# Section Handout #4: String Processing

## 1. Adding commas to numeric strings  (Chapter 8, Exercise 13, page 290)

When large numbers are written out on paper, it is traditional—at least in the United States—to use commas to separate the digits into groups of three.  For example, the number one million is usually written in the following form:

```
1,000,000
```

To make it easier for programmers to display numbers in this fashion, implement a method

```
private String addCommasToNumericString(String digits)
```

that takes a string of decimal digits representing a number and returns the string formed by inserting commas at every third position, starting on the right.  For example, if you were to execute the main program

```
public void run() {
   while (true) {
      String digits = readLine("Enter a numeric string: ");
      if (digits.length() == 0) break;
      println(addCommasToNumericString(digits));
   }
}
```

your implementation of the `addCommasToNumericString` method should be able to produce the following sample run:

## 2. Deleting characters from a string

Write a method

```
private String removeAllOccurrences(String str, char ch)
```

that removes all occurrences of the character **ch** from the string **str**. For example, your method should return the values shown:

```
removeAllOccurrences("This is a test", 't')   returns   "This is a es"
removeAllOccurrences("Summer is here!", 'e')  returns   "Summr is hr!"
removeAllOccurrences("---O---", '-')          returns   "O"
```

## 3. Converting a string to alternating capital letters

Write a method

```
private String altCaps(String str)
```

which converts a string to alternating capital letters, meaing you alternate between uppercase and lowercase. This style of typing was prevalent on the internet in the late 90s. For example:

```
altCaps("hello")              returns "hElLo"
altCaps("section is awesome") returns "sEcTiOn Is AwEsOmE"
```

Note that characters that are not letters are not changed and do not affect the alternating sequence of uppercase and lowercase letters.

## 4. Pig Latin

Write a method

```
private String pigLatin(String word)
```

that converts a single lowercase word to a simplified version of Pig Latin, a (silly) variant of English where the first letter of each word is moved to the end. The rules for translating a word to Pig Latin are as follows:

If the word starts with a vowel (a-e-i-o-u), simply append "yay" to the end of the word.

```
pigLatin("elephant")              returns "elephantyay"
pigLatin("aardvark")              returns "aardvarkyay"
```

If the word starts with a consonant, move all constants up to the first vowel to the end, and append "ay".

```
pigLatin("switch")                returns "itchsway"
pigLatin("string")                returns "ingstray"
```

You should assume that **word** has length $>= 0$ and is comprised of only lowercase letters.

## 5. Tracing method execution

For the program below, show what output is produced by the program when it runs.

```java
/*
 * File: Mystery.java
 * ------------------
 * This program doesn't do anything useful and exists only to test
 * your understanding of method calls and parameter passing.
 */

import acm.program.*;

public class Mystery extends ConsoleProgram {

   public void run() {
      ghost(13);
   }


   private void ghost(int x) {
      int y = 0;
      for (int i = 1; i < x; i *= 2) {
         y = witch(y, skeleton(x, i));
      }

      println("ghost: x = " + x + ", y = " + y);
   }


   private int witch(int x, int y) {
      x = 10 * x + y;
      println("witch: x = " + x + ", y = " + y);
      return x;
   }


   private int skeleton(int x, int y) {
      return x / y % 2;
   }
}
```