ICT167 - Laboratory Two Notes

A reminder: many of the exercises in the Lab Practice worksheets say to modify a previous one. When you have your exercises assessed, you need to show each of them as separate source code. Also, you should backup each of the exercises in case of mishap.

As System.out is globally available for output, theoretically System.in can be globally available for input. Analogy: printf and scanf, cin and cout. Therefore, the following declaration for the Scanner can be global to a class:

```
static Scanner inputStream = new Scanner(System.in);
```

Review loops: pre / post testing, fixed iteration; when it is appropriate to use them, and how to use them correctly.

Exercise Tips

1. For this exercise, make sure you use modular design; i.e. using methods (one method per task). Do **not** do everything in main. Remember what you learned about modular design and implementation in previous units. Thus you should use an input method, a method to process the input (use a `default` case to handle incorrect input), calculation methods as needed, and a display method.

   Part c: Should check input to see which of m and n is smallest (m should be smaller); swap them if necessary. When you wish to print 5 items per line, do **not** use nested loops; use one loop and the modulus operator:

   ```
   if ((i % 5) == 0)
   ```

   Part d: the statement "the sum of any two sides must be greater than the third side" means that you must check 3 combinations of the triangle sides. i.e. a+b > c and b+c > a and a+c > b. This is because you do not know what the user will input, or in what order the input numbers will be entered.

   Part e: you can search for a prime number **algorithm** online and code it. Search for "math algorithm prime numbers". Do **not** just copy code. At the end of this document there are 3 example math algorithms.

   N.B. read the material on strings (Q1 of lab practice 2 worksheet) to isolate a character input.

   N.B. it is almost always essential to validate input before using input values in calculations. The most logical place to validate input is when actually getting the input (i.e. within the input method), though this is not always the case. In assignments, even if input validation is not specified, you must do so.

2. Again, make sure you use modular design; i.e. using methods (one method per task). Option ‘f’ should be performed by calling as many methods as needed to achieve the requirements. You will need to declare a 10 element integer array. Eg:

   ```
   int arr = new int[10]
   ```

You will need to use a loop to populate the array. Each iteration requires an input from the user, with the entered value stored into the array at the appropriate index:

```
loop i=0 -> 10
    arr[i] = integer input
    increment i
```

Then have separate methods to perform each of the required calculations.

Remember to make your solution modular.

Prime Numbers:

A prime number is one that is evenly divisible only by 1 and itself. Thus 1, 2, and 3 must be prime. There are many other prime numbers that can be determined by math algorithms.

The simplest primality test is as follows:

- Given an input number n, check whether any integer m from 2 to n/2 divides n.
- If n is divisible by any m then n is composite, otherwise it is prime.

```
function isPrime(m, n)
    m = n/2
    flag = true
    loop from i = 2 to m AND flag == true
        if ((n % i) == 0)
            flag = false
    return flag
```

Other math algorithms that use the Greatest Common Divisor (gcd) are:

```
function gcd(a, b)
    while a != b
        if a > b
            a = a - b
        else
            b = b - a
    return a

function gcd(a, b)
    while b != 0
        var t = b
        b = a modulus b
        a = t
    return a
```