CSC172 PROJECT 3

WORD PUZZLE

1 Introduction

This project will require you to create a Java program that will take an input file consisting of n lines each line containing n characters. The goal of this puzzle is to find the words embedded in the the array of words.

Consider the following 4x4 example

```
this
wats
oahg
fgdt
```

The puzzle above contains the the words *this*, *two*, *fat* and *that*. The word *this* begins at row 1 column 1 and extends to row 1 column 4. The word *two* begins at row 1 column 1 and extends to row 3 column 1. The word *fat* begins at row 4 column 1 and extends to row 2 column 3. The word *that* begins at row 4 column 4 and extends to row 1 column 1. A word can have 8 possible orientations (directions).

Your program will need to read in a list of legal words. Use the list provide in the words.zip file that contains approximately 479829 words. Your program should be able to process a 20x20 puzzle in reasonable time.

2 Hash Tables

In order to do this, you should construct a hash table and read the words from the dictionary into it. Once you have built the hash table, then you should Solve the puzzle. **You must write your own hash table class rather than using a library.** There is a discussion of this puzzle in your text book. The word list can be entered into a hash table and analyzed for maximum word length. **You must write your own hash function rather than using the Java supplied hash values.** The puzzle characters can be stored in a two dimenstional array. There are eight possible orientations for words (0 : straight up, 1 : up diagonal right, 2 : left to right , 3: down diagonal right . . . 6: right to left, 8 ; up diagonal right to left). It is straight forward to generated quadruples (row, column, orientation,length) with nested loops extract the characters and test it against the hash table containing the word list. An efficient implementation of a hash table O(1) makes this size problem tractable.

3 Deliverable

For this project, you will need to write a Java program that reads in a list of words from a plain text file, and reads an nxn word puzzle from a plain text file. You can assume that the puzzle file is properly formatted. Extra credit will be awarded if your program can safely handle invalid input by printing a relevant error message in the output file.

The file "puzzle.txt" provided with this prompt contains a subset of the expressions the TAs will use when grading your program. The output of your program should exactly match "wordsfound.txt" in order to receive full credit. Your found output words should be one per line in alphabetical order. You can check if two files are equivalent by using the "diff" command on OS X or Linux, or "FC" on Windows. It is strongly recommended that you write your own (additional) test cases and submit them with your source code to demonstrate your program's capabilities.

The locations of the input and output files will be supplied to your program via the command line. The first command line argument will be the location of the input word list file, the second argument will be the location of the puzzle file, and the third argument is where your solutions should be stored. For example, if your main method were in a class called InfixCalculator, your program should be run as:

```
java WordFinder wordlist.txt puzzle.txt foundWords.txt
```

4 Hand In

Hand in the source code from this lab at the appropriate location on the Blackboard system at my.rochester.edu. You should hand in a single compressed/archived (i.e. "zipped" file that contains the following.)

1. A plain text file named README that includes your contact information, a detailed synopsis of how your code works and any notable obstacles you overcame, and a list of all files included in the submission. If you went above and beyond in your implementation and feel that you deserve extra credit for a feature in your program, be sure to explicitly state what that feature is and why it deserves extra credit.
2. Source code files (you may decide how many you need) representing the work accomplished in this project. All source code files should contain author identification in the comments at the top of the file.
3. A plain text file named OUTPUT that includes author information at the beginning and shows the compile and run steps of your code. The best way to generate this file is to cut and paste from the command line.
4. Any additional files containing extra test cases and your program's corresponding output. Theses cases should be described in your README so the grader knows what you were testing and what the expected results were.

5 Grading

70% Functionality

10% driver program that handles File I/O

	30% finding the words in the puzzle
	30% implementing an algorithm (hash function & table) with good run time efficiency
20%	Testing
	15% program passes the short tests
	5% program passes the extended tests
10%	README

Extra Credit

10%	Graphical output indicating word location (circle the words)