National University of Singapore
School of Continuing and Lifelong Education

TIC1001: Introduction to Computing and Programming I
Semester I, 2019/2020

Tutorial 1
Basics of C and Control Structures

1. Being able to trace through the execution of a program without running it is known as *program tracing*. Program tracing involves building up a *mental model* of program execution in your mind (or on paper). By tracing a program, you will gain a stronger understanding of the program execution flow and improve your programming skill. Consider the following program:

```c
#include <stdio.h>

int main(void) {
    int cur, prev1=1, prev2=1;

    cur = prev1 + prev2;
    prev2 = prev1;
    prev1 = cur;

    cur = prev1 + prev2;
    prev2 = prev1;
    prev1 = cur;

    cur = prev1 + prev2;
    prev2 = prev1;
    prev1 = cur;

    printf("cur is %d; prev1 is %d; prev2 is %d\n", cur, prev1, prev2);

    return 0;
}
```

   Trace the execution of the program, paying particular attention to variable declarations and changes to the values of the variables. Verify your program trace by including your own *debugging* `printf` statements at strategic points in the program to track the value changes of variables.

2. Assume there are three integers `a`, `b` and `c` already defined in our program, and we want to assign the value of the largest integer to variable `max`.

   For each of the following code fragments below, fill in the missing parts so that the code will fulfil the given task.

(a)
```
if (...) {
    max = a;
} else {
    if (...) {
        max = b;
    } else {
        max = c;
    }
}
```

(b)
```
max = a;
if (...) {
    max = b;
}
if (...) {
    max = c;
}
```

(c)
```
if (a > b) {
    if (...) {
        max = a;
    } else {
        max = ...
    }
} else {
    if (...) {
        max = b;
    } else {
        max = ...
    }
}
```

3. Study the following program fragment:

```
int foo(int a) {
    if (a > 0)
        if (a >= 1000)
            a = 0;
        else
            if (a < 500)
                a = a * 2;
            else
                a = a * 10;
    else
        a = a + 3;
}
```

Deduce the functionality of the program fragment and re-compose the nested `if..else` statements into a form that is easier to understand.

4. 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.

5. Given the following program fragment:

```
int n=321, count2=0, count3=0, count5=0;

for (int 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 variables `count2`, `count3` and `count5` from $i = 0$ to $i = 30$.

   (b) Using the trace above, predict the output of the program.

   (c) Verify your prediction by running the program.

6. 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 are 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 remainder 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.