```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 NULL
    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: int main(){
61:
62:     // Constructing the root node - Using Function (Recommended)
63:     struct node *p = createNode(5);
64:     struct node *p1 = createNode(3);
65:     struct node *p2 = createNode(6);
66:     struct node *p3 = createNode(1);
67:     struct node *p4 = createNode(4);
68:     // Finally The tree looks like this:
69:     //       5
70:     //      / \
71:     //     3   6
72:     //    / \
73:     //   1   4
74:
75:     // Linking the root node with left and right children
76:     p->left = p1;
77:     p->right = p2;
78:     p1->left = p3;
```

```
79:     p1->right = p4;
80:
81:     // preOrder(p);
82:     // printf("\n");
83:     // postOrder(p);
84:     // printf("\n");
85:     inOrder(p);
86:     printf("\n");
87:     // printf("%d", isBST(p));
88:     if(isBST(p)){
89:         printf("This is a bst" );
90:     }
91:     else{
92:         printf("This is not a bst");
93:     }
94:     return 0;
95: }
96:
```