[*] Untitled1

```c
#include <stdio.h>
#include <stdlib.h>
#define RED 0
#define BLACK 1
typedef struct Node {
    int data;
    int color;
    struct Node *left, *right, *parent;
} Node;
Node* createNode(int data);
Node* rotateLeft(Node *root);
Node* rotateRight(Node *root);
void fixViolations(Node **root, Node *pt);
void inOrderTraversal(Node *root);
void printTree(Node *root, int space);
Node* insert(Node *root, int data);
Node* createNode(int data) {
    Node *newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = newNode->right = newNode->parent = NULL;
    newNode->color = RED;
    return newNode
}
void inOrderTraversal(Node *root) {
    if (root == NULL) return;
    inOrderTraversal(root->left);
    printf("%d ", root->data);
    inOrderTraversal(root->right);
}
void printTree(Node *root, int space) {
    if (root == NULL) return;
    space += 10;
    printTree(root->right, space);
    printf("\n");
    for (int i = 10; i < space; i++) printf(" ");
    printf("%d(%s)\n", root->data, root->color == RED ? "R" : "B");
```

[*] Untitled1

```
    printTree(root->left, space);
}
Node* rotateLeft(Node *root) {
    Node *newRoot = root->right;
    root->right = newRoot->left;
    if (newRoot->left != NULL) newRoot->left->parent = root;
    newRoot->parent = root->parent;
    if (root->parent == NULL) {
    } else if (root == root->parent->left) {
        root->parent->left = newRoot;
    } else {
        root->parent->right = newRoot;
    }
    newRoot->left = root;
    root->parent = newRoot;
    return newRoot;
}
Node* rotateRight(Node *root) {
    Node *newRoot = root->left;
    root->left = newRoot->right;
    if (newRoot->right != NULL) newRoot->right->parent = root;
    newRoot->parent = root->parent;
    if (root->parent == NULL) {
    } else if (root == root->parent->right) {
        root->parent->right = newRoot;
    } else {
        root->parent->left = newRoot;
    }
    newRoot->right = root;
    root->parent = newRoot;
    return newRoot;
}
void fixViolations(Node **root, Node *pt) {
    Node *parent = NULL;
    Node *grandparent = NULL;
    while (pt != *root && pt->parent->color == RED) {
```

[*] Untitled1

```
    parent = pt->parent;
    grandparent = parent->parent;
    if (parent == grandparent->left) {
        Node *uncle = grandparent->right;
        if (uncle != NULL && uncle->color == RED) {
            parent->color = BLACK;
            uncle->color = BLACK;
            grandparent->color = RED;
            pt = grandparent;
        } else
            if (pt == parent->right) {
                pt = parent;
                *root = rotateLeft(*root);
            }
            parent->color = BLACK;
            grandparent->color = RED;
            *root = rotateRight(*root);
        
    } else {
        Node *uncle = grandparent->left;
        if (uncle != NULL && uncle->color == RED) {
            parent->color = BLACK;
            uncle->color = BLACK;
            grandparent->color = RED;
            pt = grandparent;
        } else {
            if (pt == parent->left) {
                pt = parent;
                *root = rotateRight(*root);
            }
            parent->color = BLACK;
            grandparent->color = RED;
            *root = rotateLeft(*root);
        }
    }
}
```

[*] Untitled1

```
    Node *newNode = createNode(data);
    if (root == NULL) {
        return newNode;
    }
    Node *parent = NULL;
    Node *current = root;
    while (current != NULL) {
        parent = current;
        if (data < current->data) {
            current = current->left;
        } else {
            current = current->right;
        }
    }
    newNode->parent = parent;
    if (data < parent->data) {
        parent->left = newNode;
    } else {
        parent->right = newNode;
    }
    fixViolations(&root, newNode);
    return root;
}
int main() {
    Node *root = NULL;
    int values[] = {10, 20, 30, 15, 25, 5, 1};

    for (int i = 0; i < sizeof(values) / sizeof(values[0]); i++) {
        root = insert(root, values[i]);
    }
    printf("In-order traversal of the Red-Black Tree:\n");
    inOrderTraversal(root);
    printf("\n");
    printf("Tree structure:\n");
    printTree(root, 0);
    return 0;
```