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

// Define the structure for an AVL tree node
typedef struct AVLNode {
    int key;
    struct AVLNode *left;
    struct AVLNode *right;
    int height;
} AVLNode;

// Function to get the height of the node
int height(AVLNode *node) {
    if (node == NULL)
        return 0;
    return node->height;
}

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

// Function to create a new node
AVLNode* createNode(int key) {
    AVLNode *node = (AVLNode *)malloc(sizeof(AVLNode));
    node->key = key;
    node->left = NULL;
    node->right = NULL;
    node->height = 1; // New node is initially at height 1
    return node;
}

// Right rotate subtree rooted with y
AVLNode* rightRotate(AVLNode *y) {
    AVLNode *x = y->left;
    AVLNode *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
AVLNode* leftRotate(AVLNode *x) {
   AVLNode *y = x->right;
   AVLNode *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(AVLNode *node) {
   if (node == NULL)
      return 0;
   return height(node->left) - height(node->right);
}

// Recursive function to insert a key in the subtree rooted with node and returns the new root
of the subtree
AVLNode* insert(AVLNode* node, int key) {
   // Perform the normal BST insertion
   if (node == NULL)
      return createNode(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;

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

   // Get the balance factor of this ancestor node to check whether this node became
unbalanced
   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 tree in order
void inOrder(AVLNode *root) {
    if (root != NULL) {
        inOrder(root->left);
        printf("%d ", root->key);
        inOrder(root->right);
    }
}

int main() {
    AVLNode *root = NULL;

    // Insert nodes
    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 30);
    root = insert(root, 40);
    root = insert(root, 50);
    root = insert(root, 25);

    // Print in-order traversal of the AVL tree
    printf("In-order traversal of the AVL tree is: ");
    inOrder(root);
```

```c
    printf("\n");

    return 0;
}
```