IDATA2302 Notes

# Foundations

## Computations

A computation is a transformation of data: It consumes some data and produces some data. Computational problems are problems which can be solved using computations.

### Algorithms

An algorithm has inputs and outputs. It consumes some data and produces some results. It is finite, meaning it cannot have infinite amount of steps. It is well defined, and each step is non-ambigous. And should be effective.

### Data Structures

An algorithm is a sequence of steps that manipulates data to solve a problem. The place where the produced data is saved is called a data structure: Organize manipulated data.

### How to describe an Algorithm?

* Flowcharts
* Natural language
* Pseudo code
* Using a program

## Computer

An ideal computer is a random access machine and underpins all imperative languages such as C, Java, Python etc.

### Random Access Machnes

RAM is an abstract machine, a model of computation.  
Components of RAM:

* An I/O device that the machine uses to exchange sequence of symbols with the user (keyboard, mouse, etc)
* Memory with infinite cells. Each memory cell can contain an arbitrary long sequence of symbols (fe numbers) and has an unique identifier.
* CPU which carries out arithmetic and logical operations. (Accumulator and holds intermediate results. Instruction pointer, containsthe address where the next instruction is located)

To use ram, we need to express our algorithm as a single list of instructions. Each instruction is a pair of natural numbers, say (1, 12). The first number us the operation code (OP) and indicates which action the machine must execute. Second number is the operand and details what piece of data the machine must manipulate. So the pair (1, 12) means LOAD 12, because 1 is LOAD. This will override the accumulator register with the sequence “12”. Instructions are stored in the memory. This means a program is just a long list of numbers.

### Programming languages

Assembly language is meant to convert human suited code to machine code. Symbolic names for memory addresses, so that we can refer to them with something that is meaningful to us. Instructions mnemonics so that we can refer to machine instructions by name rather than by operation code. Memory layout that clearly delineates between memory cells that store program instructions from those that store program, data.

The first thing the RAM compiler does, is breaking long arithmetic expressions into a sequence of binary assignments, according to the precedence rules of arithmetic operators.’

### Correctness Functional Correctness

For an algorithm to be totally correct:

* It must terminate at some point. In other words, the RAM must reach a HALT instruction, and stop. If it does not always terminate, the algorithm is, at best, partially correct.
* When it terminates, it must produce a correct output for all possible valid inputs. A correct output satisfies the constraints set by the problem. An algorithm is thus incorrect if one can fine at least one set of inputs for which the algorithm output is wrong.

A pre-condition is what we assume to hold before we execute instructions. A post-condition is what we assume to hold after we executed instructions.

### Formal Proofs

A formal proof is not a natural language argument. It is a calculation that follows precise rules. It relies on a formal notation so that proofs can be checked mechanically-by a so-called proof assistant. The syntax and semantics of the language are the basis of the deduction system we use to reason about the correctness of algorithm.

When reasoning about correctness, loops are the main obstacle.

* We tackle partial correctness by identifying a loop invariant, which is true, after and during the loop.
* We tackle termination by identifying a loop variant, which is a quantity that decreases with each iteration and can only be negative after the loop.

Testing is very useful in practice, but it comes with important theoretical implications:

* It cannot show the absence of bugs, only their presence.
* It cannot show non-termination.

## Algorithm Analysis

### Modelling Algorithm Efficiency

We model algorithms’ efficiency as a function from the input size to the measure of interest, be it time, space, or something else. This function allows us to make predictions. Input sizes can be described however we see fit.

# Recursion

## Procedure Calls

Procedure call is an early construct that helps us progress from assembly code towards higher level languages.

### The Call Stack

As developers, we see the call stack every time our programs crash. We see if after the exception. Starting from the bottom, we find out precisely where our program crashed.

## Recursion

Recursion is about self-similar structures. Applied to data types, that yields types that refer to themselves.

Recursive algorithms consume significantly more memory than their iterative equivalent, because of the underlying call stack, which grows as the recursion deepens.