CS1132 Spring 2013 Assignment 2

Adhere to the Code of Academic Integrity. You may discuss background issues and general solution strategies with others and seek help from course staff, but the homework you submit must be the work of just you. When submitting your assignment, be careful to follow the instructions summarized in Section 4 of this document.

Note: In this last homework in the course, you will design the solution to the final problem posed. You need to think carefully about how to decompose the problem into subproblems or tasks—use subfunctions. The focus here is for you to turn a problem statement written in English into a solution written as a Matlab program. Read carefully and take some time to think about the design—don't just jump into coding immediately.

Do not use the **`break`** or **`continue`** statement in any homework or test in CS1132. Note that Problem 1 has additional restrictions on the use of built-in functions.

1. Cryptography Fun!

Just several decades ago, cryptography existed solely in the domain of national governments and large corporations. However, the advent of the Internet and the wide availability of computers have now made cryptography an indispensable part of everyday digital communication. In this part of the assignment, you will implement the classical *book cipher*. The book cipher predates modern cryptography and provides an insightful look at some commonly used cryptography techniques. You will implement the book cipher as Matlab functions that can be used to encrypt (and subsequently decrypt) messages.

In cryptography, the *key* is a piece of information—a word, a number, *a book*—that is used to encrypt (hide) and later decrypt (reveal) a message. The key of the book cipher is generated from a book or some other piece of text. Such a cipher requires both sender and recipient of the encrypted message to agree beforehand on a specific piece of text to be used as the key. The book cipher traditionally uses widely available publications such as the dictionary.

In this implementation of the book cipher, individual letters of the message are replaced by the positions of the words `that` start with those letters. To illustrate, let us encode the message `APPALACHIAN ZOO`. We call the original message *plaintext* and the encoded (encrypted) version *ciphertext*. We will use as the key the following sentence from the book *Beyond Good and Evil* by Friedrich Nietzsche:

It is some basic certainty which the noble soul has about itself, something which does not allow itself to be sought out or found or perhaps even to be lost. The noble soul has reverence for itself.

To encode the first letter of the plaintext, `A`, we look for the first word in the key that starts with 'A' or 'a' – it is the 11th word, 'about'. As such, the first letter of the plaintext is encoded as 11. The second letter `P` corresponds to the 26th word, 'perhaps'. This letter is thus encoded as 26. This is repeated for all the remaining letters of the plaintext, making sure that each position can only be used once. That is to say, the second occurrence of the letter `A` will correspond to the word 'allow' at position 17 and will be encoded as 17. Any non-letter in the plaintext is ignored, so the plaintext `APPALACHIAN ZOO` is the same as `APPALACHIANZOO`. Upper and lower case letters are considered the same.

Note that if a letter in the plaintext does not have a corresponding word in the key (e.g., no word in the given key starts with 'Z'), we simply ignore that letter when encrypting the plaintext. Similarly, if we run out of words in the key that start with a certain letter, we ignore those letters in the plaintext that do not have corresponding words in the key. For example, the key above contains only two words that start with the letter 'A' while the message `APPALACHIAN ZOO` has four A's. In this case, the last two A's will be ignored.

The ciphertext for the above example is `11 26 17 30 5 10 1 8 22 23`. However, this example can be considered a poor use of the book cipher as several letters have to be omitted in the encryption. The decrypted text will not give us the complete plaintext; instead it will simply return `APALCHINOO`. It is the responsibility of the users of the encryption scheme to avoid low frequency letters and excessive repetition of letters. It is also important to ensure that the key used for the cipher is long enough such that it contains most, if not all, of the frequently used letters.

You will write several functions to implement the book cipher as described above. The cipher key will be a particular chapter of a book instead of a whole book. We provide the ASCII text file **`alice.txt`**, which is the book *Alice's Adventures in Wonderland* by Lewis Carroll. Your implemented cipher should work with any ASCII book file that has the same chapter format as **`alice.txt`**, which will be described below.

Implement these two functions for encryption and decryption:

```
function outStr = encrypt(inStr, keyFile, chapter)
% An implementation of the book cipher for encryption.
% inStr: plaintext to be encrypted, a string
% keyFile: filename of the book (in ASCII format) to be used as the key
% chapter: the number of the chapter to be used as the key, an integer
% outStr: ciphertext from encrypting inStr, a string


function outStr = decrypt(inStr, keyFile, chapter)
% An implementation of the book cipher for decryption.
% inStr: ciphertext to be decrypted, a string
% keyFile: filename of the book (in ASCII format) to be used as the key
% chapter: the number of the chapter to be used as the key, an integer
% outStr: plaintext from decrypting inStr, a string
```

Note that the ciphertext from **`encrypt`** is a string of digits, not a vector of numbers. To assist your encryption and/or decryption, implement and make effective use of the following function:

```
function ca = str2cellarray(str)
% Convert a string to a cell array of words (strings).
% str: a string storing only letters and spaces.  str may start (end) with
%   a letter or one or more spaces.
% ca: a 1-d cell array storing each word in a cell; words are delimited by
%   one or more spaces.
% Example:  If str is  'Look for     a space followed   by a letter  '
%          then ca is  {'Look','for','a','space','followed','by','a','letter'}
```

You may write additional functions and submit a zip file containing all the functions you implement for this problem.

Below is a list of useful built-in functions for handling characters, strings, and files. Use **only** these built-in functions for handling characters, strings, and files. You may not need all of them. You can of course still use general built-in functions not related specifically to strings and files, such as **`length`**, **`zeros`**, **`rem`**, …, etc.

`fopen`, **`feof`**, **`fgetl`**, **`fclose`** (as explained in Module 2)

`strcmp`, **`upper`**, **`isspace`**, **`isletter`** (as explained in Module 2)

`num2str`: convert a numeric value to a string or convert a numeric vector to a string. E.g., the expression **`[ '[CHAPTER] ' num2str(3) ]`** gives the string **`'[CHAPTER] 3'`**

Note: There is a space here.

`str2num`: convert a string of digits (and spaces) to a vector of numbers. E.g., the expression
`str2num('35 4   9')` gives the length 3 numeric vector **`[35 4 9]`**

`deblank`: remove trailing spaces from a string. E.g., the expression
`deblank('this  word to next word  ')` gives the string **`'this  word to next word'`**

Format of the key bookfile

Use a text editor to take a look at the given file **`alice.txt`** (use the Matlab editor by double-clicking on the file in Matlab's *Current Folder* pane). The bookfile stores ASCII characters only. Each chapter of the book starts on a new line beginning with the string **`'[CHAPTER] ',`**followed by the chapter digit(s), followed by the chapter title. The key text is all the characters in a chapter *excluding* the entire **`'[CHAPTER] '`** line. This means that the chapter title is *not* part of the key. There is no appendix to the book, i.e., the book ends with the last numbered chapter. For example, *Alice's Adventures in Wonderland* has ten chapters, so last word of chapter 10 is also the last word in the bookfile.

Working with the key

Although you have to read the bookfile line by line, do not store the whole book in memory. To be space efficient, your functions should store (keep) only the chapter that is being used as the key.

Along with spaces, use any non-letter characters (e.g., digits and punctuation marks) as delimiters for words in the key, even if it results in tokens that are not real words. For example, the phrase **`how late it's getting!`** gives five words of the key: **`'how' 'late' 'it' 's' 'getting'`**. Here, although 's' is not a proper English word, it is considered a word in the key. HINT: when you work with the key, first turn all the chapter text into one long string and convert to upper case. Then replace each non-letter character with a space, which can be done using vectorized code in one statement! After that, your **`str2cellarray`** function becomes handy.

2. Battleship

In this part of the assignment, you will implement a simplified, one-player version of the game Battleship.

The game is set up as an $n \times n$ grid with several ships randomly distributed on the grid with equal probability (for the purpose of this exercise, we can assume that n is greater than 5). For example, the traditional Battleship board game uses the following types of ship:

Type of Ship	*Size*
Aircraft Carrier	5
Battleship	4
Submarine	3
Destroyer	3
Patrol Boat	2

Each type of ship has a corresponding size which refers to the number of squares it will occupy on the grid. In a game, the ships are randomly chosen to be either all horizontally laid out or all vertically laid out. A ship cannot be laid diagonally on the grid. The ships should be randomly distributed on the playing grid without any overlap. You can assume that the user will only input valid parameters, i.e., the sum of sizes should be much less than $n \times n$.

After the playing grid (blue) has been set up, the player plays by guessing the squares that the ships occupy. The player strikes a particular target square by clicking on it. The game then responds by telling the player if the square was a hit or a miss, i.e., if the square is occupied by part of a ship or not. When all parts of a ship have been hit, it is sunk. The objective of the game is to sink all the ships with the least number of moves (clicks).

As the player clicks on the squares, the game should mark each square:

- If the square is a hit (i.e., part of a ship occupies that square), color the square red.
- If the square is a miss, color the square white.
- If the square has been clicked on before, display a message asking the player to click on another square.

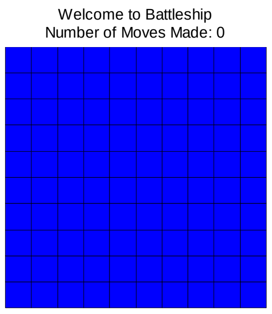
The game should also display appropriate messages notifying the player whether that move was a hit, a miss or a click on a square that has already been clicked on. Additionally, the game should display an appropriate message when a ship has been sunk (i.e., all the squares occupied by a ship have been hit), including the type of ship. All messages should be shown in the figure window using the title area of the plot.

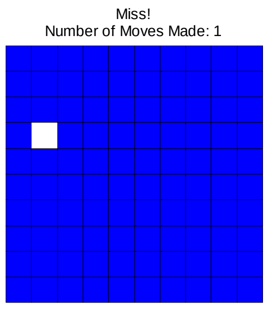
There should be a counter that keeps track of the number of valid moves the player has made; it should be shown on the figure window. A valid move refers to a hit or a miss (i.e., not a click outside the playing grid or a square that has already been clicked on). The game ends when the user has sunk all the ships; the game should display a congratulatory message.

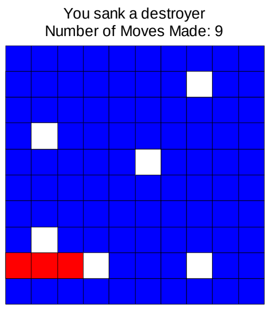
Write a function battleship to implement the game as described above. Write subfunctions to build this game in a modular fashion. Taking the time to decompose the problem and implement subfunctions will end up saving you time overall!

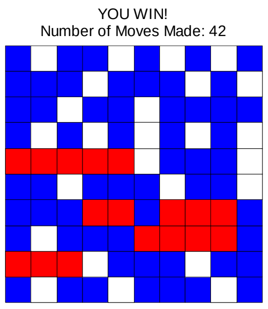
```
function battleship(n, ships)
% A one-player game of Battleship on an n-by-n grid
% n:  number of squares on each side of the playing field
% ships:  m-by-2 cell array storing information of m ships
%   ships{s,1} is a string naming the type of the sth ship, s=1:m
%   ships{s,2} is the integer size of the sth ship, s=1:m
%   E.g., ships={'aircraft carrier', 5; 'battleship', 4; 'submarine', 3}
```

Below are four snapshots of a game.









3 Submission instructions

1. Upload files **`encryption.zip`** (all functions you implemented for Problem 1) and **`battleship.m`** to CMS in the submission area corresponding to Assignment 1a in CMS.
2. *After grading:* If you resubmit the assignment, upload your corrected files and be sure to select the **Regrade Request** option.