csc 326 lab 9

```
#include <iostream>
using namespace std;
class BST {
 int data;
 BST *left, *right;
public:
 BST();
 BST(int);
 BST* Insert(BST*, int);
 void Inorder(BST*);
   void Preorder(BST*);
  void Postorder(BST*);
    void printLevelOrder(BST* root);
    void printCurrentLevel(BST* root, int level);
    int height(BST* node);
    void mirror(BST* node);
    BST* search(BST* root, int key);
    void swapSubtrees(BST* root, int key);
};
BST ::BST()
 : data(0)
 , left(NULL)
 , right(NULL)
```

```
{
}
BST ::BST(int value)
{
 data = value;
 left = right = NULL;
}
BST* BST ::Insert(BST* root, int value)
{
 if (!root) {

  return new BST(value);
 }
 if (value > root->data) {
  root->right = Insert(root->right, value);
 }
 else {
  root->left = Insert(root->left, value);
 }
/ Return 'root' node, after insertion.
 return root;
}
// Inorder traversal function.
// This gives data in sorted order.
```

```
void BST ::Inorder(BST* root)
{
 if (!root) {
  return;
 }
 Inorder(root->left);
 cout << root->data << endl;
 Inorder(root->right);
}
// Preorder traversal function.
void BST ::Preorder(BST* root)
{
 if (!root) {
  return;
 }
 cout << root->data << endl;
 Inorder(root->left);
 Inorder(root->right);
}
// Postorder traversal function.
void BST ::Postorder(BST* root)
{
 if (!root) {
  return;
```

```
 }
 Inorder(root->left);
 Inorder(root->right);
    cout << root->data << endl;
}
int BST ::height(BST* node)
{
    if (node == NULL)
        return 0;
    else {
        int lheight = height(node->left);
        int rheight = height(node->right);

        if (lheight > rheight) {
            return (lheight + 1);
        }
        else {
            return (rheight + 1);
        }
    }
}
void BST ::printLevelOrder(BST* root)
{
    int h = height(root);
```

```
    int i;
    for (i = 1; i <= h; i++)
        printCurrentLevel(root, i);
}
 void BST ::printCurrentLevel(BST* root, int level)
{
    if (root == NULL)
        return;
    if (level == 1)
        cout << root->data << " ";
    else if (level > 1) {
        printCurrentLevel(root->left, level - 1);
        printCurrentLevel(root->right, level - 1);
    }
}
void BST ::mirror(BST* node)
{
    if (node == NULL)
        return;
    else
    {
        BST* temp;

            mirror(node->left);
```

```
        mirror(node->right);


        temp     = node->left;
        node->left = node->right;
        node->right = temp;
    }
}
BST* BST:: search(BST* root, int key)
{
    if (root == NULL || root->data == key)
       return root;


    if (root->data < key)
       return search(root->right, key);


    return search(root->left, key);
}
void BST:: swapSubtrees(BST* root, int key)
{
    BST* node = search(root, key);
    if(node == NULL)
        return;
mirror(node);
}
```

```
// Driver code
int main()
{
 BST b, *root = NULL;
 root = b.Insert(root, 50);
 b.Insert(root, 30);
 b.Insert(root, 20);
 b.Insert(root, 40);
 b.Insert(root, 70);
 b.Insert(root, 60);
 b.Insert(root, 80);
    cout << " Order of level traversal before swapping: ";
 b.printLevelOrder(root);
    b.swapSubtrees(root, 50);
    cout << "\ Order of level  traversal after swapping: ";
    b.printLevelOrder(root);
 return 0;
}
```

OUTPUT: