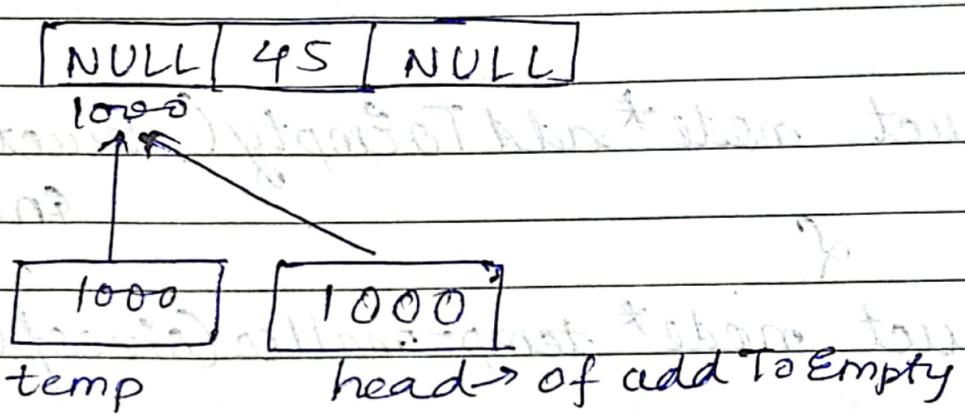
```
#include<stdio.h>
#include<stdlib.h>
```

```
struct node {

struct node* prev;
—11— *next;
int data;

};

struct node *addToEmpty(struct node *head,
                        int data)
{
struct node* temp=malloc(sizeof(struct node));

temp -> prev = NULL;
temp -> data = data;
temp -> next = NULL;
return hea
tem head = temp;
    return head;
}

int main() {
  struct node* head = NULL;
  head = addToEmpty(head, 45);
  printf("%d", head -> data);
    return 0;
  }
```

o/p
45

Internal operation

→ head & data are local to the function addToEmpty
→ A new node temp is created

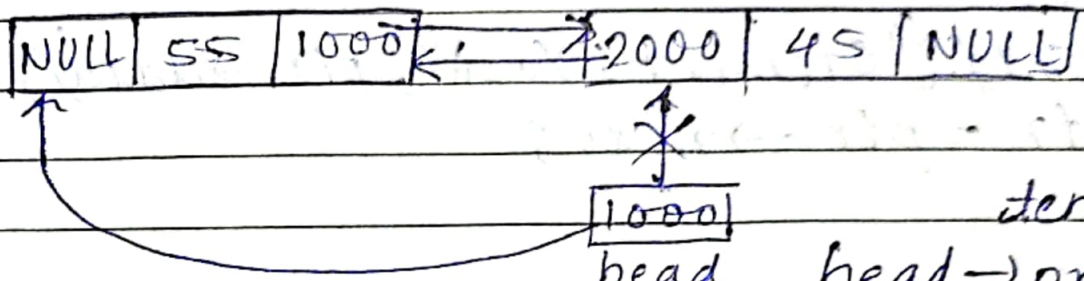


&

To store the chead of the function in the head of the main function we return head.

Note: You can use void function also when you are doing operations in between the LL.

INSERTION AT BEGINNING



temp → next = head
head → prev = temp
head = temp

Code:

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

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

};

int main(){
struct node * head = NULL;
struct node * ptr;

head = addToEmpty(head, 45);
head = addAtBeg(head, 34);
```

// Code for Traversal

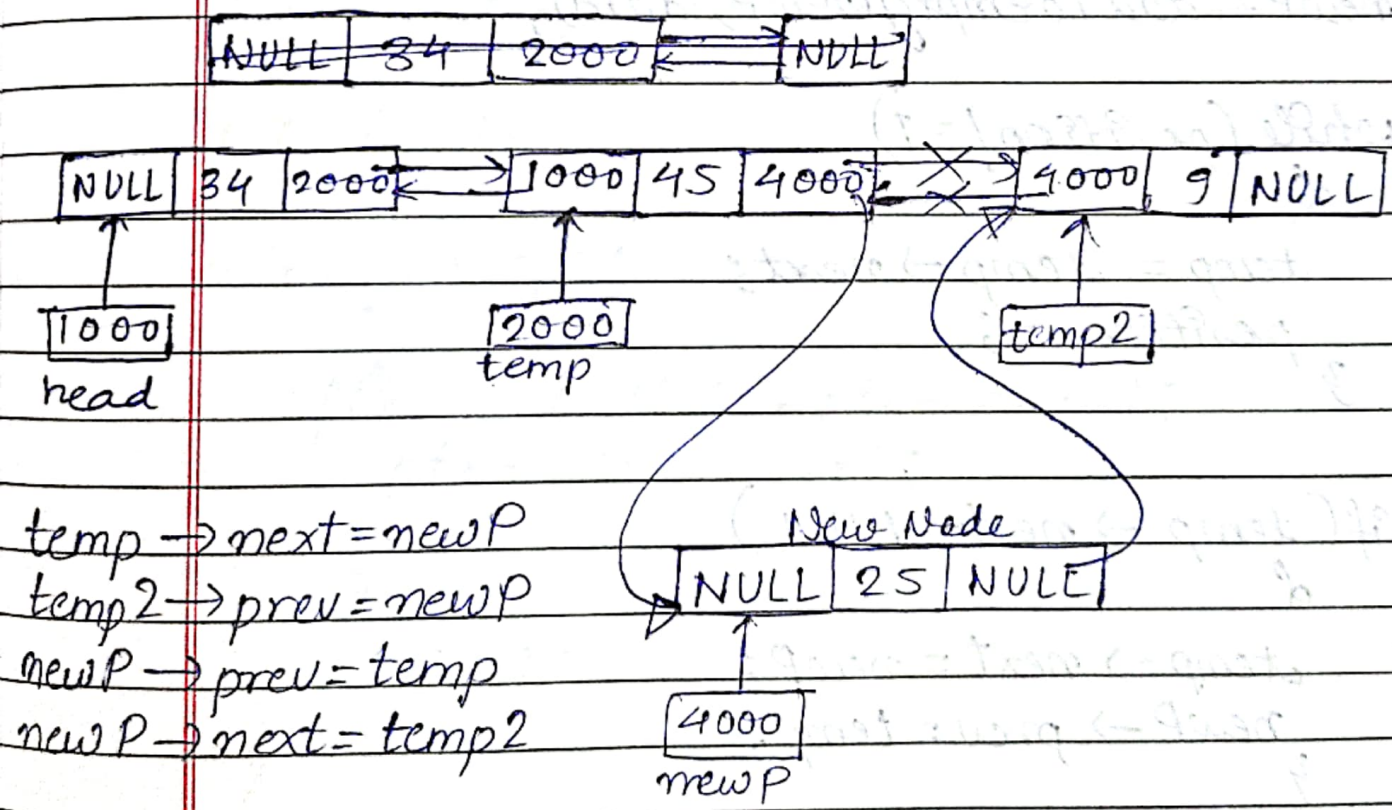
```
ptr = head;
while(ptr != NULL)
{
  printf("Element : %d\n", ptr -> data);
    ptr = ptr -> next;

}
return 0;

}

struct node * addAtBeg(
    struct node* head, int data)
{

struct node * temp = malloc(sizeof(struct node));

temp -> prev = NULL;
temp -> data = data;
temp -> next = NULL;

temp -> next = head;
head -> prev = temp;
head = temp;

return head;

}
```

Help in creating new node

o/p:

Element: 34
—11— : 45

INSERTION IN BET^N THE NODES



temp → next = newP
temp2 → prev = newP
newP → prev = temp
newP → next = temp2

Code:

```
#include<stdio.h>
#—"——<stdlib.h>
```

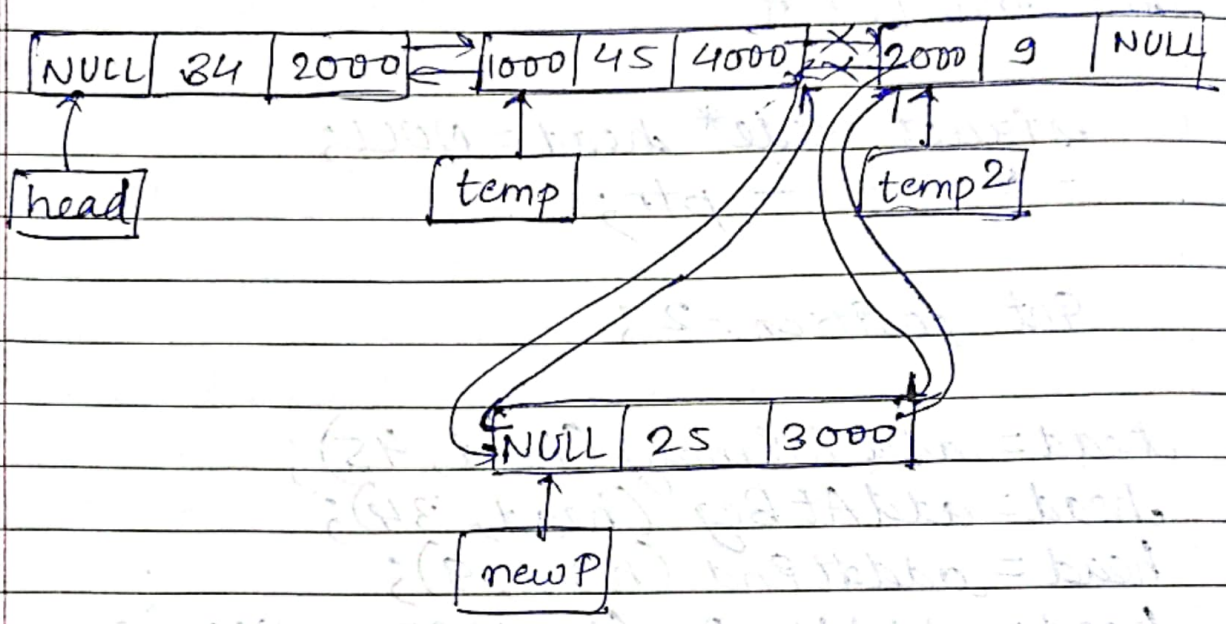
```
struct node {
    struct node* prev;
    int data;
    struct node* next;
};

struct node* addAfterPos(struct node *head,
        int data, int position)
{
struct node* newP = NULL;
struct node* temp = head;
--11-------- temp2 = NULL;

newP = addToEmpty(newP, data);

while (position != 1)
{
    temp = temp -> next;
    position--;
}

if( temp -> next == NULL)
{
    temp -> next = newP;
    newP -> prev = temp;
}
else {
    temp2 = temp -> next;
    temp -> next = newP;
    temp2 -> prev = newP;
    newP -> next = temp2;
    newP -> prev = temp;
}
```

```
return head;
}

int main() {

    struct node* head = NULL;
    —"— ptr;

    int position = 2;

head = addToEmpty(head, 45);
head = addAtBeg(head, 34);
head = addAtEnd(head, 9);
head = addAfterPos(head, 25, position);

ptr = head;

while (ptr != NULL)
    {
    pf("Element: %d\n", ptr->data);
        ptr = ptr -> next;

    }
    return 0;

}
```

O/P

```
Element : 34
—"— : 45
—"— : 25
—"— : 9
```

INSERTION BEFORE A NODE



E.g. code

Refer to Node creation etc. to prev codes

```
struct node* addBeforePos(struct node*head,
int data, int position)
{
struct node* newP = NULL;
temp = head;
temp2 = NULL;

newP = addToEmpty(newP, data);

int pos = position;

while(pos > 2){
    temp = temp->next;
    pos--;
}
```

```
If (position == 1)
{
    head = addAtBeg (head, data);
}
else{
    temp2 = temp -> next;
    temp -> next = newP;
    temp2 -> prev = newP;
    newP -> next = temp2;
    newP -> prev = temp;
}
return head;
}

int main () {
struct node* head = NULL;
— " ——— ptr;
int position = 3;

head = addToEmpty (head, 45);
head = addTo Beg (head, 34);
head = add AtEnd (head, 9);
head = add BeforePos (head, 25, position);

ptr = head;
while (ptr != NULL) {
    pf ("Element : %d\n", ptr -> data);
    ptr = ptr -> next;
}

return 0;
}
```

o/p: Element : 34

11 : 45

11 : 25

11 : 9

Function to Insert a element in a sorted DLL

```
void sortedInsert(struct node* head, data)
{
struct node* new=(struct node*)malloc(
                    sizeof(struct node));
  new->data=data;
new->next=new->prev=NULL;

  if(head==NULL)
     head=new;
```

```
else if (head -> data >= data){
    new -> next = head;
    new -> next -> prev = new;
    head = new;
}

else {
    struct node* current = head;
    while (current -> next != NULL &&
    current -> next -> data < data)
    {
        current = current -> next;
    }
    new -> next = current -> next;
// If new node is not inserted at
   the end of DLL
    if (current -> next != NULL)
    {
        new -> next -> prev = new;
    }
    current -> next = new;
    new -> prev = current;
}
}
```