***Algorithms and Data structures***

***Topic: Getting Started***

***Week: 1***

***Takeaways***

* *Takeaway 1*
  + *A computation transforms of a sequence of symbols into another.*
* *Takeaway 2*
  + *A computational problem maps input data to output data. Mathematical functions clearly capture how these inputs map to the appropriate output*
* *Takeaway 3*
  + *An algorithm is a finite sequence of non-ambiguous instructions, which processes its inputs to produce the solution of a computational problem. To work efficiently, algorithms store their data into dedicated data structures.*
* *Takeaway 4*
  + *There is an direct relationship between the actions we stipulate in an algorithm and the capabilities of the computer we use to execute it.*

***1 Computation***

* ***CS is all about computation***
  + *A computation is transformation of data*
    - *It consumes some data and produces some data*
* *What is Data?*
  + *Data is an overload term*
    - *Example:*
      * *Texts*
      * *Numbers*
      * *Pictures*
      * *Audio recording*
  1. ***Computational Problems***
* *Computational problems*
  + *Are problems which can be solved by a computation*

***1.2 Algorithms***

* *An algorithm has inputs and outputs. It consumes some data and produces some results.*
* *An algorithm is finite: it must terminate at some point and cannot have an infinite number of steps.*
* *An algorithm is well-defined, and each step is non-ambiguous.*
* *An algorithm is effective and can be carried out by either a machine or human with pen and paper in a finite amount of time. Each step must be feasible.*

*Do not confuse algorithm and computation. Algorithm is the list of steps to follow whereas the computation is what happen when a computer goes through a particular addition.*

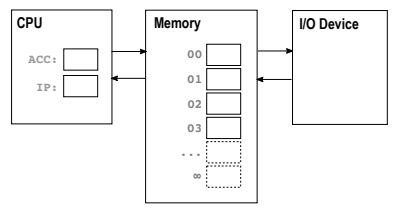
1. *How to Describe an Algorithm*

* *Using natural language*
* *Using Flowcharts*
* *Using Pseudocode*
* *Using a program*

1. *RAM*

*Takeaways*

* *Takeaway 1*
  + *The RAM model defines the actions that the machine understands.*
* *Takeaway 2*
  + *There are many ways to encode a given piece of pseudocode into machine code. It is important to understand-at a high-level-how a compiler does that.*
* *Takeaway 3*
  + *A program is an algorithm encoded using a programming language. This program can be converted into machine code for execution by a computing device.*
* *Random Access Machines*
  + *RAM is an abstract machine, a “model” of computation:*
    - *It is a blueprint that carries out computation.*
* *Architecture*
  + *A random-access machine is a machine that mimics the behavior of a real computer. It manipulates data encoded as symbols*
  + *Three following components:*
    - *An I/O device that the machine uses to exchange sequence of symbols with the user. We can think of this as a screen and a keyboard for example.*
    - *A memory which contains infinitely many cells. Each memory cell can contain an arbitrary long sequence of symbols and has a unique identifier which will allow to read and write anywhere in this memory.*
    - *A central processing unit (CPU) the carries out arithmetic and logical operations (addition, subtraction, comparisons, etc.). This CPU has two registers, namely ACC and IP which can both hold any arbitrary long sequence of digits.*
      * *ACC is the accumulator and holds intermediate results*
      * *IP is the instruction pointer and contains the address where the next instruction is located.*

**

*Algorithm Correctness*

*Takeaways*

* *Takeaway 1*
  + *Provided with valid inputs, an algorithm…*
    - *is partially correct when it never terminates with an incorrect result but may not terminate.*
    - *Is totally correct when it always terminates with a correct result,*
* *Takeaway 2*
  + *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.*
* *Takeaway 3*
  + *The syntax and the semantic of the language are the basis of the deduction system we use to reason about the correctness of algorithm.*
* *Takeaway 4*
  + *Reasoning about correctness at the RAM instruction level is possible but tedious. When possible, we will reason at the pseudo-code level.*
* *Takeaway 5*
  + *When reasoning about correctness, loops are the main obstacle.*
    - *We tackle partial correctness by identifying a loop invariant, which is true, before, 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.*
* *Takeaway 6*
  + *Testing is very useful in practice, but it comes with important theoretical implications:*
    - *It cannot show the absence of “bug”, only their presence*
    - *It cannot show non-termination.*
* *Takeaway 7*
  + *Proving, testing, and debugging all require a detailed understanding of the algorithm. The concepts we use to build proof directly support debugging programs:*
    - *Pre-conditions are checked explicitly at the beginning of procedures.*
    - *Invariants are checked within the procedures using assertion*
    - *Post conditions are checked in the automated tests.*
* *Functional Correctness*
  + *It must terminate at some point. In other words, the RAM must reach a HALT instruction and stops. If 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. A correct output satisfies the constraints set by the problem. An algorithm is thus incorrect if one can found at least one set of inputs for which the algorithm output is wrong.*

***Topic: Efficiency***

***Week: 2***