CptS 223 PA #1 - Binary Search Tree Heights

For this project, we will be empirically evaluating a binary search tree and comparing it to our own calculated analysis. To do this, we will use varying types of input values and chart out how the BST responds to the stimuli. We will test the BST with sorted data, perfectly balanced data, and randomly ordered data.

Your implementation needs to conform to a few standards. Notably, the BST MUST use templating to define what kind of data type it holds. The code must also use a Makefile to build, test, run, and clean the program. I have provided a testing object to give you the different data sets we'll be testing with, a working Makefile, and code that should compile (but not do anything useful).

During this course, I will be giving you less and less initial help with the programming assignments. This one has a strong framework for beginning, but the next one will rely more on what you've learned from this project. Make sure to read through the code to see a few key items:

1) How templating works in header files
2) How command line options are passed to the program
3) How I wrote up the makefile so you could add or edit it for your own uses

Using the Makefile

Included in the project source is a file called "Makefile". This file tells the make program how to build the program. If you're on the EECS servers it will work right out of the box. Here would be a normal series of commands to run with this Makefile. First, it will build the project, then run the program's tests, then run the program for real, lastly it cleans up the various temp files so the next time you build there won't be any leftovers around (not really a problem here, but with larger projects it is a very important feature):

```
$ make
$ make test
$ make run
$ make clean
```

Binary Search Tree (BST)

The code for this project is written in the expected C++11 language and tested using g++ on the EECS SSH servers. I have provided the interface for the BST, but very little of it is implemented. The code compiles, but there's no output.

You are to implement the BST. The most important features will be:

1) void add(value) -> Inserts an integer into the tree
2) int height() -> calculates the height of the tree
3) void printPreOrder() -> prints out the tree in pre order to STDOUT for inspection

There might be other functions or pieces you'll need to build, but that's up to your approach to solving the various project requirements. Add what you need to to make it work.

Statistics

Once you have a working BST implementation. You need to run it with several data sets and output the data to a CSV file. If you're not familiar with CSV files, please look up the file format online. Google is your friend here. It's a very simple format. It stands for Comma Separated Values. Each line in the file is like a row in a spreadsheet, where each cell is separated by the next one using a comma. A properly built CSV file can be loaded into a spreadsheet program (LibreOffice Calc, MS Excel, Google Sheets) and viewed as a spreadsheet, which is probably the easiest way to generate the charts for this assignment.

To get your data, you will need to make 7 BSTs. Each one will hold a different data set. These data sets are available in the provided TestData object in TestData.h. The BSTs will have:

1) Sorted data (in order from 0..N)
2) Perfectly balanced data
3) 5x trees, each holding scrambled (random order) data

The TestData object has interfaces to get the data:

```
int get_next_sorted()
int get_next_balanced()
int get_next_scrambled( setNumber )
```

These all return -1 when they're out of numbers. I STRONGLY suggest looking at the testing mode code to get an idea of how to use the interfaces.

The statistics you'll generate in the CSV file are the heights of the trees at every add (insert) into the tree. Each line in the CSV file will have 9 columns:

1) N: The number of elements in the tree
2) log(N): The calculation of $\log_2(N)$
3) Height of sorted tree
4) Height of balanced tree
5) Height of scrambled #0
6) Height of scrambled #1
7) Height of scrambled #2
8) Height of scrambled #3
9) Height of scrambled #4

The first line in the single CSV file sets the names of the columns. Here is a quick snapshot of how the CSV file should look opened up in Google Sheets:

	A	B	C	D	E	F	G	H	I
1	N	log_2 (N)	Sorted	Balanced	Scrambled #0	Scrambled #1	Scrambled #2	Scrambled #3	Scrambled #4
2	0	0	0	0	0	0	0	0	0
3	1	1	1	1	1	1	1	1	1
4	2	2	2	2	2	2	2	2	2
5	3								

Expected Output

Your program should generate one CSV file with the gathered statistics, called OutputData-BST.csv, which includes the statistics of the tree heights (and $\log_2$) for various N values and data orderings.

Based on that CSV file, which you can open in MS Excel, LibreOffice Calc, or Google Docs Sheets, create 2 charts plotting your calculated tree heights against the input size N. The first chart will have all 8 columns (including the sorted tree data), the second chart will be everything but the sorted tree data (since it will make everything look tiny)

- AllTreeHeights.png
- NoSortedTreeHeights.png

Given how much data there will be (over 200k data points), you should probably use very small plot points. The defaults are normally these huge diamonds, x's, or others. You just need the line to show the growth of the tree height as N increases.

Deliverables

Your program, resulting CSV file, and final chart images will need to be stored in the PA1 directory of your git repository for the class. You MUST get your code ready, tested, and finished before you do a final commit. You're more than welcome to commit and push as many times as you like as you do your development, but only one commit will be looked at for grading. That commit must be before the project due date & time to be counted as "on time". Once you have committed your code and pushed it up to the server, you submit the hash of the commit to blackboard so the TA can find it. Below is an example of using git log on the command line to find the commit hash:

```
>>> /home/acrandal/gittools/grading/cs223-acrandal/PA1 $ git log
>>> commit d10632fc21e7fce5f9886cf1c694ad87d906b02f
>>> Author: Aaron S. Crandall <acrandal@wsu.edu>
>>> Date:   Sun Sep 3 16:50:26 2017 -0700
>>>
>>>     modified for testing of main.cpp
>>>
>>> commit e9ed57cd05eb78c0404b2b2c6a833e750b4d8ed6
>>> Author: Aaron S. Crandall <acrandal@wsu.edu>
```

>>> Date: Sun Sep 3 16:18:37 2017 -0700
>>>
>>> PA1 start code for BST height charting

The long string of hex characters highlighted in red is the commit hash for the commit I'd want to upload to Blackboard. I would copy that string into a text file and upload that text file to Blackboard. The text file is either a normal file on Linux, or one that ends with .txt in MS Windows. The person grading your assignment will then check out your source code with that commit hash, which will show them exactly what you had checked in there. Any work done after that commit won't be visible when the grader runs "make", so ensure your final commit has everything you expect it to have. Use the GitHub web interface at git.eecs.wsu.edu to check that everything pushed properly.

You can also find the hash of any commit in the repository by going to the web interface at git.eecs.wsu.edu. Once you navigate to your cs223 repository, you can click on the "Graphs" tab, then the Network section. Each dot in that network represents a commit. If you click on one, you will see what changed with the commit and the hash is shown in the upper right corner of that page.

Grading Criteria

Your assignment will be judged by the following criteria:

- [70] Code operational success. Your code compiles, executes, and generates the CSV files.
- [10] Your code is well documented and generally easy to read.
- [10] Your program intelligently uses classes when appropriate and generally conforms to good OOP design (i.e. everything isn't slapped into main).
- [10] The CSV and charts are in the correct formats, with labeled axes, title, and legend (as appropriate)