```c
#include<stdio.h>
#include<malloc.h>

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

struct node* createNode(int data){
    struct node *n; // creating a node pointer
    n = (struct node *) malloc(sizeof(struct node)); // Allocating
    n->data = data; // Setting the data
    n->left = NULL; // Setting the left and right children to NULL
    n->right = NULL; // Setting the left and right children to NUL
    return n; // Finally returning the created node
}

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);
    }
}

void inOrder(struct  node* root){
    if(root!=NULL){
        inOrder(root->left);
        printf("%d ", root->data);
        inOrder(root->right);
```

```
40:     }
41: }
42:
43: int isBST(struct  node* root){
44:     static struct node *prev = NULL;
45:     if(root!=NULL){
46:         if(!isBST(root->left)){
47:             return 0;
48:         }
49:         if(prev!=NULL && root->data <= prev->data){
50:             return 0;
51:         }
52:         prev = root;
53:         return isBST(root->right);
54:     }
55:     else{
56:         return 1;
57:     }
58: }
59:
60: struct node * searchIter(struct node* root, int key){
61:     while(root!=NULL){
62:         if(key == root->data){
63:             return root;
64:         }
65:         else if(key<root->data){
66:             root = root->left;
67:         }
68:         else{
69:             root = root->right;
70:         }
71:     }
72:     return NULL;
73: }
74:
75: void insert(struct node *root, int key){
76:     struct node *prev = NULL;
77:     while(root!=NULL){
78:         prev = root;
```

```
 79:         if(key==root->data){
 80:             printf("Cannot insert %d, already in BST", key);
 81:             return;
 82:         }
 83:         else if(key<root->data){
 84:             root = root->left;
 85:         }
 86:         else{
 87:             root = root->right;
 88:         }
 89:     }
 90:     struct node* new = createNode(key);
 91:     if(key<prev->data){
 92:         prev->left = new;
 93:     }
 94:     else{
 95:         prev->right = new;
 96:     }
 97: }
 98:
 99: struct node *inOrderPredecessor(struct node* root){
100:     root = root->left;
101:     while (root->right!=NULL)
102:     {
103:         root = root->right;
104:     }
105:     return root;
106: }
107:
108: struct node *deleteNode(struct node *root, int value){
109:
110:     struct node* iPre;
111:     if (root == NULL){
112:         return NULL;
113:     }
114:     if (root->left==NULL&&root->right==NULL){
115:         free(root);
116:         return NULL;
117:     }
```

```
118:
119:     //searching for the node to be deleted
120:     if (value < root->data){
121:         root-> left = deleteNode(root->left,value);
122:     }
123:     else if (value > root->data){
124:         root-> right = deleteNode(root->right,value);
125:     }
126:     //deletion strategy when the node is found
127:     else{
128:         iPre = inOrderPredecessor(root);
129:         root->data = iPre->data;
130:         root->left = deleteNode(root->left, iPre->data);
131:     }
132:     return root;
133: }
134:
135: int main(){
136:
137:     // Constructing the root node - Using Function (Recommended
138:     struct node *p = createNode(5);
139:     struct node *p1 = createNode(3);
140:     struct node *p2 = createNode(6);
141:     struct node *p3 = createNode(1);
142:     struct node *p4 = createNode(4);
143:     // Finally The tree looks like this:
144:     //       5
145:     //      / \
146:     //     3   6
147:     //    / \
148:     //   1   4
149:
150:     // Linking the root node with left and right children
151:     p->left = p1;
152:     p->right = p2;
153:     p1->left = p3;
154:     p1->right = p4;
155:
156:     inOrder(p);
```

```
157:     printf("\n");
158:     deleteNode(p, 3);
159:     printf("/Data is %d/",p->data);
160:     inOrder(p);
161:
162:     return 0;
163: }
164:
```