2017

AQA A2 Computing Notes

AQA

St Paul's School

2/12/2017

Contents

1 Fundamentals of programming

1.1 Programming

1.1.1 Data types

Understand concept of a data type.

Understand and use the following:

- Integer
- Real/float
- Boolean
- Character
- String
- Date/time
- Records
- Arrays
- **Pointer/reference** (new to A-level)
    - Variables declared as pointers are used as stores for memory addresses of objects created at runtime (not always supported)

Define and use user-defined data types based on language-defined (built-in) data types

1.1.2 Programming concepts

Use, understand and know how the following can be used in programs:

- Variable declaration
- Constant declaration
- Assignment
- Iteration
- Selection
- Subroutines (procedures/functions)

Use definite and indefinite iteration (with the condition at start OR end of the loop)

Use nested selection and iteration structures

Use meaningful identifier names as it is important for memory and understanding of a human reading

Three key concepts: sequence, iteration, selection

1.1.3 Arithmetic operations

Be familiar with and able to use:

- Addition
- Subtraction
- Multiplication
- Real/float division
- Integer division (including remainders)

- Exponentiation
- Rounding
- Truncation

1.1.4 Relational operations

Be familiar and able to use:

- Equal to
- Not equal to
- Less than
- Less than or equal to
- Greater than
- Greater than or equal to

1.1.5 Boolean operations

Be familiar with and able to us:

- NOT
- AND
- OR
- XOR

1.1.6 Constants and variables

Variable – the value can be changed

Constant – the value cannot be changed

<u>Advantage of using a named constant:</u>

Allows you to refer to a value by its name rather than its actual value

The constant can easily be amended to produce a different output

1.1.7 String-handling operations

Be familiar with and able to use:

- Length
- Position
- Substring
- Concatenation
- Character ↔ character code
- String conversion operators
    - String ↔ integer
    - String ↔ float
    - String ↔ date/time

1.1.8 Random number generation

Be familiar with and able to use Random number generation.

1.1.9 Exception handling

Exception handling – how to handle an invalid input/error

Be familiar with how to code for it

1.1.10 Subroutines (procedures / functions)

Subroutines – an 'out of line' block of code that may be executed (called) simply by writing its name in a program statement

<u>Advantages:</u>

- Procedural and functional abstraction => easier to read
- Reduces length of code if subroutine called very often
- Easier to test/debug

1.1.10.1 Parameters

Parameters can be passed to a subroutine in order to pass data

Be able to use subroutines with an interface (i.e. the parameters it requires)

Passing by value – a copy of the parameter is passed to the subroutine so that only the value is accessible to the subroutine.

Passing by reference – the parameter itself is passed so that its value can be altered and these changes will remain in the calling routine

1.1.10.2 Returning value(s)

Be able to use subroutines that return values to the calling routine

1.1.10.3 Local variables

Subroutines can declare their own 'local' variables that:

- Exist only while the subroutine is executing
- Are accessible only within the subroutine

Be able to use local variables

It is good practice as it:

- Aids legibility by compartmentalising the code
- Easier to debug/test

1.1.11 Global variables

Opposite to a local variable

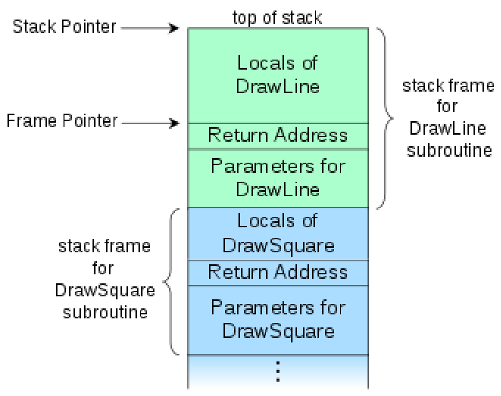
Global variable:

- Exist throughout the duration of the program
- Are accessible everywhere within the program (globally)

1.1.12 Role of stack frames in subroutine calls

Be able to explain how a stack frame is used with subroutine calls to store:

- Return addresses
- Parameters
- Local variables



Basically each time a subroutine is called, the current frame (return address + parameters + local variables) is pushed onto a stack and when the subroutine terminates, it pulls items off the stack to return to the calling procedure and so on and so on.

1.1.13 Recursive techniques

Be familiar with and able to implement recursion.

Base case – an input to the function where the answer is known without the need for further recursions

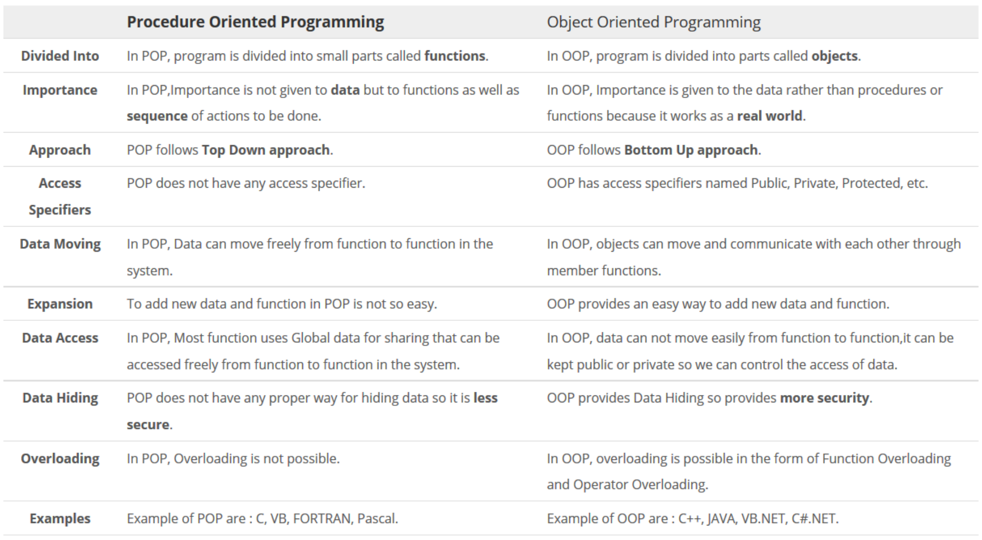
General case – the answer is not known directly => need further recursion

```
if   (some condition for which answer is known)
                         // base case
     solution statement

else                              // general case

     recursive function call
```

Example: 4! = 4 x 3! ; 3! = 3 x 2! ; 2! = 2 X 1!; **1! = 1 (base case)**

1.2 Programming paradigms

Understand the characteristics of POP and OOP and have experience with both.

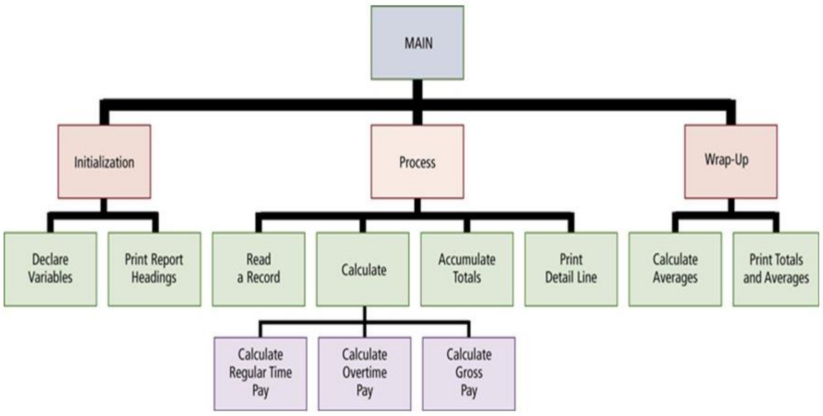


	Procedure Oriented Programming	Object Oriented Programming
Divided Into	In POP, program is divided into small parts called **functions**.	In OOP, program is divided into parts called **objects**.
Importance	In POP,Importance is not given to **data** but to functions as well as **sequence** of actions to be done.	In OOP, Importance is given to the data rather than procedures or functions because it works as a **real world**.
Approach	POP follows **Top Down approach**.	OOP follows **Bottom Up approach**.
Access Specifiers	POP does not have any access specifier.	OOP has access specifiers named Public, Private, Protected, etc.
Data Moving	In POP, Data can move freely from function to function in the system.	In OOP, objects can move and communicate with each other through member functions.
Expansion	To add new data and function in POP is not so easy.	OOP provides an easy way to add new data and function.
Data Access	In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data can not move easily from function to function,it can be kept public or private so we can control the access of data.
Data Hiding	POP does not have any proper way for hiding data so it is **less secure**.	OOP provides Data Hiding so provides **more security**.
Overloading	In POP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
Examples	Example of POP are : C, VB, FORTRAN, Pascal.	Example of OOP are : C++, JAVA, VB.NET, C#.NET.

1.2.1 Procedural-oriented programming

Understand the structured approach to program design and construction.

Hierarchy charts – which functions call which (shows program structure)

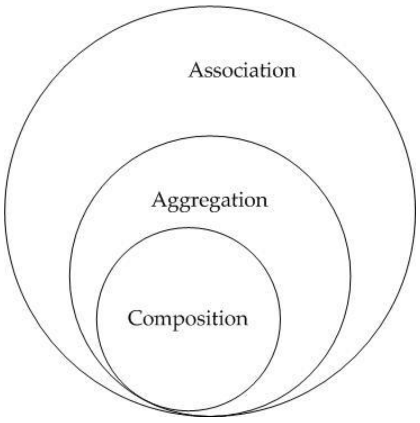


Advantages of the structured approach:

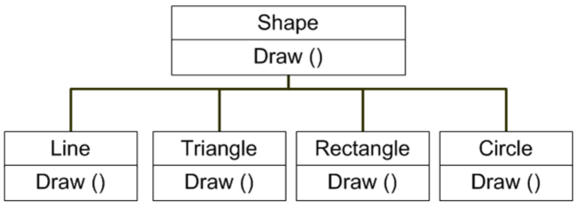
- Aids legibility as flow can be easily mapped
- Easier to debug/test
- Easier to write an algorithm for it
- Reduced complexity

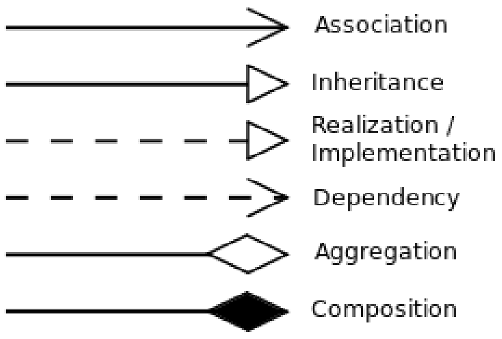
1.2.2 Object-oriented programming

- **Class**
  - Defines methods and attributes that define how the object behaves
- **Object**
  - An instance of a particular class
  - Object Oriented Programming consists of multiple objects talking to each other
- **Instantiation**
  - The creation of a new object from a given class
  - Different instances of an object are identical in behaviour but have different values for their attributes
  - Instantiation must involve calling the constructor for a given class
- **Encapsulation**
  - Independent modules do not affect one another
  - Encapsulation involves grouping things into different modules or capsules
  - The modules only interact through interfaces
  - Inner workings are hidden so as to simplify communication (information hiding)
- **Inheritance**
  - x "is a" y
  - For example: a dog is an animal
  - When class x inherits from y, it inherits the methods and attributes of its parent and can build upon those (or even override)
- **Association**
  - x "has a" y



  - <u>Simple Association</u>
    - Weak relationship
    - No ownership
    - EG: a teacher "has a" student. BUT that student can have other teachers
  - <u>Aggregation</u>
    - Stronger relationship
    - There is unique ownership
    - EG: a car "has a" wheel and no other car can have that same wheel
    - BUT if the car is destroyed, the wheel can still be reused

    - Composition
        - Strongest relationship
        - EG: a house "has a" room and if the house is destroyed then so too is the room
        - A death relationship
- **Polymorphism**
    - Having many forms >> calling the same method from different objects can have different consequences. Each shape must do something different
        - 

    - Redefining an inherited method is called overriding
- **Unified Modelling Language**
    - The standard for drawing class diagrams. Relations shown below
        - 

The style of OOP is often called the Object Oriented Paradigm

Advantages:

- Forces designers to go through an extensive planning phase >> stronger result
- Encapsulation >> source code can be written, tested and maintained independently of other objects
- Once an object is created, knowledge of its implementation is unnecessary
- New objects can easily be defined which build upon existing ones
- Re-usability >> pre-defined obejcst can be reused as part of a library
- Maintenance >> OOP is much easier to maintain than POP due to rigid modular structure

Principles:

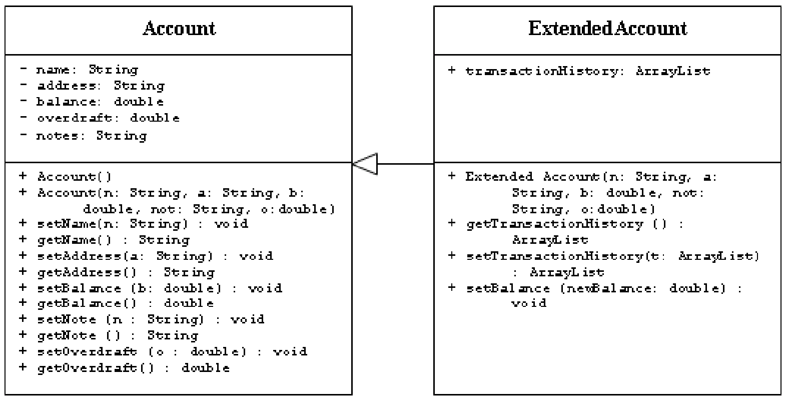
- **Encapsulate what varies**
    - Any aspects that are subject to change should be entirely self contained
    - For example the pound – dollar conversion rate should only come from the money module rather than being hard-coded in other modules >> reduces amount of future modification

- **Favour composition over inheritance**
    - Composition is a weaker and so more flexible relationship than inheritance >> preferred
    - For example, the subclass does not need to inherit all of the methods of the parent
    - Furthermore, most real-life relationships are composition rathe than inheritance
    - EG: a house “has a” door , room, wall, roof…
    - BUT a door “is not” a house
- Program to interfaces not implementation
    - Programming to an interface allows for easier collaboration to a group project as different programmers do not need to wait for others to finish coding their module
    - >> Development takes place in parallel

Be able to write Object-Oriented Programs and have experience coding user-defined classes involving:

- **Method modifiers**
    - **Abstract**
        - An abstract class cannot be instantiated
        - An abstract method is defined but must be implemented in derived classes
        - Abstract methods can only be contained in abstract classes
        - Is effectively a stronger interface >> can have non-static methods that provide more logic
    - **Static**
        - Cannot work on instance variables
        - Is shared between all instances of the class
    - **Virtual**
        - Allows a method to be overridden in a derived class
- **Access modifiers**
    - **Public** + anyone can access it
    - **Private** – only class can access it
    - **Protected** # only derived classes
    - The symbol indicates how it is denoted in UML

Be able to draw and interpret class diagrams, such as:


Account
- name: String
- address: String
- balance: double
- overdraft: double
- notes: String
+ Account()
+ Account(n: String, a: String, b: double, not: String, o:double)
+ setName(n: String) : void
+ getName() : String
+ setAddress(a: String) : void
+ getAddress() : String
+ setBalance (b: double) : void
+ getBalance() : double
+ setNote (n : String) : void
+ getNote () : String
+ setOverdraft (o : double) : void
+ getOverdraft() : double
ExtendedAccount
+ transactionHistory: ArrayList
+ Extended Account(n: String, a: String, b: double, not: String, o:double)
+ getTransactionHistory () : ArrayList
+ setTransactionHistory(t: ArrayList) : ArrayList
+ setBalance (newBalance: double) : void

2 Fundamentals of data structures

Be familiar with the concept of a data structure.

Single and multi-dimensional arrays.

Be able to read/write from a text file

Be able to read/write from a binary (non-text) file // brush up on this

2.1 Abstract data types

Be familiar with the following abstract data types:

- Queue
- Stack
- Graph
- Tree
- Hash table
- Dictionary
- Vector

Be able to implement and use the above ^^^

Static data structure – size of structure is fixed

- Easy to manage
- Predictable
- No need to store pointers or metadata about structure

Dynamic data structure – size can grow or shrink on the fly

- Do not need to know size in advance
- Can occupy less space in memory if sparse object
- Less risk of overflow

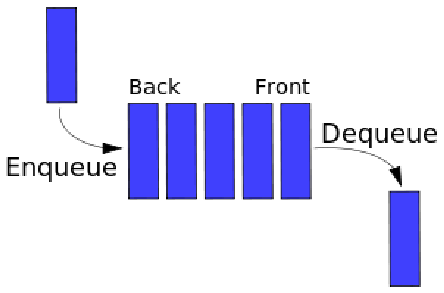
The dynamic reallocation of memory is done within the heap (which can allocate and deallocate very quickly)

Describe creation and maintenance of data within:

- Queues – FIFO (First in First Out)
    - Linear
        - Can either have a fixed size array and shift each element when enqueuing / dequeuing
            - BUT may be time-consuming
        - Can have an array and store start and end pointers
    - Circular
        - Same as second option of Linear, but pointers can loop back to start of queue
    - Priority
        - Special rules for enqueueing, advance until reaching an item with an equal or higher priority
        - E.g. print jobs
- Stacks – FILO

    - Array where elements are pushed in at the front and everything is shifted back
- Hash table
    - Item is stored at index corresponding to its hash
    - If that index is already filled => place in next empty slot (can have a fixed step)
    - When accessing, look in hashed slot, if item is not there => carry on stepping until it is

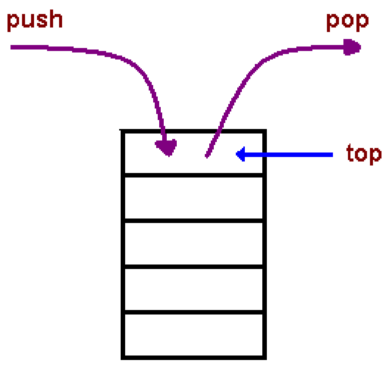
2.2 Queues



Be able to describe and apply the following to linear, circular & priority queues:

- **void Enqueue(item)**
    - Linear: add item at back of queue
    - Circular: add item just before rear pointer and update rear pointer
    - Priority: add item at back of its priority level
- **item Dequeue()**
    - Linear: return item at front of queue and take it off
    - Circular: as above ^^^ and update front pointer
    - Priority: take highest priority, oldest item
- **bool IsEmpty()**
    - Linear: size == 0
    - Circular: do rear and front pointers meet up?
    - Priority: size == 0
- **bool IsFull()**
    - Linear: size == array.size
    - Circular: do rear and front pointers cross?
    - Priority: size == array.size

2.3 Stacks



Describe and apply following operations:

- **Push** – put item onto top of stock
- **Pop** – take off and return top item
- **Peek / Top** – return top item without removing it

And test for empty stack (size == 0) and test for stack full (size == max)

STACK OVERFLOW ⇔ size > max

2.4 Graphs

Graphs are used to represent more complex relationships.

Typical uses:

- Route-finding
- Tube network
- Facebook friends

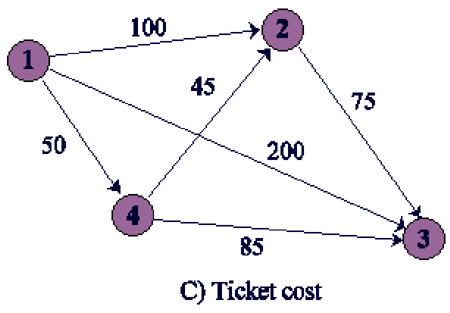


FIGURE 1 - WEIGHTED DIGRAPH

Useful terms:

- **Graph** – set of nodes connected by edges
- **Weighted Graph** – Each edge is given a value / weighting
- **Node / vertex** – the endpoints of edges
- **Edge** – a connection two nodes
- **Undirected Graph** – all edges are bidirectional
- **Directed Graph** – some edges are unidirectional (aka digraph)
- **Connected** – there always exists a path between any two nodes
- **Path** – a sequence of edges to get from node A to B

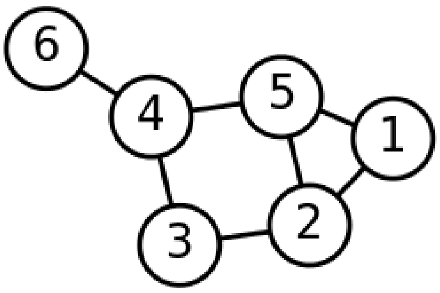
Representation:

- **Adjacency list**
  - A list of records, where each record contains a Node and a list of all nodes connected to it
  - E.g. for graph below

1	2	5	
2	1	3	5
3	2	4	
4	3	5	6
5	1	2	4
6	4		

- **Adjacency matrix**
  - A matrix of Booleans indicating whether there is an edge between two nodes
  - E.g. for graph below

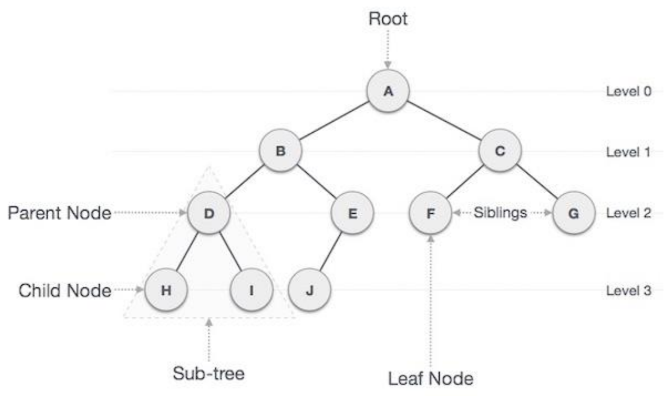
	1	**2**	**3**	**4**	**5**	**6**
1	▨	Y	N	N	Y	N
2	Y	▨	Y	N	Y	N
3	N	Y	▨	Y	N	N
4	N	N	Y	▨	Y	Y
5	Y	Y	N	Y	▨	N
6	N	N	N	Y	N	▨



Which is better?

- **Adjacency List**
    - Easy to implement
    - If graph is sparse, is more space efficient
- **Adjacency matrix**
    - Faster to traverse
    - Easy to implement weightings

2.5 Trees



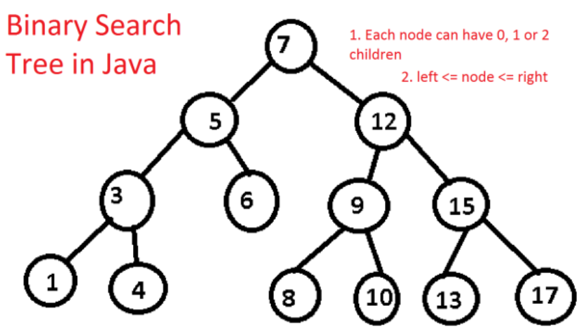
Tree – connected undirected graph with no cycles

Rooted Tree – one particular node has been designated root => sets up parent child relationship from there

Root is the only node with no parents => all other nodes are descendants of the root

Binary Tree – rooted tree where each node has at most two children

Common use for a binary tree is a binary search tree => left child is YES & right child is NO


Binary Search Tree in Java
1. Each node can have 0, 1 or 2 children
2. left <= node <= right
7
5
12
3
6
9
15
1
4
8
10
13
17

=> to search for the item 13 => 7 >R> 12 >R> 15 >L> 13

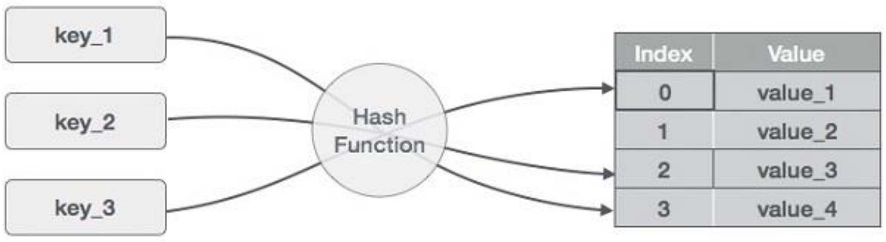
This can be used to perform a tree sort

Other uses for trees:

- Family trees
- Inheritance representation
- File directories

2.6 Hash tables

Hash table – a data structure that creates a mapping between keys and values



Basically, it is a big array, where each item is stored at the index given it by the hash of its key (some unique value).

Collision – if two keys have the same hash, a collision occurs => must *rehash* => often just store the item in the next free slot OR can look in every 3rd slot for example

Hash tables provide *constant time* lookup for items given their key

Uses: Implementing a dictionary

Simple hashing algorithms:

- **Modular division**
    - Use remainder when divided by a number
- **Folding**
    - Subdivide key into subsections and do things to them
    - E.g: 5160 => (5 + 1 + 6 + 0) % 11 = 1
- **Strings**
    - Convert characters to ASCII codes and do stuff to them

2.7 Dictionaries

Dictionary – a collection of key-value pairs accessed via the associated key

Often implemented in a hash table

Used for information retrieval / compression

Example: "the good the bad and the ugly" => 0102304

Where the associated dictionary is:

Key	0	1	2	3	4
Value	the	good	bad	and	ugly

2.8 Vectors

An n-dimensional vector can be represented as:

- A list of n numbers
- A function
- A representation of a geometric point in n-dimensional space

Vectors notations

- List of numbers: [a, b, c, d] where $a, b, c, d \in same\ field\ (e.g.\mathbb{R})$
- A 4-vector over the reals can be denoted $\mathbb{R}^4$
- Function notation: $f : S \mapsto \mathbb{R}$ where S is the basis e.g. [i, j, k]

All entries into a vector must belong to the same field

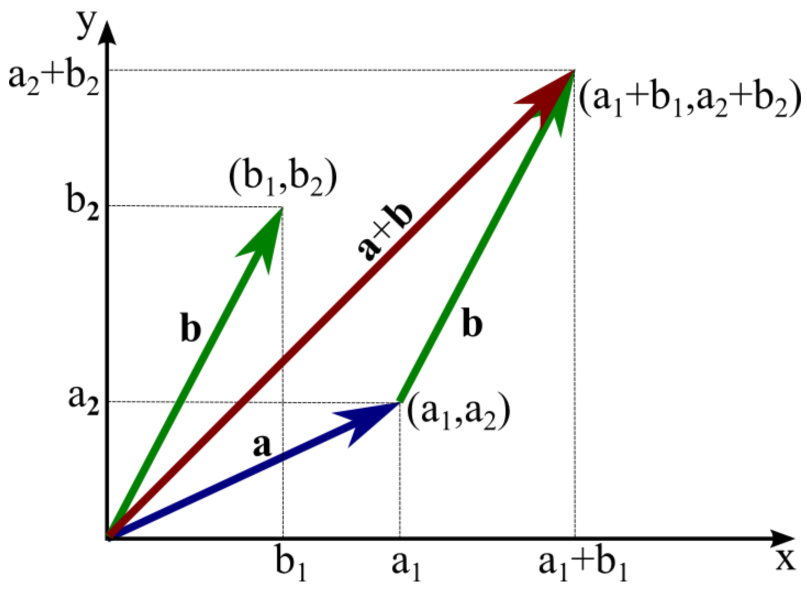
2.8.1.1 Vector representation in programming languages

The three vector 3**i** + 4**j** + 2**k** can be represented as follow

- Dictionary: [**i**:3, **j**:4, **k**:2]
- 1D array: [3, 4, 2]

2.8.1.2 Visualising vectors

Vectors can be visualised as arrows, with tail at origin and head at point specified by vector's components



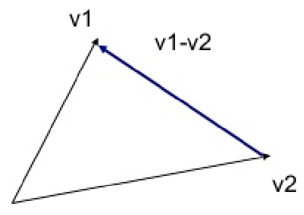
Vectors add head-to-tail >> vector addition can be used to define **TRANSLATIONS**

Vector multiplied by a scalar s increases the magnitude by a factor of s >> scalar multiplication can be used to define **SCALINGS**

2.8.1.3 Convex combinations

For two vectors (**u**, **v**), a convex combination is defined as:

$$\boldsymbol{c} = \alpha\boldsymbol{u} + \beta\boldsymbol{v} \qquad where\ \alpha + \beta = 1 \qquad and\ \alpha, \beta \geq 0$$



- c must map to a point on the line from **u** to **v**

2.8.1.4 Dot Product (aka Scalar product)

For two n-dimensional vectors (**u**, **v**), the dot product is defined as the sum of the products of all like components

$$\boldsymbol{u} \cdot \boldsymbol{v} = u_1 v_1 + u_2 v_2 + \cdots u_n v_n$$

Dot product is used in lots of maths

Can be used to find angle between two vector of known magnitude

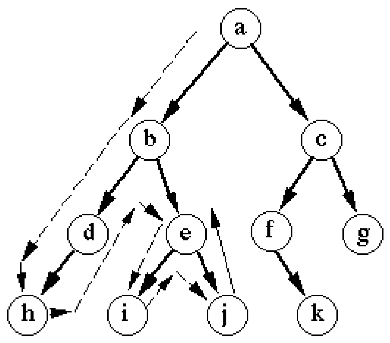
3 Fundamentals of algorithms

3.1 Graph-traversal

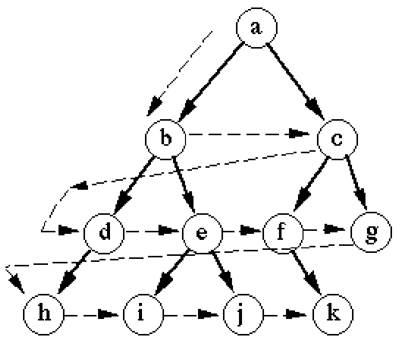
Be able to trace Breadth-first and Depth-first search algorithms

Breadth-First – Go down and search one level at a time (often uses a queue)

Depth-First – Go down as far as possible down one branch until hitting a dead end and then go back up until you can go down (often uses a stack)



Depth-first search



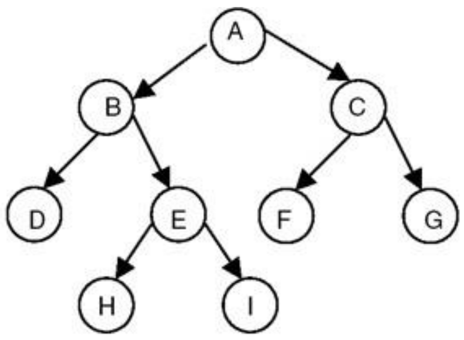
Breadth-first search

Typical applications:

- Breadth – shortest path
- Depth – navigating a maze

3.2 Tree-traversal

Types of tree traversal, written in pseudocode. Each algorithm is recursive, where each node is the root of its own subtree and the base case is a leaf:



Inorder : DBHEIAFCG
Preorder : ABDEHICFG
Postorder : DHIEBFGCA

- **Pre-Order**
    - Pseudocode:
        - **Root >> Left >> Right**
        - Visit Root Node
        - Traverse Left Subtree
        - Traverse Right Subtree
    - Uses:
        - Copying a tree
        - Producing prefix expression from an expression tree
- **Post-Order**
    - Pseudocode:
        - **Left >> Right >> Root**
        - Traverse Left Subtree
        - Traverse Right Subtree
        - Visit Root Node
    - Uses:
        - Infix to RPN (Reverse Polish Notation)
        -
- **In-Order**
    - Pseudocode:
        - **Left >> Root >> Right**
        - Traverse Left subtree
        - Visit Root Node
        - Traverse Right subtree
    - Uses:
        - Binary Search tree
        - Outputting contents of a binary search tree in ascending order

3.3 Reverse Polish

Reverse Polish Notation – Way of writing arithmetic expressions in a way that is easily computerised

Advantages:

- Eliminates need for brackets in sub-expressions
- Suitable form for evaluation suing a stack
- Uses:
    - interpreters based on a stack setup
    - calculators

Reverse Polish is also called **postfix** as opposed to the common **infix**

- **Postfix** – operation comes **after** operands
- **Infix** – operation is **in** between operands

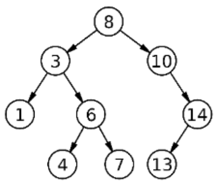
Must be able to convert between RPN and infix:

Infix	Postfix
A + B	A B +
(A + B) * C	A B + C *
(A + B) / (C + D)	A B + C D + /
(A + B / C) + (E * F)	A B C / + E F * +
(A / B) * (C + D)	A B / C D + *

3.4 Searching algorithms

Typical searching algorithms:

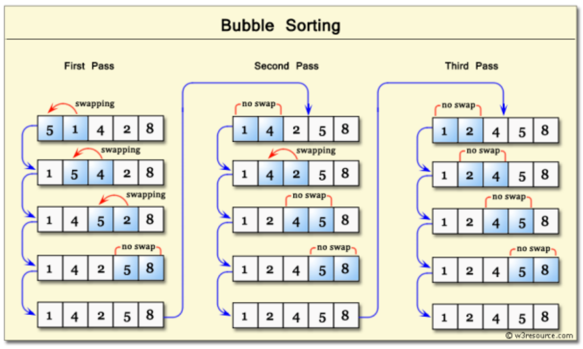
- **Linear Search**
    - Only option if items are unordered
    - Go through elements sequentially until a match is found or the end of the sequence is reached
    - Time complexity:
        - O(n) as must iterate through n elements
- **Binary Search (split)**
    - If elements are ordered in some way binary split is much more efficient
    - Ask questions of midpoints to narrow down search area
    - Pseudocode
        - BinarySplit(List, target)
            - IF target > middleItem THEN binarySplit(upperHalf)
            - ELSE IF target < middleItem THEN binarySplit(lowerHalf)
            - ELSE return middleItem
    - Time Complexity:
        - Each question halves search area
        - >> $2^Q = n$ where Q is the number of necessary questions
        - >> O(log n)
- **Binary Tree Search**
    - Same as binary split except elements ordered into a binary tree >> only ned to ask questions of root node in the subtree

- 

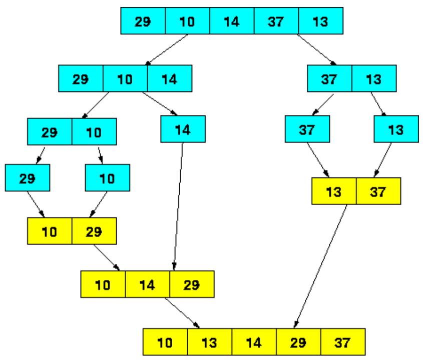
- To find item 4
    - 4 < 8 LEFT
    - 4 > 3 RIGHT
    - 4 < 6 LEFT
    - 4 = 4 DONE

- <u>Time Complexity:</u>
    - Same as binary split
    - O(log n)

3.5 Sorting algorithms

- Bubble Sort
    - Iterate through sequence
    - If two neighbouring items are in reverse order => swap them
    - Repeat until no swaps have to be made
        - 

    - <u>Time complexity</u>
        - Each time going through the list must perform n comparisons
        - It is required to go through the list about n times (if first element is at the back then must swap n times)
        - n iterations each requiring n comparisons
        - >> O(n^2)
        - Very inefficient
- Merge sort
    - Uses ease with which two sorted lists can be combined

- 

  - Divides up superlist into n sublists each containing 1 element
  - Repeatedly merge sublists of similar size until only 1 list remains
  - <u>Time complexity:</u>
    - Much more efficient
    - On a single merging round (number of lists is halved) must perform n comparisons
    - Each round doubles list size >> requires log n rounds of merging to get back up to the original list size

3.6 Optimisation algorithms

Optimisation algorithms are something that finds the optimum solution to a given problem such as:

- Timetabling
- Shortest path between two points
- Scheduling flight staff

Dijkstra's shortest path algorithm: not expected to recall steps

- Finds the shortest path between a particular origin node and all other nodes on a weighted graph
- Similar to breadth-first search except it uses a priority queue rather than a FIFO queue
- Pseudocode:

```
WHILE (priorityQueue NOT empty)
    parent <- priorityQueue.DeQueue()
    FOR EACH (child in parent.Neighbours)
        If (child.ScoreLesser)
            child.UpdateScore
            priorityQueue.EnQueue(child)
```

- Uses:
  - Route-finding
  - Tube flow simulations

4 Theory of computation

4.1 Abstraction and automation

4.1.1 Problem Solving

Be able to develop solutions to simple logic problems

Be able to check solutions to simple logic problems

4.1.2 Following and writing algorithms

Algorithm – A sequence of steps that can be followed to complete a task, which always terminates.

Be able to construct an algorithm using pseudo-code using standard conventions:

- Sequence
- Assignment
- Iteration
- Selection

Be able to hand-trace algorithms (trace-table)

Be able to convert an algorithm into high-level code

Be able to articulate how a program works, arguing for its correctness and efficiency using logical reasoning, test data and user feedback.

4.1.3 Abstraction

Abstraction of problems is very common in computing

- **Representational abstraction**
    - A representation arrived at by removing unnecessary details
        - e.g. flight simulators
- **Abstraction by generalisation / categorisation**
    - A grouping by common characteristics to reach a specific type
    - To reach an 'is a kind of' hierarchical relationship
        - e.g. geometry >> 2-D >> rectangle >> square
- **Information hiding**
    - Hiding / eliminating all non-essential details of a problem
        - e.g. Euler's bridges
- **Procedural abstraction**
    - Separation of parameters (operands) from procedures (operations)
    - This means using identifiers (names) for the parameters
    - Therefore it is easy to change the parameters to produce a different output
        - e.g. Math.Pow(3, 2) rather than 3*3
- **Functional abstraction**
    - Calling a function to return the result of a sub-problem
    - Without knowledge of how it is done
        - e.g. Console.WriteLine()
- **Data abstraction**
    - Separation of data values from data structures
    - The programmer has no knowledge of how they are stored only how to manipulate

        - e.g. double x // no idea how that does it
- **Problem abstraction / reduction**
    - The removal of details until a problem is solvable
        - e.g. calculating CoP of a model rocket

- **Decomposition**
    - Breaking down a complex problem into a series of simpler sub-problems (which may be further sub-divided)
        - e.g. to win the league must:
            - hire a manager
            - buy good players
            - play well
- **Composition**
    - Opposite to decomposition
    - Combining procedures to form a compound procedure which performs a complex task
        - e.g. find 8y – 1 with the following functions
            - f(x) = 8x
            - g(x) = x – 1
        - => g(f(y))
    - Can also refer to data composition to form compound data structures
        - e.g. arrays, lists

4.1.4 Automation

- **Automation**
    - Requires putting models (real-world abstractions) into action to solve problems independently of a human
    - This is achieved by:
        - Creating algorithms
        - Implementing the algorithms (code)
        - Implementing the models in data structures
        - Executing the code

Real-world problem >> Abstraction >> Mathematical model >> Automate >> Automaton

>> tells us more information about the real world

- **Computer science**
    - Is about building clean models of messy real-world situations

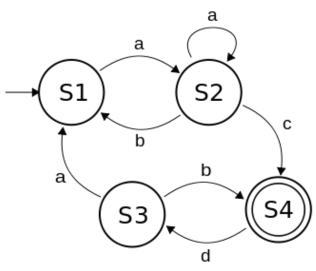
Must choose how much info to keep/discard in order to produce a solution to the required degree of accuracy

4.2 Regular languages

4.2.1 Finite State Machines (FSM's)

No output:

Be able to draw and interpret simple state transition diagrams / tables for FSM's with no output

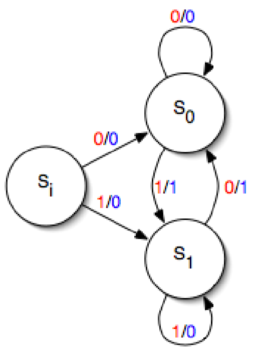


S1 is the start state; S4 is the finish state

With output:

Called a **Mealy Machine** >> each transition can output a value >> can be used to represent:

- Ciphers
- Traffic lights
- Vending Machines
- Electronic circuits



Red = input ; Blue = output

4.2.2 Maths for Regular Expressions

Set – unordered collection of values where each value is unique

Ways of defining a set {use curly braces}

- **Listing**
  - $A = \{0, 1, 4, 9\}$
- **Comprehension**
  - $A = \{n^2 \mid n \in \mathbb{N} \wedge n < 4\}$
  - Where:
    - | means such that
    - ∧ means AND
    - ∨ means OR
    - ∈ means is a member of
- Compact Representation
  - $B = \{0^n 1^n | n \geq 1\}$
  - All strings containing some number of 0's followed by the same number of 1's
  - $B = \{01, 0011, 000111, \ldots\}$

The Empty set – No members: $\emptyset = \{\}$

Types of set:

- **Finite** – finite number of elements
  - **Cardinality** – number of elements
- **Infinite** – infinite number of elements
  - **Countably Infinite** – can be ordered in some way s.t. it is counted off by the natural numbers.
  - The reals are uncountable as there is no "next" number, but integer coefficient complex numbers can be ordered by modulus >> countable

Cartesian Product

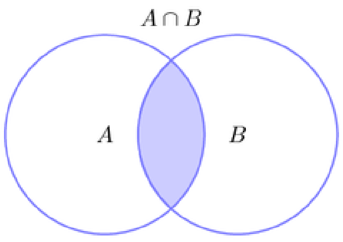
- Written as $A \times B$ where A and B are both sets
- It is the list of all ordered pairs $(a, b)\ s.t.\ a \in A\ and\ b \in B$
- Example: $\{0, 1\} \times \{2, 3, 4\} = \{(0,2), (0,3), (0,4), (1,2), (1,3), (1,4)\}$

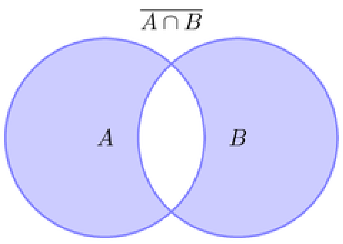
Terminology and notation

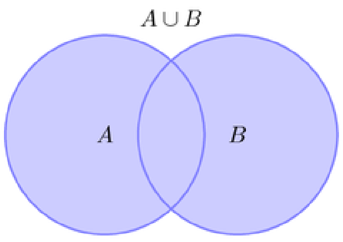
- **Subset**
  - $A \subseteq B$
  - All elements in A are in B >> A is a subset of B
  - Underlined as it is proper subset or equal to ($\subset or =$)
- **Proper Subset**
  - $A \subset B$
  - All elements in A are in B but A and B are not the same >> A is a proper subset
- **Countable set**
  - Can be numbered off by the natural numbers

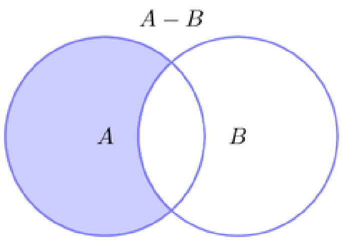
Set Operators

- **Membership** $\in$
  - $a \in A \leftrightarrow a \text{ is a member of set } A$
- **Union** $\cup$
  - $A \cup B = \text{all distinct members of } A \text{ and } B$
- **Intersection** $\cap$
  - $A \cap B = \text{all members of } A \text{ that are also in } B$
- **Difference** $-$ ***or*** $/$
  - $A - B = \text{all members of } A \text{ that are } \textbf{not} \text{ also in } B$









4.2.3 Regular Expressions

Simply a way of describing a set in shorthand >> regex allows particular languages or patterns to be described

Metacharacters:

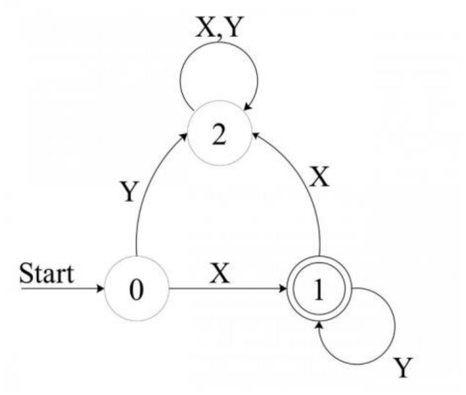
- ***** (0 or more repetitions)
- **+** (1 or more repetitions)
- **?** (0 or 1 repetitions, ie optional)
- **|** (alternation, ie or)
- **( )** to group regular expressions.
- Any others will be defined in the questions

Be able to form and use simple regex's for string manipulation and matching >> PRACTICE

Relation to FSM's:

FSMs and regex's are equivalent ways of defining a regular language

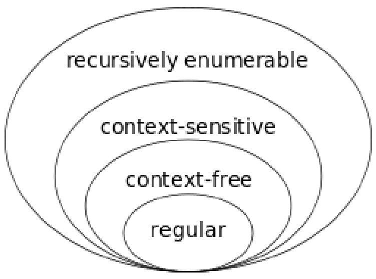
EG: Below FSM is equivalent to **XY***



Be able to convert between FSM and Regex

A language is regular if it can be represented by a regex (or an FSM)

Chomsky Hierarchy (not on spec):



4.3 Context-free languages

4.3.1 Backus-Naur Form (BNF) / syntax diagrams

There needs to be some way to unambiguously define the syntax of a programming language >> just for human consensus or to program a parser

Spoken languages are too ambiguous to be used BUT Regular Expressions would be:

- Time consuming to define the regex
- Impossible to specify certain conditions, such as the validity of nested brackets

>> a new type of language (**meta-language**) was developed to overcome this hurdle

Backus-Naur Form is one such meta-language

4.3.1.1 Construction

Consists of a list of **production** statements of the form:

LHS ::= RHS

Where **"::="** means **"is defined by"** and is known as a **meta-symbol**

The other important meta-symbol is **"|"** meaning **"or"**

Example A, to define a pin in Arduino of form digit or letter and digit:

<pin> ::= <digit> | <letter><digit>

<letter> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

<digit> ::= 0|1|2|3|4|5|6|7|8|9

Each expression in angle brackets is called a **meta-component**

OR sometimes such an expression is called a **<non-terminal>** because it's definition is not complete

Any expression hard coded in is called a **terminal** >> LHS consists of a <non-terminal> and RHS can consists of combinations of terminals and <non-terminals>

4.3.1.2 Using Recursion

Example B, to define a alphabetical string:

<word> ::= <letter> | <letter><word>

Where letter is suitably defined.

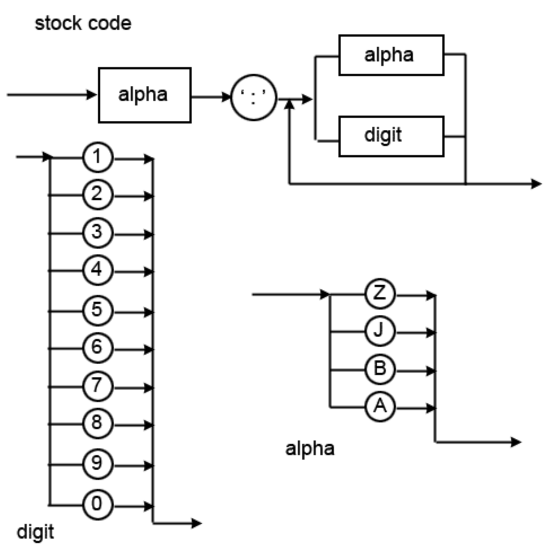
Therefore, string "abc" is checked recursively as follows:

- "a" is a <letter> => check is "bc" is a <word>
    - "b" is a <letter> => check "c" is a <word>
        - "c" is a <letter> => "c" is a <word>
    - => "bc" is a <word>
- => "abc" is a <word>

4.3.1.3 Syntax diagrams

Graphical way of defining the syntax of a language => map directly to BNF

Example, to define a stock-code of form a:s, where a is any character from "ABJZ" and s is an alphanumeric string consisting of digits and any number of a's



The BNF version of the syntax diagram is as follows:

<stock code> ::= <alpha> : <second half>

<second half> ::= <alphanumeric> | <alphanumeric> <second half>

<alphanumeric> ::= <alpha> | <digit>

<alpha> ::= A|B|J|Z

<digit> ::= 0|1|2|3|4|5|6|7|8|9

Must be able to:

- check language syntax by referring to BNF or syntax diagrams
- Formulate simple production rules
- Explain why BNF can represent some languages regular expressions cannot

4.4 Classification of algorithms

Algorithms can be compared by expressing their complexity written as a function relative to the size of the problem (denoted n)

Big-O notation >> Order

Types of complexity:

- **Time**: the runtime
- **Space**: the amount of extra resources needed (often memory)

Efficient implementation: minimal time and space complexity BUT there are always trade-offs

4.4.1 Maths for Big-O notation

Be familiar with the concept of a function, as a mapping of values from domain to range

Types of function:

- **Linear**: $y = ax$
- **Polynomial**: $y = ax^p + bx^{p-1} + \cdots z$
- **Exponential**: $y = a^x$
- **Logarithmic**: $y = \log x$

Be familiar with the notion of the permutations (orderings) of a set >> a set of n distinct elements has n! permutations

Factorials: $n! = n(n-1)(n-2)\ldots(1)$

4.4.2 Order of Complexity

Be familiar with big-O notation: $O(f(n))$ for specifying how the requirements scale with the size n of the input

Be able to derive the time complexity of an algorithm

Be able to apply it to the following time complexities:

- Constant time – O(k)
- Logarithmic time – O(log n)
- Linear – O(n)
- Polynomial – O(n^2) or higher powers
- Exponential – O(e^n) or O(n!)

In computing, factorials are regarded as exponentials

4.4.3 Limits of Computation

Algorithmic complexity and hardware impose limits on what can be computed

Classification of algorithmic problems:

- **Tractable** – problems that run in polynomial (or better) time
- **Intractable** – problems that cannot run in polynomial time

For intractable problems, heuristic methods are often adopted (narrowing sample space)

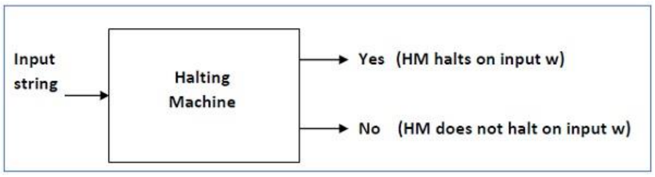
Computable and non-computable problems:

Be aware some problems cannot be solved algorithmically

4.4.4 Halting Problem

Alan Turing proposed this to prove some problems are unsolvable by a computer (or indeed mathematics)

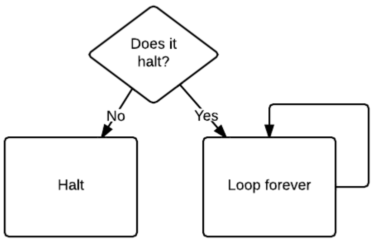
Halting problem: To determine if a particular program will stop if given a particular input or carries on forever



This problem is significant as it shows that there are some problems which are simply insoluble

Proof (not needed for A-level):

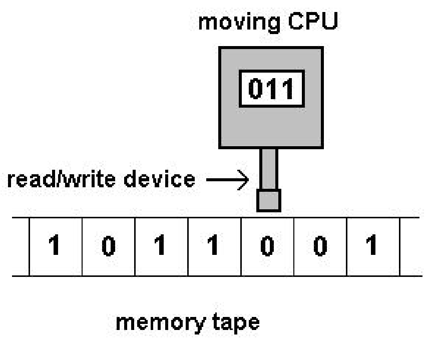
- Suppose machine H does exist
- Take output from H into another machine
    - If program halts >> loop forever
    - If program loops >> halts
- >> we have a contradiction >> H cannot exist



4.5 A model of computation – Turing Machines

A Turing machine is a mathematical abstraction of a computer with a single fixed-program, expressed using:

- A finite set of states in a state transition diagram
- A finite alphabet of symbols
- An infinite tape with marked off squares
- A sensing read-write head that can move along the tape, one square at a time



When the tape reaches a particular square, depending on:

- The content of the square
- The current state of the Turing machine (its history)

The Turing machine can:

- Move Left
- Move Right
- Change the symbol in the square
- Halt

The initial state of the Turing Machine is called the **start state**. Any states with no outgoing transitions are called **halt states**.

4.5.1 Representing transitions

Have been used to state transition diagrams:

$$\begin{tikzcd} \arrow[r] & S_1 \arrow[loop above, "1"] \arrow[r, bend right, "0"'] & S_2 \arrow[loop above, "1"] \arrow[l, bend right, "0"'] \end{tikzcd}$$

These can be written as a table, or as a **transition function (δ)**

- $\delta(State_0, Symbol_0) = (State_1, Symbol_1, Movement)$
- Where the subscripts 0 and 1 refer to the input and output, and movement denotes which way the Turing machine moves

Be able to:

- represent transition rules using a transition function
- represent transition rules using a state transition diagram
- hand-trace simple Turing machines.

4.5.2 Importance of Turing Machines

Universal Turing Machine – A Turing Machine (U) which can be programmed by the sequence at the start of its input tape to behave exactly like some other Turing Machine (M)

Turing Machines were an enormous breakthrough in computer science as:

- They provide a formal mathematical model of computation
- Define what is computable
- Led to the idea of the stored program concept

5 Fundamentals of data representation

5.1 Number Systems

Natural - $\mathbb{N}$ = {0, 1, 2, 3, … }

Integer - $\mathbb{Z}$ = { …, -3, -2, -1, 0, 1, 2, 3, … }

Rational - $\mathbb{Q}$ = can be expressed as an integer fraction

Irrational - $\mathbb{Q}'$ = cannot be expressed as an integer fraction (e.g $\sqrt{2}$)

Real - $\mathbb{R}$ = all possible real world quantities (i.e. not complex)

Ordinal = used to tell position {0th , 1st , 2nd , 3rd …}

Natural numbers are used for counting

Real numbers are used for measurement

5.2 Number bases

5.2.1 Common bases

Binary – base 2

	2^5	2^4	2^3	2^2	2^1	2^0
Bit value	0	1	1	0	0	1
Denary value:	0 +	16 +	8 +	0 +	0 +	1
Total:			25_{10}			

Denary / Decimal – base 10

You know this stuff

	10^5	10^4	10^3	10^2	10^1	10^0
Digits:	3	7	1	0	0	1
Value:	3 x 10^5 +	7 x 10^4 +	1 x 10^3 +	0 +	0 +	1 x 10^0
Total:			371001_{10}			

Hexadecimal – base 16

Den:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hex:	0	1	2	3	4	5	6	7	8	9	**a**	**b**	**c**	**d**	**e**	**f**

5.2.1.1 Conversions between

NOTE: Base is written as a subscript

Binary	**Denary**	**Hexadecimal**
$1111\ 1110_2$	254_{10}	$f\,e_{16}$

- **Denary <-> Binary:** Express as a sum of powers of 2

- **Denary <-> Hex:** Express as a sum of powers of 16

- **Binary <-> Hex:**
    - $16 = 2^4$
    - Thus, one hex digit = 4 binary digits (one nibble)

Hex:	f				7				a				1			
Denary:	**15**				**7**				**10**				**1**			
Binary:	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**1**

Therefore, Hex is used as **shorthand** for Binary

5.3 Units of Information

5.3.1 Bits and Bytes

Bit - b - **B**inary dig**IT**. Can be: {0 , 1}

Nibble – 4 bits

Byte – B - 8 bits

$2^{\mathbf{n}}$ different values can be represented with **n** bits

5.3.2 Units

	Prefix	Symbol	Value
Binary Prefixes	kibi	Ki	$2^{10} = 1024 \approx 1000$
	mebi	Mi	2^{20}
	gibi	Gi	2^{30}
	tebi	Ti	2^{40}
Denary Prefixes	kilo	K	10^{3}
	mega	M	10^{6}
	giga	G	10^{9}
	tera	T	10^{12}

E.G.: 1 kiloByte = 1 KB = 1 x 10^{3} B = 8 x 10^{3} b

5.4 Binary Number System

5.4.1 Unsigned binary

Unsigned binary – can only be +ve

Signed binary – can be +ve or -ve hence the sign

For an unsigned binary number with n digits:

$$0 \leq x \leq 2^n - 1$$

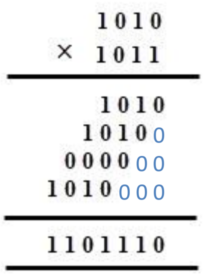
A total of 2^n different integer values

5.4.2 Unsigned binary arithmetic

5.4.2.1 Addition

$$\begin{array}{ccccccc}
 & {\scriptstyle 1} & & & & {\scriptstyle 1} & \\
 & & 1 & 0 & 0 & 0 & 1 \\
+ & & 1 & 1 & 1 & 0 & 1 \\
\hline
 & 1 & 0 & 1 & 1 & 1 & 0 \\
\hline
\end{array}$$

5.4.2.2 Multiplication



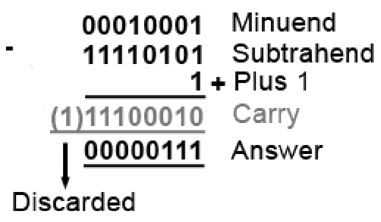
- x 1
- x 10
- x 000
- x 1000

5.4.3 Two's Complement (Signed Binary)

One way of representing negatives.

To convert +ve ⇔ –ve:

1. **Invert bits**
2. **Add 1**
3. **Discard carry**



Leading bit is a “sign” bit:

- 1 -> negative
- 0 -> positive

5.4.3.1 Range

Maximum: 0111 1111… ; Minimum: 1000 0000…

For a two’s complement integer x, with n bits:

$$-(2^{n-1}) \leq x \leq 2^{n-1}$$

Therefore, 2^n distinct integer values (including 0)

5.4.4 Fractions

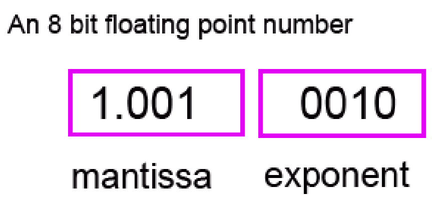
Fixed point – the position of the bicimal point is specified

E.G. for a number with 8 bits, 4 preceding and 4 after the bicimal point:

2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}	2^{-4}
1	1	0	0	.	1	0	1	0
8	4	0	0		1/2	0	1/8	0
				12.625_{10}				

Convert decimal ⇔ fixed point binary : express as a sum of powers of 2

Floating point – equivalent to standard form for binary



To get actual value: $mantissa \times 2^{exponent}$

It is the mantissa shifted the exponent number of bits to the left

Can use signed binary (two’s complement) to specify negatives

For above example:

- Mantissa = 1.001 base 2 = 1.125 base 10
- Exponent = 0010 base 2 = 2 base 10
- >> Answer = 1.125 x 2^2 = 4.5 = 100.1
    - Shifted two bits to the left

In exam, mantissa and exponent will be written in two’s complement

5.4.4.1 Rounding errors

Both fixed and floating point can only give answers to a specific number of significant figures (base 2)

>> some values can never be represented exactly such as $0.1_{10}\ and\ \sqrt{2}_{10}$

Absolute Error: The exact value the answer is off

Relative Error: The % error

Be able to calculate absolute and relative error

5.4.4.2 Fixed Point vs Floating Point

For an equal number of bits

- **Range**
  - Floating Point can represent far more values
- **Precision**
  - Fixed Point his more bits for significant figures >> can be more precise
- Speed of Calculation
  - Both fairly easy
  - Fixed Point
    - Addition: do as normal
    - Multiplication: as normal but must shift
  - Floating Point:
    - Addition: must change such that exponents are equal first
    - Multiplication: multiply mantissas add exponents

5.4.4.3 Normalisation of floating point form

In the same way normal form is normalised

>> mantissa's most significant bit must be occupied (must be 1.y NOT 0.y)

Normalisation makes data easy to maintain and sets rigid rules for underflow/overflow

Be able to normalise unnormalized floating point numbers

5.4.4.4 Errors

Overflow – product is too large to be represented

Underflow – product is too small to be represented

5.5 Information coding systems

5.5.1 Character form of a decimal digit

The number 7:

$$7_{10} = 000\ 0111_2$$

The character ‘7’:

$$‘7’ = 011\ 0111_2 \text{ (ASCII code for ‘7’)}$$

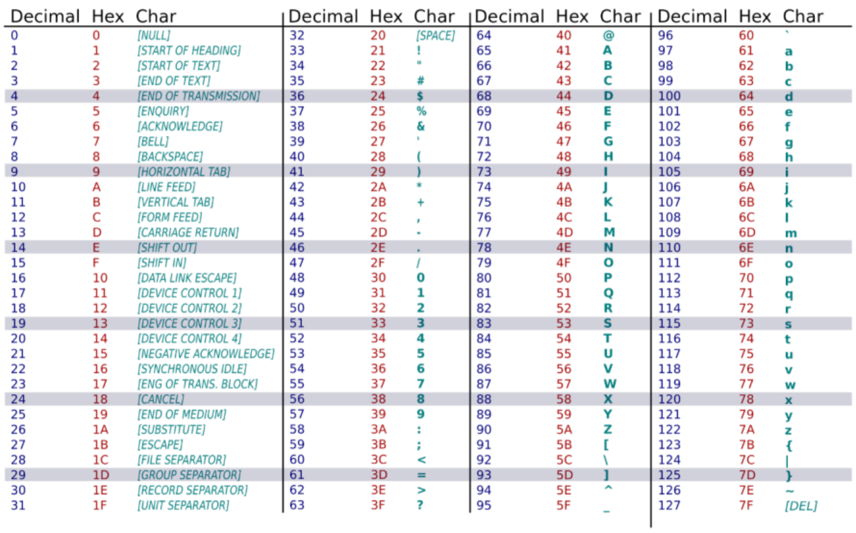
These are DIFFERENT

5.5.2 ASCII & Unicode

ASCII:

Each character is assigned an integer value to identify it => enables easy transfer of character data

ASCII uses 7 bits => 128 unique values



Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Unicode:

Identical to ASCII except there are more bits => more characters available

- UTF-16 has 16 bits => 2^{16} = 65,536 different characters

It was invented to extend the amount of characters available (include Greek, Cyrillic etc.)

The first 128 codes are identical to ASCII => compatible

BUT more bits / character => larger filesizes and transmission times

5.5.3 Error-checking & Correction

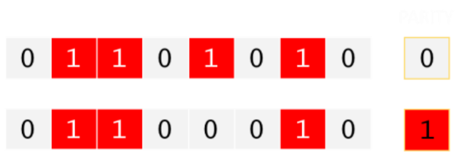
5.5.3.1 Parity Bits

One bit is reserved.

Its value is altered to make the total number of 1’s, either even or odd

- **Even parity:** Sum is even
- **Odd parity:** Sum is odd

E.G. even parity:



If when the data is received the total is not even (/odd) => Error => Request retransmission

5.5.3.2 Majority Voting

Each bit is sent 3 times

The most popular value is used

E.G. to send 1001

1			0			0			1		
1	1	1	0	🟥 1	0	0	0	0	1	1	🟥 0

The red parts are where errors have occurred.

5.5.3.3 Check Digits

Multiple digits are reserved

They contains the result of a mathematical operation (algorithm) conducted on the other digits

The receiver conducts the same operation on the digits receive

If the output is equal to the check digits then the data is correct

e.g. Hamming Code

5.5.3.4 Check Sum

Identical to Check Digits except that there is often only one

Often store the sum of all the digits sent (mod something)

5.6 Representing Images, Sound and other data

5.6.1 Bit patterns, images, sound and other data

Bit patterns can be used to represent these other forms of data.

Details to follow…

5.6.2 Analogue & Digital

Analogue – Can take a continuous range of values

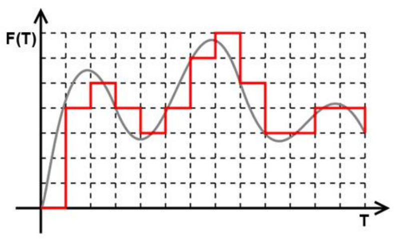
Digital – Can only take a set of discrete values

Examples:

	Analogue	Digital
Data	Sound waves	.mp3 file
Signal	Microphone signal	USB

5.6.3 Analogue ⇔ Digital conversion

5.6.3.1 Analogue to Digital Convertor (ADC)

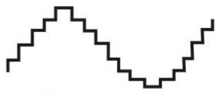


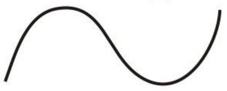
1. Analogue input signal
2. Samples taken at regular intervals
3. Samples quantised (assigned a discrete value)
4. Digital output signal

E.G. microphone >> ADC >> file

5.6.3.2 Digital to Analogue Convertor (DAC)

1. Digital input signal
2. Points are joined by a "smooth" curve (often lots of straight lines)
3. Analogue output signal





E.G. file >> ADC >> Speaker

5.6.4 Bitmaps

5.6.4.1 Principle

An image is divided into an array of pixels

The colour of each pixel is described by a binary value

The pixels are stored in sequence

Allows the computer to reconstruct the image

0	1	0	1	0
1	1	1	1	1
1	1	1	1	1
0	1	1	1	0
0	0	1	0	0

5.6.4.2 Properties

Resolution ≈ number of pixels

= width x height (in pixels)

e.g. 1080 x 1920 p

OR

= pixels / unit length

e.g. 370 ppi (pixels per inch)

Colour Depth – number of bits per pixel

e.g. Colour depth of 4 => 2^4 different colours available

5.6.4.3 Storage Requirements

Ignoring metadata:

Filesize (bits) = Resolution x Colour Depth

As resolution = number of pixels

Colour Depth = bits/pixels

5.6.4.4 Metadata

But there has to be metadata stored about the image. For example:

- Colour Depth
- Width
- Height
- Date taken
- Etc

5.6.5 Vector Graphics
Rather than storing information about each pixel, can store information on how to draw each shape

Vector graphics:

- Represent image using a list of objects
- Each object is a geometric shape with various properties such as
    - X-coordinate
    - Y-coordinate
    - Fill colour
    - Stroke colour
    - Stroke thickness

Use vector graphic primitives to create a simple vector graphic

Primitives are basic objects like: lines, curves, polygons, ellipses

5.6.6 Vector Graphics vs Bitmaps
Pros of Bitmaps:

- Smaller filesize for complex images
- Easy interface with digital cameras
- Easy to display
- Uses: digital photography

Pros of Vectors:

- Infinitely scalable with no loss of quality
- Smaller filesize for simple geometries
- Easy to produce using CAD
- Uses: corporate logos

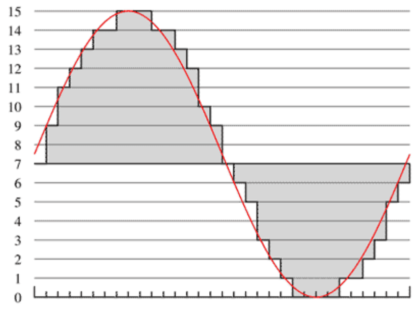
5.6.7 Digital Representation of Sound

5.6.7.1 Principle

Samples taken at a given rate

These samples are quantised (assigned a discrete value) held in binary

They are then stored as binary patterns



5.6.7.2 Properties

Sample Resolution – the number of bits used to store each sample (audio bit depth) => higher resolution => more precise amplitude reproduction

Sampling Rate – the number of samples taken / second

5.6.7.3 Nyquist's Theorem:

In order to accurately represent a sound sample

The sampling frequency must be at least

DOUBLE

The highest frequency in the sample

Therefore, as the human range is: 20 – 20,000 Hz

CD's have a sampling rate of 44 kHz

5.6.7.4 Storage Requirements

Filesize (bits) = Resolution x Sampling rate x duration

5.6.8 Music Instrument Digital Interface (MIDI)

5.6.8.1 Principle

Rather than directly recording the sound

MIDI contains instructions on how to synthesise the sound

From a given repertoire of pre-recorded samples

These instructions are carried out sequentially at a given speed to reproduce the sound

5.6.8.2 Event messages

MIDI files can communicate with other electronic devices via event messages to:

- Synchronise tempo
- Control pitch
- Change volume

5.6.8.3 Advantages over traditional storage

- Smaller filesizes w/ no loss of quality
    - As the individual sound samples do not have to be saved
- Easy to manipulate
    - Can change instrument, key, tempo etc. very easily

5.6.9 Data Compression

5.6.9.1 Why do we compress

Images and sound are often compressed (as well as text)

To reduce filesize => Speed up transmission

5.6.9.2 Types of compression

Lossy – some (less important) information is removed

- Huge reduction in filesize

Lossless – retains all the information required to recreate the original file exactly

- No loss in quality (unlike lossy)

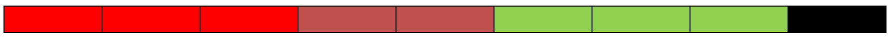
5.6.9.3 Lossless Compression Examples

5.6.9.3.1 Run Length Encoding (RLE)

Eliminates repeated data values by storing:

1. The value
2. And then the number of times it repeats

E.G encoding a bitmap



This translates into: 3 red, 2 purple, 3 green, 1 black

5.6.9.3.2 Dictionary-based methods

A dictionary is sent containing the desired items

Each item has an index

The compressed file contains the index of each item rather than the item itself

E.G. "HELLO"

Dictionary	
Entry	**Index**
H	0
E	1
L	2
0	3

Thus "HELLO" >> 01223

5.6.10 Encryption

5.6.10.1 Definition

Encryption – The transformation of data from one form to another in order to prevent an unauthorised third party from understanding it

$$\begin{tikzcd}\textbf{Plaintext} \arrow[r, shift left, "\text{Encryption}"] & \textbf{Cyphertext} \arrow[l, shift left, "\text{Decryption}"]\end{tikzcd}$$

Cypher – A particular method of encryption

Computational Security – the cipher cannot be broken in a period short enough for the information to still be valuable

5.6.10.2 Caesar Cypher

This is a type of substitution cypher

That shifts each letter a given number of positions along the alphabet

The shift distance is called the key

E.G. key of one:

hello >> ifmmp

BUT it is very easy to crack as only 25 possible values for the key

5.6.10.3 Vernam Cypher

A key *k* is shared with both parties

The XOR (modulo 2 addition) is conducted on the plaintext and key to encrypt it

To decrypt one must only XOR the cyphertext with the same key

ENCRYPTION								
Plaintext:	0	0	1	1	0	1	0	1
Key:	0	1	1	1	0	0	1	1
Ciphertext:	0	1	0	0	0	1	1	0

DECRYPTION								
Ciphertext:	0	1	0	0	0	1	1	0
Key:	0	1	1	1	0	0	1	1
Plaintext:	0	0	1	1	0	1	0	1

The key (aka One-Time Pad) must be:

1. Random
2. Used only once and destroyed after use (One-Time)
3. Equal to or longer in length to the plaintext

As the key is random and used only once, there is no mathematical relation between plain and cyphertext => PERFECT SECURITY (uncrackable). Vernam is the only cipher to be provably uncrackable.

6 Fundamentals of computer systems

6.1 Hardware & Software

Hardware – the physical components of a computer

Software – the set of programs written to make the computer function

Software enables the hardware to run

6.2 Classification of Software

Application Software – performs a function that would still be required if computers did not exist

Directly useful to the user

System Software – performs a function that would not be required if computers did not exist

<u>NOT</u> directly useful to the user

6.2.1 Application Software

There are 3 broad categories

- **General-purpose**
  - Has a variety of purposes (functions) for a range of different users
  - e.g. word processor, spreadsheet
- **Special-purpose**
  - Has a single very narrow purpose for a small set of users
  - e.g. firefly, payroll system, fingerprint scanner
- **Bespoke**
  - Custom made to the customer's specification
  - e.g. bank account software

As you descend the list:

- <u>More expensive</u> – as fewer customers, so development costs not spread out
- <u>Fits customer spec better</u> – as made with a narrower user base
- <u>More buggy</u> – as smaller user base, so less testing

6.2.2 System Software

6.2.2.1 Operating System (OS)

- **OS**
  - set of programs that:
    - hides the complexities of the hardware, by creating simplified virtual machine
    - manages computer hardware and software resources
    - provides the user with an interface to interact with the hardware
  - tasks it performs:
    - <u>Memory management</u>
      - Deciding what to keep/remove in RAM
    - <u>Processor scheduling</u>
      - Allocating which tasks get performed by the CPU to:
        - Maximise throughput & provide a reasonable response time

- Backing store management
    - Keep track of where files are kept in storage
    - And be able to manipulate them if necessary (move, delete etc)
- I/O device management
    - Managing the I/O puts from the various peripherals
- Interrupt handling
    - Similar to processor scheduling
    - If there is an urgent task, it will interrupt what the CPU is doing and perform it
- User Interface
    - Enables the user to interact with the hardware as intuitively as possible
    - Often use a GUI (Graphical User Interface)

6.2.2.2 Utility programs

Utility software – software designed to optimise the performance of the computer or perform basic functions such as backing-up files, compressing/decompressing files, firewall, encryption etc.

Examples:

- Disk defragmenter
    - Reorders the hard drive in order to make related data fragments reassemble next to eachother
    - Thus speeding up loading times as less moving of the read/write head
- Virus checker
    - Checks files for viruses and eliminates/quarantines them

6.2.2.3 Libraries

Library – a ready-compiled program that can be run when needed. A library normally contains pre-written functions, which can be invoked when needed without having to re-write the routine.

e.g. Console.ReadLine() is a function from the Console library

6.2.2.4 Translators

Source code >> Translator >> Object code

Which can then be executed

6.2.2.4.1 Assembler

Translates 2nd generation assembly code into its 1:1 machine code equivalent

Assembly Code >> Assembler >> Machine Code

6.2.2.4.2 Compiler

Translates high-level languages (e.g. C#) into machine code

The compiler first checks for any easy to spot errors in the code

And then translates it into machine code

High-level code >> Compiler >> Machine Code

The object code can be saved and executed without the presence of the compiler

6.2.2.4.3 Interpreter

Still works on high-level code

Rather than translating the entire source code into object code in one go

An interpreter goes through line by line translating and executing as it goes

BUT it normally performs some syntax checks before running

High-level code >> Interpreter >> Executed

6.2.2.4.4 Compilers vs Interpreters

- Advantages of the compiler
    - Faster to execute once compiled as no need to translate
    - The object code can be saved and run when required without translation
    - The object code is more secure as it cannot be read easily (without huge reverse engineering)
- Advantages of the interpreter
    - Useful for program development as
        - No need for a lengthy compilation after every alteration
        - And so easier to partially test / debug programs

Compilers are used in most commercial applications

Interpreters used in program development

6.2.2.4.5 ByteCode

Possible to combine compiling and interpretation

High-level code >> compiler >> Bytecode >> interpreter >> executes

The intermediate stage of bytecode is advantageous as:

- Platform independence
    - as the bytecode can be translated into any processor-specific machine code
- Faster execution than just an interpreter
    - As the bytecode is already partially translated
- Extra security
    - As it is interpreted there is less risk of malware infecting/crashing the computer

6.3 Classification of programming languages

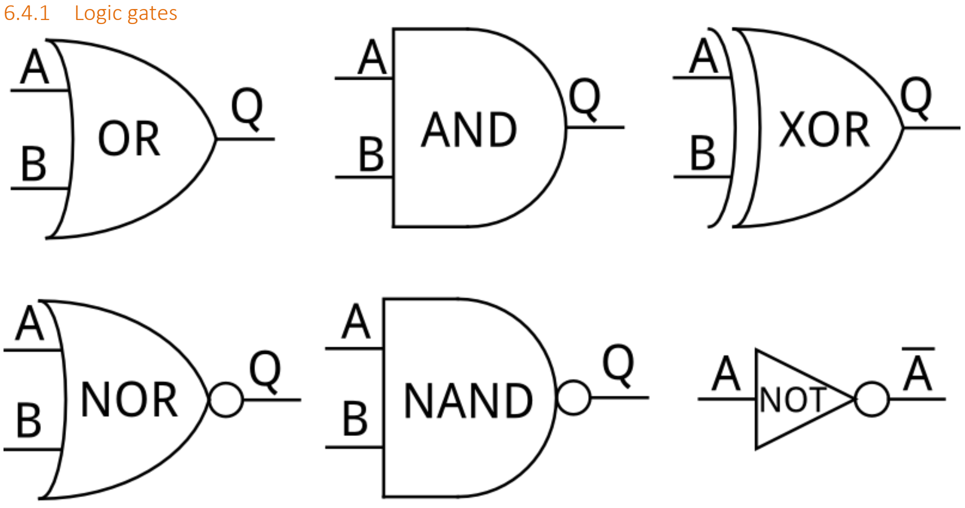
- **Low level**
  - In a format that closely mirrors the processor operations
    - <u>Machine code</u>
      - Strings of 1's and 0's directly fed into the processor
    - <u>Assembly code</u>
      - 1:1 correspondence with machine code
      - BUT it uses mnemonics in order to make it easier for a human programmer to remember and understand
- **High level**
  - Abstract from the processor operations
  - and so allow programmers to think more generally in terms of algorithms
  - without worrying about machine complexities
    - <u>Imperative</u>
      - Instructions executed in a programmer specified order
    - <u>Declarative</u> (not in spec)
      - Programmer simply provides rules which are then applied to the inputs
      - Only really used in A.I

6.3.1 Low-level vs High-level

- <u>Advantages of High-level</u>
  - Allow coders to think generally in algorithms and so it becomes easier
  - Much more compact as one high-level statement translates into many low level-ones => programmers can write much more complex programs
  - Used in most program development
- <u>Advantages of Low-level</u>
  - Can be used to fine tweak and optimise:
    - Execution speed
    - Size of the file
  - Can manipulate individual bits and bytes
  - Used for processor-specific functions or embedded systems

6.4 Logic

6.4.1 Logic gates



- **OR**
  - Q is high if A OR B are high
- **NOR**
  - Q is high if neither A NOR B are high
- **AND**
  - Q is high if A AND B are high
- **NAND**
  - Q is high if A AND B are NOT high
- **XOR**
  - Q is high if A OR B are high but NOT BOTH
- **NOT**
  - Q is high if A is low

A	**B**	**Q**					
		OR	NOR	AND	NAND	XOR	NOT (only A)
0	0	0	1	0	1	0	1
0	1	1	0	0	1	1	1
1	0	1	0	0	1	1	0
1	1	1	0	1	0	0	0

You must:

- Be familiar with drawing/interpreting logic circuits
- Complete a truth table for a given logic circuit
- Write a Boolean expression from a given logic circuit
- Draw a logic circuit from a given Boolean expression

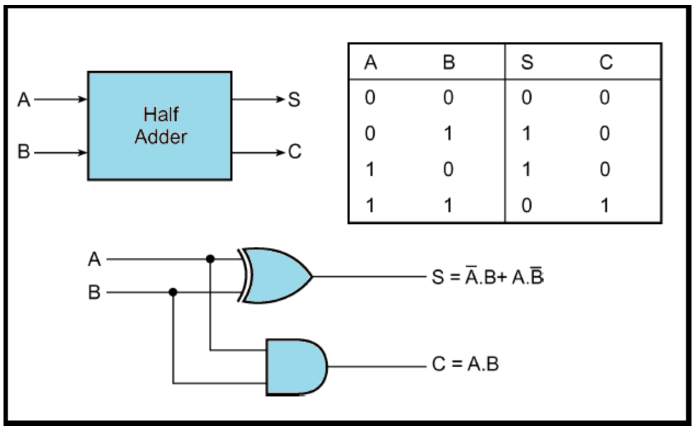
6.4.2 Specific Circuitry

6.4.2.1 Half-Adder

Performs addition operation from two inputs A, B

Two outputs containing the sum and a carry digit

>> If combined can form a very fast addition circuit >> Premise of full-adder



A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

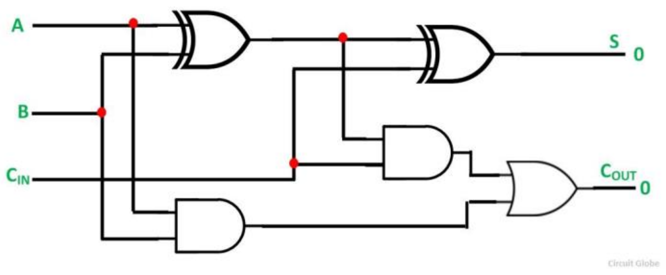
Must be able to recognise and reconstruct the half adder

6.4.2.2 Full-adder

Two half adders are combined to form a full-adder

It adds A, B and a carry digit C_{in} from a previous calculation

Has two outputs: sum S and carry out C_{out}



Inputs			Outputs	
A	B	CIN	COUT	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

6.4.2.3 Edge-triggered D-type flip-flop

The edge-triggered D-type flip-flop is used as a memory unit

Stores a bit

Do not need to know how it works

6.5 Boolean Algebra

Conventions:

$$A + B = A \text{ or } B$$

$$A \cdot B = A \text{ and } B$$

$$\bar{A} = not\ (A)$$

These all make sense + is a plus and $\cdot$ is a multiply

NOTE: $X \cdot Y = XY$

6.5.1.1 De Morgan's Laws

1st law:

$$\bar{A} \cdot \bar{B} = \overline{A + B}$$

2nd law:

$$\overline{A \cdot B} = \bar{A} + \bar{B}$$

6.5.1.2 Some further results

General stupid laws:

$$X \cdot 0 = 0$$

$$X \cdot 1 = X$$

$$X \cdot X = X$$

$$X + 0 = X$$

$$X + 1 = 1$$

$$X + X = X$$

$$\bar{\bar{X}} = X$$

Some clever laws:

$$X \cdot \bar{X} = 0$$

$$X + \bar{X} = 1$$

Commutative Law:

$$X \cdot Y = Y \cdot X$$

$$X + Y = Y + X$$

Associative Law:

$$X(YZ) = (XY)Z$$

$$X + (Y + Z) = (X + Y) + Z$$

Distributive Law:

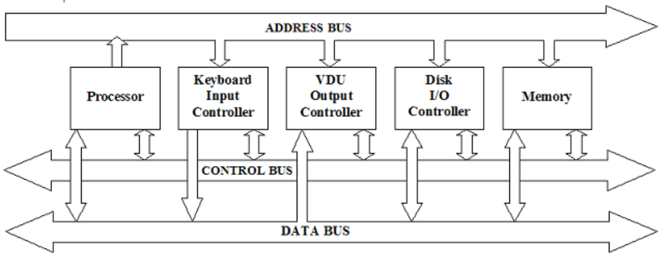
$$X(Y + Z) = XY + XZ$$

$$(W + X)(Y + Z) = WY + WZ + XY + XZ$$

7 Fundamentals of computer organisation and architecture

7.1 Internal hardware components of a computer

7.1.1 Some basic components

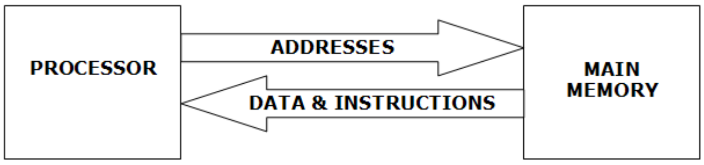


- **Processor**
  - effects the desired operations
- **Main memory**
  - stores the opcodes and operands
- **Buses**
  - A set of parallel wires connecting computer components
  - Signals transmitted in parallel along the various lines of each bus
  - **Address Bus**
    - used to access a specified location (address) in memory
    - The address bus contains the address in memory
    - Width of address bus determines maximum addressable memory capacity (32 lines => 2^{32} = 4 Gib)
  - **Data Bus**
    - Bi-directional path for moving data
    - Width of data bus is key for performance (8 lines => 1 byte moved at a time)
  - **Control Bus**
    - Ensures that there is no conflict over the use of the other buses (as they are shared) => transmits commands, timings and status info between components
    - Some example control lines:
      - *Memory Write*: data on data bus written to addressed location
      - *Memory Read*: data from the addressed location
      - *Bus Request*: A device is requesting data bus use
      - *Clock*: synchronises operations
      - *Reset*: initialises all components
- **Secondary storage**
  - Stores large amounts of data even when they are not needed in memory
  - e.g. Hard Drive Disk (HDD) Solid State Drive (SSD)
- **I/O controllers**
  - Interfaces between the processor and an input / output device
  - I/O controllers required to
    - Free up processing power
    - Ensure one peripheral is compatible with lots of processors

7.1.2 Von Neumann & Harvard

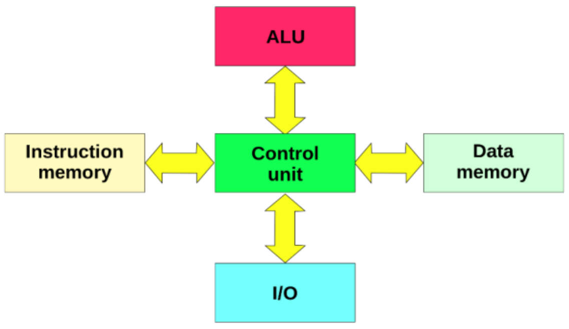
Von Neumann Architecture:

- Instructions (program) and Data (operands) are stored together in the same store (memory)
- Called the **stored program** concept



Harvard Architecture:

- Instructions and data are held separately



7.1.2.1 When to use each

Von Neumann:

- used in most computers as it is simplest

Harvard:

- used in embedded systems (special-purpose e.g. satnav)
- as potential to be faster
- as instructions and data fetched in parallel
- instructions held in Read Only Memory (ROM) and data held in Read/Write Memory

7.1.3 Addressable memory

RAM (Random Access Memory) is divided into a series of blocks

Each block has an address

The address can be used to refer to (and thus access) the data held within that box

7.2 The processor

7.2.1 Stored Program Concept

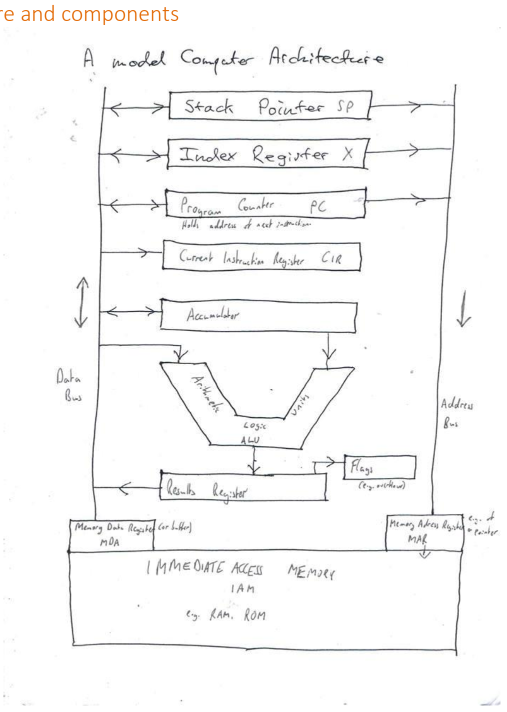
In Von Neumann machines

Machine code instructions are held in main memory together with the operands

Processor fetches and executes these serially (one-by-one)

By performing Arithmetic and Logical Operations

7.2.2 Processor structure and components



- **ALU**
    - Arithmetic Logic Unit
    - Performs arithmetic and logical operations on the data
    - e.g. + - x as well as AND OR XOR NOT etc
- **Control Unit**
    - Controls and coordinates activities of the CPU, such as data flow etc
    - Accepts next instruction, decodes it into several sequential steps, manages its execution and manages the output
- **Clock**
    - Synchronises operations
    - Switches between 1 and 0 at a few GHz
    - Each internal action takes 1 clock cycle
- **General Purpose Registers**

    - Very fast memory locations used as temporary stores
    - Labelled R0 – R…
    - If there is only one, referred to as the accumulator
- **Special Purpose Registers**
    - **PC**
        - Program Counter
        - Holds the address of the next instruction to be executed
        - Increments during the fetch-execute cycle
    - **CIR**
        - Current Instruction Register
        - Holds the current instruction being executed
    - **MAR**
        - Memory Address Register
        - Holds the address of the memory location which is to be accessed (either for reading or writing)
    - **MDR**
        - Memory Data Register aka Memory Buffer Register (MBR)
        - Temporarily holds the data read from / is to be written to memory
    - **SR**
        - Status Register
        - Contains various flags (bits) that are set to 0 or 1 depending on the output of the last operation
        - e.g. if there is an overflow a certain bit is set to 1

7.2.3 Fetch-Execute cycle

Fetch:

1. MAR ← PC

Address of next instruction copied from PC to MAR

2. MDR ← [MAR]

Contents of the memory address held in MAR copied to MDR

3. PC ← PC + 1

PC incremented to point to next instruction (for subsequent fetch-execute cycle)

4. CIR ← MDR

Contents of MDR (instruction) copied to CIR

Decode:

5. Instruction decoded

The CIR decodes the instruction and sets up the processor accordingly

Execute:

Instruction executed

7.3 Processor Instruction Set

Processor Instruction Set – the complete list of instructions supported by the specific hardware

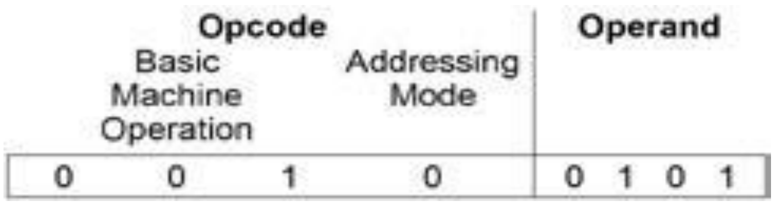
=> Different processors have different instruction sets in terms of

a) operations available
b) "language" in which they are written (what each opcode refers to)

7.3.1 A typical instruction

Consists of two parts:

1) **Opcode** – the type of operation/action (Operation Code)
   a) Basic Machine Operation
   b) Addressing Mode
2) **Operand** – what the Opcode is performed on



In more complicated systems the length of each operation can be increased

- ⇨ Larger instruction set
- ⇨ More than one operand

7.3.2 Addressing Modes

- **Immediate**
  - The operand holds the **actual** value to be operated on
- **Direct**
  - The operand holds the **memory address** of the value to be operated on

<u>e.g. for the above instruction</u>

Suppose 001 means Load the Accumulator

Suppose 0 means immediate addressing

Then 0101 simply refers to the number 5

Therefore, the above instruction is:

Accumulator ← 5

7.4 Machine / Assembly Code operations

1st generation: **Machine Code** – the instructions are written in binary => in the exact form computers would use

BUT impossible to remember. Therefore,

2nd generation: **Assembly Code** – Mnemonics used to help a human to remember/understand BUT still 1 to 1 correspondence with machine code

7.4.1 Some example Assembly code instructions

The exact mnemonics used will be defined in the question.

Some ones to be familiar with:

- Load
- Add
- Subtract
- Store
- Branching
  - Conditional
  - Unconditional
- Compare
- Logical bitwise operations (AND, OR, NOT, XOR)
- Shift right/left
- Halt

Addressing modes:

- Immediate - #operand
- Direct - operand

E.G. to add 7 the contents of memory location 1 and store in memory location 2

LDA 1

ADD #7

STA 2

7.5 Interrupts

At the end of each fetch-execute cycle, the processor checks to see if there are any interrupts which need to be handled.

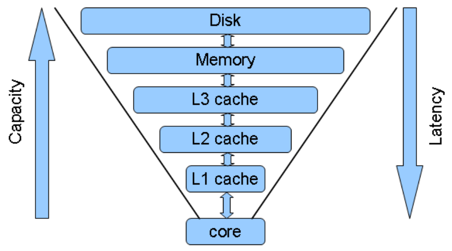
If there is an unhandled interrupt, this triggers the Interrupt Service Routine (ISR) which

- Halts the execution of the current task
- Pushes the current state variables onto the stack
- Services the interrupt
- Pulls the state off the stack
- Resumes operation

Interrupts are use to immediately service time-critical tasks e.g. printing mid-software update

7.6 Factors affecting processor performance

- **Multiple cores**
  - By using more than one processor, can operate in parallel => faster
  - BUT programs often do not exploit this
- **Cache memory**
  - Memory even faster than RAM held very near to the CPU
  - => the larger the cache, the faster data is extracted



- **Clock Speed**
  - The number of actions taken per second
  - => by overclocking, more instructions executed per second
- **Word Length**
  - Word length is the number of bits a CPU can process simultaneously (e.g. 32-bit, 64-bit)
  - => longer words, faster performance
- **Address bus width**
  - Wider this is, the more memory locations can be accessed per transmission
- **Data bus width**
  - Wider this is, the more bits of data can be sent per transmission

7.7 External hardware devices

7.7.1 Input and Output Devices

Each I/O device needs a suitable I/O controller

Know main characteristics, purposes, suitability and principles of operation of the following:

7.7.1.1 Barcode Reader

The barcode is divided into equal width segments

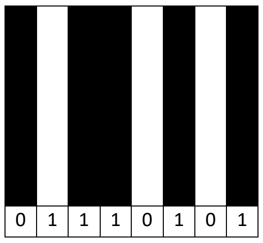
Each segment is coloured white or black to represent a 1 or 0 (respectively)

The black absorbs light whereas the white reflects it

The pattern of black and white bars is sensed by an optical reader

Often by shining a light and examining the reflection

This is then interpreted as a binary sequence



7.7.1.1.1 Types of readers

- Pen-type readers
    - The pen is dragged across the barcode
    - The tip contains a light source and an LDR => only triggered by white below
- Laser scanners
    - Same principle as the pen but they use a bar of laser light
- CCD readers (charge-coupled devices)
    - An array of hundreds of light sensors measures the intensity of light directly in front
    - Produces a voltage pattern identical to the barcode when examined sequentially
- Camera based readers
    - A photo is taken of the barcode
    - It is then processed to determine the bar pattern

7.7.1.2 Digital Camera

A digital camera uses a CCD or **CMOS** (Complementary Metal Oxide Semiconductor) sensor

This is a grid of millions of tiny light sensors

The shutter opens and an image is projected onto the sensor through the lens

Each sensor measures the colour and intensity of each pixel

This is represented in binary data

It is then saved as a bitmap

7.7.1.3 Laser Printer

The printer converts the page to a bitmap

It uses a laser to draw a negative (inverse) onto a negatively charged drum

The laser discharges the parts it hits

The drum then rotates past a hopper containing positively charged toner

The toner is only attracted to the negative parts of the drum (have not been lasered)

The toner is then transferred to the paper using pressure and heat

7.7.1.4 RFID (Radio Frequency IDentification)

An RFID chip contains a transponder and an antenna

These are used to emit radio waves at a given frequency pattern (encoding data)

This frequency pattern is decoded by a receiver (identifies the item)

- **Active**
    - There is an in-built power source used to generate the radio waves
    - Thus have a longer range than passive
    - Uses:
        - Cars through a motorway toll booth
        - Runners going through mile markers in a marathon
- **Passive**
    - No on-board power supply
    - Rely on the radio waves emitted from the reader (must be nearby ~1m) to provide sufficient energy to emit its own radio waves
    - Thus they are smaller and cheaper than active
    - Uses:
        - RFID tags in dogs
        - Anti-theft tags in shops
        - Oyster card / contactless payment

7.7.2 Secondary Storage Devices

7.7.2.1 Need for secondary storage

Primary store is Random Access Memory

BUT

RAM loses its contents once powered off

So there needs to be a secondary store

Need to know the:

- Main characteristics
- Purposes
- Suitability
- Understand principles of operation (maintain a toggle state of 0 or 1 without power)

Of the following devices:

7.7.2.2 Hard Disk Drive (HDD)

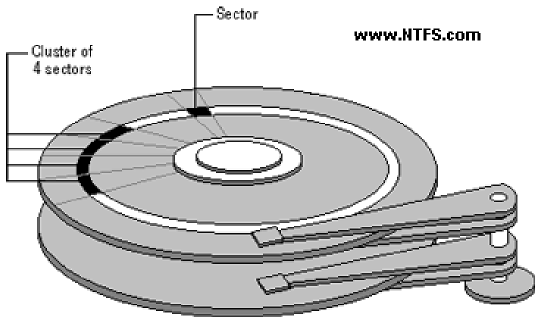
An HDD consists of a platter coated with magnetic material

The particles on the disk can be polarised north or south (represents 0 or 1)

Each disk is divided into concentric tracks and subdivided into sectors

A read-write head is able to sense or alter the polarisation of each bit

As the disk spins very fast (~10,000 RPM) the read-write head is able to access all the data on a track



The number of platters can be increased to increase capacity

7.7.2.3 Optical Disk Drive (ODD)

e.g. CD-ROM, DVD, Blu-Ray

A high power laser is used to burn parts of the surface to make them non-reflective

Reflective and non-reflective bits represent a 1 or a 0

They are arranged in a single track on a tight spiral (like a vinyl)

A low powered-laser is used to read the disk as it rotates

By measuring the intensity of the reflection the binary sequence can be reproduced

Rewriteable disks exist and they use some clever chemistry alongside magnetism to set a reflective or non-reflective state

7.7.2.4 Solid-State Disk (SSD)

Same standard dimensions of a hard drive to retain compatibility BUT completely different innards

No moving parts

SSD contains an array of chips containing **NAND flash memory cells** and a controller to manage it all

A NAND flash memory cell contains a floating gate transistor

They trap and store charge => toggled to 1

If they are left then they stay at 0

Rewriting:

No easy way to change a single 1 to a 0

In order to rewrite a section (page)

Must erase the entire page (set to 0)

And then insert the new values

7.7.3 Relative (dis)advantages of each

- **HDD** - laptops
    - Huge storage capacity
        - BUT
    - Long access times as head needs to move and disk needs to spin
    - Moving parts may get damaged over time

- **ODD** – digital media
    - Cheap to produce
    - Easy to package and distribute
        - BUT

    - Easily damaged by excessive sunlight or scratches

- **SSD** – mobile phones
    - Very fast access (low latency) as no need to move a read-write head
    - No moving parts therefore:
            - Less power consumption
            - More durable
            - Quieter
            - Cooler
        - BUT
    - Expensive
    - Lower storage capacity for now

8 Consequences of uses of computing

Show awareness of the current:

- individual (moral)
- social (ethical)
- legal
- cultural

Opportunities and risks of computing

- Technological developments have revolutionised communications and information flow through societies.
- This leads to an increased capacity to:
    - monitor behaviour
    - amass and analyse personal information
    - distribute, publish, communicate and disseminate personal information
- computer scientists therefore have great power and thus responsibility
- software/algorithms can have moral and cultural consequences
- As computers are now present worldwide. Computer scientist have a huge scale of impact: for better or worse

It is very difficult for legislators as they must consider between:

- Personal privacy
- Societal security
- Rapidly advancing technology, leading to stronger cloaking

8.1 Case studies

8.1.1 Edward Snowden and the NSA (National Security Agency)

2013 Edward Snowden leaked documents about the NSA's surveillance capabilities

He broke the law and his contract in doing so, under **US Espionage Act 1917**

PRISM – the surveillance programme gave them ability to

Query and intercept almost any form of communication, such as:

- Phone calls
- Emails
- Text messages

Arguments for PRISM:

- Heightened national security

Arguments against PRISM:

- Breach of personal privacy
- Right to anonymity online
- The tool may fall into the wrong hands e.g. Frank Underwood to win an election

8.1.2 Cyber-attacks
Cyber-attacks are becoming increasingly common

8.1.2.1 PSN outage 2011
PSN hacked by **LulzSec**

Just done for fun and to show how poor the security was

Compromised personally identifiable details of **77m** accounts

Incredibly easy as the details were not encrypted

PSN down April 20 - May 3

Cost Sony **$171m**

Benefits:

- Tighter security on PSN now

Negatives:

- Personal details compromised
- PSN went down for 2 weeks

8.1.2.2 Estonia cyber-attacks 2007
Attacks aimed to cripple the ability for Estonia to function normally

Hacked and downed many websites for: parliament, banks, ministries, newspapers and broadcasters

Carried out by a private group BUT sponsored by the **Russian** government

In retaliation for the relocation of Bronze Soldier of Tallinn, a Soviet-era grave-marker

Very frightening as first example of **political cyber-warfare**

Further to that,

Among Snowden’s leaked docs, was President Obama’s directive to prepare offensive cyber-operations.

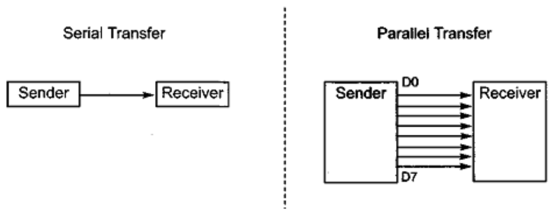
9 Fundamentals of communication and networking

9.1 Communication

9.1.1 Serial v Parallel transmission

Serial – bits are sent one at a time through a single wire

Parallel – several bits are sent simultaneously over multiple parallel wires



Advantages of serial over parallel:

- Size reduction in cabling reduces cost
- More reliable
    - no risk of crosstalk (interference between parallel wires – worsened at higher frequencies)
    - no risk of skew over long distances (path difference => different arrival times)
- Can transmit at higher frequency => higher baud rate => sometimes faster

BUT parallel is often faster

9.1.2 Synchronous v Asynchronous

Synchronous – data is sent at regular intervals, timed by a clocking signal

- ⇨ Steady and reliable transmission
- ⇨ e.g. real-time video

Asynchronous – data is sent in small chunks (often characters) at unspecified intervals

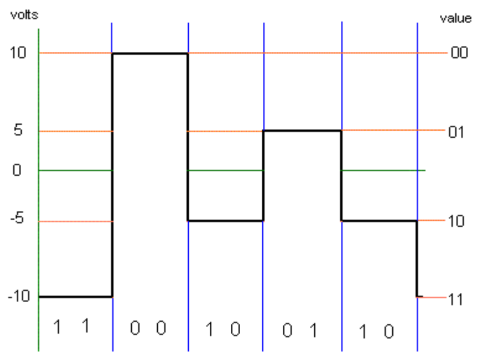
- ⇨ start bit present to
    - alert the receiver of an incoming signal
    - synchronises the clocks => go at the same baud
- ⇨ stop bit present to
    - signal the end of the data chunk
    - so it does not interpret the subsequent low as a string of 0's
- ⇨ parity bit often present as insurance

- ⇨ Fast, cheap & flexible transmission
- ⇨ Used in e.g. PC's

9.1.3 Communication basics

- **Baud rate**
  - number of signal changes / second
- **Bit rate**
  - number of bits sent / second

(Bit Rate) = (Baud Rate) x (number of bits per signal)



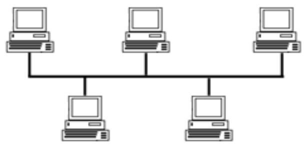
if there are 4 voltage levels, each signal can represent 2 bits

- **Bandwidth**
  - amount of data transferred per unit time
  - Equivalent to bitrate but sometimes different units
  - e.g. Megabits per second (Mbps)
- **Latency**
  - Time delay between sending and receiving
  - Normally just distance travelled / speed of light
- **Protocol**
  - A set of rules relating to communication between devices
  - Allows two devices to communicate by normalising the data format
  - e.g.
    - baud rate
    - parity type
    - parallel / serial etc

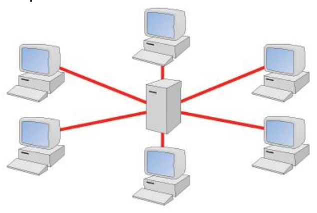
9.2 Networking

9.2.1 Network topology

- **LAN** – Local Area Network (e.g. school computers)
- **Bus topology**
  - All computers are connected to a single common cable
  - Data is transmitted only in one direction at one time
  - All stations (nodes) receive all the traffic
  - All stations have equal transmission priority and must request access to use the bus



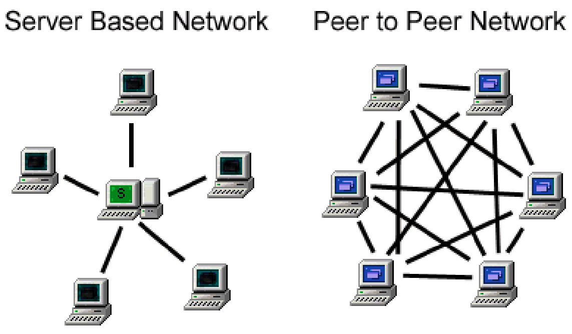
  - CSMA/CD (Carrier Sense Multiple Access with Collision Detection)
    - If two stations try to transmit simultaneously
    - Both stop and wait a random time before trying to retransmit
    - This is also used in wireless networks
  - Advantages
    - Inexpensive to install as less cable and no additional hardware
    - New devices can easily be added
    - Well suited to small networks (not requiring high speeds)
  - Disadvantages
    - If main cable fails, whole network goes down
    - Slow performance with heavy traffic
    - Low security (all stations can see all traffic)
- **Star topology**
  - Contains a central node (normally a switch, hub or computer) which acts as a router to transmit messages by selectively linking computers



  - Advantages
    - If a cable fails, only one station affected => easy to isolate faults
    - Consistently fast performance (even with heavy traffic)
    - No problems about data collisions
    - More secure (only sender, switch and receiver see data)
    - Different stations can transmit at different speeds
    - Easy to add new stations
    - Can behave like a bus using appropriate switching
  - Disadvantages
    - Expensive due to immense length of cable and cost of switch
    - If the central switch goes down, the whole system goes down

9.2.2 Types of networking between hosts

- **Peer-to-Peer (P2P)**
  - No central server
  - Computers are connected directly to each other as peers (equal status)
  - So they can share files, data etc.
  - <u>Advantages</u>
    - Cheap to setup
    - Easy to run
    - More anonymity
    - Enables sharing of resources
  - Disadvantages
    - Facilitates piracy (e.g. Napster, Popcorn Time)
    - Less regulation
  - <u>Examples</u>
    - Online gaming (e.g. old Call of Duty)
    - Small LAN's (Home, Small office etc.)
    - Torrenting



- **Client-Server**
  - Multiple computers (**clients**)
  - Connected to a powerful central computer (**server**)
  - Client requests a service from the server
  - <u>Advantages</u>
    - Better security as files and access to them is centralised
    - Backups easier to perform as data is centralised
    - Resources/data can be shared easily
  - <u>Disadvantages</u>
    - Expensive to install due to hardware
    - Expensive to maintain as it requires IT staff to manage
  - <u>Examples</u>
    - Print server – manages print requests
    - Email server – stores emails
    - File server – stores files
    - Web server – manages access to the web

9.2.3 Wireless networking

9.2.3.1 WiFi

WiFi is a local area wireless technology

Enables devices to connect to

The internet or a network resource (e.g. a printer)

Via a Wireless network Access Point (**WAP**)

At a short-ish range (20m indoors, more outdoors)

WiFi is an internationally standardised technology => compatible everywhere

9.2.3.2 Components required for wireless networking

- **Wireless network adapter** aka Wireless Network Interface Controller
    - A piece of hardware that enables a computer to
        - communicate wirelessly
        - via radio signals
        - when tuned in to a specific frequency
    - Most adapters are built in to the computer (e.g. smartphone, laptop etc.)
    - Station = Computer + Adapter i.e. a station is a wirelessly capable computer
- **Wireless Access Point (WAP)**
    - This is the part that emits and receives the Wi-Fi radio signals
    - To make the right connections and connect each device to the requested part of the internet, a router is required (similar to star network)
    - BUT the WAP and router are often built in to the same device

- **Modem**
    - Device that translates
    - Analogue signals from the telephone line
    - Into digital signals
    - The internet "comes from the telephone line"

9.2.3.3 Securing a wireless network

- **WPA** & **WPA2** (WiFi Protected Access v1 and v2)
  - Security protocols
  - Used to encrypt transmissions
  - WPA2 built in to Wireless Network Adapters
- **SSID** (Service Set IDentification)
  - The informal name of the network (used to identify and remember it)
  - e.g. BTHub5-HX5X. User must enter a password to gain access
- **MAC address whitelist**
  - Media Access Control address is unique to each network card and thus computer
  - A whitelist can be set up to only grant certain MAC addresses access

9.2.3.4 Collision avoidance

Carrier Sense Multiple Access with Collision Avoidance (**CSMA/CA**)

9.2.3.4.1 Without RTS/CTS

```
START

If (channel is idle)

{

    Transmit Data & END

}

Wait random amount of time

Go back to START
```

9.2.3.4.2 With RTS/CTS

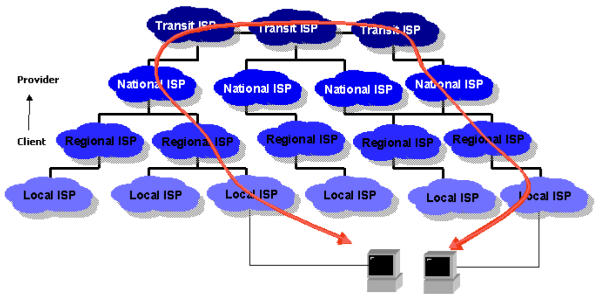
RTS – Request to Send   CTS – Clear to Send   reliable as removes hidden nodes problem

```
START

If (channel is idle)

{

    Transmit RTS

    If (CTS received)

    {

        Transmit Data & END

    }

}

Wait random amount of time

Go back to START
```

9.3 The Internet

9.3.1 How the internet works

Structure of the internet – interconnected web of machines that communicate with each other via cables. Can have some degree of hierarchy as to who owns which pipes



Packet switching – splitting up of data into small chunks (packets) which are then sent out through the network and reassembled on arrival at the destination not necessarily taking the same path. This is better for shared networks as somebody does not require continuous use of a pipe.

Router – a bridge between at least two networks (e.g. LAN and ISP's network). The router receives a packet and forwards it to the next router until the destination is reached. The router may inspect the IP address and so favour particular routes if it has seen it before. Each jump between routers is called a hop. The router must have an internet unique IP address >> must be assigned by the registrar.

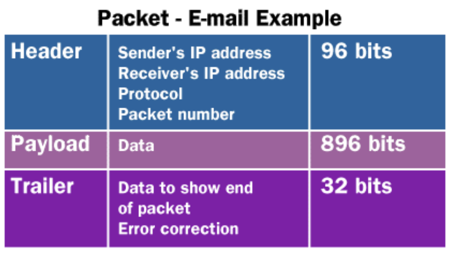
Gateway – similar purpose to a router for bridging networks. BUT it is used when the two networks are using different protocols >> the gateway translates the header data so that it can be consistent with each. For example, the gateway is what marries the WiFi protocols with IP

Packet – small chunk of message to be sent (~1KB). Consists of a Header, Payload and Trailer.

The header will have the sender and receiver IP address and ports, as well as the protocol used, the packet number in the sequence and details of the hop limit.

The payload has the data to be sent

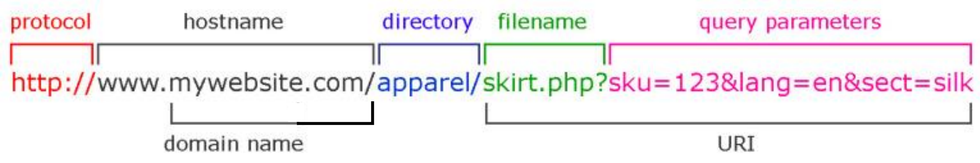
The trailer often has some form of error correction or detection (e.g. checksum) to show the data has not been corrupted or otherwise fiddled with.


Packet - E-mail Example

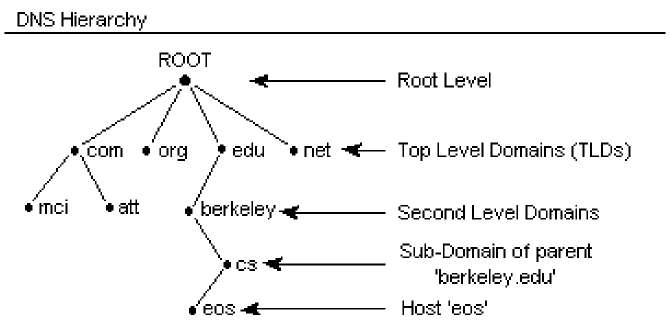
Header	Sender's IP address Receiver's IP address Protocol Packet number	96 bits
Payload	Data	896 bits
Trailer	Data to show end of packet Error correction	32 bits

Time To Live (TTL) – the maximum number of hops the packet can go through before being discarded (aka hop limit)

Uniform Resource Locator (URL) – full address for an internet resource. It instructs the browser how to request a particular file from a particular web server


protocol
hostname
directory
filename
query parameters
http://www.mywebsite.com/apparel/skirt.php?sku=123&lang=en&sect=silk
domain name
URI

Domain Name – identifies the area (domain) that an internet resource resides in. These are structured hierarchically, starting at the root (invisible trailing '.'), then the Top Level Domains (TLDs) and so on.



Fully Qualified Domain Name (FQDN) – an FQDN also includes the hostname as well as the domain name of the website

FQDN = host name + domain name

mail.google.com

IP address – a unique address that identifies a network device. It is the home mail address of a machine, as it indicates where to send the packet to. If a domain name is associated with a specific IP address then the server the website resides on has that IP address.

IPv4 (32-bits) It is split into 4 chunks of 1 byte separated by a '.' IPv6 (128-bits) has substantially more.

130.142.37.108

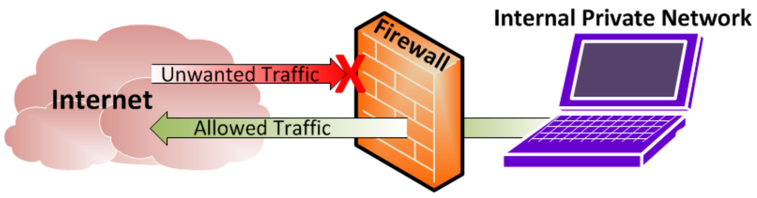
Domain Name Server (DNS) – there needs to be a system for translating from an FQDN to an IP address. When a browser is trying to access "www.example.com" it sends a DNS request to the Domain Name Server which contains a big list of domain names and corresponding IP addresses. If the first server does not know it then asks another server such as the corresponding TLD server. The browser then gets sent the translated IP address and can start sending packets.

Internet Registry – 5 organisations governed by ICANN (Internet Corporation for Assigned Names and Numbers) that hold databases of domain names and IP addresses (among other things) to prove ownership, map between them and make sure they are all distinct

Internet Registrar – an organisation (like GoDaddy) that sells domain names to registrants

9.3.2 Internet Security

Firewall – security checkpoint to prevent unauthorised access (entry/exit) between two networks. The firewall is typically a separate machine with two Network Interface Cards (one internal and one external) and analyses each packet that tries to make it across. A firewall can also act as a proxy server.



Packet Filtering – denies access based on the source/destination IP address. Can also deny access to particular ports that are designated for unwanted protocols e.g. Port 23 for Telnet used for remote access. Packet filtering is often called static filtering as it does not change much, can have an IP whitelist or blacklist and not much else.

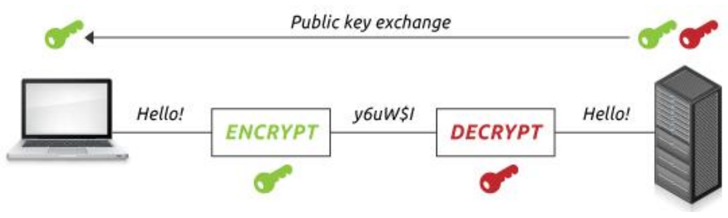
Stateful Inspection – applies more intelligent rules to decide if a packet is malicious. Such as examining the payload contents, or checking to see if a machine on the trusted side of the network initiated the conversation. Called dynamic filtering as it is more subject to change.

Denied packet –can either be dropped (quietly removed) or rejected (rejection notice sent back to sender).

Proxy Server – a server which your traffic goes through and alters your header data such that your IP address remains hidden to the destination of your packets – and nobody knows which IP addresses you are visiting. Firewalls can be configured to act as proxies.

Encryption – converting plaintext into ciphertext such that only the intended recipient can translate it back

- **Symmetric (Private key)**
  - Same key is used by both parties to encrypt and decrypt
  - The key must be exchanged first (key exchange) >> this is often done using asymmetric encryption
- **Asymmetric (Public key)**
  - Denote Public Key p and Private Key q
  - p and q are related but it is almost impossible to guess one from the other
  - Plaintext <p> Ciphertext <q> Plaintext
  - Above chain can go in either direction
  - p is made public to everyone >> to send a message to the owner of the q, simply encrypt your message with p and send
  - The owner of the q will be able to decrypt using q



Digital Signature – There is still a problem of somebody using your public key to pretend to be you >> need some way to prove ownership of the private key >> digital signatures.

Digest calculated from hash of plaintext message. The digest is now encrypted using the sender's private key. The whole message can then be optionally encrypted further. The recipient now uses the sender's public key to decrypt the digest and also recalculates the digest from the received plaintext. If the two values match up it proves that

a) the message came from the owner of the private key

b) it was not modified en-route

Digital Certificate – these are issued by official Certificate Authorities (CAs) and verify the trustworthiness of a sender or website. The certificate contains a serial number, the expiry date, name of holder, their public key and the Cas digital signature to verify the certificate is not spoofed. It is effectively a stamp that approves the website as genuine.

Malware – malicious software such as: worms, trojans and viruses

Virus – self-replicating malware. Attaches itself to on other files to spread

Worm – also self-replicating malware. But it is standalone software

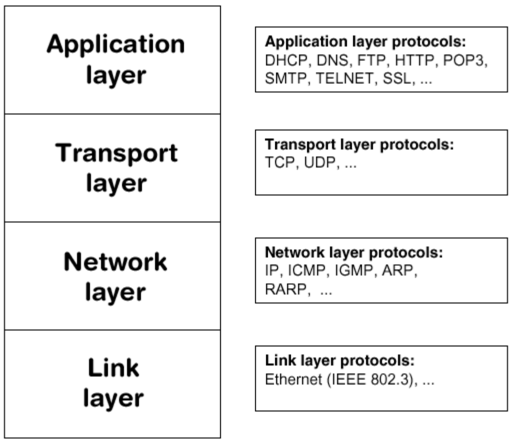
Trojan – Malware disguised as a helpful program (such as a game) and the payload is delivered when installed. Often used to open a backdoor for other exploits.

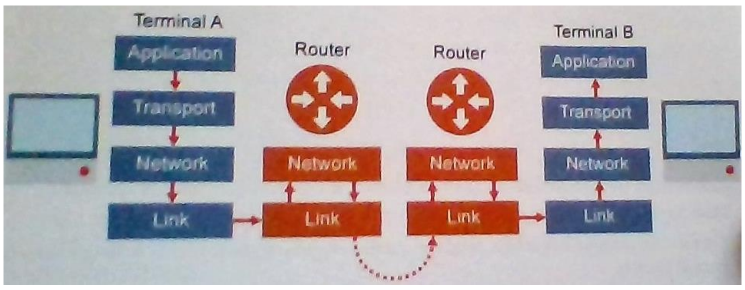
Vulnerability – flaws in security that may be exploited. Either due to user error (not renewing antivirus, switching off the firewall) or software bugs (such as the Heartbleed bug or admin privileges)

Protection – Can improve code quality such that fewer bugs are able to be exploited. For example, avoiding buffer overflow attacks or zip bombs. Awareness is key to protect against social engineering (phishing, casual conversations) and to ensure users update their software and antivirus (to protect against the newest malware)

9.3.3 The Transmission Control Protocol/Internet Protocol (TCP/IP) protocol

Various different protocols operate at different layers of the TCP/IP stack (used to send packets across a network). Descending each layer puts another level of wrapping and ascending takes them off.


Application layer
Application layer protocols: DHCP, DNS, FTP, HTTP, POP3, SMTP, TELNET, SSL, ...
Transport layer
Transport layer protocols: TCP, UDP, ...
Network layer
Network layer protocols: IP, ICMP, IGMP, ARP, RARP, ...
Link layer
Link layer protocols: Ethernet (IEEE 802.3), ...



Layers:

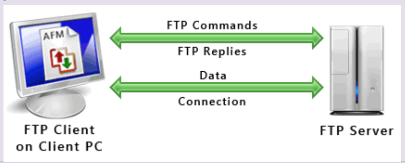
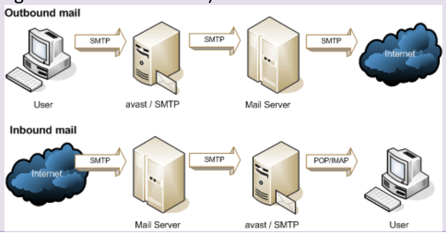
- **Application Layer** – used by high-level applications (e.g. a browser) to ask that a particular piece of data be sent across the network. It simply selects the right high-level protocol (e.g. http) and makes the request
- **Transport Layer** – this splits the data into packets using the Transmission Control Protocol (TCP) to establish an end-to end connection. Each packet is numbered out of the total number of packets and the destination port (entry point to destination machine) is specified. The transport layer ensures that all the packets have arrived and are correct and makes necessary requests for retransmission if not.
- **Network Layer** – aka the IP or internet layer. This adds the source and destination IP addresses. Routers operate on the Network layer to know where to forward the packets.
- **Link Layer** – this adds the Media Access Control (MAC) addresses for the Network Interface Cards (NICs) of the source and destination for that hop only (i.e. the two routers along the way). Establishes the physical pipe to send the packets down.

Socket – an IP address plus a port equals a socket. If a packet is a letter then the name of the receiver is the port, the building is the IP address. You must know who to deliver to. The port is just the entry point to the machine

Socket = 192.168.1.1 : 80 = IP address : port

Media Access Control (MAC) – a unique 12-digit hex code (96 bits) that uniquely identifies a Network Interface Card. Therefore, it uniquely identifies any networked device (printer, phone, computer etc). This allows data packets to be sent directly to a particular NIC.

00-71-5B-A9-38-4A

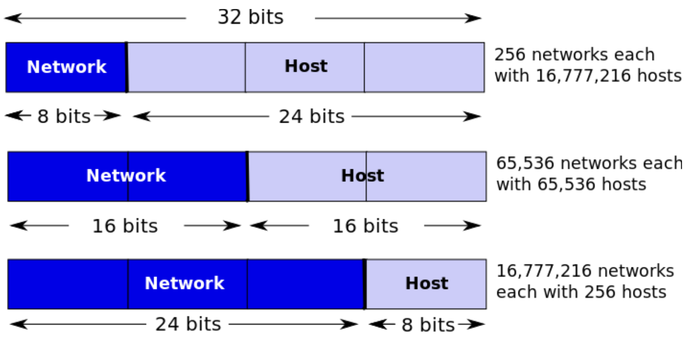
Server port number	Protocol	Description
20	FTP – data	File Transfer Protocol. Used to transfer large files efficiently across a network. Port 20 is used for sending data.  FTP Commands FTP Replies Data Connection FTP Client on Client PC FTP Server
21	FTP – control	Port 21 sends control instructions for FTP. An FTP server is the machine with the files you are trying to access. An FTP client is the piece of software you are using to browse and access those files (can be configured to require login or anonymous)
22	SSH	Secure Shell. Used to remotely login to / access a computer and feed it commands. It is entirely encrypted and legitimate. Using SSH you can make a TCP connection to a remote port to make requests using other protocols
23	Telnet	Also used for remote access but is unencrypted and often used for hacking
25	SMTP	Simple Mail Transfer Protocol. Used to send emails to a mail server (either when sending or when transferring between mail servers)  Outbound mail User SMTP avast / SMTP SMTP Mail Server SMTP Internet Inbound mail Internet SMTP Mail Server SMTP avast / SMTP POP/IMAP User
110	POP3	Post Office Protocol v3. Used to retrieve emails from a mail server, once they are retrieved they are saved on your machine and deleted from the server >> hard to maintain synchronicity between devices
143	IMAP	Internet Message Access Protocol. Also to retrieve emails from mail server but the emails are not deleted form the server so you can access them from different devices.
80 & 8080	HTTP	HyperText Transfer Protocol. The web server receives an HTTP request for files needed to display the webpage. These resources includes: html, css, javascript as well as media files. The browser makes as many http requests are necessary until it has all the files it needs to render the webpage.
443	HTTPS	HyperText Transfer Protocol Secure. Same as HTTP but encrypted

The return port on the client machine is often randomly chosen >> stop hackers knowing which ports are open on a client machine.

9.3.4 IP addresses

IP address – unique numerical address used to identify a machine communicating over IP

Structure – IP addresses are split into a network identifier and host identifier. With more or less bits being devoted to each depending on the size of the network



32 bits
Network
Host
256 networks each with 16,777,216 hosts
8 bits
24 bits
Network
Host
65,536 networks each with 65,536 hosts
16 bits
16 bits
Network
Host
16,777,216 networks each with 256 hosts
24 bits
8 bits

IPv4 – 32 bits >> only 4 billion machines >> running out >> IPv6 invented. IPv4 addresses are written in '.' Separated byte chunks

172.16.114.35

IPv6 – 128 bits >> many more devices (represented as hex string)

3DFB:1730:4935:0007:0340:FE2F:FB71:48AF

Subnet masking – a method to find the network identifier of an IP address. For example, if a network has all IP addresses of form 112.38.38.X , where X is any integer 0-255, then the subnet mask would be 255.255.255.0. In order to retrieve the network identifier you simply AND the mask and the IP address

Source IP (A)	112	38	38	27
Destination IP (B)	112	38	38	3
Subnet Mask (C)	255	255	255	0
$A \cdot C$	112	38	38	0
$B \cdot C$	112	38	38	0

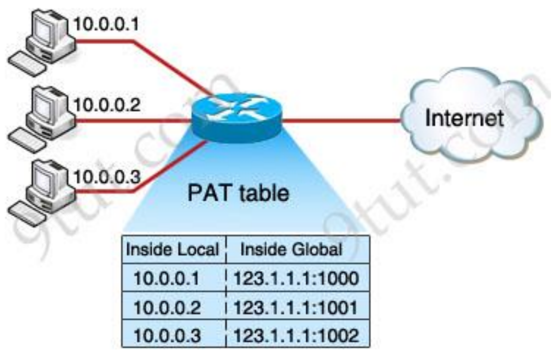
It is used to decide whether the source and destination IP addresses are on the same subnetwork >> no need to send the packet out through a router to the external web. They are on the same subnet if A.C = A.B

Public IP address – these are routable >> can be reached by any machine on the internet. This is because they are unique (and so must be registered by the internet registry)

Private IP address – for space limitations of IPv4 and pain of registering IP addresses, private networks often use private IP addresses that are local to that network >> are un-routable. The standard for private networks is: 192.168.1.X.

BUT clearly if a machine on this private network wants to communicate with the outside world some kind of translation between private and public addresses is required; this called Network Address Translation (NAT)

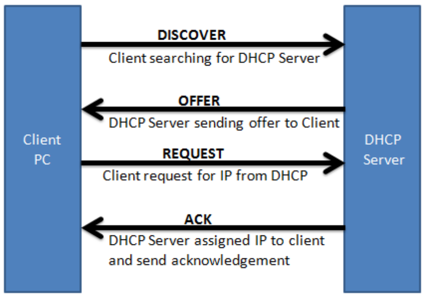
Network Address Translation (NAT) – this is used to convert between public and private IP addresses. Every time a packet goes through a router to the outside world, the router rebadges the source IP address to point to the router itself >> when a response is received wanting to get to that IP address, the router looks up in its table what local address that corresponds to and sends it on its way.



Inside Local	Inside Global
10.0.0.1	123.1.1.1:1000
10.0.0.2	123.1.1.1:1001
10.0.0.3	123.1.1.1:1002

Port forwarding – this is a product of NAT. When a router receives an incoming request for a particular port it has to forward it to the correct machine and port on the internal network

Dynamic Host Configuration Protocol (DHCP) – a method to dynamically assign an IP address from a pool of unused ones to a machine that is trying to operate on a public network. This is because IPv4 addresses are scarce. When a machine wants a public IP address it makes a request to the DHCP server for one (details in picture below). DHCP is also used on private networks (e.g. home wi-fi) to prevent overuse of addresses.


Client PC
DHCP Server
DISCOVER
Client searching for DHCP Server
OFFER
DHCP Server sending offer to Client
REQUEST
Client request for IP from DHCP
ACK
DHCP Server assigned IP to client and send acknowledgement

9.3.5 Client server model

Client server model – a client sends a server a request message and the server returns an appropriate response message

Websocket protocol – the websocket specification defines an API (Application Programming Interface) to establish a persistent bi-directional connection between client and server over TCP. The persistent connection allows for fast two-way asynchronous communication >> often used for gaming

CRUD applications – web CRUD applications allow access to a relational database, offering these four functions at a minimum

	CRUD	HTTP request methods	SQL database fucntion
C	Create	POST	INSERT
R	Retrieve	GET	SELECT
U	Update	PUT	UPDATE
D	Destroy	DELETE	DELETE

Representational State Transfer (REST) – a style of systems design that demands the use of HTTP request to interact with an online database via a web server. The client does not need to know how the requests are handled. A system or API that conforms to the REST specification is RESTful

Connecting to a database from a browser w/ HTTP:

1. Browser makes HTTP request(s) to web server to load a standard web page and resources
2. HTML file contains reference to the JavaScript file
3. JavaScript is executed client side to call the RESTful API
4. The database server responds with the data in XML or JSON format
5. The browser deals with the XML or JSON accordingly

Advantages of JSON over XML:

1. Easier for a human to read
2. More compact
3. Easier to create
4. Easier/faster for computers to parse

Thin/Thick-client computing – the thicker a client the more processing it does compared to the server.

Advantages of thin	Disadvantages of thin
More secure as data centralised	Totally reliant on server >> what if server is overloaded
Easy to add more dumb terminals	Often cannot run more powerful applications
Software updates need only be made on server	Requires a very powerful server
Clients can have lower specs	

10 Fundamentals of databases

10.1 Entity Relationship diagrams

Produce a data model from given data requirements for a simple scenario involving multiple entities

Entity – a category of object about which data is to be recorded

Describing entities: Entity1 (Primary-Key, Attribute1, Attribute2…)

Relational Database – data is held in a collection of tables linked by common attributes

Attribute – a particular data value an entity has

Record – an instance of an entity (row)

Primary key – an attribute that uniquely identifies a particular record (underlined in entity definition)

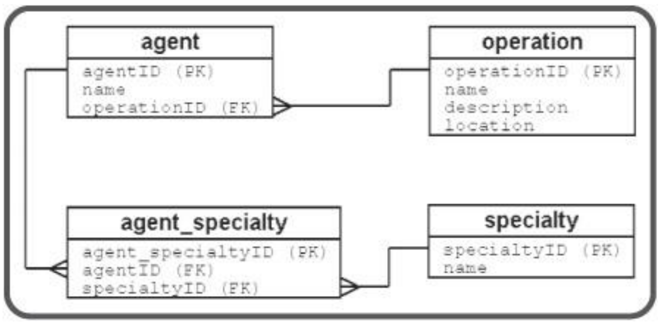
Composite primary key – multiple attributes used as primary key (also underlined)

Foreign key – to link two records you include the primary key of one as an attribute of the other >> this is called a foreign key

Entity relations:

1. One-to-one : straight line
2. One-to-many : straight line to fork
3. Many-to-many : double fork

Example entity relationship diagram:



10.2 Normalisation

Normal Forms:

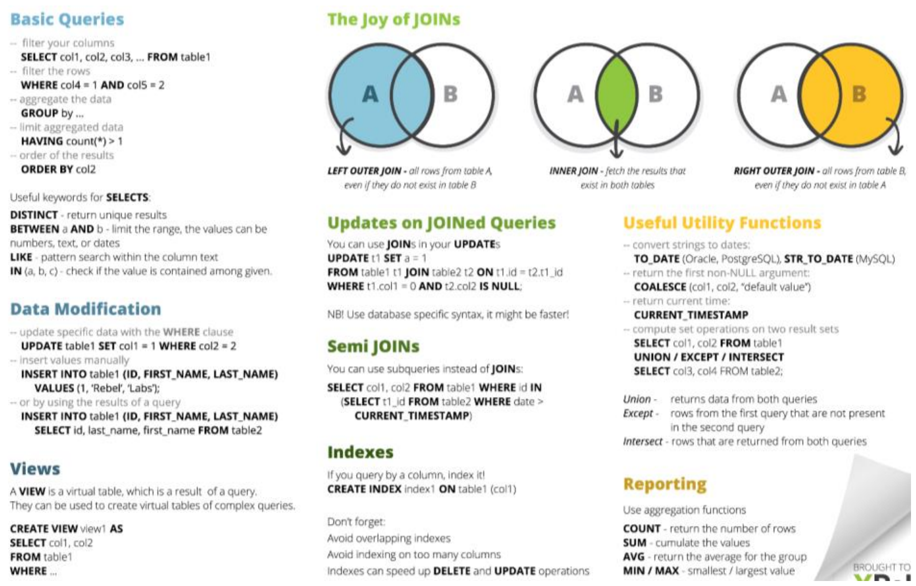
1. **No repeating attributes or groups of attributes**
    a. e.g. no TelephoneA, TelephoneB...
    b. This is a many to many relation >> introduce link table
    c. Such as the agent-specialty entity above
2. **No partial key dependencies**
    a. An attribute depends on only part of a composite key >> move into other table
3. **No non-key dependencies**
    a. The value of an attribute is dependant on a non-key attribute
    b. All attributes must depend on the key, the whole key and nothing but the key (so help me Codd)

Why normalise:

- **Modification**
    - Easier to update as only a change in place is needed >> quicker and no update anomalies where inconsistent data is kept
    - E.g. a normalised customer entity means that a change in address needs only be made once rather than on each invoice
- **Faster Querying**
    - Smaller tables with fewer attributes
    - >> faster searching and sorting as well as reduced storage requirements
- **Deleting Records**
    - A well-defined normalised databse will not allow records to be deleted on only one side of a relation
    - E.g. a customer cannot be deleted from the customer table if he still has unpaid invoices

10.3 Structured Query Language (SQL) >> PRACTICE

Use SQL to retrieve, update, insert and delete data from multiple tables of a relational database


A
B
A
B
A
B
BROUGHT TO

Basic Queries

```
-- filter your columns
   SELECT col1, col2, col3, ... FROM table1
-- filter the rows
   WHERE col4 = 1 AND col5 = 2
-- aggregate the data
   GROUP by ...
-- limit aggregated data
   HAVING count(*) > 1
-- order of the results
   ORDER BY col2
```

Useful keywords for **SELECTS**:

DISTINCT - return unique results
BETWEEN a **AND** b - limit the range, the values can be numbers, text, or dates
LIKE - pattern search within the column text
IN (a, b, c) - check if the value is contained among given.

Data Modification

```
-- update specific data with the WHERE clause
   UPDATE table1 SET col1 = 1 WHERE col2 = 2
-- insert values manually
   INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)
      VALUES (1, 'Rebel', 'Labs');
-- or by using the results of a query
   INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)
      SELECT id, last_name, first_name FROM table2
```

Views

A **VIEW** is a virtual table, which is a result of a query.
They can be used to create virtual tables of complex queries.

```
CREATE VIEW view1 AS
SELECT col1, col2
FROM table1
WHERE ...
```

The Joy of JOINs

***LEFT OUTER JOIN** - all rows from table A, even if they do not exist in table B*

***INNER JOIN** - fetch the results that exist in both tables*

***RIGHT OUTER JOIN** - all rows from table B, even if they do not exist in table A*

Updates on JOINed Queries

You can use **JOIN**s in your **UPDATE**s

```
UPDATE t1 SET a = 1
FROM table1 t1 JOIN table2 t2 ON t1.id = t2.t1_id
WHERE t1.col1 = 0 AND t2.col2 IS NULL;
```

NB! Use database specific syntax, it might be faster!

Semi JOINs

You can use subqueries instead of **JOIN**s:

```
SELECT col1, col2 FROM table1 WHERE id IN
  (SELECT t1_id FROM table2 WHERE date >
    CURRENT_TIMESTAMP)
```

Indexes

If you query by a column, index it!
CREATE INDEX index1 **ON** table1 (col1)

Don't forget:
Avoid overlapping indexes
Avoid indexing on too many columns
Indexes can speed up **DELETE** and **UPDATE** operations

Useful Utility Functions

```
-- convert strings to dates:
   TO_DATE (Oracle, PostgreSQL), STR_TO_DATE (MySQL)
-- return the first non-NULL argument:
   COALESCE (col1, col2, "default value")
-- return current time:
   CURRENT_TIMESTAMP
-- compute set operations on two result sets
   SELECT col1, col2 FROM table1
   UNION / EXCEPT / INTERSECT
   SELECT col3, col4 FROM table2;
```

Union - returns data from both queries
Except - rows from the first query that are not present in the second query
Intersect - rows that are returned from both queries

Reporting

Use aggregation functions

COUNT - return the number of rows
SUM - cumulate the values
AVG - return the average for the group
MIN / MAX - smallest / largest value

```
SELECT a.id, b.id

FROM tableA as a JOIN tableB as b

ON a.id = b.id

WHERE a.name LIKE “Harry”

ORDER BY ASC
```

Use SQL to define a database table (note the value types)

```
CREATE TABLE Employee

{

        EmpID INT PRIMARY KEY,

        EmpName VarChar(20) NOT NULL, // string of length 20

        HireDate DATE,

        StartTime TIME,

        Height FLOAT,

        Male BOOLEAN,

        Salary CURRENCY

}
```

10.4 Client Server Databases

Client server database – provides simultaneous access to the server held database for multiple clients. A centralised database avoids issues of inconsistency if private copies are edited. More convenient for users. Access rights and backups can be managed centrally.

BUT since multiple people are trying to access the database you may run into issues if two people try to update a record at the same time. How to deal with it:

- **Record locks**
    - When a user accesses a record a lock is put on it such that no other users can access it
    - BUT may end up in deadlock where two users want the other person's locked resource
- **Serialisation**
    - Ensures transactions cannot overlap in time as they only happen one at a time >> a transaction cannot start until the previous one is finished
    - Often implemented using Timestamp ordering
- **Timestamp ordering**
    - Every object has a read and a write timestamp >> if you read from an object and when you come to write to it and the most recent read timestamp is not the one you made then you can flag there is an issue
- **Commitment Ordering**
    - Transactions are not lost when two users simultaneously access the same object >> transactions are ordered in terms of their dependencies as well as time >> can avoid deadlock

11 Big Data

Big Data – catch all term for data that does not fit in usual containers

Properties:

- **Volume** – too big to fit on a single server
- **Velocity** – data coming in very quickly (e.g. continuously streamed from networked appliances)
- **Variety** – in many forms

Hardest aspect of analysis is its lack of structure as:

- Algorithms need to be complex (machine learning often used to discern patterns)
- Cannot be structured in a relational database format (plus its size means it cannot be held on a single machine >> relational databases hard to share)

When it cannot fit onto a single server:

- Processing must be distributed across multiple machines
- Functional programming favoured as:
  - Easy to write correct and efficient distributed code
  - Statelessness makes it easily parallelisable
  - Immutable data structures avoid update anomalies

Fact based model – alternative to databases. Immutable facts which capture a single piece of information (e.g. a person's name) are recorded with a timestamp >> infinitely scalable and easy to look into past

Graph schema – a graph database data is stored as nodes and relationships (edges) between them each of which have properties

12 Fundamentals of functional programming

A function f maps an item from set D (**Domain**) to an element in set R (**Range** or Co-domain) according to a set of rules

$$f: d \in D \rightarrow r \in R$$

The type of D is the argument type and the type of R is the result type

First-class object – objects which may:

- Appear in expressions
- Be assigned to a variable
- Be assigned as arguments
- Be returned in function calls

A function is a first-class object in a functional programming language

Function application – f(arg) is the application of function f to the argument arg

Argument – what a function works on. f(x,y) can be said to take two arguments but really it just takes the pair (x,y)

Partial functions – a function that operates on a restricted domain >> more specific function

Add a b:

Return a + b

$$Add: integer \rightarrow (integer \rightarrow integer)$$

The function add has two arguments which returns another partial function (addA) which takes one argument which returns an integer

For example the call:

Add 3 7

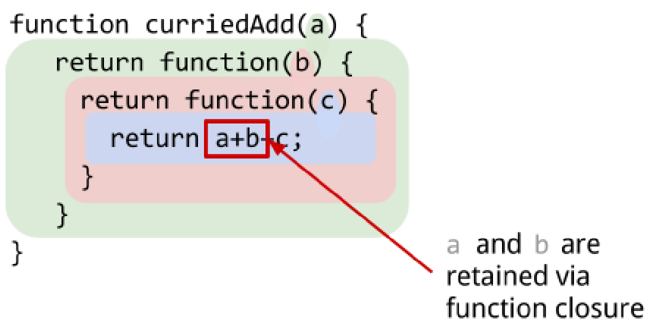
Is equivalent to:

Add7To b:

    Return 7 + b

And the call: Add7To 3

>> functions only take one argument at a time and return another partial function which goes on and on



```
function curriedAdd(a) {
   return function(b) {
     return function(c) {
       return a+b+c;
     }
   }
}
```

4.12.1.4 Partial function application – do not understand

Commented [LT1]: Clarify

Content	Additional information
Know what is meant by partial function application for one, two and three argument functions and be able to use the notations shown opposite.	The function *add* takes two *integers* as arguments and gives an *integer* as a result. Viewed as follows in the partial function application scheme: *add: integer → (integer → integer)* add 4 returns a function which when applied to another integer adds 4 to that integer. The brackets may be dropped so function *add* becomes add: *integer → integer → integer* The function add is now viewed as taking one argument after another and returning a result of data type *integer*.

Composition of functions – g(f(x)) is the composition of g and f >> first do f(x) then g(x). This is effectively a new function h(x)

12.1 Writing functional programs

Show experience of constructing simple programs in a functional programming language (e.g. F#)

Higher-order functions – takes a function as an argument or returns a function as a result or both

Have experience using the following higher order functions:

- **Map** – applies given function to each member of a list and returns the new list
- **Filter** – processes a data structure and returns a new one that contains only elements that match a given condition
- **Reduce or Fold** – reduces a list of values to a single value by repeatedly applying a combining function to the values

12.2 Lists in functional programming

List can be expressed as a concatenation of head and tail, where the head is an item and the tail is another list

[1,2,3] = 1:[2,3]

List = Head:Tail

The list can then be recursively defined given the base case of an empty list []

[1,2,3] = 1:[2,3]

[2,3] = 2:[3]

[3] = 3:[]

Describe and apply following operations:

- Return head of list
- Return tail of list
- Test for empty list
- Return length of list
- Construct an empty list
- Prepend an item to a list
- Append an item to a list

Have experience of writing functional programs for these operations >> PRACTICE

13 Systematic approach to problem solving

Analysis >> Design >> Implementation >> Testing >> Evaluation

13.1 Analysis

Before a computer-based problem can be solved it must be:

- Fully defined
- The requirements of the computer system to solve it must be established
- A data model created

Analysis may require prototyping (agile development)

13.2 Design

Before constructing a solution, it should be defined and specified:

- Planning data structures
- Designing algorithms
- Dividing up steps to solution
- Designing UI

Design can be iterative (agile)

13.3 Implementation

The solution must be implemented in two parts:

- Data structures
- Instructions (code) that a computer can understand

13.4 Testing

The solution must be tested for correctness using test data that is:

- Normal
- Boundary (edge of range)
- Erroneous (wrong format)

13.5 Evaluation

The solution must be evaluated against the original specification.

Example criteria might be:

- Runtime
- Data format
- UI