```
#include "../src/main.cpp"
#define CATCH_CONFIG_MAIN
#include "catch.hpp"

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

TEST_CASE("BST Insert", "[flag]"){
        /*
                MyAVLTree tree;   // Create a Tree object
                tree.insert(3);
                tree.insert(2);
                tree.insert(1);
                std::vector<int> actualOutput = tree.inorder();
                std::vector<int> expectedOutput = {1, 2, 3};
                REQUIRE(expectedOutput.size() == actualOutput.size());
                REQUIRE(actualOutput == expectedOutput);
        */
        REQUIRE(1 == 1);
}
// sample test 1
TEST_CASE("BST Insert", "[flag]"){
   AVL inputTree;
   inputTree.insert(3,"three");
   inputTree.insert(2,"two");
   inputTree.insert(1,"one");
   std::vector<std::string> actualOutput;
   inputTree.preorder(inputTree.root, actualOutput);
   std::vector<std::string> expectedOutput = {"two", "one", "three"};
   REQUIRE(expectedOutput.size() == actualOutput.size());
   REQUIRE(actualOutput == expectedOutput);
}

//sample test 2
TEST_CASE("BST Insert Large", "[flag]"){
   AVL inputTree;
   std::vector<std::string> expectedOutput, actualOutput;

   for(int i = 0; i < 100000; i++)
   {
      int randomInput = rand();
```

```
        if (std::count(expectedOutput.begin(), expectedOutput.end(), randomInput) == 0)
        {
            expectedOutput.push_back(std::to_string(randomInput));
            inputTree.insert(randomInput, std::to_string(randomInput));
        }
    }

    inputTree.inorder(inputTree.root, actualOutput);
    REQUIRE(expectedOutput.size() == actualOutput.size());
    //REQUIRE_FALSE(expectedOutput == actualOutput);    //This assertion can be wrong. Don't
use
    std::sort(expectedOutput.begin(), expectedOutput.end());
    REQUIRE(expectedOutput == actualOutput);
}
// Every rotation test
TEST_CASE("BST Insert", "[flag]"){
    AVL inputTree;
    inputTree.insert(10,"10");
    inputTree.insert(4,"4");
    inputTree.insert(2,"2");
        inputTree.insert(11,"11");
    inputTree.insert(12,"12");
    inputTree.insert(8,"8");
    inputTree.insert(0,"0");
        inputTree.insert(1,"1");
    std::vector<std::string> actualOutput;
    inputTree.preorder(inputTree.root, actualOutput);
    std::vector<std::string> expectedOutput = {"10", "4", "1","0","2","8","11","12"};
    REQUIRE(expectedOutput.size() == actualOutput.size());
    REQUIRE(actualOutput == expectedOutput);
}

// insertion break test
TEST_CASE("BST Insert", "[flag]"){
    AVL inputTree;
        std::vector<std::string> expectedOutput, actualOutput;
    for(int i = 0; i<1000<i++){
                inputTree.insert(i,std::to_string(i));
                expectedOutput.push_back(std::to_string(i));
        }
    inputTree.inorder(inputTree.root, actualOutput);
    REQUIRE(expectedOutput.size() == actualOutput.size());
    REQUIRE(actualOutput == expectedOutput);
}
```

```
// test deletion of entire tree
TEST_CASE("BST Insert", "[flag]"){
   AVL inputTree;
   inputTree.insert(10,"10");
   inputTree.insert(4,"4");
   inputTree.insert(2,"2");
         inputTree.insert(11,"11");
   inputTree.insert(12,"12");
   inputTree.insert(8,"8");
   inputTree.insert(0,"0");
         inputTree.insert(1,"1");
         inputTree.insert(19,"19");
         inputTree.insert(25,"25");

         for(int 10; i>0,i-- ){
                  inputTree.removeInorder(i-1);
         }
   std::vector<std::string> actualOutput;
         inputTree.preorder(inputTree.root, actualOutput);
   std::vector<std::string> expectedOutput = {};
   REQUIRE(expectedOutput.size() == actualOutput.size());
   REQUIRE(actualOutput == expectedOutput);
```