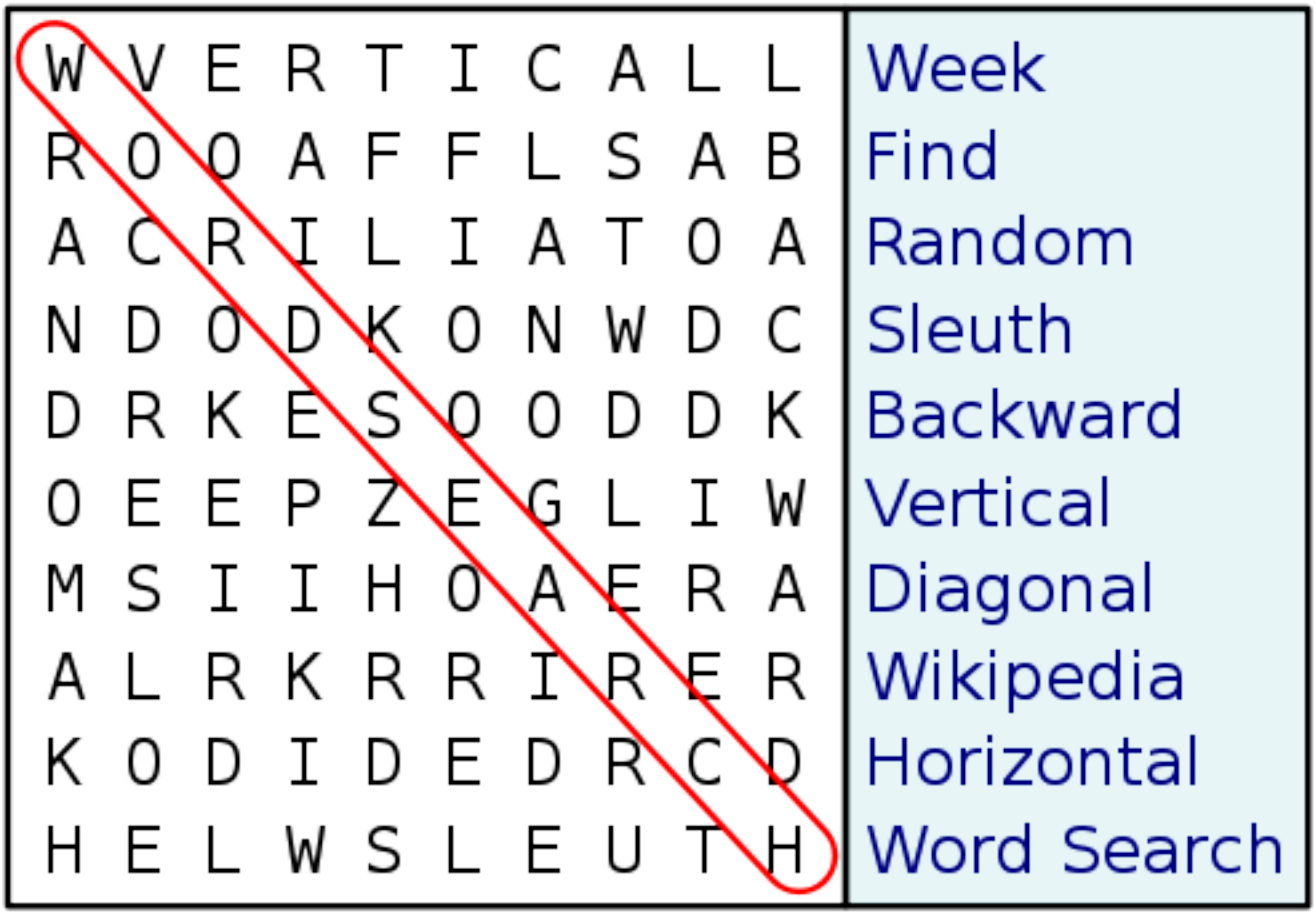
World Puzzle Problem

Consider the problem called word search, illustrated in the diagram below.



In this 10x10 grid of letters, the goal is to find words in the puzzle in any of the eight directions. The word circled in red is in the south-east direction. Words can go backward (although none do in this example), but they cannot "wrap around" from one side to the other (or from top to bottom, etc.).

For the purposes of this lab, your program will be presented with a grid of letters and a dictionary of words. All words in the dictionary that are in the grid, in any of the 8 directions, are to be outputted.

A string of letters from the grid will depend on four values:

- The *x* value of the starting letter
- The *y* value of the starting letter
- The direciton, *d*, of the word (directions can be represented as integers 1-8, if that is easier)
- The length, *l*, of the string

This implies that your code will have quad-nested `for` loops.

The goal for the pre-lab is to get your puzzle solver working, using a hash table that you write, without worrying about efficiency. The goal of the post-lab is to increase it's efficiency.

A few notes about the length of the words: the program to implement will not consider words of

length less than 3, and the maximum word length is a constant (which is defined as the longest word in the input dictionary file provided to the program at run-time).

There are a number of optimizations that one can implement, a few of which are mentioned here. Depending on how you implement them, they may not work all that well, of course.

- Choose a good load factor, λ, for your hash table
- Implement a reasonable collision resolution strategy
- In addition to storing each word, *W*, you can also store the *prefixes* of that word. So if the word is "amazing", you would store "ama", "amaz", "amazi", "amazin", and "amazing" in the hash table. There would need to be some way to differntiate between prefixes ("amaz") and the actual words ("amazing"). This way if you are working in a given direction, and the particualr string you are generating is not a prefix, then you know there are no further words in the dictionary in the given direction.

You can keep track of a previous hash to help compute the next one faster. For example, if you have just computed the hash for "foo", then you can keep that hash value on hand to compute the hash for "food" faster.