

```

1: #include <stdio.h>
2: #include <stdlib.h>
3:
4: struct Node
5: {
6:     int key;
7:     struct Node *left;
8:     struct Node *right;
9:     int height;
10: };
11:
12: int getHeight(struct Node *n){
13:     if(n==NULL)
14:         return 0;
15:     return n->height;
16: }
17:
18: struct Node *createNode(int key){
19:     struct Node* node = (struct Node *) malloc(sizeof(struct Node));
20:     node->key = key;
21:     node->left = NULL;
22:     node->right = NULL;
23:     node->height = 1;
24:     return node;
25: }
26:
27: int max (int a, int b){
28:     return (a>b)?a:b;
29: }
30:
31: int getBalanceFactor(struct Node * n){
32:     if(n==NULL){
33:         return 0;
34:     }
35:     return getHeight(n->left) - getHeight(n->right);
36: }
37:
38: struct Node* rightRotate(struct Node* y){
39:     struct Node* x = y->left;

```

```

40:     struct Node* T2 = x->right;
41:
42:     x->right = y;
43:     y->left = T2;
44:
45:     x->height = max(getHeight(x->right), getHeight(x->left)) + 1;
46:     y->height = max(getHeight(y->right), getHeight(y->left)) + 1;
47:
48:     return x;
49: }
50:
51: struct Node* leftRotate(struct Node* x){
52:     struct Node* y = x->right;
53:     struct Node* T2 = y->left;
54:
55:     y->left = x;
56:     x->right = T2;
57:
58:     x->height = max(getHeight(x->right), getHeight(x->left)) + 1;
59:     y->height = max(getHeight(y->right), getHeight(y->left)) + 1;
60:
61:     return y;
62: }
63:
64: struct Node *insert(struct Node* node, int key){
65:     if (node == NULL)
66:         return createNode(key);
67:
68:     if (key < node->key)
69:         node->left = insert(node->left, key);
70:     else if (key > node->key)
71:         node->right = insert(node->right, key);
72:
73:     node->height = 1 + max(getHeight(node->left), getHeight(node->right));
74:     int bf = getBalanceFactor(node);
75:
76:     // Left Left Case
77:     if(bf>1 && key < node->left->key){
78:         return rightRotate(node);

```

```

79:         }
80:         // Right Right Case
81:         if(bf<-1 && key > node->right->key){
82:             return leftRotate(node);
83:         }
84:         // Left Right Case
85:         if(bf>1 && key > node->left->key){
86:             node->left = leftRotate(node->left);
87:             return rightRotate(node);
88:         }
89:         // Right Left Case
90:         if(bf<-1 && key < node->right->key){
91:             node->right = rightRotate(node->right);
92:             return leftRotate(node);
93:         }
94:         return node;
95:     }
96:
97: void preOrder(struct Node *root)
98: {
99:     if(root != NULL)
100:     {
101:         printf("%d ", root->key);
102:         preOrder(root->left);
103:         preOrder(root->right);
104:     }
105: }
106:
107: int main(){
108:     struct Node * root = NULL;
109:
110:
111:     root = insert(root, 1);
112:     root = insert(root, 2);
113:     root = insert(root, 4);
114:     root = insert(root, 5);
115:     root = insert(root, 6);
116:     root = insert(root, 3);
117:     preOrder(root);

```

```
118:    return 0;  
119: }  
120:
```