The three basic operations on binary search trees are *finding*, *inserting*, and *removing* a node with a specified key. As we know, the time required for these operations is proportional to the height of the tree. The purpose of this assignment is to compare the heights of non-rebalancing BSTs on the one hand and AVL trees on the other. In the worst case, their heights will be dramatically different: linear in the number of nodes for non-rebalancing BSTs and logarithmic for AVL trees. But we will consider trees constructed with randomly generated keys to reason about the average case. Asymptotically there is no difference: it can be shown that the expected height of a randomly built BST or AVL tree is $O(\log n)$. But let's see if the constants of proportionality are different by comparing actual constructions experimentally.

You are given the header files for two template classes, BST and AvlTree. BST represents a non-rebalancing binary search tree and AvlTree is a subclass of BST. It overrides the *insert* method to first perform an ordinary BST insertion and then perform trinode restructuring if the resulting tree is unbalanced. Your job is to implement these classes.

The most substantial part of this assignment by far is the implementation of *AvlTree::trinode_restructure*. My own solution includes ASCII tree art in the documentation to illustrate the four cases to be handled. These diagrams make the code easy to understand and they helped me to organize my own thoughts before starting to code. You will find the diagrams in the file *restructure.txt* and you are welcome to copy them into your own documentation if that helps.

You are given a test program (*tree_test.cpp*) that constructs and outputs a tree of each type built with a fixed set of 12 keys. If the program produces correct output using your BST and AvlTree implementations then your code is probably correct (but in this case the input is too small to reveal memory leaks). I will also test your code more rigorously with a program that a creates a large AVL tree with random keys and then traverses it to verify that each node is balanced (i.e., its children differ in height by at most 1). I recommend that you test your code in the same way.

Write a program called *main.cpp* that compares the heights of randomly built trees of both types for a range of sizes (2^1, 2^2, 2^3, ..., 2^{20}). The program's output should have the following form:

```
 2^1 keys:
 2^2 keys:
 2^3 keys:
     ⋮
2^19 keys:
2^20 keys:
```

In your program's actual output, the text on each line shown above will be followed by two numbers, the height of a non-rebalancing BST and the height of an AVL tree constructed with the same set of randomly generated keys. Both columns of numbers should line up neatly.

What to submit: *bst.cpp*, *avl_tree.cpp*, *main.cpp*