NAME: MRUDHULA

REGNO: 192372290

//1.RED BLACK TREE:

```
/** C implementation for
        Red-Black Tree Insertion
        This code is provided by
        costheta_z **/
#include <stdio.h>
#include <stdlib.h>

// Structure to represent each
// node in a red-black tree
struct node {
        int d; // data
        int c; // 1-red, 0-black
        struct node* p; // parent
        struct node* r; // right-child
        struct node* l; // left child
};

// global root for the entire tree
struct node* root = NULL;

// function to perform BST insertion of a node
struct node* bst(struct node* trav,
                                        struct node* temp)
{
        // If the tree is empty,
```

```
        // return a new node
        if (trav == NULL)
                return temp;

        // Otherwise recur down the tree
        if (temp->d < trav->d)
        {
                trav->l = bst(trav->l, temp);
                trav->l->p = trav;
        }
        else if (temp->d > trav->d)
        {
                trav->r = bst(trav->r, temp);
                trav->r->p = trav;
        }

        // Return the (unchanged) node pointer
        return trav;
}

// Function performing right rotation
// of the passed node
void rightrotate(struct node* temp)
{
        struct node* left = temp->l;
        temp->l = left->r;
        if (temp->l)
```

```
            temp->l->p = temp;
        left->p = temp->p;
        if (!temp->p)
            root = left;
        else if (temp == temp->p->l)
            temp->p->l = left;
        else
            temp->p->r = left;
        left->r = temp;
        temp->p = left;
}

// Function performing left rotation
// of the passed node
void leftrotate(struct node* temp)
{
        struct node* right = temp->r;
        temp->r = right->l;
        if (temp->r)
            temp->r->p = temp;
        right->p = temp->p;
        if (!temp->p)
            root = right;
        else if (temp == temp->p->l)
            temp->p->l = right;
        else
            temp->p->r = right;
```

```
        right->l = temp;
        temp->p = right;
}

// This function fixes violations
// caused by BST insertion
void fixup(struct node* root, struct node* pt)
{
        struct node* parent_pt = NULL;
        struct node* grand_parent_pt = NULL;

        while ((pt != root) && (pt->c != 0)
                && (pt->p->c == 1))
        {
                parent_pt = pt->p;
                grand_parent_pt = pt->p->p;

                /* Case : A
                        Parent of pt is left child
                        of Grand-parent of
                pt */
                if (parent_pt == grand_parent_pt->l)
                {

                        struct node* uncle_pt = grand_parent_pt->r;

                        /* Case : 1
```

```
        The uncle of pt is also red
        Only Recoloring required */
if (uncle_pt != NULL && uncle_pt->c == 1)
{
        grand_parent_pt->c = 1;
        parent_pt->c = 0;
        uncle_pt->c = 0;
        pt = grand_parent_pt;
}

else {

        /* Case : 2
                pt is right child of its parent
                Left-rotation required */
        if (pt == parent_pt->r) {
                leftrotate(parent_pt);
                pt = parent_pt;
                parent_pt = pt->p;
        }

        /* Case : 3
                pt is left child of its parent
                Right-rotation required */
        rightrotate(grand_parent_pt);
        int t = parent_pt->c;
        parent_pt->c = grand_parent_pt->c;
```

```
                grand_parent_pt->c = t;
                pt = parent_pt;
        }
}

/* Case : B
        Parent of pt is right
        child of Grand-parent of
pt */
else {
        struct node* uncle_pt = grand_parent_pt->l;

        /* Case : 1
                The uncle of pt is also red
                Only Recoloring required */
        if ((uncle_pt != NULL) && (uncle_pt->c == 1))
        {
                grand_parent_pt->c = 1;
                parent_pt->c = 0;
                uncle_pt->c = 0;
                pt = grand_parent_pt;
        }
        else {
                /* Case : 2
                pt is left child of its parent
                Right-rotation required */
                if (pt == parent_pt->l) {
```

```
                        rightrotate(parent_pt);
                        pt = parent_pt;
                        parent_pt = pt->p;
                    }

                    /* Case : 3
                        pt is right child of its parent
                        Left-rotation required */
                    leftrotate(grand_parent_pt);
                    int t = parent_pt->c;
                    parent_pt->c = grand_parent_pt->c;
                    grand_parent_pt->c = t;
                    pt = parent_pt;
                }
            }
        }
}

// Function to print inorder traversal
// of the fixated tree
void inorder(struct node* trav)
{
    if (trav == NULL)
        return;
    inorder(trav->l);
    printf("%d ", trav->d);
    inorder(trav->r);
```

```
}

// driver code
int main()
{
        int n = 7;
        int a[7] = { 7, 6, 5, 4, 3, 2, 1 };

        for (int i = 0; i < n; i++) {

                // allocating memory to the node and initializing:
                // 1. color as red
                // 2. parent, left and right pointers as NULL
                // 3. data as i-th value in the array
                struct node* temp
                        = (struct node*)malloc(sizeof(struct node));
                temp->r = NULL;
                temp->l = NULL;
                temp->p = NULL;
                temp->d = a[i];
                temp->c = 1;

                // calling function that performs bst insertion of
                // this newly created node
                root = bst(root, temp);

                // calling function to preserve properties of rb
```

```
            // tree
            fixup(root, temp);
            root->c = 0;
        }

        printf("Inorder Traversal of Created Tree\n");
        inorder(root);

        return 0;
}
```

//2.SPLAY TREE:

```
#include <stdio.h>
#include <stdlib.h>
struct node {
  int data;
  struct node *leftChild, *rightChild;
};
struct node* newNode(int data){
  struct node* Node = (struct node*)malloc(sizeof(struct node));
  Node->data = data;
  Node->leftChild = Node->rightChild = NULL;
  return (Node);
}
struct node* rightRotate(struct node *x){
  struct node *y = x->leftChild;
  x->leftChild = y->rightChild;
  y->rightChild = x;
  return y;
}
struct node* leftRotate(struct node *x){
```

```
  struct node *y = x->rightChild;
  x->rightChild = y->leftChild;
  y->leftChild = x;
  return y;
}
struct node* splay(struct node *root, int data){
  if (root == NULL || root->data == data)
    return root;
  if (root->data > data) {
    if (root->leftChild == NULL) return root;
    if (root->leftChild->data > data) {
      root->leftChild->leftChild = splay(root->leftChild->leftChild, data);
      root = rightRotate(root);
    } else if (root->leftChild->data < data) {
      root->leftChild->rightChild = splay(root->leftChild->rightChild, data);
      if (root->leftChild->rightChild != NULL)
        root->leftChild = leftRotate(root->leftChild);
    }
    return (root->leftChild == NULL)? root: rightRotate(root);
  } else {
    if (root->rightChild == NULL) return root;
    if (root->rightChild->data > data) {
      root->rightChild->leftChild = splay(root->rightChild->leftChild, data);
      if (root->rightChild->leftChild != NULL)
        root->rightChild = rightRotate(root->rightChild);
    } else if (root->rightChild->data < data) {
      root->rightChild->rightChild = splay(root->rightChild->rightChild, data);
      root = leftRotate(root);
    }
    return (root->rightChild == NULL)? root: leftRotate(root);
  }
```

```
}
struct node* insert(struct node *root, int k){
  if (root == NULL) return newNode(k);
  root = splay(root, k);
  if (root->data == k) return root;
  struct node *newnode = newNode(k);
  if (root->data > k) {
    newnode->rightChild = root;
    newnode->leftChild = root->leftChild;
    root->leftChild = NULL;
  } else {
    newnode->leftChild = root;
    newnode->rightChild = root->rightChild;
    root->rightChild = NULL;
  }
  return newnode;
}
void printTree(struct node *root){
  if (root == NULL)
    return;
  if (root != NULL) {
    printTree(root->leftChild);
    printf("%d ", root->data);
    printTree(root->rightChild);
  }
}
int main(){
  struct node* root = newNode(34);
  root->leftChild = newNode(15);
  root->rightChild = newNode(40);
  root->leftChild->leftChild = newNode(12);
```

```
  root->leftChild->leftChild->rightChild = newNode(14);
  root->rightChild->rightChild = newNode(59);
  printf("The Splay tree is: \n");
  printTree(root);
  return 0;
}
```