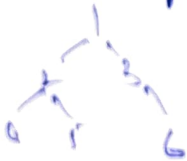
```
// Write a C program to implement the Tree Traversals
// (Inorder, Preorder, Postorder)
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node* left;
    struct node* right;
};
struct node* createNode(int data) {
    struct node* newNode = (struct node*) malloc(sizeof(struct node));
newNode->data = data;
newNode->left = NULL;
newNode->right = NULL;
return newNode;
}
void inorderTraversal(struct node* root) {
  if (root == NULL)
    return;
inorderTraversal(root->left);
printf("%d", root->data);
inorderTraversal(root->right); }
void preorderTraversal(struct Node* root) {
    if (root == NULL)
      return;
Print("%d", root->data);
preorderTraversal(root->left);
preorderTraversal(root->right); }
void postordertraversal(struct Node* root) {
```

```
if (root == NULL)
    return;
inorderTraversal(root->left);
preorderTraversal(root->right);
printf("%d", root->data);
}

int main() {
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);
printf("Inorder Traversal:");
inOrderTraversal(root);
printf("\n");
printf("Preorder Traversal:");
preorderTraversal(root);
printf("\n");
printf("PostOrder Traversal:");
PostOrderTraversal(root);
printf("\n");
return 0; }
```

Input: Creating the tree.



Output:

Inorder : 4 2 5 1 3 6
Preorder : 1 2 4 5 3 6
Postorder : 4 5 2 6 3 1

Construct AVL tree for the following elements 3,2,1,4,5,6,7 followed by 10 to 16 in reverse order.

To construct an AVL tree for the given elements:

Elements to Insert

- First Sequence: 3, 2, 1, 4, 5, 6, 7
- Second Sequence (reverse order): 16, 15, 14, 13, 12, 11, 10

Steps to Construct the AVL Tree:

1) Insert 3:

2) Insert 2:

* Balance factor for node 3 is 1, so no rotation needed.

3) Insert 1

* Balance factor for node 3 is 2, and node 2 is 1, so we need a right rotation at node 3.

* After rotation, the tree becomes:

4) Insert 4:

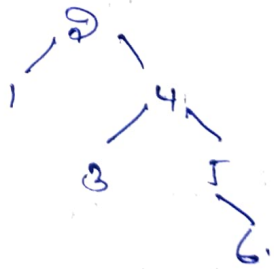
* Balance factor for node 2 is 0, so no rotation needed

5) Insert 5.

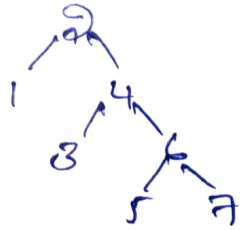
* Balancing factor for node 3 is -2, and node 4 is -1, so we needed a Left rotation at node 3.

* After rotation

Insert 6:-

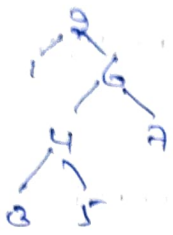


Insert 7:-

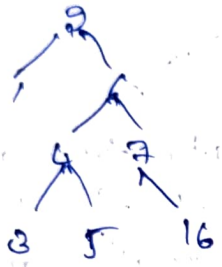


*Balance factor for node 4 is -2 and node 6 is -1, so we need left rotation at node 4.

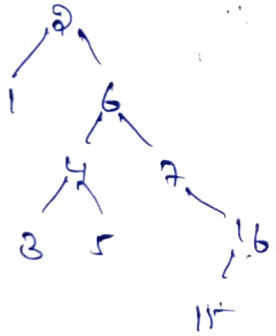
After rotation:-



8.) Insert 16.

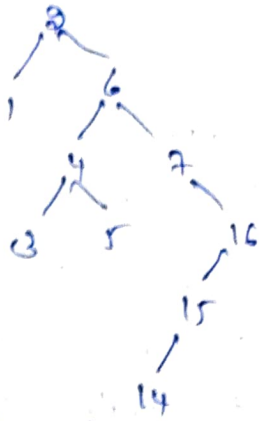


9.) Insert 15



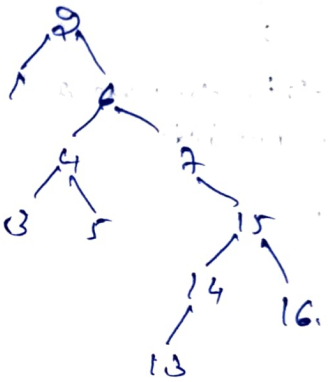
*Balance factor for node 16 is 1, so no rotation needed 16.

Insert 14:-

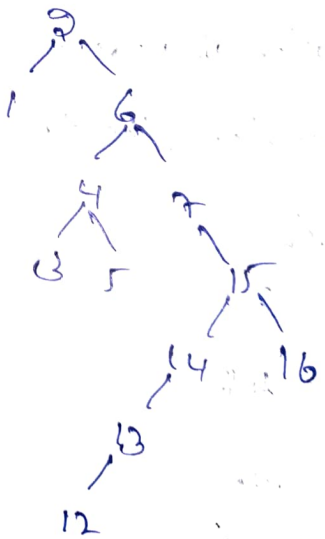


* Balance factor for node 16 is 2, node 15 is 1, so, we need a right rotation at node 15.

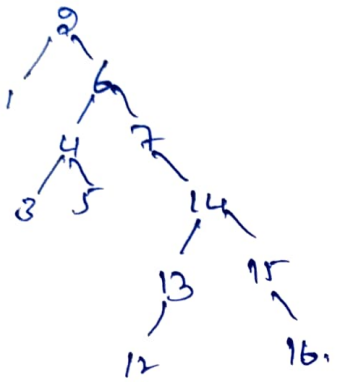
11.) Insert 13:-



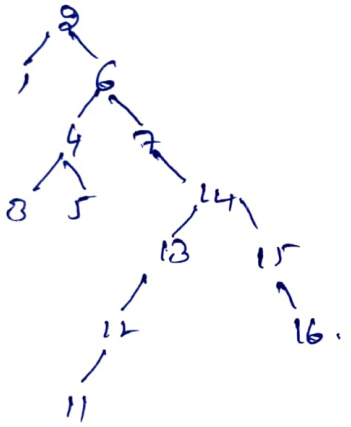
12.) Insert 12:-



balance factor for node 15 is 2, 14 is 1, So, we needs a rotation at node 14.

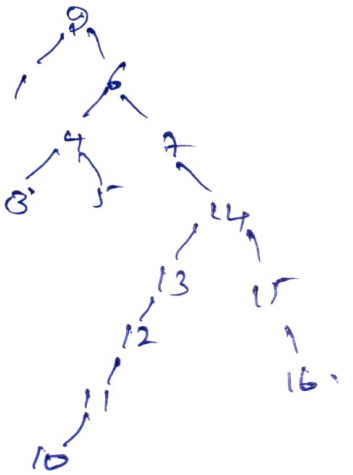


13) Insert 11

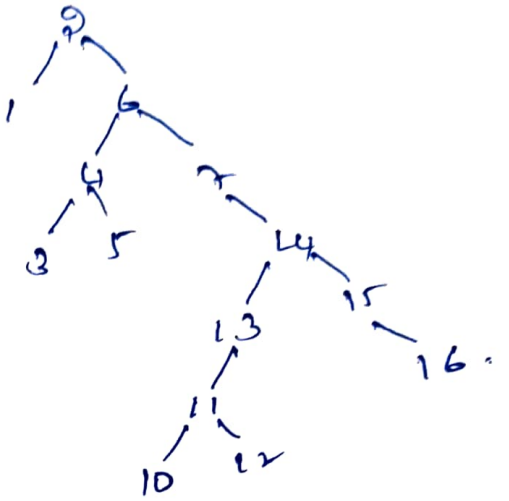


*Balance factor for node 14 is 1 So, no rotation needed.

14 Insert 10:



* After rotation, the final tree.



This AVL Tree is now balance with given sequence of insertion.