**Linked List**

### Simple lined list implementation :

#include<iostream>

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;

}

### OUTPUT :

### 10-->20-->30

### Traversing a linked list in c++

### traversal of a linked list from head to last node.

#include<iostream>

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;

}

### OUTPUT :

### 10 20 30 40

### Recursive traversal of singly linked list

//time complexity o(n) and auxilary space is o(n)

#include<iostream>

using namespace std;

struct Node

{

    int data;

    Node \*next;

    Node(int x)

    {

        data=x;

        next=NULL;

    }

};

void printList(Node \*head)

{

    if(head==NULL)

        return;

    else

        {

            cout<<head->data<<" ";

            printList(head->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;

}

### OUTPUT :

### 10 20 30 40

### Insert at the begin of singly linked list

// time complexity of insert funciton is o(1)

#include<iostream>

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;

}

### OUTPUT : 10 20 30

### Insert at the end of singly linked list :

#include<iostream>

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;

} output : 10 20 30

### Delete first Node of singly linked list :

// time complexity of delHead funciton is o(1)

#include<iostream>

using namespace std;

struct Node

{

    int data;

    Node \*next=NULL;

    Node(int x)

    {

        data=x;

        next=NULL;

    }

};

Node \*delHead(Node \*head)

{

    if(head==NULL)

        return NULL;

    else

    {

        Node \*temp=head->next;

        delete(head);

        return temp;

    }

}

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);

    head->next=new Node(20);

    head->next->next=new Node(30);

    printList(head);

    head=delHead(head);

    printList(head);

    return 0;

} OUTPUT : 10 20 30

20 30

### Delete last Node of singly linked list :

#include<iostream>

using namespace std;

struct Node

{

    int data;

    Node \*next;

    Node(int x)

    {

        data=x;

        next=NULL;

    }

};

Node \*delTail(Node \*head)

{

    if(head==NULL)

        return NULL;

     if(head->next==NULL)

    {

        delete head;

        return NULL;

    }

    else

    {

        Node \*curr=head;

        while (curr->next->next!=NULL)

            curr=curr->next;

        delete(curr->next);

        curr->next=NULL;

        return head;

    }

}

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);

    head->next=new Node(20);

    head->next->next=new Node(30);

    printList(head);

    head=delTail(head);

    printList(head);

    return 0;

}

### OUTPUT :

### 10 20 30

### 10 20

### Insert at given position in linked list:

#include<iostream>

using namespace std;

struct Node

{

    int data;

    Node \*next;

    Node(int x)

    {

        data=x;

        next=NULL;

    }

};

Node \*insertPos(Node \*head, int pos, int data)

{

    Node \*temp=new Node(data);

    if(pos==1)

    {

        temp->next=head;

        return temp;

    }

    Node \*curr=head;

    for(int i=1;i<=pos-2 && curr!=NULL;i++)

        curr=curr->next;

    if(curr==NULL)

        return head;

    temp->next=curr->next;

    curr->next=temp;

    return head;

}

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);

    head->next=new Node(20);

    head->next->next=new Node(30);

    printList(head);

    head=insertPos(head,2,15);

    printList(head);

    return 0;

}

### OUTPUT :

### 10 20 30

### 10 15 20 30

### Search in linked list :

### Iterative :

//time complexity of o(n) auxiliary space theta(n)

#include<iostream>

using namespace std;

struct Node

{

    int data;

    Node \*next;

    Node(int x)

    {

        data=x;

        next=NULL;

    }

};

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;

}

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);

    head->next=new Node(20);

    head->next->next=new Node(30);

    printList(head);

    cout<<"Position of element 20 in linked list "<<search(head,20);

    return 0;

}

### OUTPUT :

### 10 20 30

### Position of element 20 in linked list 2

### Recursive :

//time complexity of o(n) auxiliary space o(1)

#include<iostream>

using namespace std;

struct Node

{

    int data;

    Node \*next;

    Node(int x)

    {

        data=x;

        next=NULL;

    }

};

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;

    }

}

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);

    head->next=new Node(20);

    head->next->next=new Node(30);

    printList(head);

    cout<<"Position of element 20 in linked list is : "<<search(head,20);

    return 0;

}

### OUTPUT ;

### 10 20 30

### Position of element 20 in linked list is : 2

### Doubly linked list :

#include<iostream>

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;

}

### OUTPUT : 10 20 30

### Insert at begin of doubly linked list :

#include<iostream>

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;

    printList(head);

    head=insertBegin(head, 5);

    printList(head);

    return 0;

}

### OUTPUT :

### 10 20 30

### 5 10 20 30

### Insert at end of doubly linked list :

#include<iostream>

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 \*insertTail(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;

    printList(head);

    head=insertTail(head,40);

    printList(head);

    return 0;

}

### OUTPUT :

### 10 20 30

### 10 20 30 40

### Reverse a doubly linked list:

#include<iostream>

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 \*reverseDLL(Node \*head)

{

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

        return head;

    Node \*prev=NULL , \*curr=head;

    while (curr!=NULL)

    {

        prev=curr->prev;      //swapping

        curr->prev=curr->next;

        curr->next=prev;

        curr=curr->prev;

    }

    return prev->prev;

}

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);

    head=reverseDLL(head);

    printList(head);

    return 0;

}

### OUTPUT :

### 10 20 30

### 30 20 10

### Delete head of doubly linked list :

#include<iostream>

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;

        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;

    printList(head);

    head=delHead(head);

    printList(head);

    return 0;

}

### OUTPUT :

### 10 20 30

### 20 30

### Delete last of doubly linked list :

#include<iostream>

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;

    printList(head);

    head=delLast(head);

    printList(head);

    return 0;

}

### OUTPUT ; 10 20 30\n 10 20

### Circular linked list :

#include<iostream>

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;

}

### Circular linked list traversal :

### Method 1 : for loop :

#include<iostream>

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;

}

### OUTPUT :

### 10 5 20 15

### Method 2 : do while loop :

#include<iostream>

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;

}

### OUTPUT :

### 10 5 20 15

### 

### Insert at begin of circular linked list :

### Naïve : O(n) :

//time complexity o(n)

#include<iostream>

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);

    cout<<endl;

}

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;

    printList(head);

    head=insertBegin(head,15);

    printList(head);

    return 0;

}

### OUTPUT :

### 10 20 30

### 15 10 20 30

### Efficient : O(n) :

//time complexity o(1)

#include<iostream>

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);

    cout<<endl;

}

Node \*insertBegin(Node \*head, int x)

{

    Node \*temp=new Node(x);

    if(head==NULL)

    {

        temp->next=temp;

        return temp;

    }

    else

    {

        temp->next=head->next;// insert temp in between

        head->next=temp;      // head and head->next

        int t=head->data; //swappping

        head->data=temp->data;

        temp->data=t;

        return head;

    }

}

int main()

{

    Node \*head=new Node(10);

    head->next=new Node(20);

    head->next->next=new Node(30);

    head->next->next->next=head;

    printList(head);

    head=insertBegin(head,15);

    printList(head);

    return 0;

}

### OUTPUT : 10 20 30

### 15 10 20 30

### Insert at end of circular linked list :

### Niave : O(n) :

//time complexity O(n)

#include<iostream>

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);

    cout<<endl;

}

Node \*insertEnd(Node \*head, int x)

{

    Node \*temp=new Node(x);

    if(head==NULL)

    {

        temp->next=temp;

        return temp;

    }

    else

    {

        Node \*curr=head;

        while(curr->next!=head)

            curr=curr->next;

        curr->next=temp;

        temp->next=head;

        return head;

    }

}

int main()

{

    Node \*head=new Node(10);

    head->next=new Node(20);

    head->next->next=new Node(30);

    head->next->next->next=head;

    printList(head);

    head=insertEnd(head,15);

    printList(head);

    return 0;

}

### OUTPUT :

### 10 20 30

### 10 20 30 15

### Efficient : O(n) :

//time complexity O(1)

#include<iostream>

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);

    cout<<endl;

}

Node \*insertEnd(Node \*head, int x)

{

    Node \*temp=new Node(x);

    if(head==NULL)

        temp->next=temp;

    else{

        temp->next=head->next; //insert temp in between

        head->next=temp;  // head and head->next

        int t=head->data;  //swapping

        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;

    printList(head);

    head=insertEnd(head,15);

    printList(head);

    return 0;

}

### OUTPUT ;

### 10 20 30

### 10 20 30 15

### delete head of circular linked list :

### Naïve :

#include<iostream>

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);

    cout<<endl;

}

Node \*delHead(Node \*head)

{

    if(head==NULL)

        return NULL;

    if(head->next==head)

    {

        delete head;

        return NULL;

    }

    Node \*curr=head;

    while(curr->next!=head)

        curr=curr->next;

    curr->next=head->next;

    delete head;

    return 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);

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

    printList(head);

    head=delHead(head);

    printList(head);

    return 0;

}

### OUTPUT :

### 10 20 30 40

### 20 30 40

### Efficient :

#include<iostream>

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);

    cout<<endl;

}

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;

    printList(head);

    head=delHead(head);

    printList(head);

    return 0;

}

### OUTPUT :

### 10 20 30 40

### 20 30 40

### Delete kth of circular linked list :

//deleting kth node of a circular linked list where k is less than or equal to the number of nodes in the list.

#include<iostream>

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);

    cout<<endl;

}

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;

    printList(head);

    head=deleteKth(head,3);

    printList(head);

    return 0;

}

### OUTPUT :

### 10 20 30 40

### 10 20 40

### Insert at head of circular Doubly linked list :

#include<iostream>

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);

    cout<<endl;

}

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;

    printList(head);

    head=insertAtHead(head,5);

    printList(head);

    return 0;

}

### OUTPUT :

### 10 20 30

### 5 10 20 30

### Sorted insert in singly linked list :

//something is not null

//if this is null then there will be probelm

#include<iostream>

using namespace std;

struct Node

{

    int data;

    Node \*next;

    Node(int d)

    {

        data=d;

        next=NULL;

    }

};

void printList(Node \*head)

{

    Node \*curr=head;

    while(curr!=NULL)

    {

        cout<<curr->data<<" ";

        curr=curr->next;

    }

    cout<<endl;

}

Node \*sortedInsert(Node \*head, int x)

{

    Node \*temp=new Node(x);

    if(head==NULL)

        return temp;

    if(x<head->data)

    {

        temp->next=head;

        return temp;

    }

    Node \*curr=head;

    while(curr->next!=NULL && curr->next->data<x)

        curr=curr->next;

    temp->next=curr->next;

    curr->next=temp;

    return head;

}

int main()

{

    Node \*head=NULL;

    head=sortedInsert(head, 50);

    printList(head);

    head=sortedInsert(head,40);

    printList(head);

    head=sortedInsert(head,80);

    printList(head);

    head=sortedInsert(head,60);

    printList(head);

    return 0;

}

### OUTPUT :

### 50

### 40 50

### 40 50 80

### 40 50 60 80

### Middle of Linked list :

### This is an important interview problem where one needs to find the middle of a linked list of a given linked list.

### Naïve :

// This is an important interview problem where one needs

//to find the middle of a linked list of a given linked list.

//if there are even elemnt then then print second middle element

//and if odd print middle element

#include<iostream>

using namespace std;

struct Node

{

    int data;

    Node \*next;

    Node(int d)

    {

        data=d;

        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;

    int count=0;

    Node \*curr;

    for(curr=head;curr!=NULL;curr=curr->next)

        count++;

    curr=head;

    for(int i=0;i<count/2;i++)

        curr=curr->next;

    cout<<curr->data;

}

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;

}

### OUTPUT :

### 10 20 30 40 50

### Middle of Linked List: 30

### Efficient for Middle of linked list :

// This is an important interview problem where one needs

//to find the middle of a linked list of a given linked list.

//if there are even elemnt then then print second middle element

//and if odd print middle element

#include<iostream>

using namespace std;

struct  Node

{

    int data;

    Node \*next;

    Node(int d)

    {

        data=d;

        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;

}

### OUTPUT : 10 20 30 40 50

### Middle of Linked List: 30

### Nth node from end of linked list :

### Naïve :Method 1: using length linked list :

//problem on finding the n-th node from the end of a given linked list.

// Method 1(Using length of Linked List)

#include<iostream>

using namespace std;

struct Node

{

    int data;

    Node \*next;

    Node(int d)

    {

        data=d;

        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)

{

    Node \*curr;

    int len=0;

    for(curr=head;curr!=NULL;curr=curr->next)

        len++;

    curr=head;

    for(int i=1;i<len-n+1;i++)

        curr=curr->next;

    cout<<curr->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;

}

### OUTPUT :

### 10 20 30 40 50

### Nth node from end of Linked List: 40

### Efficient :Method 2: using two pointers :

//problem on finding the n-th node from the end of a given linked list.

// Method 2(Using Two Pointers/References)

#include<iostream>

using namespace std;

struct Node

{

    int data;

    Node \*next;

    Node(int d)

    {

        data=d;

        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) //check nth is less than/equalto

            return;     // no of element of head

        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);

}

### OUTPUT :

### 10 20 30 40 50

### Nth Node from end of linked list: 40

### Reverse a linked list iterative :

### Naïve :

//it required two traversal

//auxiliary space O(n)

#include<iostream>

#include<vector>

using namespace std;

struct Node

{

    int data;

    Node \*next;

    Node(int d)

    {

        data=d;

        next=NULL;

    }

};

void printList(Node \*head)

{

    Node \*curr=head;

    while(curr!=NULL)

    {

        cout<<curr->data<<" ";

        curr=curr->next;

    }

    cout<<endl;

}

Node \*revList(Node \*head)

{

    vector<int> arr;

    for(Node \*curr=head;curr!=NULL;curr=curr->next)

        arr.push\_back(curr->data);

    for(Node \*curr=head;curr!=NULL;curr=curr->next)

    {

        curr->data=arr.back();

        arr.pop\_back();

    }

    return head;

}

int main()

{

    Node \*head=new Node(10);

    head->next=new Node(20);

    head->next->next=new Node(30);

    printList(head);

    head=revList(head);

    printList(head);

    return 0;

}

### OUTPUT :

### 10 20 30

### 30 20 10

### Efficient for reverse linked list iterative :

//the idea is changing the link rather than data

#include<iostream>

using namespace std;

struct Node

{

    int data;

    Node \*next;

    Node(int d)

    {

        data=d;

        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;

}

### OUTPUT :

### 10 20 30

### 30 20 10

### Recursive reverse linked list part 1:

#include<iostream>

using namespace std;

struct Node

{

    int data;

    Node \*next;

    Node(int d)

    {

        data=d;

        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;

}

### OUTPUT :

### 10 20 30

### 30 20 10

### Recursive reverse linked list part 2:

//In this method a tail recursive solution is discussed to reverse the linked list.

//This method simply follows the iterative solution.

#include<iostream>

using namespace std;

struct Node

{

    int data;

    Node \*next;

    Node(int d)

    {

        data=d;

        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;

}

### OUTPUT :

### 10 20 30

### 30 20 10

### Remove duplicate from a sorted singly linked list :

#include<iostream>

using namespace std;

struct Node

{

    int data;

    Node \*next;

    Node(int d)

    {

        data=d;

        next=NULL;

    }

};

void printList(Node \*head)

{

    Node \*curr=head;

    while(curr!=NULL)

    {

        cout<<curr->data<<" ";

        curr=curr->next;

    }

    cout<<endl;

}

Node \*remDup(Node \*head)

{

    Node \*curr=head;

    while (curr!=NULL && curr->next!=NULL)

    {

        if(curr->data==curr->next->data)

        {

            Node \*temp=curr->next;

            curr->next=curr->next->next;

            delete temp;

        }

        else

            curr=curr->next;

    }

    return head;

}

int main()

{

    Node \*head=new Node(10);

    head->next=new Node(20);

    head->next->next=new Node(20);

    head->next->next->next=new Node(30);

    printList(head);

    head=remDup(head);

    printList(head);

}

### OUTPUT :

### 10 20 20 30

### 10 20 30

### Reverse a linked list in a group of size k :

### Iterative :

#include<iostream>

using namespace std;

struct Node

{

    int data;

    Node \*next;

    Node(int d)

    {

        data=d;

        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;

}

### OUTPUT :

### 10 20 30 40 50 60 70

### 30 20 10 60 50 40 70

### Recursive for reverse a linked list in a group of size k :

//time complextity O(n) and auxiliary space n/k

#include<iostream>

using namespace std;

struct Node

{

    int data;

    Node \*next;

    Node(int d)

    {

        data=d;

        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;

}

### OUTPUT :

### 10 20 30 40 50 60 70

### 30 20 10 60 50 40 70

### Detect Loop :

### the problem of checking whether a linked list contains any loop or not. We would discuss the four methods involved in detecting loops in a linked list, one more efficient than other.

### Method 1 : using visited aray

### Method 3 : changes refernce/pointer :

// the problem of checking whether a linked list contains any loop or not. We would discuss the

// four methods involved in detecting loops in a linked list, one more efficient than other.

#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;

}

### OUTPUT :

### Loop found

### Method 4 : Hashing :

#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;

}

### OUTPUT :

### Loop found

### Detect Loop using Floyd cycle detection :

// fast\_p will enter into the loop before (or at the same time as slow\_p)

//let fast\_p be k distance ahead of slow\_p when slow\_p enter the loops where k>=0

//this distance keeps on increasing by one in every movement of both pointers

//when distance become lenght of cycle , they meet

#include<iostream>

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;

}

### OUTPUT : Loop found

### Detect and remove loop in linked list:

#include<iostream>

using namespace std;

struct Node{

    int data;

    Node\* next;

    Node(int x)

    {

        data=x;

        next=NULL;

    }

};

void detectRemovalLoop(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;

    detectRemovalLoop(head);

    return 0;

}

### Delete Node with only pointer given to it :

### This is one of the tricky problem asked in an interview where a random address to a node of the linked list is given and the user needs to delete the node of the given address. The address can point to any random node in-between a linked list.

//it does not work for last node

#include<iostream>

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;

}

### OUTPUT :

### 10 20 30 40 25

### 10 20 40 25

### Segregate Even odd node in 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 \*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;

}

### OUTPUT :

### 17 15 8 12 10 5 4

### 8 12 10 4 17 15 5

### Intersection of Two linked list :

### Method 1 : hashing :

// 1) create an empty hash set hs

// 2) traverse the first list and put all of its node into the hs

// 3) travese the second list and look for every node in hs. as soon

//    as we find a node print in hs, we return value of it

#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

    \*/

   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);

}

### OUTPUT :

### 15

### Method 2 :

// The GetCount method returns the number of items in the collection.

// The collection is dynamic; the number of items in the collection reflects

// the current conditions, not the conditions when the Collection object

//  was created. A closed collection will return 0 items.

//method 2

// 1) Count Node in both the list let count be c1 and c2

// 2) traverse the bigger list abs(c1-c2) times

// 3) traverse both the lists simultaneously until we  the common node

#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);

}

### OUTPUT :

### 15

### Pairwise swapped linked list :

### Method 1 : swapping data:

#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;

}

### OUTPUT :

### 1 2 3 4 5

### 2 1 4 3 5

### Method 2 : changing pointer/ reference :

#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;

}

### OUTPUT :

### 1 2 3 4 5

### 2 1 4 3 5

### Clone a Linked List Using Random Pointer :

### Method 1 : Hashing :

#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;

}

### OUTPUT :

### Original List :

### Data = 10 , Random = 20

### Data = 5 , Random = 15

### Data = 20 , Random = 10

### Data = 15 , Random = 20

### Data = 20 , Random = 15

### Cloned List :

### Data = 10 , Random = 20

### Data = 5 , Random = 15

### Data = 20 , Random = 10

### Data = 15 , Random = 20

### Data = 20 , Random = 15

### 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):NULL;

    }

     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;

}

### OUTPUT:

### Original list :

### Data = 10, Random = 20

### Data = 5, Random = 15

### Data = 20, Random = 10

### Data = 15, Random = 20

### Data = 20, Random = 15

### Cloned list :

### Data = 10, Random = 20

### Data = 5, Random = 15

### Data = 20, Random = 10

### Data = 15, Random = 20

### Data = 20, Random = 15

### LRU cache design efficient :

#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;

    }

}

### OUTPUT :

### Going to set the (key, value) : (1, 10)

### Going to set the (key, value) : (2, 20)

### Value for the key: 1 is Got the value : 10 for the key: 1

### 10

### Going to set the (key, value) : (3, 30)

### Value for the key: 2 is Did not get any value for the key: 2

### -1

### Going to set the (key, value) : (4, 40)

### Value for the key: 1 is Did not get any value for the key: 1

### -1

### Value for the key: 3 is Got the value : 30 for the key: 3

### 30

### Value for the key: 4 is Got the value : 40 for the key: 4

### 40

### Merged Two Sorted Linked List :

### A O(m+n) time and O(1) auxiliary space solution is discussed

#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;

}

### OUTPUT :

### 5 10 20 30 35

### Merged Two Sorted Linked List :

### Using stack naive:

#include<bits/stdc++.h>

using namespace std;

struct Node{

    char data;

    Node \*next;

    Node(char x){

        data=x;

        next=NULL;

    }

};

bool isPalindrome(Node \*head){

    stack<char>st;

    for(Node \*curr=head;curr!=NULL;curr=curr->next)

        st.push(curr->data);

    for(Node \*curr=head;curr!=NULL;curr=curr->next)

    {

        if(st.top()!=curr->data)

            return false;

        st.pop();

    }

    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;

}

### OUTPUT :

### Yes

### Efficient :

#include<bits/stdc++.h>

using namespace std;

//reverse a half of linked list

struct Node{

    char data;

    Node \*next;

    Node(int 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;

}

### OUTPUT :

### Yes