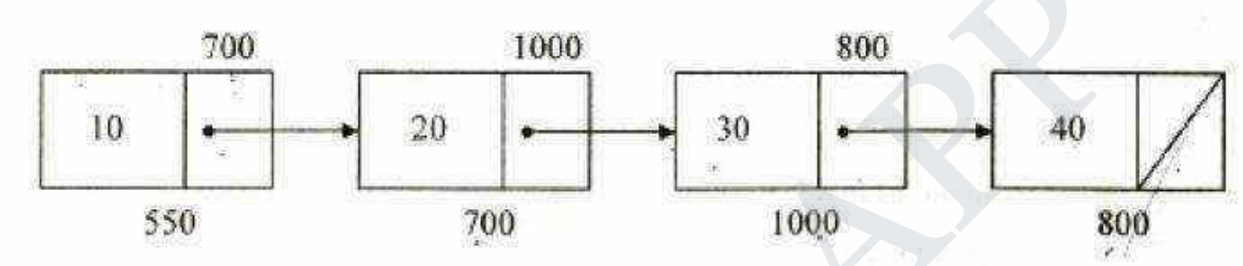
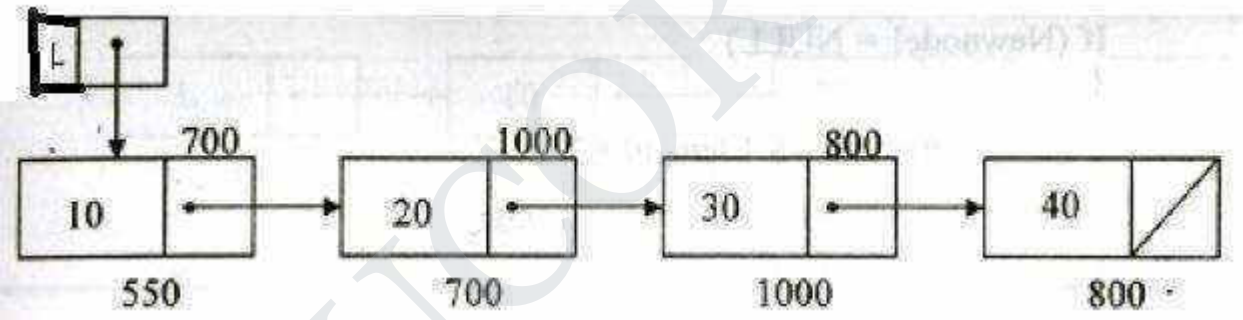
variable ptr. It returns a pointer to the first byte of the memory block. The allocated new block may be or may not be at the same region. Thus, we see that realloc() takes two arguments. The first is the pointer referencing the memory and the second is the total number of bytes you want to reallocate.

Singly Linked List

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



SLL



SLL with a Header

Basic operations on a singly-linked list are:

1. Insert – Inserts a new node in the list.
2. Delete – Deletes any node from the list.
3. Find – Finds the position(address) of any node in the list.
4. FindPrevious - Finds the position(address) of the previous node in the list.
5. FindNext- Finds the position(address) of the next node in the list.
6. Display-display the date in the list
7. Search-find whether a element is present in the list or not

Declaration of Linked List

```
void insert(int X,List L,position P);
void find(List L,int X); void
delete(int x , List L); typedef
struct  node  *position;
position  L,p,newnode;
struct node
{
        int data;
        position next;
};
```

Creation of the list:

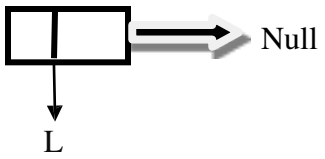
This routine creates a list by getting the number of nodes from the user. Assume n=4 for this example.

```
void create()
{
int i,n;
L=NULL;
newnode=(struct node*)malloc(sizeof(struct node));
printf("\n Enter the number of nodes to be inserted\n");
scanf("%d",&n);
printf("\n Enter the data\n");
scanf("%d",&newnode->data);
newnode->next=NULL;
L=newnode;
p=L;
for(i=2;i<=n;i++)
{
newnode=(struct node *)malloc(sizeof(struct node));
scanf("%d",&newnode->data);
newnode->next=NULL;
```

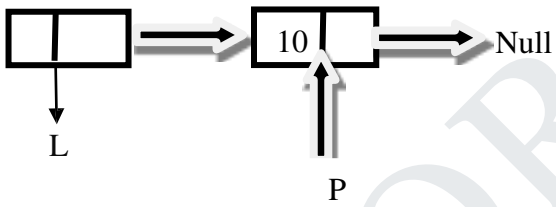
```
p->next=newnode;
p=newnode;
}
}
```

Initially the list is empty

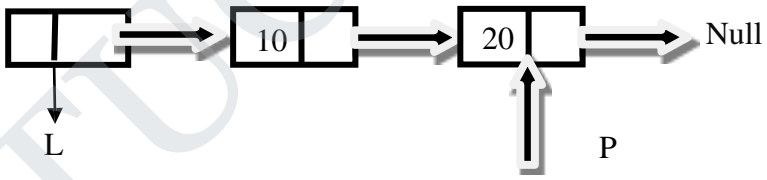
List L



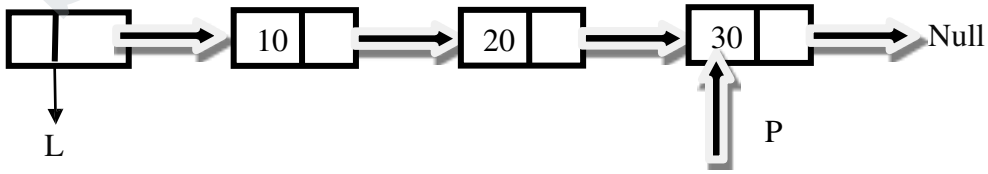
Insert(10,List L)- A new node with data 10 is inserted and the next field is updated to NULL. The next field of previous node is updated to store the address of new node.



Insert(20,L) - A new node with data 20 is inserted and the next field is updated to NULL. The next field of previous node is updated to store the address of new node.



Insert(30,L) - A new node with data 30 is inserted and the next field is updated to NULL. The next field of previous node is updated to store the address of new node.



Case 1:Routine to insert an element in list at the beginning

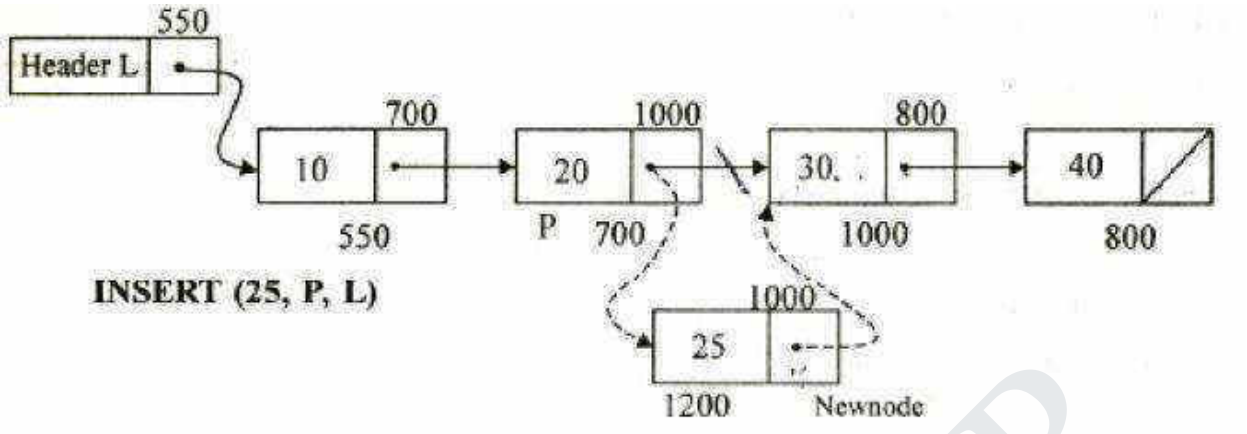
```
void insert(int X, List L, position p)
{
        p=L;
        newnode=(struct node*)malloc(sizeof(struct node));
        printf("\nEnter the data to be inserted\n");
    scanf("%d",&newnode->data);
    newnode->next=L;
    L=newnode;
}
```

Case 2:Routine to insert an element in list at Position

This routine inserts an element X after the position P.

```
Void Insert(int X, List L, position p)
{
    position newnode;
    newnode =(struct node*) malloc( sizeof( struct node ));
    if( newnode = = NULL )
        Fatal error( " Out of Space " );
    else
      {
        Newnode -> data = x ;
        Newnode -> next = p ->next ;
        P -> next = newnode ;
      }
}
```

Insert(25,L, P) - A new node with data 25 is inserted after the position P and the next field is updated to NULL. The next field of previous node is updated to store the address of new node.



Case 3:Routine to insert an element in list at the end of the list

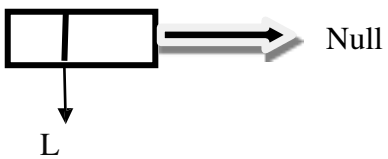
```
void insert(int X, List L, position p)

{
p=L;
    newnode=(struct node*)malloc(sizeof(struct node));
    printf("\nEnter the data to be inserted\n");
    scanf("%d",&newnode->data);
    while(p->next!=NULL)
    p=p->next;
    newnode->next=NULL;
    p->next=newnode;
    p=newnode;
}
```

Routine to check whether a list is Empty

This routine checks whether the list is empty .If the lis t is empty it returns 1

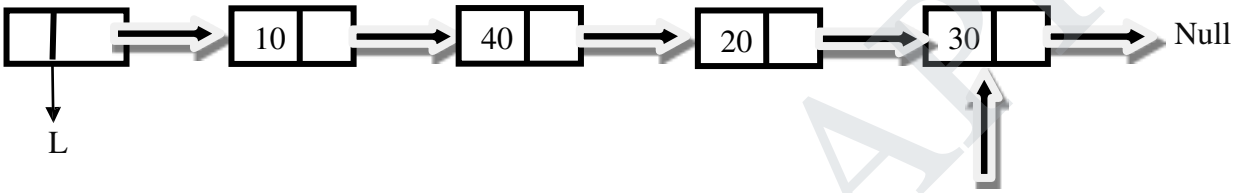
```
int IsEmpty( List L )
{
  if ( L -> next = = NULL )
        return(1);
}
```



Routine to check whether the current position is last in the List

This routine checks whether the current position p is the last position in the list. It returns 1 if position p is the last position

```
int IsLast(List L , position p)
{
        if( p -> next= =NULL)
                return(1);
}
```

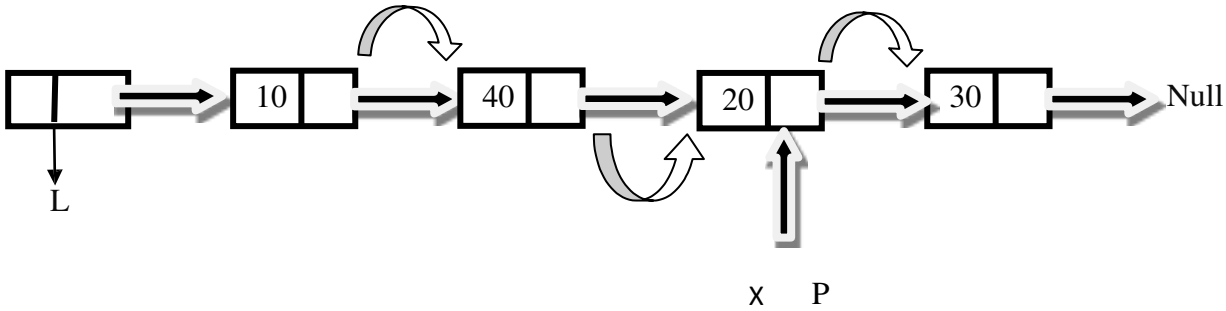


Routine to Find the Element in the List:

This routine returns the position of X in the list L

```
position find(List L, int X)
{
        position p;
        p=L->next;
        while(p!=NULL && p->data!=X)
                p=p->next;
        return(p);
}
```

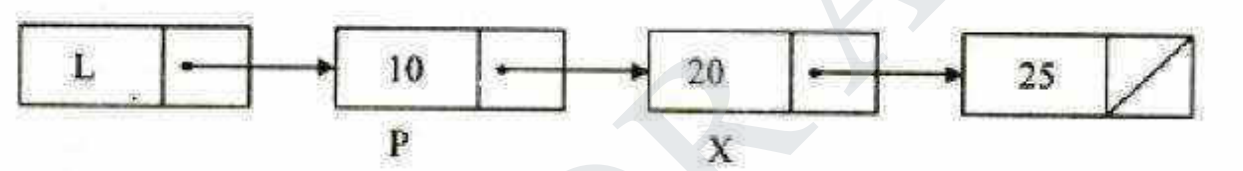
Find(List L, 20) - To find an element X traverse from the first node of the list and move to the next with the help of the address stored in the next field until data is equal to X or till the end of the list



Find Previous

It returns the position of its predecessor.

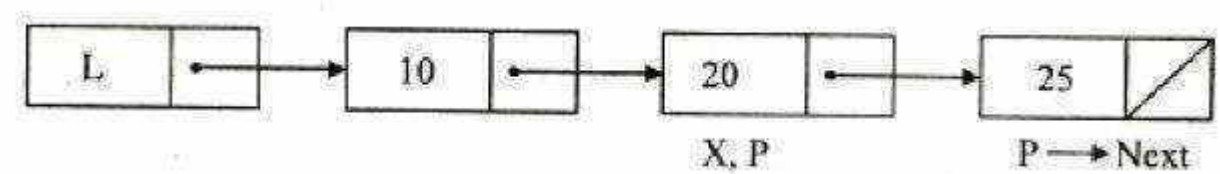
```
position FindPrevious (int X, List L)
{
        position p;
        p = L;
        while( p -> next ! = NULL && p -> next -> data! = X )
                p = p -> next;
        return P;
}
```



Routine to find next Element in the List

It returns the position of successor.

```
    void FindNext(int X, List L)
    {
        position P;
        P=L->next;
        while(P!=NULL && P->data!=X)
                P = P->next;
        return P->next;
}
```



Routine to Count the Element in the List:

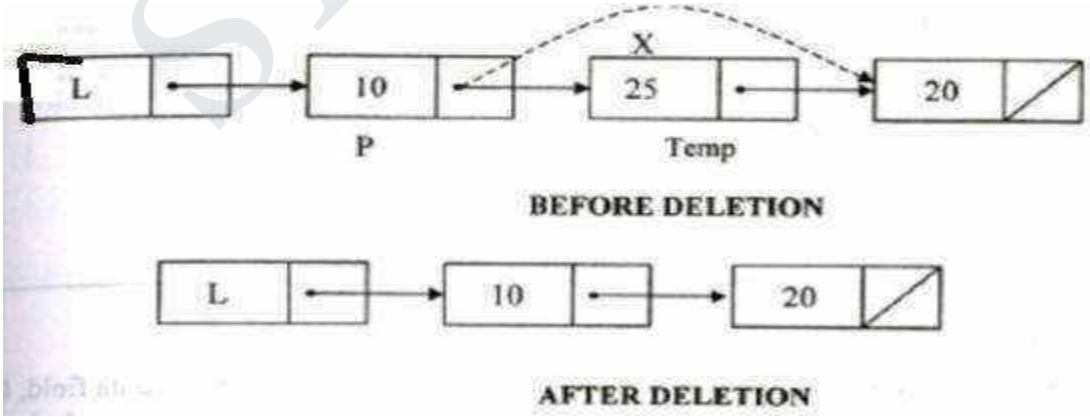
This routine counts the number of elements in the list

```
 void count(List L)
 {
        P = L -> next;
        while( p != NULL )
        {
                count++;
                p = p -> next;
        }
  print count;
}
```

Routine to Delete an Element in the List:

It delete the first occurrence of element X from the list L

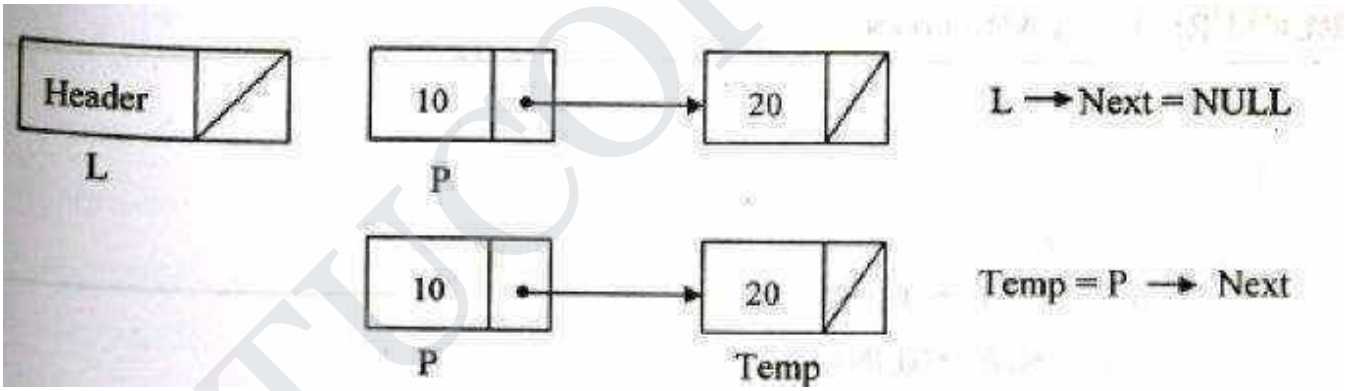
```
   void Delete( int x , List L){
        position p, Temp;
        p = FindPrevious( X, L);
        if( ! IsLast (p, L)){
                temp = p -> next;
                P -> next  =  temp -> next;
                free ( temp );
          }}
```



Routine to Delete the List

This routine deleted the entire list.

```
void Delete_list(List L)
    {
        position P,temp;
        P=L->next;
        L->next=NULL;
        while(P!=NULL)
        {
                temp=P->next;
                free(P);
                P=temp;
        }
}
```



Program 1: Implementation of Singly linked List	Output

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void create();
void display();
void insert();
void find();
void delete();
typedef struct node *position;
position L,p,newnode;
struct node
{
	1.create
	2.display
	3.insert
	4.find
	5.delete
	
	Enter your choice

```
int data;
position next;
};
void main()
{
int choice;
clrscr();
do
{
printf("1.create\n2.display\n3.insert\n4.find\n5.delete\n\n\n");
printf("Enter your choice\n\n");
scanf("%d",&choice);
switch(choice)
{
case 1:
          create();
          break;
case  2:
          display();
          break;
case  3:
          insert();
          break;
case  4:
          find();
          break;
case 5:
          delete();
          break;
case 6:
          exit(0);
}
}
while(choice<7);
getch();
}
void create()
{
int i,n;
L=NULL;
newnode=(struct node*)malloc(sizeof(struct node));
printf("\n Enter the number of nodes to be inserted\n");
scanf("%d",&n);
printf("\n Enter the data\n");
scanf("%d",&newnode->data);
newnode->next=NULL;
```

```
1

Enter the number of nodes to be inserted
5
Enter the data
1
2
3
4
5
1.create
2.display
3.insert
4.find
5.delete
Enter your choice
2
1 -> 2 -> 3 -> 4 -> 5 -> Null
1.create
2.display
3.insert
4.find
5.delete


Enter your choice

3
```

```
L=newnode; p=L;
for(i=2;i<=n;i++)
{
newnode=(struct node *)malloc(sizeof(struct node));
scanf("%d",&newnode->data);
newnode->next=NULL;
p->next=newnode;
p=newnode;
}
}
void display()
 { p=L;
  while(p!=NULL)
  {
  printf("%d -> ",p->data);
  p=p->next;
  }
  printf("Null\n");
 }
void insert()
{
int ch;
        printf("\nEnter ur choice\n");
printf("\n1.first\n2.middle\n3.end\n");
scanf("%d",&ch);
switch(ch)
{
case 2:
        {
        int pos,i=1;
        p=L;
                newnode=(struct node*)malloc(sizeof(struct node));
                printf("\nEnter the data to be inserted\n");
        scanf("%d",&newnode->data);
        printf("\nEnter the position to be inserted\n");
        scanf("%d",&pos);
        newnode->next=NULL;
        while(i<pos-1)
          {
           p=p->next;
           i++;
          }
        newnode->next=p->next;
        p->next=newnode;
```

```
Enter ur choice

1.first
2.middle
3.end
1

Enter the data to be inserted
7
7 -> 1 -> 2 -> 3 -> 4 -> 5 -> Null
1.create
2.display
3.insert
4.find
5.delete

Enter your choice
```

```
            p=newnode;
            display();
            break;
            }
case 1:
            {
            p=L;
                  newnode=(struct node*)malloc(sizeof(struct node));
                  printf("\nEnter the data to be inserted\n");
            scanf("%d",&newnode->data);
            newnode->next=L;
            L=newnode;
            display();
            break;
            }
case 3:
            {
            p=L;
                  newnode=(struct node*)malloc(sizeof(struct node));
                  printf("\nEnter the data to be inserted\n");
            scanf("%d",&newnode->data);
            while(p->next!=NULL)
            p=p->next;
            newnode->next=NULL;
            p->next=newnode;
            p=newnode;
            display();
            break;
            }
}
}
void find()
{
int search,count=0;
printf("\n Enter the element to be found:\n");
scanf("%d",&search);
p=L;
while(p!=NULL)
{
if(p->data==search)
 {
  count++;
  break;
 }
p=p->next;
}
```

```
if(count==0)
printf("\n Element Not present\n");
else
printf("\n Element present in the list \n\n");
}
void delete()
{
position p,temp;
int x; p=L;
if(p==NULL)
{
printf("empty list\n");
}
else
{
printf("\nEnter the data to be deleted\n");
scanf("%d",&x);
if(x==p->data)
{ temp=p;
L=p->next;
free(temp);
display();
}
else
{
while(p->next!=NULL && p->next->data!=x)
{
p=p->next;
}
temp=p->next;
p->next=p->next->next;
free(temp);
display();
}
}
}
```

Advantages of SLL

1.The elements can be accessed using the next link

2.Occupies less memory than DLL as it has only one next field.

Disadvantages of SLL

1.Traversal in the backwards is not possible

2.Less efficient to for insertion and deletion.