CS1010E Programming Methodology
Semester 1 2016/2017

Week of 5 September – 9 September 2016
Tutorial 3
Control Structures – Selection and Repetition

1. Given the following program fragment.

```
int i=1;

while (i > 0) {
   i = i + 1;
}

printf("%d\n", i);
```

   (a) What do you think will happen?
   (b) Run the program, observe what happens and make your own deductions.

2. Given the following program fragment.

```
int i, n, count2=0, count3=0, count5=0;

scanf("%d", &n);

for (i = 0; i <= n; i=i+1) {
   if (i%5 == 0) {
      count5 = count5 + 1;
      if (i%3 == 0) {
         count3 = count3 + 1;
      }
   } else {
      if (i%2 == 0) {
         count2 = count2 + 1;
      }
   }
}

printf("%d %d %d\n", count5, count3, count2);
```

   (a) Perform a timeline trace of the `count2/3/5` variables from $i = 0$ to $i = 30$.
   (b) Using the trace in question 2a, predict the output of the program for input `321`.
   (c) Verify your prediction by running the program.

3. Use loop constructs to generate the following number sequences:

   (a) Write a program that reads as input an integer $n (\geq 0)$ and outputs the sum of n terms of the series $1, 2, 3, 4, 5, \ldots$.
   (b) Rewrite the program to output the sum of n terms of the series $1, 3, 5, 7, 9, \ldots$.
   (c) Rewrite the program to output the sum of n terms of the series $1, -3, 5, -7, 9, \ldots$.
   (d) Rewrite the program to output the n terms approximation of π given by:

$$\frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \ldots$$

   (e) Observe that as n increases, the π approximations get progressively closer. Rewrite the program to read as input a floating-point tolerance value δ. The program outputs the i^{th} approximation π_i for which the difference between the two most recent approximations (π_i and π_{i-1}) does not exceed δ. What are the π approximations for $t = 0.001$, $t = 0.0001$ and $t = 0.00001$?

4. Write a program that takes in two positive integers and returns the greatest common divisor (*gcd*) of the two integers. For example, the *gcd* of 539 and 84 is 7. Two algorithms for determining the *gcd* is given below.

   (a) Set a variable `gcd` to be the smaller of the two values. If this value of `gcd` completely divides the two numbers, then return this value as the *gcd*. Otherwise, reduce the value of `gcd` by one and repeat the test.
   (b) We apply the Euclidean algorithm. Let the two values be `a` and `b`. Replace `b` with the result of `a%b`, and `a` with the original value of `b` (before the replacement). Keep doing this until `b` becomes zero; the value of `a` is the *gcd*.

5. Every positive integer greater than one can be expressed **uniquely** as a product of primes. For example, the number 10 can be expressed as 2×5 and the number 20 can be expressed as $2 \times 2 \times 5$. This process is sometimes known as *prime factorization*.

   Write a program that reads an integer n (> 1) as user input and outputs all prime factors of n in increasing order. Sample runs of the program are given below. User input is underlined.

```
Enter n (> 1): 2
2
```

```
Enter n (> 1): 10
2 x 5
```

```
Enter n (> 1): 20
2 x 2 x 5
```

```
Enter n (> 1): 1010
2 x 5 x 101
```

```
Enter n (> 1): 223092870
2 x 3 x 5 x 7 x 11 x 13 x 17 x 19 x 23
```