Introduction to Computer Science I
Spring 2013
Mid-term exam solutions

1. Question: (10 points) For the following sequence of expressions, indicate the value that is assigned to each variable:

   ```
   int x = 8;
   int y = 3;
   int z = x / y;
   float a = x / y;
   float b = (float)x / y;
   float c = x / (float)y;
   float d = (float)(x / y);
   ```

   Solution:

   ```
   z = 2
   a = 2.0
   b = 2.6666667
   c = 2.6666667
   d = 2.0
   ```

   Observations: Just about every possible mistake was made on this problem. This problem requires a clear and unambiguous notion of the order in which operations—particularly, *casting* and *arithmetic* operations—are performed. If you do not know what type each value will be at each step, then you likely missed one or more of these assignments.
   Some people asserted, as their answers, that one or more of the expressions were invalid, and thus would cause a failure of either *compilation* (i.e., `javac` would reject it) or *execution* (i.e., it would cause the program to crash). I consider it underhanded and unreasonable for me to ask, *What does this code do?*, when the answer is, *Ha ha! Tricked you! This isn't working code!* I will never place such a question on a written test.

2. QUESTION: (15 points) Consider the following recursive method:

```
public static long factorial (int n) {

    System.out.println("Begin: " + n);
    long result;
    if (n == 0) {
        result = 1;
    } else {
        result = n * factorial(n - 1);
    }

    System.out.println("End: " + n + " is " + result);
    return result;

}
```

If some method (e.g., `main()`) were to call this one, passing an argument of 5, **what output would this method generate?**

SOLUTION:

```
Begin: 5
Begin: 4
Begin: 3
Begin: 2
Begin: 1
Begin: 0
End: 0 is 1
End: 1 is 1
End: 2 is 2
End: 3 is 6
End: 4 is 24
End: 5 is 120
```

OBSERVATIONS: The most common mistake with this problem was a failure consider the output generated by the "inner" calls to this recursive method. That is, with `main()` making the initial call of `factorial(5)`, the output of that one call is only part of the output generated by the multiple calls to this method that will ensue. Some people did account for the many calls, but forgot to handle both the `Begin` and `End` messages; others accounted for both but expressed their ordering incorrectly. Once again, a few people asserted that this code would either not compile or would not run. I would never ask a question of this form for which that was the answer.

3. QUESTION: (20 points) Write a method named `printSidewaysTriangle()`. Like other methods of this kind that you have written, it has one parameter, `size`, that indicates how many rows the triangle has. A triangle of size 7 should look like this:

```
*
***
*****
*******
*****
***
*
```

SOLUTION:

```java
public static void printSidewaysTriangle (int size) {
    for (int i = 1; i < size; i = i + 2) {
        printRow('*', i);
    }
    for (int i = size; i >= 1; i = i - 2) {
        printRow('*', i);
    }
}

public static void printRow (char c, int length) {
    for (int i = 0; i < length; i = i + 1) {
        System.out.print(c);
    }
    System.out.println();
}
```

OBSERVATIONS: First, **nobody** broke off the task of printing a row of stars to a separate method, even though the code to perform that task was written in at least two places. Second, many people over-thought their solutions, attempting fancy arithmetic to determine where the middle of the shape was, and how to create rows of the correct length. I allowed just about any half-sane interpretation about what this method should print in the presence of an even-valued *size* parameter.
Sadly, some also attempting needlessly complex structures involved nested loops (beyond the need for inner loops to print the rows). These were difficult to interpret, and while some printed a portion of some meaningful shape, none of them created a correct structure overall.

4. Question: (25 points) **Write the bodies of the following methods** that operate on arrays:

   (a) `public static void charPrintLn (char[] msg)` A method that takes, as a paremeter, a pointer to an array of `char` (named `msg`) and treats it like a string, printing the characters one at a time and then moving to the next line after printing them all.

   (b) `public static double[] extract (double[] source, int[] indices)` A method that takes, as parameters, two arrays of integers: one that is a `source` sequence of values (whose meaning is unknown and unimportant), and a sequence of `indices` that specify positions within the `source`. This method should **make a new array of** `double`, populating its entries with the values from `source` specified by the positions in `indices`. For example, consider these two arrays:

```
               0      1     2    3       4
source:   13.3  -2.81  0.05  1.8   17.09

          0  1  2
indices: 2  0  3
```

   The method should produce the following resultant array:

```
               0     1    2
result:   0.05  13.3  1.8
```

Solution:

```
public static void charPrintLn (char[] msg) {
    for (int i = 0; i < msg.length; i = i + 1) {
        System.out.print(msg[i]);
    }
    System.out.println();
}

public static double[] extract (double[] source, int[] indices) {
    double[] result = new double[indices.length];
    for (int i = 0; i < indices.length; i = i + 1) {
        result[i] = source[indices[i]];
    }
    return result;
}
```

Observations: The first method was intended to be a “gimme”—all that is required is the individual printing of each character. Those who missed this portion of the question should be sure that the basic types of *character* and *array* are reviewed and clearly understood.
The second method was trickier, in that the `indices` array itself contains index values that must be used to select values from the `source` array. While most managed this key insight (and received majority credit for it), some erred in expressing the length of `result` or the corresponding length of the loop that copied values.

5. QUESTION: (30 points) Consider the following beginning to a method...

   ```
   public static int charsToInt (char[] chars)
   ```

   This method accepts a sequence of characters that **is** be a textual representation of an integer value. (Don't worry about handling malformed character sequences like `{'P', 'o', 'o', 'd', 'l', 'e'}` ; the sequence is guaranteed to be something of the form `{'3', '1', '5', 7'}`.) The method should convert the sequence of characters into an honest-to-goodness `int` value, and then it should return that value. **Notice:** 5 of the points for this problem are for handling character sequences that specify a *negative* integer, such as `{'-', '9', '0', '2'}`. If this seems a complication that gets in the way of solving the rest of the problem, sacrifice the 5 points to focus on the other 25.

   SOLUTION:

   ```
   public static int charsToInt (char[] chars) {
       int start = 0;
       if (chars[0] == '-') {
           start = 1;
       }
       int sum = 0;
       for (int i = start; i < chars.length; i = i + 1) {
           int digit = chars[i] - '0';
           sum = (sum * 10) + digit;
       }
       if (start == 1) {
           sum = -sum;
       }
       return sum;
   }
   ```

   OBSERVATIONS: A number of you, in spite of my in-class announcement, misunderstood this method as converting each digit into its own integer value, rather than parsing the sequence of characters as a single integer. Once that hurdle was cleared, though, most attempted a reasonable set of calculations to perform the task. A most common error was to take each character and use it as the digit itself, without subtracting the character value `'0'` to convert the character into its corresponding integer value. Another, more bizzare error was some attempt to use a method like `Integer.parseInt()`: its invocation was invariable incorrect, but even if it had been used properly, its use was a fundamental dodge that could not be given credit.