**Linked List**

**Definition Linked List : “** A linked list is a linear data structure, in which the elements are not stored at contiguous memory location.”

**Properties:**

**1)** Size can be Modified **2)** Non-Contiguous Memory **3)** Insertion and deletion any point is easier

**Insert in Linked list**: insert at **head**, insert at **tail**, **search**, **delete**, **delete at head** & **displayed**.

#include<bits/stdc++.h>

using namespace std;

class node{

    public:

        int data;

        node\* next;

        node(int val){

            data = val;

            next = NULL;

        }

};

void insertAtHead(node\* &head, int val){

    node\* n = new node(val);

    n->next = head;

    head = n;

}

void insertAtTail(node\* &head, int val){

    node\* n = new node(val);

    if(head==NULL){

        head = n;

        return;

    }

    node\* temp = head;

    while(temp->next!=NULL){

        temp = temp->next;

    }

    temp->next = n;

}

void display(node\* head){

    node\* temp = head;

    while(temp!=NULL){

        cout<<temp->data<<"->";

        temp = temp->next;

    }

    cout<<"NULL"<<endl;

}

bool search(node\* head, int key){

    node\* temp = head;

    while(temp!=NULL){

        if(temp->data==key){

            return true;

        }

        temp = temp->next;

    }

    return false;

}

void deleteAtHead(node\* &head){

    node\* todelete = head;

    head = head->next;

    delete todelete;

}

void delection(node\* &head, int val){

// good practise to Conner case to delete

    if(head==NULL){

        return;

    }

    if(head->next==NULL){

        deleteAtHead(head);

        return;

    }

    node\* temp = head;

    while(temp->next->data!=val){

        temp = temp->next;

    }

    node\* todelete = temp->next;

    temp->next = temp->next->next;

    delete todelete;

}

int main(){

    node\* head = NULL;

    insertAtTail(head,1);

    insertAtTail(head,2);

    insertAtTail(head,3);

    display(head); // 1->2->3->NULL

    insertAtHead(head,4);

    display(head); // 4->1->2->3->NULL

    cout<<search(head,2)<<endl;// 1

    delection(head,2);

    display(head); // 4->1->3->NULL

    deleteAtHead(head);

    display(head); // 1->3->NULL

    return 0;

}

**22.2 Reverse a Linked List:**

**1. Reverse at Iterator Method.**

class node{….

void insertAtTail(node\* &head,int val){….

void display(node\* head){….

node\* reverse(node\* &head){//iterator method

    node\* prevptr = NULL;

    node\* currptr = head;

    node\* nextptr;

    while(currptr!=NULL){

        nextptr = currptr->next;

        currptr->next = prevptr;

        prevptr = currptr;

        currptr = nextptr;

    }

    return prevptr;

}

int main(){

    node\* head = NULL;

    insertAtTail(head,1);

    insertAtTail(head,2);

    insertAtTail(head,3);

    insertAtTail(head,4);

    display(head); // 1->2->3->4->NULL

    node\* newhead = reverse(head);

    display(newhead);// 4->3->2->1->NULL

    return 0;

}

**2. Reverse at Recursive Method.**

class node……….

void insertAtTail(node\* &head, int val)……….

void display(node\* head)……….

node\* reverseRecursive(node\* &head){        // recursive way

    if(head==NULL || head->next==NULL){

        return head;   //base case

    }

    node\* newhead = reverseRecursive(head->next);

    head->next->next = head;

    head->next = NULL;

    return newhead;

}

int main(){

    node\* head = NULL;

    insertAtTail(head,1);

    insertAtTail(head,2);

    insertAtTail(head,3);

    insertAtTail(head,4);

    display(head);          // 1->2->3->4->NULL

    node\* newhead = reverseRecursive(head);

    display(newhead); // 4->3->2->1->NULL

    return 0;

}

**22.4 Reverse at K-Nodes. K=2**

class node………….

void insertAtTail(node\* &head, int val)……….

void display(node\* head)……….

node\* reverseknodes(node\* &head, int k){

    node\* prevptr = NULL;

    node\* currptr = head;

    node\* nextptr;

    int count = 0;

    while(currptr!=NULL && count<k){

        nextptr=currptr->next;

        currptr->next=prevptr;

        prevptr=currptr;

        currptr=nextptr;

        count++;

    }

    if(nextptr!=NULL){

        head->next = reverseknodes(nextptr,k);

    }

    return prevptr;

}

int main(){

    node\* head = NULL;

    insertAtTail(head,1);

    insertAtTail(head,2);

    insertAtTail(head,3);

    insertAtTail(head,4);

    insertAtTail(head,5);

    insertAtTail(head,6);

    display(head);          // 1->2->3->4->5->6->NULL

    int k = 2;

    node\* newhead = reverseknodes(head,k);

    display(newhead); // 2->1->4->3->6->5->NULL

    return 0;

}

**22.5 Detection and Removal of Cycle in Linked List. (Important)**

#include<bits/stdc++.h>

using namespace std;

class node{

    public:

        int data;

        node\* next;

        node(int val){

            data = val;

            next = NULL;

        }

};

void insertAtTail(node\* &head, int val){

    node\* n = new node(val);

    if(head==NULL){

        head = n;

        return;

    }

    node\* temp = head;

    while(temp->next!=NULL){

        temp = temp->next;

    }

    temp->next = n;

}

void display(node\* head){

    node\* temp = head;

    while(temp!=NULL){

        cout<<temp->data<<"->";

        temp = temp->next;

    }

    cout<<"NULL"<<endl;

}

void makeCycle(node\* &head, int pos){

    node\* temp = head;

    node\* startNode;

    int count = 1;

    while(temp->next!=NULL){

        if(count==pos){

            startNode = temp;

        }

        temp = temp->next;

        count++;

    }

    temp->next = startNode;

}

bool detectCycle(node\* &head){

    node\* slow = head;

    node\* fast = head;

    while(fast!=NULL && fast->next!=NULL){

        slow = slow->next;

        fast = fast->next->next;

        if(fast==slow){

            return true;

        }

    }

    return false;

}

void removeCycle(node\* head){

    node\* slow = head;

    node\* fast = head;

    do{

        slow = slow->next;

        fast = fast->next->next;

    }while(slow!=fast);

    fast = head;

    while(slow->next!=fast->next){

        slow = slow->next;

        fast = fast->next;

    }

    slow->next = NULL;

}

int main(){

    node\* head = NULL;

    insertAtTail(head,1);

    insertAtTail(head,2);

    insertAtTail(head,3);

    insertAtTail(head,4);

    insertAtTail(head,5);

    insertAtTail(head,6);

    makeCycle(head,3);

    // display(head);

    cout<<detectCycle(head)<<endl;

    removeCycle(head);

    cout<<detectCycle(head)<<endl;

    display(head);

    return 0;

}

**Output:**

1

0

1->2->3->4->5->6->NULL

**22.6 Doubly Linked List: Date:08/09/22**

#include<bits/stdc++.h>

using namespace std;

class node{

public:

    int data;

    node\* next;

    node\* prev;

    node(int val){

        data = val;

        next = NULL;

        prev = NULL;

    }

};

void insertAtHead(node\* &head, int val){

    node\* n = new node(val);

    n->next = head;

    if(head!=NULL){

        head->prev = n;

    }

    head = n;

}

void insertAtTail(node\* &head, int val){

    if(head==NULL){

        insertAtHead(head,val);

        return;

    }

    node\* n = new node(val);

    node\* temp = head;

    while(temp->next != NULL){

        temp = temp->next;

    }

    temp->next = n;

    n->prev = temp;

}

void display(node\* head){

    node\* temp = head;

    while(temp != NULL){

        cout<<temp->data<<" ";

        temp = temp->next;

    }

    cout<<endl;

}

void deleteAtHead(node\* &head){

    node\* todelete = head;

    head = head->next;

    head->prev = NULL;

    delete todelete;

}

void deletion(node\* &head, int pos){

    if(pos==1){

        deleteAtHead(head);

        return;

    }

    node\* temp = head;

    int count = 1;

    while(temp != NULL && count != pos){

        temp = temp->next;

        count++;

    }

    temp->prev->next = temp->next;

    if(temp->next != NULL)

        temp->next->prev = temp->prev;

    delete temp;

}

int main(){

    node\* head = NULL;

    insertAtTail(head,1);

    insertAtTail(head,2);

    insertAtTail(head,3);

    insertAtTail(head,4);

    display(head);

    insertAtHead(head,5);

    display(head);

    deletion(head,5);

    display(head);

    return 0;

}

**Output:**

1 2 3 4

5 1 2 3 4

5 1 2 3

**22.7 Append Last K-Node of a Linked List: Date:08/09/22**

#include<bits/stdc++.h>

using namespace std;

class node{

public:

    int data;

    node\* next;

    node\* prev;

    node(int val){

        data = val;

        next = NULL;

        prev = NULL;

    }

};

void insertAtHead(node\* &head, int val){

    node\* n = new node(val);

    n->next = head;

    if(head!=NULL){

        head->prev = n;

    }

    head = n;

}

void insertAtTail(node\* &head, int val){

    if(head==NULL){

        insertAtHead(head,val);

        return;

    }

    node\* n = new node(val);

    node\* temp = head;

    while(temp->next != NULL){

        temp = temp->next;

    }

    temp->next = n;

    n->prev = temp;

}

void display(node\* head){

    node\* temp = head;

    while(temp != NULL){

        cout<<temp->data<<"->";

        temp = temp->next;

    }

    cout<<"NULL"<<endl;

}

int length(node \*head){

    int l = 0;

    node\* temp = head;

    while(temp != NULL){

        l++;

        temp = temp->next;

    }

    return l;

}

node\* kappend(node\* &head, int k){

    node\* newHead;

    node\* newTail;

    node\* Tail = head;

    int l = length(head);

    k = k%l;

    int count = 1;

    while(Tail->next != NULL){

        if(count==l-k){

            newTail = Tail;

        }

        if(count==l-k+1){

            newHead = Tail;

        }

        Tail = Tail->next;

        count++;

    }

    newTail->next = NULL;

    Tail->next = head;

    return newHead;

}

int main(){

    node\* head = NULL;

    int arr[] = {1,2,3,4,5,6};

    for(int i=0;i<6;i++){

        insertAtTail(head,arr[i]);

    }

    display(head); //1->2->3->4->5->6->NULL

    node\* newHead = kappend(head,3);

    display(newHead);//4->5->6->1->2->3->NULL

    return 0;

}

**8. Find Intersection Point of 2 Linked List:**

#include<bits/stdc++.h>

using namespace std;

class node{….

void insertAtHead(node\* &head, int val){….

void insertAtTail(node\* &head, int val){….

void display(node\* head){….

int length(node \*head){

    int l = 0;

    node\* temp = head;

    while(temp != NULL){

        l++;

        temp = temp->next;

    }

    return l;

}

int intersection(node\* &head1, node\* &head2){

    int l1 = length(head1);

    int l2 = length(head2);

    int d = 0;

    node\* ptr1;

    node\* ptr2;

    if(l1 > l2){

        d = l1-l2;

        ptr1 = head1;

        ptr2 = head2;

    }

    else{

        d = l2-l1;

        ptr1 = head2;

        ptr2 = head1;

    }

    while(d){

        ptr1 = ptr1->next;

        if(ptr1==NULL)

            return -1;

        d--;

    }

    while(ptr1 != NULL && ptr2 != NULL){

        if(ptr1==ptr2)

            return ptr1->data;

        ptr1 = ptr1->next;

        ptr2 = ptr2->next;

    }

    return -1;

}

void intersect(node\* &head1, node\* &head2, int pos){

    node\* temp1 = head1;

    pos--;

    while(pos--){

        temp1 = temp1->next;

    }

    node\* temp2 = head2;

    while(temp2->next != NULL){

        temp2 = temp2->next;

    }

    temp2->next = temp1;

}

int main(){

    node\* head1 = NULL;

    node\* head2 = NULL;

    insertAtTail(head1,1);

    insertAtTail(head1,2);

    insertAtTail(head1,3);

    insertAtTail(head1,4);

    insertAtTail(head1,5);

    insertAtTail(head1,6);

    insertAtTail(head2,9);

    insertAtTail(head2,10);

    display(head1);

// 1->2->3->4->5->6->NULL

    display(head2);

// 9->10->NULL

intersect(head1,head2,2);

    display(head1);

// 1->2->3->4->5->6->NULL

    display(head2);

// 9->10->2->3->4->5->6->NULL

    cout<<intersection(head1,head2);

// 2

    return 0;

}

**9. Merge 2 Sorted Linked list:**

#include<bits/stdc++.h>

using namespace std;

class node{….

void insertAtHead(node\* &head, int val){….

void insertAtTail(node\* &head, int val){….

void display(node\* head){….

node\* Merge(node\* &head1, node\* &head2){

    node\* p1 = head1;

    node\* p2 = head2;

    node\* dummyNode = new node(-1);

    node\* p3 = dummyNode;

    while(p1!=NULL && p2!=NULL){

        if(p1->data < p2->data){

            p3->next = p1;

            p1 = p1->next;

        }

        else{

            p3->next = p2;

            p2 = p2->next;

        }

        p3 = p3->next;

    }

    while(p1!=NULL){

        p3->next = p1;

        p1 = p1->next;

        p3 = p3->next;

    }

    while(p2!=NULL){

        p3->next = p2;

        p2 = p2->next;

        p3 = p3->next;

    }

    return dummyNode->next;

}

node\* MergeRecursive(node\* &head1, node\* head2){

    if(head1==NULL){

        return head2;

    }

    if(head2==NULL){

        return head1;

    }

    node\* result;

    if(head1->data < head2->data){

        result = head1;

        result->next = MergeRecursive

(head1->next,head2);

    }

    else{

        result = head2;

        result->next = MergeRecursive

(head1,head2->next);

    }

    return result;

}

int main(){

    node\* head1 = NULL;

    node\* head2 = NULL;

    int arr1[]={1,4,5,7};

    int arr2[]={2,3,6};

    for(int i=0;i<4;i++){

        insertAtTail(head1,arr1[i]);

    }

    for(int i=0;i<3;i++){

        insertAtTail(head2,arr2[i]);

    }

    display(head1); // 1->4->5->7->NULL

    display(head2); // 2->3->6->NULL

    node\* newhead = MergeRecursive

(head1,head2);

    display(newhead);//1->2->3->4->5->6->7->NULL

    node\* head3 = NULL;

    node\* head4 = NULL;

    for(int i=0;i<4;i++){

        insertAtTail(head3,arr1[i]);

    }

    for(int i=0;i<3;i++){

        insertAtTail(head4,arr2[i]);

    }

    node\* headnew = Merge(head3,head4);

    display(headnew);//1->2->3->4->5->6->7->NULL

    return 0;

}

**10. Circular Linked List:**

#include<bits/stdc++.h>

using namespace std;

class node{

    public:

        int data;

        node\* next;

        node(int val){

            data=val;

            next=NULL;

        }

};

void insertAtHead(node\* &head, int val){

    node\* n = new node(val);

    if(head==NULL){

        n->next=n;

        head=n;

        return;

    }

    node\* temp=head;

    while(temp->next!=head){

        temp = temp->next;

    }

    temp->next=n;

    n->next=head;

    head=n;

}

void insertAtTail(node\* &head, int val){

    if(head==NULL){

        insertAtHead(head,val);

        return;

    }

    node\* n = new node(val);

    node\* temp=head;

    while(temp->next!=head){

        temp = temp->next;

    }

    temp->next=n;

    n->next=head;

}

void deleteAtHead(node\* &head){

    node\* temp = head;

    while(temp->next!=head){

        temp = temp->next;

    }

    node\* todelete = head;

    temp->next = head->next;

    head = head->next;

    delete todelete;

}

void deletion(node\* &head, int pos){

    if(pos==1){

        deleteAtHead(head);

        return;

    }

    node\* temp = head;

    int count = 1;

    while(count != pos-1){

        temp = temp->next;

        count++;

    }

    node\* todelete = temp->next;

    temp->next = temp->next->next;

    delete todelete;

}

void display(node\* head){

    node\* temp=head;

    do{

        cout<<temp->data<<"->";

        temp=temp->next;

    }while(temp!=head);

    cout<<endl;

}

int main(){

    node\* head = NULL;

    insertAtTail(head,1);

    insertAtTail(head,2);

    insertAtTail(head,3);

    insertAtTail(head,4);

    insertAtTail(head,5);

    display(head); // 1->2->3->4->5->

    insertAtHead(head,6);

    display(head); // 6->1->2->3->4->5->

    deletion(head,5);

    display(head); // 6->1->2->3->5->

    return 0;

}

**11. Put Even Position Nodes after Odd Position Nodes:**

#include<iostream>

using namespace std;

class node{

public:

    int data;

    node\* next;

    node\* prev;

    node(int val){

        data = val;

        next = NULL;

        prev = NULL;

    }

};

void insertAtHead(node\* &head, int val){

    node\* n = new node(val);

    n->next = head;

    if(head!=NULL){

        head->prev = n;

    }

    head = n;

}

void insertAtTail(node\* &head, int val){

    if(head==NULL){

        insertAtHead(head,val);

        return;

    }

    node\* n = new node(val);

    node\* temp = head;

    while(temp->next != NULL){

        temp = temp->next;

    }

    temp->next = n;

    n->prev = temp;

}

void display(node\* head){

    node\* temp = head;

    while(temp != NULL){

        cout<<temp->data<<"->";

        temp = temp->next;

    }

    cout<<"NULL"<<endl;

}

void EvenAfterOdd(node\* &head){

    node\* odd = head;

    node\* even = head->next;

    node\* evenStart = even;

    while(odd->next!=NULL

&& even->next!=NULL){

        odd->next = even->next;

        odd = odd->next;

        even->next = odd->next;

        even = even->next;

    }

    odd->next = evenStart;

    if(odd->next!=NULL){

        even->next=NULL;

    }

}

int main(){

    node\* head = NULL;

    int arr[] = {1,2,3,4,5,6};

    for(int i=0; i<6; i++){

        insertAtTail(head,arr[i]);

    }

    display(head); // 1->2->3->4->5->6->NULL

    EvenAfterOdd(head);

    display(head); // 1->3->5->2->4->6->NULL

    return 0;

}