**PROGRAMS:** Binary Search Tree

**Input:**

#include<stdio.h>

struct Node{

    int data;

    struct Node \*rchild;

    struct Node \*lchild;

};

struct Node \*root=NULL;

void insertnr(){

    int x;

    printf("Enter element to insert: ");

    scanf("%d", &x);

    struct Node \*temp, \*ptr, \*p;

    ptr=root;

    p=NULL;

    while(ptr!=NULL){

        p=ptr;

        if(x<ptr->data)

            ptr=ptr->lchild;

        else if(x>ptr->data)

            ptr=ptr->rchild;

        else{

            printf("element already present\n");

            return root;

        }

    }

    temp=(struct Node\*)malloc(sizeof(struct Node));

    temp->data=x;

    temp->lchild=NULL;

    temp->rchild=NULL;

    if(p==NULL)

        root=temp;

    else if(x<p->data)

        p->lchild=temp;

    else

        p->rchild=temp;

        return root;

}

Expt no: 10

struct Node \*insert\_r(struct Node\*p, int x){

    if(p==NULL){

        p=(struct Node\*)malloc(sizeof(struct Node));

        p->data=x;

        p->lchild=NULL;

        p->rchild=NULL;

    }

    else if(x<p->data)

        p->lchild=insert\_r(p->lchild, x);

     else if(x>p->data)

        p->rchild=insert\_r(p->rchild, x);

    else

        printf("Duplicate Key\n");

        return p;

}

struct Node \*del\_child(struct Node\* n, int x){

    if(n==NULL)

        return n;

    if(n->data>x){

        n->lchild=del\_child(n->lchild, x);

        return n;

    }

    else if(n->data<x){

        n->rchild=del\_child(n->rchild, x);

        return n;

    }

    if (n->lchild == NULL) {

        struct Node\* temp = n->rchild;

        free(n);

        return temp;

    }

    else if (n->rchild == NULL) {

        struct Node\* temp = n->lchild;

        free(n);

        return temp;

    }

    else {

        struct Node\* succ1 = n;

        struct Node\* succ = n->rchild;

        while (succ->lchild!= NULL) {

            succ1 = succ;

            succ = succ->lchild;

        }

        if (succ1 != n)

            succ1->lchild = succ->rchild;

        else

            succ1->rchild = succ->rchild;

        n->data = succ->data;

        free(succ);

        return n;

    }

}

void In\_order(struct Node \*root){

   if(root!=NULL){

    In\_order(root->lchild);

    printf(" %d ", root->data);

    In\_order(root->rchild);

   }

}

void Post\_order(struct Node \*root){

    if(root==NULL)

        return;

    else

        Post\_order(root->lchild);

        Post\_order(root->rchild);

        printf(" %d ", root->data);

}

void Pre\_order(struct Node\* root){

    if(root==NULL){

Expt no: 10

        return;

    }

    else{

        printf(" %d ", root->data);

        Pre\_order(root->lchild);

        Pre\_order(root->rchild);

    }

}

int main()

{

    int a, x, y;

    printf("1.Insertion Without Recursion\n2.Insert using Recursion\n3.Deletion\n4.Inorder Traversal\n5.Preorder Traversal\n6.Postorder Traversal\n7.Exit\n");

    while(1)

    {

        printf("\nEnter choice: ");

        scanf("%d", &a);

        switch(a)

        {

            case 1:

                insertnr();

                break;

            case 2:

                printf("Enter element to insert: ");

                scanf("%d", &x);

                insert\_r(root, x);

                break;

            case 3:

                printf("Enter element to delete: ");

                scanf("%d", &y);

                del\_child(root, y);

                break;

            case 4:

                In\_order(root);

                printf("\n");

                break;

            case 5:

                Pre\_order(root);

                printf("\n");

                break;

            case 6:

                Post\_order(root);

                printf("\n");

                break;

            case 7:

                exit(1);

            default:

                printf("Invalid Choice\n");

                break;

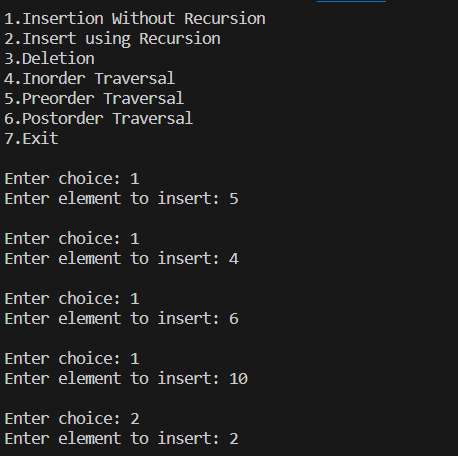
        }

    }

}

**Output:**

1. Insertion:



1. Inorder traversal:

****

1. Preorder traversal:

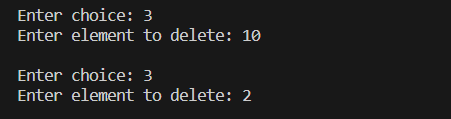
****

1. Postorder traversal:

****

1. Deletion:

Expt no: 10

****

1. Inorder traversal:

****