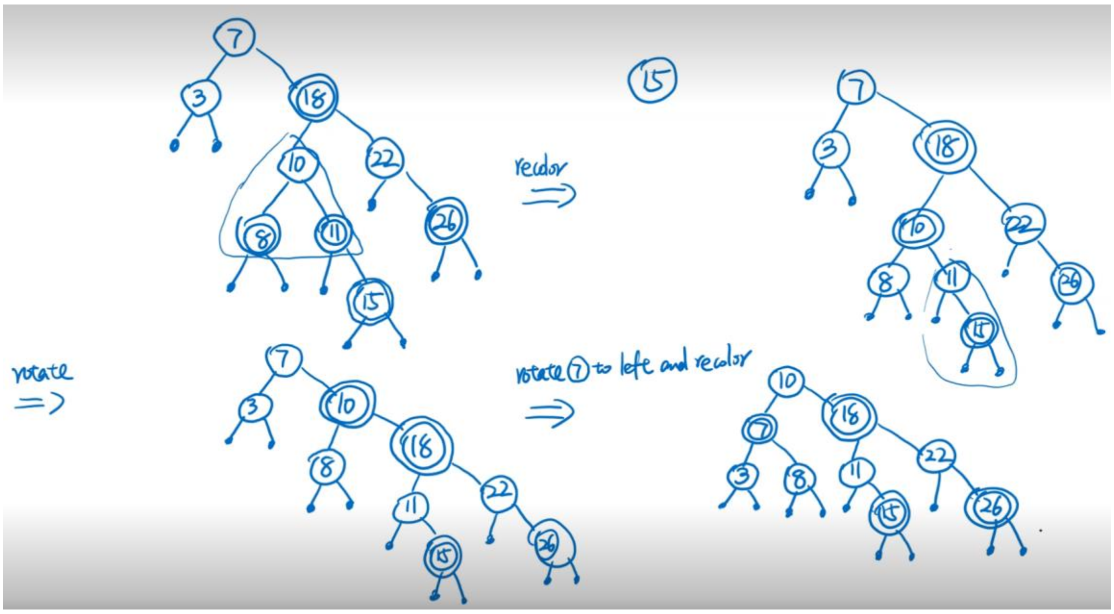
HW4_YIXIAO CHEN_002198256

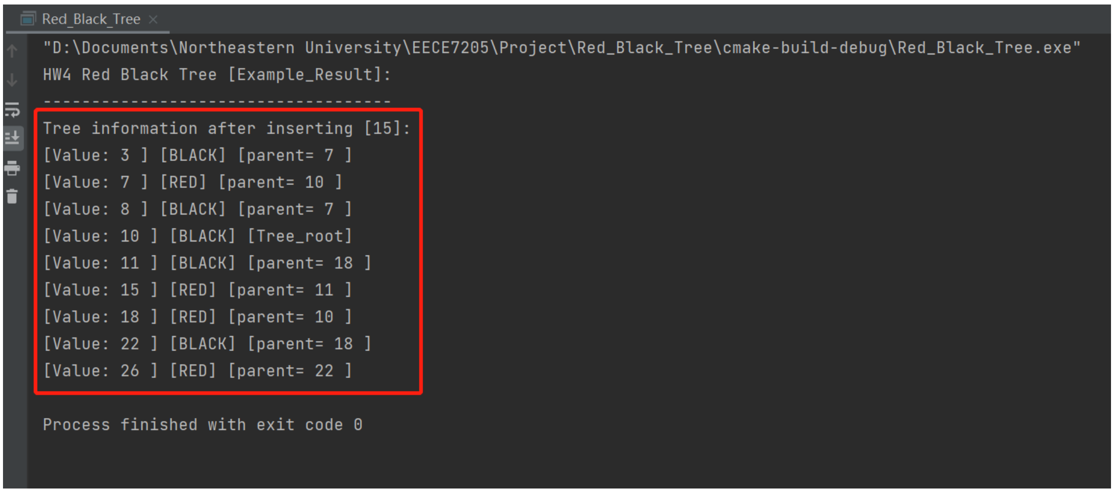
Question:

Question 1 (5 pt.) Red-Black Tree: Write codes for RB-Insert. Use the codes to reproduce the result of the example used in the lecture.



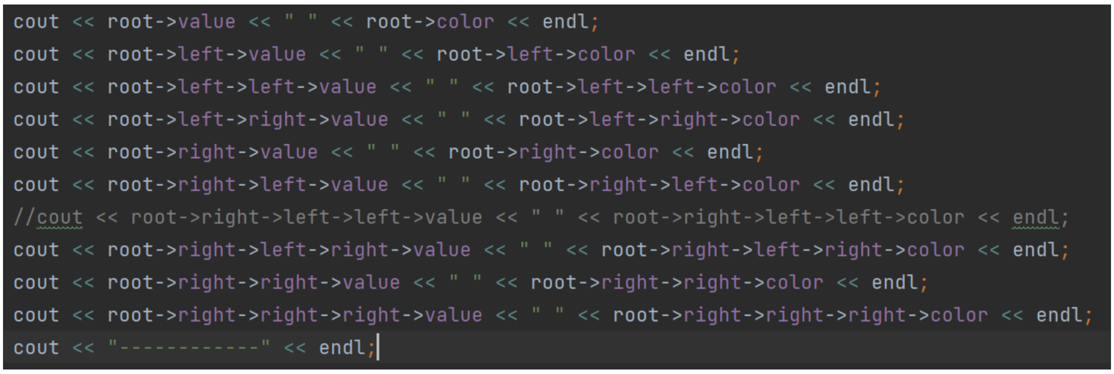
------**Codes** after result for every question below------

1.[RB-tree_Insert]:



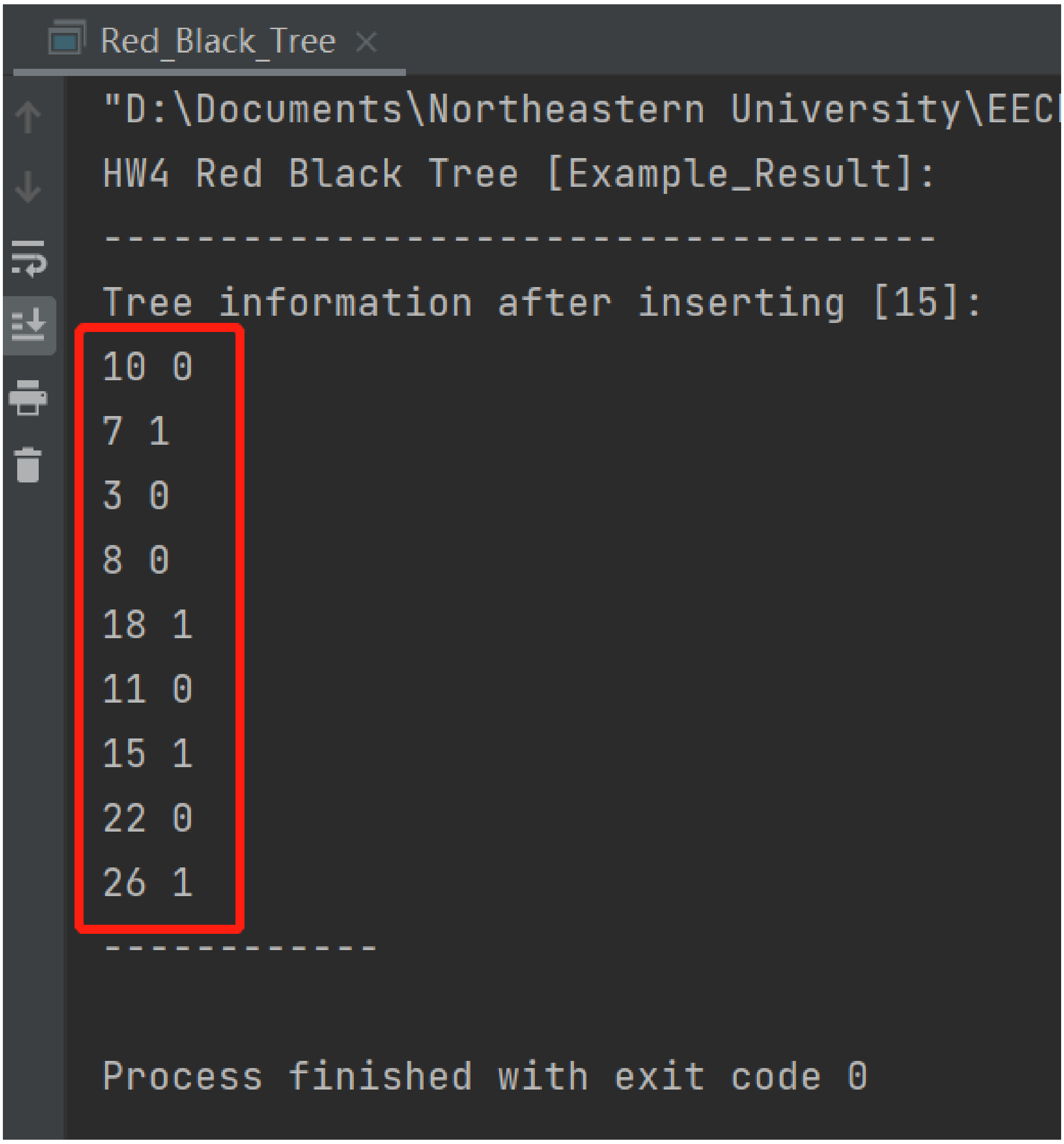
```
"D:\Documents\Northeastern University\EECE7205\Project\Red_Black_Tree\cmake-build-debug\Red_Black_Tree.exe"
HW4 Red Black Tree [Example_Result]:
------------------------------------
Tree information after inserting [15]:
[Value: 3 ] [BLACK] [parent= 7 ]
[Value: 7 ] [RED] [parent= 10 ]
[Value: 8 ] [BLACK] [parent= 7 ]
[Value: 10 ] [BLACK] [Tree_root]
[Value: 11 ] [BLACK] [parent= 18 ]
[Value: 15 ] [RED] [parent= 11 ]
[Value: 18 ] [RED] [parent= 10 ]
[Value: 22 ] [BLACK] [parent= 18 ]
[Value: 26 ] [RED] [parent= 22 ]

Process finished with exit code 0
```

2.[Testing Code & Result]:



```
cout << root->value << " " << root->color << endl;
cout << root->left->value << " " << root->left->color << endl;
cout << root->left->left->value << " " << root->left->left->color << endl;
cout << root->left->right->value << " " << root->left->right->color << endl;
cout << root->right->value << " " << root->right->color << endl;
cout << root->right->left->value << " " << root->right->left->color << endl;
//cout << root->right->left->left->value << " " << root->right->left->left->color << endl;
cout << root->right->left->right->value << " " << root->right->left->right->color << endl;
cout << root->right->right->value << " " << root->right->right->color << endl;
cout << root->right->right->right->value << " " << root->right->right->right->color << endl;
cout << "-----------" << endl;
```

Printing each node according to the final tree structure:



```
"D:\Documents\Northeastern University\EEC
HW4 Red Black Tree [Example_Result]:
------------------------------------
Tree information after inserting [15]:
10 0
7 1
3 0
8 0
18 1
11 0
15 1
22 0
26 1
------------


Process finished with exit code 0
```

【RB-tree_Insert】_by cyx

```
#include <iostream>

using namespace std;

struct Node {
    int value;
    int color;
    Node *right;
    Node *left;
    Node *parent;

    /** 结构体_构造函数 **/
    Node(int key, Node *&p) {
        value = key;
        color = 1;///新建节点的时候默认 color=1[red]
        left = nullptr;
        right = nullptr;
        parent = p;
    }
};

Node *root = NULL;///定义初始节点 Tree_root
Node *x, *y, *z, *parent;///全局定义后续函数可能用到的结构体

/** 移植函数 算法导论中有所提及 HW4 暂未涉及**/
/*void BP_transplant(){
}*/

void left_rotate(Node *x)/** 节点左旋 **/{
    y = x->right;
    /** 将 y 的左子节点 转变为 x 的右子节点 **/
    x->right = y->left;
    if (y->left != NULL) {
        y->left->parent = x;
    }
    /** 将 y 的父节点(x)变为 x 的父节点 **/
    y->parent = x->parent;
    /** 判断 x 节点是否为[根节点] 或 父节点的左/右节点 **/
    if (x->parent == NULL) {
        root = y;
    } else if (x == x->parent->left) {
        x->parent->left = y;
    } else {
        x->parent->right = y;
    }
    /** 交换x/y节点之间的父子关系 **/
    y->left = x;
    x->parent = y;
}

void right_rotate(Node *x)/** 节点右旋 **/{
    y = x->left;
    /** 将y的右子节点 转变为x的左子节点 **/
    x->left = y->right;
    if (y->right != NULL) {
        y->right->parent = x;
    }
    y->parent = x->parent;
    /** 关键判断：根节点 **/
    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 BP_insert_fixup(Node *z)/** 插入节点后 调整以维持RBT_property
**/{
    while (z != root && z->parent->color == 1) {
        if (z->parent == z->parent->parent->left) {
            y = z->parent->parent->right;
            if (y->color == 1) {
                z->parent->color = 0;
                y->color = 0;
                z->parent->parent->color = 1;
                z = z->parent->parent;
            } else {
                if (z == z->parent->right) {
                    z = z->parent;
                    left_rotate(z);
                }
                z->parent->color = 0;
                z->parent->parent->color = 1;
                right_rotate(z->parent->parent);
            }
        } else {
            /** same with "right"&"left" exchanged **/
            y = z->parent->parent->left;
            if (y->color == 1) {
                z->parent->color = 0;
                y->color = 0;
                z->parent->parent->color = 1;
                z = z->parent->parent;
            } else {
                if (z == z->parent->left) {
                    z = z->parent;
                    right_rotate(z);
                }
                z->parent->color = 0;
                z->parent->parent->color = 1;
                left_rotate(z->parent->parent);
            }
        }
    }
    root->color = 0;
}

void BP_insert(int a)/** 插入节点 **/{
    /*Node *parent = nullptr;
    Node *x = root;
    Node *y = NULL;
    Node *z = new Node(a, parent);*/
    parent = nullptr;
    x = root;
    y = NULL;
    z = new Node(a, parent);
    while (x != NULL) {
        y = x;
        if (z->value < x->value) {
            x = x->left;
        } else {
            x = x->right;
        }
    }
    z->parent = y;
    if (y == NULL) {
        root = z;
    } else if (z->value < y->value) {
        y->left = z;
    } else {
        y->right = z;
    }
    BP_insert_fixup(z);
}

/** HW4 中暂时没有涉及到RB_delete相关操作 但算法导论中有所提及 所以暂时保
留 **/
/*void BP_delete() {
}
void BP_delete_fixup() {
}*/

void RB_tree_walk(Node *x)/** 树遍历输出函数 参考[binary
tree]Inorder_tree_walk **/{
    if (x != NULL) {
        RB_tree_walk(x->left);
        cout << "[Value: " << x->value << " ] ";
        /** 判断颜色并输出 **/
        if (x->color == 1) {
            cout << "[RED] ";
        } else {
            cout << "[BLACK] ";
        }
        /** 判断是否为根节点 并输出父节点 **/
        if (x == root) {
            cout << "[Tree_root]\n";
        } else {
            cout << "[parent= " << x->parent->value << " ]\n";
        }
        RB_tree_walk(x->right);
    }
}

int main() {
    cout << "HW4 Red Black Tree [Example_Result]:" << endl;
    cout << "-------------------------------------" << endl;
    int input[9] = {7, 3, 18, 10, 22, 8, 11, 26, 15};
    for (int i = 1; i <= 9; i++) {
        BP_insert(input[i - 1]);
    }
    cout << "Tree information after inserting [15]:" << endl;
    RB_tree_walk(root);
    //getchar();
    //getchar();
    /** 调试专用：单步输入_输出测试代码 **/
    //BP_insert(7);
    /*cout << root->value << " " << root->color << endl;
    cout <<"--------------"<< endl;*/
    //BP_insert(3);
    /*cout << root->value << " " << root->color << endl;
    cout << root->left->value << " " << root->left->color << endl;
    cout <<"--------------"<< endl;*/
    //BP_insert(18);
    /*cout << root->value << " " << root->color << endl;
    cout << root->left->value << " " << root->left->color << endl;
    cout << root->right->value << " " << root->right->color <<
endl;
    cout <<"--------------"<< endl;*/
    //BP_insert(10);
    /*cout << root->value << " " << root->color << endl;
    cout << root->left->value << " " << root->left->color << endl;
    cout << root->right->value << " " << root->right->color <<
endl;
    cout << root->right->left->value << " " <<
root->right->left->color << endl;
    cout <<"--------------"<< endl;*/
    //BP_insert(22);
    /*cout << root->value << " " << root->color << endl;
    cout << root->left->value << " " << root->left->color << endl;
    cout << root->right->value << " " << root->right->color <<
endl;
    cout << root->right->left->value << " " <<
root->right->left->color << endl;
    cout << root->right->right->value << " " <<
root->right->right->color << endl;*/
    //BP_insert(8);
    /*cout << root->value << " " << root->color << endl;
    cout << root->left->value << " " << root->left->color << endl;
    cout << root->right->value << " " << root->right->color <<
endl;
    cout << root->right->left->value << " " <<
root->right->left->color << endl;
    cout << root->right->left->left->value << " " <<
root->right->left->left->color << endl;
    cout << root->right->right->value << " " <<
```

```
root->right->right->color << endl;*/
    //BP_insert(11);
    /*cout << root->value << " " << root->color << endl;
    cout << root->left->value << " " << root->left->color << endl;
    cout << root->right->value << " " << root->right->color <<
endl;
    cout << root->right->left->value << " " <<
root->right->left->color << endl;
    cout << root->right->left->left->value << " " <<
root->right->left->left->color << endl;
    cout << root->right->left->right->value << " " <<
root->right->left->right->color << endl;
    cout << root->right->right->value << " " <<
root->right->right->color << endl;*/
    //BP_insert(26);
    /*cout << root->value << " " << root->color << endl;
    cout << root->left->value << " " << root->left->color << endl;
    cout << root->right->value << " " << root->right->color <<
endl;
    cout << root->right->left->value << " " <<
root->right->left->color << endl;
    cout << root->right->left->left->value << " " <<
root->right->left->left->color << endl;
    cout << root->right->left->right->value << " " <<
root->right->left->right->color << endl;
    cout << root->right->right->value << " " <<
root->right->right->color << endl;
    cout << root->right->right->right->value << " " <<
root->right->right->right->color << endl;*/
    //BP_insert(15);
    /*cout << root->value << " " << root->color << endl;
    cout << root->left->value << " " << root->left->color << endl;
    cout << root->left->left->value << " " <<
root->left->left->color << endl;
    cout << root->left->right->value << " " <<
root->left->right->color << endl;
    cout << root->right->value << " " << root->right->color <<
endl;
    cout << root->right->left->value << " " <<
root->right->left->color << endl;
    //cout << root->right->left->left->value << " " <<
root->right->left->left->color << endl;
    cout << root->right->left->right->value << " " <<
root->right->left->right->color << endl;
```

```
    cout << root->right->right->value << " " <<
root->right->right->color << endl;
    cout << root->right->right->right->value << " " <<
root->right->right->right->color << endl;
    cout << "------------" << endl;*/
    //错误结果输出代码
    /*cout << root->value << " " << root->color << endl;
    cout << root->left->value << " " << root->left->color << endl;
    cout << root->right->value << " " << root->right->color <<
endl;
    cout << root->right->left->value << " " <<
root->right->left->color << endl;
    cout << root->right->right->value << " " <<
root->right->right->color << endl;
    cout << root->right->right->left->value << " " <<
root->right->right->left->color << endl;
    cout << root->right->right->left->left->value << " " <<
root->right->right->left->left->color << endl;
    cout << root->right->right->right->value << " " <<
root->right->right->right->color << endl;
    cout << root->right->right->left->left->right->value << " " <<
root->right->right->left->left->right->color << endl;*/
    return 0;
}
```