```
########## Singli liked list

############################
#include<bits/stdc++.h>
using namespace std;
class node{
   public:
   int data;
   node *next;
};

void addAtBegin(node **head,int n){
   node* tmp=new node();
   tmp->data=n;
   tmp->next=*head;
   *head=tmp;
}
void addAtLast(node**head,int val){
   node*tmp=new node();
   tmp->data=val;
   tmp->next=NULL;
   if(*head==NULL){
       *head=tmp;
   }
   else{
       node* tmp1=*head;
       while(tmp1->next!=NULL){
           tmp1=tmp1->next;
       }
       tmp1->next=tmp;
   }
}

void addAtPosition(node** head,int pos,int val){
     node*tmp=*head;
     for(int i=0;i<pos-1;i++){
       tmp=tmp->next;
         if(tmp==NULL){
             cout<<"The position is more more bigger so can't
insert"<<endl;
```

```
            //    return;
          }
      }
      node* final=new node();

       final->next=tmp->next;
        final->data=val;
        tmp->next=final;
}

void deletionAtBegin(node** head){
    node* tmp=*head;
    if(*head==NULL){
       cout<<"empty"<<endl;
    }
    else{
        *head=tmp->next;
        delete(tmp);
        return;
    }
}

void deleteAtEnd(node**head){
   node* tmp=*head;
   node* fn;
   if(*head==NULL){
       cout<<"empty"<<endl;
   }
   else{
       while(tmp->next->next!=NULL){
           tmp=tmp->next;
       }
       fn=tmp->next;
       delete(fn);
       tmp->next=NULL;


   }
}
```

```
// void deleteBySearch(node**head){
//     int n;
//     cout<<"Enter the element you want to delete : ";
//     cin>>n;
//     node*tmp=*head;
//     node* fn;
//     if(*head==NULL){
//         cout<<"sorry there is no elemnt to delete"<<endl;
//     }
//     else{
//         while(tmp->next=NULL){

//             if(tmp->data==n){
//                 fn=tmp->next;
//                 tmp->next=fn->next;
//                 delete(fn);
//                 return;
//             }
//             tmp=tmp->next;
//         }
//     }
// }

void count(node**head){
   node* tmp=*head;
   int c=0;
   while(tmp!=NULL){
       c++;
       tmp=tmp->next;
   }
   cout<<"NO of element is = "<<c<<endl;
}

void search(node**head){
   node*tmp=*head;
   int n,c=0;
   cout<<"enter the element that you want to search = ";
   cin>>n;

   while(tmp!=NULL){
```

```
        if(tmp->data==n){
           cout<<"element found!!"<<endl;
           break;
        }
        tmp=tmp->next;
    }
    if(tmp==NULL){
        cout<<"Not found"<<endl;
    }
}

void print(node**head){
    if(*head==NULL){
        cout<<"Empty"<<endl;
    }
    else{
        node* tmp=*head;
        while(tmp!=NULL){
            cout<<tmp->data<<"->";
            tmp=tmp->next;
        }
        cout<<endl;
    }
}




int main(){

  node* head=NULL;
  addAtBegin(&head,7);
  addAtBegin(&head,9);
  addAtBegin(&head,15);
  print(&head);
  addAtLast(&head,10);
  addAtLast(&head,11);
  addAtLast(&head,12);
  print(&head);
  addAtPosition(&head,3,100);
```

```
  print(&head);
  count(&head);
  search(&head);
   // addAtPosition(&head,11,100);
  deletionAtBegin(&head);
  print(&head);
  deletionAtBegin(&head);
  print(&head);
  deletionAtBegin(&head);
  print(&head);
  deleteAtEnd(&head);
  print(&head);
  deleteAtEnd(&head);
  print(&head);
//    deleteBySearch(&head);
//    print(&head);
  count(&head);



   return 0;
}
```

```
################### Doubly Linkd List ################
#include<bits/stdc++.h>
using namespace std;

// note that in ll position start from 1 not form 0 as in array

class node{
   public:
   int data;
   node* next;
   node* pre;

};
```

```
void print(node**head){
   if(*head==NULL){
       cout<<"empty"<<endl;
   }
   else{
       node*tmp=*head;
       while(tmp!=NULL){
           cout<<tmp->data<<" ";
           tmp=tmp->next;
       }
       cout<<endl;
   }
}

void count(node**head){
   if(*head==NULL){
       cout<<"empty"<<endl;
   }
   else{
       int c=0;
       node*tmp=*head;
       while(tmp!=NULL){
         c++;
           tmp=tmp->next;
       }
        cout<<"size is = "<<c++<<endl;
   }
}

void AddBegin(node**head,int val){
   // for details see programiz
  node* tmp=new node();
  tmp->data=val;
  tmp->next=(*head);
   tmp->pre=NULL;
   if(*head!=NULL){
       (*head)->pre=tmp;
   }
  *head=tmp;
```

```
}

void AddEnd(node**head,int val){
   node*tmp=new node();
   tmp->data=val;
   tmp->next=NULL;
   // cheque if the ll is empty
   if(*head==NULL){
       tmp->pre=NULL;
       *head=tmp;
       return;
   }
   // if not empty
   else{
       node* t=*head;
       //traverse till end
       while(t->next!=NULL){
           t=t->next;
       }
       // now t is in the lsat node
       t->next=tmp;
       tmp->pre=t;
   }
}

void AddAfter(node**head,int pos,int val){
        //allocate node for the given value
        node* tmp=new node();
        tmp->data=val;
        tmp->next=NULL;
        tmp->pre=NULL;

        //base case . cheque of it is >0
        if(pos<1){
            cout<<"position is invalid"<<endl;
        }
        // if the position is 1 then make new node as head
        else if(pos==1){
            tmp->next=*head;
            (*head)->pre=tmp;
```

```
            *head=tmp;
        }
        //traverse till the given position and the add on this position
        else{
            node* t=*head;
            for(int i=1;i<pos-1;i++){
                if(t!=NULL){
                    t=t->next;
                }
            }
            // now t is in the (pos-1) position
            // *** if the value of t is null that means this position is
            // greater then the size of ll


            // and if it is not null then the position belongs to the ll
            if(t!=NULL){
                tmp->next=t->next;
                tmp->pre=t;
                t->next=tmp;
                // if after new node ther are no element
                if(tmp->next!=NULL){
                    tmp->next->pre=tmp;
                }
            }
           //  *** case
             else{
                 cout<<"the pos is out of link list"<<endl;
             }
        }
}

void deletion(node**head,int pos){
   // delete first or startion node
   if(pos==1){
       node*tmp=*head;
       tmp->next->pre=NULL;
       *head=tmp->next;
       tmp->next=NULL;
       delete tmp;
```

```
    }
    // delete any node that can be second ,mid, last any one
    else{
        node*curr=*head;
        node*prev=NULL;


        int c=1;
        while(c<pos){
            prev=curr;
            curr=curr->next;
            c++;
        }
        //now curr is in the pos index and prev is in the (pos-1) index
        curr->pre=NULL;
        prev->next=curr->next;
        curr->next=NULL;
        delete curr;
    }
}



int main(){

    node* head=NULL;
    node*pre=NULL;
    AddBegin(&head,12);
    print(&head);
   AddBegin(&head,10);
    print(&head);
   AddBegin(&head,11);
    print(&head);
    count(&head);
    AddEnd(&head,112);
    print(&head);
    AddAfter(&head,3,21);
    print(&head);
    AddAfter(&head,7,21);
    print(&head);
```

```
        AddAfter(&head,5,21);
    print(&head);
    deletion(&head,3);
    print(&head);

   return 0;
}
```

```
#######################  L2  ####################3
#include <bits/stdc++.h>
using namespace std;

class node
{
public:
   int data;
   node *next;
};

void add(node **head, int data)
{
   node *tmp = new node();
   tmp->data = data;
   tmp->next = NULL;
   if (*head == NULL)
   {
       *head = tmp;
       return;
   }
   else
   {
       node *t = *head;
       while (t->next != NULL)
```

```
        {
            t = t->next;
        }
        t->next = tmp;
    }
}

int getlengt(node **head)
{
    int len = 0;
    while ((*head) != NULL)
    {
        len++;
        (*head) = (*head)->next;
    }
    return len;
}

void print(node **head)
{
    node *tmp = *head;
    while (tmp != NULL)
    {
        cout << tmp->data << " ";
        tmp = tmp->next;
    }
    cout << endl;
}

void reverseW1(node **head)
{

    stack<node *> s;
    node *tmp = *head;
    while (tmp->next != NULL)
    {
        s.push(tmp);
        tmp = tmp->next;
    }
    *head = tmp;
```

```
    while (!s.empty())
    {
        tmp->next = s.top();
        s.pop();
        tmp = tmp->next;
    }
    tmp->next = NULL;
}

void reverseW2(node **head_ref)
{
    node *temp = NULL;
    node *prev = NULL;
    node *current = (*head_ref);
    while (current != NULL)
    {
        temp = current->next;
        current->next = prev;
        prev = current;
        current = temp;
    }
    (*head_ref) = prev;
}

void getMedil1(node **head)
{

    // its mine creation
    int c = 0;
    node *t = *head;
    while (t != NULL)
    {
        c++;
        t = t->next;
    }
    int mid = c / 2;
    cout << mid << endl;
    node *n = *head;
    int i = 1;
    while (i < mid)
```

```
    {
        n = n->next;
        i++;
    }
    cout << n->data << endl;
}

void getMedil2(node *head_ref)
{
    node *slow = head_ref;
    node *fast = head_ref;
    if (head_ref != NULL)
    {
        while (fast != NULL && fast->next != NULL)
        {
            slow = slow->next;
            fast = fast->next->next;
        }
        cout << slow->data << endl;
    }
}

int main()
{
    node *head = NULL;
    add(&head, 12);
    add(&head, 15);
    add(&head, 10);
    add(&head, 16);
    add(&head, 14);
    // add(&head,17);
    print(&head);
    // reverseW2(&head);
    // print(&head);
    getMedil2(head);
    return 0;
}


######################  L2  ####################3
```

```
#include <bits/stdc++.h>
using namespace std;

class node
{
public:
   int data;
   node *next;
};

void add(node **head, int data)
{
   node *tmp = new node();
   tmp->data = data;
   tmp->next = NULL;
   if (*head == NULL)
   {
       *head = tmp;
       return;
   }
   else
   {
       node *t = *head;
       while (t->next != NULL)
       {
           t = t->next;
       }
       t->next = tmp;
   }
}

int getlengt(node **head)
{
   int len = 0;
   while ((*head) != NULL)
   {
       len++;
       (*head) = (*head)->next;
   }
   return len;
```

```
}

void print(node **head)
{
   node *tmp = *head;
   while (tmp != NULL)
   {
       cout << tmp->data << " ";
       tmp = tmp->next;
   }
   cout << endl;
}

void reverseW1(node **head)
{

   stack<node *> s;
   node *tmp = *head;
   while (tmp->next != NULL)
   {
       s.push(tmp);
       tmp = tmp->next;
   }
   *head = tmp;
   while (!s.empty())
   {
       tmp->next = s.top();
       s.pop();
       tmp = tmp->next;
   }
   tmp->next = NULL;
}

void reverseW2(node **head_ref)
{
   node *temp = NULL;
   node *prev = NULL;
   node *current = (*head_ref);
   while (current != NULL)
   {
```

```
        temp = current->next;
        current->next = prev;
        prev = current;
        current = temp;
    }
    (*head_ref) = prev;
}

void getMedil1(node **head)
{

    // its mine creation
    int c = 0;
    node *t = *head;
    while (t != NULL)
    {
        c++;
        t = t->next;
    }
    int mid = c / 2;
    cout << mid << endl;
    node *n = *head;
    int i = 1;
    while (i < mid)
    {
        n = n->next;
        i++;
    }
    cout << n->data << endl;
}

void getMedil2(node *head_ref)
{
    node *slow = head_ref;
    node *fast = head_ref;
    if (head_ref != NULL)
    {
        while (fast != NULL && fast->next != NULL)
        {
            slow = slow->next;
```

```
            fast = fast->next->next;
        }
        cout << slow->data << endl;
    }
}

int main()
{
    node *head = NULL;
    add(&head, 12);
    add(&head, 15);
    add(&head, 10);
    add(&head, 16);
    add(&head, 14);
    // add(&head,17);
    print(&head);
    // reverseW2(&head);
    // print(&head);
    getMedil2(head);
    return 0;
}
```

```
#################### L3 ##################
#include <bits/stdc++.h>
using namespace std;

class node
{
public:
    int data;
    node *next;
};

void add(node **head_ref, int val)
{
```

```
    node *tmp = new node();
    tmp->data = val;
    tmp->next = NULL;
    if (*head_ref == NULL)
    {
        *head_ref = tmp;
    }
    else
    {
        node *travers = *head_ref;
        while (travers->next != NULL)
        {
            travers = travers->next;
        }
        travers->next = tmp;
    }
}

void print(node **head_ref)
{
    node *tmp = *head_ref;
    while (tmp != NULL)
    {
        cout << tmp->data << " ";
        tmp = tmp->next;
    }
    cout << endl;
}

node *kRevase(node *head, int k)
{
    // base case
    if (head == NULL)
    {
        return NULL;
    }
    // Step 1: reverse 1st k node
    node *nextN = NULL;
    node *curr = head;
    node *pre = NULL;
```

```
    int count = 0;
    while (curr != NULL && count < k)
    {
        nextN = curr->next;
        curr->next = pre;
        pre = curr;
        curr = nextN;
        count++;
    }

    // step 2: baki ta recucrsion dekha nibe
    if (nextN != NULL)
    {
        head->next = kRevase(nextN, k);
    }
    // step 3 : return head of reversed list
    return pre;
}

bool isCircula1(node **head_ref)
{
    // case 1 : if the list is empty
    if (*head_ref == NULL)
    {
        return true;
    }
    // creat a node that point to the head next
    node *tmp = (*head_ref)->next;

    // cheque 2 condition
    // 1 if non circular then it will be null
    // 2 if cicular then it should not come in head
    while (tmp != NULL && tmp != (*head_ref))
    {
        tmp = tmp->next;
    }
    if (tmp == (*head_ref))
    {
        return true;
    }
```

```
    return false;
}

// function to find the circular linked list.
bool isCircular(node *head)
{
    node *temp = head;
    while (temp != NULL)
    { // if temp points to head then it has completed a circle,thus a
circular linked list.
        if (temp->next == head)
            return true;
        temp = temp->next;
    }
    return false;
}

int main()
{

    node *head = NULL;
    add(&head, 1);
    add(&head, 2);
    add(&head, 3);
    add(&head, 4);
    add(&head, 5);
    add(&head, 6);
    add(&head, 3);
    print(&head);

    head = kRevase(head, 3);
    print(&head);
    if (isCircular(head))
    {
        cout << "YES" << endl;
    }
    else
    {
        cout << "NO" << endl;
    }
```

```
   return 0;
}
```

```
######################### L3_1 ######################3

// isCiricular ahar gula te kaj korbe na
// karon agra sob nood ar ses a null point korace


#include <iostream>
using namespace std;
//node structure
struct node{
int data;
struct node* next;
};
//function to find the circular linked list.
bool isCircular(node *head){
node *temp=head;
while(temp!=NULL)
{ //if temp points to head then it has completed a circle,thus a circular
linked list.
   if(temp->next==head)
       return true;
   temp=temp->next;
}

return false;

}
//function for inserting new nodes.
node *newNode(int data){
```

```
   struct node *temp=new node;
   temp->data=data;
   temp->next=NULL;
}
int main() {
 //first case
 struct node* head=newNode(1);
 head->next=newNode(2);
 head->next->next=newNode(3);
   head->next->next->next=newNode(4);
   head->next->next->next->next=head;
 if(isCircular(head))
 cout<<"yes"<<endl;
 else
 cout<<"no"<<endl;
   //second case
 struct node* head1=newNode(1);
 head1->next=newNode(2);
  if(isCircular(head1))
 cout<<"yes";
 else
 cout<<"no";
 return 0;
}
```

```
###################### L4   ######################

#include <bits/stdc++.h>
using namespace std;
int c=0;

class node
{
public:
   int data;
   node *next;
```

```
   int flag=0;
 //constructor
  node(int data){
      this->data=data;
      this->next=NULL;
  }

//    destructor
 ~node(){
     int value=this->data;
     //memory free
     if(this->next!=NULL){
         delete next;
         this->next=NULL;
     }
     cout<<"memory is free for node with data"<<endl;
 }

};


void add(node **head_ref, int val)
{
   node *tmp = new node(val);
   tmp->data = val;
   tmp->next = NULL;
   if (*head_ref == NULL)
   {
       *head_ref = tmp;
   }
   else
   {
       node *travers = *head_ref;
       while (travers->next != NULL)
       {
           travers = travers->next;
       }
       travers->next = tmp;
   }
   c++;
```

```
}


void print(node **head_ref)
{
   node *tmp = *head_ref;
   while (tmp != NULL)
   {
       cout << tmp->data << " ";
       tmp = tmp->next;
   }
   cout << endl;
}

node *kRevase(node *head, int k)
{
   // base case
   if (head == NULL)
   {
       return NULL;
   }
   // Step 1: reverse 1st k node
   node *nextN = NULL;
   node *curr = head;
   node *pre = NULL;
   int count = 0;
   while (curr != NULL && count < k)
   {
       nextN = curr->next;
       curr->next = pre;
       pre = curr;
       curr = nextN;
       count++;
   }

   // step 2: baki ta recucrsion dekha nibe
   if (nextN != NULL)
   {
       head->next = kRevase(nextN, k);
   }
```

```
    // step 3 : return head of reversed list
    return pre;
}

bool isCircula1(node **head_ref)
{
    // case 1 : if the list is empty
    if (*head_ref == NULL)
    {
        return true;
    }
    // creat a node that point to the head next
    node *tmp = (*head_ref)->next;

    // cheque 2 condition
    // 1 if non circular then it will be null
    // 2 if cicular then it should not come in head
    while (tmp != NULL && tmp != (*head_ref))
    {
        tmp = tmp->next;
    }
    if (tmp == (*head_ref))
    {
        return true;
    }
    return false;
}


bool detectLoop1(node* head) {

    if(head == NULL)
        return false;

    map<node*, bool> visited;

    node* temp = head;

    while(temp !=NULL) {
```

```
        //cycle is present
        if(visited[temp] == true) {
            return true;
        }

        visited[temp] = true;
        temp = temp -> next;

    }
    return false;

}

bool detectLoop2(node* head){
    while(head!=NULL){
    //base case
    if(head->flag==1){
        return true;
    }

    head->flag=1;
    head=head->next;
    }
    return false;
}
//GFG
// floyed cycle detection algorithm
bool detectLoop3(node* head_ref){

    if(head_ref==NULL){
        return false;
    }

    node* fast=head_ref;
    node* slow=head_ref;
    while(fast!=NULL && slow!=NULL){
        fast=fast->next;
        if(fast!=NULL){
            fast=fast->next;
        }
```

```
        slow=slow->next;

        if(fast==slow){
            cout<<"present at "<<slow->data<<endl;
            return 1;
        }

    }
    // as the loop end so it  is false
    return false;
}

// find the begingn of the loop
node* begingng_of_loop(node*head){
    //if the list is empty or have only one element
    if(head==NULL ||head->next==NULL){
        return NULL;
    }

    node*fast=head;
    node* slow=head;
    //move fast and slow ahead respectively
    slow=slow->next;
    fast=fast->next->next;
    //search for loop using fast and slow pointer
    while(fast!=NULL && slow!=NULL){
          if(slow==fast){
              break;
          }
          slow=slow->next;
          fast=fast->next->next;
    }
    // if loop does not exist
    if(slow!=fast){
        return NULL;
    }

    // if loop exist then start slow from head and fast from the point of
intersecton
    slow=head;
```

```
    while (slow!=fast)
    {
        slow=slow->next;
        fast=fast->next;
    }

    return slow;

}




//lave babbor
node* detectLoop3_1(node* head_ref){

   if(head_ref==NULL){
       return NULL;
   }

   node* fast=head_ref;
   node* slow=head_ref;
   while(fast!=NULL && slow!=NULL){
       fast=fast->next;
       if(fast!=NULL){
           fast=fast->next;
       }
       slow=slow->next;

       if(fast==slow){
           cout<<"present at "<<slow->data<<endl;
           return slow;
       }

   }
   // as the loop end so it  is false
   return NULL;
}
```

```
// detect fast element of the cycle
node* getStartNode(node*head){
   if(head==NULL){
       return NULL;
   }

   node* intersection=detectLoop3_1(head);
   node*slow=head;
   while (slow!=intersection )
   {
       slow=slow->next;
       intersection=intersection->next;
   }
   return slow;

}

void removeLoop(node* head){
   if(head==NULL){
       return;
   }
   node*startLoop=detectLoop3_1(head);
   node* temp=startLoop;
   while(temp->next!=startLoop){
       temp=temp->next;
   }
   temp->next=NULL;

}
int main()
{

   node* node1=new node(10);
   node*head=node1;
   node*tail=node1;
   add(&head, 1);
   add(&head, 2);
   add(&head, 3);
   add(&head, 4);
   add(&head, 5);
```

```
    add(&head, 6);
    add(&head, 3);
    add(&head, 4);
    print(&head);
    // it is a process to take a tail pointer
    int count=0;
    cout<<c<<endl;
    while(count<c && tail->next!=NULL){
            tail=tail->next;
    }
    // end of process
     tail->next=head->next->next;

    // cout<<"head ="<<head->data<<endl;
    // cout<<"tail = "<<tail->data<<endl;
    if(detectLoop3(head)){
        cout<<"Loop present"<<endl;
    }
    else{
        cout<<"Loop not present"<<endl;
    }

    // node* detecfastElement=begingng_of_loop(head);
    // cout<<"First element of  loop is = "<<detecfastElement->data<<endl;
        node* detecfastElement=getStartNode(head);
    cout<<"First element of  loop is = "<<detecfastElement->data<<endl;
    removeLoop(head);
    print(&head);
        if(detectLoop3(head)){
        cout<<"Loop present"<<endl;
    }
    else{
        cout<<"Loop not present"<<endl;
    }

    return 0;
}
```

```
##########################  L5 ##################33
#include <bits/stdc++.h>
using namespace std;
int c=0;
class node
{
public:
   int data;
   node *next;
};

void add(node **head, int val)
{
   node *t = new node();
   t->data = val;
   t->next = NULL;

   if (*head == NULL)
   {
       *head = t;
   }
   else
   {
       node *tmp = *head;
       while (tmp->next!= NULL)
       {
           tmp = tmp->next;
       }
       tmp->next = t;
   }
   c++;
}
void print(node **head)
{
   node *tmp = *head;
   while (tmp != NULL)
```

```
    {
        cout << tmp->data << " ";
        tmp = tmp->next;
    }
    cout << endl;
}

// in sorted list
node*SLremoveDuplicte(node*head){
    //empty list
    if(head==NULL){
        return NULL;
    }
    // non empty list
    node*curr=head;
    while (curr!=NULL){
        // if element are equal
        if((curr->next!=NULL) && curr->data==curr->next->data){
                    //creat a node the point to the curr->next->next
        node* next_next=curr->next->next;
        node* nodeToDelete=curr->next;
        delete(nodeToDelete);
        curr->next=next_next;
        }
        // if not equal
        else{
            curr=curr->next;
        }
    }
    return head;

}

// remove dupliaction in unsorter list
// O(N^2) solution
void USL1removeDupliction(node*head){
     node*tmp1=head;
     node* dup;
     while(tmp1!=NULL && tmp1->next!=NULL){
         node* tmp2=tmp1;
```

```
         while(tmp2->next!=NULL){
             if(tmp1->data==tmp2->next->data){
                 dup=tmp2->next;
                 tmp2->next=tmp2->next->next;
                 delete(dup);

             }
             else{
                 tmp2=tmp2->next;
             }
         }
         tmp1=tmp1->next;
     }

}


// using sort
void sort(node*head){
   node*current=head;
   int stor;
   if(head==NULL){
       return;
   }
   else{
       while(current!=NULL){
           node* tmp=current->next;
           while (tmp!=NULL)
           {
               //if current node data is greater then tmp node data,then
swap them
               if(current->data>tmp->data){
                   stor=current->data;
                   current->data=tmp->data;
                   tmp->data=stor;
               }
               tmp=tmp->next;
           }
           current=current->next;
```

```
        }
    }
}

// using map


// splite a circular linked list into two ciricular ll
// refarence link
//GFG
void splitList(node *head, node **head1_ref,
                           node **head2_ref)
{
    node *slow_ptr = head;
    node *fast_ptr = head;

    if(head == NULL)
        return;

    /* If there are odd nodes in the circular list then
       fast_ptr->next becomes head and for even nodes
       fast_ptr->next->next becomes head */
    while(fast_ptr->next != head &&
          fast_ptr->next->next != head)
    {
        fast_ptr = fast_ptr->next->next;
        slow_ptr = slow_ptr->next;
    }

    /* If there are even elements in list
       then move fast_ptr */
    if(fast_ptr->next->next == head)
        fast_ptr = fast_ptr->next;

    /* Set the head pointer of first half */
    *head1_ref = head;

    /* Set the head pointer of second half */
    if(head->next != head)
        *head2_ref = slow_ptr->next;
```

```
    /* Make second half circular */
    fast_ptr->next = slow_ptr->next;

    /* Make first half circular */
    slow_ptr->next = head;
}


int main()
{
    node *head = NULL;
    node*tail=NULL;
    add(&head, 1);
    add(&head, 4);
    add(&head, 2);
    add(&head, 3);
    add(&head, 6);
    add(&head, 5);
    add(&head, 3);
    add(&head, 6);
    add(&head, 5);
    print(&head);
    // SLremoveDuplicte(head);
    // print(&head);
    // USL1removeDupliction(head);
    // print(&head);
    sort(head);
    print(&head);
    SLremoveDuplicte(head);
    print(&head);
    // it is a process to take a tail pointer
    int count=0;
    cout<<c<<endl;
    while(count<c && tail->next!=NULL){
            tail=tail->next;
    }
    // end of process
```

```
   //process of make circular
   //  tail->next=head;

    // splite the linked list
    node*head1=NULL;
    node*head2=NULL;

   /* Split the list */
   splitList(head, &head1, &head2);

   cout << "\nFirst Circular Linked List";
   print(&head1);

   cout << "\nSecond Circular Linked List";
   print(&head2);



   return 0;
}

##########################  L5 ###################33
#include <bits/stdc++.h>
using namespace std;
int c=0;
class node
{
public:
   int data;
   node *next;
};

void add(node **head, int val)
{
   node *t = new node();
   t->data = val;
   t->next = NULL;

   if (*head == NULL)
   {
```

```
        *head = t;
    }
    else
    {
        node *tmp = *head;
        while (tmp->next!= NULL)
        {
            tmp = tmp->next;
        }
        tmp->next = t;
    }
    c++;
}
void print(node **head)
{
    node *tmp = *head;
    while (tmp != NULL)
    {
        cout << tmp->data << " ";
        tmp = tmp->next;
    }
    cout << endl;
}

// in sorted list
node*SLremoveDuplicte(node*head){
    //empty list
    if(head==NULL){
        return NULL;
    }
    // non empty list
    node*curr=head;
    while (curr!=NULL){
        // if element are equal
        if((curr->next!=NULL) && curr->data==curr->next->data){
                    //creat a node the point to the curr->next->next
        node* next_next=curr->next->next;
        node* nodeToDelete=curr->next;
        delete(nodeToDelete);
        curr->next=next_next;
```

```
        }
        // if not equal
        else{
            curr=curr->next;
        }
    }
    return head;

}

// remove dupliaction in unsorter list
// O(N^2) solution
void USL1removeDupliction(node*head){
     node*tmp1=head;
     node* dup;
     while(tmp1!=NULL && tmp1->next!=NULL){
         node* tmp2=tmp1;
         while(tmp2->next!=NULL){
             if(tmp1->data==tmp2->next->data){
                 dup=tmp2->next;
                 tmp2->next=tmp2->next->next;
                 delete(dup);

             }
             else{
                 tmp2=tmp2->next;
             }
         }
         tmp1=tmp1->next;
     }

}


// using sort
void sort(node*head){
    node*current=head;
    int stor;
    if(head==NULL){
        return;
```

```
   }
   else{
       while(current!=NULL){
           node* tmp=current->next;
           while (tmp!=NULL)
           {
               //if current node data is greater then tmp node data,then
swap them
               if(current->data>tmp->data){
                   stor=current->data;
                   current->data=tmp->data;
                   tmp->data=stor;
               }
               tmp=tmp->next;
           }
           current=current->next;

       }
   }
}

// using map


// splite a circular linked list into two ciricular ll
// refarence link
//GFG
void splitList(node *head, node **head1_ref,
                          node **head2_ref)
{
   node *slow_ptr = head;
   node *fast_ptr = head;

   if(head == NULL)
       return;

   /* If there are odd nodes in the circular list then
      fast_ptr->next becomes head and for even nodes
      fast_ptr->next->next becomes head */
   while(fast_ptr->next != head &&
```

```
          fast_ptr->next->next != head)
    {
        fast_ptr = fast_ptr->next->next;
        slow_ptr = slow_ptr->next;
    }

    /* If there are even elements in list
       then move fast_ptr */
    if(fast_ptr->next->next == head)
        fast_ptr = fast_ptr->next;

    /* Set the head pointer of first half */
    *head1_ref = head;

    /* Set the head pointer of second half */
    if(head->next != head)
        *head2_ref = slow_ptr->next;

    /* Make second half circular */
    fast_ptr->next = slow_ptr->next;

    /* Make first half circular */
    slow_ptr->next = head;
}


int main()
{
    node *head = NULL;
    node*tail=NULL;
    add(&head, 1);
    add(&head, 4);
    add(&head, 2);
    add(&head, 3);
    add(&head, 6);
    add(&head, 5);
    add(&head, 3);
    add(&head, 6);
    add(&head, 5);
    print(&head);
```

```
   // SLremoveDuplicte(head);
   // print(&head);
   // USL1removeDupliction(head);
   // print(&head);
   sort(head);
   print(&head);
   SLremoveDuplicte(head);
   print(&head);
   // it is a process to take a tail pointer
   int count=0;
   cout<<c<<endl;
   while(count<c && tail->next!=NULL){
           tail=tail->next;
   }
   // end of process


   //process of make circular
   //  tail->next=head;

    // splite the linked list
    node*head1=NULL;
    node*head2=NULL;

   /* Split the list */
   splitList(head, &head1, &head2);

   cout << "\nFirst Circular Linked List";
   print(&head1);

   cout << "\nSecond Circular Linked List";
   print(&head2);



   return 0;
}
```

```
###################  L6  ######################

#include <bits/stdc++.h>
using namespace std;
class node
{
public:
   int data;
   node *next;
};

void add(node **head, int val)
{
   node *t = new node();
   t->data = val;
   t->next = NULL;

   if (*head == NULL)
   {
       *head = t;
   }
   else
   {
       node *tmp = *head;
       while (tmp->next != NULL)
       {
           tmp = tmp->next;
       }
       tmp->next = t;
   }
}
void print(node **head)
{
   node *tmp = *head;
   while (tmp != NULL)
   {
       cout << tmp->data << " ";
       tmp = tmp->next;
```

```
    }
    cout << endl;
}

// Q1 sort list of 0s 1s 2s

void sortListW1(node *head)
{
    if (head == NULL)
    {
        return;
    }

    int zc = 0, oc = 0, tc = 0;
    node *tmp = head;
    while (tmp != NULL)
    {
        if (tmp->data == 0)
        {
            zc++;
        }
        if (tmp->data == 1)
        {
            oc++;
        }
        if (tmp->data == 2)
        {
            tc++;
        }
        tmp = tmp->next;
    }
    tmp = head;
    while (tmp != NULL)
    {
        if (zc != 0)
        {
            tmp->data = 0;
            zc--;
        }
        else if (oc != 0)
```

```
        {
            tmp->data = 1;

            oc--;
        }
        else if (tc != 0)
        {
            tmp->data = 2;

            tc--;
        }
        tmp = tmp->next;
    }
}


// Way 2

void insertAtTail(node *&tail, node *curr)
{
    tail->next = curr;
    tail = curr;
}
node *sortListW2(node *head)
{
    // creat three dummy node
    node *zeroHead = new node();zeroHead->data = 0;
    node *zeroTail = zeroHead;

    node *oneHead = new node();oneHead->data = 0;
    node *oneTail = oneHead;

    node *twoHead = new node();twoHead->data = 0;
    node *twoTail = twoHead;

    // traversal
    node *curr = head;
    while (curr != NULL)
    {
        int value = curr->data;
```

```
        if (value == 0)
        {
            insertAtTail(zeroTail, curr);
        }
       else if (value == 1)
        {
            insertAtTail(oneTail, curr);
        }
       else if (value == 2)
        {
            insertAtTail(twoTail, curr);
        }
        curr=curr->next;
    }


    //marge 3 sublist
    // 1s list is not empty
    if(oneHead->next!=NULL){
        zeroTail->next=oneHead->next;
    }
    else{  // 1s list is empty
    zeroTail->next=twoHead->next;

    }
    oneTail->next=twoHead->next;
    twoTail->next=NULL;

    //setup head
    head=zeroHead->next;

    //delete dummy node
    delete zeroHead;
    delete oneHead;
    delete twoHead;

    return head;
}
```

```
//marge two sorted linked list
node* solve(node*,node*);
node* sortTwoList(node*first,node*second){
    // if the first linked list is null then just return 2nd list
    if(first==NULL){
        return second;
    }
    // if the 2nd linked list is null then just return 1st list
    if(second==NULL){
        return first;
    }
   // jai linkd list ar 1st data choto sata diyai marging suru korte hoba
   if(first->data<second->data){
       return solve(first,second);
   }
   else{
       return solve(second,first);
   }

}


node* solve(node*first,node*second){
    //coner case show it at the last
    // if there is only one element in the first list
    if(first->next==NULL){
        first->next=second;
        return first;
    }
    // end of base case

    //initilize all pointer
    node*curr1=first;
    node*next1=first->next;

    node*curr2=second;
    node*next2=second->next;


    while(next1!=NULL && next2!=NULL){
```

```
        // if 2nd list data lies between curr1 and next1
        if((curr2->data)>=(curr1->data)    &&
(curr2->data)<=(next1->data)){
            curr1->next=curr2;
            next2=curr2->next;
            curr2->next=next1;
            // curr1 o curr2 k aga barai
            curr1=curr2;
            curr2=next2;

        }
        else{
            // curr and next ko aga barao
            curr1=curr1->next;
            next1=next1->next;

            // if next1 is null
            if(next1==NULL){
                curr1->next=curr2;
                return first;
            }
        }

    }
    return first;
}


int main()
{

//      node *head = NULL;
//      add(&head, 1);
//      add(&head, 0);
//      add(&head, 2);
//      add(&head, 1);
//      add(&head, 0);
//      add(&head, 2);
//      add(&head, 2);
//      add(&head, 1);
```

```
//     add(&head, 0);
//     print(&head);

//    head= sortListW2(head);
//     print(&head);
node*head1=NULL;
   add(&head1, 1);
   add(&head1, 3);
   add(&head1, 5);
   print(&head1);
node*head2=NULL;
   add(&head2, 2);
   add(&head2, 4);
   add(&head2, 6);
   print(&head2);

   node* margehead=sortTwoList(head1,head2);
   print(&margehead);

   return 0;
}



###################  L6  ######################

#include <bits/stdc++.h>
using namespace std;
class node
{
public:
   int data;
   node *next;
};

void add(node **head, int val)
{
   node *t = new node();
   t->data = val;
   t->next = NULL;
```

```
    if (*head == NULL)
    {
        *head = t;
    }
    else
    {
        node *tmp = *head;
        while (tmp->next != NULL)
        {
            tmp = tmp->next;
        }
        tmp->next = t;
    }
}
void print(node **head)
{
    node *tmp = *head;
    while (tmp != NULL)
    {
        cout << tmp->data << " ";
        tmp = tmp->next;
    }
    cout << endl;
}

// Q1 sort list of 0s 1s 2s

void sortListW1(node *head)
{
    if (head == NULL)
    {
        return;
    }

    int zc = 0, oc = 0, tc = 0;
    node *tmp = head;
    while (tmp != NULL)
    {
        if (tmp->data == 0)
```

```
        {
            zc++;
        }
        if (tmp->data == 1)
        {
            oc++;
        }
        if (tmp->data == 2)
        {
            tc++;
        }
        tmp = tmp->next;
    }
    tmp = head;
    while (tmp != NULL)
    {
        if (zc != 0)
        {
            tmp->data = 0;
            zc--;
        }
        else if (oc != 0)
        {
            tmp->data = 1;

            oc--;
        }
        else if (tc != 0)
        {
            tmp->data = 2;

            tc--;
        }
        tmp = tmp->next;
    }
}


// Way 2
```

```
void insertAtTail(node *&tail, node *curr)
{
   tail->next = curr;
   tail = curr;
}
node *sortListW2(node *head)
{
   // creat three dummy node
   node *zeroHead = new node();zeroHead->data = 0;
   node *zeroTail = zeroHead;

   node *oneHead = new node();oneHead->data = 0;
   node *oneTail = oneHead;

   node *twoHead = new node();twoHead->data = 0;
   node *twoTail = twoHead;

   // traversal
   node *curr = head;
   while (curr != NULL)
   {
       int value = curr->data;
       if (value == 0)
       {
           insertAtTail(zeroTail, curr);
       }
      else if (value == 1)
       {
           insertAtTail(oneTail, curr);
       }
      else if (value == 2)
       {
           insertAtTail(twoTail, curr);
       }
       curr=curr->next;
   }


   //marge 3 sublist
   // 1s list is not empty
```

```
    if(oneHead->next!=NULL){
        zeroTail->next=oneHead->next;
    }
    else{  // 1s list is empty
    zeroTail->next=twoHead->next;

    }
    oneTail->next=twoHead->next;
    twoTail->next=NULL;

    //setup head
    head=zeroHead->next;

    //delete dummy node
    delete zeroHead;
    delete oneHead;
    delete twoHead;

    return head;
}


//marge two sorted linked list
node* solve(node*,node*);
node* sortTwoList(node*first,node*second){
    // if the first linked list is null then just return 2nd list
    if(first==NULL){
        return second;
    }
    // if the 2nd linked list is null then just return 1st list
    if(second==NULL){
        return first;
    }
   // jai linkd list ar 1st data choto sata diyai marging suru korte hoba
   if(first->data<second->data){
        return solve(first,second);
   }
   else{
        return solve(second,first);
   }
```

```
}


node* solve(node*first,node*second){
    //coner case show it at the last
    // if there is only one element in the first list
    if(first->next==NULL){
        first->next=second;
        return first;
    }
    // end of base case

    //initilize all pointer
    node*curr1=first;
    node*next1=first->next;

    node*curr2=second;
    node*next2=second->next;


    while(next1!=NULL && next2!=NULL){
        // if 2nd list data lies between curr1 and next1
        if((curr2->data)>=(curr1->data)    &&
(curr2->data)<=(next1->data)){
            curr1->next=curr2;
            next2=curr2->next;
            curr2->next=next1;
            // curr1 o curr2 k aga barai
            curr1=curr2;
            curr2=next2;

        }
        else{
            // curr and next ko aga barao
            curr1=curr1->next;
            next1=next1->next;

            // if next1 is null
            if(next1==NULL){
```

```
                curr1->next=curr2;
                return first;
            }
        }

    }
    return first;
}


int main()
{

//     node *head = NULL;
//     add(&head, 1);
//     add(&head, 0);
//     add(&head, 2);
//     add(&head, 1);
//     add(&head, 0);
//     add(&head, 2);
//     add(&head, 2);
//     add(&head, 1);
//     add(&head, 0);
//     print(&head);

//    head= sortListW2(head);
//     print(&head);
node*head1=NULL;
   add(&head1, 1);
   add(&head1, 3);
   add(&head1, 5);
   print(&head1);
node*head2=NULL;
   add(&head2, 2);
   add(&head2, 4);
   add(&head2, 6);
   print(&head2);

   node* margehead=sortTwoList(head1,head2);
   print(&margehead);
```

```
   return 0;
}
```

```
####################  L7  ########################
#include<bits/stdc++.h>
using namespace std;
class node
{
public:
   int data;
   node *next;
};

void add(node **head, int val)
{
   node *t = new node();
   t->data = val;
   t->next = NULL;

   if (*head == NULL)
   {
       *head = t;
   }
   else
   {
       node *tmp = *head;
       while (tmp->next != NULL)
       {
           tmp = tmp->next;
       }
       tmp->next = t;
   }
}
void print(node **head)
```

```
{
   node *tmp = *head;
   while (tmp != NULL)
   {
       cout << tmp->data << " ";
       tmp = tmp->next;
   }
   cout << endl;
}

//Approch 1
bool ChequePlaindrom(vector<int> vec){
   int n=vec.size();
   int s=0;
   int e=n-1;
   while(s<=e){
       if(vec[s]!=vec[e]){
           return 0;
       }
       s++;
       e--;
   }
   return 1;
}
bool isPalindrom1(node*head){
   // push data in a array and cheque if  the array is  palindrom or not
   vector<int> arr;
   node*tmp=head;
   while(tmp!=NULL){
       arr.push_back(tmp->data);
       tmp=tmp->next;
   }
   // now all the element are in array
    return ChequePlaindrom(arr);

}



//Approch 2
```

```
node* getMid(node*head){
   node* fast=head->next;
   node* slow=head;
   while(fast!=NULL && fast->next!=NULL){
       fast=fast->next->next;
       slow=slow->next;
   }
   return slow;
}

node*reverse(node*head){
   node*curr=head;
   node*prev=NULL;
   node* next=NULL;
   while(curr!=NULL){
       next=curr->next;
       curr->next=prev;
       prev=curr;
       curr=prev;
   }
   return prev;
}


// see it again
bool isPalindrom2(node*head){
   if(head->next==NULL){
       return true;
   }
   // step 1 : find Middle
   node* middle=getMid(head);

   //step 2: reverse list after middle
   node* tmp=middle->next;
   middle->next=reverse(tmp);

   //step 3 :compair both half
   node* head1=head;
   node* head2=middle->next;
   while(head2!=NULL){
```

```
        if((head1->data)!=(head2->data)){
            return false;
        }
        // head k aga barao
        head1=head1->next;
        head2=head2->next;
    }
    // step 4: repeat step 2
       tmp=middle->next;
       middle->next=reverse(tmp);
    return true;
}

int main(){
    node* head=NULL;
    add(&head, 1);
    add(&head, 3);
    add(&head, 5);
    add(&head, 6);
    add(&head, 2);
    add(&head, 1);
    print(&head);
    // if(isPalindrom2(head)){
    //     cout<<"Yes "<<endl;
    // }
    // else{
    //     cout<<"NO"<<endl;
    // }
    // cout<<getMid(head)->data<<endl;
    node* t=reverse(head);
    print(&t);

    return 0;
}
```

```
####################  L8 #############
```

```
#include<bits/stdc++.h>
using namespace std;
class node
{
public:
   int data;
   node *next;
};

void add(node **head, int val)
{
   node *t = new node();
   t->data = val;
   t->next = NULL;

   if (*head == NULL)
   {
       *head = t;
   }
   else
   {
       node *tmp = *head;
       while (tmp->next != NULL)
       {
           tmp = tmp->next;
       }
       tmp->next = t;
   }
}
void print(node **head)
{
   node *tmp = *head;
   while (tmp != NULL)
   {
       cout << tmp->data << " ";
       tmp = tmp->next;
   }
   cout << endl;
}
```

```
int main(){



   return 0;
}
```

```
################### L10 #################

#include <bits/stdc++.h>
using namespace std;

class node
{
public:
   int data;
   node *next;
};

void add(node **head, int val)
{

   node *tmp = new node();
   tmp->data = val;
   tmp->next = NULL;
   if (*head == NULL)
   {
       *head = tmp;
   }
   else
   {
       node *t = *head;
       while (t->next != NULL)
```

```
        {
            t = t->next;
        }
        t->next = tmp;
    }
}

void print(node **head)
{
    node *tmp = *head;
    while (tmp != NULL)
    {
        cout << tmp->data << " ";
        tmp = tmp->next;
    }
    cout << endl;
}

// node *getMid(node *head)
// {
//     node *slow = head;
//     node *fast = head->next;
//     while (fast != NULL && fast->next != NULL)
//     {
//         fast = fast->next->next;
//         slow = slow->next;
//     }
//     return slow;
// }

// node *marge(node *h1, node *h2)
// {
//     if (h1 == NULL)
//     {
//         return h2;
//     }
//     if (h2 == NULL)
//     {
//         return h1;
//     }
```

```
//     node *ans = new node();
//     ans->data = -1;
//     node *tmp = ans;

//     // marge two sorted linked list
//     while (h1 != NULL && h2 != NULL)
//     {
//         if ((h1->data) < (h2->data))
//         {
//             tmp->next = h1;
//             tmp = h1;
//             h1 = h1->next;
//         }
//         else
//         {
//             tmp->next = h2;
//             tmp = h2;
//             h2 = h2->next;
//         }
//     }
//     // if ther exist only h1 element
//     while (h1 != NULL)
//     {
//         tmp->next = h1;
//         tmp = h1;
//         h1 = h1->next;
//     }
//     // if ther exist only h2 element
//     while (h2 != NULL)
//     {
//         tmp->next = h2;
//         tmp = h2;
//         h2 = h2->next;
//     }
//     // dummy node k a ghor aga baria dai
//     ans = ans->next;
//     return ans;
// }
// node *margeSort(node *head)
```

```
// {
//     // base case
//     // if the list is empty or ther is only one element
//     if (head == NULL && head->next != NULL)
//     {
//         return head;
//     }

//     // break list into 2 half ,after finding mid
//     node *mid = getMid(head);
//     node *left = head;
//     node *right = mid->next;
//     mid->next = NULL;

//     // recursive calls to sort both half
//     left = margeSort(left);
//     right = margeSort(right);

//     // marge both left and right half
//     // marge it by using the concept of marge two sorted list
//     node *result = marge(left, right);

//     return result;
// }

node* findMid(node* head) {
   node* slow = head;
   node* fast = head -> next;

   while(fast != NULL && fast -> next != NULL) {
       slow = slow -> next;
       fast = fast -> next -> next;
   }
   return slow;
}

node* merge(node* left, node* right) {

   if(left == NULL)
       return right;
```

```
    if(right == NULL)
        return left;

    node* ans = new node();
    ans->data=-1;
    node* temp = ans;

    //merge 2 sorted Linked List
    while(left != NULL && right != NULL) {
        if(left -> data < right -> data ) {
            temp -> next = left;
            temp = left;
            left = left -> next;
        }
        else
        {
            temp -> next = right;
            temp = right;
            right = right -> next;
        }
    }

    while(left != NULL) {
        temp -> next = left;
        temp = left;
        left = left -> next;
    }

    while(right != NULL) {
        temp -> next = right;
        temp = right;
        right = right -> next;
    }

    ans = ans -> next;
    return ans;

}
```

```
node* mergeSort(node *head) {
   //base case
   if( head == NULL || head -> next == NULL ) {
       return head;
   }

   // break linked list into 2 halvs, after finding mid
   node* mid = findMid(head);

   node* left = head;
   node* right = mid->next;
   mid -> next = NULL;

   //recursive calls to sort both halves
   left = mergeSort(left);
   right = mergeSort(right);

   //merge both left and right halves
   node* result = merge(left, right);

   return result;
}


int main()
{

   node *head = NULL;
   add(&head, 10);
   add(&head, 13);
   add(&head, 14);
   add(&head, 15);
   add(&head, 10);
   add(&head, 5);
   add(&head, 20);
   add(&head, 3);
   add(&head, 2);
   print(&head);
   // node* fHead=margeSort(head);
```

```
    // print(&fHead);
    // cout<<(getMid(head))->data<<endl;
    // margeSort(head);
    // cout<<"Sorted list is = ";
    // print(&head);
    mergeSort(head);
    print(&head);

    return 0;
}
```