The State Universtiy of New York, Korea
Computer Science

CSE 114 Handout 16: PS 11 May 31, 2019

This problem set is due **Monday, June 10 at 11:59pm, KST**. Note that the due date that you see on Blackboard is not accurate since it shows the time in EST. You should go by the due date in this handout. Also note that this is our last problem set for the semester. **Try to finish first four problems soon and work on the last one dealing with JavaFX after our lecture on June 3.**

- Follow the specification exactly. It makes grading much easier.
- Each method that you write must be properly commented and attractively formatted.
- You may create and run your programs using Eclipse if you want, or continue using emacs and command line interface.
- Your programs should be formatted in a way thats readable. In other words, indent appropriately, use informative names for variables, etc. If you are uncertain about what a readable style is, see the examples from class and textbook as a starting point for a reasonable coding style.
- **Be sure to include** a comment at the top of each file submitted that gives **your name** and **email address**.

What to Submit

Submit the following files as a **single zip or tar file** on **Blackboard**. Multiple submissions are allowed before the due date. Please do **not** submit `.class` files, Eclipse-specific project files, or any that I did not ask for.

```
Recursion.java
Arith.java
Unique.java
More.java

TTT.java
o.gif  (preserving folder structure so that I will NOT have to
x.gif   edit your TTT.java to run your program.)
```

What Java Features to Use

For this assignment, you may use anything that we have studied this semester.

Partial vs. Complete Solutions

Please read what I said in PS 1 on this issue.

Naming Conventions In Java And Programming Style In General

Refer to the ones given in PS 1.

Problem 1 (50 points)

Start with `Recursion.java` in the [given] folder and introduce the following methods. These methods will naturally be `static` methods.

1. (5 pts) Write a recursive method called `power` that takes a double x and an integer n and that returns x^n. **Hint:** a recursive definition of this operation is `power(x, n) = x * power(x, n - 1)`. Also, remember that anything raised to the zeroth power is 1. In your `main` call `power` as follows:

```
System.out.println("power(3, 0) = " + power(3, 0));
System.out.println("power(2, 5) = " + power(2, 5));
System.out.println("power(3, 4) = " + power(3, 4));
```

2. (5 pts) This following algorithm is known as Euclid's Algorithm because it appears in Euclid's *Elements* (Book 7, ca. 300 B.C.). It may be the oldest nontrivial algorithm.

   The algorithm is based on the observation that if r is the remainder when a is divided by b, then the common divisors of a and b are the same as the common divisors of b and r. Thus we can use the equation:

   $$gcd(a, b) = gcd(b, r)$$

   to successively reduce the problem of computing a GCD to the problem of computing the GCD of smaller and smaller pairs of integers. For example,

   $$gcd(36, 20) = gcd(20, 16) = gcd(16, 4) = gcd(4, 0) = 4$$

   implies that the FCD of 36 and 20 is 4. It can be shown that for any two starting numbers, this repeated reduction eventually produces a pair where the second number is 0. Then, the GCD is the other number in the pair.

   Write a recursive method called `gcd` that takes two integer parameters and that uses Euclid's Algorithm to compute and return the greatest common divisor of the two numbers. In your `main` call `gcd` as follows:

```
System.out.println("gcd(36, 20) = " + gcd(36, 20));
System.out.println("gcd(34, 0) = " + gcd(34, 0));
System.out.println("gcd(3346, 468) = " + gcd(3346, 468));
```

3. (5 pts) A function f is defined by the following rule:

   $$f(n) = \begin{cases} n & \text{if } n < 3 \\ f(n-1) + 2f(n-2) + 3f(n-3) & \text{if } n \geq 3 \end{cases}$$

   Write a recursive method called `f` that takes an `int` as a parameter and that computes and returns the value of the given function. In your `main` call `f` as follows:

```
System.out.println("f(2) = " + f(2));
System.out.println("f(3) = " + f(3));
System.out.println("f(10) = " + f(10));
```

4. (5 pts) Consider computing the sequence of Fibonacci numbers, in which each number is the sum of the preceding two: $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$

   In general, the Fibonacci numbers can be defined by the rule:

$$Fib(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ Fib(n-1) + Fib(n-2) & \text{otherwise} \end{cases}$$

   Write a recursive method called `fib` that takes an `int` as a parameter and that computes and returns the value of the Fibonacci function. In your `main` call `fib` as follows:

```
System.out.println("fib(0) = " + fib(0));   // returns 0
System.out.println("fib(1) = " + fib(1));
System.out.println("fib(2) = " + fib(2));
System.out.println("fib(3) = " + fib(3));   // returns 2
System.out.println("fib(4) = " + fib(4));   // returns 3
System.out.println("fib(10) = " + fib(10)); // returns 55
```

5. (10 pts) We have seen how to use the `charAt, substring,` and `length` functions defined on `String`. In fact, for `substring`, we saw two variations: one taking one argument and another taking two arguments. You will be using those functions (and others if you need) to solve the next several problems. First, write a recursive method called `printBackward` that takes a `String` as a parameter and that prints the characters of the `String` backwards (one character per line). It should be a `void` method. In your `main` call `printBackward` as follows:

```
printBackward("Java Fun!");
```

6. (10 pts) Write a recursive method called `printString` that does the same thing as `printBackward` but that prints the `String` in the same order of characters in the `String` given as the argument (one character per line). In your `main` call `printString` as follows:

```
printString("Java Fun!");
```

7. (10 pts) Write a recursive method called `reverseString` that takes a `String` as a parameter and that returns a new `String` as a return value. The new `String` should contain the same characters as the parameter, but in reverse order. For example, the output of the following code

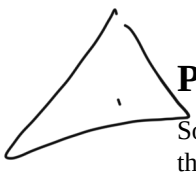
```
String reversed = reverseString("Java Fun!");
System.out.println(reversed);
```

   should be

```
!nuF avaJ
```

   In your `main` call it as given above.

Hand in your `Recursion.java`.



Problem 2 (40 points)

Some functions can be defined in terms of a small number of simple primitives. In this problem we will see how this is true for even the simplest mathematical operations. In this problem you will *not* be allowed to use some

common arithmetic operators such as addition (`'+'`), subtraction (`'-'`), multiplication (`'*'`), or division (`'/'`). Of course, you are free to define auxiliary functions and you may assume that the first definitions "carry over" into the next ones. I will give you the beginning of the solution for this problem in `Arith.java` and your task is to complete it.

(a) (10 pts) Define a recursive function named `add` which takes two numeric arguments and returns their sum. For example, `add(9, 10)` returns 19 and add(-15, 8) returns -7. Don't even think that the arithmetic operator `'+'` would be useful for you to use.

(b) (10 pts) Define a recursive function named `sub` which returns the difference of its two numeric arguments. For example, `sub(14, 9)` returns 5 and `sub(-15, -5)` returns -10.

(c) (10 pts) Define a recursive function named `mul` which takes two numeric arguments and returns their product. For example, `mul(3, 7)` returns 21 and `mul(-4, 8)` returns -32.

(d) (10 pts) Define a recursive function named `quo` to compute the quotient of its two numeric arguments. `quo(10, 5)` returns 2 and `quo(11, 5)` returns 2.

Problem 3 (20 points)

Write a method that satisfies the following description:

```
// Returns true if x occurs in the range [i, nums.length - 1] in nums.
// That is, if x occurs in nums between the indices i and
// nums.length - 1 inclusive.  This method must be recursive.
public static boolean occurs(int x, int[] nums, int i) {
    // fill this in
}
```

Now write another method that satisfies the following description. This method would find `occurs` that you defined above useful.

```
// Returns true if the elements in nums are unique, i.e., if there
// are no duplicates.
public static boolean unique(int[] nums) {
    // fill this in
}
```

Note that `unique` itself may not be recursive, but you would find it helpful to define an auxiliary function that is recursive. If you write such an auxiliary function, make it recursive.

In the [given] folder I provided a skeleton (`Unique.java`) with a `main` that you should use to develop your solution.

Problem 4 (20 points)

Add the following methods to `More.java` in the [given] folder and test them using the `main` method in `More.java`.

(a) (10 pts) Define a recursive function named `hasMoreVowelsThanConsonants` which takes one string as an argument and returns true if the string contains more vowels and consonants. The string may contain upper-case and lower-case letters. Add at least two test cases in the `main` that return different results.

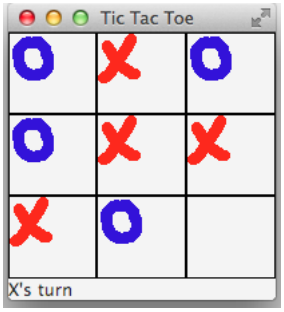
(b) (10 pts) Define a recursive function named `putEvensBeforeOdds` which takes one array of integers as an argument and rearranges the elements so that all the even values appear before all the odd values. The relative order among even values must remain the same. Similarly for odd values. If you define an auxiliary

function that uses an array as a parameter, that function must be recursive as well. Add at least three test cases in the `main`: one array containing only even values, second array containing only odd values, and the third array containing a mix of even and odd values in some arbitrary order. You are not allowed to use any additional array.

Problem 5 (70 points)

Start with `TTT.java` in the [given] folder and do the following.

1. Run `TTT.java` to see how it works and how it appears to you, the user.
2. The original `TTT.java` given displays an 'X' with two lines and an 'O' as an ellipse. Compare this one with the one you will work on in Lab 22 on June 4. The one in Lab 22 displays an 'X' as an image with an 'X' in red, and similarly for an 'O' too but in blue. In this problem you will revise the given `TTT.java` in such a way that it will display 'X's and 'O's as images like the ones in Lab 22 does. Once you are done, it will look something like the following, which is also given as `Sample.png` in the [given] folder:



3. This much is required for this problem set

Optionally (meaning this part is *not* required and I am not offering any extra credit), you can improve the given game as follows:

1. As you can see in `Sample.png`, the letters 'X' and 'O' are not centered in each cell. Make them appear in the center of each cell. They should stay in the center even if you resize the window.
2. The given version of Tic Tac Toe continues until the entire board is filled up even when it is already a tie game. Improve your game so that it can detect a tie game if it is a tie game even if the board has not been completely filled up yet as we did in one of our earlier problem sets.
3. Improve it further so that it is played by a human against a computer as we did in one of our earlier problem sets.

Hand in your revised `TTT.java` along with the image files. Make sure that you preserve the folder structure in such a way that I will not have to edit your `TTT.java` to run your program.