Insert at the beginning of a singly linked list

The interesting thing about adding element in the beginning of a singly linked list is the order in which you add the elements is the reverse of the actual linked list created.

For eg :- you inserted 30 then 20 then 10
The linked list will be created will be in the order 10->20->30

```
Node *InsertBegin(Node * head, int x) {
        Node *temp = new Node(x);
        temp->next = head;
        return temp;
}

int main( ) {
        Node * head = NULL;
        head = InsertBegin(head,20);
        head = InsertBegin(head,30);
}
```

1. Create a new element
2. Store it in using temp
3. Make the next of temp = head
4. Return the temp
5. Make head in main as null
6. Then store the function in head

Insert at the end of singly linked list

```
Node *insertend(Node * head , int x) {
        Node * temp = new Node(x);
        if(head == NULL) return temp;

        Node *curr  = head;
        while(curr -> next != NULL) {
                Curr = curr->next;
        }
        Curr ->next = temp;
        return head;
}
```

1. Create a temp
2. Check the head condition
3. Create a curr pointer and traverses it till the curr-> next it becomes null
4. Then store the temp value in the curr->next
5. Then return head back

Delete a node from beginning of singly linked list

```
Node *delhead(Node *head) {
        if(head == NULL) return NULL;

        Else{
                Node *temp = head->next;
                delete(head);
                return temp;
        }
int main( ) {
        Node *head = new Node(30);
        head = delhead(head);
}
```

1. Create a temp pointer variable and store the head of next
2. Then deallocate the memory from head
3. Return the temp
4. It will store the new head

Delete a node from end of singly linked list

Deleting a node from the end of a linked list first thing we should take care of is the head pointer will not going to change like in the above cases
Head pointer will be changed only when there are less than two elements in a linked list - note that

```
Node *deltail(Node *head) {
        if(head == NULL) return NULL;     // corner case when LL is empty
        if(head->next == NULL)            // corner case when the LL has only single element
                {
                delete head;
                return NULL;
                }
```

```
        Node * curr = head;
        while(curr->next->next != NULL) {
                curr = curr->next;
        }
        delete(curr->next);
        curr->next = NULL;
        return head;
```

Doubly linked list insert at the beginning of DLL

```
Node *insertbegin(Node *head , int x){

        Node *temp = new Node(data);
        temp->next = head;
        if(head != NULL)
        head->prev = temp;
        Return temp;

}
```

Insert at the end of DLL

```
Node *insertEnd(Node *head,int data){
   Node *temp=new Node(data);
   if(head==NULL)return temp;
   Node *curr=head;
   while(curr->next!=NULL){
      curr=curr->next;
   }
   curr->next=temp;
   temp->prev=curr;
   return head;
}
```

Delete the head of DLL

```
Node *delHead(Node *head){
   if(head==NULL)return NULL;
   if(head->next==NULL){
      delete head;
      return NULL;
   }
   else{
      Node *temp=head;
      head=head->next;
      head->prev=NULL;
      delete temp;
      return head;
   }
```

Delete the node from end of DLL

```
Node *delLast(Node *head){
   if(head==NULL)return NULL;
   if(head->next==NULL){
      delete head;
      return NULL;
   }
   Node *curr=head;
   while(curr->next!=NULL)
      curr=curr->next;
   curr->prev->next=NULL;
   delete curr;
   return head;

}
```

Traversal of a Circular linked list

```
void printlist(Node *head) {
   if(head==NULL)return;
   cout<<head->data<<" ";
   for(Node *p=head->next;p!=head;p=p->next)
      cout<<p->data<<" ";
}
```

Using do while loop

```
void printlist(Node *head){
   if(head==NULL)return;
   Node *p=head;
   do{
      cout<<p->data<<" ";
      p=p->next;
   }while(p!=head);
}
```

Find middle of a linked list

We have two approaches for this first is the naive approach and the other is the efficient approach

First talking about the naive approach which is first we traverse the whole linked list till null then We get the count for it then we traverse again till the count/2 and then cout the the count

```
void printMiddle(Node * head){
   if(head==NULL)return;
   int count=0;
   Node *curr;
   for(curr=head;curr!=NULL;curr=curr->next)
      count++;
   curr=head;
   for(int i=0;i<count/2;i++)
      curr=curr->next;
   cout<<curr->data;
}
```

Efficient approach
In this we will keep fast and slow pointers and increment the slow pointer one and fast moves 2 moves at the time this will lead to get the middle of linked list we will have to keep two separate cases for even and the odd values of linked list

```
void printMiddle(Node * head){
   if(head==NULL)return;
   Node *slow=head,*fast=head;
   while(fast!=NULL && fast->next!=NULL){
      slow=slow->next;
      fast=fast->next->next;
   }
   cout<<slow->data;
}
```

Nth node from the end of linked list