```c
#include <stdio.h>
#include <stdlib.h>
#define RED 1
#define BLACK 0
typedef struct RBTreeNode {
    int data;
    int color;
    struct RBTreeNode *left, *right, *parent;
} RBTreeNode;
RBTreeNode* createNode(int data);
RBTreeNode* leftRotate(RBTreeNode *root, RBTreeNode *x);
RBTreeNode* rightRotate(RBTreeNode *root, RBTreeNode *y);
void fixInsert(RBTreeNode **root, RBTreeNode *pt);
void inorderTraversal(RBTreeNode *root);
void insert(RBTreeNode **root, int data);
RBTreeNode* createNode(int data) {
    RBTreeNode *newNode = (RBTreeNode *)malloc(sizeof(RBTreeNode));
    newNode->data = data;
    newNode->left = newNode->right = newNode->parent = NULL;
    newNode->color = RED;
    return newNode;
}
RBTreeNode* leftRotate(RBTreeNode *root, RBTreeNode *x) {
    RBTreeNode *y = x->right;
    x->right = y->left;
    if (y->left != NULL) y->left->parent = x;
    y->parent = x->parent;
    if (x->parent == NULL) root = y;
    else if (x == x->parent->left) x->parent->left = y;
    else x->parent->right = y;
    y->left = x;
    x->parent = y;
    return root;
}
RBTreeNode* rightRotate(RBTreeNode *root, RBTreeNode *y) {
    RBTreeNode *x = y->left;
```

```
RBTreeNode* rightRotate(RBTreeNode *root, RBTreeNode *y) {
    RBTreeNode *x = y->left;
    y->left = x->right;
    if (x->right != NULL) x->right->parent = y;
    x->parent = y->parent;
    if (y->parent == NULL) root = x;
    else if (y == y->parent->left) y->parent->left = x;
    else y->parent->right = x;
    x->right = y;
    y->parent = x;
    return root;
}
void fixInsert(RBTreeNode **root, RBTreeNode *pt) {
    RBTreeNode *parent = NULL;
    RBTreeNode *grandparent = NULL;
    while (pt != *root && pt->parent->color == RED) {
        parent = pt->parent;
        grandparent = parent->parent;
        if (parent == grandparent->left) {
            RBTreeNode *uncle = grandparent->right;

            if (uncle && uncle->color == RED) {
                parent->color = BLACK;
                uncle->color = BLACK;
                grandparent->color = RED;
                pt = grandparent;
            } else {
                if (pt == parent->right) {
                    pt = parent;
                    *root = leftRotate(*root, pt);
                    parent = pt->parent;
                }
                parent->color = BLACK;
                grandparent->color = RED;
                *root = rightRotate(*root, grandparent);
```

```
            if (uncle && uncle->color == RED) {
                parent->color = BLACK;
                uncle->color = BLACK;
                grandparent->color = RED;
                pt = grandparent;
            } else {
                if (pt == parent->left) {
                    pt = parent;
                    *root = rightRotate(*root, pt);
                    parent = pt->parent;
                }
                parent->color = BLACK;
                grandparent->color = RED;
                *root = leftRotate(*root, grandparent);
            }
        }
    }
    (*root)->color = BLACK;
}
void insert(RBTreeNode **root, int data) {
    RBTreeNode *newNode = createNode(data);
    RBTreeNode *y = NULL;
    RBTreeNode *x = *root;

    while (x != NULL) {
        y = x;
        if (newNode->data < x->data) x = x->left;
        else x = x->right;
    }
    newNode->parent = y;
    if (y == NULL) *root = newNode;
    else if (newNode->data < y->data) y->left = newNode;
    else y->right = newNode;

    fixInsert(root, newNode);
```

```
    while (x != NULL) {
        y = x;
        if (newNode->data < x->data) x = x->left;
        else x = x->right;
    }
    newNode->parent = y;
    if (y == NULL) *root = newNode;
    else if (newNode->data < y->data) y->left = newNode;
    else y->right = newNode;

    fixInsert(root, newNode);
}
void inorderTraversal(RBTreeNode *root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d (%s) ", root->data, root->color == RED ? "RED" : "BLACK");
        inorderTraversal(root->right);
    }
}
int main() {
    RBTreeNode *root = NULL;

    insert(&root, 10);
    insert(&root, 20);
    insert(&root, 30);
    insert(&root, 15);
    insert(&root, 25);
    insert(&root, 5);
    insert(&root, 1);
    printf("Inorder Traversal of the Red-Black Tree:\n");
    inorderTraversal(root);
    return 0;
}
```