Navlakhi®

BST

Methodology and Program

By Abhishek Navlakhi

Semester 3: Data Structures

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


struct tree_node
{
int data;
struct tree_node *left;
struct tree_node *right;
};

struct headnode
{
int count;
struct tree_node *head;
}*tree;


void createTree( )
{
tree=(struct headnode*)malloc(sizeof(struct headnode));
tree->count=0;
tree->head=NULL;
}
```

```
void insertNode( int dataIn )
{
 struct tree_node *pNew,*pLoc;
 if (tree->count==0)
 {      tree->head=(struct tree_node*)malloc(sizeof(struct tree_node));
        tree->head->left=NULL;
        tree->head->right=NULL;
        tree->head->data=dataIn;
        tree->count=1;
 }
 else
 {
        pNew=(struct tree_node*)malloc(sizeof(struct tree_node));
        pNew->left=NULL;
        pNew->right=NULL;
        pNew->data=dataIn;
        pLoc=tree->head;
        while(1)
        {
        if (pNew->data>pLoc->data)
        {
                if (pLoc->right!=NULL) pLoc=pLoc->right;
                else
                        {pLoc->right=pNew; tree->count+=1;break;}
        }
        if (pNew->data<pLoc->data)
        {
                if (pLoc->left!=NULL) pLoc=pLoc->left;
                else
                        {pLoc->left=pNew; tree->count+=1;break;}
        }
        if (pNew->data==pLoc->data)
        {
                printf("Data already present....\n");
                break;
        }
        }/*end of while*/

 }/*end of count !=0*/
}
```

```
void RecursivePreorderTraverse(struct tree_node *p)
{
 if (p!=NULL)
 {
        printf("%d\n",p->data);
        RecursivePreorderTraverse(p->left);
        RecursivePreorderTraverse(p->right);
 }
}


void RecursivePostorderTraverse(struct tree_node *p)
{
 if (p!=NULL)
 {
        RecursivePostorderTraverse(p->left);
        RecursivePostorderTraverse(p->right);
        printf("%d\n",p->data);
 }
}

void RecursiveInorderTraverse(struct tree_node *p)
{
 if (p!=NULL)
 {
        RecursiveInorderTraverse(p->left);
        printf("%d\n",p->data);
        RecursiveInorderTraverse(p->right);
 }
}
```

```
void DeleteNode(struct tree_node **root,int dltKey)
{
 if ((*root)==NULL) {printf("DATA NOT FOUND\n");}
 if (dltKey < (*root)->data) DeleteNode(&(*root)->left,dltKey);
 else if (dltKey>(*root)->data) DeleteNode(&(*root)->right,dltKey);
 else if ((*root)->left==NULL)
        {       struct tree_node *delPtr;
                delPtr=(*root);
                (*root)=(*root)->right;
                free(delPtr);
                tree->count-=1;
        }
 else if ((*root)->right==NULL)
        {
                struct tree_node *delPtr;
                delPtr=(*root);
                (*root)=(*root)->left;
                free(delPtr);
                tree->count-=1;
        }
 else  {
                struct tree_node *delPtr;
                delPtr=(*root)->left;
                while(delPtr->right!=NULL)
                        delPtr=delPtr->right;
                (*root)->data=delPtr->data;
                DeleteNode(&(*root)->left,delPtr->data);
        }
}
```

```
void main( )
{
int choice,delKey,dataIn;
createTree();
do
{
        printf("1.Add Node\n");
        printf("2. Preorder Traversal with Recursion\n");
        printf("3. Inorder Traversal with Recursion\n");
        printf("4. Postorder Traversal with Recursion\n");
        printf("5. Delete Node\n");
        printf("6.Exit\n");
        printf("Feed in your choice: ");
        scanf("%d",&choice);
        switch(choice)
        {
         case 1:        printf("Feed in the data: ");
                        scanf("%d",&dataIn);
                        insertNode(dataIn);
                        break;
         case 2:        if (tree->count!=0) RecursivePreorderTraverse(tree->head);
                        break;
         case 3:        if (tree->count!=0)RecursiveInorderTraverse(tree->head);
                        break;
         case 4:        if (tree->count!=0)RecursivePostorderTraverse(tree->head);
                        break;
         case 5:       printf("Feed in data to delete: ");
                        scanf("%d",&delKey);
                        DeleteNode(&(tree->head),delKey);
                        break;
        }

}while(choice !=6 );
}
```