CSC111 Introduction to Computer Science Fall 2017
Lab 1

Note: The instructions here assume you are working on **gottlieb**. For this project, you are free to work on your own laptop in your preferred environment. However, you are encouraged to work on **gottlieb**, especially if you plan to be a Computer Science major or minor; you will gain experience in the UNIX (Linux) environment. If you are working on **gottlieb**, you should review the Unix tutorial on:

http://menehune.opt.wfu.edu/Unix/unixtut/index.html

Working on gottlieb:

Make a directory named `Lab1` ; keep all work for this assignment in that directory. In the following parts, you will write several distinct Java programs. Remember to put the lab number and part labels (e.g, "Lab1, part A", "Lab1, part B",... etc.) and **your name** in a comment at the beginning of each program file.

Writing a Program (on gottlieb) See the handout entitled "**Getting Started on Gottlieb**" for help getting started with **gedit** and the command line.

Getting Started with Java

1. Converting Fahrenheit Temperature to Celsius

   (a) Write a Java program to take input (from the keyboard) consisting of a temperature in Fahrenheit. Output the equivalent temperature in Celsius. The formula is:
   $$C = \frac{5}{9}(F - 32)$$
   Be mindful of the distinction between integer division and floating point division. The variables F and C should be type **double**. Use the Scanner class method **nextDouble()** to read the Fahrenheit temperature. Name your Java source[1] code "`ftoc.java`".

   Test your program with $F = 70$, again with $F = 32$, $F = 212$, $F = 0$, and finally with $F = -40$. What Celsius temperatures do you get ?

   (b) Modify your program from part 1a to check for valid input. Use **exception handling** (i.e., use a **try-catch** control structure) to determine if the input is acceptable. If the input is unacceptable (e.g., the user enters "seventy" instead of "70"), your program must catch the exception and print an error message.

   Hint: Run your code from part 1a with text input 'seventy' to discover what type of exception is generated. Be sure to include an **import** *statement to import the definition of the exception your program will catch.*

   Name your source code "`ftoc_check.java`".

[1] According to the Google Style Guide regarding file names: "Filenames should be all lowercase and can include underscores (_) or dashes (-). Follow the convention that your project uses. If there is no consistent local pattern to follow, prefer "_". Java source files should end in **.java**. You may also see so called "camel case" used as an alternate standard for file names.

Do not use spaces, punctuation, or special characters other than underscore in your file names. These characters have meaning to the UNIX shell and will cause confusion if used in file names.

2. **Using loops and programming logic:** A **perfect number** is a number that is equal to the sum of its positive proper divisors. For example, $1 + 2 + 3 = 6$. Notice that 1, 2, and 3 are the proper divisors of 6. The number 6 itself is not included in the definition of a "proper divisor".

   (a) **Using a loop:** Write a Java program that reads in integer N from the keyboard and decides if it is a perfect number. Recall that a number k is a divisor of N if and only if `( N % k ) == 0` . If the input number is a perfect number, print a message (including the number) stating that the number is a perfect number. If the input number is a not perfect number, print a message (including the number) stating that the number is not perfect. To test your program, note that 6 and 28 are perfect numbers.

   Name your source code file "**perfect.java**".

   (b) **Using two nested loops:** Modify your program from part 2a to find all perfect numbers less than 1000. As you find each new perfect number, store it in the next available position in an array. After you've checked all numbers up to 10000, print the array of the perfect numbers you have found. Note: There are fewer than 11 perfect numbers less than or equal to 10000. *Hint: You can allocate an array size 10.*

   Your program should produce the following output:

   ```
   Perfect numbers: 6 28 496 8128
   ```

   Name your source code file "**perf_list.java**".

Turn in: Change to your home directory. Create a file named "lab1.tar" using the command:

```
tar cf lab1.tar Lab1
```

Upload the file "lab1.tar" using `sftp` to your account on telesto. (telesto.cs.wfu.edu)