1. WRITE A C PROGRAM ON RED-BLACK TREE.

```
#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);

void leftRotate(Node **root, Node *x);

void rightRotate(Node **root, Node *y);

void insertFixup(Node **root, Node *node);

void insert(Node **root, int data);

void inorderTraversal(Node *root);

void freeTree(Node *root);

Node* createNode(int data) {

Node *newNode = (Node *)malloc(sizeof(Node));

newNode->data = data;

newNode->color = RED;

newNode->left = newNode->right = newNode->parent = NULL;

return newNode;

}

void leftRotate(Node **root, Node *x) {

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(Node **root, Node *y) {

Node *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;}

void insertFixup(Node **root, Node *node) {

Node *parent, *grandparent;

while (node != *root && node->parent->color == RED) {

parent = node->parent;

grandparent = parent->parent;

if (parent == grandparent->left) {

Node *uncle = grandparent->right;

if (uncle && uncle->color == RED) {

parent->color = BLACK;

uncle->color = BLACK;

grandparent->color = RED;

node = grandparent;

} else {

if (node == parent->right) {

node = parent;

leftRotate(root, node);

}

parent->color = BLACK;

grandparent->color = RED;

rightRotate(root, grandparent);

}

} else {

Node *uncle = grandparent->left;
```

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

```
if (root != NULL) {
inorderTraversal(root->left);
printf("%d (%s) ", root->data, root->color == RED ? "RED" : "BLACK");
inorderTraversal(root->right);
}
}
void freeTree(Node *root) {
if (root != NULL) {
freeTree(root->left);
freeTree(root->right);
free(root);
}
}
int main() {
Node *root = NULL;
int values[] = {8,18,5,15,17,25,40,80};
int n = sizeof(values) / sizeof(values[0]);
for (int i = 0; i < n; i++) {
insert(&root, values[i]);
}
printf("Inorder traversal of Red-Black Tree:\n");
inorderTraversal(root);
printf("\n");
freeTree(root);
return 0;
}
```

SAMPLE OUTPUT:

```
Inorder traversal of Red-Black Tree:
5 (BLACK) 8 (RED) 15 (BLACK) 17 (BLACK) 18 (RED) 25 (RED) 40 (BLACK) 80 (RED)
```

2. WRITE A C PROGRAM ON SPLAY TREE.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
int data;
struct Node *left, *right;
} Node;
Node* createNode(int data);
Node* rightRotate(Node *y);
Node* leftRotate(Node *x);
Node* splay(Node *root, int key);
```

```
Node* insert(Node *root, int key);

Node* search(Node *root, int key);

void inorderTraversal(Node *root);

void freeTree(Node *root);

Node* createNode(int data) {

Node *node = (Node *)malloc(sizeof(Node));

node->data = data;

node->left = node->right = NULL;

return node;

}

Node* rightRotate(Node *y) {

Node *x = y->left;

Node *T = x->right;x->right = y;

y->left = T;

return x;

}

Node* leftRotate(Node *x) {

Node *y = x->right;

Node *T = y->left;

y->left = x;

x->right = T;

return y;

}

Node* splay(Node *root, int key) {

if (root == NULL || root->data == key) {

return root;

}

if (key < root->data) {

if (root->left == NULL) return root;

if (key < root->left->data) {

root->left->left = splay(root->left->left, key);

root = rightRotate(root);

} else if (key > root->left->data) {

root->left->right = splay(root->left->right, key);

if (root->left->right != NULL) {

root->left = leftRotate(root->left);

}

}

return (root->left == NULL) ? root : rightRotate(root);

} else {
```

```
if (root->right == NULL) return root;
if (key > root->right->data) {
root->right->right = splay(root->right->right, key);
root = leftRotate(root);
} else if (key < root->right->data) {
root->right->left = splay(root->right->left, key);
if (root->right->left != NULL) {
root->right = rightRotate(root->right);
}
}
return (root->right == NULL) ? root : leftRotate(root);
}
}
Node* insert(Node *root, int key) {
if (root == NULL) return createNode(key);
root = splay(root, key);
if (root->data == key) return root;
Node *newNode = createNode(key);
if (key < root->data) {
newNode->right = root;
newNode->left = root->left;
root->left = NULL;
} else {
newNode->left = root;
newNode->right = root->right;
root->right = NULL;
}
return newNode;
}
Node* search(Node *root, int key) {
return splay(root, key);
}
void inorderTraversal(Node *root) {
if (root != NULL) {
inorderTraversal(root->left);printf("%d ", root->data);
inorderTraversal(root->right);
}
}
void freeTree(Node *root) {
if (root != NULL) {
```

```
freeTree(root->left);

freeTree(root->right);

free(root);

}

}

int main() {

Node *root = NULL;

root = insert(root, 10);

root = insert(root, 20);

root = insert(root, 30);

root = insert(root, 15);

root = insert(root, 25);

root = insert(root, 5);

printf("Inorder Traversal of Splay Tree:\n");

inorderTraversal(root);

printf("\n");

int key = 15;

root = search(root, key);

if (root != NULL && root->data == key) {

printf("Found %d in the tree.\n", key);

} else {

printf("%d not found in the tree.\n", key);

}

freeTree(root);

return 0;

}
```

SAMPLE OUTPUT:

```
Inorder Traversal of Splay Tree:

5 10 15 20 25 30

Found 15 in the tree.
```