CS515

Assignment 7

The purpose of this assignment is to give you practice working with binary search trees and help you gain a better understanding of the Map ADT. You are asked to implement a non-generic Map container using an underlying sorted binary search tree structure. To start, download all files from `~cs515/public/7P`

Our Map is implemented as an **ordered collection**, and the underling BST maintains a sorted ordering (ascending) among all key values. The map is not generic, and it only maps from a string key to a string value. (We will work on generic containers later in the course.)

You need to implement all the methods that are declared in the `map.h` file, implementing them in the source file `map.cpp`. You must also implement at least 2 tests that exercise each method of Map, building on the `mapTest.cpp` file you submitted for Assignment `5P`. You will implement a simple iterator for the Map as well as at least one test for each method. The iterator won't actually iterate since the tree is not threaded. Thus, you should not implement the overloaded operator++ for the iterator.

Some helper methods have code already given that you do not need to change. Note the BST has a pointer `_root` that points to a dummy root node. When the tree is empty, `_root->left` is set equal to `_root`.

Some Notes About Map Erase and Insert

- You must implement the map erase() method following the algorithm discussed in class. In particular, the node to be deleted is swapped with its *in-order successor*, and then is deleted.
- There will be two ways to insert a key-data pair into a Map:
  - Approach 1: Use the insert() method. The method will insert the key/value pair into the Map if the key doesn't exist, and return true. If the key already exists, the method will return false.
  - Approach 2: Use the overloaded operator[]. The operator[] is overloaded so we can search and access the data in a map similar as in a built-in array. Consider a map ***m*** contains entries that map string to string, and suppose ***m*** contains an entry associated with key "`a`", then
    ```
    string s = m["a"];
    ```
    will access the string indexed by the key "`a`" and assign its value to `s`.
    ```
    m["a"] = "new string value";
    ```
    will update the string that is associated with "a" to a new string `"new string value"`

    If, however, ***m*** does not contain any entries associated with key "`a`", then
    ```
    m["a"];
    ```
    will insert into map ***m*** a new entry whose key is "`a`", and its data (an empty string) is returned so we can update it with an assignment.
    Again, suppose ***m*** does *not* contain an entry associated with key "`a`", then
    ```
    m["a"] = "new string value";
    ```
    will insert a new entry made of the key "`a`" and the value "`new string value`".

Make sure to test your program thoroughly, considering all corner cases you can think of (in particular, try to find all basis paths through each method). The provided sample test is very inadequate and is there mostly to show you how the tree grows with each new insert, for initial debugging. You should use the tests that you wrote for 5P as a starting point for thorough tests here.

You need to keep in mind this BST-based Map implementation is not guaranteed to be O(logN) since a BST is not guaranteed to be balanced.

Submission:

- Submit **map.cpp map.h**, **mapTest.cpp**, **makefile**, and a **README** file.
  Make sure the provided header file map.h remains unchanged. You may submit the provided makefile, or use your own. However, the submitted `makefile` must have the following lines to ensure proper compilation with our grading test cases.

```
all: map.o mapTest
map.o: map.cpp map.h
       $(CXX) -c map.cpp

mapTest: mapTest.cpp map.o map.h
       $(CXX) mapTest.cpp map.o -lgtest -lpthread -o mapTest
       ./mapTest
```

- You should also fill out the README file and submit it as well. To submit your files, use the following command on agate:

  `~cs515/sub515` **`7P map.cpp map.h mapTest.cpp makefile README`**

- Do not turn in executables or object code.
- Make sure your submission compiles successfully on Agate. Programs that produce compile time errors or warnings will receive a zero mark.
- Be sure to provide comments in your program. You must include the information as the section of comments below:

```
/**    CS515 Assignment X
File: XXX.cpp
Name: XXX
Section: X
Date: XXX
Collaboration Declaration: assistance received from TA, PAC etc.
*/
```

Some notes on grading:

- Programs are graded for correctness (output results and code details), following directions and using specified features, documentation and style.
- To successfully pass the provided sample tests is not an indication of a potential good grade; to fail one or more of these tests is an indication of a potential bad grade.
- You must test thoroughly your program with your own test data/cases to ensure all the requirements are fulfilled. We will use additional test data/cases other than the sample tests to grade your program.
- Here is a tentative grading scheme.

	Test itself	Valgrind
Tests for constructors, assignment, etc. from 5P	2	
Tests for find() in mapTest.cpp	2	
Tests for begin() and end() in mapTest.cpp	4	
Tests for Iterator operators *, ->, == and !=	6	
map test runs 1-3 (each)	8	2
map test runs 4-6 (each)	10	2
map test run 7	8	2
Map test run 8 with Valgrind	10	
Others		
Use of STL containers	-100	
Fail to implement the given delete algorithm	-40	

Note: if one test-run program crashes at run time with a segmentation fault or bus error, the test fails completely and you will receive an error for this test case as well as the corresponding Valgrind test.