Problem Set 3: Maze and Cipher

Please send back via NYU Classes

- A zip archive named as
  `PS03_<your name as FirstLast>.zip`

containing the C code files that implement all aspects of all problems.

Total points: 100

Problem 1: Maze using Recursion (50 Points)

`maze_recursion.c`

You are given the files **maze.c, maze.h** and **maze.txt**. It is your task to create **maze_recursion.c** which contains function **solveMaze()** that recursively solves the maze. The structure of **maze_recursion.c** should be:

```
#include <stdbool.h>
#include "maze.h"

bool solveMaze(int i, int j) {

    // (your code here)
}
```

You must write the function **solveMaze(i, j),** in such a way that it can call itself recursively to solve the maze. The maze is file **maze.txt,** which is read into the global character array **grid[i][j]**. This is done in **maze.c**. You position is (i, j) where i and j are declared in **main()** and are the array indices in **grid[i][j]**. Variable i is the row index, or North/South, and variable j is the column index or East/West.

To start (that is, on the first call to your function):

- The position (i, j) is at ‘S’ in the maze.

On each call to **solveMaze():**

1. If maze grid character at the current position is ‘G’ then you found the goal and are done, so return true
2. If maze grid character at the current position is:
    - A Wall, then return false
    - The “already visited” character (i.e. ‘.’), then return false
    - Outside of maze, then return false
3. If none of the conditions in (2) are false, then drop a breadcrumb (i.e. set the grid character to ‘.’) at the current position to indicate that you have visited this position
4. Display maze grid by calling **display()**

5. It may be helpful to define and initialize the four points of the compass relative to current position. Note that the maze grid has origin (0,0) at upper left corner (so N is -1, S is +1, etc.).
   - `Ni = i+1;        Nj = j;`
   - `Si = i-1;        Sj = j;`
   - `Ei = i;          Ej = j-1;`
   - `Wi = i;          Wj = j+1;`
6. For each of the N, S, E, W integer pairs, call the function **solveMaze()** and
   - If return value is true:
     - Set grid character at the current position (i, j) to the return path indicator character '*'. This is the "backtrace" path that is the implicit solution provided by the recursion process.
     - Display maze grid by calling display()
     - Return true
   - If return is false:
     - Call **solveMaze()** for the next point of the compass
   - Repeat for remaining points of the compass
   - If all four calls return false:
     - return false

Note: because of the "early exit" properties of the if() statement having multiple conditional expressions, step (6) above can be structured in this way:

```
if (solveMaze(North) || solveMaze(South) ||
        solveMaze(East) || solveMaze(West)) {
      // (statements if true)
    }
    else {
        return false;
    }
```

Problem 2: Vigenère's cipher (50 points)

`vigenere.c`

Start with the instructor-supplied file **vigenere.c**. Use this framework to create a program that encrypts messages using using Vigenère's cipher. Vigenère's cipher is a keyword cipher with a different shift at each position in the text; the value of the shift is determined by a repeating keyword.

For example, suppose the plaintext to be encrypted is:

```
WATERTHEPLANTS
```

The person sending the message chooses a keyword and repeats is until it matches the length of the plaintext. In the case of the keyword TURNIP it would be:

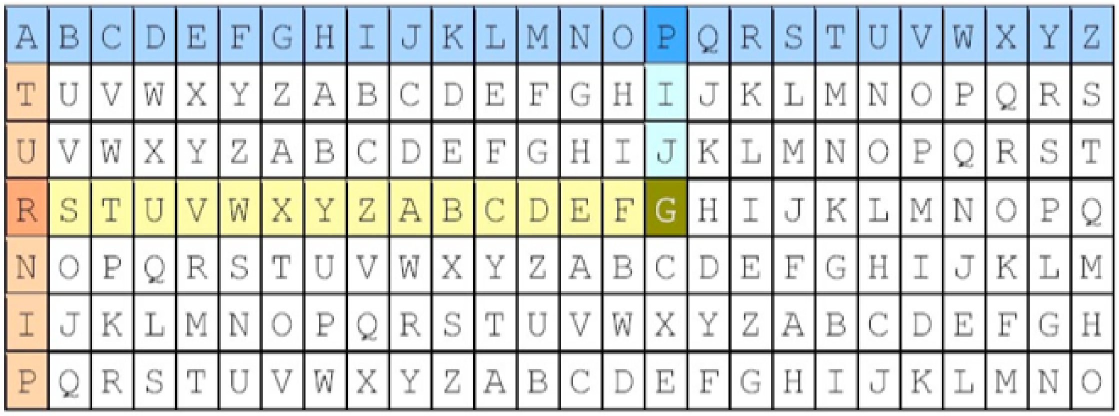
```
TURNIPTURNIPTU
```

Plaintext:	`WATERTHEPLANTS`
Key:	`TURNIP`
Ciphertext:	`PUKRZIAYGYICMM`

For example, here is a visualization of how one of the characters is encoded:

```
WATERTHEPLANTS
TURNIPTURNIPTU
PUKRZIAYGYICMM
```

Letters used in the plaintext message are in the top row, and the keyword letters are in the leftmost column:



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

Programmatically, the ciphertext letter is P' + ('R' - 'A'). If the result is past Z then you have to wrap back to the letter A. This will work for both upper and lower case letters.

Your program must have command-line arguments as follows:

```
./vigenere [-e|-d] keyWord input_file.txt output_file.txt
```

where:

- **-e** is an optional argument indicates the program performs in "encode" or encryption mode. This is the default mode.
- **-d** is an optional argument indicates the program performs in "decode" or decryption mode.
- **keyWord** is a word. It must be all alpha characters.
- **input_file.txt** is a file containing a ASCII text. If in "encode" mode, then this will be the clear text, otherwise it will be ciphertext.
- **output_file.txt** is the output file. If in "encode" mode, then this will be the ciphertext, otherwise it will be the clear text.

If your program is executed without any command-line arguments, or with the incorrect number of command-line arguments, it should print a "usage" message and exit, for example:

```
Usage: ./vigenere [-e|-d] keyWord input_file.tx output_file.txt
where: -e indicates "encode" or encryption mode (default)
       -d indicates "decode" or decryption mode
       keyWord is a word (all alpha characters)
```

As a first step, your program should parse the command line:

- Determine if **–e** or **–d** is present and set a mode variable accordingly
- Parse and save **keyWord**
- Open the input file. Print an error with diagnostic message if file cannot be opened.
- Open the output file Print an error with diagnostic message if file cannot be opened.

For each line in the input file, it should:

- Read a line from the input file
- Encrypt or decrypt the line of text, depending on the mode (don't forget to put a NULL character at the end of the output characters to make it a proper C-language character string!
- Write the line to the output file

Finally:

- Close all files and exit

Here are some additional constraints:

- Your program must preserve case: capitalized letters, though rotated, must remain capitalized letters; lowercase letters, though rotated, must remain lowercase letters.
- Your program must not alter any non-alpha characters (e.g. numerals or punctuation).

Use your program to encrypt the clear text file **clear.txt**, whose contents is shown here:

```
The quick brown fox jumped over 1000 lazy dogs.
Open the crate but don't break the glass.
Add the sum of 5 and 6 to the product of 7, 8 and 9.
```

When you run your program as

```
./ vigenere -e hello clear.txt output_file.txt
```

the contents of the output file should be:

```
Alp bipgv mfvay qce nfxdlh zgsy 1000 plkm ksrd.
Vtpy hoi ncoai mfh ksy'e pyilv hoi rwozw.
Hho evl wfx cm 5 eyo 6 hv xsp dysofqa sq 7, 8 lbk 9.
```