CS 3345 Data Structures Project 2

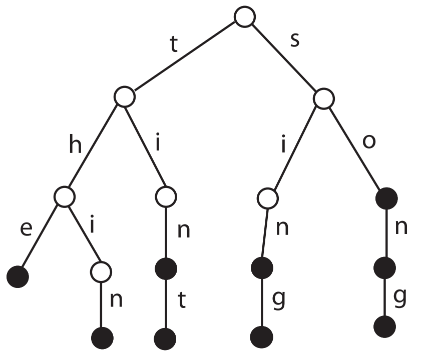
You are going to write a class to implement a Trie data structure and a program to test it

The trie efficiently stores words, as in a spell-checker. To simplify matters, words will be restricted to strings of lower-case characters, including no spaces or punctuation marks.

The member functions will be:

```
boolean insert(String s);      // returns false if s is already present, true otherwise
boolean isPresent(String s);   // returns true if s is present, false otherwise
boolean delete(String s);      // returns false if s is not present, true otherwise
int membership();              // returns the number of words in the data structure
void listAll();                // list all members of the Trie in alphabetical order
```

A Trie is a tree of degree 26. The following Trie contains the words: the, thin, tin, tint, sin, sing, son, song.



There is a boolean variable, `terminal` in each node. In the black Nodes that variable set to `true` to indicate the end of a word.

Here are the class variables for a Node:

```
class Node {
   boolean terminal;
   int outDegree;
   children Node[];
}
```

Each Node includes an array of 26 references to Node. When a Node is created, all the array elements will be initialized to null and the outDegree will be set to zero. No other variables may be added to the Node class.

In the above Trie, the root node has only two children, corresponding to characters 's' and 't', or positions 19 and 20 in the array of references.

A search begins at the root Node and follows the links, one level for each character in the given word. To find the word "tin" we begin with array element 20 in the root Node and, if it is present, we check element 8 in the array of the second level node. Finally, we check element 14 in the array of the third level Node. If that Node is marked as a terminal, we return "true".

The insert function begins with a search for the given word. If a null reference is found, a sequence of new Nodes is created corresponding to the missing word ending.

To insert the word “soap” to the above Trie, two new nodes would be added, corresponding to ’a’ and ’p’. The ’a’ in this suffix would be referenced by the third level Node corresponding to the prefix “so”.

Deletion also begins with a search for the given word. If found, there are two cases. Say we are deleting “sin” from the above Trie. Since the terminal character of the given word has children, deletion only requires flipping the boolean variable that indicates the end of a word.

If the word “thin” is deleted, the suffix “in” must be unlinked from the third level node. To do this, when recursing out from the end of the given word, remove links to the characters of that word until a Node is found with degree greater than 1, or a terminal node is found. The former case occurs when deleting the word “thin”. The word “the” must remain. The latter case occurs when deleting the word “song.” The word “so” must remain.

Your program will read a text file from System.in that contains text commands. In response to each command your program will output one or more lines of text to system.out.

Here are examples of the commands:

```
A soap    // Insert the word 'soap'
          // Print one of the strings "Word inserted" or "Word already exists"
          // followed by a newline.
D sin     // Delete the word 'sin'.
          // Print "Word deleted" or "Word not found" followed by a newline.
S fortune // Search for the word "fortune".
          // Print "Word found" or "Word not found" followed by a newline.
M         // Print "Membership is ####" where #### is the number of words in the Trie
C wet     // Search for the word wet in the Trie and, if not present,
          // output the phrase "Spelling mistake" followed by a space and then the word wet.
          // If the word is found, output the phrase "Spelling OK" followed by a space and
          // then the word wet.
L         // Print a list of the elements of the Trie in alphabetical order, one word per line.
E         // The end of the input file
```

Here are sample input and output data files:

```
Sample Input                      Sample Output
------------                      --------------
A ant                             Word inserted
A goat                            Word inserted
A frog                            Word inserted
A art                             Word inserted
A goad                            Word inserted
A antler                          Word inserted
A go                              Word inserted
A from                            Word inserted
A part                            Word inserted
A past                            Word inserted
A arts                            Word inserted
A part                            Word already exists
A frond                           Word inserted
A fries                           Word inserted
A text                            Word inserted
A message                         Word inserted
A mess                            Word inserted
A mean                            Word inserted
A goal                            Word inserted
A interpretation                  Word inserted
A coat                            Word inserted
M                                 Membership is 20
```

```
D art                             Word deleted
D art                             Word not found
D mess                            Word deleted
S art                             Word not found
M                                 Membership is 18
S antler                          Word found
S part                            Word found
S text                            Word found
S freis                           Word not found
C pert                            Spelling mistake pert
C interpretacion                  Spelling mistake interpretacion
C anteler                         Spelling mistake anteler
C frog                            Spelling OK frog
L                                 ant
E                                 antler
                                  arts
                                  coat
                                  fries
                                  frog
                                  frond
                                  from
                                  go
                                  goad
                                  goal
                                  goat
                                  interpretation
                                  mean
                                  message
                                  part
                                  past
                                  text
```

RULES FOR PROGRAMMING AND SUBMISSION:

1. Write your program as one source file and do not use the“package” construct in your Java source.
2. Name your source file as $N_1N_2F_1F_2$P2.java where your given name begins with the characters N_1N_2 and your family name begins with the characters F_1F_2. For example my name is Ivor Page, so my source file will be called IVPAP2.java. Note that in all but the “java” extension, the characters are upper case.
3. Your program must not output any prompts. Its output must exactly match the format of the Sample Output above
4. Do not use any Java Collection Classes, except Strings and arrays. If in doubt, ask me.
5. Use good style and layout. Comment your code well.
6. Your program MUST read from System.in and write to the System.out (cin, cout for C++ programs.
7. Submit your ONE source code file to the eLearning Assignment drop box for this assignment. Don’t submit a compressed file.
8. There will be a 1% penalty for each minute of lateness. After 60 minutes of lateness, a grade of zero will be recorded.

Send me any questions/corrections (ivor@utdallas.edu).