CSCI 145 Week 4
Lab 3

Objectives for this Lab

1. Practice in the use of 2-dimensional arrays.
2. Practice in the use of the command line parameters.

Task Description

Your task is to write a program to perform data encryption with a grid cipher. The grid cipher is a form of permutation cipher, in which none of the characters in the plain-text message is changed but the characters are shuffled into another order.

The Grid Cipher

Encryption with the grid cipher (in its simplest form) is performed by writing the plain-text characters (the original text before encryption) into a rectangular grid, row by row, and then reading the characters from the grid column by column. The width of the grid is the encryption key.

For example, given the plain-text message "FRIENDSROMANSCOUNTRYMENLENDMEYOUREARS" and a grid width 6, the plain-text characters are arranged in a grid as follows:

F	R	I	E	N	D
S	R	O	M	A	N
S	C	O	U	N	T
R	Y	M	E	N	L
E	N	D	M	E	Y
O	U	R	E	A	R
S					

The cipher text (the text after encryption) is read from the grid column by column giving

"FSSREOSRRCYNUIOOMDREMUEMENANNEADNTLYR".

In its more general form, the grid cipher transposes the columns. Instead of reading the columns from left to right, it reads them in some order determined by a more complicated encryption key. However, for this exercise, you only need to handle the simplest form of grid cipher, with the columns read from left to right.

Cryptanalysis of a Grid Cipher

*Note: This is for your information only. Your program does **not** have to do this. This is what GridWriter.java generates.*

Since the number of possible encryption keys is small, the easiest method of cryptanalysis is brute force, in which we just try various grid widths to see if any of them result in plain-text that looks like English. For example, here is the output of the GridWriter program for the encrypted text shown above, with grid widths 2 through 8:

```
Grid width 2: "FESMSUREEMOESNRARNCNYENAUDINOTOLMYDRR"
Grid width 3: "FINSOASONRMNEDEORASEDRMNRUTCELYMYNERU"
Grid width 4: "FYENSNMESUUARIEDEOMNOOETSMNLRDAYRRNRC"
Grid width 5: "FRMMASCDEDSYRNNRNEATEUMNLOIUNYSOEERRO"
Grid width 6: "FRIENDSROMANSCOUNTRYMENLENDMEYOUREARS"
Grid width 7: "FSUDENNSRIRMNTSROEEELRCOMNAYEYMUADRON"
Grid width 8: "FOYOMNETSSNMUAALSRUDENDYRRIRMNNRECOEE"
```

It is obvious from this output that the grid width was 6.

Requirements

1. Your program must get the grid width and the name of the input file from the command line. The command line arguments are:

   - The grid width to be used for encryption.
   - The name of the input file.

   Both of the command-line arguments are required. If too few, or too many, or incorrect command line arguments are given or if the input file cannot be read, the program is to display an appropriate message and terminate without any further processing.

2. Your program will get a single line from the input file and will output a single encrypted line using the grid width specified for encryption.

3. Your program is to use arrays declared as exactly the correct size for the input data. You should get the line of input as a `String` and then use the `length()` method on `String`s to determine the exact number of characters that were input. Assuming you use the `nextLine()` method on a `Scanner` to get the input, you do not have to account for the end-of-line characters in your program. `Scanner` will remove these characters before returning. See `GridWriter.java` for an example of how to do this.

4. You cannot assume that the number of characters in the input data is an exact multiple of the grid width. Therefore, as demonstrated in the example above, the last row of the grid might be incomplete. Your program will have to account for this.

Notes

1. Remember that command line arguments are `String`s. You can obtain the `int` value of a string by using the expression:

   ```
   Integer.parseInt(aString)
   ```

   If `aString` cannot be converted to an `int`, this expression will raise a `NumberFormatException`. You will have to use a

```
try {
   …
} catch (NumberFormatException ex){
   …
}
```

block to catch the exception and issue an appropriate error message. The GridWriter program has an example of how to do this.

2. You can test to see if a file can be read using the `canRead()` method of a `File` object. Again, see the GridWriter program.
3. To assist you in writing your program, the decryption program, `GridWriter.java`, used to generate the example cryptanalysis, shown above, is provided in `lab3.zip`, on Canvas, along with several examples of plain-text messages and the corresponding encrypted messages. Since I did not provide you with the grid width used to generate the cipher text files, you should compile and use the gridwriter program to determine the correct width. Copying and modifying code from `GridWriter.java` will help you complete your lab quickly.

References

1. Chapter 7, Arrays, in the textbook. Section 7.5 discusses multi-dimensional arrays.
2. Section 7.4, pages 484-485 has a brief discussion of command line arguments.
3. This page: http://docs.oracle.com/javase/tutorial/essential/environment/cmdLineArgs.html has a brief tutorial on command line arguments.
4. The jGrasp “how to”, http://www.jgrasp.org/jgrasp/howto1.htm, under the heading “How To Run a Program / Pass Command Line Arguments”, has a brief discussion of how to test your program in jGrasp with command line arguments.

Coding Standards

Your program should follow the same standards described in Lab 1.

Your program will be graded on conformance to the coding standards as well as correct functionality.

Turn in your program to Canvas

Look for the Lab 3 link on the Assignments Canvas page.

Assignment 1

If you have time left after you have completed this lab, feel free to start on Assignment 1 which has been posted on Canvas.