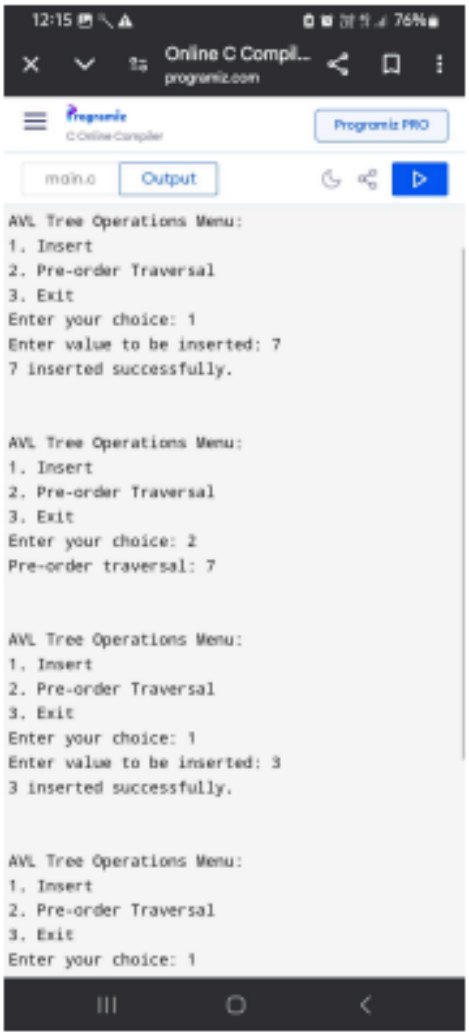
IMPLEMENTATION OF AVL TREE

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

// AVL Tree Node
struct Node
{
    int key;
    struct Node *left;
    struct Node *right;
    int height;
};

// Function to get the height of the tree
int height(struct Node *N)
{
    if (N == NULL)
        return 0;
    return N->height;
}

// Function to get the maximum of two integers
int max(int a, int b)
{
    return (a > b) ? a : b;
}

// Create a new node
struct Node* newNode(int key)
{
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->key = key;
    node->left = NULL;
    node->right = NULL;
    node->height = 1; // New node is initially added at leaf
    return(node);
}

// Right rotate subtree rooted with y
struct Node *rightRotate(struct Node *y)
{
    struct Node *x = y->left;
    struct Node *T2 = x->right;
```

```
    // Perform rotation
    x->right = y;
    y->left = T2;

    // Update heights
    y->height = max(height(y->left), height(y->right)) + 1;
    x->height = max(height(x->left), height(x->right)) + 1;

    // Return new root
    return x;
}

// Left rotate subtree rooted with x
struct Node *leftRotate(struct Node *x)
{
    struct Node *y = x->right;
    struct Node *T2 = y->left;

    // Perform rotation
    y->left = x;
    x->right = T2;

    // Update heights
    x->height = max(height(x->left), height(x->right)) + 1;
    y->height = max(height(y->left), height(y->right)) + 1;

    // Return new root
    return y;
}

// Get Balance factor of node N
int getBalance(struct Node *N)
{
    if (N == NULL)
        return 0;
    return height(N->left) - height(N->right);
}

// Recursive function to insert a key in the subtree rooted with node
struct Node* insert(struct Node* node, int key)
{
    // 1. Perform the normal BST rotation
    if (node == NULL)
        return(newNode(key));

    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
```

```
        node->right = insert(node->right, key);
    else // Equal keys are not allowed in BST
        return node;

    // 2. Update height of this ancestor node
    node->height = 1 + max(height(node->left), height(node->right));

    // 3. Get the balance factor of this ancestor node
    int balance = getBalance(node);

    // If this node becomes unbalanced, then there are 4 cases

    // Left Left Case
    if (balance > 1 && key < node->left->key)
        return rightRotate(node);

    // Right Right Case
    if (balance < -1 && key > node->right->key)
        return leftRotate(node);

    // Left Right Case
    if (balance > 1 && key > node->left->key) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }

    // Right Left Case
    if (balance < -1 && key < node->right->key) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

    // Return the (unchanged) node pointer
    return node;
}

// A utility function to print the preorder traversal of the tree
void preOrder(struct Node *root)
{
    if (root != NULL)
    {
        printf("%d ", root->key);
        preOrder(root->left);
        preOrder(root->right);
    }
}

// Menu-driven AVL Tree implementation
```

```
int main()
{
    struct Node *root = NULL;
    int choice, key;

    while(1)
    {
        printf("\n\nAVL Tree Operations Menu:\n");
        printf("1. Insert\n");
        printf("2. Pre-order Traversal\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice)
        {
            case 1:
                printf("Enter value to be inserted: ");
                scanf("%d", &key);
                root = insert(root, key);
                printf("%d inserted successfully.\n", key);
                break;

            case 2:
                printf("Pre-order traversal: ");
                preOrder(root);
                printf("\n");
                break;

            case 3:
                printf("Exiting...\n");
                exit(0);

            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
```


12:15
76%
Online C Compil...
programiz.com
Programiz
C Online Compiler
Programiz PRO
main.c
Output
AVL Tree Operations Menu:
1. Insert
2. Pre-order Traversal
3. Exit
Enter your choice: 1
Enter value to be inserted: 7
7 inserted successfully.
AVL Tree Operations Menu:
1. Insert
2. Pre-order Traversal
3. Exit
Enter your choice: 2
Pre-order traversal: 7
AVL Tree Operations Menu:
1. Insert
2. Pre-order Traversal
3. Exit
Enter your choice: 1
Enter value to be inserted: 3
3 inserted successfully.
AVL Tree Operations Menu:
1. Insert
2. Pre-order Traversal
3. Exit
Enter your choice: 1