NAME: Haresh Kumar N L (192425009)

COURSE NAME: DATA STRUCTURES FOR MODERN COMPUTING SYSTEMS

COURSE CODE: CSA0302

Experiment 26: RED - BLACK Tree

Code:

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

enum Color { RED, BLACK };

struct Node {
    int data;
    enum Color color;
    struct Node *left, *right, *parent;
};

struct Node *root = NULL;

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->color = RED;
    newNode->left = newNode->right = newNode->parent = NULL;
    return newNode;
}

void rotateLeft(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 rotateRight(struct Node **root, struct Node *x) {
    struct Node *y = x->left;
    x->left = y->right;
    if(y->right != NULL)
        y->right->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->right = x;
    x->parent = y;
}

void fixViolation(struct Node **root, struct Node *pt) {
    struct Node *parent_pt = NULL;
struct Node *grand_parent_pt = NULL;

while((pt != *root) && (pt->color != BLACK) && (pt->parent->color == RED)) {
   parent_pt = pt->parent;
   grand_parent_pt = parent_pt->parent;

   if(parent_pt == grand_parent_pt->left) {
      struct Node *uncle_pt = grand_parent_pt->right;

      if(uncle_pt != NULL && uncle_pt->color == RED) {
         grand_parent_pt->color = RED;
         parent_pt->color = BLACK;
         uncle_pt->color = BLACK;
         pt = grand_parent_pt;
      } else {
         if(pt == parent_pt->right) {
            pt = parent_pt;
            rotateLeft(root, pt);
         }
         parent_pt->color = BLACK;
         grand_parent_pt->color = RED;
         rotateRight(root, grand_parent_pt);
      }
   } else {
      struct Node *uncle_pt = grand_parent_pt->left;

      if(uncle_pt != NULL && uncle_pt->color == RED) {
         grand_parent_pt->color = RED;
         parent_pt->color = BLACK;
         uncle_pt->color = BLACK;
         pt = grand_parent_pt;
```

```
            } else {
                if(pt == parent_pt->left) {
                    pt = parent_pt;
                    rotateRight(root, pt);
                }
                parent_pt->color = BLACK;
                grand_parent_pt->color = RED;
                rotateLeft(root, grand_parent_pt);
            }
        }
    }
    (*root)->color = BLACK;
}

void insert(const int data) {
    struct Node *pt = createNode(data);
    struct Node *parent = NULL;
    struct Node *current = root;

    while(current != NULL) {
        parent = current;
        if(pt->data < current->data)
            current = current->left;
        else
            current = current->right;
    }

    pt->parent = parent;
    if(parent == NULL)
        root = pt;
    else if(pt->data < parent->data)
```

```
        parent->left = pt;
    else
        parent->right = pt;

    fixViolation(&root, pt);
}

void inorder(struct Node *root) {
    if(root == NULL)
        return;
    inorder(root->left);
    printf("%d(%s) ", root->data, root->color == RED ? "R" : "B");
    inorder(root->right);
}

int main() {
    int choice, value;
    while(1) {
        printf("\n--- Red-Black Tree Menu ---\n");
        printf("1. Insert\n2. Display (Inorder)\n3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch(choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insert(value);
                break;
            case 2:
                printf("Inorder Traversal: ");
                inorder(root);
            printf("\n");
            break;
          case 3:
            exit(0);
          default:
            printf("Invalid choice\n");
        }
    }
    return 0;
}
```

Output:

```
--- Red-Black Tree Menu ---
1. Insert
2. Display (Inorder)
3. Exit
Enter your choice: 1
Enter value to insert: 10

--- Red-Black Tree Menu ---
1. Insert
2. Display (Inorder)
3. Exit
Enter your choice: 1
Enter value to insert: 20

--- Red-Black Tree Menu ---
1. Insert
2. Display (Inorder)
3. Exit
Enter your choice: 1
Enter value to insert: 30

--- Red-Black Tree Menu ---
1. Insert
2. Display (Inorder)
3. Exit
Enter your choice: 2
Inorder Traversal: 10(R) 20(B) 30(R)
```