

Lab 10

LAB 10:-

Program:-

Write a program

- a) To construct a binary Search tree.
- b) To traverse the tree using all the methods i.e., in-order, preorder and post order
- c) To display the elements in the tree.

Program:-

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

struct node
{
    int info;
    struct node *rlink;
    struct node *llink;
};

typedef struct node *NODE;

NODE getnode()
{
    NODE x;
    x=(NODE)malloc(sizeof(struct node));
    if(x==NULL)
    {
        printf("mem full\n");
        exit(0);
    }
    return x;
}

void freenode(NODE x)
```

```
{  
    free(x);  
}  
  
NODE insert(NODE root,int item)  
{  
    NODE temp,cur,prev;  
    temp=getnode();  
    temp->rlink=NULL;  
    temp->llink=NULL;  
    temp->info=item;  
    if(root==NULL)  
        return temp;  
    prev=NULL;  
    cur=root;  
    while(cur!=NULL)  
    {  
        prev=cur;  
        cur=(item<cur->info)?cur->llink:cur->rlink;  
    }  
    if(item<prev->info)  
        prev->llink=temp;  
    else  
        prev->rlink=temp;  
    return root;  
}  
  
void display(NODE root,int i)  
{  
    int j;
```

```

if(root!=NULL)
{
    display(root->rlink,i+1);
    for(j=0;j<i;j++)
        printf("   ");
    printf("%d\n",root->info);
    display(root->llink,i+1);
}
}

NODE delete(NODE root,int item)
{
    NODE cur,parent,q,suc;
    if(root==NULL)
    {
        printf("empty\n");
        return root;
    }
    parent=NULL;
    cur=root;
    while(cur!=NULL&&item!=cur->info)
    {
        parent=cur;
        cur=(item<cur->info)?cur->llink:cur->rlink;
    }
    if(cur==NULL)
    {
        printf("not found\n");
        return root;
    }
}

```

```

    }

    if(cur->llink==NULL)
        q=cur->rlink;
    else if(cur->rlink==NULL)
        q=cur->llink;
    else
    {
        suc=cur->rlink;
        while(suc->llink!=NULL)
            suc=suc->llink;
        suc->llink=cur->llink;
        q=cur->rlink;
    }
    if(parent==NULL)
        return q;
    if(cur==parent->llink)
        parent->llink=q;
    else
        parent->rlink=q;
    freenode(cur);
    return root;
}

```

```

void preorder(NODE root)
{
    if(root!=NULL)
    {
        printf("%d\n",root->info);

```

```
    preorder(root->llink);
    preorder(root->rlink);
}
}

void postorder(NODE root)
{
if(root!=NULL)
{
    postorder(root->llink);
    postorder(root->rlink);
    printf("%d\n",root->info);
}
}

void inorder(NODE root)
{
if(root!=NULL)
{
    inorder(root->llink);
    printf("%d\n",root->info);
    inorder(root->rlink);
}
}

int main()
{
int item,choice;
NODE root=NULL;
```

```
for(;;)
{
printf("\n1.insert\n2.display\n3.pre\n4.post\n5.in\n6.delete\n7.exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
{
    case 1:printf("enter the item\n");
              scanf("%d",&item);
              root=insert(root,item);
              break;
    case 2:display(root,0);
              break;
    case 3:preorder(root);
              break;
    case 4:postorder(root);
              break;
    case 5:inorder(root);
              break;
    case 6:printf("enter the item\n");
              scanf("%d",&item);
              root=delete(root,item);
              break;
    default:exit(0);
              break;
}
}
```

Screenshots:-

```
D:\dsBST.exe

1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
1
enter the item
10

1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
1
enter the item
20

1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
1
enter the item
30

1.insert
2.display
3.pre
4.post
```

```
D:\ddsBST.exe
6.delete
7.exit
enter the choice
1
enter the item
5
5

1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
1
enter the item
2
2

1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
2
2
30
20
10
5
2

1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
```

```
D:\d\BST.exe
enter the choice
3
10
5
2
20
30
1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
4
2
5
30
20
10
1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
5
2
5
10
20
30
1.insert
2.display
3.pre
```

```
D:\d\BST.exe
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
6
enter the item
2

1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
2
    30
    20
10
5

1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
```

Written:-

Lab 10

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int info;
    struct node *rlink;
    struct node *llink;
};
typedef struct node * NODE;
```

```
NODE getnode()
{
    NODE x;
    x = (NODE) malloc (sizeof(struct node));
    if (x == NULL)
    {
        printf("memfull\n");
    }
}
```

```
void freenode(NODE x)
{
    free(x);
}
```

```
NODE insert (NODE root, int item)
{
    NODE temp, cur, prev;
    temp = getnode ();
    temp->rlink = NULL;
    temp->llink = NULL;
    temp->info = item;
    if (root == NULL)
        return temp;
```

prev = NULL;
cur = root;
while (cur != NULL)
{
 prev = cur;
 cur = cur->link;
 cur = (item < cur->info) ? cur->link : cur->rlink;
}
if (~~temp~~ item < prev->info)
 prev->llink = temp;
else
 prev->rlink = temp;
return root;

}

void display (NODE root, int i)
{
 int j;
 if (root != NULL)
 {
 display (root->rlink, i+1);
 for (j=0; j<i; j++)
 printf (" ");
 printf ("%d\n", root->info);
 display (root->link, i+1);
 }
}

NODE delete (NODE root, int item)
{

NODE cur, parent, q, rlc;
if (root == NULL)
{
 printf ("empty\n");

return root;

}

parent = NULL;

cur = root;

while (cur != NULL && item != cur->info)

{

parent = cur;

cur = (item < cur->info) ? cur->link : cur->rlink;

}

if (cur == NULL)

printf("not found %d");

return root;

}

if (cur->llink == NULL)

q = cur->rlink;

else if (cur->rlink == NULL)

q = cur->llink;

else

{

suc = cur->rlink;

while (suc->llink != NULL)

suc = suc->llink;

suc->llink = cur->llink;

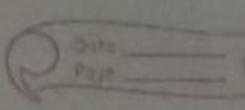
q = cur->rlink;

}

freemode(cur);

return root;

}



```
#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node *left;
    Node *right;
};

Node* insert(Node* root, int item) {
    if (root == NULL) {
        root = new Node();
        root->data = item;
        return root;
    }
    if (item < root->data)
        root->left = insert(root->left, item);
    else
        root->right = insert(root->right, item);
    return root;
}

void display(Node* root) {
    if (root != NULL) {
        cout << root->data << " ";
        display(root->left);
        display(root->right);
    }
}

void preOrder(Node* root) {
    if (root != NULL) {
        cout << root->data << " ";
        preOrder(root->left);
        preOrder(root->right);
    }
}

void postOrder(Node* root) {
    if (root != NULL) {
        postOrder(root->left);
        postOrder(root->right);
        cout << root->data << " ";
    }
}

void inOrder(Node* root) {
    if (root != NULL) {
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}

Node* delete(Node* root, int item) {
    if (root == NULL)
        return root;
    if (item < root->data)
        root->left = delete(root->left, item);
    else if (item > root->data)
        root->right = delete(root->right, item);
    else {
        if (root->left == NULL || root->right == NULL) {
            Node* temp = root;
            root = (root->left) ? root->left : root->right;
            delete(temp);
        } else {
            Node* temp = root->right;
            while (temp->left != NULL)
                temp = temp->left;
            root->data = temp->data;
            root->right = delete(root->right, temp->data);
        }
    }
    return root;
}

int main() {
    Node* root = NULL;
    char choice;
    int item;
    cout << "1. Insert 2. display 3. pre order 4. post order 5. delete 6. exit\n";
    cin >> choice;
    switch (choice) {
        case '1':
            cout << "enter the item\n";
            cin >> item;
            root = insert(root, item);
            break;
        case '2':
            display(root);
            break;
        case '3':
            preOrder(root);
            break;
        case '4':
            postOrder(root);
            break;
        case '5':
            inOrder(root);
            break;
        case '6':
            cout << "enter the item\n";
            cin >> item;
            root = delete(root, item);
            break;
        default:
            cout << "invalid choice\n";
    }
}
```

