COP 3502: Program Word Search - Suggested Functional Breakdown

The Word Search Program can be broken down into a few smaller programs. If you are having trouble with it, I suggest you write these programs separately and then work on figuring out how to turn the programs into functions and stitching together the appropriate functions calls to solve the whole program.

Problem 1: Reading in the Dictionary

Write a program that opens the file "dictionary.txt", which stores the dictionary in the format given in the assignment and read the dictionary into an array of strings. Your program should then print the dictionary in reverse order (to double check that it was really read in correctly.)

Problem 2: Binary Searching for a string in the Dictionary

Add the code for this program to the end of the previous one. After reading in the dictionary, ask the user to enter a word and then have your program print out if the word is in the dictionary or not. Make sure your program implements a binary search and not a linear search. You should write a function for your binary search with the function signature shown below:

```
int inDictionary(char** words, int numWords, char* searchWord);
```

The function should return true if searchWord appears in the array words and false otherwise. numWords must equal the number of strings in the array words.

Problem 3: Write a function which finds a string of a particular length in a particular direction from a particular start point in a word search grid. The function should return that string, null terminated.

For this problem, please use the following DX, DY arrays:

```
const int DX[] = {-1,-1,-1,0,0,1,1,1};
const int DY[] = {-1,0,1,-1,1,-1,0,1};
```

Your function should take in the following parameters:

1) char** grid for the word search grid
2) int numRows for the number of rows in the word search grid
3) int numCols for the number of columns in the word search grid
4) int x, for the value of the row number of the starting spot
5) int y, for the value of the column number of the starting spot
6) int len, for the length of the string desired
7) int dir, for the integer direction (based on DX, DY) that we want to move to build our word.

Your function should return a char*, a single string that is dynamically allocated (so it can persist after the function completes.)

For example, if the grid inputted was:

```
twttdad
ghsadfb
yaedcad
utbrdtf
abcdeth
```

Then here are a few function calls and what they should return:

Function Call	Return String
`formString(grid, 5, 7, 3, 2, 3, 0)`	`"bag"`
`formString(grid, 5, 7, 3, 2, 4, 1)`	`"best"`
`formString(grid, 5, 7, 3, 4, 3, 2)`	`"dab"`
`formString(grid, 5, 7, 4, 5, 3, 3)`	`"ted"`
`formString(grid, 5, 7, 1, 2, 3, 4)`	`"sad"`
`formString(grid, 5, 7, 1, 6, 3, 5)`	`"bad"`
`formString(grid, 5, 7, 1, 1, 3, 6)`	`"hat"`
`formString(grid, 5, 7, 0, 0, 5, 7)`	`"there"`

This task requires no dictionary. But one key element is that it requires careful use of the DX, DY array, dynamic memory allocation for the single string to be returned and placing a null character ('\0') at the end of the string before returning it. Your program should read the grid from standard input, a starting position, a length and a direction, and print out the appropriate string to the screen.

Problem 4: Print out EVERY string of EVERY LENGTH (4-19) from a specified starting position in a grid.

This program should simply call the function written for the third task multiple times. Your program should ask the user just to enter a starting location in the grid and then should print out every string of a valid length that can be formed searching in any direction from that position. Be very careful not to do array out of bounds accesses on the grid. (The function you write for this task should take in 5 parameters: the grid, its dimensions and the start location for each string.)

For example, the function call `printWords(grid, 5, 7, 0, 0)` should print the following for the grid shown above:

```
twtt
twttd
twttda
twttdad
tgyu
tgyua
ther
there
```