National University of Singapore
School of Continuing and Lifelong Education

TIC1001: Introduction to Computing and Programming I
Semester I, 2018/2019

Tutorial 3
Functional Abstraction

1. In tutorial 1, we performed a trace of a program containing only the main function. Extend your mental model to include traces of program execution involving more than one function using the program below.

```c
#include <stdio.h>

int f(int x, int y);

int main(void) {
    int x = 3, y = 4;

    x = f(x,y);
    y = f(x, f(y,x));
    printf("x = %d; y = %d\n", x, y);
    return 0;
}

int f(int x, int y) {
    return x*10 + y;
}
```

   Keep in mind the following notions while you trace.

   - function call with evaluated arguments
   - function activation with parameter declaration
   - pass-by-value
   - lexical scoping
   - function termination upon return

2. The basis representation theorem states the following:

   Every $n \in \mathbb{Z}^+$ can be uniquely expressed as a sum of terms (a_i) such that

$$n = \sum_{i=0}^{k} a_i b^i, a_k \neq 0$$

   where b is the base of the number.

   For example, 130_{10} in base 10 can be expressed as 1010_5 in base 5 since

$$1 \times 10^2 + 3 \times 10^1 + 0 \times 10^0 = 1 \times 5^3 + 0 \times 5^2 + 1 \times 5^1 + 0 \times 5^0$$

(a) Implement a function `void print_10_to_b(int n, int b)` which takes in a number n in decimal (base 10) and prints the equivalent number in base b, for $2 \leq b \leq 10$.

(b) Implement a function `void print_b1_to_b2(int n, int b1, int b2)` that takes in a number n in base b_1 and prints the equivalent number in base b_2, with $2 \leq b1, b2 \leq 10$.

3. A function can be viewed as a black box. All you need to know are the arguments it takes as input and what its output is.

(a) One way of calculating the area of a triangle is using the formula $area = \frac{1}{2} \times base \times height$.

Implement a function `area1` that calculates and returns the area of any given triangle using this formula.

Decide what arguments the function requires as input.

```
double area1(/* your arguments */) {
    // Return area of the triangle using the formula
    // area = 1/2 * base * height.
}
```

(b) Another way of calculating the area of a triangle with sides `A, B, C` is using the trigonometric ratio sine to get $area = \frac{1}{2} \times A \times B \times sin(AB)$, where `AB` is the included angle between sides `A` and `B`.

The `sin` function is provided by the `math.h` library. You can call it by using `sin` after including the line `#include <math.h>` at the top of your source file.

Implement a function `area2` that calculates and returns the area of any given triangle using this formula.

Decide what arguments the function requires as input.

```
double area2(/* your arguments */) {
    // Return area of the triangle using the formula:
    // area = 1/2 * A * B * sin(AB).
}
```

(c) We can also calculate the area of triangle using Heron's Formula,

$$area = \sqrt{s(s-a)(s-b)(s-c)} \text{ where } s = \frac{a+b+c}{2}$$

Implement a function `area3` that calculates and returns the area of any given triangle using Heron's Formula.

Decide what arguments the function requires as input.

```
double area3(/* your arguments */) {
    // Return area of the triangle using Heron's formula
}
```

(d) All three functions calculate the same result. Can they be directly substituted for each other? Why?

4. Implement a function that takes in three integers, and rearranges them in ascending order. For example, after the following code snippet is executed:

   ```
   int a = 5, b = 7, c = 2;
   sort(&a, &b, &c);
   printf("%d, %d, %d", a, b, c);
   ```

   the output will be `2, 5, 7`.

   Assuming that an in-place `swap` function that takes in two variables via pointers and swap their values is available, try implementing your sort function without declaring any new variables.

   Try it using C++ references as well.

5. For this question, we will use an integer to represent the amount of money in a bank account. Implement the following functions:

   - `void deposit(int *account, int amount)` that takes in the bank account and an amount of money, and increments the account by the amount.
   - `bool withdrawl(int *account, int amount)` that takes in the bank account and an amount of money, and decrements the account by the amount if there is sufficient money in the account. It returns true if the account was successfully decremented and false if not.
   - `bool transfer(int *from, int *to, int amount)` that takes in two bank accounts and an amount of money, and transfers the amount from the first account to the second, if there is sufficient money in the first account. It returns true if the account was successfully decremented and false if not.

   As a practice on functional abstraction, you should reuse functions as much as possible.

   When done, try using C++ references too.