CS1112 Fall 2015 Project 5 due Thursday 11/5 at 11pm

You must work either on your own or with one partner. If you work with a partner you must first register as a group in CMS and then submit your work as a group. *Adhere to the Code of Academic Integrity.* For a group, "you" below refers to "your group." You may discuss background issues and general strategies with others, but the work that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions with others. It is not ok for you to see or hear another student's code and it is certainly not ok to copy code from another person or from published/Internet sources. If you feel that you cannot complete the assignment on you own, please seek help from the course staff.

Objectives

Completing this project will solidify your understanding of character array and cell array and give you practice with processing text data files.

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 made cryptography an indispensable part of everyday digital communication. In this project, 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 (hide), and subsequently decrypt (reveal), messages.

In cryptography, the *key* is a piece of information—a word, a number, *a book*—that is used to encrypt and later decrypt a message. The key of the book cipher is generated from a book or some other piece of text. Such a cipher requires both the sender and the 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 our 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*. In this example 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 of the plaintext, '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, 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 decryption 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. The user should also 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.

1.1 Allowed built-in functions

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. Do *not* use built-in function `find`.

- File handling: `fopen`, `feof`, `fgetl`, `fclose`
- Characters and strings: `strcmp`, `upper`, `isletter`, `isspace`
- `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'`
- `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]`

The next four sections describe the four functions that you will write for this project; read them next. *Before* you start writing code, however, read the last section of this project, which suggests a work flow that facilitates program development for the entire project. Those are important ideas to learn.

1.2 Building a key from a string

Implement the following function as specified:

```
function ca = makeKeyArray(str)
% Convert a string to a cell array of words (strings) in capital letters.
% str: a 1-d array of characters
% ca: a 1-d cell array storing each word of str in a cell; words are
%   delimited by one or more non-letter characters.
% Example: If str is  'Check adjacent  eLements 4special pat-terns  '
%         then ca is {'CHECK','ADJACENT','ELEMENTS','SPECIAL','PAT','TERNS'}
```

Note that any non-letter characters (e.g., digits, spaces, and punctuation marks) are treated 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.

1.3 Getting a chapter from the book

Implement the following function as specified:

```
function key = getChapter(keyFile, chapter)
% key is a string containing all the text in the specified chapter of keyFile.
% 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
% Assume that the file is not empty and that the specified chapter exists
% and is not empty
```

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 contains 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 the last word of chapter 10 is also the last word in the bookfile. The beginning of the bookfile contains the text of bibliographic information—the file does not begin with chapter 1. If another book is used with our implementation of the book cipher, it will have the same format as described above.

1.4 Encrypting a message

Implement the following function as specified:

```
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
```

Note that the ciphertext from `encrypt` is a string of digits, not a vector of numbers. This function should make effective use of the two functions implemented above (to get the chapter text and to make the key cell array). Additionally, implement subfunctions to simplify the code in function `encrypt`. For example, a subfunction that searches a cell array for a string that begins with a user-specified letter would be very handy.

1.5 Decrypting a message

Implement the following function as specified:

```
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 incoming ciphertext is a string of digits, not a vector of numbers. This function should make effective use of functions `getChapter` and `makeKeyArray`. Since the key uses only capital letters, the returned plaintext will be in capital letters as well. Furthermore, any spaces and punctuation marks (non-letters) of the original message are excluded during the encryption process and therefore cannot be recovered in the decryption process.

1.6 Program development

Now that you have read about the four functions that you need to implement, how do you decide which one to work on first? Normally, beginning with the function that doesn't depend on any other function would be a good choice. Choosing the simplest function as the starting point is also a good choice!

A. Start with function `makeKeyArray`. Although the key is made from the text of a book chapter, the function itself does not require the completion of the `getChapter` function. The input parameter of `makeKeyArray` is simply a string, so you can use any string for testing—you do not need to have the chapter text read into memory first. The string given in the function comments is a test case that you can use. You should come up with other test cases for yourself as well.

B. We suggest working on function `encrypt` next because (1) it is directly related to `makeKeyArray` which you have completed, and (2) function `getChapter` requires the use of several built-in functions that you will need to review—save that task for last. Of course, you know that `encrypt` is *supposed to* call function `getChapter` to get the chapter text in order to make the key cell array. For *program development*, however,

it is better to use a much, much shorter string than a whole chapter to develop and test your `encrypt` function!
The simplest thing to do is to write a "dummy version" of `getChapter` for now: write just one statement as the body of function `getChapter` to assign a short string to the return parameter, ignoring all the input parameters. One of strings that you used for testing in step **A** above is perfect for this.
Now back in `encrypt`, you can call `getChapter` and store the returned string. Use this string to build the key cell array; then write the code for encryption using the key cell array. Test your `encrypt` function by encrypting several short messages and confirming by hand that the encryption works for each message.

C. Work on `decrypt` next, when `encrypt` is still fresh on your mind. Both of these functions make use of the key cell array so it's good to work on them one after the other. Again, make use of the dummy `getChapter` function. Test thoroughly!

D. Now work on `getChapter` for real. Start by reviewing from the lecture notes and examples the built-in functions `fopen`, `feof`, `fgetl`, and `fclose`. Then replace the dummy code you wrote previously with the real thing that actually reads the book file and stores the text of the specified chapter in the return parameter as a string. For testing, be sure to check that your function returns all the text of the specified chapter (excluding the chapter heading) in a string. Did you check the first chapter, the last chapter, and some middle chapter?

E. Now that you have completed `getChapter` for real, test `encrypt` and `decrypt` again! Have fun encypting secret messages!

Submit your files `encrypt.m`, `decrypt.m`, `makeKeyArray.m`, and `getChapter.m` on CMS.