Principles of Programming Languages 242

Home Assignment 1

By **Guy Moalem** and **Hadar Gantz**

Part 1: Theoretical Questions

Question #1

(a) Explain the following programming paradigms:

i. Imperative

The Imperative Programming Paradigm is a programming paradigm in which the programmer uses a sequence of commands to change the program's state. In this paradigm, we describe exactly how the program should operate.

ii. Procedural

The Procedural Programming Paradigm is a programming paradigm in which we use procedures to pack blocks of codes. These procedures can call each other. The program then consists of statements and function calls. This paradigm is a special case of the Imperative Programming Paradigm.

iii. Functional

The Functional Programming Paradigm is a programming paradigm in which programs are built using functions and their composition. In this paradigm, we evaluate expressions rather than simply execute commands. Functions do not have "side effects" and data is immutable.

(b) How does the procedural paradigm improve over the imperative paradigm?

Code is more readable when using the Imperative paradigm – rather than jumps in the code or long blocks of code that has no meaning to a normal human being, using the Imperative programming paradigm allows us to name blocks of code. That way we can read actual meaningful words/phrases and understand, at least to some extent – what is going on in the program. Moreover, we can pass arguments to cause a change in a certain code's behavior – this means we do not have to copy-paste large chunks of code or change values of variables when, for example, we do testing.

(c) How does the functional paradigm improve over the procedural paradigm?

It is even easier to test programs that use the Functional paradigm – during the running of the function, variables do not change their value, making it easier to predict what'll happen and how to fix bugs in the code/logic. The immutability of data prevents bugs that are caused when changing it. This implies more safety when running several processes (that may use shared resources) simultaneously. Functional programming also allows us to use higher order functions. In addition, because we mainly use functions or compositions of functions, the code is more readable and flexible.

Question #2

```
const getDiscountedProductAveragePrice: (inventory: Product[]) => number =
(inventory: Product[]) =>

    inventory

    .filter(product => product.discounted)

    .reduce(

    (acc: number, curr: Product, index: number, array: Product[]) =>

    acc + curr.price / array.length, 0);
```

Question #3

(a) The type of the expression `(x, y) => x.some(y)`
is `<T>(arr: T[], cond: (c: T) => boolean) => boolean`.

(b) The type of the expression `x => x.reduce((acc, cur) => acc + cur, 0)`
is `(arr: number[]) => number`.

(c) The type of the expression `(x, y) => x ? y[0] : y[1]`
is `<T1, T2>(x: T1, y: T2[]) => T2`.

(d) The type of the expression `(f, g) => x => f(g(x+1))`
is `<T1, T2>(f: (y: T1) => T2, g: (x: number) => T1) => x: number => T2`.