1] Create structure with ~~student~~ item members itemname, quantity, Price, total amount. Calculate Party expense

```
#include<stdio.h>

struct itemdetails{
    char *itemdname;
    float quantity;
    float Price;
    float total_amount;
};
int main(){
struct itemdetails items[5] = {{"lemonade",90,20,1800},
    {"cake",8,1000,8000},{"hats",70,10,700},{"plates",80,15,
                                                1200}};

int total_exp=0;
for (size_t i=0; i<5; i++)
{
    total_exp+= items[i].total_amount;
}
Printf("total expense of the Party is %d", total_exp);
return 0;
}
```

Output

total sum/expense of the party is 12600

4] Given an array of [illegible] feet sizeNum, such that each element of the array represent a distance in the form of {inch-feet}. The task is to find all the N inch-feet distances using structures

```
#include <stdio.h>
void main() {
float a[5][5] = {{1,1.7}, {1,1.5}, {6,8}};
float b = 0.0, c = 0.0;
for (int i = 0; i < 3; i++){
    b = b + a[i][0];
    c = c + a[i][1];
}
Printf("Feet sum = %f Inch sum = %f", b, c);
}
```

output

Feet sum = 8.000

Inch sum = 11.200

2]

```
#include <stdio.h>
struct student {
    char name[20];
    char usn[20];
};
void main () {
int n;
int backlog credits = 0;
int c;
char b;
Scanf ("%d", &n);
Struct student a[n];
int Prof [n];
for (int i = 0; i < n; i++){
Printf ("Enter the student name: \n");
Scanf ("%s", a[i].name);
Printf ("Enter usn");
Scanf ("%s", a[i].usn);
Printf ("Enter info of 1st sem");
for (int i = 0; i < 8; i++) {
Printf("Enter the credit & grade");
Scanf ("%d %c", &c, &b);
if (b == 'F' || b == 'f'){
    backlog.credits.f = c;
}
}
Printf ("Enter the in to for sem 2");
for (int i = 0; i < 8; i++}
Printf ("Enter the credits & grade");
Scanf ("%d %c", &c &b);
if (b == 'F' || b == 'f'){
```

```
if (backlog_credits >= 16){
    Prof[i] = 0;
else {
    Prof[i] = 0;
}
back log_credits = 0;
}
for (int i=0; i<n; i++)
{
if (Prof[i] == 0){
Printf ("%s is not eligible \n", a[i].name);
}
else{
    Printf("%s is eligible \n", a[i].name)
}
}
```

5) Create a structure with the name student book with members: name, USN, book id, issue date. A student of max will be issued 4 books for 90 days. If the student crosses 90 days to return back, each day fine for each book is Rs 50 along with book cost. Display status as follows:

- no. of days remaining return for books
- no. of books still students can be granted out of 4
- any pendening return with fine calculation

```
#include <stdio.h>
struct student books
{
 char name[20]
 char USN [10]
 int bookid[4]
 char issuedate[10];
} std1;
int main(){
    int days[4], Price[4], n=0;
    Printf (" enter name \n ");
    Scanf ("%.s", std1.name);
    Printf (" enter USN \n");
    Scanf ("%.s", std1.USN);
    for (size_t i =0; i<4; i++)
{
    Printf (" enter bookid \n");
    Scanf ("%.d", & std1.bookid[i]);
        break;
}
n++;
Printf (" Book Price : ");
Scanf ("%.d", & Price[i]);
Printf (" Number of days after issue: ");
Scanf ("%.d", & days[i]);
}
```

```
Printf("\n\n\n");
for (int i=0; i<n; i++){
    Printf("number of days remaining for bookid=%d
            is %d \n", std1.bookid[i], 90-days[i]);
    if (90-days[i]<0){
        int fine = 50*(days[i]-90)+Price[i];
        Printf("Return of bookid=%d overdue total fine
            for this book = Rs %d \n", std1.bookid[i],
                                        fine);
    }
}
    Printf("number of books issuable %d \n", 4-n);
return 0;
}
```

Output

```
enter name
Varnith
enter USN
1BM22CS329
enter book id
3
book Price : 100
no. of days after issue 45
enter book id
1
book Price: 300
no. of days after issue: 89
enter book id
5
book Price : 250
no. of days after issue : 93
enter book id
10
book Price: 790
no. of days after issue 100
```

```
no. of days remaining for bookid=3 is 45
                                   is 1
                                   is -3

return of bookid = 5 overdue total fine for book=400
                                   is -10

return of bookid = 10 overdue total fine is book=790.
```

Malloc Function

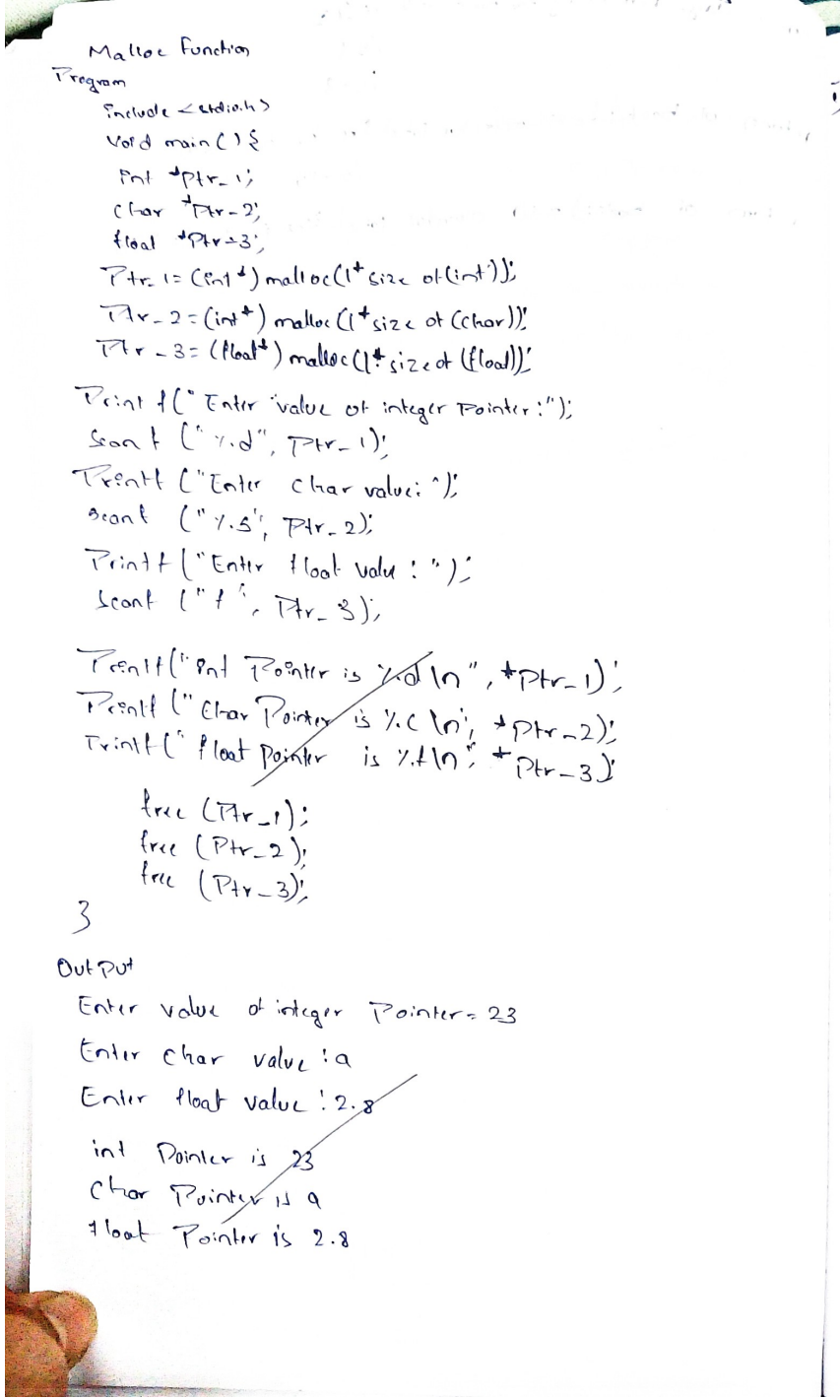
Program

```
#include <stdio.h>
void main() {
 int *ptr_1;
 char *ptr_2;
 float *ptr_3;
 ptr_1 = (int *) malloc(1*sizeof(int));
 ptr_2 = (int *) malloc(1*sizeof(char));
 ptr_3 = (float *) malloc(1*sizeof(float));
 Printf("Enter value of integer Pointer:");
 Scanf("%d", ptr_1);
 Printf("Enter char value:");
 Scanf("%c", ptr_2);
 Printf("Enter float value:");
 Scanf("%f", ptr_3);

 Printf("Int Pointer is %d\n", *ptr_1);
 Printf("Char Pointer is %c\n", *ptr_2);
 Printf("float Pointer is %f\n", *ptr_3);
    free(ptr_1);
    free(ptr_2);
    free(ptr_3);
}
```

Output

```
Enter value of integer Pointer = 23
Enter char value : a
Enter float value : 2.8
int Pointer is 23
Char Pointer is a
float Pointer is 2.8
```



ii) Stack

Pseudocode

```
Void Push (int n)
{
  if (top == (size-1)
  {
    Printf("Overflow");
  }
  else {
      top ++;
      Stack[top] = n;
  }
}

Void POP()
{
  if (top == -1){
      Printf("Underflow");
  }
  else{
      Printf("%d", stack[top]);
      top --;
  }
}
```

```
void display()
{
  if (top==-1){
     Printf("stack empty");
     }
  else {
     for (top; top>=0; top--)
      {
       Printf("%d\t", stack[top]);
      }
  }
}
```

Output:

1. Push . 2. POP 3. display 4. exit

2

Stack underflow

1. Push 2. POP 3. display 4. exit

1

enter

2] infix to Postfix

```
int Priority (char x)
{
  if (x=='(')
     return 0;
  if (x=='+' || x=='-')
     return 1;
  if (x=='*' || x=='/')
     return 2;
  return 0;
}

int main()
{
  char exp[100], *e, x;
  scanf("%s", exp);
  e= exp;
  while (*e!='\0')
  {
    if (isalnum(*e))
       Printf("%c", *e);
    else if (*e=='(')
       Push(*e);
    else if (*e==')')
    {
      while (x!='(')
      {
        x = POP()
        Printf("%c", x);
      }
    }
  }
```

01/01/24

08/01/2026

5] Queue

```
#include <stdio.h>
int max=10;
int q[max], front=-1, rear=-1;
void insert(int);
int delete();
void display();
void insert (int num)
{
  if (rear == Max-1)
      Printf("\n overflow");
  else if (front == -1 && rear == -1)
  {
     front=0;
     rear=0;
  }
  else
  rear++
  q[rear]=num;
}
int delete
{
 int val;
 if (front == -1 || front > rear)
 {
 Printf("\n underflow");
   return -1;
  }
  else{
  val = q[front++];
  if (front > rear){
  rear = -1; front = -1;}
  return val; }
  }
```

```
void display()
{
 int i;
 Printf("\n");
 if(front == -1 || front > rear)
     Printf("\n Queue is empty");
 else{
    for(i=front; i<=rear; i++)
      Printf("\t %d", q[i]);
 }
}
Void main()
{
Printf("\n1. Insert \n 2. Delete \n 3. Display \n 4. Exit");
Printf("\n Enter your choice");
 int ch; int val;
  Scanf("%d", &ch);
  while(ch!=)
 {
 case 1: Printf("\n Enter Value to be inserted:");
         Scanf("%d", &val);
         insert(val);
 Case 2: Printf("\n the queue is:");
         displa.
 case2: val = delete();
         if(val != -1)
         Printf("\n Deleted number is %d", val)
         break;
 Case 3: Printf("\n The queue is:");
         display();
         break;
 default: Printf("\n Wrong Option");
 }
}
```

Out Put:

1] Insert 2] Delete 3] Display 4] Exit

1. Enter your choice: 1
   Enter value to be inserted: 10
1. Insert 2. Delete 3]. Display 4. Exit
   Enter choice : 2
   Enter value to be inserted: 12
1. Insert 2. Delete 3. Display 4. Exit
   Enter your choice 3
   10
1. Insert 2. Delete 3. Display 4. Exit
   Enter your choice: 2
   Deleted number is 10

3] Circular Queue

```
#include <stdio.h>
#include <Process.h>
#define QUE_SIZE 3
int item, front=0, rear=-1, q[QUE_SIZE], count=0;
Void insert rear()
{
if (count == QUE-SIZE)
{
Printf("Queue Overflow\n");
return;
}
rear=(rear+1)% QUE-SIZE;
q[rear]=item;
count + +;
}
int delete front()
{
if (count == 0) return-1;
item=q[front];
front=(front+1) % QUE-SIZE;
count=count-1;
return item;
}
Void display Q()
{
int i,f;
if (count == 0)
{
Printf("queue is empty \n");
return;
}
f=front;
Printf("contents of queue \n");
for (i=1; i<= count; i++)
{
Printf("%d\n",q[f]);
f=(f+1) % QUE-SIZE;
}
}
Void main()
{ int choice;
for (;;)
{
Printf("\n 1: insert rear \n 2: delete front \n
3: display \n 4. exit \n");
Printf("enter the choice \n");
Scanf("%d", & choice);
switch(choice)
{
case 1: Printf("enter the item to be inserted \n");
Scanf("%d", & item);
insert rear();
break;
case 2: item = delete front();
if (item == -1)
Printf("queue is empty \n");
else
Printf("item deleted = %d\n", item);
break;
case 3: display Q();
break;
default: exit();
}
}
getch();
}
```

Output

1. insert 2. delete 3. display 4. exit

1

enter the item to be inserted 23

1. insert 2. delete 3 display 4. exit

1

enter the item to be inserted 34

1. insert 2. delete 3. display 4. exit

1

enter the item to be inserted 45

1. insert 2. delete 3. display 4. exit

3

Content of queue 23 34 45

8/01/24

22/1/24

1] WAP to Implement singly linked list with following operations

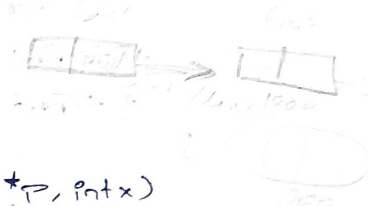
a] create a linked list

b] Insertion

c] Delete

Insert

```
struct Node
{
    int data;
    struct Node *Next;
};
*Head = Null;
```



```
Void insert first (struct Node *P, int x)
{
    struct Node *t;
    t=(struct Node *)Malloc (size of struct Node *);
    t -> next=Head;
    t => data =x;
    Head=t;
}

Void insert_last (struct Node *P, int x)
{
    while (P->next != Null)
        P=P->next;
    struct Node *t;
    t=(struct Node *)Malloc (size of struct Node *);
    t -> data =x;
    P -> next = t;
    t -> next =Null;
}
```

```
{  Struct node *q;
t = (Struct Node *)Malloc(Size of (Struct Node *));
t -> data = x;
for(struct Node
for (Pnt i=1; i< Pos; i++ ){
        q=P;
        P=P->next;
        t->next = P;
        q->next = t;
    }
Void delete_first (Struct Node *P)
{
    if (Head ->next==Null)
        { free (Head);
          return ;
        }
    Head = Head ->next;
    free (P);
}
Void delete_last (Struct Node *P)
    {
        Struct Node *q;
        While (P->next != Null){
            q=P
            P=P->next
        }
        q ->next = Null;
        free (P);
    }
```

```
Void delete_Pos (Struct Node *P, int Pos)
{
    Struct Node *q;
    For (int i=0 ; i< Pos; i++){
        q=P;
        P=P->next;
    }
    q->next = P->next;
    free (P);
}
Void display (Struct Node *P)
{
    While (P->next != Null)
    {
        int x= P->data;
        Print f(" %d \t", x);
        P=P->next;
    }
}
int main( ){
    insert_first (Head,8);
    insert_first (Head, 9);
    insert_last (Head ; 7)
    insert -Pos(Head, 2, 2);
    display (Head);
    delete_first (Head);
    display (Head );
    delete_last (Head);
    display (Head);
```

```
delete_Pos(head,2);
display(head);
insert_last(head,5);
display(head);

return 0;
}
```

Output

9 -> 2 -> 8 -> 7 -> Null

2 -> 8 -> 7 -> Null

2 -> 8 -> Null

2 -> Null

2 -> 5 -> Null

22/01/24

AIM: to implement doubly linked list with Primitive Operations

a] Create a doubly linked list

b] Insert a new node to the left of the node

c] Delete the node based on a specific value.

```
#include<stdio.h>
struct node
{ struct node *Prev;
  int data
  struct node* next;
} *Head=Null;

Create
Void createNode( int Value)
{
  Struct Node *newnode = ((Struct Node *) Malloc
                          (sizeof (Struct Node)));
  newnode -> data = value;
  newnode -> next = Null;
  newnode -> prev = Null;
  return new node;
}
Void insert_at_beg(int Value)
{
  Struct Node * ptr = Create node (Value);
  pt (Head == Null)
  {
    Head = ptr;
  }
else
  {
    ptr -> data = Value;
    Ptr -> next = Head;
    Ptr -> prev = Null;
    Head -> prev = ptr;
  } Head = ptr;
```

```
Void delete_from_Pos()
{
    Struct Node * temp, *Ptr;
    if (Head == Null)
    {
        Printf("Empty");
    }
    else
    int val;
    Printf("Enter the value");
    Scanf("%d", &val);
    temp = Head
    while(temp->data != val)
    {
        temp = temp->next;
        if(temp->next->next == Null)
        {
            temp->next = Null;
            Printf("Node deleted");
        }
        else
        {
            Ptr = temp->next;
            temp->next = Ptr->next;
            Ptr->next->prev = temp;
            free(Ptr);
            Printf("Node deleted");
        }
}
```

```
int main()
{
    insert_at_beg(5);
    insert_at_beg(6);
    insert_at_beg(23);
    insert_at_beg(34);
    insert_at_beg(37);
    display();
    delete_from_Pos(23);
    display();
}
```

Output

37 ← 34 ← 6 ← 5

Write a program

a] To construct Binary Search Tree.

b] Traverse the tree using inorder, Postorder, Preorder

c] Display the element in the tree.

```
#include <stdio.h>
#include <stdlib.h>

struct BST {
        int data;
        struct BST *left, *right;
    };

struct BST *root =NULL, *temp;

void create() {
    temp=(struct BST *) malloc (sizeof (struct BST));
    Printf("Enter data:");
    Scanf ("%d", &temp->data);
    temp->left = temp->right =Null;
    return temp;
  }

void insert struct BST *root, struct BST *temp)
{
    if (temp->data < root->data)
        if (root->left != Null)
            insert (root->left, temp);
        else
            root->left = temp;
    if (temp->data > root->data)
    {
        if (root->right != Null)
            insert (root->right, temp)
        else
            root->right = temp;
    }
```

```
void Inorder (struct BST *root)
  {
      if (root != Null)
        {
          inorder (root->left);
          Printf ("%d", root->data);
          inorder (root->right);
        }
    }

void Postorder (struct BST *root)
  {
    if (root != Null)
      {
        Post order(root->left);
        Post ordir (root->right);
        Printf ("%d", root->data);
      }
    }

void Preorder (struct BST *root)
  {
    if (root != Null)
      {
        Printf ("%d", root->data);
        Preorder (root->left);
        Preorder (root->right);
      }
  }

void main ()
{ int choice; char ch;
  Printf ("Enter Operation \n". 1. create \n
    2. display using inorder \n 3. display using
    Post order \n 4. display using Preorder \n ");
```

```
while()
{
  Printf("enter operation");
  Scanf("%d", &choice);

  if(choice==-1)
    { return ;}

  else
  { switch(choice)
    {
      case 1:
          {
            temp = create();
            if(root==NULL)
                root=temp;

            else
              insert(root,temp);

            Printf(" D
            break;
      case 2: Printf("element of tree are:");
              inorder(root);
              break;
      Case 3: Printf("elements of tree are:");
              Post order(root);
              break;
      Case 4: Printf("elements of tree are:");
              PreOrder(root);
      default: Printf("invalid operation");
    }
  }
}
```

Out PUt

1. Insert 2. inOrder 3. Postorder 4. Preorder

1
data : 4
1
data : 5
1
data : 2
1
data : 7
1 data : 9
1 data : 3

2
element of tree are; inorder: 2 3 4 5 7 9
element of tree are : postorder: 3 2 9 7 5 4
element of tree are : Preorder: 4 2 3 5 7 9

Middle of Linklist

```
Struct node{
    int data;
    Struct node* next;
}
Strut ListNode* deleteMiddle(struct ListNode*
    if(head==NULL)
      return head;
    if (head -> next==NULL)
      return NULL;
    Struct node *temp=head;
    int n=0;
    while (temp!=NULL){
        n++;
        temp = temp-> next;
    }
    temp= head;
    for (int i=0; i<n/2-1; i++){
        temp = temp-> next;
    }
    struct node* curr =temp -> next;
    temp -> next = curr -> next;
    free(curr)
    return head;
}
```

odd Even link list leetcode

```
struct node * oddEvenList(struct node*(head){
  if (head == NULL || head->next==NULL)
  {
    return head;
  }
  Struct node *odd =head;
  Struct node* even = head ->next;
  Struct event = even;
  while (even!=NULL && even ->next !=NULL)
  {
    odd =next = even ->next;
    *odd =odd ->next
    even ->next =odd ->next
    even = even ->next;
  }
  Odd -> next = event
    return
  }
```

19.02.24

Breadth first Search

```
#include <stdio.h>
void bfs (int a[10][10], int n, int u){
 {
    int f, r, q[10], v;
    int s[10]={0};
    Printf ("The nodes visited from %d:", u);
    f=0;
    r=-1
    q[++r]=u;
    s[u]=1;
    Printf ("%d", u);
    while(f<=r){
        u=q[f++];
        for (v=0; v<n; v++){
           if (a[u][v]==1 && s[v]==0)
           { Printf("%d", v);
             s[v]=1;
             q[++r]=v;
           }
        }
    }
    Printf("\n");
}
int main (){
    int n, a[10][10], i, j;
    Printf (" Enter no. of nodes: ");
    Scanf ("%d", &n);
    Printf (" Enter the adjcency Matrix: \n");
    for (i=0; i<n; i++){
        for (j=0; j<n; j++){
            Scanf ("%d", &a[i][j]);
        }
    }
    for (int source=0 ; source<n; source++){
            bfs(a, n, source);
    }
    return 0;
}
```

Output:

```
Enter no of nodes : 5
Enter the adjacency Matrix:
1 0 1 0 1
0 1 0 1 0
1 0 0 1 0
1 1 0 0 1
1 0 0 0 1
The nodes visited from 0 : 0 2 4 3 1
The nodes visited from 1 : 1 3 0 4 2
The nodes visited from 2 : 2 0 3 4 1
The nodes visited from 3 : 3 0 1 4 2
The nodes visited from 4 : 4 0 2 3 1
```

```c
#include<stdio.h>
void dfs(int a[10][10], int n, int u, int visited[]){
 int v;
 Printf("%d ", u);
 visited[u]= 1;
 for(v=0; v<n; v++){
      if(a[u][v] == 1 && !visited[v]){
          dfs(a,n,v, visited);
      }
     }
 }
int main(){
    int n, a[10][10], source, i, j;
    int visited[10] = {0};
    Printf("Enter the number of Vertices: ");
    Scanf("%d", &n);
    Printf("Enter the adjacency Matrix: ");
    for(i=0; i<n; i++){
        for(j=0; i<n; j++){
            Scanf("%d", &a[i][j]);
    }
  }
 Printf("DFS Traversal: ");
 for(source = 0; source<n; source++){
        if(!visited[source]){
            dfs(a,n, source, visited);
    }
  }
  return 0;
}
```

OUTPUT

```
Enter the number of Vertices: 5
Enter the adjacency matrix:
1 0 1 0 1
0 1 0 1 0
1 0 0 1 0
1 1 0 0 1
1 0 0 0 1
DFS Traversal: 0 2 3 1 4
```

513: Find Bottom Left Tree Value

```
int findBottomLeftValue(struct TreeNode* root){
    int value = root->val;
    int maxDepth = 0;
    void traverse(struct TreeNode *p, int depth){
        if(!p)
            return;
        if(depth > maxDepth){
            maxDepth = depth;
            value = p->val;
        }
        traverse(p->left, depth+1);
        traverse(p->right, depth+1);
    }
    traverse(root, 0);
    return value;
}
```

450: Delete Node from BST

```
struct TreeNode *deleteNode(struct TreeNode* root, int key){
    if(root){
        if(key < root->val)
            root->left = deleteNode(root->left, key);
        else if(key > root->val)
            root->right = deleteNode(root->right, key);
        else{
            if(!root->left && !root->right)
                return NULL;
            if(!root->left || !root->right)
                return root->left ? root->left : root->right;
            struct TreeNode* temp = root->left;
            while(temp->right != NULL)
                temp = temp->right;
            root->val = temp->val;
            root->left = deleteNode(root->left, temp->val);
        }
    }
    return root;
}
```

26.02.24