**CO2412 Computational Thinking**

# Section 1

## Computational Thinking Concepts

One of the Computational Thinking concepts described by Brennan and Resnick is **Sequences.** It is a “key concept” that represents a step or instructions that are executed by the computer. A sequence “specifies the behavior or action that should be produced”.

**Loops** are also one of the concepts mentioned. These are a mechanism for running sequences of instructions until a specified condition is met.

Brennan and Resnick also described **Events** a computational thinking concept as “one thing causing another to happen”. This could be the movement of an input device (a mouse clicking a button) or a signal from an output device (like a printer saying it is out of ink, prompting a pop-up warning to appear).

Furthermore, **Parallelism** was included. A computer language is said to support parallelism when it lets the computer execute multiple sequences at the same time.

**Conditionals** let the program make decisions based on meeting certain conditions. ‘If’ blocks allow the program to take two or more different paths based on the output of the evaluated condition.

The programmer can perform manipulation of various data types using **Operators**. This includes addition, multiplication, division as well as more advanced calculations like trigonometry.

The last of the concepts is **Data**. This involves “storing, retrieving and updating values”. This can be represented by a variable inside the program, or a file written to storage which contains values that the program can read.

## What does Abstraction mean?

In a computational thinking sense, abstraction is when aspects of a large and complex problem are modelled and appropriately represented to allow people to go through said problem more easily. It allows the programmers to change a complex system without the need to understand it at a very low level.

## Why multiple levels of abstraction?

This is necessary for programmers to be able to fully model a problem and a solution or algorithm. It allows programmers to code an effective solution as they can break down a large problem into smaller and smaller pieces that can be solved by writing computer code, whereas a large problem could not be solved easily.

## What do I understand by decomposition?

My understanding is that with decomposition, a large problem can be broken down into smaller, more manageable chunks that can be solved more quickly. Decomposition in programming can be achieved by using small functions that solve a part of a larger problem. Beyond programming, moving to a new house can be a large problem to overcome. One can then start breaking it down into smaller chunks, such as organising transport to move one’s items over, changing address and utilities, packing everything up. These tasks can then be broken down further into finding contact information for a moving company, looking at which services one is registered where change of address is necessary, or buying bubble wrap to ensure items remain safe after packing up.

# Section 2

Setting up a development environment.

# Section 3

A C# reminder.

# Section 4

Variation in the number of iterations that BogoSort performs varies, as each iteration is randomly generated.

BogoSort has an algorithmic efficiency of n-factorial, O(n!)

Recursion is not a good approach for a BogoSort implementation, as there is a limit to how many stack frames (“data structures containing subroutine information”) can be stored in the stack. Each currently running function that has not yet returned, is a frame in the stack.

# Section 5

Insertion sort algorithm in C#

## How does insertion sort work?

Like a human with a selection of cards in their hand, insertion sort starts at the second element in the array (as the first is already sorted), and for every element it compares it to the previous element, swapping them if the current element is smaller (if the array is to be sorted in ascending order).