Faculty of Arts & Sciences
Department of Computer Science
CMPS 200—Introduction to Programming
Assignment 10 – Due Monday Dec 3, 2012

Notes and Announcements

- Reading material: Chapter 8 of the text (for Program #6)

Exercises

1. Insertion Sort of integers

Write an `InsertionSort.java` program that reads an array of integers from standard input and sorts the array in increasing order using an insertion sort method. Insertion sort works by repeatedly looping through the array: at iteration i, the method should insert the i-th element in a location such that everything to its left is smaller that it, and "pushing" all elements between the target location and the i-th location to the right. See the Wikipedia entry http://en.wikipedia.org/wiki/Insertion_sort for an illustration. Insertion sort is a famous sorting algorithm that has the property that after k iterations, the first k+1 elements are in sorted order. When humans manually sort something (for example, a deck of playing cards), most use a method that is similar to insertion sort.

Your program should contain the following methods:

```
public static void insertionSort(int[] a)         // insertion sort an array in place
public static void printArray(int[] a)            // prints an array to standard output
```

and a main method that reads from standard input, creates an array of ints, and then calls `insertionSort()`, and `printArray()` as needed. (You may assume that the first entry in the input is the number of elements in the array and is followed by the values in the array.)

In addition, you should write a method that doesn't change its argument, but returns a new sorted array
`public static int[] insertionSort2(int[][] a)`
Hint: Use the `insertionSort()` method you wrote above to implement this method.

2. Insertion Sort of Strings

Write a program that takes two file names as command line arguments. The program should read the contents of the first file and write its strings in lexicographic order (the dictionary order) in the second file. All capitalized words should appear before the non-capitalized ones.

The program has the read the input file twice: the first time to count how many strings it contains, and the second time to build an array of strings. Use the AUB Mission statement file as sample input to test your program.

3. Format

The unix fmt program reads lines of text, combining and breaking lines so as to create an output file with lines as close to without exceeding 72 characters long as possible. The rules for combining and breaking lines are as follows.

- A new line may be started anywhere there is a space in the input. If a new line is started, there will be no trailing blanks at the end of the previous line or at the beginning of the new line.
- A line break in the input may be eliminated in the output, provided it is not at the end of a blank or empty line and is not followed by a space or another line break. If a line break is eliminated, it is replaced by a space.
- Spaces never appear at the end of a line.
- If a sequence of characters longer than 72 characters appears without a space or line break, it appears by itself on a line.

Sample Input

```
The unix fmt program reads lines of text, combining
and breaking lines so as to create an
output file with lines as close to without exceeding
72 characters long as possible.  The rules for combining and breaking
lines are as follows.

   1.  A new line may be started anywhere there is a space in the input.
If a new line is started, there will be no trailing blanks at the
end of the previous line or at the beginning of the new line.

   2.  A line break in the input may be eliminated in the output, provided
it is not followed by a space or another line break.  If a line
break is eliminated, it is replaced by a space.
```

Sample Output

```
The unix fmt program reads lines of text, combining and breaking lines
so as to create an output file with lines as close to without exceeding
72 characters long as possible.  The rules for combining and breaking
lines are as follows.

   1.  A new line may be started anywhere there is a space in the input.
If a new line is started, there will be no trailing blanks at the end of
the previous line or at the beginning of the new line.

   2.  A line break in the input may be eliminated in the output,
provided it is not followed by a space or another line break.  If a line
break is eliminated, it is replaced by a space.
```

4. Grid Traversal

Suppose you have a grid of NxN cells with one number per cell. Write a program GridTraversal.java that traverses the grid from outside to inside (as shown below) and list the values of the traversed cells. The program takes the name of an input file as a command line argument and prints its output to standard output.

Input Format

The input has the following format:

N

$v_{11}, v_{12}, \ldots, v_{1N}$

$v_{21}, v_{22}, \ldots, v_{2N}$

…

$v_{N1}, v_{N2}, \ldots, v_{NN}$

N is the dimension of the grid, and $v_{i,j}$ corresponds to the value in the cell at *i*, *j*.

Output Format

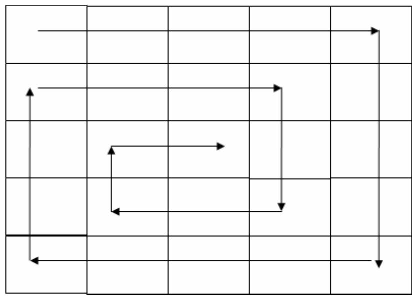
Display the values of the cells in their traversal order separated by blanks spaces on the screen.

Sample Input file:

```
4
1, 2, 3, 4
12,13,14,5
11,16,15,6
10, 9, 8,7
```

Sample Output

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
```


Traversing a 5x5 grid from outside to inside.

5. Pascal's Triangle

Write a program `Pascal.java` that builds and prints a two-dimensional *jagged* array a such that a[n][k] contains the coefficient of the k-th term in the binomial expansion of $(x + y)^n$. These coefficients can be organized in a triangle, famously known as Pascal's triangle. Every row in the triangle may be computed from the previous row by adding adjacent pairs of values together. Your textbook has a description of the algorithm on pages 499-503. Below is a sample invocation of the program.

```
> java Pascal 7
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
```

6. Fraction Data Type

Write a data type `Fraction` that can support the usual arithmetic operations on fractions. A fraction may be represented by a numerator and denominator: two integer private data members. A possible API for `Fraction` is shown below:

```
 public          Fraction (int n, int d)   // constructor (n / d)
 public          Fraction (int v)          // constructor, default denominator =1
 public Fraction times(Fraction f)         // multiplication
 public Fraction plus(Fraction f)          // addition
 public boolean  equals(Fraction f)        // equality check
 public boolean  lessThan(Fraction f)      // < operator
 public String   toString()                // returns printed representation
```

Your code should be in a file called `Fraction.java`. The file should include a `main()` method to test the various methods of the class. Use the `main()` below and augment it with a few additional tests. You may also put `main()` in another client program.

```
public static void main(String[] args) {
    Fraction x = new Fraction(1, 2);
    Fraction y = new Fraction(1, 4);
    Fraction z = x.times(y);
    Fraction w = x.plus(y);
    System.out.println(x + " * " + y " = " + z);
    System.out.println(x + " + " + y " = " + w);
}
```