No matter how one may think himself accomplished, when he sets out to learn a new language, science, or the bicycle, he has entered a new realm as truly as if he were a child newly born into the world. ~ Frances Willard, *How I Learned to Ride the Bicycle*

CS1010 Programming Methodology
Week 3: Computational Thinking/Algorithms

To students:

Discussion sessions are small group forums where **you** present and discuss answers to the questions, and raise your doubts for clarification. Answers come from you, not from your discussion leader (DL). Your DL and classmates can then build on your contribution to bring up any intricate points that have been overlooked, or to correct any misconceptions.

The success of discussion sessions hinges very much on (1) your **PREPARATION** beforehand, (2) your **ACTIVE PARTICIPATION** in class, and (3) after-class **REVISION**. Unless otherwise instructed, you do not need to submit your work for grading.

Please cooperate with your DL to work towards a fruitful and enriching learning experience.

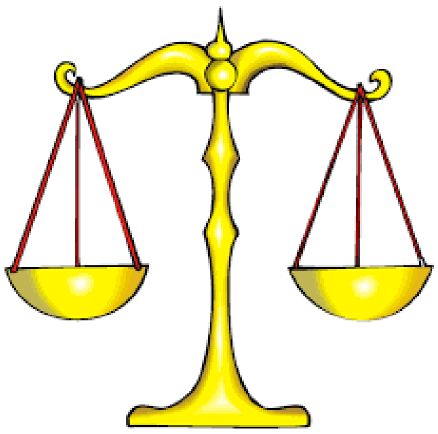
Due to time constraint, sometimes not all the questions in the Discussion Sheet are discussed in class. Your DL has the discretion to choose the questions (or you may request your DL to discuss certain questions) or set his/her own questions for you. You may continue to discuss the questions on IVLE forums after class.

Also, why limit yourself only to the exercises here? You may find exercises from other sources for your own practice.

I. Computational Thinking

1. Santa's Dirty Socks

One very common technique in Computer Science is "Divide and Conquer". This comes well under the concept of "Decomposition" in Computational Thinking, for "Divide and Conquer" is an algorithm design paradigm which works by repeatedly breaking down a problem into two or more sub-problems of the same type, until these become simple enough to be solved directly. The "Divide and Conquer" paradigm gives rise to many famous algorithms in Computer Science.



For today, you are told that Santa's elves have mistakenly wrapped a pair of dirty socks. Time is running out and Santa is about to deliver the 1024 wrapped boxes of socks. Which one of the 1024 boxes contains the dirty socks? There is no time to unwrap all the boxes, find the one with the dirty socks, and wrap the rest up again. There is a balancing scale where you can put two boxes, and the heavier one which contains the dirty socks can be identified. But there isn't enough time to put all the boxes one at a time on the balancing scale and you can put any number of boxes onto the scale. How can you help Santa to locate the dirty socks as quickly as possible?

(Do not google! The answer can easily be found on the Internet!)

II. Writing Pseudocode

You may assume that for the problems below, the input data have been read into the list. That is, you need not concern yourself about how to read the data into the list.

2. Given a list of *N* integers, find out how many of them are negative.
   An algorithm is shown below, which includes all 3 **control structures**: sequence, selection and repetition. For a list, we may use the subscript (such as a_k) to represent individual items in the list.

   Study the algorithm and trace it with some test data.

   Let the list of integers be a_1, a_2, … a_N.

```
for $k$ from 1 to $N$
    if ( $a_k < 0$ ) countNeg ← countNeg + 1;
print countNeg
```

   a) Can you spot an error in the algorithm and correct it?
   b) If the problem is changed to "find out how many of the integers are positive", how would you modify the pseudocode?
   c) If the problem is changed again to "find out how many of the integers are odd", how would you modify the pseudocode?
   d) If the problem is yet again changed to "find out the sum of the odd-indexed integers" (i.e. $a_1 + a_3 + a_5 + …$), how would you modify the pseudocode?

3. *N* white and black balls are arranged in a row. The example below shows a case of *N*=7.

   ● ● ○ ● ○ ● ○

   Of course, we shall choose an appropriate notation to represent our objects (balls): **B** for black ball and **W** for white ball. So the above may be presented as BBWBWBW.

   The task is to determine the least number of 'swaps' you need to shift all the white balls to the left of all the black balls, subject to the condition that you may only swap two neighbouring balls. For our example above, the least number of swaps you need is 9. (Work it out yourself! Try out other examples as well.)

   Write an **algorithm** (not a C program!) to compute the answer. For this problem, a good algorithm needs not even carry out the swapping.

   To help you start, here's a suggestion:

In solving algorithmic problem, we must name our data. Let’s name the balls, from left to right, B_1, B_2, B_3, …, B_N for N balls.

III. Using CodeCrunch

Refer to the “Introduction to CodeCrunch” on the module website (“CA” → “Labs”).

You are to familiarize yourself with CodeCrunch. Though Lab #0 will not be graded, you are to submit it before this discussion session to indicate that you know how to use CodeCrunch. Your DL will check that you have submitted. Clear your doubts on CodeCrunch with your DL.

Every week, practice exercises will be mounted on CodeCrunch for you to try on your own.