```cpp
#include "../src/main.cpp"
#include "../src/BalancedBST.cpp"
#include "../src/BalancedBST.h"
#define CATCH_CONFIG_MAIN
#include "catch.hpp"
#include <iostream>
#include <random>
#include <set>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

/*
To check output (At the Project1 directory):
g++ -std=c++14 -Werror -Wuninitialized -o build/test test-unit/test.cpp &&
build/test
*/

TEST_CASE("200 Items", "[flag]"){
mt19937 engine(random_device{}());
uniform_int_distribution<int> distribution(10000000, 99999999);
set<int> unique_numbers;
while (unique_numbers.size() < 200) {
unique_numbers.insert(distribution(engine));
}
vector<int> random_numbers(unique_numbers.begin(), unique_numbers.end());
sort(random_numbers.begin(), random_numbers.end());
for (const auto& number : random_numbers) {
tree.insert("name", number);
}

vector<Node*> inorder = tree.printInorderHelper();
vector<string> inorderS;
for (int i = 0; i < inorder.size(); i++) {
inorderS.push_back(inorder[i]->name);
}
REQUIRE(inorderS == random_numbers);
}
```

```
TEST_CASE("Left Rotate", "[flag]"){
BalancedBST tree; // Create a Tree object
tree.insert("a", 11111111);
tree.insert("b", 22222222);
tree.insert("c", 33333333);
vector<Node*> inorder = tree.printInorderHelper();
vector<Node*> preorder = tree.printPreorderHelper();
vector<Node*> postorder = tree.printPostorderHelper();
vector<string> inorderS;
vector<string> preorderS;
vector<string> postorderS;
for (int i = 0; i < inorder.size(); i++) {
inorderS.push_back(inorder[i]->name);
preorderS.push_back(preorder[i]->name);
postorderS.push_back(postorder[i]->name);
}
REQUIRE(inorderS == vector<string>{"a", "b", "c"});
REQUIRE(preorderS == vector<string>{"b", "a", "c"});
REQUIRE(postorderS == vector<string>{"a", "c", "b"});
}

TEST_CASE("Right Rotate", "[flag]"){
BalancedBST tree; // Create a Tree object
tree.insert("c", 33333333);
tree.insert("b", 22222222);
tree.insert("a", 11111111);
vector<Node*> inorder = tree.printInorderHelper();
vector<Node*> preorder = tree.printPreorderHelper();
vector<Node*> postorder = tree.printPostorderHelper();
vector<string> inorderS;
vector<string> preorderS;
vector<string> postorderS;
for (int i = 0; i < inorder.size(); i++) {
inorderS.push_back(inorder[i]->name);
preorderS.push_back(preorder[i]->name);
postorderS.push_back(postorder[i]->name);
}
REQUIRE(inorderS == vector<string>{"a", "b", "c"});
REQUIRE(preorderS == vector<string>{"b", "a", "c"});
REQUIRE(postorderS == vector<string>{"a", "c", "b"});
```

```
}

TEST_CASE("Right Left Rotate", "[flag]"){
BalancedBST tree; // Create a Tree object
tree.insert("a", 11111111);
tree.insert("c", 33333333);
tree.insert("b", 22222222);
vector<Node*> inorder = tree.printInorderHelper();
vector<Node*> preorder = tree.printPreorderHelper();
vector<Node*> postorder = tree.printPostorderHelper();
vector<string> inorderS;
vector<string> preorderS;
vector<string> postorderS;
for (int i = 0; i < inorder.size(); i++) {
inorderS.push_back(inorder[i]->name);
preorderS.push_back(preorder[i]->name);
postorderS.push_back(postorder[i]->name);
}
REQUIRE(inorderS == vector<string>{"a", "b", "c"});
REQUIRE(preorderS == vector<string>{"b", "a", "c"});
REQUIRE(postorderS == vector<string>{"a", "c", "b"});
}

TEST_CASE("Left Right Rotate", "[flag]"){
BalancedBST tree; // Create a Tree object
tree.insert("c", 33333333);
tree.insert("a", 11111111);
tree.insert("b", 22222222);
vector<Node*> inorder = tree.printInorderHelper();
vector<Node*> preorder = tree.printPreorderHelper();
vector<Node*> postorder = tree.printPostorderHelper();
vector<string> inorderS;
vector<string> preorderS;
vector<string> postorderS;
for (int i = 0; i < inorder.size(); i++) {
inorderS.push_back(inorder[i]->name);
preorderS.push_back(preorder[i]->name);
postorderS.push_back(postorder[i]->name);
}
REQUIRE(inorderS == vector<string>{"a", "b", "c"});
REQUIRE(preorderS == vector<string>{"b", "a", "c"});
REQUIRE(postorderS == vector<string>{"a", "c", "b"});
}
```