1. Write a c program for RED BLACK Tree

```c
#include <stdio.h>
#include <stdlib.h>
enum Color {RED, BLACK};
struct Node {
    int data;
    enum Color color;
    struct Node *left, *right, *parent;
};
struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->color = RED;
    node->left = node->right = node->parent = NULL;
    return node;
}
void leftRotate(struct Node** root, struct Node* x) {
    struct Node* 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;
}
void rightRotate(struct Node** root, struct Node* y) {
    struct Node* x = y->left;
    y->left = x->right;
```

```
/tmp/6HZqqiU4kd.o
20 10 15 30 25

=== Code Execution Successful ===
```

```c
    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->right)
        y->parent->right = x;
    else
        y->parent->left = x;

    x->right = y;
    y->parent = x;
}
void fixInsert(struct Node** root, struct Node* k) {
    struct Node* parent = NULL;
    struct Node* grandparent = NULL;
    while ((k != *root) && (k->color != BLACK) && (k->parent->color == RED)) {
        parent = k->parent;
        grandparent = k->parent->parent;
        if (parent == grandparent->left) {
            struct Node* uncle = grandparent->right;
            if (uncle != NULL && uncle->color == RED) {
                grandparent->color = RED;
                parent->color = BLACK;
                uncle->color = BLACK;
                k = grandparent;
            } else {
                if (k == parent->right) {
                    leftRotate(root, parent);
                    k = parent;
                    parent = k->parent;
                }
                rightRotate(root, grandparent);
```

```
/tmp/6HZqqiU4kd.o
20 10 15 30 25

=== Code Execution Successful ===
```

```
                rightRotate(root, grandparent);
                enum Color temp = parent->color;
                parent->color = grandparent->color;
                grandparent->color = temp;
                k = parent;
            }
        } else {
            struct Node* uncle = grandparent->left;
            if (uncle != NULL && uncle->color == RED) {
                grandparent->color = RED;
                parent->color = BLACK;
                uncle->color = BLACK;
                k = grandparent;
            } else {
                if (k == parent->left) {
                    rightRotate(root, parent);
                    k = parent;
                    parent = k->parent;
                }
                leftRotate(root, grandparent);
                enum Color temp = parent->color;
                parent->color = grandparent->color;
                grandparent->color = temp;
                k = parent;
            }
        }
    }
    (*root)->color = BLACK;
}
void insert(struct Node** root, int data) {
    struct Node* node = newNode(data);

    if (*root == NULL) {
        node->color = BLACK;
```

```
/tmp/6HZqqiU4kd.o
20 10 15 30 25

=== Code Execution Successful ===
```

```
        *root = node;
    } else {
        struct Node* current = *root;
        struct Node* parent = NULL;

        while (current != NULL) {
            parent = current;
            if (node->data < current->data)
                current = current->left;
            else
                current = current->right;
        }
        node->parent = parent;
        if (node->data < parent->data)
            parent->left = node;
        else
            parent->right = node;
        fixInsert(root, node);
    }
}
void preOrderTraversal(struct Node* root) {
    if (root == NULL)
        return;
    printf("%d ", root->data);
    preOrderTraversal(root->left);
    preOrderTraversal(root->right);
}
int main() {
    struct Node* root = NULL;
    insert(&root, 10);
    insert(&root, 20);
    insert(&root, 30);
    insert(&root, 15);
    insert(&root, 25);
```

```
/tmp/6HZqqiU4kd.o
20 10 15 30 25

=== Code Execution Successful ===
```

```
int main() {
    struct Node* root = NULL;
    insert(&root, 10);
    insert(&root, 20);
    insert(&root, 30);
    insert(&root, 15);
    insert(&root, 25);
    preOrderTraversal(root);
    return 0;
}
```