INTRODUCTION TO COMPUTER SCIENCE I
PROJECT 3A
Conditionals and loops

Here is the first of a two-part project that will exercise your use of our new-found *conditional*[1] and *iterative*[2] *statements*. You will use them, along with your experience with *calling* and *writing methods*, in various combinations to perform a few new types of calculations.

1 Factorials

Consider the *factorial function*:

$$fact(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \times fact(n-1) & \text{if } n > 0 \end{cases}$$

This function forms a *sequence*, where the 0^{th} entry is 1, and the n^{th} entry is the product of all numbers from 1 to n. The sequence begins: $1, 1, 2, 6, 24, 120, 720, \ldots$

A program to write: You must write a new program, from scratch, composed of a few methods. Here is the progression that you should follow:

1. **Getting started:** Login to `remus/romulus`. At the command line, create a new directory and change into it, like so[3]:

   ```
   $ mkdir project-3
   $ cd project-3
   ```

   Then, create and open a new, blank source code file with *Emacs*:

   ```
   $ emacs Factorial.java
   ```

   Finally, inside this file, put in the usual stuff that surrounds the methods that you write:

[1] a.k.a., `if-then-else`.

[2] a.k.a., `while` loops.

[3] A number of people have been a bit confused about directories and the commands that allow you to manipulate your files and directories at the shell prompt. First, note that the first command, `mkdir`, is something that you should do only once at the beginning of each project, thus making a directory for that project. The second command here, `cd`, makes the shell change its current directory so that all of the files that you open, compile, and run from that point forward are in that directory. Thus, the `cd` command is one that you must use each time you login to `remus/romulus` before using `emacs`, `javac`, `java`, or `cs111-submit`. If you want to know more about Linux/UNIX basics, shell commands, and file/directory management, start with dthe Information Technology Department's web pages on UNIX and the tools associated with it

```
import java.util.Scanner;

public class Factorial {

  public static Scanner keyboard = new Scanner(System.in);

  // YOUR METHODS WILL GO HERE.

}
```

2. **Write a Factorial calculating method:** Write a method named `fact` that accepts, as a parameter, an integer n. This method should then calculate the $n!$ (the n^{th} factorial number) and return it. That is, your method should begin:

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

   Fill in the body of this method with appropriate code to perform the needed calculation. You should be using *recursion*—that is, a method that calls itself—to perform this calculation. Notice the return type of `long` for this method. Since factorial numbers grow to be quite large rather quickly, using a larger integer type is important and useful here.

   As always, feel free to write a temporary `main` method to test your `fact` method. Have `main` pass some known value for n to your `fact` method, and the print the result that is returned. Verify that the value computed by your `fact` method is correct.

3. **Write a factorial search method:** Even a `long` integer has a limited range. Any number larger than about 8 quintillion cannot be stored in such a variable. In fact, if you take the largest positive value that can be stored in a `long` integer and then add 1 to it, the value will *wrap around* into the negative numbers. Consequently, if we try increasing values of n on our `fact` method, eventually we will find a value (let's call it n_{max}) that yields the **largest** factorial number that can be correctly contained in a `long` integer variable—let's call that one f_{max}. That is, if we pass $n_{max} + 1$ to `fact`, the value returned will appear negative, which is, of course, incorrect.

   We seek n_{max}. **Write a method** named `findMaxFact` that accepts no parameters and returns the value of n_{max}—that is, the largest n for which your `fact` method returns a correct result. The method should begin like this:

   ```
   public static int findMaxFact () {
   ```

4. **Write your `main` method:** If you previously wrote a `main` method to test your previous work,[4], now is the time to delete that code and start the body of `main` anew. Specifically, you should write `main` so that it calls your `findMaxFact` method to obtain n_{max}. It should then call `fact` directly, passing it n_{max} to obtain that largest correct factorial number, f_{max}. Finally, `main` should print both numbers, like so:

[4] Always a good idea

```
fact(22) = 712371238124
```

[**Warning:** $n_{max} \neq 22$, and $f_{max} \neq 712371238124$. Your program should emit the correct answer in that *format*, but not using those exact values, which are incorrect.]

2 Finding part B

Once you have the correct f_{max} value, use it to fill in for `XYZ` in the following web address that you should provide to a web browser:

```
https://www.cs.amherst.edu/~sfkaplan/courses/2013/spring/COSC-111/
projects/XYZ.html
```

[**Warning:** Some people, on Project 2, encountered difficulties when copying-and-pasting this web address from this PDF document into their browser. In particular, the *tilde* character (~) seems not to be copied correctly on some types of computers, and thus mangles the web address. Specifically, another character that also looks quite *tilde*-ish tends to be pasted, but it is not the same, "true" *tilde* produced by typing the character on the keyboard yourself. So, to be truly cautious about this problem, type the entire web address by hand.]

This web page will redirect you to part B of this project.

3 How to submit your work

As usual, use the `cs111-submit` command:

```
cs111-submit project-3a Factorial.java
```

Part A is due on Sunday, February 24, at 11:59 pm.