Question 1. Red-Black Tree: Write codes for RB-Insert. Use the codes to reproduce the result of the example used in the lecture.

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

typedef struct RBTree_Node{
    RBTree_Node *parent, *left, *right;
    int value;
    int color_type;
}RBTree_Node;

RBTree_Node Nil;
RBTree_Node * nil = &Nil;

void LRotate(RBTree_Node * &T, RBTree_Node * selected_node){ // Left
Rotation.
    RBTree_Node * rotate_node;
    rotate_node = selected_node->right;
    selected_node->right = rotate_node->left;
    if(rotate_node->left != nil) rotate_node->left->parent =
selected_node;
    rotate_node->parent = selected_node->parent;
    int judgement = 0;
    if(selected_node->parent == nil) judgement = 1;
    else if(selected_node == selected_node->parent->left)
selected_node->parent->left = rotate_node;
    else selected_node->parent->right = rotate_node;
    rotate_node->left = selected_node;
    selected_node->parent = rotate_node;
    if(judgement == 1) T = rotate_node;
}

void RRotate(RBTree_Node * &T, RBTree_Node * selected_node){ // Right
Rotation.
    RBTree_Node * rotate_node;
    rotate_node = selected_node->left;
    selected_node->left = rotate_node->right;
    if(rotate_node->right != nil) rotate_node->right->parent =
selected_node;
    rotate_node->parent = selected_node->parent;
    int judgement = 0;
```

```
    if(selected_node->parent == nil) judgement = 1;
    else if(selected_node == selected_node->parent->right)
selected_node->parent->right = rotate_node;
    else selected_node->parent->left = rotate_node;
    rotate_node->right = selected_node;
    selected_node->parent = rotate_node;
    if(judgement == 1) T = rotate_node;
}

void RB_insert_changeStructure(RBTree_Node * &T, RBTree_Node *
insert_node){
    RBTree_Node * tmp;
        while(insert_node->parent->color_type == 'r'){
            if(insert_node->parent ==
insert_node->parent->parent->left){
                tmp = insert_node->parent->parent->right;
                if(tmp->color_type == 'r'){
                    tmp->color_type = 'b';
                    insert_node->parent->color_type = 'b';
                    insert_node->parent->parent->color_type = 'r';
                    insert_node = insert_node->parent->parent;
                }
                else if(insert_node == insert_node->parent->right){
                    insert_node = insert_node->parent;
                    LRotate(T, insert_node);
                }
                else{
                    insert_node->parent->color_type = 'b';
                    insert_node->parent->parent->color_type = 'r';
                    RRotate(T, insert_node->parent->parent);
                }
            }
            else{
                tmp = insert_node->parent->parent->left;
                if(tmp->color_type == 'r'){
                    insert_node->parent->color_type = 'b';
                    tmp->color_type = 'b';
                    insert_node->parent->parent->color_type = 'r';
                    insert_node = insert_node->parent->parent;
                }
                else if(insert_node == insert_node->parent->left){
                    insert_node = insert_node->parent;
                    RRotate(T, insert_node);
                }
```

```
                else{
                    insert_node->parent->color_type = 'b';
                    insert_node->parent->parent->color_type = 'r';
                    LRotate(T, insert_node->parent->parent);
                }
            }
        }
    if(insert_node->parent == nil){
        insert_node->color_type = 'b';
        return;
    }
    else{
        return;
    }
}

void RB_insert(RBTree_Node * &T, RBTree_Node * insert_node){
    RBTree_Node * tmp = nil;
    RBTree_Node * insert_position = T;
    while(insert_position != nil){  // Find the position of insert node.
        tmp = insert_position;
        if(insert_node->value < insert_position->value)
            insert_position = insert_position->left;
        else
            insert_position = insert_position->right;
    }
    insert_node->parent = tmp;
    if(tmp == nil) T = insert_node;
    else if(insert_node->value < tmp->value) tmp->left = insert_node;
    else tmp->right = insert_node;
    insert_node->left = nil; // Normal BTree insertion completed.
    insert_node->right = nil;
    insert_node->color_type = 'r';
    RB_insert_changeStructure(T, insert_node);
}

int main()
{
    nil->color_type = 'b';  // Default color_type is black.
    nil->left = nil;
    nil->right = nil;
    nil->parent = nil;
    RBTree_Node *T;
    T = nil;
```

```
    RBTree_Node container[9];
    int A[9] = {7,3,18,10,22,8,11,26,15};
    for(int i = 0; i < 9; i++)
    {
        container[i].value = A[i];
        RBTree_Node * insert_node = &container[i];
        RB_insert(T, insert_node);
    }
    printf("          (%d,%c)\n",T->value,T->color_type);
    printf("    (%d,%c)        (%d,%c)\n",T->left->value,T->left->color_
type,T->right->value,T->right->color_type);
    printf("(%d,%c)  (%d,%c)  ",T->left->left->value,T->left->left->colo
r_type,T->left->right->value,T->left->right->color_type);
    printf("(%d,%c)  (%d,%c)\n",T->right->left->value,T->right->left->co
lor_type,T->right->right->value,T->right->right->color_type);
    printf("          (%d,%c)        (%d,%c)\n",T->right->left->right->v
alue,T->right->left->right->color_type,T->right->right->right->value,T-
>right->right->right->color_type);
    return 0;
}
```

Results:

```
PS D:\Code> cd "d:\Code\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
          (10,b)
    (7,r)        (18,r)
(3,b)  (8,b)  (11,b)  (22,b)
          (15,r)        (26,r)
```