CS 302
Homework, Asst. #07

Purpose: Learn concepts regarding ***AVLtree*** and ***trie*** data structures.
Due: Thursday (10/16) → Must be submitted on-line before class.
Points: Part A → 25 pts, Part B → 50 pts

Assignment:

Part A:
Update the *avlTree.h* template class to add a function *isPrefix()*. The *isPrefix()* function should work the same as the previous assignment, but function on the ***avlTree*** data structure. No other functions need be updated or added. Test the updated data structure to ensure it works correctly.

When working, update the header and implementation files for the wordPuzzle class from the previous assignment. Replace the ***trie*** and use ***avlTree<string>*** instead. Compile and test.

When working, execute and time the main from assignment #6 (using the ***trie***) three times. Next, execute and time the main from assignment #7 (using the ***avlTree***) three times. To automate the timing and capture the results, create a script file that accepts the executable files (from asst #6 and asst #7, in that order) as arguments, performs the three executions each, and captures the results to a text file.

Part B:
Create and submit a brief write-up including the following:

- Name, Assignment, Section.
- Summarize the results from the timing script.
- Compare using an ***trie*** to an ***avlTree*** for this problem.
  - Compare the run-time complexity for the insert, search, and isPrefix functions for each data structure.
- Estimate the data space required for the the small dictionary for each data structure. You can use the node count result for each data structure times the size of each node as an estimate.
- As an appendix, summarize the hardware environment and include the timing results (at the end) of the timing script.

Submission:
- Submit a compressed zip file of the program source files, header files, and makefile via the on-line submission by 23:50.
- Submit a copy of the write-up (open document, word, or PDF format).

All necessary files must be included in the ZIP file. The grader will download, uncompress, and type **make** (so you must have a valid, working *makefile*).

Make File:

You will need to develop a make file. You should be able to type:

```
make
```

Which should create the executables.