1. **DIVISIBILITY:**

#include <stdio.h>

int divCount(int arr[],int x,int k,int n)

{

    if(x==k+1)

        return 0;

    int flag=1;

    for(int i=0;i<n;i++)

    {

        if(x%arr[i]==0)

        {

            return (1+divCount(arr,x+1,k,n));

        }

    }

    return (0+divCount(arr,x+1,k,n));

}

int main(){

    int t;

    scanf("%d",&t);

    while(t--)

    {

        int size;

        scanf("%d",&size);

        int userArray[size];

        for(int i=0;i<size;i++)

        {

            scanf("%d",&userArray[i]);

        }

        int last;

        scanf("%d",&last);

        printf("%d\n",divCount(userArray,1,last,size));

    }

}

**Non-singleton access:8**

1. **ZIGZAG TREE TRAVERSAL:**

#include<bits/stdc++.h>

using namespace std;

struct bTree

{

    string data;

    struct bTree \*right,\*left;

    bTree(string d): data{d},right{nullptr},left{nullptr}{}

};

typedef struct bTree node;

node \*root=NULL;

void insert(string str[],int n)

{

    int i=0;

    root=new bTree(str[i]);

    queue<node \*> q;

    q.push(root);

    while(i<n && !(q.empty()))

    {

        node \*temp=q.front();

        q.pop();

        if(str[++i]!="N" && i<n)

        {

            temp->left=new bTree(str[i]);

            q.push(temp->left);

        }

        if(!(i<n))

            break;

        if(str[++i]!="N" && i<n)

        {

            temp->right=new bTree(str[i]);

            q.push(temp->right);

        }

    }

}

void inorder(node \*root)

{

    if(!root)

        return ;

    inorder(root->left);

    cout<<root->data;

    inorder(root->right);

}

void zigzag(node\* root)

{

    string s="";

    s+=root->data;

    queue <node \*> q;

    q.push(root);

    while(!q.empty())

    {

        node \*temp=q.front();

        q.pop();

        if(temp->right!=NULL)

        {

            s+=" "+temp->right->data;

            q.push(temp->right);

        }

        if(temp->left!=NULL)

        {

            s+=" "+temp->left->data;

            q.push(temp->left);

        }

    }

    cout<<s<<endl;

}

int main()

{

    int t;

    cin>>t;

    while(t--)

    {

        char a[1000];

        string str[100];

        int k=0;

        scanf(" %[^\n]s",a);

        int n=strlen(a);

        for(int i=0;i<n;i++)

        {

            string s="";

            while(a[i]!=' ' && i<n)

            {

                s+=a[i++];

            }

            str[k++]=s;

        }

        insert(str,k);

        zigzag(root);

        // inorder(root);

        root=NULL;

    }

}

**Non-Singleton access:36**

1. **MERGE SORT:**

#include <bits/stdc++.h>

using namespace std;

int\* merge(int arr[],int low,int high,int piv)

{

    for(int x=low;x<=high;x++)

    {

        int key=arr[x];

        if(arr[x]>0)

            continue;

        int y=x-1;

        while(y>=0 && arr[y]>0)

        {

            arr[y+1]=arr[y];

            y--;

        }

        arr[y+1]=key;

    }

    return arr;

}

int\* mergeSort(int arr[],int low,int high)

{

    int piv=(low+high)/2;

    if(low<high)

    {

        arr=mergeSort(arr,low,piv);

        arr=mergeSort(arr,piv+1,high);

        arr=merge(arr,low,high,piv);

    }

    return arr;

}

int main() {

    int n;

    cin>>n;

    int a[n];

    for(int i=0;i<n;i++)

        cin>>a[i];

    int \*arr=mergeSort(a,0,n-1);

    for(int i=0;i<n;i++)

        cout<<arr[i]<<" ";

}

**Non-Singleton Access:18**

1. **REMOVE ALL ADJACENT DUPLICATES:**

#include<stdio.h>

#include<string.h>

char\* removeDup(char a[],int iterator,int flag,int n)

{

    if(iterator==n)

        return a;

    if(a[iterator]==a[iterator+1])

    {

        a[iterator]=0;

        return removeDup(a,iterator+1,1,n);

    }

    if(flag)

    {

        a[iterator]=0;

        return removeDup(a,iterator,0,n);

    }

    return removeDup(a,iterator+1,0,n);

}

int main()

{

    int t;

    scanf("%d",&t);

    while(t--)

    {

        char a[50];

        memset(a,0,50);

        scanf("%s",a);

        int n=strlen(a);

        char \*x=removeDup(a,0,0,n);

        for(int i=0;i<n;i++)

        {

            if(x[i]!=0)

            printf("%c",a[i]);

        }

        printf("\n");

    }

}

**Non-Singleton Access: 11**

1. **JOSEPH’S CIRCLE:**

import java.util.\*;

class CircularLL {

    Node last;

    class Node{

        int data;

        Node next;

        Node(int d)

        {

            data=d;

            next=null;

        }

    }

    void insert(int val)

    {

        Node newnnode=new Node(val);

        if(this.last==null)

        {

            this.last=newnnode;

            this.last.next=this.last;

        }

        else

        {

            newnnode.next=this.last.next;

            this.last.next=newnnode;

            this.last=newnnode;

        }

    }

    void circle(int m,int n)

    {

        Node temp=this.last;

        while(n!=1)

        {

            for(int i=1;i<m;i++)

            {

                temp=temp.next;

            }

            temp.next=temp.next.next;

            n--;

        }

        System.out.print(temp.data);

    }

    void display()

    {

        Node temp=this.last.next;

        do

        {

            System.out.print(temp.data+" ");

            temp=temp.next;

        }while(temp!=this.last);

        System.out.print(this.last.data);

    }

    public static void main(String args[] ) throws Exception {

        Scanner sc=new Scanner(System.in);

        CircularLL list=new CircularLL();

        int n=sc.nextInt(),m=sc.nextInt();

        for(int i=1;i<=n;i++)

        {

            list.insert(i);

        }

        list.circle(m,n);

    }

}

**Non-Singleton Access:13**

1. **TILT OF BINARY TREE:**

#include<bits/stdc++.h>

using namespace std;

struct btree

{

    int data,tilt;

    struct btree \*left,\*right;

    btree(int d):data{d},left{nullptr},right{nullptr},tilt{0}{};

};

typedef struct btree node;

node\* readInput()

{

    int n;

    map<int,node\*> mp;

    node\* root=NULL;

    cin>>n;

    while(n--)

    {

        int n1,n2;

        char side;

        node\* parent,\*child;

        scanf("%d %d %c",&n1,&n2,&side);

        if(mp.find(n1)==mp.end())

        {

            mp[n1]=new btree(n1);

            parent=mp[n1];

            if(root==NULL)

            {

                root=mp[n1];

            }

        }

        else

        {

        parent=mp[n1];

        }

        child=new btree(n2);

        if(side=='L')

        {

            parent->left=child;

        }

        else

        {

            parent->right=child;

        }

        mp[n2]=child;

    }

    return root;

}

int findSum(node\* root)

{

    if(!root)       return 0;

    return root->data+(findSum(root->left) + findSum(root->right));

}

void findTilt(node\* root)

{

    if(!root)   return ;

    findTilt(root->left);

    findTilt(root->right);

    root->tilt=abs(findSum(root->left)-findSum(root->right));

}

int tiltTree(node\* root)

{

    if(!root)       return 0;

    findTilt(root);

    return root->tilt+(tiltTree(root->left) + tiltTree(root->right));

}

int main()

{

    int t;

    cin>>t;

    while(t--)      cout<<tiltTree(readInput())<<endl;

}

**Non-Singleton Access:21**

1. **REVERSE PARTS OF A LINKED LIST:**

#include <bits/stdc++.h>

using namespace std;

class Node

{

private:

    int data;

    Node \*next;

public:

    void setVal(int d)

    {

        data = d;

    }

    int getval()

    {

        return data;

    }

    void setNext(Node \*nex)

    {

        next = nex;

    }

    Node \*getNext()

    {

        return next;

    }

    Node(int d) : data{d}, next{NULL} {};

};

Node \*insert(vector<int> vec)

{

    Node \*head = NULL;

    Node \*curr = NULL;

    for (auto i : vec)

    {

        if (head == NULL)

        {

            head = new Node(i);

            curr = head;

        }

        else{

            curr->setNext(new Node(i));

            curr = curr->getNext();

        }

    }

    return head;

}

void cutRotate(Node\* head,int k)

{

    Node \*curr = head;

    Node \*temp = NULL;

    while(curr->getNext())

    {

        for (int i = 0; i < k;i++)

        {

        }

        temp = curr->getNext();

        curr->setNext(curr);

        curr = temp;

    }

    head = curr;

}

int main()

{

    Node \*head = nullptr;

    string str = "";

    getline(cin, str);

    int k=0;

    cin >> k;

    stringstream ss(str);

    vector<int> vec;

    try

    {

        while (ss >> str)

        {

            vec.push\_back(stoi(str));

        }

    }

    catch (exception e)

    {

        cout << e.what();

    }

    head = insert(vec);

    cutRotate(head,k);

    cout << head->getval();

}

**Non-singleton Access:15**

1. **QUEUE USING DEQUEUE:**

#include<stdio.h>

#include<stdlib.h>

struct queue{

    int data;

    struct queue \*next;

};

typedef struct queue \*node;

node front=NULL;

void enqueue(int val)

{

    node newnode;

    newnode=(node)malloc(sizeof(node));

    newnode->data=val;

    newnode->next=NULL;

    if(front==NULL)

    {

        front=(node)malloc(sizeof(node));

        front=newnode;

    }

    else

    {

        node temp;

        temp=front;

        while(!temp->next==NULL)

        {

            temp=temp->next;

        }

        temp->next=newnode;

    }

}

void dequeue()

{

    if(front==NULL)

    {

        printf("-1 ");

    }

    else

    {

        printf("%d ",front->data);

        front=front->next;

    }

}

int main()

{

    int queries;

    scanf("%d",&queries);

    int x,y;

    for(int i=0;i<queries;i++)

    {

        scanf("%d",&x);

        if(x==1)

        {

            scanf("%d",&y);

            enqueue(y);

        }

        if(x==2)

        {

            dequeue();

        }

    }

    return 0;

}

**Non-Singleton Access:12**

1. **PARANTHESIS CHECKER:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

struct stack

{

    char data;

    struct stack \*next;

};

typedef struct stack \*node;

node top=NULL;

void display()

{

    node temp=top;

    while(!temp==NULL)

    {

        printf("%c",temp->data);

        temp=temp->next;

    }

}

void push(char x)

{

    node newnode=(node)malloc(sizeof(node));

    newnode->data=x;

    newnode->next=NULL;

    if(top==NULL)

    {

        top=newnode;

    }

    else{

    newnode->next=top;

    top=newnode;

    }

}

char pop()

{

    if(top==NULL)

    {

        return 'x';

    }

    char t;

    t=top->data;

    top=top->next;

    return t;

}

int main()

{

    int inputs;

    scanf("%d",&inputs);

    getchar();

    while(inputs--)

    {

        char values[1000];

        scanf("%s",values);

        int n=strlen(values);

        int flag=1,i;

        for( i=0;i<n;i++)

        {

            if(values[i]=='{' || values[i]=='[' || values[i]=='(')

            {

                push(values[i]);

            }

            if(values[i]=='}' || values[i]==']' || values[i]==')')

            {

                char test=pop();

                //printf("%c %c",test,values[i]);

                if(values[i]==')')

                {

                    if(!(test=='('))

                    {

                        flag=0;

                        break;

                    }

                }

                if(values[i]=='}')

                {

                    if(!(test=='{'))

                    {

                        flag=0;

                        break;

                    }

                }

                if(values[i]=='[')

                {

                    if(!(test==']'))

                    {

                        flag=0;

                        break;

                    }

                }

            }

        }

        if(top!=NULL)

        {

            flag=0;

        }

        if(flag)

        {

            printf("balanced\n");

        }

        else

        {

            printf("not balanced\n");

        }

    }

}

**Non-Singleton Access:27**