Lab 9 – Crouching Lion, Hidden Dragon

In this lab we'll implement a program that searches for hidden words in a two-dimensional grid of letters contained in a file. A sample interaction with the finished program is shown below. Notice that both the position and direction of a word are displayed. If the same word appears more than once on the grid, you only need to report the first one found, although if you wish you may report all occurrences of the word.

The position of a letter on the grid is given by its *row* and *column* coordinate, counting from zero. For example, the letter `x` in the grid below is at row 0, column 4.

```
>>> findwords("grid1.txt")

c a t n x f
k c o r e r
d i o k c o
l r a w i g
d n i l m i
s o w b l p
n o g a r d

Enter a word to search for: cat
cat found at (0, 0) going right

Enter a word to search for: bird
bird found at (5, 3) going up left

Enter a word to search for: dog
dog found at (4, 0) going down right

Enter a word to search for: rock
rock found at (1, 3) going left

Enter a word to search for: puppy
puppy not found

Enter a word to search for: quit
Thanks for playing!
>>>
```

Download and unzip the folder **lab09_files.zip**. The above letter grid is available in the file **grid1.txt**. It contains the following hidden words: *cat*, *cow*, *frog*, *dog*, *rock*, *bird*, *mice*, *snake*, *owl*, *pig*, *lion*, and *dragon*.

We will represent the search direction as one of the following eight strings: "up", "down", "left", "right", "up left", "up right", "down left", or "down right". For convenience, a list called `directionNames` containing these strings has already been defined for you in the file **findwords.py**, which you should use as your starting point for this lab.

Implementing the Program

We will break the problem into pieces, and use the pieces as building blocks. Write, test, and debug the following functions one by one, in the order shown, using earlier functions to help you define later ones.

1. In class, we wrote the function **readGrid(*filename*)**. This function reads in a grid of letters from a text file, and returns a two-dimensional list (a list of lists) containing the letters, with each row of the grid represented as an ordinary one-dimensional list. We also wrote the function **printGrid(*grid*)**, which takes a grid as input and prints out the grid nicely with the letters separated by spaces, as well as the functions **numRows(*grid*)** and **numColumns(*grid*)**, which takes a grid and returns the number of rows or columns in the grid. Test out these four functions using the file **grid1.txt**, as well as the larger file **grid2.txt**. You will first need to create a two-dimensional list of the file contents using **readGrid** and store the resulting list in a variable. Then you can pass this variable to the other functions that you want to test. An example is shown on the next page.

```
>>> g = readGrid("grid1.txt")

>>> print(g)
[['c', 'a', 't', 'n', 'x', 'f'], ['k', 'c', 'o', 'r', 'e', 'r'], ['d', 'i', 'o', 'k', 'c', 'o'],
['l', 'r', 'a', 'w', 'i', 'g'], ['d', 'n', 'i', 'l', 'm', 'i'], ['s', 'o', 'w', 'b', 'l', 'p'],
['n', 'o', 'g', 'a', 'r', 'd']]

>>> printGrid(g)
c a t n x f
k c o r e r
d i o k c o
l r a w i g
d n i l m i
s o w b l p
n o g a r d

>>> print(numRows(g))
7

>>> print(numColumns(g))
6
```

2. We also need a function **nextPosition(*r, c, direction*)** to help us "navigate" the grid in a particular direction. This function will take a starting position, specified as the row number *r* and column number *c*, as input, along with a *direction* string such as "down" or "up right", and will return a pair of values representing the new row and column numbers after moving one grid position in the specified direction. For example, `nextPosition(1, 4, "down")` would return the values 2, 4, since moving down increases the row number by one and keeps the column number the same, and `nextPosition(2, 2, "up right")` would return 1, 3, since going up and to the right decreases the row and increases the column. This function, which we wrote together in class, is already included in **findwords.py**.

3. Write a function **inBounds(*row, col, grid*)** that takes two index numbers representing a row and column position as input and returns True or False indicating whether the position lies within the bounds of the given grid. Make sure your function works with grids of any size. Hint: use **numRows** and **numColumns** as helper functions. Test your function on the following examples. You can also run the autotester program by typing `test(inBounds)`.

   `inBounds(5, 0, g)` should return True

   `inBounds(7, 1, g)` should return False, since the maximum row number allowed in grid1 is 6

   `inBounds(5, -1, g)` should return False, since the minimum column number allowed in grid1 is 0

   `inBounds(-1, 2, g)` should return False, since the minimum row number allowed in grid1 is 0

4. Write a function **searchDirection(*word*, *row*, *col*, *direction*, *grid*)** that takes a word to find, a row and column starting position, a direction string such as `"down"`, and a grid, and searches for the word in the given direction only, from the given starting position. If the word is found, a message is printed and True is returned. Otherwise, nothing is printed and False is returned. Hint: use **nextPosition** as a helper function. Test your function on the following examples. You can also run the autotester by typing `test(searchDirection)`.

   `searchDirection("frog", 0, 5, "down", g)` should print "frog found at (0, 5) going down" and return True

   `searchDirection("frog", 0, 5, "right", g)` should return False, without printing anything

   `searchDirection("froggy", 0, 5, "down", g)` should return False, without printing anything

   `searchDirection("cow", 2, 4, "right", g)` should return False, without printing anything

5. Write a function **searchFromPosition(*word*, *row*, *col, grid*)** that takes a word to find, a row and column position, and a grid, and searches for the word in all possible directions from the given starting position. If the word is found in any direction, a message is printed. If the word occurs multiple times, you only need to report the first occurrence, although you can report them all if you wish. The value True or False should be returned from the function, indicating whether or not the word was found. Hint: use **searchDirection** as a helper function, and a for-loop to cycle through all the directions in the directionNames list. Test your function on some examples as shown below:

```
>>> searchFromPosition("frog", 0, 5, g)
frog found at (0, 5) going down
True

>>> searchFromPosition("cow", 1, 1, g)
cow found at (1, 1) going down right
True

>>> searchFromPosition("cow", 0, 0, g)
False
```

6. Write a function **searchGrid(*word, grid*)** that takes a word to find and a grid, and searches for the word from all possible starting positions in all possible directions. If the word occurs multiple times, you only need to report the first occurrence, although you can report them all if you wish. The value True or False should be returned from the function, indicating whether or not the word was found. Hint: use **searchFromPosition** as a helper function.

```
>>> searchGrid("cow", g)
cow found at (1, 1) going down right
True

>>> searchGrid("frog", g)
frog found at (0, 5) going down
True

>>> searchGrid("puppy", g)
False
```

7. Finally, write the function **findwords(*filename*)**, which will be the main program. This function should create the letter grid from the specified file (using **readGrid**), print it out (using **printGrid**), then repeatedly prompt the user for a word to search for on the grid. If the word does not appear anywhere, the program should print a message saying that the word was not found. When the user types in `quit`, the program should terminate. Test your program as shown in the example session on the first page of the lab worksheet. You can also run all of your functions through the autotester by typing `testall()`.

8. Once you are confident that your program completely works, try it out on the larger **grid2.txt**, shown below, which contains twenty different flavors of ice cream. Can you find them all? There may even be a lion or dragon (or tiger) hiding in there as well...

```
o i h c a t s i p b b n g a r d
o r l v c n a i l t a t y n e u
c h e a a y a u a t m u r o l c
r h c c e n e c i u t n r m a p
e r o a i b i l e u l l e a l i
g o c c e r o l n p b a h n m h
i c n r o p o l l a r w c n o c
t k r i a l e c n a u e e i n t
c y u e i z a a i o r l t c d n
o r n o a c n t e l g p t t l i
f o n h e a k c e i b a b i u m
f a t a p u m p k i n m r g g b
e d s t r a w b e r r y t d r e
e t h g u o d e i k o o c r r a
```