CSE 11 **START EARLY!**
Fall 2013
Homework Assignment #6

Due: 16 November 2013, 11:00pm (Saturday!)
Covers Chapters: 14,15

This is a programming assignment with a number of independent programs. It allows you to explore arrays.

You will turn in several files for this assignment:

```
Reverse.java
ReverseBuf.java
SwapBytes.java
MaxInt.java
MatMul.java
```

Bundle your homework with the program

```
$ /home/linux/ieng6/cs11e/public/bin/bundleP6
```

Please make sure to include your
NAME:
LOGIN:
ID:
in every file you turn in.

Problem #1 (25 points)

Use the standard Java Scanner class, as in Homework #5, to code two variants of the same program. Each program, `Reverse.java`, and `ReverseBuf.java` takes a single argument on the command line that is the name of a text file (e.g., java source files, your PR6.txt file, but not PDF files or java class files) and prints the file in reverse line order. That is, the last line of the file is printed first, second-to-last is printed second, and the first line is printed last. You do not need to check if the file is a text file, simply assume in grading that your program will be either given a text file, or the file does not exist. If the file does not exist, print out "Cannot open file" and the name of the file. You should be able to test your programs with

```
$ java Reverse Reverse.java
```

it should print out your source code in reverse order. (also make sure to test when the file does not exist and when the file contains no lines).

1. Reverse.java. **Read the input file twice**. The first time, read it to find out how many lines it contains. Once you know the number of lines in the files, it should be straight forward to read the file again, place contents in a properly sized array, and then print in reverse order.
2. ReverseBuf.java. **Read the input file once**. To do this, read the input file once into a fixed size "buffer". Define a constant called BUFSIZE initialized to 10, and initialize your buffer to be this size. If the file, is longer than the buffer (you only know when you read the file a line at a time), create a new buffer that has been extended to have additional BUFSIZE entries, and continue reading until you have read the entire file. It is highly recommended that you create a

private method with the following signature

private static String [] expandArray(String [] array, int extend). This method will make an new array that is "extend" slots longer than the original. It should then copy the contents of the old array to the new array and return the new array.

Problem #2 (35 points)

Create a program called SwapBytes, and is called as follows

$ java SwapBytes <filename> [index1] [index2] [index3] ...

1. Reads the file <filename> a line at a time
2. Uses the the String.charAt() method to copy each line into an array of characters. (`char`)
3. It then swaps the characters at index1 and index1 + 1, at index2 and index2 + 1, etc.
4. It prints out the character array as a line with the characters swapped.
5. An index could be negative. If it is negative, it should not affect the output

Example:
Suppose the input file mytest.txt has the following 4 lines

```
please excuse my dear aunt sally
today
and yesterday

OR
tomorrow
```

```
$ java SwapBytes 0 18
```

should print out

```
lpease excuse my daer aunt sally
otday
nad yesterday

RO
otmorrow
```

```
$ java SwapBytes 1 18 26
```

should print out

```
pelase excuse my daer aunts ally
tdoay
adn yesterday

OR
tmoorrow
```

A few things to keep in mind:

- The number indexes to swap may be zero, one, two, three, ...
- If no indexes are on the command line, the file should be printed, line-by-line with no changes
- Do NOT include the newline character at the end of the line as one of the characters.
- Not all lines will be long enough to swap at a particular index. That should not be an error, this

results in no characters being swapped at that index for that particular input.

- Your code should not have any run time exceptions. (try to "break" with various inputs. You might try SwapBytes <filename> 0 0 and see what happens. The assignment does not specify what should happen here (there are two logical possibilities))

Problem #3 (10 points)

Create a program called `MaxInt.java`
MaxInt reads a file of integers (one integer per line) and prints out the
the Maximum integer in the file and the number of times the maximum occurs.
Call as

$ java MaxInt <filename>

Use the following format statement:

System.out.format("%d : %d\n", max, ntimes);

where max is the maximum found, ntimes is the number of times it occurs.
If the file is empty (no lines), the program should print nothing.

Problem #4 (30 points) `Matrix Multiplication.`

Look up how to perform matrix multiplication (the Matrix product) . (if you don't recall the specifics, a very good introduction is at http://en.wikipedia.org/wiki/Matrix_multiplication) In this exercise you will multiply two randomly-generated square matrices of dimension N. Write a program called `MatMul.java` that takes two arguments "N", Max. Call as

$ java MatMul <N> <Max>

MatMul.java will do the following:

1. Create two randomly-generated NxN matrices of integers called A and B. Each entry of A and B should be in the closed interval of [-MAX, MAX]
2. It should calculate C = A * B; where * is standard matrix multiplication.
3. It should also calculate
   (a) The sum of each row of C
   (b) The sum of each column of C
   (c) The sum of the diagonal elements of C

Your program should print out
A =
<each row of A on a separate line>
B =
<each row of B on a separate line>
C =
<each row of C on a separate line>
RowSum =
<one line that is the sum of each row of C>
ColSum =
<one line that is the sum of each column of C>
DiagSum =
<one number that is the sum of the diagonal elements of C>