**Programming Lab**

**Assignment-5**

1. Write a menu-driven program for a binary tree using linked representation to

(a)Create (b) Preorder traversal (c) Inorder traversal (d) Postorder traversal

2. Write a menu-driven program for a binary tree using an array to

(a)Create (b) Preorder traversal (c) Inorder traversal (d) Postorder traversal

3. Implement a threaded binary tree (inorder)

4. Write a menu-driven program for a binary search tree to

(a) Create (b) search an element (c) insert element (d) delete an element

5. Write a menu-driven program to implement an AVL tree through functions

(a)Create (b) search an element (c) insert element (d) delete an element

**Program 1:**

#include<stdio.h>

#include<stdlib.h>

struct node{

    int data;

    struct node \*left;

    struct node \*right;

};

struct node\* createNode(int val){

    struct node \*newnode = (struct node \*)malloc(sizeof(struct node));

    newnode->data = val;

    newnode->left = NULL;

    newnode->right = NULL;

    return newnode;

}

struct node \*insert(struct node \*root,int val){

    if(root == NULL){

        root = createNode(val);

    } else if(val > root->data){

        root->right = insert(root->right,val);

    }else{

        root->left = insert(root->left,val);

    }

    return root;

}

struct node \*createTree(struct node \*root){

  int val;

    printf("Enter the value \n");

    scanf("%d",&val);

    if(val == -1){

        return NULL;

    }

    root = createNode(val);

    printf("Enter a value at the left of %d \n",val);

    root->left = createTree(root->left);

    printf("Enter a value at the right of %d \n",val);

    root->right = createTree(root->right);

    return root;

}

void inorder(struct node \*root){

    if(root != NULL){

        inorder(root->left);

        printf("%d ",root->data);

        inorder(root->right);

    }

}

void preorder(struct node \*root){

    if(root != NULL){

        printf("%d ",root->data);

        preorder(root->left);

        preorder(root->right);

    }

}

void postorder(struct node \*root){

    if(root != NULL){

        postorder(root->left);

        postorder(root->right);

        printf("%d ",root->data);

    }

}

int main(){

    struct node \*root = NULL;

    int op;

    printf("1 to create the tree\n2 to display the tree in inorder\n3 to display the tree in preorder\n4 to display the tree in postorder\n");

    while(1){

        printf("Enter your operation...\n");

        scanf("%d",&op);

        switch (op)

        {

        case 1:

            root = createTree(root);

            break;

        case 2:

            inorder(root);

            printf("\n");

            break;

        case 3:

            preorder(root);

            printf("\n");

            break;

        case 4:

            postorder(root);

            printf("\n");

            break;

        default:

            exit(0);

        }

    }

}

**Program 2:**

#include<stdio.h>

#include<stdlib.h>

void createTree(int arr[],int ind,int \*limit){

    int n;

    printf("Enter the value to insert...(-1 to exit)\n");

    scanf("%d",&n);

    if(n != -1){

        arr[ind] = n;

        \*limit = \*limit + 1;

        printf("Enter element at left of %d\n",n);

        createTree(arr,2\*ind,limit);

        printf("Enter element at right of %d\n",n);

        createTree(arr,2\*ind + 1,limit);

    }

}

void inorder(int arr[],int ind,int limit){

    if(ind <= limit){

        inorder(arr,2\*ind,limit);

        printf("%d ",arr[ind]);

        inorder(arr, 2\*ind + 1,limit);

    }

}

void preorder(int arr[],int ind,int limit){

    if(ind <= limit){

        printf("%d ",arr[ind]);

        preorder(arr,2\*ind,limit);

        preorder(arr, 2\*ind + 1,limit);

    }

}

void postorder(int arr[],int ind,int limit){

    if(ind <= limit){

        postorder(arr,2\*ind,limit);

        postorder(arr, 2\*ind + 1,limit);

        printf("%d ",arr[ind]);

    }

}

int main(){

    int op,arr[100],limit = 0;

    arr[0] = -1;

    printf("1 to create the tree(max 100 elements are allowed)\n2 to display the tree in inorder\n3 to display the tree in preorder\n4 to display the tree in postorder\n");

    while(1){

        printf("Enter your operation...\n");

        scanf("%d",&op);

        switch (op)

        {

        case 1:

            createTree(arr,1,&limit);

            printf("\n%d\n",limit);

            break;

        case 2:

            inorder(arr,1,limit);

            printf("\n");

            break;

        case 3:

            preorder(arr,1,limit);

            printf("\n");

            break;

        case 4:

            postorder(arr,1,limit);

            printf("\n");

            break;

        default:

            exit(0);

        }

    }

}

**Program 3:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

struct node {

    int data;

    struct node \*left;

    struct node \*right;

    int lthread;

    int rthread;

};

struct node \*createNode(int val){

    struct node \*newNode = (struct node \*)malloc(sizeof(struct node));

    newNode->data = val;

    newNode->left = NULL;

    newNode->right = NULL;

    newNode->lthread = 1;

    newNode->rthread = 1;

}

struct node \*insertNode(struct node \*root,int val){

    struct node \*queue[1000];

    struct node \*temp,\*newNode;

    int front = 0,top = 1;

    queue[front] = root;

    while(1){

        temp = queue[front];

        front++;

        if(temp->lthread == 1){

            newNode = createNode(val);

            newNode->left = temp->left;

            newNode->right = temp;

            temp->left = newNode;

            temp->lthread = 0;

            return root;

        } else {

            queue[top] = temp->left;

            top++;

        }

        if(temp->rthread == 1){

            newNode = createNode(val);

            newNode->right = temp->right;

            newNode->left = temp;

            temp->right = newNode;

            temp->rthread = 0;

            return root;

        } else {

            queue[top] = temp->right;

            top++;

        }

    }

}

struct node \*createTree(struct node \*root){

    int n;

    while(1){

        printf("Enter the value... ");

        scanf("%d",&n);

        if(n == -1)

            break;

        if(root == NULL)

            root = createNode(n);

        else

            root = insertNode(root,n);

    }

    return root;

}

struct node \*leftMost(struct node \*root){

    while(root->lthread == 0)

        root = root->left;

    return root;

}

void inorder(struct node \*root){

    struct node \*cur = leftMost(root);

    int count = 0;

    while(cur){

        if(count == 10)

            break;

        printf("%d ",cur->data);

        if(cur->rthread)

            cur = cur->right;

        else

            cur = leftMost(cur->right);

        count++;

    }

}

int main(){

    struct node \*root = NULL;

    root = createTree(root);

    printf("Inorder of the tree is: ");

    inorder(root);

}

**Program 4:**

#include<stdio.h>

#include<stdlib.h>

struct node{

    int data;

    struct node \*left;

    struct node \*right;

};

struct node\* createNode(int val){

    struct node \*newnode = (struct node \*)malloc(sizeof(struct node));

    newnode->data = val;

    newnode->left = NULL;

    newnode->right = NULL;

    return newnode;

}

struct node \*insert(struct node \*root,int val){

    if(root == NULL){

        root = createNode(val);

    } else if(val > root->data){

        root->right = insert(root->right,val);

    }else{

        root->left = insert(root->left,val);

    }

    return root;

}

struct node \*createTree(struct node \*root){

    int val;

    printf("Enter the value...(-1 to exit)\n");

    while (1)

    {

        scanf("%d",&val);

        if(val == -1){

            break;

        }

        root = insert(root,val);

    }

    return root;

}

void levelorder(struct node \*root){

    struct node \*queue[100];

    for(int j=0;j<100;j++){

        queue[j] = NULL;

    }

    int front = 0,i = 2;

    queue[0] = root;

    while(1){

        if(queue[front] == 0){

            i++;

            printf("\n");

            if(queue[front + 1] == 0)

                break;

        } else {

            printf("%d ",queue[front]->data);

            if(queue[front]->left)

                queue[i++] = queue[front]->left;

            if(queue[front]->right)

                queue[i++] = queue[front]->right;

        }

        front++;

    }

}

void search(struct node \*root,int val){

    if(root == NULL)

        printf("Element doesn't exist...\n");

    else{

        if(root->data == val)

            printf("It exists...\n");

        else if(val > root->data)

            search(root->right,val);

        else

            search(root->left,val);

    }

}

int get\_max(struct node \*root){

    int val;

    struct node \*temp = root;

    while (temp->right){

        temp = temp->right;

    }

    return temp->data;

}

struct node \*delete(struct node \*root,int val){

    if(root->data == val){

        // no child

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

            return NULL;

        // only left

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

            return root->left;

        // only right

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

            return root->right;

        // both child

        else {

            int max\_left = get\_max(root->left);

            root->data = max\_left;

            root->left = delete(root->left,max\_left);

        }

    } else if(val > root->data)

        root->right = delete(root->right,val);

    else

        root->left = delete(root->left,val);

    return root;

}

int main(){

    struct node \*root = NULL;

    int op,n;

    printf("1 to create the tree\n2 to display the tree in levelorder\n3 to insert an element\n4 to search an element\n5 to delete an element\n");

    while(1){

        printf("Enter your operation...\n");

        scanf("%d",&op);

        switch (op)

        {

        case 1:

            root = createTree(root);

            break;

        case 2:

            levelorder(root);

            printf("\n");

            break;

        case 3:

            printf("Enter the lement to insert...\n");

            scanf("%d",&n);

            root = insert(root,n);

            printf("Element inserted...\n");

            break;

        case 4:

            printf("Enter the element to search...\n");

            scanf("%d",&n);

            search(root,n);

            break;

        case 5:

            printf("Enter the element to delete...\n");

            scanf("%d",&n);

            root = delete(root,n);

            printf("Element deleted...\n");

            break;

        default:

            exit(0);

        }

    }}

**Program 5:**

#include<stdio.h>

#include<stdlib.h>

struct node{

    int data;

    struct node \*left,\*right;

    int height;

};

struct node \*createNode(int val){

    struct node \*newNode = (struct node \*)malloc(sizeof(struct node));

    newNode->data = val;

    newNode->left = NULL;

    newNode->right = NULL;

    newNode->height = 1;

    return newNode;

}

int max(int a,int b){

    return (a > b)? a:b;

}

int height(struct node \*node){

    if(node == NULL)

        return 0;

    return node->height;

}

int getBalance(struct node \*node){

    if(node == NULL)

        return 0;

    return height(node->left) - height(node->right);

}

int get\_max(struct node \*root){

    int val;

    struct node \*temp = root;

    while (temp->right){

        temp = temp->right;

    }

    return temp->data;

}

struct node \*rightRotation(struct node \*a){

    struct node \*b = a->left;

    struct node \*TR = b->right;

    // rotation

    b->right = a;

    a->left = TR;

    a->height = max(height(a->left),height(a->right)) + 1;

    b->height = max(height(b->left),height(b->right)) + 1;

    return b;

}

struct node \*leftRotation(struct node \*a){

    struct node \*b = a->right;

    struct node \*TL = b->left;

    // rotation

    b->left = a;

    a->right = TL;

    a->height = max(height(a->left),height(a->right)) + 1;

    b->height = max(height(b->left),height(b->right)) + 1;

    return b;

}

struct node \*insertNode(struct node \*root,int val){

    if(root == NULL)

        return createNode(val);

    else if(val > root->data)

        root->right = insertNode(root->right,val);

    else

        root->left = insertNode(root->left,val);

    root->height = max(height(root->left),height(root->right)) + 1;

    int balance = getBalance(root);

    // left left

    if(balance > 1 && (val < root->left->data))

        return rightRotation(root);

    // right right

    else if(balance < -1 && (val > root->right->data))

        return leftRotation(root);

    // left right

    else if(balance > 1 && (val > root->left->data)){

        root->left = leftRotation(root->left);

        return rightRotation(root);

    }

    // right left

    else if(balance < -1 && (val < root->right->data)){

        root->right = rightRotation(root->right);

        return leftRotation(root);

    }

    return root;

}

struct node \*createTree(struct node \*root){

    int n;

    while(1){

        printf("Enter the value: ");

        scanf("%d",&n);

        if(n == -1)

            break;

        root = insertNode(root,n);

    }

    return root;

}

void search(struct node \*root,int val){

    if(root == NULL)

        printf("Element doesn't exist...\n");

    else{

        if(root->data == val)

            printf("It exists...\n");

        else if(val > root->data)

            search(root->right,val);

        else

            search(root->left,val);

    }

}

struct node \*delete(struct node \*root,int val){

    if(root->data == val){

        // no child

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

            return NULL;

        // only left

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

            return root->left;

        // only right

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

            return root->right;

        // both child

        else {

            int max\_left = get\_max(root->left);

            root->data = max\_left;

            root->left = delete(root->left,max\_left);

        }

    } else if(val > root->data)

        root->right = delete(root->right,val);

    else

        root->left = delete(root->left,val);

    root->height = max(height(root->left),height(root->right)) + 1;

    int balance = getBalance(root);

    // left left

    if(balance > 1 && getBalance(root->left) >=0)

        return rightRotation(root);

    // right right

    else if(balance < -1 && getBalance(root->right) <= 0)

        return leftRotation(root);

    // left right

    else if(balance > 1 && getBalance(root->left) < 0){

        root->left = leftRotation(root->left);

        return rightRotation(root);

    }

    // right left

    else if(balance < -1 && getBalance(root->right) > 0){

        root->right = rightRotation(root->right);

        return leftRotation(root);

    }

    return root;

}

void preorder(struct node \*root){

    if(root){

        printf("%d ",root->data);

        preorder(root->left);

        preorder(root->right);

    }

}

int main(){

    struct node \*root = NULL;

    int op,n;

    printf("1 to create the tree\n2 to display the tree in preorder\n3 to insert an element\n4 to search an element\n5 to delete an element\n");

    while(1){

        printf("Enter your operation...\n");

        scanf("%d",&op);

        switch (op)

        {

        case 1:

            root = createTree(root);

            break;

        case 2:

            preorder(root);

            printf("\n");

            break;

        case 3:

            printf("Enter the lement to insert...\n");

            scanf("%d",&n);

            root = insertNode(root,n);

            printf("Element inserted...\n");

            break;

        case 4:

            printf("Enter the element to search...\n");

            scanf("%d",&n);

            search(root,n);

            break;

        case 5:

            printf("Enter the element to delete...\n");

            scanf("%d",&n);

            root = delete(root,n);

            printf("Element deleted...\n");

            break;

        default:

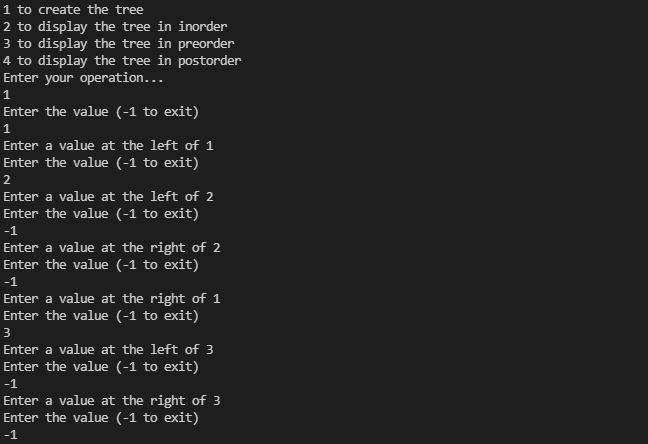
            exit(0);

        }

    }

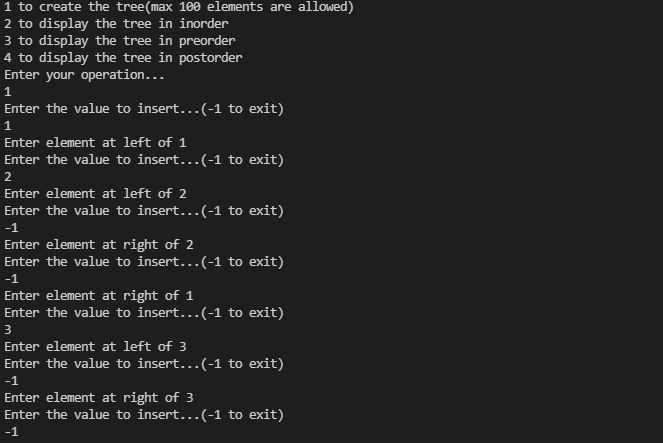
}

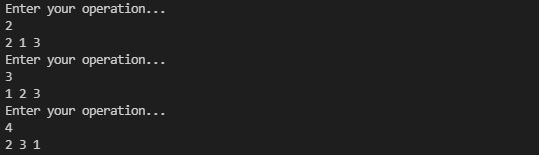
**Output 1:**

****

****

**Output 2:**

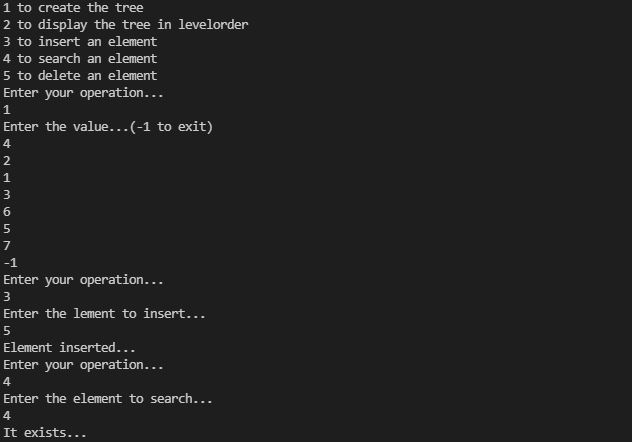
****

****

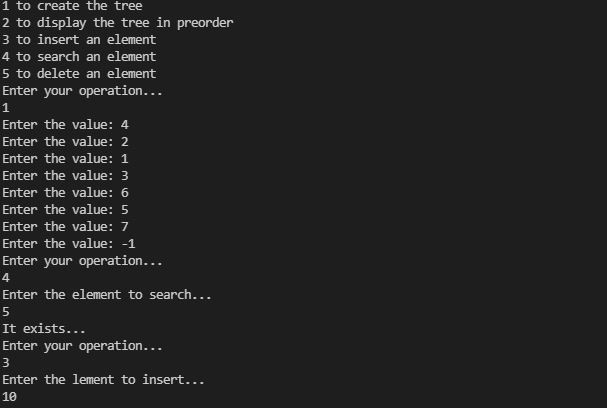
**Output 3:**



**Output 4:**

****

**Output 5:**

****

