```
1: #include<stdio.h>
 2: #include<malloc.h>
 3:
 4: struct node{
 5:
        int data;
        struct node* left;
 6:
        struct node* right;
 7:
 8: };
 9:
10: struct node* createNode(int data){
        struct node *n; // creating a node pointer
11:
12:
        n = (struct node *) malloc(sizeof(struct node)); // Allocating
        n->data = data; // Setting the data
13:
        n->left = NULL; // Setting the left and right children to NULL
14:
        n->right = NULL; // Setting the Left and right children to NUL
15:
        return n; // Finally returning the created node
16:
17: }
18:
19: void preOrder(struct node* root){
        if(root!=NULL){
20:
            printf("%d ", root->data);
21:
            preOrder(root->left);
22:
            preOrder(root->right);
23:
        }
24:
25: }
26:
27: void postOrder(struct node* root){
28:
        if(root!=NULL){
29:
            postOrder(root->left);
30:
            postOrder(root->right);
            printf("%d ", root->data);
31:
        }
32:
33: }
34:
35: void inOrder(struct
                         node* root){
36:
        if(root!=NULL){
37:
            inOrder(root->left);
38:
            printf("%d ", root->data);
            inOrder(root->right);
39:
```

```
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:
            if(prev!=NULL && root->data <= prev->data){
49:
50:
                 return 0;
51:
            }
52:
            prev = root;
53:
            return isBST(root->right);
54:
        else{
55:
56:
            return 1;
        }
57:
58: }
59:
60: struct node * searchIter(struct node* root, int key){
61:
        while(root!=NULL){
            if(key == root->data){
62:
63:
                 return root;
64:
            else if(key<root->data){
65:
66:
                 root = root->left;
67:
            }
            else{
68:
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){
                printf("Cannot insert %d, already in BST", key);
 80:
 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);
        if(key<prev->data){
 91:
            prev->left = new;
 92:
 93:
        }
        else{
 94:
 95:
            prev->right = new;
        }
 96:
 97:
 98: }
 99:
100: int main(){
101:
         // Constructing the root node - Using Function (Recommended
102:
103:
         struct node *p = createNode(5);
         struct node *p1 = createNode(3);
104:
         struct node *p2 = createNode(6);
105:
         struct node *p3 = createNode(1);
106:
         struct node *p4 = createNode(4);
107:
108:
         // Finally The tree looks like this:
109:
         //
                  5
110:
         //
         //
111:
               3 6
112:
         // /\
113:
         // 1 4
114:
115:
         // Linking the root node with left and right children
116:
         p\rightarrowleft = p1;
117:
         p-right = p2;
```

```
118:    p1->left = p3;
119:    p1->right = p4;
120:
121:    insert(p, 16);
122:    printf("%d\n", p->right->right->data);
123:    inOrder(p);
124:    return 0;
125: }
126:
```