C CODE

```
/*Program to implement TOH Problem */
#include<stdio.h>
#include<conio.h>
void transfer(int n, char from, char to, char temp);

void main(){
    int n;
    char ch;
    do{
      printf("\nEnter the no. of disks : ");
      scanf("%d", &n);
      transfer(n,'L','R','C');
      fseek(stdin,0,SEEK_END);
            printf("\n\n Do you want to look steps for other number of
                  disks (Y/N)? ");
            scanf("%c",&ch);
      }
      while(ch=='Y' || ch=='Y');
      printf("\n\t Program ended successfully !!!");
}

void transfer(int n, char from, char to, char temp){
    if(n==0)
        return;
    else{
        transfer(n-1,from,temp,to);
        printf("\n Move disk %d from %c to %c",n,from,to);
        transfer(n-1,temp,to,from);
    }
}
```

OUTPUT

```
Enter the no. of disks : 4

Enter the no. of disks : 1

 Move disk 1 from L to R
 Do you want to look steps for other number of disks (Y/N)? Y

Enter the no. of disks : 2

 Move disk 1 from L to C
 Move disk 2 from L to R
 Move disk 1 from C to R

 Do you want to look steps for other number of disks (Y/N)? Y

Enter the no. of disks : 3

 Move disk 1 from L to R
 Move disk 2 from L to C
 Move disk 1 from R to C
 Move disk 3 from L to R
 Move disk 1 from C to L
 Move disk 2 from C to R
 Move disk 1 from L to R

 Do you want to look steps for other number of disks (Y/N)? Y

Enter the no. of disks : 4

 Move disk 1 from L to C
 Move disk 2 from L to R
 Move disk 1 from C to R
 Move disk 3 from L to C
 Move disk 1 from R to L
 Move disk 2 from R to C
 Move disk 1 from L to C
 Move disk 4 from L to R
 Move disk 1 from C to R
 Move disk 2 from C to L
 Move disk 1 from R to L
 Move disk 3 from C to R
 Move disk 1 from L to C
 Move disk 2 from L to R
 Move disk 1 from C to R

 Do you want to look steps for other number of disks (Y/N)? N

         Program ended successfully !!!
--------------------------------
Process exited with return value 33
Press any key to continue . . .
```

C CODE

```
/* Program to implement Binary Search Tree */
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

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

typedef struct BST BST;

BST *root;

BST *insert(BST *root, int num);
void preorder(BST *);
void inorder(BST *);
void postorder(BST *);
void search(BST *,int);
BST *del(BST*, int);
BST *findmin(BST *);
BST *findmax(BST *);
void destroy(BST *);
void internal(BST *);
void external(BST *);
void child(BST *);

int main() {
    BST *temp;
    int ch, num;
    printf("\n\t 1. Insert an element ");
    printf("\n\t 2. Preorder traversal");
    printf("\n\t 3. Inorder traversal");
    printf("\n\t 4. Postorder traversal");
    printf("\n\t 5. Search a value");
    printf("\n\t 6. Delete a node");
    printf("\n\t 7. Find min node");
    printf("\n\t 8. Find max node");
    printf("\n\t 9. Destroy binary tree");
    printf("\n\t 10. Display internal nodes");
    printf("\n\t 11. Display external nodes");
    printf("\n\t 12. Display nodes with single child");
    printf("\n\t 13. Exit");
    while(1){
        printf("\n Enter option: ");
        scanf("%d",&ch);
        switch (ch){
        case 1:
            printf("Enter element to insert: ");
            scanf("%d",&num);
            root = insert(root,num);
            break;
        case 2:
            if (root==NULL)
                printf("Empty tree");
            else{
                printf("Elements in preorder traversal are :- \n");
                preorder(root);
            }
            break;
        case 3:
            if (root==NULL)
                printf("Empty tree");
            else{
                printf("Elements in inorder traversal are :- \n");
                inorder(root);
            }
            break;
        case 4:
            if (root==NULL)
                printf("Empty tree");
            else{
                printf("Elements in postorder traversal are :- \n");
                postorder(root);
            }
            break;
        case 5:
            if (root==NULL)
                printf("Empty tree");
            else{
                 printf("Enter a number to search : ");
                 scanf("%d",&num);
                 search(root,num);
                 }
            break;
        case 6:
            if (root==NULL)
                printf("Empty tree");
            else{
                 printf("Enter a number to delete : ");
                 scanf("%d",&num);
                 root=del(root,num);
                 }
            break;
        case 7:
            temp=findmin(root);
            if (temp==NULL)
                printf("Empty tree");
            else
                 printf("The minimum node is : %d",temp->info);
            break;
        case 8:
           temp=findmax(root);
            if (temp==NULL)
                printf("Empty tree");
            else
                 printf("The maximum node is : %d",temp->info);
            break;
        case 9:
           destroy(root);
           root=NULL;
            printf("\n\t The tree is now empty");
            break;
        case 10:
            if(root==NULL)
                printf("Empty tree");
            else{
                printf("The internal roots are :- \n");
                internal(root);
```

```
            }
            break;
        case 11:
            if(root==NULL)
                printf("Empty tree");
            else{
                printf("The external roots are :- \n");
                external(root);
            }
            break;
        case 12:
            if(root==NULL)
                printf("Empty tree");
            else{
                printf("The nodes with single child are :- \n");
                child(root);
            }
            break;
        case 13:
            printf("Program ended successfully !!!");
            break;
        default:
            printf("Please enter correct choice !!!\n");
            break;
        }
        if(ch==13)
            break;
    }
}

BST *insert(BST *r, int num){
    if(r==NULL){
        r=(BST*)malloc(sizeof(BST));
        r->info=num;
        r->left=NULL;
        r->right=NULL;
    }
    else if(num<r->info)
        r->left=insert(r->left,num);
    else if(num>r->info)
        r->right=insert(r->right,num);
    else if(r->info==num){
        printf("The node already exist \n");
    }
    return r;
}
void preorder(BST *root){
    if(root!=NULL){
        printf("\t%d", root->info);
        preorder(root->left);
        preorder(root->right);
    }
}
void inorder(BST *root){
    if(root!=NULL){
        inorder(root->left);
        printf("\t%d", root->info);
        inorder(root->right);
    }
}
void postorder(BST *root){
    if(root!=NULL){
        postorder(root->left);
        postorder(root->right);
        printf("\t%d", root->info);
    }
}
void search(BST *root,int num){
      if(root==NULL)
            printf("\t Search unsuccesful !!!");
      else if(num==root->info)
            printf("\t Search successful !!! \n\t Number %d is found",
                  root->info);
      else if(num < root->info)
            search(root->left,num);
      else
            search(root->right,num);
}
BST *del(BST *r, int num){
      BST *temp, *root=r;
      if(root==NULL){
            printf("No such node available");
            return root;
      }
    else if(num<root->info)
        root->left=del(root->left,num);
    else if(num>root->info)
        root->right=del(root->right,num);
    else if(root->left!=NULL && root->right!=NULL){
        temp=findmin(root->right);
        root->info=temp->info;
        root->right=del(root->right,root->info);
    }
    else{
        temp=root;
        if(root->left==NULL)
            root=root->right;
        else if(root->right=NULL)
            root=root->left;
        free(temp);
    }
    return root;
}
BST *findmin(BST *root){
    BST *temp=root;
    if(temp==NULL)
        return NULL;
    if(temp->left==NULL)
        return temp;
    else
        return findmin(temp->left);
}
BST *findmax(BST *root){
    BST *temp=root;
    if(temp==NULL)
        return NULL;
    if(temp->right==NULL)
        return temp;
    else
        return findmax(temp->right);
}
void destroy(BST *root){
    BST *t=root;
    if(t!=NULL){
        destroy(t->left);
        destroy(t->right);
        printf("\n Value %d is deleted",t->info);
        free(t);
    }
}
void internal(BST *root){
    if(root!=NULL){
        if(root->left!=NULL || root->right!=NULL)
            printf("\t%d",root->info);
        internal(root->left);
        internal(root->right);
    }
}
void external(BST *root){
    if(root!=NULL){
        if(root->left==NULL && root->right==NULL)
            printf("\t%d",root->info);
        external(root->left);
        external(root->right);
    }
}
void child(BST *root){
    if(root!=NULL){
        if(root->left==NULL){
            if(root->right!=NULL)
                printf("\t%d",root->info);
        }
        else if(root->right==NULL){
            if(root->left!=NULL)
                printf("\t%d",root->info);
        }
        child(root->left);
        child(root->right);
    }
}
```

OUTPUT

```
        1. Insert an element
        2. Preorder traversal
        3. Inorder traversal
        4. Postorder traversal
        5. Search a value
        6. Delete a node
        7. Find min node
        8. Find max node
        9. Destroy binary tree
        10. Display internal nodes
        11. Display external nodes
        12. Display nodes with single child
        13. Exit
 Enter option: 1
```

```
Enter element to insert: 20

 Enter option: 1
Enter element to insert: 17

 Enter option: 1
Enter element to insert: 6

 Enter option: 1
Enter element to insert: 8

 Enter option: 1
Enter element to insert: 10

 Enter option: 1
Enter element to insert: 7

 Enter option: 1
Enter element to insert: 18

 Enter option: 1
Enter element to insert: 13

 Enter option: 1
Enter element to insert: 12

 Enter option: 1
Enter element to insert: 5

 Enter option: 12
The nodes with single child are :-
        20      10      13
 Enter option: 2
Elements in preorder traversal are :-
        20      17      6       5       8       7       10      13      12      18
 Enter option: 3
Elements in inorder traversal are :-
        5       6       7       8       10      12      13      17      18      20
 Enter option: 4
Elements in postorder traversal are :-
        5       7       12      13      10      8       6       18      17      20
 Enter option: 10
The internal roots are :-
        20      17      6       8       10      13
 Enter option: 11
The external roots are :-
        5       7       12      18
 Enter option: 12
The nodes with single child are :-
        20      10      13
 Enter option: 8
The maximum node is : 20
```

```
 Enter option: 7
The minimum node is : 5
 Enter option: 9

 Value 5 is deleted
 Value 7 is deleted
 Value 12 is deleted
 Value 13 is deleted
 Value 10 is deleted
 Value 8 is deleted
 Value 6 is deleted
 Value 18 is deleted
 Value 17 is deleted
 Value 20 is deleted
        The tree is now empty
 Enter option: 2
Empty tree
 Enter option: 1
Enter element to insert: 62

 Enter option: 1
Enter element to insert: 50

 Enter option: 1
Enter element to insert: 75

 Enter option: 1
Enter element to insert: 60

 Enter option: 1
Enter element to insert: 65

 Enter option: 1
Enter element to insert: 90

 Enter option: 1
Enter element to insert: 80

 Enter option: 1
Enter element to insert: 85

 Enter option: 2
Elements in preorder traversal are :-
        62      50      60      75      65      90      80      85
 Enter option: 3
Elements in inorder traversal are :-
        50      60      62      65      75      80      85      90
 Enter option: 4
Elements in postorder traversal are :-
        60      50      65      85      80      90      75      62
 Enter option: 10
The internal roots are :-
```

```
        62      50      75      90      80
 Enter option: 11
The external roots are :-
        60      65      85
 Enter option: 12
The nodes with single child are :-
        50      90      80
 Enter option: 6
Enter a number to delete : 85
 Enter option: 2
Elements in preorder traversal are :-
        62      50      60      75      65      90      80
 Enter option: 3
Elements in inorder traversal are :-
        50      60      62      65      75      80      90
 Enter option: 4
Elements in postorder traversal are :-
        60      50      65      80      90      75      62
 Enter option: 10
The internal roots are :-
        62      50      75      90
 Enter option: 11
The external roots are :-
        60      65      80
 Enter option: 12
The nodes with single child are :-
        50      90
 Enter option: 6
Enter a number to delete : 50
 Enter option: 2
Elements in preorder traversal are :-
        62      60      75      65      90      80
 Enter option: 3
Elements in inorder traversal are :-
        60      62      65      75      80      90
 Enter option: 4
Elements in postorder traversal are :-
        60      65      80      90      75      62
 Enter option: 10
The internal roots are :-
        62      75      90
 Enter option: 11
The external roots are :-
        60      65      80
 Enter option: 12
The nodes with single child are :-
        90
 Enter option: 13
Program ended successfully !!!
--------------------------------
Process exited with return value 13
Press any key to continue . . .
```