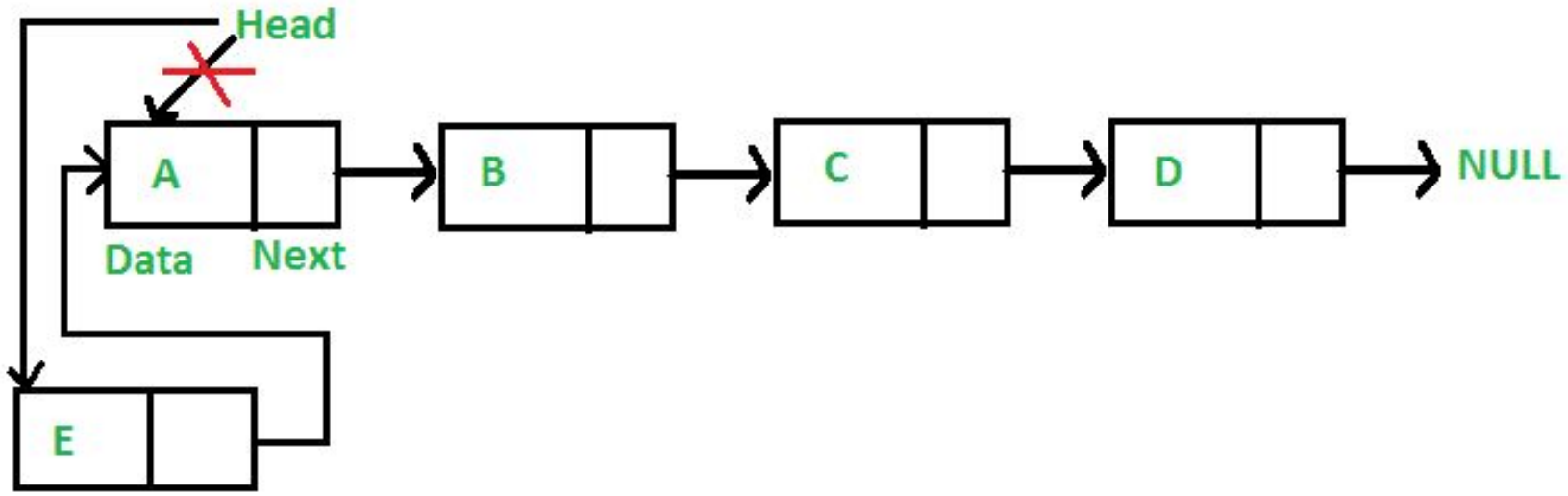
LINKED LIST

To insert a node at the start/beginning/front of a Linked List, we need to:

Make the first node of Linked List linked to the new node
Remove the head from the original first node of Linked List
Make the new node as the Head of the Linked List.



```
struct node {
   int data;
   struct node *next;
};
struct node *head = NULL, *current = NULL;

// display the list
void printList()
{
   struct node *p = head;

      //start from the beginning of list
   while(p != NULL) {
      printf(" %d ",p->data);
      p = p->next;
   }

}
```

```
//insertion at the beginning
void insertatbegin(int data)
{
  //create a new node
  struct node *newnode = (struct node*) malloc(sizeof(struct node));
  newnode->data = data;

  // point it to old first node
  newnode->next = head;

  //point first to new first node
  head = newnode;
}
```

```
void main()
{
  insertatbegin(12);
 printList();
insertatbegin(22);
 printList();
  insertatbegin(32);
printList();
insertatbegin(42);
printList();
}
```

Output

[12 ->]

[22 -> 12 ->]

[32 -> 22 -> 12 ->]

[42 -> 32 -> 22 -> 12 ->]

```
void insertatend(int data)
   //create a newnode
   struct node *newnode;
   newnode= (struct node*) malloc(sizeof(struct node));
   newnode->data = data;
   struct node *save = head;

   // point it to last node by traversing
   while(save->next != NULL)
      save = save->next;

   //link  to new first node
   save->next = newnode;
}
```

```
void main()
{
 insertatbegin(12);
 printList();
insertatbegin(22);
 printList();
insertatbegin(32);
 printList();
insertatbegin(42);
printList();
 insertatend(30);
 printList();
 insertatend(44);
printList();
 insertatbegin(50);
   printList();
}
```

```
[ 12 ->]
[ 22 -> 12 ->]
[ 32 -> 22 -> 12 ->]
[ 42 -> 32 -> 22 -> 12 ->]
[ 42 -> 32 -> 22 -> 12 -> 30 ->]
[ 42 -> 32 -> 22 -> 12 -> 30 -> 44 ->]
[ 50 -> 42 -> 32 -> 22 -> 12 -> 30 -> 44 ->]
```

```
void insertafternode(struct node *list, int data)
{
   struct node *lk = (struct node*) malloc(sizeof(struct node));
   lk->data = data;
   lk->next = list->next;
   list->next = lk;
}
```

```
void main()
{
   insertatbegin(12);
   insertatbegin(22);
   insertatend(30);
   insertatend(44);
   insertatbegin(50);
   insertafternode(head->next->next, 33);
   printf("Linked List: ");
```

```
[ 12 ->]
[ 22 -> 12 ->]
[ 22 -> 12 -> 30 ->]
[ 22 -> 12 -> 30 -> 44 ->]
[ 50 -> 22 -> 12 -> 30 -> 44 ->]
 Linked List:
[ 50 -> 22 -> 12 -> 33 -> 30 -> 44 ->]
```

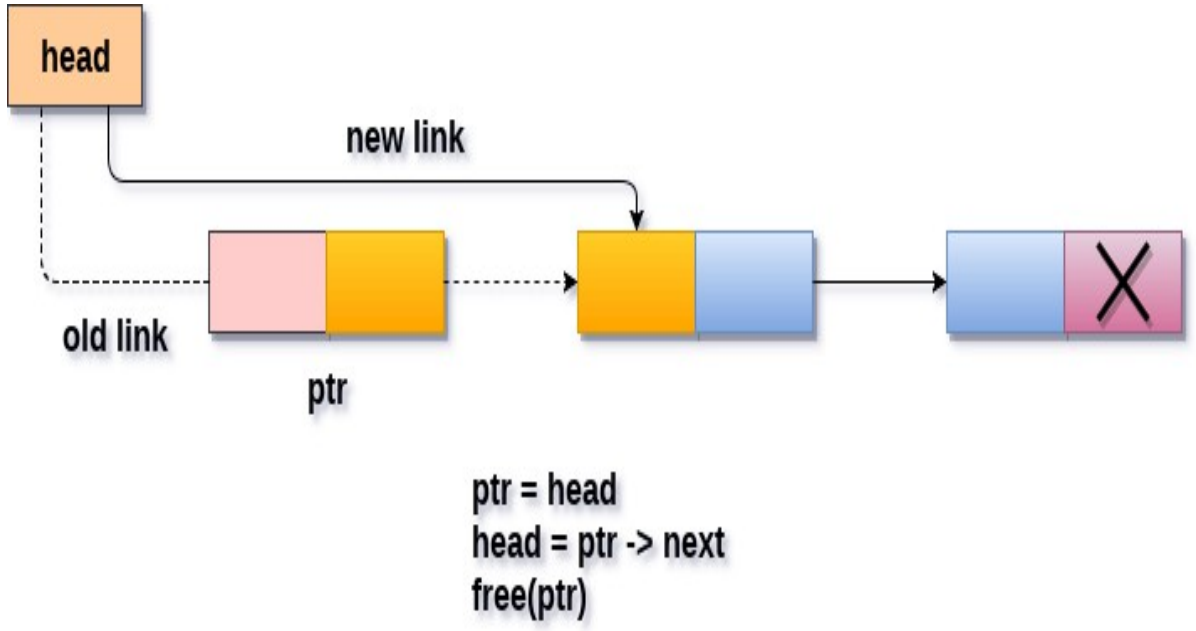
```
void deleteatbegin()
{
   head = head->next;
// how do we free the deleted node?
}

void deleteatend()
{
   struct node *linkedlist = head;
   while (linkedlist->next->next != NULL)
      linkedlist = linkedlist->next;
   linkedlist->next = NULL;
}
```

[12 ->]
[22 -> 12 ->]
[22 -> 12 -> 30 ->]
[22 -> 12 -> 30 -> 44 ->]
[50 -> 22 -> 12 -> 30 -> 44 ->]
Linked List:
[50 -> 22 -> 12 -> 33 -> 30 -> 44 ->]
Linked List after deleting first node:
[22 -> 12 -> 33 -> 30 -> 44 ->]
Linked List after deleting last node
[22 -> 12 -> 33 -> 30 ->]
Linked List after deletion of given node 12
[22 -> 33 -> 30 ->]

Delete a node at the front

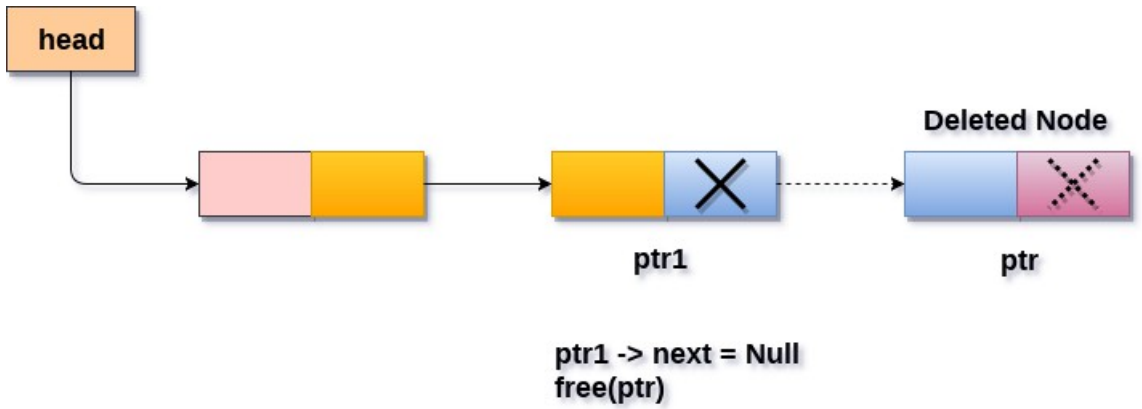
```
void Pop()
  {
    struct node *ptr;
    if(head == NULL)
    {
      printf("\nList is empty");
    }
    else
    {
      ptr = head;
      head = ptr->next;
      free(ptr);
      printf("\n Node deleted from the begining ...");
    }
  }
```



Delete a node at the end

```
void end_delete()
  {
    struct node *ptr,*ptr1;
    if(head == NULL)
    {
      printf("\nlist is empty");
    }
    else if(head -> next == NULL)
    {
      free(head); head = NULL;
     printf("\nOnly node of the list deleted ...");
    }

    else
    {
      ptr = head;
      while(ptr->next != NULL)
        {
          ptr1 = ptr;
          ptr = ptr ->next;
        }
        ptr1->next = NULL; free(ptr);
        printf("\n Deleted Node from the last ...");
      }
    }
```



Singly Linked List Implementation

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int info;
   struct node *next;
};
typedef struct node *NODE;
NODE insertFront(NODE first);
NODE insertRear(NODE first);
NODE insertAfter(NODE first);
NODE insertBefore(NODE first);
NODE insertAtPos(NODE first);
NODE deleteFront(NODE first);
NODE deleteRear(NODE first);
NODE deleteAfterEle(NODE first);
NODE deleteBeforeEle(NODE first);
NODE deleteElement(NODE first);
NODE deletePos(NODE first);
void display(NODE first);
```

Singly Linked List Implementation

```
void main()
{
   NODE first=NULL;
   int choice;
   while(1)
   {
      printf("\n\n*****Singly linked list implementation*****");
      printf("\n 1. Insert Front \n 2. Insert rear \n 3. Insert After \n
                        4. Insert Before \n 5. Insert At Position \n
                        6. Delete Front \n 7. Delete Rear \n
                        8. Delete After \n 9. Delete Before \n 10. Delete Element \n
                        11. Delete At Position \n 12. Display \n 13. Exit");
      printf("\n\t**********");
      printf("\nEnter your choice ");
      scanf("%d",&choice);
```

```
*****Singly linked list implementation*****
 1. Insert Front
 2. Insert rear
 3. Insert After
 4. Insert Before
 5. Insert At Position
 6. Delete Front
 7. Delete Rear
 8. Delete After
 9. Delete Before
 10. Delete Element
 11. Delete At Position
 12. Display
 13. Exit
      **********
Enter your choice
```

Singly Linked List Implementation

```
switch (choice)
        {
        case 1: first=insertFront(first); break;
        case 2: first=insertRear(first); break;
        case 3: first=insertAfter(first); break;
        case 4: first=insertBefore(first); break;
        case 5: first=insertAtPos(first); break;
        case 6: first=deleteFront(first); break;
        case 7: first=deleteRear(first); break;
        case 8: first=deleteAfterEle(first); break;
        case 9: first=deleteBeforeEle(first); break;
        case 10: first=deleteElement(first); break;
        case 11: first=deletePos(first); break;
        case 12: display(first); break;
        case 13: printf("\n Program exits now"); exit(0);
        default: printf("enter valid choice");
        }
    }
}
```

Display

```
void display(NODE first)
{
   NODE cur;
   if(first==NULL)
   printf("No elements to display");
   else
   {
      cur=first;
      printf("\n Elements of Singly linked list are:\t");
      while(cur!=NULL)
      {
         printf("%d\t",cur->info);
         cur=cur->next;
      }
   }
}
```

Insert in Front

```
NODE insertFront(NODE first)
{
   NODE temp=NULL;
   temp=(NODE)malloc(sizeof(struct node));
   if (temp==NULL)
   {
      printf("Insufficient memory");
      return first;
   }
   printf("\n Enter element to be inserted");
   scanf("%d",&temp->info);
   temp->next=NULL;
   if(first==NULL)
   first=temp;
   else    {
      temp->next=first;
      first=temp;
      return first;
   }
}
```

Insert at Rear

```
NODE insertRear(NODE first)
{
   NODE temp=NULL,cur=NULL;
   temp=(NODE)malloc(sizeof(struct node));
   if (temp==NULL)
   {
      printf("Insufficient memory");
      return first;
   }
   printf("\n Enter element to be inserted");
   scanf("%d",&temp->info);
   temp->next=NULL;
   if(first==NULL)
        first=temp;
   else
   {
      cur=first;
      while(cur->next!=NULL)
      {
         cur=cur->next;
      }
      cur->next=temp;
   }    return first;   }
```

Insert After an item

```
NODE insertAfter(NODE first)
{
   NODE temp=NULL,cur=NULL;
   temp=(NODE)malloc(sizeof(struct node));
   if (temp==NULL)
   {
      printf("Insufficient memory");
      return first;
   }
   int ele,item;
   if(first==NULL)
   {
      printf("linked list is empty");
      return first;
   }
      printf("\n Enter element after which new node to be inserted");
      scanf("%d",&ele);
```

Insert After an item

```
cur=first;
      while(cur!=NULL&&cur->info!=ele)
         cur=cur->next;
      if(cur==NULL)
      {
         printf("Element not found");
         return first;
      }
      printf("\nEnter element to be inserted");
      scanf("%d",&item);
      temp->info=item;
      temp->next=NULL;
      temp->next=cur->next;
      cur->next=temp;
      return first;
}
```

```
Elements of Singly linked list are:   2     1     3

*****Singly linked list implementation*****
 1. Insert Front
 2. Insert rear
 3. Insert After
 4. Insert Before
 5. Insert At Position
 6. Delete Front
 7. Delete Rear
 8. Delete After
 9. Delete Before
 10. Delete Element
 11. Delete At Position
 12. Display
 13. Exit
      ***********
Enter your choice 3
 Enter element after which new node to be inserted 1
Enter element to be inserted5

Enter your choice 12

 Elements of Singly linked list are:   2     1     5     3
```

Insert Before an item

```
NODE insertBefore(NODE first)
{
   NODE temp=NULL, cur=NULL, prev=NULL;
   int ele,item;
   temp=(NODE)malloc(sizeof(struct node));
   if (temp==NULL)
   {
      printf("Insufficient memory");
      return first;
   }
   printf("\n Enter element before which new node to be inserted");
   scanf("%d",&ele);
```

Insert Before an item

Elements of Singly linked list are: 2 1 3

```
cur=first;
      while(cur!=NULL&&cur->info!=ele)
      {   prev=cur;
         cur=cur->next;       }
      if(cur==NULL)
      {     printf("Element not found");
          return first;       }
      printf("Element to be inserted");
      scanf("%d",&item);
      temp->info=item;
      temp->next=NULL;
      temp->next=cur;
      if(prev!=NULL)
         prev->next=temp;
      else
         first=temp;
      return first;
}
```

Insert at Position

```
NODE insertAtPos(NODE first)
{
   NODE temp=NULL,cur=NULL;
   temp=(NODE)malloc(sizeof(struct node));
   if (temp==NULL)
   {
      printf("Insufficient memory");
      return first;
   }
   int ele,pos;
   if(first==NULL)
   {
      printf("linked list is empty");
      return first;
   }
```

Insert at Position

```
printf("Enter pos at which new element to be inserted ");
    scanf("%d",&pos);
    printf("Enter element to be inserted at pos ");
    scanf("%d",&ele);
    if(pos==1)
    {
        first=insertFront(first);
        return first;
    }
```

Insert at Position

```
cur=first;
  int i=1;
  while(cur!=NULL&&i<pos-1)
  {
     cur=cur->next;
     i++;
  }
  if(cur==NULL)
     {
        printf("Element not found");
        return first;
     }
     temp->info=ele;
     temp->next=NULL;
     temp->next=cur->next;
     cur->next=temp;
     return first;
}
```

Elements of Singly linked list are: 2 6 1 5 3

*****Singly linked list implementation*****

1. Insert Front
2. Insert rear
3. Insert After
4. Insert Before
5. Insert At Position
6. Delete Front
7. Delete Rear
8. Delete After
9. Delete Before
10. Delete Element
11. Delete At Position
12. Display
13. Exit

****

Enter your choice 5

Enter pos at which new element to be inserted 4

Enter element to be inserted at pos 7

Enter your choice 12

Elements of Singly linked list are: 2 6 1 7 5 3

Delete Front

```
NODE deleteFront(NODE first)
{
    NODE temp=NULL;
    if(first==NULL)
    {
        printf("Linked List is empty, create Linked list");
        return first;
    }
    temp=first;
    first=first->next;
    printf("Element being deleted is %d",temp->info);
    free(temp);
    return first;
}
```

Delete Rear

```
NODE deleteRear(NODE first)
{
   NODE cur=NULL,prev=NULL;
   prev=(NODE)malloc(sizeof(struct node));
   if (prev==NULL)
   {
      printf("Insufficient memory");
      return first;
   }
   if(first==NULL)
   {
      printf("LL is empty");
      return first;
   }
```

Delete Rear

```
cur=first;
    prev=NULL;
    while(cur->next!=NULL)
    {
        prev=cur;
        cur=cur->next;
    }
    prev->next=NULL;
    printf("Element being deleted is %d",cur->info);
    free(cur);
    return first;
}
```

Delete Element

```
NODE deleteElement(NODE first)
{
    NODE cur = NULL, prev = NULL;
    int item;
    if(first==NULL)
    {
        printf("\nThe list is empty\n");
        return first;
    }
    printf("\nEnter the element to be deleted :");
    scanf("%d",&item);
```

Delete Element

```
cur=first;
   while(cur!=NULL && cur->info!=item)
   {   prev=cur;
       cur=cur->next;    }
   if(cur==NULL)
   {   printf("\nElement to be deleted doesnt exist in the list\n");
       return first;    }
   if(prev==NULL)
   {   first = deleteFront(first);
       return first;    }
   prev->next = cur->next;
   printf("\nElement being deleted is : %d\n", cur->info);
   free(cur);
   return first;
}
```

Delete at Position

```
NODE deletePos(NODE first)
{
   NODE cur = NULL, prev = NULL;
   int pos, k;
   if(first==NULL)
   {       printf("\nThe list is empty.. no elements to delete...\n");
          return first;
   }
   printf("\nEnter the position of element to be deleted :");
   scanf("%d",&pos);
   if(pos==1)
   {
      first = deleteFront(first);
      return first;
   }
```

Delete at Position

```
cur=first;
   k=1;
   while(cur!=NULL && k<pos)
   {  prev=cur;
      cur=cur->next;
      k++;
   }
   if(cur==NULL)
   {   printf("\nPosition doesnt exist in the list\n");
      return first;    }
   prev->next = cur->next;
   printf("\nElement being deleted is : %d\n", cur->info);
   free(cur);
   return first;
   }
```

Delete Before Element

```
NODE deleteBeforeEle(NODE first)
{  NODE cur = NULL, prev = NULL, pprev = NULL;
   int ele;
   if(first==NULL)
   {   printf("\nThe list is empty.. no elements to delete...\n");
       return first;
   }
   printf("\nEnter an element whose left element to be deleted :");
   scanf("%d",&ele);
   cur=first;
   while(cur!=NULL && cur->info!=ele)
   {    pprev = prev;
       prev=cur;
       cur=cur->next;
   }
```

Delete Before Element

```
 if(cur==NULL)
    {
       printf("\nElement doesnt exist in the list\n");
       return first;
    }
    if(pprev==NULL)
    {
       first = deleteFront(first);
       return first;
    }
    pprev->next = prev->next;
    printf("\nElement being deleted is : %d\n", prev->info);
    free(prev);
    return first;
}
```

Delete After Element

```
NODE deleteAfterEle(NODE first)
{
    NODE cur = NULL, temp = NULL;
    int ele;
    if(first==NULL)
    {   printf("\nThe list is empty.. no elements to delete...\n");
        return first;
    }
    printf("\nEnter an element whose right element to be deleted :");
    scanf("%d",&ele);
    cur=first;
    while(cur!=NULL && cur->info!=ele)
    {       cur=cur->next;
    }
```

Delete After Element

```
 if(cur==NULL)
    {
        printf("\nElement doesnt exist in the list\n");
        return first;
    }
    if(cur->next == NULL)
    {   printf("\nNo elements to delete after the given element...");
        return first;
    }
    temp = cur->next;
    cur->next = temp->next;
    printf("\nElement being deleted is : %d\n", temp->info);
    free(temp);
    return first;
}
```