INSERTION DELETION AND TRAVERSALS -->{IN C }

struct node\* deletenode(struct node \* root,int data){

    struct node \*ipre=NULL;

    if(root==NULL){

        return NULL;

    }

    if(root->right==NULL && root->left==NULL)

    {

        free(root);

        return NULL;

    }

    else if(data<root->data)

    {

        root->left=deletenode(root->left,data);

    }

    else if(data>root->data)

    {

        root->right=deletenode(root->right,data);

    }

    else{

        ipre=inorderpre(root);

        root->data=ipre->data;

        root->left=deletenode(root->left,ipre->data);

    }

    return root;

}

void inorder(struct node\* root)

{

    if(root!=NULL)

    {

        inorder(root->left);

        printf("%d ",root->data);

        inorder(root->right);

    }

}

void insert(struct node\* root,int data)

{

    struct node\* prev=NULL;

    while(root!=NULL)

    {

        if(data<root->data)

        {

            prev=root;

            root=root->left;

        }

        else

        {

            prev=root;

            root=root->right;

        }

    }

    struct node\* n=createnode(data);

    if(n->data<prev->data)

    {

        prev->left=n;

    }

    else

    {

        prev->right=n;

    }

}

void preorder(struct node \*root) {

    if (root == NULL) {

        return;

    }

    struct stack \*stack = (struct stack \*)malloc(sizeof(struct stack));

    stack->size = 100;

    stack->top = -1;

    stack->arr = (int \*)malloc(stack->size \* sizeof(int));

    push(stack, root);

    while (!isEmpty(stack)) {

        struct node \*node = pop(stack);

        printf("%d ", node->data);

        if (node->right != NULL) {

            push(stack, node->right);

        }

        if (node->left != NULL) {

            push(stack, node->left);

        }

    }

}

void postorder(struct node \*root) {

    if (root == NULL) {

        return;

    }

    struct stack \*stack1 = (struct stack \*)malloc(sizeof(struct stack));

    struct stack \*stack2 = (struct stack \*)malloc(sizeof(struct stack));

    stack1->size = 100;

    stack1->top = -1;

    stack1->arr = (int \*)malloc(stack1->size \* sizeof(int));

    stack2->size = 100;

    stack2->top = -1;

    stack2->arr = (int \*)malloc(stack2->size \* sizeof(int));

    push(stack1, root);

    while (!isEmpty(stack1)) {

        struct node \*node = pop(stack1);

        push(stack2, node);

        if (node->left != NULL) {

            push(stack1, node->left);

        }

        if (node->right != NULL) {

            push(stack1, node->right);

        }

    }

    while (!isEmpty(stack2)) {

        struct node \*node = pop(stack2);

        printf("%d ", node->data);

    }

}

void inorder1(struct node \*root) {

    struct stack \*stack = NULL;

    struct node \*current = root;

    while (current != NULL || !isEmpty(stack)) {

        while (current != NULL) {

            push(stack, current);

            current = current->left;

        }

        current = pop(stack);

        printf("%d ", current->data);

        current = current->right;

    }

}

void inorder2(struct node \*root)

{   struct stack \*stack1 = (struct stack \*)malloc(sizeof(struct stack));

    struct stack \*stack2 = (struct stack \*)malloc(sizeof(struct stack));

    stack1->size = 100;

    stack1->top = -1;

    stack1->arr = (int \*)malloc(stack1->size \* sizeof(int));

    struct node \*temp=root;

    while(temp!=NULL)

    {

        push(stack1,temp);

        temp=temp->left;

    }

    while(!isEmpty(stack1))

    {

        temp=pop(stack1);

        printf("%d ",temp->data);

        temp=temp->right;

        while(temp!=NULL)

        {

            push(stack1,temp);

            temp=temp->left;

        }

    }

}