SNS COLLEGE OF TECHNOLOGY

Coimbatore-35

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

COURSE NAME : 23ItT201 DATA STRUCTURES

II YEAR/ III SEMESTER

UNIT – II LISTS

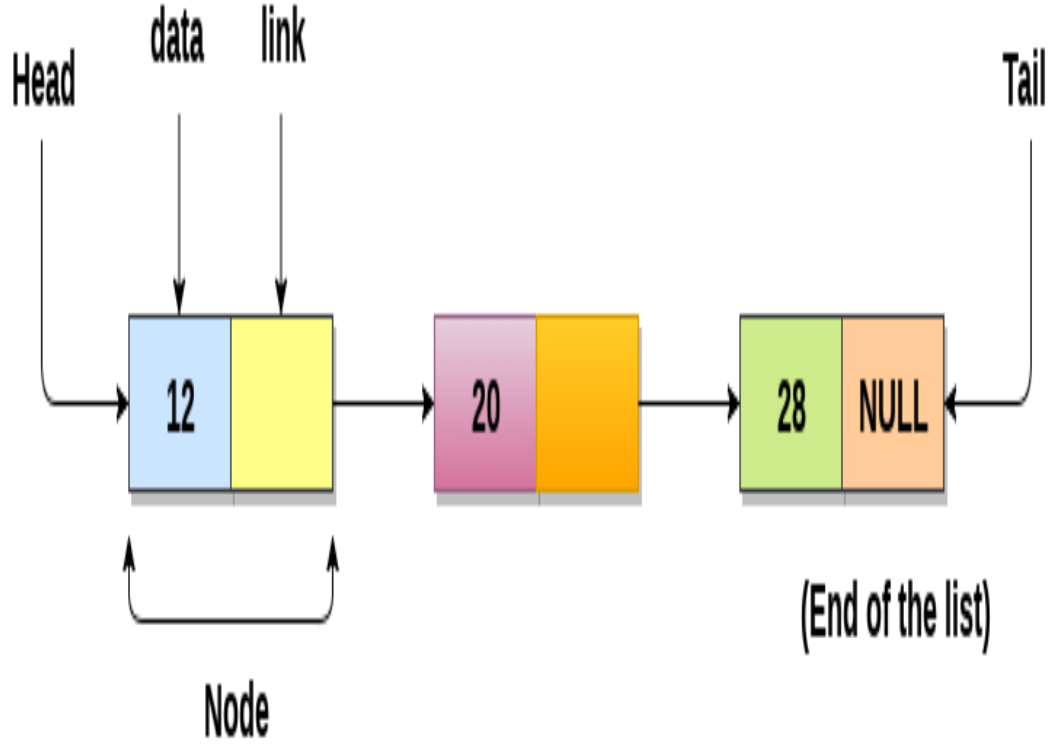
Topic: Doubly Linked List

Linked List

- Linked List can be defined as collection of objects called **nodes** that are randomly stored in the memory.

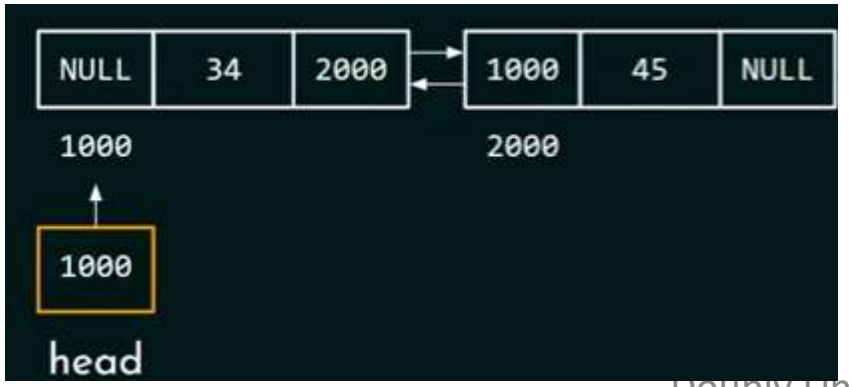
Types of Linked List

- ➢Singly Linked List
- ➢Doubly Linked list
- ➢Circularly Linked List
- Singly Linked List
- A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.
- The last node of the list contains pointer to the null.



Doubly Linked List

- Each node has three fields
  - Data field
  - Forward Link(FLINK)
  - Backward link(BLINK)
- FLINK points to the successor node in the list
- BLINK points to the predecessor node



Singly Linked List

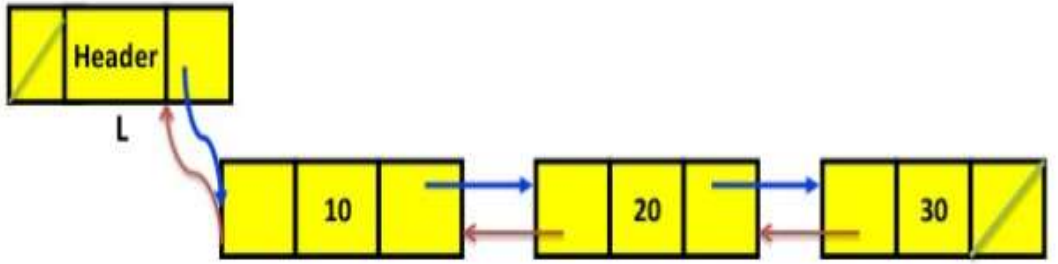
```
struct node {
    int data;
    struct node* link;
};
```

Doubly linked list

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

Doubly Linked List

- Each node has three fields
  - Data field
  - Forward Link(FLINK)
  - Backward link(BLINK)
- FLINK points to the successor node in the list
- BLINK points to the predecessor node

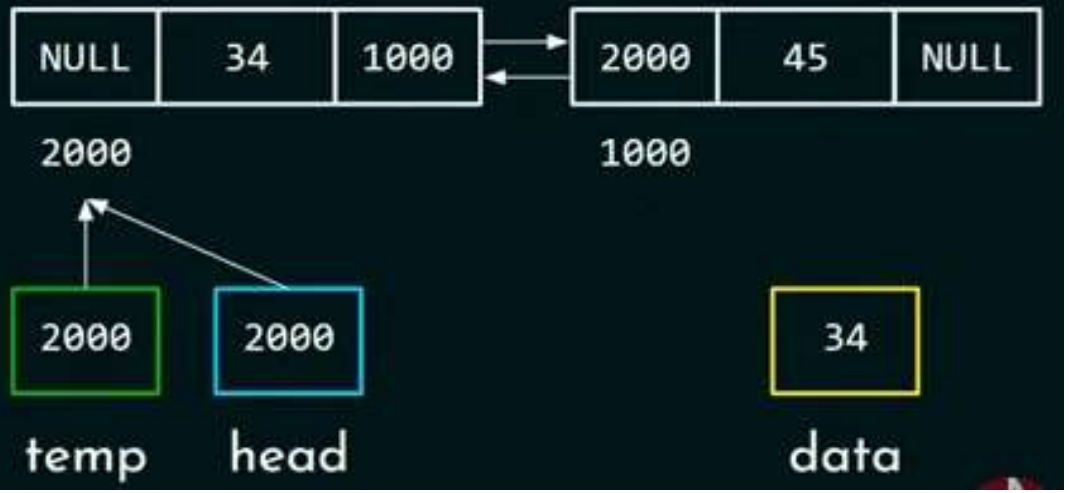


Structure Declaration

```
Struct Node
{
Int element;
Struct Node *FLINK;
Struct Node *BLINK;
}
```

Insertion at Beginning

```
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;
}
```



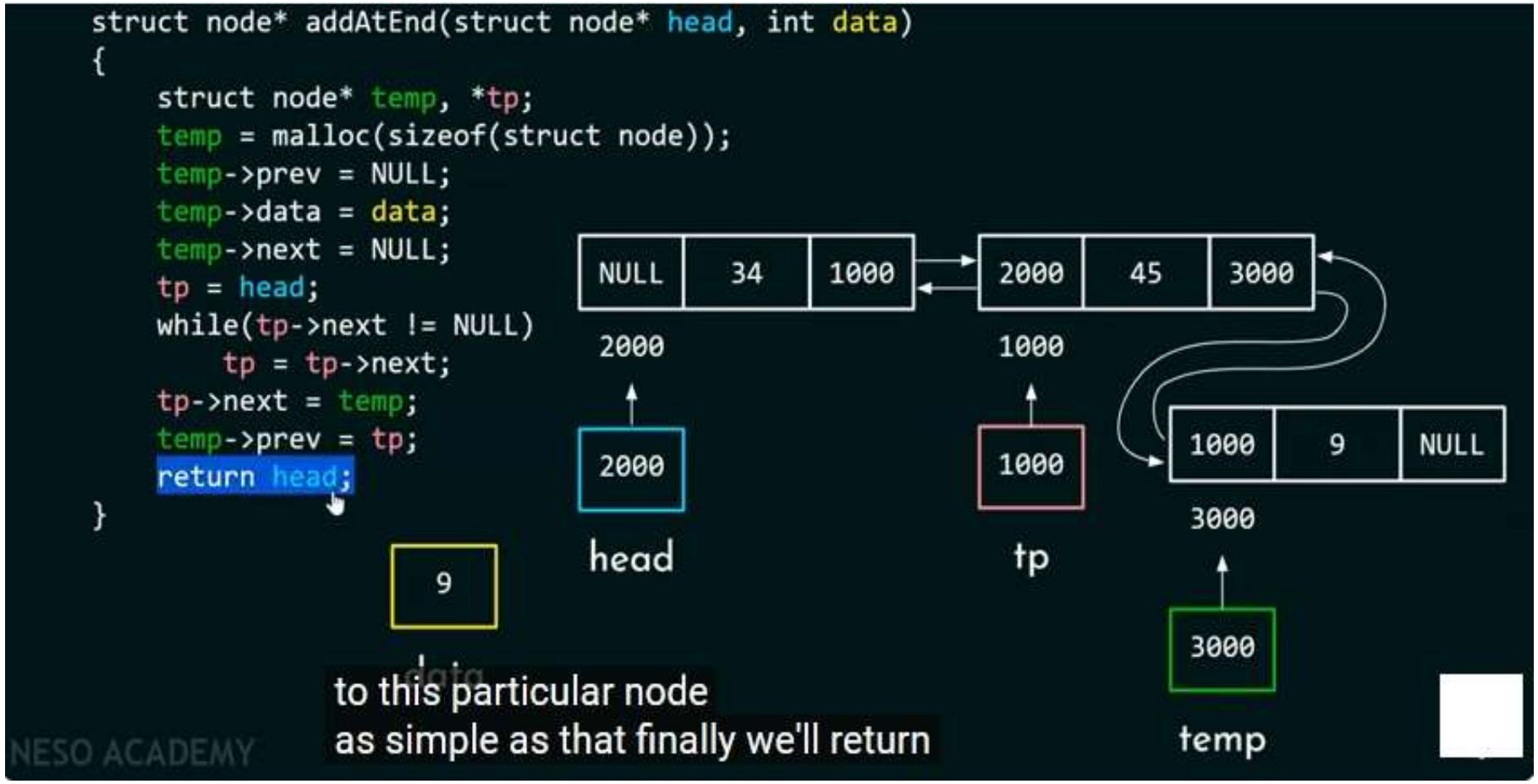
Insertion at Specific Position

addBeforePos Function

```
struct node* addBeforePos(struct node* head, int data, int position)
{
    struct node* newP = NULL;
    struct node* temp = head;
    struct node* temp2 = NULL;
    newP = addToEmpty(newP, data);
    int pos = position;
    while(pos > 2)
    {
        temp = temp->next;
        pos--;
    }
    temp2 = temp->next;
    temp->next = newP;
    temp2->prev = newP;
    newP->next = temp2;
    newP->prev = temp;
    return head;
};
```

Insertion at End

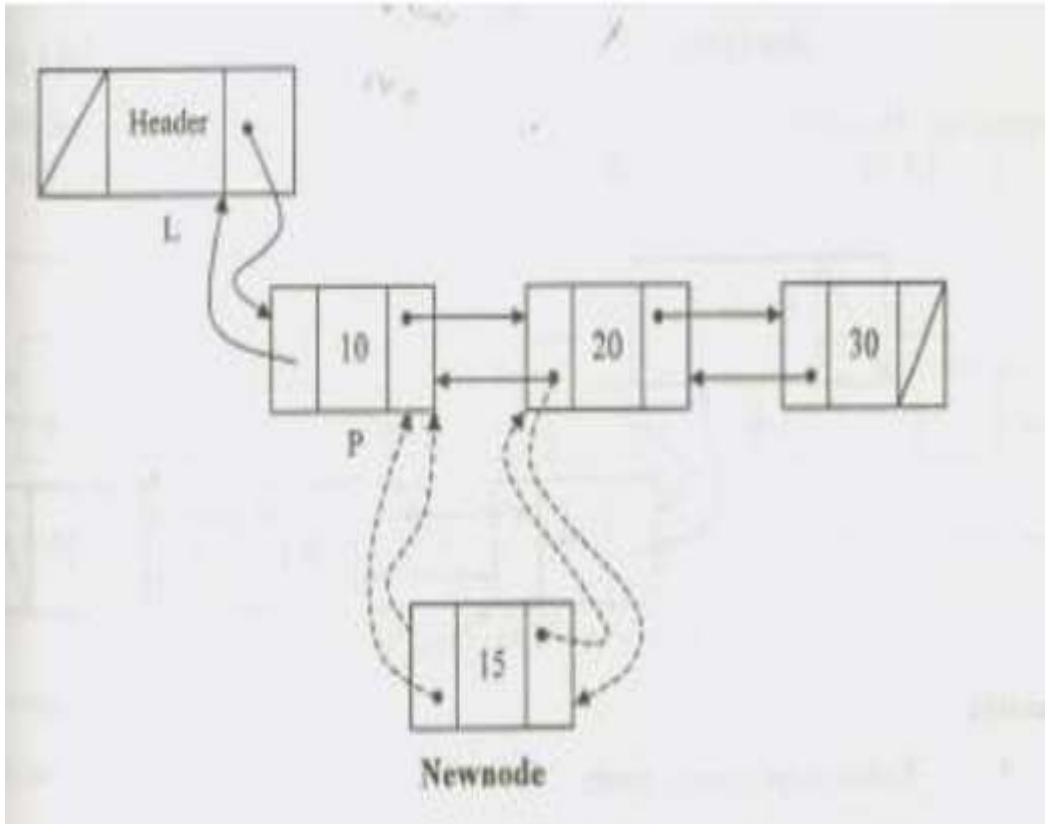
```
struct node* addAtEnd(struct node* head, int data)
{
    struct node* temp, *tp;
    temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    tp = head;
    while(tp->next != NULL)
        tp = tp->next;
    tp->next = temp;
    temp->prev = tp;
    return head;
}
```



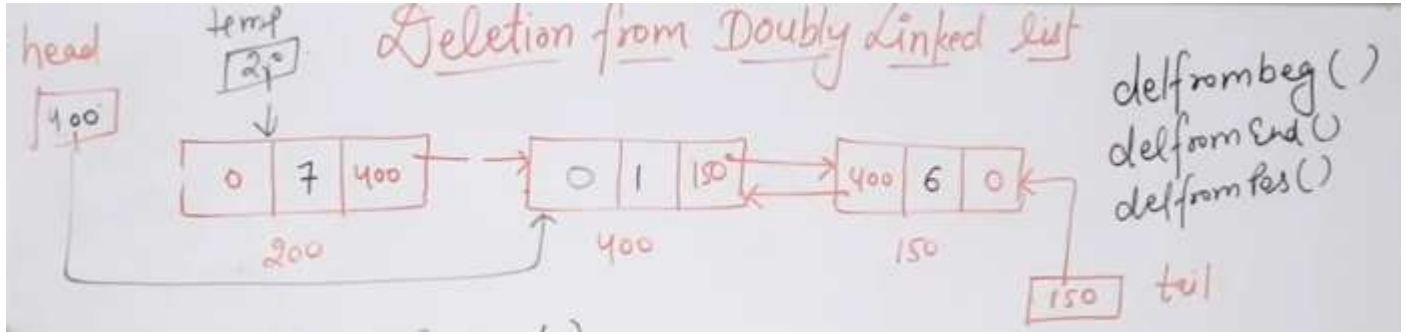
Doubly Linked List

Routine to insert an element in a doubly linked list

```
Void insert(int x, list l, position p)
{
    struct Node *Newnode;
    Newnode= malloc(size of(Struct Node));
    If (Newnode !=NULL)
    {
        Newnode → Element =x;
        Newnode → Flink = p→ Flink;
        P → flink → blink =Newnode;
        P → Flink = Newnode;
        Newnode → Blink = p;
    }
}
```

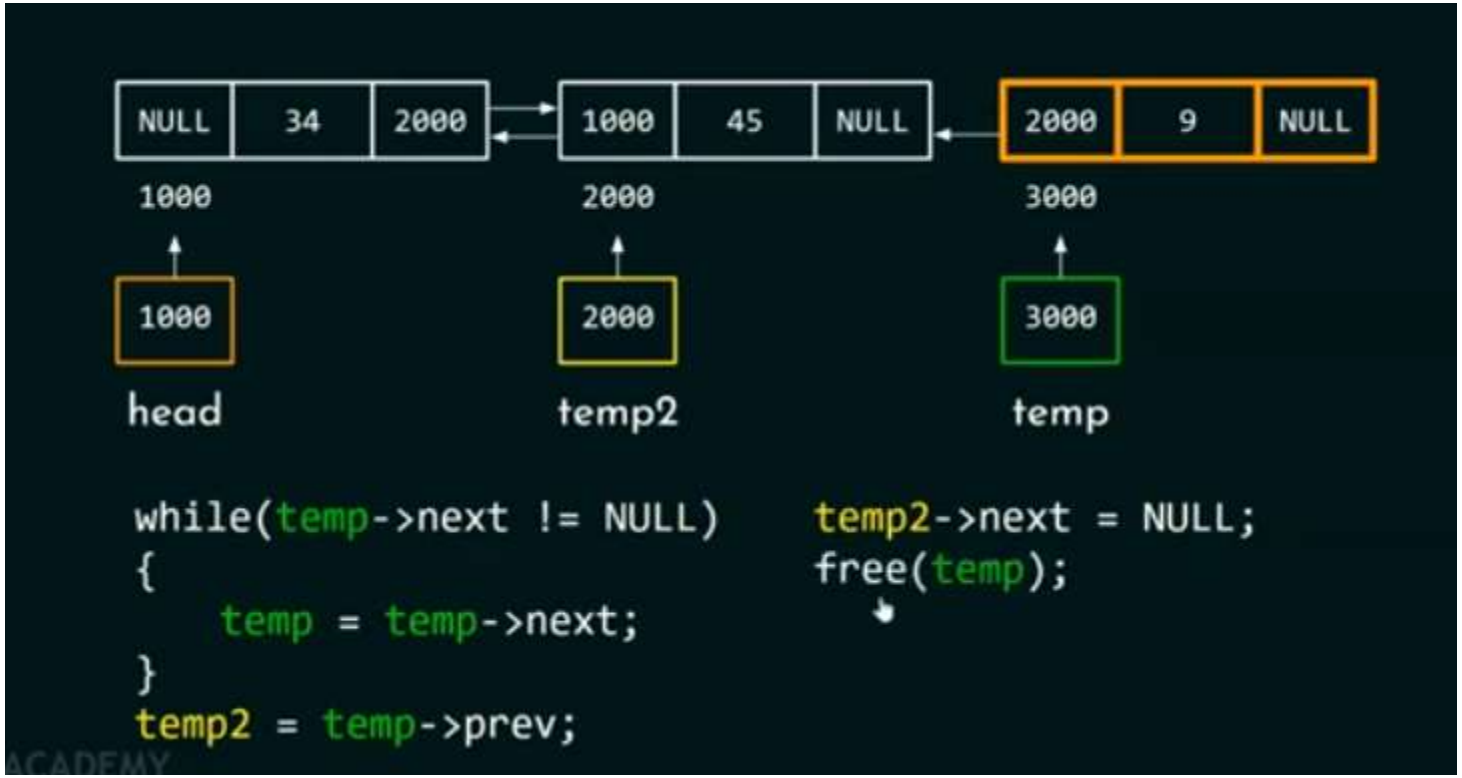


Deletion at Beginning

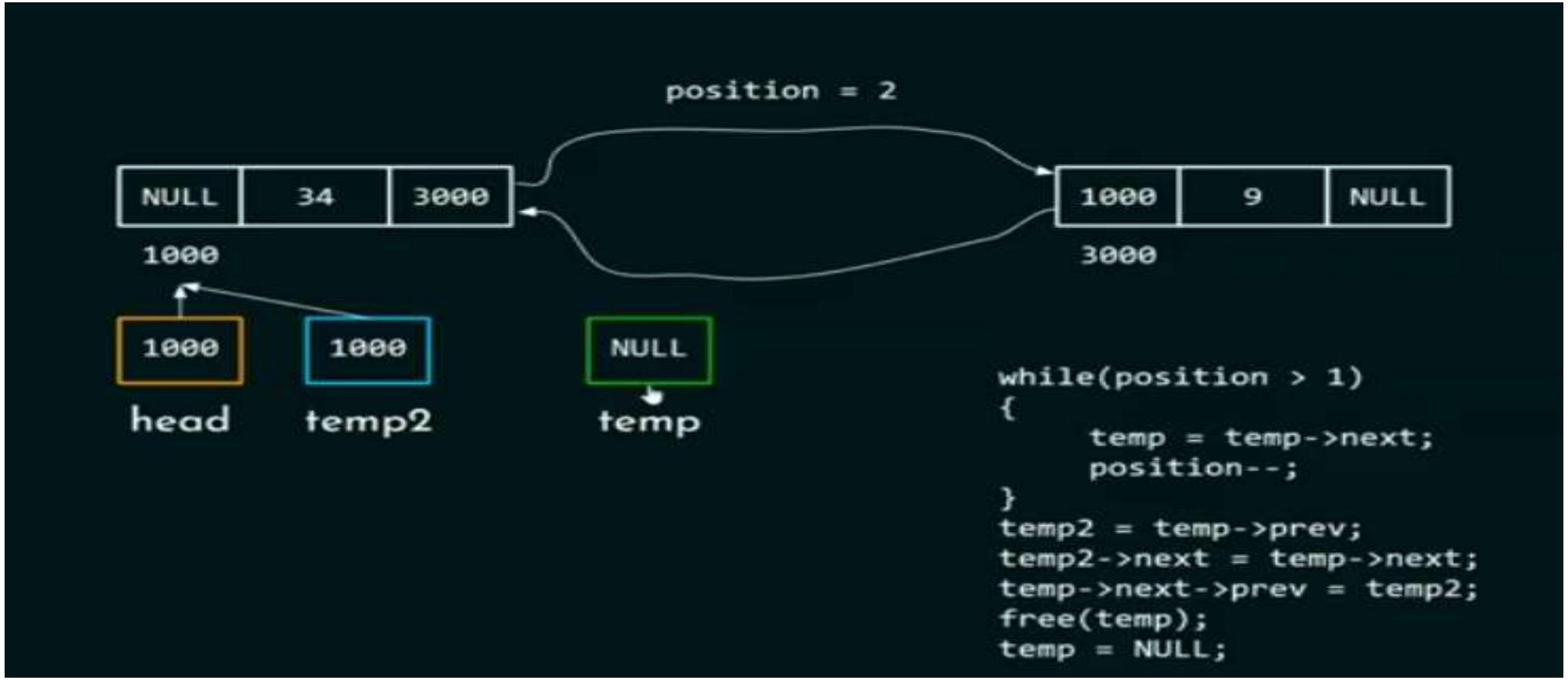



```
void delFromBeg()
{
  struct node *temp;
  if(head==0)
    {
      Printf("list is empty");
    }
  else
    {
      temp = head;
      head = head->next;
      head->prev = 0;
    }
free(temp);
}
```

Deletion at End

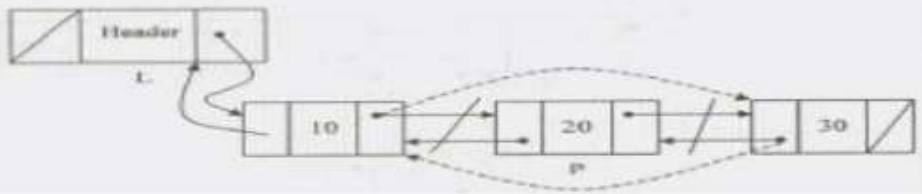


Deleting Intermediate Node



Doubly Linked List

```
void Delete (int X, List L)
{
        position P;
        P = Find (X, L);
        If ( IsLast (P, L))
        {
                Temp = P;
                P -> Blink -> Flink = NULL;
                free (Temp);
        }
        else
        {
                Temp = P;
                P -> Blink -> Flink = P -> Flink;
                P -> Flink -> Blink = P -> Blink;
                free (Temp);
        }
}
```

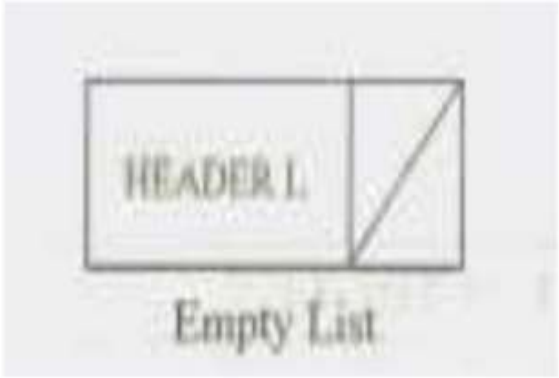


- Advantages
  - Deletion operation is easier
  - Finding predecessor & successor of a node is easier
- Disadvantages
  - More memory space required since it has two pointers

Doubly Linked List

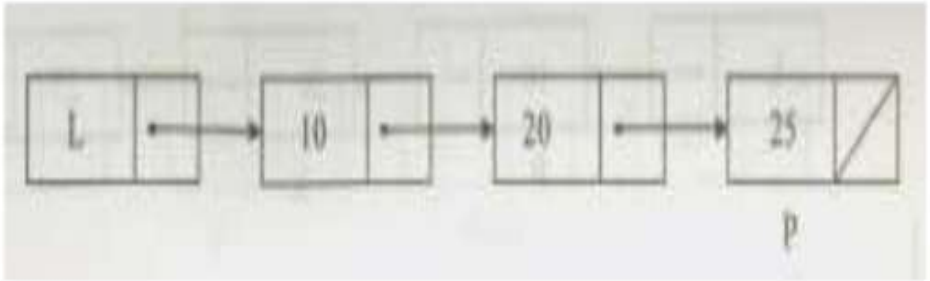
- **Routine to check whether list is empty**

```
int IsEmpty(List L) // returns 1 if L is empty
{
  if (L →Next == NULL)
  return(1);
}
```



- **Routine to check whether current position is last**

```
int IsLast(position p,List L) // returns 1 if P is the last
  position in L
{
  if (P →Next == NULL)
  return(1);
}
```



Assessment 2

How to Insert an element in the Given list at beginning , at end and at middle

References

1. M. A. Weiss, “Data Structures and Algorithm Analysis in C”, Pearson Education, 2nd Edition, 2002.
2. A. V. Aho, J. E. Hopcroft and J. D. Ullman, “Data Structures and Algorithms”, Pearson Education, 2nd Edition, 2007
3. Ashok Kamthane, " Data Structures Using C ", Pearson Education, 2nd Edition, 2012.
4. Sahni Horowitz, “Fundamentals of Data Structures in C”Universities Press; Second edition 2008

Thank You