LINKED LIST

1. IMPLEMENTATION

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

struct Node{
    int data;
    Node* next;
    Node(int x){
        data=x;
        next=NULL;
    }
};

int main()
{
        Node *head=new Node(10);
        Node *temp1=new Node(20);
        Node *temp2=new Node(30);
        head->next=temp1;
        temp1->next=temp2;
        cout<<head->data<<"-->"<<temp1->data<<"-->"<<temp2->data;
        return 0;
}
```

2. TRAVERSAL

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

struct Node{
    int data;
```

```
    Node* next;
    Node(int x){
        data=x;
        next=NULL;
    }
};

void printlist(Node *head){
    Node *curr=head;
    while(curr!=NULL){
        cout<<curr->data<<" ";
        curr=curr->next;
    }
}

int main()
{
        Node *head=new Node(10);
        head->next=new Node(20);
        head->next->next=new Node(30);
        head->next->next->next=new Node(40);
        printlist(head);
        return 0;
}
```

3. INSERTION AT BEGINNING

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

struct Node{
    int data;
    Node* next;
    Node(int x){
```

```
        data=x;
        next=NULL;
    }
};

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

}

void printlist(Node *head){
        Node *curr=head;
        while(curr!=NULL){
        cout<<curr->data<<" ";
        curr=curr->next;
        }
}
int main()
{
         Node *head=NULL;
         head=insertBegin(head,30);
         head=insertBegin(head,20);
         head=insertBegin(head,10);
         printlist(head);
         return 0;
}
```

4. INSERTION AT END OF SINGLY LINKED LIST

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

```
struct Node{
    int data;
    Node* next;
    Node(int x){
        data=x;
        next=NULL;
    }
};

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;

}

void printlist(Node *head){
        Node *curr=head;
        while(curr!=NULL){
        cout<<curr->data<<" ";
        curr=curr->next;
        }
}
int main()
{
         Node *head=NULL;
         head=insertEnd(head,10);
         head=insertEnd(head,20);
         head=insertEnd(head,30);
         printlist(head);
```

```
        return 0;
}
```

5. DELETE FIRST NODE OF SINGLY LINKED LIST

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

struct Node{
    int data;
    Node* next;
    Node(int x){
        data=x;
        next=NULL;
    }
};

void printlist(Node *head){
    Node *curr=head;
    while(curr!=NULL){
        cout<<curr->data<<" ";
        curr=curr->next;
    }cout<<endl;
}

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

```
{
        Node *head=new Node(10);
        head->next=new Node(20);
        head->next->next=new Node(30);
        printlist(head);
        head=delHead(head);
        printlist(head);

        return 0;
}
```

6. DELETE LAST NODE OF SINGLY LINKED LIST

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

struct Node{
    int data;
    Node* next;
    Node(int x){
        data=x;
        next=NULL;
    }
};

void printlist(Node *head){
    Node *curr=head;
    while(curr!=NULL){
        cout<<curr->data<<" ";
        curr=curr->next;
    }cout<<endl;
}

Node *delTail(Node *head){
```

```
    if(head==NULL)return NULL;
    if(head->next==NULL){
        delete head;
        return NULL;
    }
    Node *curr=head;
    while(curr->next->next!=NULL)
        curr=curr->next;
    delete (curr->next);
    curr->next=NULL;
    return head;
}
int main()
{
        Node *head=new Node(10);
        head->next=new Node(20);
        head->next->next=new Node(30);
        printlist(head);
        head=delTail(head);
        printlist(head);

        return 0;
}
```

7. INSERT AT GIVEN POSITION IN SINGLY LINKED LIST

8. SEARCH IN A SINGLY LINKED LIST

ITERATIVE

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

struct Node{
    int data;
    Node* next;
    Node(int x){
```

```
        data=x;
        next=NULL;
    }
};

void printlist(Node *head){
    Node *curr=head;
    while(curr!=NULL){
        cout<<curr->data<<" ";
        curr=curr->next;
    }cout<<endl;
}

int search(Node * head, int x){
    int pos=1;
    Node *curr=head;
    while(curr!=NULL){
        if(curr->data==x)
            return pos;
        else{
            pos++;
            curr=curr->next;
        }
    }
    return -1;
}

int main()
{
            Node *head=new Node(10);
            head->next=new Node(20);
            head->next->next=new Node(30);
            printlist(head);
            cout<<"Position of element in Linked List: "<<search(head,20);
            return 0;
}
```

RECURSIVE

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

struct Node{
    int data;
```

```
    Node* next;
    Node(int x){
        data=x;
        next=NULL;
    }
};

void printlist(Node *head){
    Node *curr=head;
    while(curr!=NULL){
        cout<<curr->data<<" ";
        curr=curr->next;
    }cout<<endl;
}

int search(Node * head, int x){
    if(head==NULL)return -1;
    if(head->data==x)return 1;
    else{
        int res=search(head->next,x);
        if(res==-1)return -1;
        else return res+1;
    }
}

int main()
{
        Node *head=new Node(10);
        head->next=new Node(20);
        head->next->next=new Node(30);
        printlist(head);
        cout<<"Position of element in Linked List: "<<search(head,20);
        return 0;
}
```

9. DOUBLY LINKED LIST

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

struct Node{
```

```
    int data;
    Node* prev;
    Node* next;
    Node(int d){
        data=d;
        prev=NULL;
        next=NULL;
    }
};

void printlist(Node *head){
    Node *curr=head;
    while(curr!=NULL){
        cout<<curr->data<<" ";
        curr=curr->next;
    }cout<<endl;
}

int main()
{
        Node *head=new Node(10);
        Node *temp1=new Node(20);
        Node *temp2=new Node(30);
        head->next=temp1;
        temp1->prev=head;
        temp1->next=temp2;
        temp2->prev=temp1;
        printlist(head);
        return 0;
}
```

10. INSERT AT BEGIN OF DOUBLY LINKED LIST

```
#include <bits/stdc++.h>
```

```
using namespace std;

struct Node{
    int data;
    Node* prev;
    Node* next;
    Node(int d){
        data=d;
        prev=NULL;
        next=NULL;
    }
};

void printlist(Node *head){
    Node *curr=head;
    while(curr!=NULL){
        cout<<curr->data<<" ";
        curr=curr->next;
    }cout<<endl;
}

Node *insertBegin(Node *head,int data){
    Node *temp=new Node(data);
    temp->next=head;
    if(head!=NULL)head->prev=temp;
    return temp;

}

int main()
{
        Node *head=new Node(10);
        Node *temp1=new Node(20);
        Node *temp2=new Node(30);
        head->next=temp1;
```

```
        temp1->prev=head;
        temp1->next=temp2;
        temp2->prev=temp1;
        head=insertBegin(head,5);
        printlist(head);
        return 0;
}
```

11. INSERT AT END OF DOUBLY LINKED LIST

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

struct Node{
    int data;
    Node* prev;
    Node* next;
    Node(int d){
        data=d;
        prev=NULL;
        next=NULL;
    }
};

void printlist(Node *head){
    Node *curr=head;
    while(curr!=NULL){
        cout<<curr->data<<" ";
        curr=curr->next;
    }cout<<endl;
}

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;
}

int main()
{
        Node *head=new Node(10);
        Node *temp1=new Node(20);
        Node *temp2=new Node(30);
        head->next=temp1;
        temp1->prev=head;
        temp1->next=temp2;
        temp2->prev=temp1;
        head=insertEnd(head,40);
        printlist(head);
        return 0;
}
```

12. REVERSE A DOUBLY LINKED LIST

13. DELETE HEAD OF A DOUBLY LINKED LIST

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

struct Node{
    int data;
    Node* prev;
```

```
    Node* next;
    Node(int d){
        data=d;
        prev=NULL;
        next=NULL;
    }
};

void printlist(Node *head){
    Node *curr=head;
    while(curr!=NULL){
        cout<<curr->data<<" ";
        curr=curr->next;
    }cout<<endl;
}

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;
    }
}

int main()
{
         Node *head=new Node(10);
         Node *temp1=new Node(20);
```

```
        Node *temp2=new Node(30);
        head->next=temp1;
        temp1->prev=head;
        temp1->next=temp2;
        temp2->prev=temp1;
        head=delHead(head);
        printlist(head);
        return 0;
}
```

14. DELETE LAST OF A DOUBLY LINKED LIST

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

struct Node{
    int data;
    Node* prev;
    Node* next;
    Node(int d){
        data=d;
        prev=NULL;
        next=NULL;
    }
};

void printlist(Node *head){
    Node *curr=head;
    while(curr!=NULL){
        cout<<curr->data<<" ";
        curr=curr->next;
    }cout<<endl;
}
```

```
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;

}

int main()
{
        Node *head=new Node(10);
        Node *temp1=new Node(20);
        Node *temp2=new Node(30);
        head->next=temp1;
        temp1->prev=head;
        temp1->next=temp2;
        temp2->prev=temp1;
        head=delLast(head);
        printlist(head);
        return 0;
}
```

15. CIRCULAR LINKED LIST

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

```
struct Node{
    int data;
    Node* next;
    Node(int d){
        data=d;
        next=NULL;
    }
};

int main()
{
        Node *head=new Node(10);
        head->next=new Node(5);
        head->next->next=new Node(20);
        head->next->next->next=new Node(15);
        head->next->next->next->next=head;
        return 0;
}
```

16. TRAVERSAL

FOR LOOP

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

struct Node{
    int data;
    Node* next;
    Node(int d){
        data=d;
        next=NULL;
    }
};
```

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

int main()
{
        Node *head=new Node(10);
        head->next=new Node(5);
        head->next->next=new Node(20);
        head->next->next->next=new Node(15);
        head->next->next->next->next=head;
        printlist(head);
        return 0;
}
```

DO WHILE LOOP

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

struct Node{
    int data;
    Node* next;
    Node(int d){
        data=d;
        next=NULL;
    }
};

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

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

int main()
{
        Node *head=new Node(10);
        head->next=new Node(5);
        head->next->next=new Node(20);
        head->next->next->next=new Node(15);
        head->next->next->next->next=head;
        printlist(head);
        return 0;
}
```

17. INSERT AT BEGIN OF CIRCULAR LINKED LIST

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

struct Node{
    int data;
    Node* next;
    Node(int d){
        data=d;
        next=NULL;
    }
};

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

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

Node *insertBegin(Node * head,int x){
    Node *temp=new Node(x);
    if(head==NULL)
        temp->next=temp;
    else{
        Node *curr=head;
        while(curr->next!=head)
            curr=curr->next;
        curr->next=temp;
        temp->next=head;
    }
    return temp;
}

int main()
{
        Node *head=new Node(10);
        head->next=new Node(20);
        head->next->next=new Node(30);
        head->next->next->next=head;
        head=insertBegin(head,15);
        printlist(head);
        return 0;
}
```

18. INSERT AT END OF CIRCULAR LINKED LIST

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

struct Node{
    int data;
    Node* next;
    Node(int d){
        data=d;
        next=NULL;
    }
};

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

Node *insertEnd(Node *head,int x){
    Node *temp=new Node(x);
    if(head==NULL){
        temp->next=temp;
        return temp;
    }
    else{
        temp->next=head->next;
        head->next=temp;
        int t=head->data;
        head->data=temp->data;
        temp->data=t;
        return temp;
    }
```

```
}

int main()
{
        Node *head=new Node(10);
        head->next=new Node(20);
        head->next->next=new Node(30);
        head->next->next->next=head;
        head=insertEnd(head,15);
        printlist(head);
        return 0;
}
```

19. DELETE HEAD OF CIRCULAR LINKED LIST

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

struct Node{
    int data;
    Node* next;
    Node(int d){
        data=d;
        next=NULL;
    }
};

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

```
}

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

int main()
{
        Node *head=new Node(10);
        head->next=new Node(20);
        head->next->next=new Node(30);
        head->next->next->next=new Node(40);
        head->next->next->next->next=head;
        head=delHead(head);
        printlist(head);
        return 0;
}
```

20. DELETE KTH NODE OF CIRCULAR LINKED LIST

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

struct Node{
```

```
    int data;
    Node* next;
    Node(int d){
        data=d;
        next=NULL;
    }
};

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

Node *deleteHead(Node *head){
    if(head==NULL)return NULL;
    if(head->next==head){
        delete head;
        return NULL;
    }
    head->data=head->next->data;
    Node *temp=head->next;
    head->next=head->next->next;
    delete temp;
    return head;
}

Node *deleteKth(Node *head,int k){
    if(head==NULL)return head;
    if(k==1)return deleteHead(head);
        Node *curr=head;
    for(int i=0;i<k-2;i++)
```

```
        curr=curr->next;
    Node *temp=curr->next;
    curr->next=curr->next->next;
    delete temp;
    return head;
}

int main()
{
        Node *head=new Node(10);
        head->next=new Node(20);
        head->next->next=new Node(30);
        head->next->next->next=new Node(40);
        head->next->next->next->next=head;
        head=deleteKth(head,3);
        printlist(head);
        return 0;
}
```

21. CIRCULAR DOUBLY LINKED LIST

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

struct Node{
    int data;
    Node *prev;
    Node* next;
    Node(int d){
        data=d;
        prev=NULL;
        next=NULL;
    }
};
```

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

Node *insertAtHead(Node *head,int x){
    Node *temp=new Node(x);
    if(head==NULL){
        temp->next=temp;
        temp->prev=temp;
        return temp;
    }
    temp->prev=head->prev;
    temp->next=head;
    head->prev->next=temp;
    head->prev=temp;
    return temp;
}


int main()
{
        Node *head=new Node(10);
        Node *temp1=new Node(20);
        Node *temp2=new Node(30);
        head->next=temp1;
        temp1->next=temp2;
        temp2->next=head;
        temp2->prev=temp1;
        temp1->prev=head;
```

```
        head->prev=temp2;
        head=insertAtHead(head,5);
        printlist(head);
        return 0;
}
```

22. SORTED INSERT IN A SINGLY LINKED LIST

23. MIDDLE OF A SINGLY LINKED LIST

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

struct Node{
    int data;
    Node* next;
    Node(int x){
        data=x;
        next=NULL;
    }
};

void printlist(Node *head){
    Node *curr=head;
    while(curr!=NULL){
        cout<<curr->data<<" ";
        curr=curr->next;
    }cout<<endl;
}

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;
}

int main()
{
        Node *head=new Node(10);
        head->next=new Node(20);
        head->next->next=new Node(30);
        head->next->next->next=new Node(40);
        head->next->next->next->next=new Node(50);
        printlist(head);
        cout<<"Middle of Linked List: ";
        printMiddle(head);
        return 0;
}
```

24. NTH NODE FROM END OF A LINKED LIST

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

struct Node{
    int data;
    Node* next;
    Node(int x){
        data=x;
        next=NULL;
    }
};

void printlist(Node *head){
```

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

void printNthFromEnd(Node * head,int n){
    if(head==NULL)return;
    Node *first=head;
    for(int i=0;i<n;i++){
        if(first==NULL)return;
        first=first->next;
    }
    Node *second=head;
    while(first!=NULL){
        second=second->next;
        first=first->next;
    }
    cout<<(second->data);
}

int main()
{
        Node *head=new Node(10);
        head->next=new Node(20);
        head->next->next=new Node(30);
        head->next->next->next=new Node(40);
        head->next->next->next->next=new Node(50);
        printlist(head);
        cout<<"Nth node from end of Linked List: ";
        printNthFromEnd(head,2);
        return 0;
}
```

25. REVERSE A LINKED LIST

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

struct Node{
    int data;
    Node* next;
    Node(int x){
        data=x;
        next=NULL;
    }
};

void printlist(Node *head){
    Node *curr=head;
    while(curr!=NULL){
        cout<<curr->data<<" ";
        curr=curr->next;
    }cout<<endl;
}

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

```
int main()
{
        Node *head=new Node(10);
        head->next=new Node(20);
        head->next->next=new Node(30);
        printlist(head);
        head=reverse(head);
        printlist(head);
        return 0;
}
```

26. RECURSIVE REVERSE OF A LINKED LIST

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

struct Node{
   int data;
   Node* next;
   Node(int x){
      data=x;
      next=NULL;
   }
};

void printlist(Node *head){
   Node *curr=head;
   while(curr!=NULL){
      cout<<curr->data<<" ";
      curr=curr->next;
   }cout<<endl;
}

Node *recRevL(Node *head){
```

```
    if(head==NULL||head->next==NULL)return head;
    Node *rest_head=recRevL(head->next);
    Node *rest_tail=head->next;
    rest_tail->next=head;
    head->next=NULL;
    return rest_head;
}

int main()
{
        Node *head=new Node(10);
        head->next=new Node(20);
        head->next->next=new Node(30);
        printlist(head);
        head=recRevL(head);
        printlist(head);
        return 0;
}
```

27. RECURSIVE REVERSE LINKED LIST PART 2

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

struct Node{
    int data;
    Node* next;
    Node(int x){
        data=x;
        next=NULL;
    }
};

void printlist(Node *head){
```

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

Node *recRevL(Node *curr,Node *prev){
    if(curr==NULL)return prev;
    Node *next=curr->next;
    curr->next=prev;
    return recRevL(next,curr);
}

int main()
{
        Node *head=new Node(10);
        head->next=new Node(20);
        head->next->next=new Node(30);
        printlist(head);
        head=recRevL(head,NULL);
        printlist(head);
        return 0;
}
```

28. REMOVE DUPLICATES FROM A SORTED LINKED LIST

29. REVERSE A LINKED LIST IN GROUPS OF SIZE K

RECURSIVE

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

```
struct Node{
    int data;
    Node* next;
    Node(int x){
        data=x;
        next=NULL;
    }
};

void printlist(Node *head){
    Node *curr=head;
    while(curr!=NULL){
        cout<<curr->data<<" ";
        curr=curr->next;
    }cout<<endl;
}

Node *reverseK(Node *head,int k){
    Node *curr=head,*next=NULL,*prev=NULL;
    int count=0;
    while(curr!=NULL&&count<k){
        next=curr->next;
        curr->next=prev;
        prev=curr;
        curr=next;
        count++;
    }
    if(next!=NULL){
        Node *rest_head=reverseK(next,k);
        head->next=rest_head;
    }
    return prev;
}

int main()
```

```
{
        Node *head=new Node(10);
        head->next=new Node(20);
        head->next->next=new Node(30);
        head->next->next->next=new Node(40);
        head->next->next->next->next=new Node(50);
        head->next->next->next->next->next=new Node(60);
        head->next->next->next->next->next->next=new Node(70);
        printlist(head);
        head=reverseK(head,3);
        printlist(head);
        return 0;
}
```

ITERATIVE

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

struct Node{
   int data;
   Node* next;
   Node(int x){
      data=x;
      next=NULL;
   }
};

void printlist(Node *head){
   Node *curr=head;
   while(curr!=NULL){
      cout<<curr->data<<" ";
      curr=curr->next;
   }cout<<endl;
}
```

```
Node *reverseK(Node *head,int k){
   Node *curr=head,*prevFirst=NULL;
   bool isFirstPass=true;
   while(curr!=NULL){
      Node *first=curr,*prev=NULL;
      int count=0;
      while(curr!=NULL && count<k){
      Node *next=curr->next;
      curr->next=prev;
      prev=curr;
      curr=next;
      count++;
      }
      if(isFirstPass){head=prev;isFirstPass=false;}
      else{prevFirst->next=prev;}
      prevFirst=first;
   }
   return head;
}

int main()
{
        Node *head=new Node(10);
        head->next=new Node(20);
        head->next->next=new Node(30);
        head->next->next->next=new Node(40);
        head->next->next->next->next=new Node(50);
        head->next->next->next->next->next=new Node(60);
        head->next->next->next->next->next->next=new Node(70);
        printlist(head);
        head=reverseK(head,3);
        printlist(head);
        return 0;
}
```

30. DETECT CYCLE

METHOD 1:

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

struct Node{
    int data;
    Node* next;
    Node(int x){
        data=x;
        next=NULL;
    }
};

bool isLoop(Node* head)
{   Node* temp=new Node(0);
    Node *curr=head;
    while (curr != NULL) {
        if (curr->next==NULL)
            return false;
        if(curr->next==temp)
            return true;
        Node *curr_next=curr->next;
        curr->next=temp;
        curr=curr_next;
    }
    return false;
}

int main()
{
```

```
        Node *head=new Node(15);
        head->next=new Node(10);
        head->next->next=new Node(12);
        head->next->next->next=new Node(20);
        head->next->next->next->next=head->next;
        if (isLoop(head))
       cout << "Loop found";
   else
       cout << "No Loop";
        return 0;
}
```

METHOD 2:

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

struct Node{
   int data;
   Node* next;
   Node(int x){
      data=x;
      next=NULL;
   }
};

bool isLoop(Node* head)
{
   unordered_set<Node*> s;
   for(Node *curr=head;curr!=NULL;curr=curr->next) {
      if (s.find(curr) != s.end())
         return true;
      s.insert(curr);
   }
   return false;
```

```
}

int main()
{
        Node *head=new Node(15);
        head->next=new Node(10);
        head->next->next=new Node(12);
        head->next->next->next=new Node(20);
        head->next->next->next->next=head->next;
        if (isLoop(head))
       cout << "Loop found";
   else
       cout << "No Loop";
        return 0;
}
```

31. DETECT LOOP USING FLOYD CYCLE DETECTION

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

struct Node{
   int data;
   Node* next;
   Node(int x){
      data=x;
      next=NULL;
   }
};

bool isLoop(Node* head)
{
```

```
    Node *slow_p = head, *fast_p = head;

    while (fast_p!=NULL && fast_p->next!=NULL) {
        slow_p = slow_p->next;
        fast_p = fast_p->next->next;
        if (slow_p == fast_p) {
            return true;
        }
    }
    return false;
}

int main()
{
         Node *head=new Node(15);
         head->next=new Node(10);
         head->next->next=new Node(12);
         head->next->next->next=new Node(20);
         head->next->next->next->next=head->next;
         if (isLoop(head))
        cout << "Loop found";
    else
        cout << "No Loop";
         return 0;
}
```

32. DETECT AND REMOVE LOOP IN LINKED LIST

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

struct Node{
    int data;
    Node* next;
```

```
    Node(int x){
        data=x;
        next=NULL;
    }
};

void detectRemoveLoop(Node* head)
{
    Node *slow = head, *fast = head;

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

int main()
{
         Node *head=new Node(15);
         head->next=new Node(10);
         head->next->next=new Node(12);
         head->next->next->next=new Node(20);
         head->next->next->next->next=head->next;
         detectRemoveLoop(head);
```

```
        return 0;
}
```

33. DELETE NODE WITH ONLY POINTER GIVEN

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

struct Node{
    int data;
    Node* next;
    Node(int x){
        data=x;
        next=NULL;
    }
};

void printlist(Node *head){
    Node *curr=head;
    while(curr!=NULL){
        cout<<curr->data<<" ";
        curr=curr->next;
    }cout<<endl;
}

void deleteNode(Node *ptr){
    Node *temp=ptr->next;
    ptr->data=temp->data;
    ptr->next=temp->next;
    delete(temp);
}

int main()
{
```

```
        Node *head=new Node(10);
        head->next=new Node(20);
        Node *ptr=new Node(30);
        head->next->next=ptr;
        head->next->next->next=new Node(40);
        head->next->next->next->next=new Node(25);
        printlist(head);
        deleteNode(ptr);
        printlist(head);
        return 0;
}
```

34. SEGREGATE EVEN ODD NODES

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

struct Node{
    int data;
    Node* next;
    Node(int x){
        data=x;
        next=NULL;
    }
};

void printlist(Node *head){
    Node *curr=head;
    while(curr!=NULL){
        cout<<curr->data<<" ";
        curr=curr->next;
    }cout<<endl;
}
```

```
Node *segregate(Node *head){
    Node *eS=NULL,*eE=NULL,*oS=NULL,*oE=NULL;
    for(Node *curr=head;curr!=NULL;curr=curr->next){
        int x=curr->data;
        if(x%2==0){
            if(eS==NULL){
                eS=curr;
                eE=eS;
            }else{
                eE->next=curr;
                eE=eE->next;
            }
        }else{
            if(oS==NULL){
                oS=curr;
                oE=oS;
            }else{
                oE->next=curr;
                oE=oE->next;
            }
        }
    }
    if(oS==NULL||eS==NULL)
        return head;
    eE->next=oS;
    oE->next=NULL;
    return eS;
}

int main()
{
        Node *head=new Node(17);
        head->next=new Node(15);
        head->next->next=new Node(8);
        head->next->next->next=new Node(12);
```

```
        head->next->next->next->next=new Node(10);
        head->next->next->next->next->next=new Node(5);
        head->next->next->next->next->next->next=new Node(4);
        printlist(head);
        head=segregate(head);
        printlist(head);
        return 0;
}
```

35. INTERSECTION OF 2 LINKED LIST

METHOD 1:

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

struct Node {
        int data;
        Node* next;
        Node (int x){
            data=x;
            next=NULL;
        }
};

int getIntersection(Node* head1, Node* head2)
{
    unordered_set<Node*> s;
    Node* curr=head1;
    while(curr!=NULL){
        s.insert(curr);
        curr=curr->next;
    }
    curr=head2;
```

```
    while(curr!=NULL){
        if(s.find(curr)!=s.end())
            return curr->data;
        curr=curr->next;
    }
    return -1;
}

int main()
{
        /*
                Creation of two linked lists

                1st 3->6->9->15->30
                2nd 10->15->30

                15 is the intersection point
        */

        Node* newNode;

        Node* head1 = new Node(10);

        Node* head2 = new Node(3);

        newNode = new Node(6);
        head2->next = newNode;

        newNode = new Node(9);
        head2->next->next = newNode;

        newNode = new Node(15);
        head1->next = newNode;
        head2->next->next->next = newNode;
```

```
        newNode = new Node(30);
        head1->next->next = newNode;

        head1->next->next->next = NULL;

        cout <<getIntersection(head1, head2);
}
```

METHOD 2:

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

struct Node {
        int data;
        Node* next;
        Node (int x){
           data=x;
           next=NULL;
        }
};

int getCount(Node* head)
{
   Node* curr = head;
   int count = 0;
   while (curr != NULL) {
      count++;
      curr = curr->next;
   }
   return count;
}

int _getIntersection(int d, Node* head1, Node* head2)
{
```

```
    Node* current1 = head1;
    Node* current2 = head2;

    for (int i = 0; i < d; i++) {
        if (current1 == NULL) {
            return -1;
        }
        current1 = current1->next;
    }

    while (current1 != NULL && current2 != NULL) {
        if (current1 == current2)
            return current1->data;

        current1 = current1->next;
        current2 = current2->next;
    }

    return -1;
}

int getIntersection(Node* head1, Node* head2)
{
    int c1 = getCount(head1);
    int c2 = getCount(head2);
    int d;

    if (c1 > c2) {
        d = c1 - c2;
        return _getIntersection(d, head1, head2);
    }
    else {
        d = c2 - c1;
        return _getIntersection(d, head2, head1);
    }
```

```
}

int main()
{
	/*
		Creation of two linked lists

		1st 3->6->9->15->30
		2nd 10->15->30

		15 is the intersection point
	*/

	Node* newNode;

	Node* head1 = new Node(10);

	Node* head2 = new Node(3);

	newNode = new Node(6);
	head2->next = newNode;

	newNode = new Node(9);
	head2->next->next = newNode;

	newNode = new Node(15);
	head1->next = newNode;
	head2->next->next->next = newNode;

	newNode = new Node(30);
	head1->next->next = newNode;

	head1->next->next->next = NULL;

	cout <<getIntersection(head1, head2);
```

```
}
```

36. PAIRWISE SWAP NODES

METHOD 1:

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

struct Node{
    int data;
    Node* next;
    Node(int x){
        data=x;
        next=NULL;
    }
};

void printlist(Node *head){
    Node *curr=head;
    while(curr!=NULL){
        cout<<curr->data<<" ";
        curr=curr->next;
    }cout<<endl;
}

void pairwiseSwap(Node *head){
    Node *curr=head;
    while(curr!=NULL&&curr->next!=NULL){
        swap(curr->data,curr->next->data);
        curr=curr->next->next;
    }
}
```

```
int main()
{
        Node *head=new Node(1);
        head->next=new Node(2);
        head->next->next=new Node(3);
        head->next->next->next=new Node(4);
        head->next->next->next->next=new Node(5);
        printlist(head);
        pairwiseSwap(head);
        printlist(head);
        return 0;
}
```

METHOD 2:

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

struct Node{
    int data;
    Node* next;
    Node(int x){
        data=x;
        next=NULL;
    }
};

void printlist(Node *head){
    Node *curr=head;
    while(curr!=NULL){
        cout<<curr->data<<" ";
        curr=curr->next;
    }cout<<endl;
}
```

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

int main()
{
        Node *head=new Node(1);
        head->next=new Node(2);
        head->next->next=new Node(3);
        head->next->next->next=new Node(4);
        head->next->next->next->next=new Node(5);
        printlist(head);
        head=pairwiseSwap(head);
        printlist(head);
        return 0;
}
```

37. CLONE A LINKED LIST USING A RANDOM POINTER

METHOD 1:

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

struct Node
{
    int data;
    Node *next,*random;
    Node(int x)
    {
        data = x;
        next = random = NULL;
    }
};

void print(Node *start)
{
    Node *ptr = start;
    while (ptr)
    {
        cout << "Data = " << ptr->data << ", Random  = "<<
ptr->random->data << endl;
        ptr = ptr->next;
    }
}

Node* clone(Node *head)
{
    unordered_map<Node*,Node*> hm;
    for(Node *curr=head;curr!=NULL;curr=curr->next)
        hm[curr]=new Node(curr->data);
```

```
    for(Node *curr=head;curr!=NULL;curr=curr->next){
        Node *cloneCurr=hm[curr];
        cloneCurr->next=hm[curr->next];
        cloneCurr->random=hm[curr->random];
    }
    Node *head2=hm[head];

    return head2;
}

int main()
{
        Node* head = new Node(10);
    head->next = new Node(5);
    head->next->next = new Node(20);
    head->next->next->next = new Node(15);
    head->next->next->next->next = new Node(20);

    head->random = head->next->next;
    head->next->random=head->next->next->next;
    head->next->next->random=head;
    head->next->next->next->random=head->next->next;
    head->next->next->next->next->random=head->next->next->next;

    cout << "Original list : \n";
    print(head);

    cout << "\nCloned list : \n";
    Node *cloned_list = clone(head);
    print(cloned_list);

    return 0;
}
```

METHOD 2:

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

struct Node
{
    int data;
    Node *next,*random;
    Node(int x)
    {
        data = x;
        next = random = NULL;
    }
};

void print(Node *start)
{
    Node *ptr = start;
    while (ptr)
    {
        cout << "Data = " << ptr->data << ", Random  = "<<
ptr->random->data << endl;
        ptr = ptr->next;
    }
}

Node* clone(Node *head)
{
    Node *next,*temp;
    for(Node *curr=head;curr!=NULL;){
        next=curr->next;
        curr->next=new Node(curr->data);
        curr->next->next=next;
        curr=next;
```

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

curr->next->random=(curr->random!=NULL)?(curr->random->next):N
ULL;
    }

     Node* original = head, *copy = head->next;

    temp = copy;

    while (original && copy)
    {
        original->next =
         original->next? original->next->next : original->next;

        copy->next = copy->next?copy->next->next:copy->next;
        original = original->next;
        copy = copy->next;
    }

    return temp;
}

int main()
{
         Node* head = new Node(10);
    head->next = new Node(5);
    head->next->next = new Node(20);
    head->next->next->next = new Node(15);
    head->next->next->next->next = new Node(20);

    head->random = head->next->next;
    head->next->random=head->next->next->next;
    head->next->next->random=head;
```

```
    head->next->next->next->random=head->next->next;
    head->next->next->next->next->random=head->next->next->next;

    cout << "Original list : \n";
    print(head);

    cout << "\nCloned list : \n";
    Node *cloned_list = clone(head);
    print(cloned_list);

    return 0;
}
```

38. LRU CACHE DESIGN

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

class Node {
    public:
        int key;
        int value;
        Node *pre;
        Node *next;


        Node(int k, int v)
        {
                key = k;
                value = v;
                pre=NULL;next=NULL;
        }
```

```
};

class LRUCache {
    public:
        unordered_map<int, Node*> map;
        int capacity, count;
        Node *head, *tail;


        LRUCache(int c)
        {
                capacity = c;
                head = new Node(0, 0);
                tail = new Node(0, 0);
                head->next = tail;
                tail->pre = head;
                head->pre = NULL;
                tail->next = NULL;
                count = 0;
        }

        void deleteNode(Node *node)
        {
                node->pre->next = node->next;
                node->next->pre = node->pre;
        }

        void addToHead(Node *node)
        {
                node->next = head->next;
                node->next->pre = node;
                node->pre = head;
                head->next = node;
        }
```

```
int get(int key)
{
        if (map[key] != NULL) {
                Node *node = map[key];
                int result = node->value;
                deleteNode(node);
                addToHead(node);
                cout<<"Got the value : " <<
                        result << " for the key: " << key<<endl;
                return result;
        }
        cout<<"Did not get any value" <<
                                                " for the key: " << key<<endl;
        return -1;
}

void set(int key, int value)
{
        cout<<"Going to set the (key, "<<
                "value) : (" << key << ", " << value << ")"<<endl;
        if (map[key] != NULL) {
                Node *node = map[key];
                node->value = value;
                deleteNode(node);
                addToHead(node);
        }
        else {
                Node *node = new Node(key, value);
                map[key]= node;
                if (count < capacity) {
                        count++;
                        addToHead(node);
                }
                else {
                        map.erase(tail->pre->key);
```

```
                deleteNode(tail->pre);
                addToHead(node);
            }
        }
    }
};

int main(){
   {

        LRUCache cache(2);

        // it will store a key (1) with value
        // 10 in the cache.
        cache.set(1, 10);

        // it will store a key (2) with value 20 in the cache.
        cache.set(2, 20);
        cout<<"Value for the key: 1 is " << cache.get(1)<<endl; //
returns 10

        // removing key 2 and store a key (3) with value 30 in the
cache.
        cache.set(3, 30);

        cout<<"Value for the key: 2 is " <<
                cache.get(2)<<endl; // returns -1 (not found)

        // removing key 1 and store a key (4) with value 40 in the
cache.
        cache.set(4, 40);
        cout<<"Value for the key: 1 is " <<
            cache.get(1)<<endl; // returns -1 (not found)
        cout<<"Value for the key: 3 is " <<
                                cache.get(3)<<endl; // returns 30
```

```
        cout<<"Value for the key: 4 is " <<
                                    cache.get(4)<<endl; // return 40

        return 0;
    }
}
```

39. MERGE 2 SORTED LINKED LISTS

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

struct Node{
    int data;
    Node* next;
    Node(int x){
        data=x;
        next=NULL;
    }
};

void printlist(Node *head){
    Node *curr=head;
    while(curr!=NULL){
        cout<<curr->data<<" ";
        curr=curr->next;
    }cout<<endl;
}

Node *sortedMerge(Node *a,Node *b){
    if(a==NULL)return b;
    if(b==NULL)return a;
    Node *head=NULL,*tail=NULL;
    if(a->data<=b->data){
```

```
        head=tail=a;a=a->next;
    }
    else{
        head=tail=b;b=b->next;
    }
    while(a!=NULL&&b!=NULL){
        if(a->data<=b->data){
            tail->next=a;tail=a;a=a->next;
        }
        else{
            tail->next=b;tail=b;b=b->next;
        }
    }
    if(a==NULL){tail->next=b;}
    else{
        tail->next=a;
    }
    return head;
}

int main()
{
        Node *a=new Node(10);
        a->next=new Node(20);
        a->next->next=new Node(30);
        Node *b=new Node(5);
        b->next=new Node(35);
        printlist(sortedMerge(a,b));
        return 0;
}
```

40. PALINDROME LINKED LIST

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

struct Node{
    char data;
    Node* next;
    Node(char x){
        data=x;
        next=NULL;
    }
};

Node *reverseList(Node *head){
    if(head==NULL||head->next==NULL)return head;
    Node *rest_head=reverseList(head->next);
    Node *rest_tail=head->next;
    rest_tail->next=head;
    head->next=NULL;
    return rest_head;
}

bool isPalindrome(Node *head){
        if(head==NULL)return true;
        Node *slow=head,*fast=head;
        while(fast->next!=NULL&&fast->next->next!=NULL){
            slow=slow->next;
            fast=fast->next->next;
        }
        Node *rev=reverseList(slow->next);
        Node *curr=head;
        while(rev!=NULL){
            if(rev->data!=curr->data)
```

```
            return false;
        rev=rev->next;
        curr=curr->next;
      }
      return true;
   }


int main()
{
       Node *head=new Node('g');
       head->next=new Node('f');
       head->next->next=new Node('g');
       if(isPalindrome(head))
          cout<<"Yes";
       else
          cout<<"No";
       return 0;
}
```