Project #5 - Building a word autocomplete application using an alphabet trie

Learning Objectives

- Demonstrate effective use of memory management techniques in C++
- Implement a data structure to meet given specifications
- Design, implement, and use a trie data structure
- Analyze operations for time complexity

Overview

Your task for this assignment is to implement an alphabet trie data structure, and to use this data structure to write an autocomplete program using an English dictionary.

The Trie Class

In order to implement your autocomplete program, you will need to create a trie data structure (class Trie) to facilitate efficient word searches from an English dictionary.

- **`bool Trie::insert(string)`** – Insert a new word into the trie. Duplicate words are not allowed. This function should return **true** if the word is successfully inserted into the trie, and **false** if the word could not be inserted (for example, due to a duplicate word already found in the trie).

- **`int Trie::count()`** – return the number of **words** in the trie

- **`int Trie::getSize()`** – return the total number of **nodes** in the trie

- **`bool Trie::find(string)`** – if the given word is found in the trie, this function should return **true**, otherwise this function should return **false**

- **`int Trie::completeCount(string)`** – this function should return the number of words in the dictionary that begin with the given input string. If no matching words are found, the function should return zero.

- **`vector<string> Trie::complete(string)`** – this function should return a C++ vector of strings containing all of the words in the dictionary that begin with the given input string. For each word found in the trie, there will be one value in the vector. If no matching words are found, the function should return an empty vector.

  Example: The call **`resultVector = myTrie.complete("addr")`** were called on a trie built with the wordlist.txt file provided with this project should return a vector containing the strings: {"address", "addressable", "addressed", "addressee", "addressees", "addresses", "addressing"}.

- **`Trie& Trie::operator=(const Trie&)`** – **`trie1 = trie2`** should remove all contents of trie1 (without memory leaks) and make an independent copy of trie2 in trie1.

- **`Copy constructor`** – must correctly make an independent trie that is an exact copy of the original trie.

Main program using the trie

You should test your trie with a main() function that loads the provided English dictionary file, prompts the user for a prefix, and then uses the trie to find all completions for the prefix. An example of the execution of your main program follows:

```
Please enter a word prefix (or press enter to exit):  addr
There are 7 completions for the prefix 'addr'.  Show completions?  Yes
Completions
-----------
address
addressable
addressed
addressee
addressees
addresses
addressing

Please enter a word prefix (or press enter to exit):
```

Turn in and Grading

- The Trie class should use a seperate Trie.h and Trie.cpp file.
- The autocomplete program should be in a file named project5.cpp.
- Please zip your entire project directory into a single file called Project5.zip.
- Your trie will be evaluated using **both** your autocomplete program, **and** with a test harness, so be sure that your functions conform completely to the specifications in this assignment.

This project is worth 50 points, distributed as follows:

Task	Points
`Trie::insert` stores words correctly in the trie, and correctly rejects duplicate words	5
`Trie::count` correctly returns the number of words in the trie	2
`Trie::getSize` correctly returns the number of nodes in the trie	4
`Trie::find` correctly determines if a word is present in the trie	4
`Trie::completeCount` correctly returns the number of words in the trie matching a given prefix	5
`Trie::complete` correctly returns a C++ vector of words (strings) that begin with the given prefix	5
Copy constructor works correctly.	5
`Trie::operator=` works correctly.	5
Note that your code must implement an alphabet trie to get credit for any of the above items.	

No memory leaks	5
Autocomplete program correctly reads user input, prints number of words that begin with the prefix, and outputs all words with the prefix.	5
Code is well organized, well documented, and properly formatted. Variable names are clear, and readable.	5