Homework 1: Imperative Programming

Due: 5:00 PM, February 10, 2017

Contents

1 Conversions (Practice) 5
2 Induction (Practice) 6
3 Cryptography 7
   3.1 Caesar Cipher 7
   3.2 Polybius square 9
4 Bubble Sort 9

Objectives

By the end of this homework, you will know:

• effective methods to keep your secrets from falling into the hands of your enemies
• the basics of the CS 18 style and testing requirements

By the end of this homework, you will be able to:

• write programs that loop
• use libraries to manipulate strings
• prove by induction that a loop invariant holds
• use one- and two-dimensionals arrays as data structures

How to Hand In

For this (and all) homework assignments, you should hand in answers for all the non-practice questions. **For this homework specifically, this entails answering the Caesar Cipher, Polybius Square, and Bubble Sort questions.**

In order to hand in your solutions to these problems, they must be stored in appropriately-named files with the appropriate package header in an appropriately-named directory. The source code files should comprise the \texttt{hw01.src} package, while your solution files comprise the \texttt{hw01.sol} package.
Begin by copying the source code from the course directory to your own personal directory. That is, copy the following file from /course/cs0180/src/hw01/src/*.java to ~/course/cs018/workspace/javaproject/src/hw01/src:

- CaesarCipherSupport.java

Do not alter this file!

After completing this assignment, the following solution files should be in your ~/course/cs0180/workspace/javaproject/sol/hw01/sol directory:

- CaesarCipher.java, containing the class public class CaesarCipher, with the functions:
  - public static String encrypt(String plainText, int shift)
  - public static String decrypt(String encryptedText, int shift)
- CaesarCipherTest.java, containing the class public class CaesarCipherTest.
- PolybiusSquare.java, containing the class public class PolybiusSquare, with the functions:
  - public static String decrypt(int[] encryptedText, char[][] polybiusSquare)
  - public static int[] encrypt(String plainText, char[][] polybiusSquare)
- PolybiusSquareTest.java, containing the class public class PolybiusSquareTest.
- PolybiusSquare.tex and PolybiusSquare.pdf, which should contain your analysis of the encrypt and decrypt functions in the Polybius Square problem.
- BubbleSort.java, containing the class public class BubbleSort, with the functions:
  - public static void bubbleSort(int[] a)
- BubbleSortTest.java, containing the class public class BubbleSortTest.
- BubbleSort.tex and BubbleSort.pdf, which should contain your analysis of bubble sort.

To hand in your files, navigate to the ~/course/cs018/workspace/javaproject/ directory, and run the command `cs018_handin hw01`. This will automatically hand in all of the above files. Once you have handed in your homework, you should receive an email, more or less immediately, confirming that fact. If you don’t receive this email, try handing in again, or ask the TAs what went wrong.

A Note on Style

As in CS 17, your code in CS 18 will be graded for functionality, efficiency, and style. Our expectations for good coding practices are laid out in the CS 18 [Good Coding Practices] document. If you follow these guidelines, you will find that your code is easier to read and debug. And as a bonus, you won’t lose points for style.
Testing

In this, and all, CS 17 / 18 assignments, you must thoroughly test your code. What that means is: we expect to see test cases for every function that you write.

In Lab 1, you learned how to use the Prima Testing Library (tester.jar). We expect you to follow the testing format outlined in Lab 1 on this and all future assignments. If you’re having difficulty, or need further clarification, come to TA Hours, and we’ll be happy to assist.

Testing Interaction Between Different Functions  Many of the programs you write in CS 18 will be complex, consisting of multiple parts that interact with one another in various ways. In addition to (first!) testing the functionality of each individual component, it is also important to verify the correctness of these potential interactions.

For example, if we had a Math class that contained a (public static) addTwo function that returned 2 more than its input, and a (public static) subOne function that returned 1 less than its input, we might test the interaction between these functions as follows:

```java
    t.checkExpect(Math.addTwo(1), 3);
    t.checkExpect(Math.subOne(1), 0);
    t.checkExpect(Math.subOne(Math.addTwo(1)), 2);
    t.checkExpect(Math.addTwo(Math.subOne(1)), 2);
    t.checkExpect(Math.subOne(Math.subOne(Math.addTwo(Math.addTwo(1)))), 3);
```

Testing Edge Cases  An edge case is one in which an input to a function is an extreme value. For example, in a recursive function that operates on (all) natural numbers, an edge case would be the base case, 0. Likewise, in a recursive function that operates only on non-empty lists, an edge case would be a list of length 1.

In CS 17, it was quite natural to test such cases, because our programs always closely followed the structure of our data. In CS 18, you should do your best to continue to program in this way, so that your edge cases are easy to identify, and then you should be sure to test them all. It is quite likely that you will find edge cases to be one of the most common sources of bugs in your CS 18 programs.

Built-In Classes

For this homework, you will make use of the built-in Character and String classes. What this means is that there will be operations that you can perform on Characters and Strings in this assignment that you did not write yourself. For example, you may wish to make use of the toCharArray method defined for Strings:

```java
java> "Raven".toCharArray()
{'R', 'a', 'v', 'e', 'n'}
```

1Don’t worry if you don’t know what a class is just yet!
2A Character is distinct from a char.
You can also create Strings from arrays of characters:

```java
java> char[] raven = new char[]{'R', 'a', 'v', 'e', 'n'}
raven: char[] = {'R', 'a', 'v', 'e', 'n'}

java> String ravenString = new String(raven)
ravenString: String = "Raven"
```

In addition to allowing you to use various methods provided in the Java libraries, we are also providing you with a small amount of support code. This support code makes use of further methods—isLowerCase and isUpperCase—defined for the Character class, which work as follows:

```java
java> Character.isLowerCase('a')
true

java> Character.isLowerCase('Z')
false

java> Character.isUpperCase('a')
false

java> Character.isUpperCase('Z')
true
```

To understand the support code, you should also know that chars, in Java, are represented internally as (unsigned) integers ranging from 0 to $2^{16} - 1$. The integer value associated with each char is specified by the popular ASCII encoding standard. This means that you can do arithmetic with chars. For example:

```java
java> (int) 'a'
97

java> 'a' + 1
98

java> (char) ('a' + 1)
'b'

java> 'a' < 'b'
true
```

For more useful operations on characters and strings, see the ‘Method Summary’ sections in the Java documentation: [here](#) and [here](#).

---

3 which you need not to complete this assignment

4 Here is a link to a table that lists the ASCII character encodings: [http://asciitable.com](http://asciitable.com)
String Formatting

Sometimes (e.g., in the Coin Conversions practice problem on this assignment), you will want to control how numbers are formatted when printing. To do this, you can embed your desired format within a string to be printed. This string is then sent as an argument to the `System.out.format` method, along with the numbers to be formatted as specified. For example:

```
// Prints "You have 5 dollars and 31 cents"
System.out.format("You have %d dollars and %d cents", 5, 31);
```

The first argument to `System.out.format` is the `String` to be formatted. The two most common format specifiers are `%d` and `%f`. The former, `%d`, is used as a placeholder for an integer in the string, while the latter, `%f`, is used as a placeholder for a floating point number (a `float` or `double`). When using the `%f` specifier, you can specify the number of decimal places you want to print. For example, to round to three decimal places, you could use the following:

```
System.out.format("%.3f", 5.5); // Prints "5.500"
```

For more information about string formatting in Java, visit this [tutorial](#).

Practice

1 Conversions (Practice)

Temperature Conversions

No doubt you have fond memories of using Racket to convert temperatures during the beginning of CS 17. Not surprisingly, you can do the same thing in Java!

**Task:** Write a function, `fahrenheitToCelcius`, that takes as input a temperature in Fahrenheit, represented as a `double`, and returns the equivalent temperature in Celcius, also as a `double`.

**Hint:** The formula $c = (f - 32)(5/9)$ converts a temperature $f$ in Fahrenheit to a temperature $c$ in Celcius.

**Task:** Write a function, `celciusToFahrenheit`, that takes as input a temperature in Celcius and returns its equivalent in Fahrenheit.

Grade Conversions

The grade distribution in CSCI0001 (Introduction to Rainbows) was very skewed last year. A 99 and above was an A, a 66 was a B, a 33 was a C, and anything below that was an F.

**Task:** Write a function, `letterGrade`, that takes as input a student’s grade, represented as an `int`, and returns their letter grade, a `char`. 


Coin Conversions

Your next task is to write a function `dollarValue` that calculates the value, in dollars, of a collection of coins—specifically, pennies, nickels, dimes, and quarters.

Task: The `dollarValue` function naturally decomposes into four simpler functions, one for pennies, one for nickels, one for dimes, and one for quarters, each of which calculates the corresponding value in dollars. Write these four sub-functions. Each should take as input an `int` and return a `double`. For example:

```plaintext
dollarValueOfPennies(18);  // should return 0.18
dollarValueOfNickels(18);  // should return 0.90
dollarValueOfDimes(18);    // should return 1.80
dollarValueOfQuarters(18); // should return 4.50
```

Task: Now write the `dollarValue` function. This function should take as input a number of pennies, nickels, dimes, and quarters, all represented as `ints`, and return the total dollar value of its inputs, represented as a `double`. Internally, this function should call your four sub-functions. For example:

```plaintext
dollarValue(1, 2, 3, 4);  // should return 1.41
```

2 Induction (Practice)

Here are two practice problems designed to strengthen your understanding of mathematical induction.

Factorial Fun (Practice)

Task: Prove by induction that the closed form of \( F(n) \) is \( (n + 1)! - 1 \).

\[
F(n) = \begin{cases} 
0, & \text{if } n = 0 \\
 n(n!) + F(n - 1), & \text{otherwise} 
\end{cases}
\]  \quad (1)

Domino Tiling (Practice)

A tesselation, or a tiling, of a plane is a collection of planar shapes that covers the plane without any overlaps.\(^5\) For example, consider a \( 2 \times n \) rectangle. Such a rectangle can be tiled easily using \( n \) dominos. Still, an interesting question arises, namely: how many different ways can such a rectangle be tiled using \( n \) dominos? The solution to problem might surprise you!

Task: Prove by strong induction that the number of ways that a \( 2 \times n \) rectangle can be tiled by \( n \) dominos is the \( n \)th Fibonacci number.

\(^5\)The Dutch graphic artist M. C. Escher incorporated tesselations into his designs.
3 Cryptography

In ancient Egyptian hieroglyphics and on astrological Babylonian tablets, there appear to be many examples of codes, in which a word like “Babylon” is everywhere substituted for “Rome.” Substitutions like these are carried out to confuse the reader, particularly in times of war.

For example, in 1941 the Japanese created the code “Higashi no kaze ame” (“East wind, rain”) to communicate via their daily weather reports an intent to declare war on the United States. Codes are limited in their flexibility, however. While the Japanese had also created the code “Nishi no kaze hare” (“West wind, clear”) to indicate an intent to declare war on Britain, had they instead decided to declare war on Germany, they would not have been able to convey this message.

Enter ciphers. Ciphers are used to encrypt what is referred to as plaintext—meaning, you know, plain ol’ text—letter by letter. And by those in the know, ciphers can also be used to decrypt encrypted text back into plaintext.

3.1 Caesar Cipher

Beast Boy wishes to send a secret message to Cyborg, but Robin won’t leave the two alone, and they don’t want Robin to intercept and read the message. Knowing that it’ll stump Robin (and he’ll finally realize he’s not cool enough to hang with them), Beast Boy decides to add an extra layer of security to his message by using a cipher.

The Caesar cipher is a substitution cipher in which each letter in the alphabet is replaced by a letter some fixed number of positions to the right in the alphabet (with wraparound). For example, if the communicators fix “shift 1”, then abc becomes bcd.

<table>
<thead>
<tr>
<th>plain</th>
<th>...</th>
<th>Z</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>encrypted</td>
<td>...</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>...</td>
</tr>
</tbody>
</table>

Similarly, using “shift −2”, boo becomes zmm.

<table>
<thead>
<tr>
<th>...</th>
<th>Z</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>A</td>
<td>...</td>
</tr>
</tbody>
</table>

**Task:** Begin by copying `course/cs018/src/hw01/src/CaesarCipherSupport.java` to your `~/course/cs018/javaproject/src/hw01/src` directory.

**Note:** You can always find any support code we release for a problem in a correspondingly named file residing in a correspondingly named directory.

The support code for this problem includes the `shiftChar` function. This function takes as input the plaintext char to be shifted to the right, along with the amount of the shift, and it returns the appropriately encoded character.

In order to make use of this function in your solution, you can call it like this:

```java
char myChar = hw01.src.CaesarCipherSupport.shiftChar(‘a’, 2); // myChar equals ‘c’ now
```
Alternatively, after the package declaration at the top of your CaesarCipher.java file, you can write

```java
import static hw01.src.CaesarCipherSupport.shiftChar;
```

after which you can call `shiftChar` like this instead:

```java
char myChar = shiftChar('a', 2); // myChar equals 'c' now
```

**Task:** Write a function called `encrypt`, which will enable Beast Boy to send an encrypted message to Cyborg. Your function should take as input a message as plaintext (a `String`) and the intended shift (an `int`), and it should return the encrypted message, again represented as a `String`.

For example:

```java
java> encrypt("Zab", 1)
"Abc"
java> encrypt("z", 1)
"a"
java> encrypt("Z", -1)
"Y"
```

**Note:** You can assume that the plaintext includes only letters. You are not required to handle punctuation marks, white space, or any other symbols.

**Task:** Now write a function called `decrypt` so that Cyborg can decrypt Beast Boy’s message. This function should take as input an encrypted message, represented as a `String`; and an `int`, the shift; and should return the plaintext of the message, again represented as a `String`.

For example:

```java
java> decrypt("abc", 1)
"zab"
java> decrypt("a", 1)
"z"
java> decrypt("A", -1)
"B"
```

**Note:** You can assume that the encrypted message includes only letters. You are not required to handle punctuation marks, white space, or any other symbols.

**Hint:** Our `decrypt` function consists of only one line (which is necessarily a `return` statement).

**Just for Fun:** What is the run time of your Caesar cipher?
3.2 Polybius square

The Trojans decrypted the Caesar Cipher, so Achilles must use a more complex cipher: the Polybius Square. The Polybius square is constructed by putting the alphabet in a $5 \times 5$ array. This array, using the English alphabet, would look something like this:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
</tr>
<tr>
<td>2</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>3</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
</tr>
<tr>
<td>4</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
</tr>
</tbody>
</table>

Now, to encrypt the message “STARFIRE”, we would use 16 numbers—2 for each letter. S is in row 3 and column 3, so we use 3 (the row) and 3 (the column) as the first two numbers in our message. T is in row 3 and column 4, so we use 3 (the row) and 4 (the column) as the next two numbers in our message. And if we continue encrypting like this, we get the encrypted message:

```
3334003210133204
```

Note: We assume the multidimensional array representing a Polybius square stores letters in row-major order: that is, the first index corresponds to the row, and the second index, the column.

Task: Write a function `decrypt` that takes as input an encrypted message, represented as an array of `ints`, and a Polybius square, represented as a two-dimensional array of `chars`, and returns the decrypted message, represented as a `String`.

Note: You can assume the encrypted message contains only valid indices, meaning indices that reference letters in the given Polybius square. You can also assume the array of integers representing the encrypted message is of even length.

Task: Write a function `encrypt` that takes as input a plaintext message, represented as a `String`, and a Polybius square, and returns the encrypted message, represented as an array of `ints`.

Note: You can assume the characters in the plaintext message are all contained in the given Polybius square. You cannot assume the Polybius square is always a $5 \times 5$ array; it might not even be a square, but it will always be a rectangle.

Task: Compare the run times of `encrypt` and `decrypt`, as a function of the size of the message to be encrypted or decrypted.

State your answer in big-$O$ notation, clearly explaining the meanings of all the variables you use. Give a brief justification; a formal proof is not necessary.

Hint: How does the size of the Polybius square affect the run time of `encrypt` and/or `decrypt`?

4 Bubble Sort

As described in lecture, bubble sort is a classic sorting algorithm that works by repeatedly making passes through the array to be sorted, comparing adjacent elements, and swapping them (in place) if they are not already in sorted order.
Task: Write a `bubbleSort` function which takes as input an array of (possibly duplicated) `int`s and sorts them using the bubble sort algorithm. The return type of `bubbleSort` should be `void`; it should sort the input array as a side effect of being called.

Task: As there are two loops in bubble sort, there are two invariants, one corresponding to the inner loop and the other corresponding to the outer loop. What are they?

Task: Prove that bubble sort maintains your two invariants.

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS18 document by filling out the anonymous feedback form:

[http://cs.brown.edu/courses/cs018/feedback](http://cs.brown.edu/courses/cs018/feedback)