

1. Write a c program for RED BLACK Tree

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 enum Color {RED, BLACK};
4 struct Node {
5     int data;
6     enum Color color;
7     struct Node *left, *right, *parent;
8 };
9 struct Node* newNode(int data) {
10     struct Node* node = (struct Node*)malloc(sizeof(struct Node));
11     node->data = data;
12     node->color = RED;
13     node->left = node->right = node->parent = NULL;
14     return node;
15 }
16 void leftRotate(struct Node** root, struct Node* x) {
17     struct Node* y = x->right;
18     x->right = y->left;
19     if (y->left != NULL)
20         y->left->parent = x;
21     y->parent = x->parent;
22     if (x->parent == NULL)
23         *root = y;
24     else if (x == x->parent->left)
25         x->parent->left = y;
26     else
27         x->parent->right = y;
28
29     y->left = x;
30     x->parent = y;
31 }
32 void rightRotate(struct Node** root, struct Node* y) {
33     struct Node* x = y->left;
34     y->left = x->right;
```

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

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

```
34     y->left = x->right;
35     if (x->right != NULL)
36         x->right->parent = y;
37     x->parent = y->parent;
38     if (y->parent == NULL)
39         *root = x;
40     else if (y == y->parent->right)
41         y->parent->right = x;
42     else
43         y->parent->left = x;
44
45     x->right = y;
46     y->parent = x;
47 }
48 void fixInsert(struct Node** root, struct Node* k) {
49     struct Node* parent = NULL;
50     struct Node* grandparent = NULL;
51     while ((k != *root) && (k->color != BLACK) && (k->parent->color == RED)) {
52         parent = k->parent;
53         grandparent = k->parent->parent;
54         if (parent == grandparent->left) {
55             struct Node* uncle = grandparent->right;
56             if (uncle != NULL && uncle->color == RED) {
57                 grandparent->color = RED;
58                 parent->color = BLACK;
59                 uncle->color = BLACK;
60                 k = grandparent;
61             } else {
62                 if (k == parent->right) {
63                     leftRotate(root, parent);
64                     k = parent;
65                     parent = k->parent;
66                 }
67                 rightRotate(root, grandparent);
```

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

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

```

67         rightRotate(root, grandparent);
68         enum Color temp = parent->color;
69         parent->color = grandparent->color;
70         grandparent->color = temp;
71         k = parent;
72     }
73 } else {
74     struct Node* uncle = grandparent->left;
75     if (uncle != NULL && uncle->color == RED) {
76         grandparent->color = RED;
77         parent->color = BLACK;
78         uncle->color = BLACK;
79         k = grandparent;
80     } else {
81         if (k == parent->left) {
82             rightRotate(root, parent);
83             k = parent;
84             parent = k->parent;
85         }
86         leftRotate(root, grandparent);
87         enum Color temp = parent->color;
88         parent->color = grandparent->color;
89         grandparent->color = temp;
90         k = parent;
91     }
92 }
93 }
94 (*root)->color = BLACK;
95 }
96 void insert(struct Node** root, int data) {
97     struct Node* node = newNode(data);
98
99     if (*root == NULL) {
100         node->color = BLACK;

```

```

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

=== Code Execution Successful ===

```

```

101     *root = node;
102 } else {
103     struct Node* current = *root;
104     struct Node* parent = NULL;
105
106     while (current != NULL) {
107         parent = current;
108         if (node->data < current->data)
109             current = current->left;
110         else
111             current = current->right;
112     }
113     node->parent = parent;
114     if (node->data < parent->data)
115         parent->left = node;
116     else
117         parent->right = node;
118     fixInsert(root, node);
119 }
120 }
121 void preOrderTraversal(struct Node* root) {
122     if (root == NULL)
123         return;
124     printf("%d ", root->data);
125     preOrderTraversal(root->left);
126     preOrderTraversal(root->right);
127 }
128 int main() {
129     struct Node* root = NULL;
130     insert(&root, 10);
131     insert(&root, 20);
132     insert(&root, 30);
133     insert(&root, 15);
134     insert(&root, 25);
135
136     preOrderTraversal(root);
137     return 0;

```

```

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

=== Code Execution Successful ===

```