Experiment No.8:

Height and Depth in BST
Develop a program that constructs a Binary Search Tree and computes the height of the tree and the depth of a given node.

```
    1
   / \
  2   3
 / \   \
4   5   6
   /
  7
```

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

// Define node structure
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Create new node
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Insert nodes manually for demo
struct Node* createSampleTree() {
    struct Node* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);
    root->right->right = createNode(6);
    root->left->right->left = createNode(7);
    return root;
}

// Count leaf nodes
int countLeaf(struct Node* root) {
    if (root == NULL)
        return 0;
    if (root->left == NULL && root->right == NULL)
```

```
        return 1;
    return countLeaf(root->left) + countLeaf(root->right);
}

// Count internal nodes (nodes with at least one child)
int countInternal(struct Node* root) {
    if (root == NULL || (root->left == NULL && root->right == NULL))
        return 0;
    return 1 + countInternal(root->left) + countInternal(root->right);
}

// Count nodes with only one child
int countOneChild(struct Node* root) {
    if (root == NULL)
        return 0;
    int count = 0;
    if ((root->left == NULL && root->right != NULL) ||
        (root->left != NULL && root->right == NULL))
        count = 1;

    return count + countOneChild(root->left) + countOneChild(root->right);
}

int main() {
    struct Node* root = createSampleTree();

    printf("Counting nodes in the binary tree...\n");

    printf("Leaf Nodes: %d\n", countLeaf(root));
    printf("Internal Nodes: %d\n", countInternal(root));
    printf("Nodes with only one child: %d\n", countOneChild(root));

    return 0;
}
```