Linked List:-

When working on linked list code, it's a good habit to remember to check the empty list case to verify that it works too. Sometimes the empty list case works the same as all the cases, but sometimes it requires some special case code. No matter what, it's a good case to at least think about.

```
void demo1() { //Inefficient way of creating a linked list
//Here 3 nodes having values 1, 2, 3, are created and head points to the first node
        node a, b, c; //You have to know how many nodes you need and allot space for them!
        a.num = 1;
        a.next = &b;
        b.num = 2;
        b.next = &c;
        c.num = 3;
        c.next = 0;

        node* p = &a;
        show(p); //function to display the linked list
//The above is not a good way to manage a linked lists
//It is better to allot space as and when required, as shown in main()
}
```

Efficient way to create a linked list:-

```
struct node
{int num;
node* next;
};
int main()
{
node* head;
head = new node;
head->num=10;
head->next=NULL //or 0;

node *ptr1, *ptr2;
        ptr1 = head;
        for (int i=11; i<14; i++) {
                ptr2 = new node;
                ptr1->next = ptr2;
                ptr2->num = i;
                ptr2->next = 0;
                ptr1 = ptr2;
        }
```

The malloc() function from C, still exists in C++, but it is recommended to avoid using malloc() function. The main advantage of new over malloc() is that new doesn't just allocate memory, it constructs objects which is prime purpose of C++.

Use as
node* curr;
curr=new node;

```
//allocated new memory of node size
```

```
void WrongPush(struct node* head, int data)
{ struct node* newNode = malloc(sizeof(struct node));
newNode->data = data;
newNode->next = head; head = newNode; // NO this line does not work! }
 void WrongPushTest()
{ List head = BuildTwoThree();
WrongPush(head, 1); // try to push a 1 on front -- doesn't work
```

My code to 1. Show head 2. Print Linked List 3. Remove head item 4. Length 5. Find 6. Insert 7. Append

```
#include <iostream>


using namespace std;

struct node
   {int num;
   node* next;
   };

void show(node* head)
   {cout<<head->num;}

node* insert(node* head, int item)
   {node* ptr;
   ptr = new node;
   ptr->num=item;
   ptr->next=head;
   head = ptr;
   return head;
   }

void append(node* head, int item1)
   {node *ptr;
   ptr = new node;
   ptr->num=item1;
   ptr->next=NULL;
   node *bottom;
   bottom = new node;
   bottom = head;
   while(bottom->next!=NULL)
   {bottom=bottom->next;}
   bottom->next=ptr;
   bottom->num=item1;
  //ptr->num=item1;
   }//append does not need to return anything because pointer to head of list is not changed.

node* remove (node* head)
   {head=head->next;
   return head;}

int length (node* head)
   {int count;
   node* curr;
   curr=new node;
   curr=head;
   while (curr->next!=NULL)
   {curr=curr->next;
    count++;
   }
   return count;
   }

node* find(node* head, int item)
   {node* curr;
   curr=new node;
   curr=head;
   int count=1;
    while(curr->num!=item)
   {curr=curr->next;
   count++;
   }
   cout<<endl<<"At index "<<count<<endl;
   return curr;

   }

void print(node* head)
   {
   node* curr;
   curr = new node;
   curr=head;
   while(curr->next!=NULL)
   {cout<<curr->num<<" ";
   curr=curr->next;
   }

}
```

```
int main ()
   {node* head;
    node* ptr;
    ptr = new node;
    head = ptr;
    for (int i=1;i<=5;i++)
    {node* ptr2;
    ptr2 = new node;
    ptr2->next=NULL;
    ptr->num=i;
    ptr->next=ptr2;
    ptr=ptr2;
    }
    print(head);
    cout<<endl;
    show(head);
    cout<<endl;
    head=insert(head,0);
    print(head);
    append(head, 6);
    cout<<endl;
    print (head);
    cout<<endl;
    head=remove(head);
    print(head);
    cout<<endl;
    cout<<length(head);
    node* add=find(head,5);
    cout<<"at address "<<add;
return 0;
}
```

Push Function in a linked list (As as stack)

```
#include <iostream>
using namespace std;
struct node
   {int data;
   node *next; };

void push(node** headref, int data)                    // Sending a pointer to a pointer
   {node *ptr;
   ptr=new node;
   ptr->data=data;
   ptr->next=*headref;                                 // Dereferencing a pointer
   *headref=ptr;}

int main ()
   {node *head;
    head=NULL;
    for (int i=1;i<=10;i++)
   {
    push(&head,i);                                          // Push function called
   }
    node *curr;
    curr=head;
    int count = 0;
    while (curr!=NULL)
      {cout<<curr->data<<"->";
      curr=curr->next;
       count++;
      }
     cout<<"NULL";
    cout<<"\nSize "<<count;
    return 0;
}
```

Output :-

```
10->9->8->7->6->5->4->3->2->1->NULL
Size 10
```

Segmentation fault in Linked List means somewhere forgotten
struct node* ptr = new node

Merge two sorted lists :-

```
Node* MergeLists(Node *headA, Node* headB)
{
   struct Node* curr;
   curr=new Node;
   curr->data=0;
   curr->next=NULL;
  struct Node* head;
   head=curr;
  while (1)
    {if (headA==NULL)
         {curr->next=headB;
          break;
         }
     else if (headB==NULL)
        {curr->next=headA;
        break;
        }
     if (headA->data<=headB->data)
         {curr->next=headA;
         headA=headA->next;
         }
     else
         {curr->next=headB;
         headB=headB->next;
```

```
            }
        curr=curr->next;
        }

    head=head->next;
    return head;

}
```

Reverse Print:-

```
void ReversePrint(Node *head)
{
   if (head!=NULL&&head->next!=NULL)
    {
    Node* prev;
    Node* curr;
    Node* tail;
    tail=head;
    curr=head;
    prev=head;
    Node* temp;
    while (curr!=NULL)
    {temp = curr->next;
    curr->next=prev;
    prev=curr;
    curr=temp;
    }
    tail->next=NULL;

    while(prev!=NULL)
        {cout<<prev->data<<endl;
        prev=prev->next;}
   }
}
```

<u>Do merge sort:-</u>