Section I: Checklist

Implementation Summary

	BST□LEAF	BST□ROOT	BST□RAND	AVL	HASH□OPEN	HASH□BUCKET
My template class has the required name, file extension, and template parameters. [y/n]	Y	Y	Y	Y	N	N
My template class was implemented efficiently using the required technique as described in the project description. [y/n]	Y	Y	Y	Y	N	N
I have implemented and throughly tested each of the required methods and they all behave correctly. [y/n]	Y	Y	Y	Y	N	N
AVL-based map only: I have throughly tested and verified that at the completion of an insertion/removal that the tree and all of its subtrees are AVL balanced.	—	—	—	N	—	—
I have implemented and throughly tested each of the Group 1 methods and they all behave correctly (bonus). [y/n]	Y	Y	Y	Y	N	N

I have implemented and throughly tested efficient iterators (both const and non-const) over a map instance's data and my map supports the bonus Group 2 methods iterator creation operations (bonus). [y/n]	Y	Y	Y	Y	N	N
I have verified by testing that a const instance of my map class and the data it holds cannot be modified, either through the map operations nor via iterator over the map's elements. [y/n]	N	N	N	N	N	N
I wrote my tests using CATCH. [y/n]	Y	Y	Y	Y	N	N
I have verified my map class is memory-leak free using valgrind. [y/n]	Y	Y	Y	Y	N	N
I certify that all of the responses I have given for this list class are TRUE [your initials]	mtm					

Section II: Learning Experience

This project, compared to project 1 was definitely more complex, but somehow felt simpler, maybe due to having less *required* things to implement. The easiest part of the project was traversing the binary tree; that is, inserting at a leaf, doing the `contains` method, and other parts where you had to simply iterate through the binary tree given a key. The hardest part was removing an item from the BST. It required the most code out of any of the methods, and it was mainly covering different cases which made it tedious. This is especially true on the AVL tree, where one had to intertwine the code to rebalance the tree with the `remove` code.

I think that the projects educational objectives were to force us into thinking about the advantages of BST's, and to imagine the worse case for a inserting at the leaf or root (as to solidify why self-balancing trees and bucket-based hash maps are important). I believe I achieved these goals: implementing the `height` and `balance` methods in particular really helped me visualize how inefficient the tree could become if you insert a bunch of sequential keys to the map.

Section III: Testing

The maps all rely on the same test code to test the required methods and the part 1 bonus methods; two test cases cover inserting, contains, lookup, and the size of the map. Another test case ensures that the copy-constructors work.

An additional test case was made specifically for the AVL tree to ensure that it remains AVL balanced as it goes through the operations.

AVL

Compile command

```
g++ ./avl/tests/main.cpp -std=c++11 -Wall -Wextra -Wpedantic
```

Program output

```
===============================================================================
==
All tests passed (560 assertions in 4 test cases)
```

Valgrind output

```
==11127== Memcheck, a memory error detector
==11127== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==11127== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright
info
==11127== Command: ./a.out
==11127==
===============================================================================
==
All tests passed (560 assertions in 4 test cases)
```

```
==11127==
==11127== HEAP SUMMARY:
==11127==     in use at exit: 0 bytes in 0 blocks
==11127==   total heap usage: 1,390 allocs, 1,390 frees, 369,391 bytes
allocated
==11127==
==11127== All heap blocks were freed -- no leaks are possible
==11127==
==11127== For counts of detected and suppressed errors, rerun with: -v
==11127== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

BSTLEAF

Compile command

```
g++ ./bstleaf/tests/main.cpp -std=c++11 -Wall -Wextra -Wpedantic
```

Program output

```
=============================================================================
==
All tests passed (60 assertions in 3 test cases)
```

Valgrind output

```
==11148== Memcheck, a memory error detector
==11148== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==11148== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright
info
==11148== Command: ./a.out
==11148==
=============================================================================
==
All tests passed (60 assertions in 3 test cases)

==11148==
==11148== HEAP SUMMARY:
==11148==     in use at exit: 24 bytes in 1 blocks
==11148==   total heap usage: 383 allocs, 382 frees, 112,250 bytes allocated
==11148==
==11148== LEAK SUMMARY:
```

```
==11148==    definitely lost: 24 bytes in 1 blocks
==11148==    indirectly lost: 0 bytes in 0 blocks
==11148==      possibly lost: 0 bytes in 0 blocks
==11148==    still reachable: 0 bytes in 0 blocks
==11148==         suppressed: 0 bytes in 0 blocks
==11148== Rerun with --leak-check=full to see details of leaked memory
==11148==
==11148== For counts of detected and suppressed errors, rerun with: -v
==11148== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

BSTRAND

Compile command

```
g++ ./bstrand/tests/main.cpp -std=c++11 -Wall -Wextra -Wpedantic
```

Program output

```
===============================================================================
==
All tests passed (60 assertions in 3 test cases)
```

Valgrind output

```
==11163== Memcheck, a memory error detector
==11163== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==11163== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright
info
==11163== Command: ./a.out
==11163==
===============================================================================
==
All tests passed (60 assertions in 3 test cases)

==11163==
==11163== HEAP SUMMARY:
==11163==     in use at exit: 24 bytes in 1 blocks
==11163==   total heap usage: 383 allocs, 382 frees, 112,250 bytes allocated
==11163==
==11163== LEAK SUMMARY:
==11163==    definitely lost: 24 bytes in 1 blocks
```

```
==11163==    indirectly lost: 0 bytes in 0 blocks
==11163==      possibly lost: 0 bytes in 0 blocks
==11163==    still reachable: 0 bytes in 0 blocks
==11163==         suppressed: 0 bytes in 0 blocks
==11163== Rerun with --leak-check=full to see details of leaked memory
==11163==
==11163== For counts of detected and suppressed errors, rerun with: -v
==11163== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

BSTROOT

Compile command

```
g++ ./bstroot/tests/main.cpp -std=c++11 -Wall -Wextra -Wpedantic
```

Program output

```
=============================================================================
==
All tests passed (60 assertions in 3 test cases)
```

Valgrind output

```
==11176== Memcheck, a memory error detector
==11176== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==11176== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright
info
==11176== Command: ./a.out
==11176==
=============================================================================
==
All tests passed (60 assertions in 3 test cases)

==11176==
==11176== HEAP SUMMARY:
==11176==     in use at exit: 24 bytes in 1 blocks
==11176==   total heap usage: 384 allocs, 383 frees, 112,274 bytes allocated
==11176==
==11176== LEAK SUMMARY:
==11176==    definitely lost: 24 bytes in 1 blocks
==11176==    indirectly lost: 0 bytes in 0 blocks
```

```
==11176==      possibly lost: 0 bytes in 0 blocks
==11176==    still reachable: 0 bytes in 0 blocks
==11176==         suppressed: 0 bytes in 0 blocks
==11176== Rerun with --leak-check=full to see details of leaked memory
==11176==
==11176== For counts of detected and suppressed errors, rerun with: -v
==11176== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```