Fundamentals of
Data Structures using C

Singly Linked List

B.Bhuvaneswaran, AP (SG) / CSE

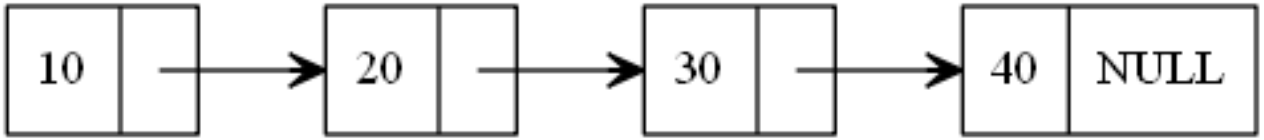
9791519152

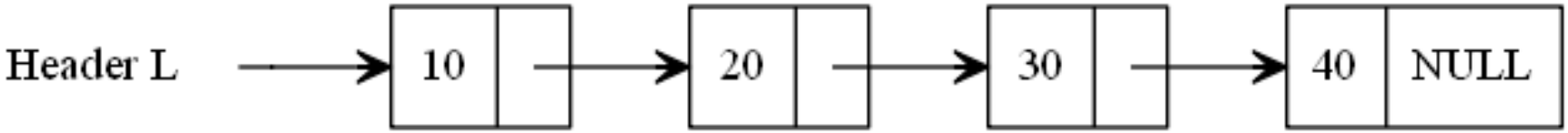
bhuvaneswaran@rajalakshmi.edu.in

RAJALAKSHMI
ENGINEERING COLLEGE

Introduction

- A singly linked list is a linked list in which each node contains only one link field pointing to the next node in the list.

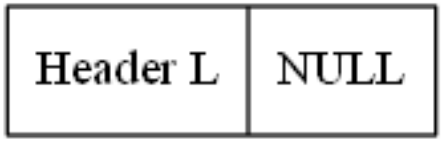




Type declarations for singly linked list

```
struct node
{
        int Element;
        struct node *Next;
};
typedef struct node Node;
```

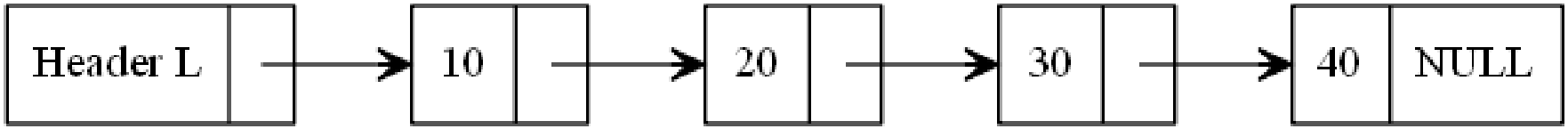
Empty List with Header



Routine

```
int IsEmpty(Node *List)
{
        if(List->Next == NULL)
                return 1;
        else
                return 0;
}
```

Current Position is Last

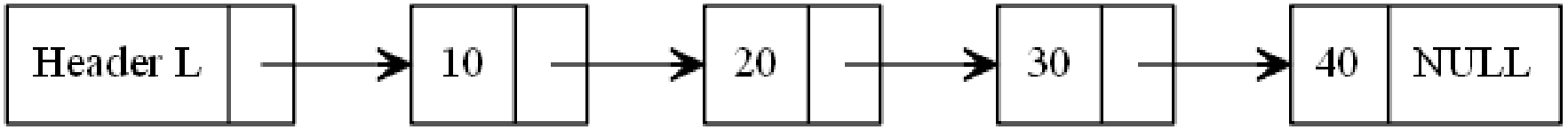


Routine

```
int IsLast(Node *Position)
{
        if(Position->Next == NULL)
                return 1;
        else
                return 0;
}
```

Find

- Find(List, 30)

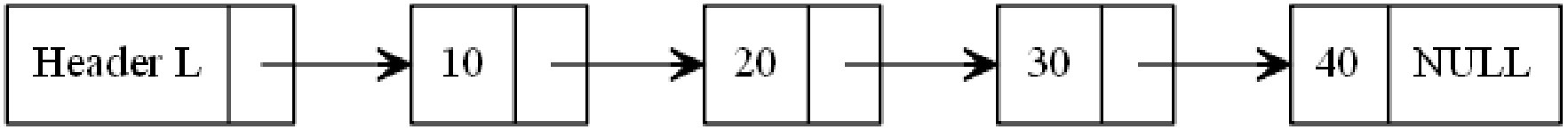


Routine

```
Node *Find(Node *List, int x)
{
        Node *Position;
        Position = List->Next;
        while(Position != NULL && Position->Element != x)
                        Position = Position->Next;
        return Position;
}
```

FindPrevious

- FindPrevious(List, 30)

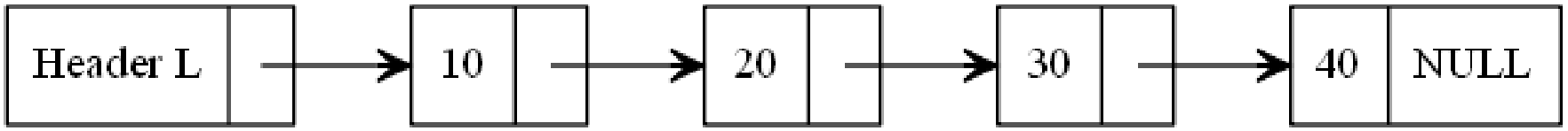


Routine

```
Node *FindPrevious(Node *List, int x)
{
        Node *Position;
        Position = List;
        while(Position->Next!=NULL && Position->Next->Element!=x)
                        Position = Position->Next;
        return Position;
}
```

FindNext

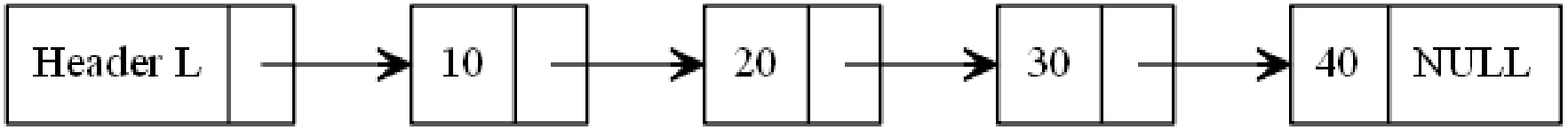
- FindNext(List, 20)



Routine

```
Node *FindNext(Node *List, int x)
{
        Node *Position;
        Position = Find(List, x);
        return Position->Next;
}
```

Traverse the List



Routine

```
void Traverse(Node *List)
{
        if(!IsEmpty(List))
        {
                Node *Position;
                Position = List;
                while(Position->Next != NULL)
                {
                        Position = Position->Next;
                        printf("%d\t", Position->Element);
                }
                printf("\n");
        }
        else
                printf("List is empty...!");
}
```

Insert

- The insert command requires obtaining a new cell from the system by using an malloc call and then executing two pointer maneuvers.

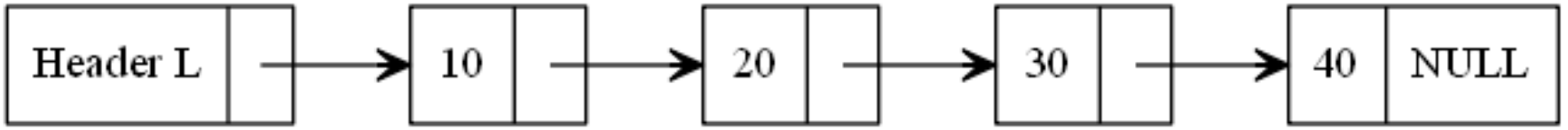
Insert

- We will pass an element to be inserted along with the list L and a position P.
- Our particular insertion routine will insert an element after the position implied by P.
- This decision is arbitrary and meant to show that there are no set rules for what insertion does.
- It is quite possible to insert the new element into position P (which means before the element currently in position p), but doing this requires knowledge of the element before position P.
- This could be obtained by a call to Find.

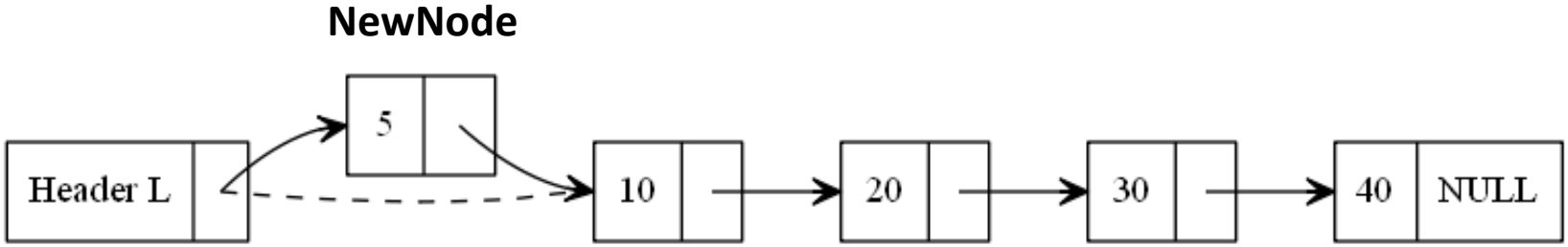
Insertion

- Insert an element at the beginning
- Insert an element at the end
- Insert an element in the middle

Insert an Element at the Beginning



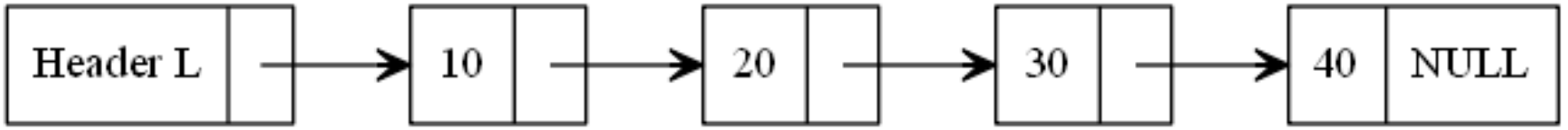
InsertBeg(List, 5)



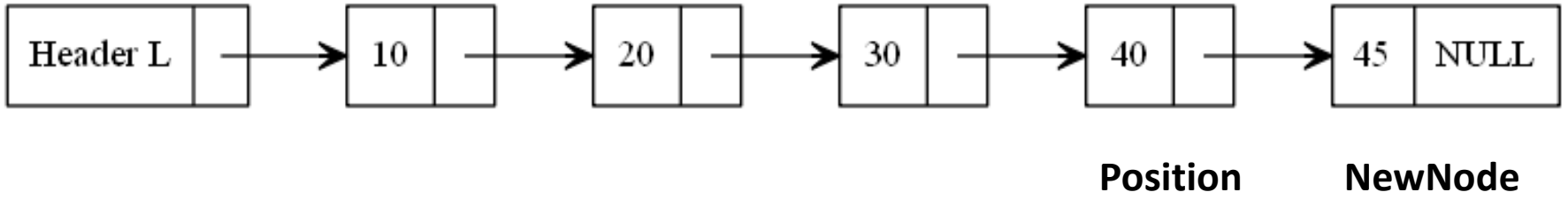
Routine

```
void InsertBeg(Node *List, int e)
{
        Node *NewNode = malloc(sizeof(Node));
        NewNode->Element = e;
        if(IsEmpty(List))
                NewNode->Next = NULL;
        else
                NewNode->Next = List->Next;
        List->Next = NewNode;
}
```

Insert an Element at the End



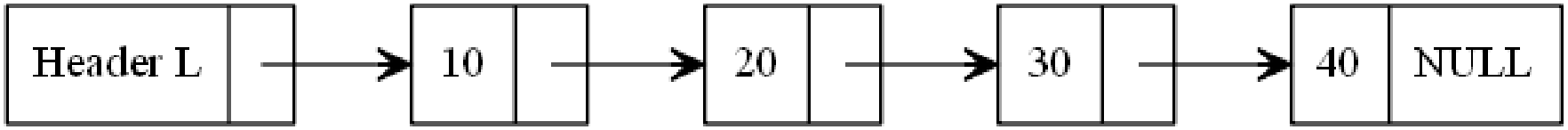
InsertLast(List, 45)



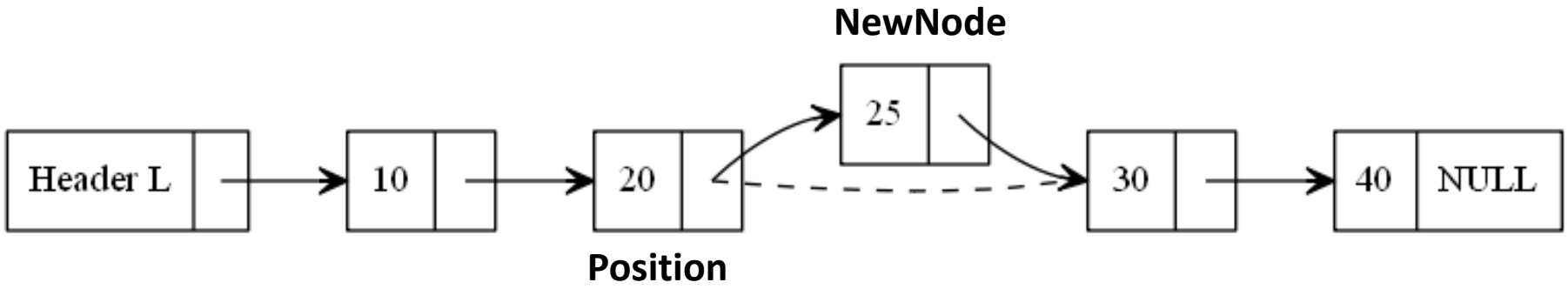
Routine

```
void InsertLast(Node *List, int e)
{
        Node *NewNode = malloc(sizeof(Node));
        Node *Position;
        NewNode->Element = e;
        NewNode->Next = NULL;
        if(IsEmpty(List))
                List->Next = NewNode;
        else
        {
                Position = List;
                while(Position->Next != NULL)
                        Position = Position->Next;
                Position->Next = NewNode;
        }
}
```

Insert an Element in the Middle



InsertMid(List, 20, 25)



Routine

```
void InsertMid(Node *List, int p, int e)
{
        Node *NewNode = malloc(sizeof(Node));
        Node *Position;
        Position = Find(List, p);
        NewNode->Element = e;
        NewNode->Next = Position->Next;
        Position->Next = NewNode;
}
```

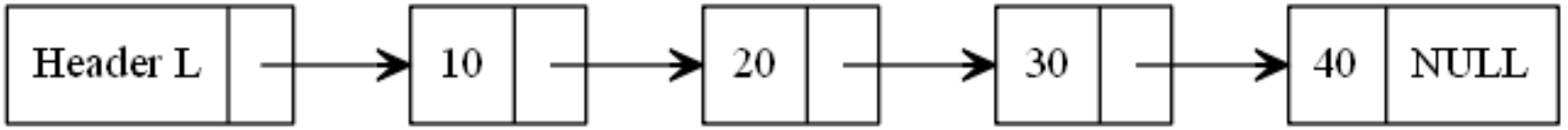
Delete

- The delete command can be executed in one pointer change.
- Our routine will delete some element X in list L.
- We need to decide what to do if x occurs more than once or not at all.
- Our routine deletes the first occurrence of x and does nothing if x is not in the list.
- To do this, we find p, which is the cell prior to the one containing x, via a call to FindPrevious.

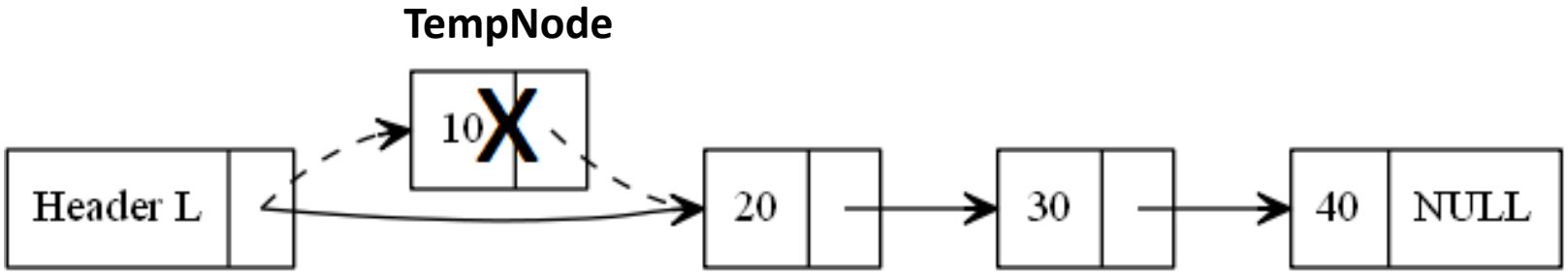
Deletion

- Delete an element from the beginning
- Delete an element from the end
- Delete an element from the middle

Delete an Element from the Beginning



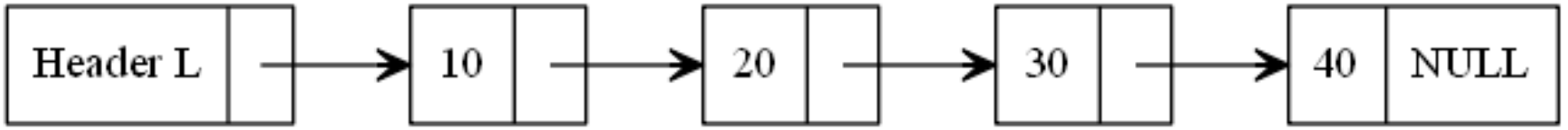
DeleteBeg(List)



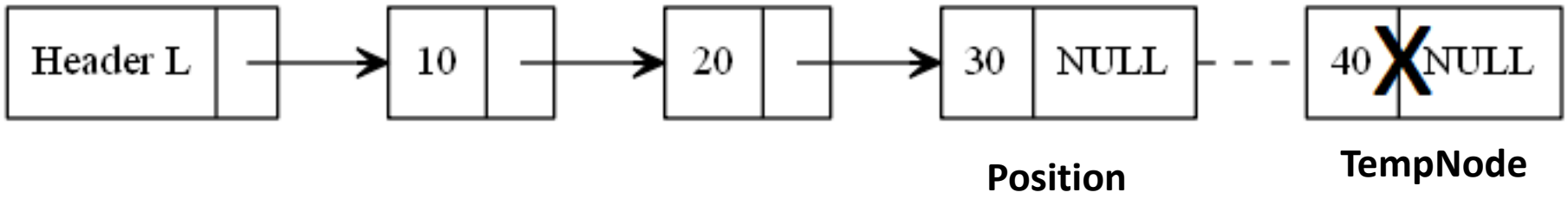
Routine

```
void DeleteBeg(Node *List)
{
        if(!IsEmpty(List))
        {
                Node *TempNode;
                TempNode = List->Next;
                List->Next = TempNode->Next;
                printf("The deleted item is %d\n", TempNode->Element);
                free(TempNode);
        }
        else
                printf("List is empty...!\n");
}
```

Delete an Element from the End



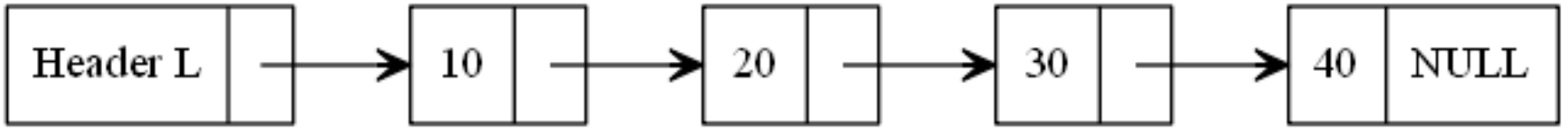
DeleteEnd(List)



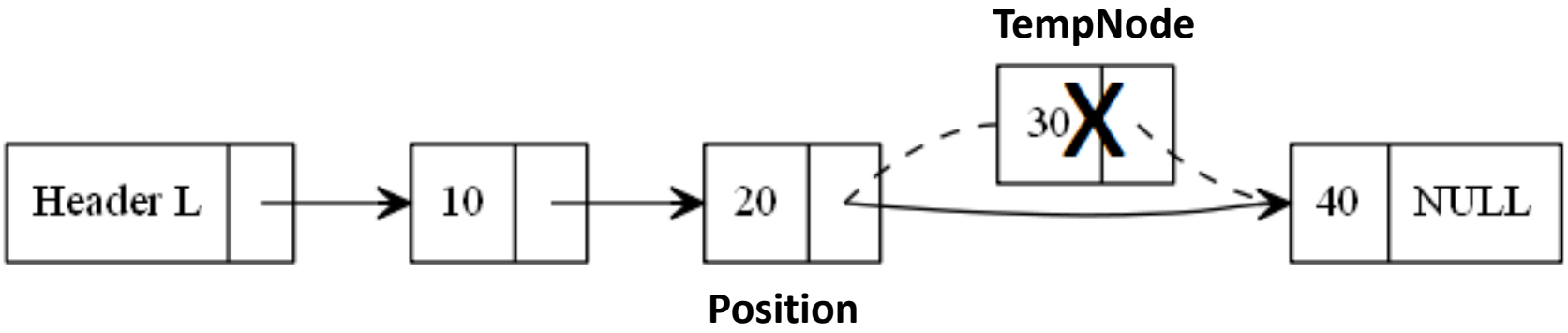
Routine

```
void DeleteEnd(Node *List)
{
        if(!IsEmpty(List))
        {
                Node *Position;
                Node *TempNode;
                Position = List;
                while(Position->Next->Next != NULL)
                        Position = Position->Next;
                TempNode = Position->Next;
                Position->Next = NULL;
                printf("The deleted item is %d\n", TempNode->Element);
                free(TempNode);
        }
        else
                printf("List is empty...!\n");
}
```

Delete an Element from the Middle



DeleteMid(List, 30)



Routine

```
void DeleteMid(Node *List, int e)
{
        if(!IsEmpty(List))
        {
                Node *Position;
                Node *TempNode;
                Position = FindPrevious(List, e);
                if(!IsLast(Position))
                {
                        TempNode = Position->Next;
                        Position->Next = TempNode->Next;
                        printf("The deleted item is %d\n", TempNode->Element);
                        free(TempNode);
                }
        }
        else
                printf("List is empty...!\n");
}
```

Queries?

Thank You!