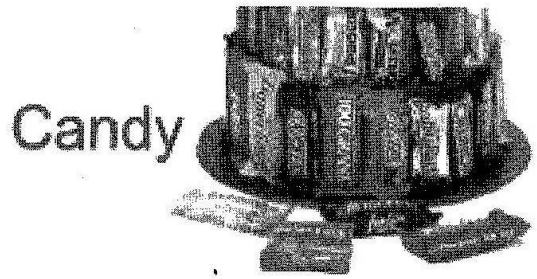
2015 Fall Computer Science I Program #2: Word Search
Please consult Webcourse for the due date/time

A common game in newspapers is a word search. You are given a grid of letters and must find words that appear in the grid. If you can choose the starting letter of the word within the grid, and then go in one of the 8 possible directions (up, diagonal up and right, right, diagonal down and right, down, diagonal down and left, left, diagonal up and left) and spell out the word exactly as you go by each letter, then the word is in the grid. Here is an example of a partially solved word search:



Almond Joy	~~Kiss~~	Reeses
Baby Ruth	KitKat	~~Skittles~~
Butterfinger	Lollipop	~~Smarties~~
Crunch	~~Nerds~~	Twix
Fun Dip	Milkyway	~~Twizzlers~~
Hershey Bar	~~Payday~~	
Jolly Rancher	Pixy Stix	

M	A	L	M	O	N	D	J	O	Y	W	M	A
I	L	O	L	L	I	P	O	P	Q	Z	I	L
L	P	S	K	I	T	T	L	E	S	S	L	X
K	F	U	N	D	I	P	L	Y	X	E	K	B
Y	A	D	Y	A	P	Q	Y	S	I	I	D	Z
W	R	E	G	N	I	F	R	E	T	T	U	B
A	E	T	H	S	M	E	A	Q	S	R	D	A
Y	E	A	W	T	L	W	N	C	Y	A	S	B
M	S	K	O	Z	Q	E	C	R	X	M	K	Y
X	E	T	Z	O	R	W	H	U	I	S	I	R
I	S	I	L	D	B	V	E	N	P	V	S	U
W	W	K	S	Z	X	G	R	C	Z	N	S	T
T	B	P	R	A	B	Y	E	H	S	R	E	H

In a typical word search, you are given a list of words, each of which can be found in the grid. In this example, we can see that the word "TWIZZLERS" appears starting in the last row and first column, going along the up and left diagonal. We can find "PAYDAY" starting on the 5th row and 6th column, going left.

Obviously, solving a word search is very time consuming. You've decided to use your programming skills to very quickly solve word searches so that you can impress your friends with your "vision"!

Programming Tip: DX and DY arrays

In many programming applications, one must start in some location on a two dimensional grid and then they must move in one of several directions. In terms of coding, it would be nice if one could simply "loop" through each of the possible directions in a seamless manner. The best way to do this is to define constants that store each of the possible directions of movement. For this program, these constants look like the following:

```
const int DX_SIZE = 8;
const int DX[] = {-1,-1,-1,0,0,1,1,1};
const int DY[] = {-1,0,1,-1,1,-1,0,1};
```

These should be defined before main, where constants are typically defined.

Now, if we are currently at a spot (x,y), we can move to each of the adjacent locations as follows:

```
for (i=0; i<DX_SIZE; i++) {
    int nextX = x + DX[i];
    int nextY = y + DY[i];
    // (nextx, nexty) represents the adjacent location in
    // direction i.
}
```

For this assignment adjustments will have to be made to this framework, but the overarching idea is very, very useful as it avoids an if statement with 8 branches which is very, very error prone. In this framework, the logic has to be written once and the DX and DY arrays have to be double checked, that's it!

Dictionary Information and Binary Search

Before the program processes any word search grids, your program should load in the dictionary from the file, dictionary.txt. The first line of this file will have a single positive integer, *n*, representing the number of words in the dictionary. The words follow on the next *n* lines, one word per line. Each word will only contain lowercase letters and be in between 4 and 19 letters long, inclusive.

It is recommended that you use a binary search to see if a particular string is in the dictionary. If you use a linear search, your program will not earn full credit.

The Problem
Given an input dictionary read in from a file to use for all test cases, and several word search grids, identify all words from the dictionary that appear in each word search grid.

The Input (of word search grids, to be read in from standard input)
The first line of the input file will contain a single positive integer, *c (c ≤ 100)*, representing the number of input cases. The input cases follow, one per line. The first line of each input case will contain two positive integers, *r (4 ≤ r ≤ 300)*, and *c (4 ≤ c ≤ 300)*, representing the number of rows and columns, respectively, in the word search grid. The following *r* lines will contain strings of exactly c characters, all of which will be lowercase letters for each row of the word search grid for that input case.

The Output
For each case, output a header with the following format:

```
Words Found Grid #k:
```

where k is the grid number starting with 1. (**Please pay attention to the capitalization above, the pound sign and the colon. You will lose a small amount of credit if you don't use this exact format.)**

Output all the words from the dictionary found in the grid on the following lines, one per line. You may output these in any order and you may output the same word more than once, **but you should not output any string that isn't in the dictionary, or output any valid word that does not appear anywhere in the grid.** I will assign a small amount of extra credit if you can only output each valid word once and another little bit of extra credit if you output these words in alphabetical order. You can only get the extra credit if your solution is 100% correct.

Sample Input

```
2
4 4
syrt
gtrp
faaq
pmrc
5 6
swingh
abcdef
rettel
ayzgfd
cmtydh
```

Sample Output (using posted dicationary.txt)

```
Words Found Grid #1:
trap
part
cats
Words Found Grid #2:
swing
wing
letter
```

Specification Details
You must use dynamic memory allocation to store the input dictionary and each individual word search grid. You must free your memory appropriately. You must read the input dictionary from the file dictionary.txt, which will be posted online.

Deliverables

Please turn in a single source file, *wordsearch.c*, with your solution to this problem via Webcourses before the due date/time for the assignment. Make sure that your program reads the puzzles from standard in and outputs all information to standard out, as previously shown in lab. Note that the dictionary will be read in from a file.