Specification Content

5 Fundamentals of Data Representation

5.1 Number Systems

4.5.1.1 • The set of **natural numbers** are all positive whole numbers including 0.

$\mathbb{N} = \{0, 1, 2, 3, ...\}$

4.5.1.2 • The set of **integers** are all positive and negative whole numbers, including 0.

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

4.5.1.3 • The set of **rational numbers** is the set of all numbers that can be expressed as the ratio of two integers.

$\mathbb{Q}$

4.5.1.4 • The set of **irrational numbers** is the set of all numbers that cannot be expressed as the ratio of two integers.

4.5.1.5 • The set of **Real numbers** is the set made by combining the rationals with the irrationals. It's the set of all "possible real world possibilities"

$\mathbb{R}$

4.5.1.6 • **Ordinals** are numbers used to identify the position relative to other numbers. For example, use of *First*, *Second*, *Third*, etc, is essentially use of ordinal numbers.

4.5.1.7 We are use to using natural and real numbers in every day life, as we use natural numbers to count objects, and real numbers for measurements.

5.2 Number bases

4.5.2.1 A number base indicates the number of digits available within a system, e.g. base 10 for decimal, base 2 for binary. The accepted method for representing different number bases is by using subscript numbers to represent their base. e.g:

- 67_{10} Decimal (base 10)
- 10011011_2 Binary (base 2)
- AE_{16} Hexadecimal (base 16)

Hexadecimal is useful because it allows us to represent large numbers with less digits, e.g: $1101001_2 = 211_{10} = D3_{16}$. Through some calculations, you can show that binary number of length n will be converted to a decimal number of length $\log_{10}(2)n$ which will be converted to a hexadecimal number of length $\frac{n}{4}$. This means that 8 bits (= 1 byte) can be written as two hex digits. This is also the reason why hexadecimal is used to show memory addresses and colour codes.

Decimal	Binary	Hexadecimal
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10

Here are a list of conversions which you need to be comfortable with:

- Binary $\rightarrow$ Decimal
  1. Write down the binary number with spaces between each bit.
  2. Above the Least significant bit write the number 1.
  3. As you move left from the LSB, double the value of the previous number.
  4. Wherever there's a one on the bottom, add those decimal values together.

  For Example:

$2^7 = 128$	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
1	1	0	0	1	1	1	1

  $11001111_2 = 128 + 64 + 8 + 4 + 2 + 1 = 207$

- Decimal $\rightarrow$ Binary
  1. Divide by two and note down the remainder until you get the result 0 remainder 0.
  2. reverse the order in which you noted down the remainders. For example:

207/2	=103	remainder	**1**
103/2	=51	remainder	**1**
51/2	=25	remainder	**1**
25/2	=12	remainder	**1**
12/2	=6	remainder	**0**
6/2	=3	remainder	**0**
3/2	=1	remainder	**1**
1/2	=0	remainder	**1**
0/2	=0	remainder	**0**

  $207_{10} = 11001111_{16}$

- Hexadecimal $\rightarrow$ Decimal
  1. Write down the hexadecimal number with spaces between each hex digit.
  2. Beneath each hex digit, write down the decimal equivalent.
  3. Above the Least significant bit write the number 1.
  4. As you move left from the LSB, multiply the value by 16 the value of the previous number.
  5. multiply the value in the bottom row with the corresponding value in the top row, add these decimal values together.

  For Example:

$16^2 = 256$	$16^1 = 16$	$16^0 = 1$
9	0	A
9	0	10

  $90A_{16} = 9 \times 256 + 0 \times 16 + 10 \times 1 = 2314_{10}$

- Decimal $\rightarrow$ Hexadecimal
  1. Divide by 16 and note down the remainder until you get the result 0 remainder 0.
  2. Convert the remainders to hex digits.

3. reverse the order in which you noted down the remainders. For example:

2314/16	=144	remainder	10	**A**
144/16	=9	remainder	0	**0**
9/16	=0	remainder	9	**9**
0/16	=0	remainder	0	**0**

$2314_{10} = 90A_{16}$

- Binary → Hexadecimal

1. Write out the binary number, splitting it into blocks of four from the right.
2. Convert each of the binary blocks into hexadecimal (which can be done via a conversion to decimal)
3. Stick all of the blocks together to make one hexadecimal number.

1100	1010	1110	0111
12	10	14	7
C	A	E	7

$1100101011100111_2 = CAE7_{16}$

- Hexadecimal → Binary

1. Write out the hexadecimal number with spaces between each digit.
2. Convert each digit from hexadecimal to binary, which can be done via decimal, or just remembering how each binary number convert to hex digits. MAKE SURE ALL THE BLOCKS ARE FOUR DIGITS LONG.
3. Put all the blocks of binary numbers together to form the binary number

C	A	E	7
12	10	14	7
1100	1010	1110	0111

$CAE7_{16} = 1100101011100111_2$

5.3 Units of Information

4.5.3.1 A binary digit (bit) is the fundamental unit of information, and can take the value 0 or 1. due to the fact that one byte can only store 2 possible values (0 or 1), bytes are more often used which consist of 8 bits (meaning they can store 256 different values). We can calculate the number of different values which can be obtained from using n bits by calculating 2^n. From this we can see that

- If we use 3 bits, we can get $2^3 = 8$ different values.
- if we use 8 bits (1 byte), we can get $2^8 = 256$ different values.
- if we use 16 bits (2 bytes), we can get $2^{16} = 65,536$ different values.
- if we use 32 bits (4 bytes), we can get $2^{32} = 4,294,967,296$ different values.

4.5.3.2 A unit describes a grouping of bits, for example one byte is unit of 8 bits. You can use prefixes to dictate the size of a unit. The prefixes used fall into the categories of binary and decimal, because binary prefixes can be written as 2^n whereas decimal prefixes can be written as 10^n, these prefixes are approximately equal because $2^{10} \approxeq 10^3 \implies 1024 \approxeq 1000$. All the prefixes we need to know are:

Binary			Decimal		
Kibi	Ki	2^{10}	Kilo	K	10^3
Mebi	Ki	2^{20}	Mega	K	10^6
Gibi	Ki	2^{30}	Giga	K	10^9
Tebi	Ki	2^{40}	Tera	K	10^{12}

5.4 Binary Number System

4.5.4.1 Unsigned binary numbers are binary numbers that have no sign, they are all positive. On the other hand, signed binary numbers are numbers that do have a sign, they can be positive or negative. We know from above that is an unsigned binary number has n bits, it can represent 2^n values, meaning it could represent the values $1 \to 2^n$ however, if we want the first binary number it represents to be 0, then the range of values n bits can represent is $0 \to 2^n - 1$.

4.5.4.2 To form binary addition we can simply perform column addition. For example, to do $11000101_2 + 10100011_2$ we would do:

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

We can do the something similar for multiplication, in that we perform column multiplication. For example, to do $100101_2 \times 101_2$ we would do:

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

4.5.4.3 As said above signed binary numbers can be used to represent both positive and negative numbers, one possible way of coding this is by using two's complement. Within two's complement if you decide to use n bits, then the range of values that can be used is $-2^{n-1} \to 2^{n-1} - 1$. To convert a binary two's complement to decimal, we do a similar method to convert an unsigned binary number to decimal, except we make the most significant bit negative, for example:

$2^7 = -128$	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
1	1	0	0	1	1	1	1

$11001111_2 = -128 + 64 + 8 + 4 + 2 + 1 = -49_{10}$

To convert a positive integer into a two's complement, you simply use the method to convert a decimal number to binary, making sure the MSB is zero. To convert a negative integer into a two's complement, you do the following:

- Make the number positive (e.g. $-58 \to 58$)
- Convert the positive number into a positive two's complement (as described above)
- Starting from the LSB, working to the left, write down all the digits including the first 1 you come across.
- After the first 1, reverse all of the bits ($1 \to 0$, $0 \to 1$)

For example:

$-49_{10} \to 49_{10}$

$49_{10} = 0110001_2$

$-49_{10} = 1001111_2$

To perform subtraction you simply convert one of the numbers into two's complement. For example:

$102_{10} - 73_{10} = 01100110_2 - (01001001_2)$

$-01001001_2 = 10110111_2 \implies 01100110_2 - 01001001_2 = 01100110_2 + 10110111_2$

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

Mantissa									Exponent			
0	0	0	1	0	1	1	0		0	1	0	1

$01100110_2 - (01001001_2) = 00011101_2 = 29_{10}$

4.5.4.4 Numbers with a fractional part can be represented in many ways, one way to represent them is using fixed point binary numbers. This method means that the binary is in a fixed place and the programmer can determine how many are before and after the binary point. Conversion from binary (signed or unsigned) to decimal is similar to always, except you place the one before the decimal place, doubling as you go to the left, and halving as you move right. To convert from decimal to binary, you can break it into two parts:

1. The part before the binary point (the integer)
2. The part after the binary point (the fraction)

You can deal with part 1 by simply using the original method to convert from decimal to binary, to deal with part 2, you will have to try and seperate the fraction to show it as the sum of fractions in the form $\frac{1}{2^n}$.
Another way to represent numbers that have a fractional part is to use floating point numbers. This method allows the binary point to move around to allow for a larger range of values than is available within fixed point numbers, the decimal equivalent is scientific notation, where a number like 2960000000000 will be represented as 2.96×10^{12} or 0.000000184 will be represented as 1.84×10^{-7}. In the first example, we can see that the notation can be broken up into two parts, the mantissa 2.96 and the exponent 12, the exponent tells how to move the decimal point, positive is to the right, negative is to the left. In floating point we use a similar system, using an 8 bit mantissa and a 4 bit exponent and we look at the floating point number 000101100101, we can write this more clearly as We can see now that the exponent is +5 so there are two ways which we could procede:

1. Convert the mantissa to decimal (being aware that there is a binary pint after the first bit), then multiply the answer by 2 to the power of the exponent, which in this case would be $2^5 = 32$
2. write out the mantissa with a binary point after the first bit, then move the binary point in accordance to the exponent (negative to the left, positive to the right), then convert the resulting number into decimal

The second method is less error prone and is general easier to carry out and will get you marks in the paper, however if you feel you understand the maths behind it, the first method can be much faster (generally in the case of negative exponents). Before we continue, I will note that in the exam the mantissa and exponent are both given in two's complement, so that both positive and negative numbers can be represented. Below I will go through both methods:
METHOD 1
$0.0010110_2 = \frac{1}{2^3} + \frac{1}{2^5} + \frac{1}{2^6} = 0.171875_{10}$
$0.171875 \times 2^5 = 5.5_{10}$
so we get a final answer of 5.5
METHOD 2
$0.0010110_2 \implies 000101.10_2 = 5.5_{10}$
so we yet again get a final answer of 5.5
Now to convert from a decimal number into a normalised floating point binary, you basically do the second method from above, but in reverse so you:

1. Write your number as a binary fixed point number in two's complement (the number of bits you use to write the number doesn't matter too much at this step)
2. Normalise the fixed point number, which involves moving the binary point so that the first bit of the mantissa after the binary point is opposite to the MSB (meaning normalised negative numbers begin 10 and normalised positive numbers 01)
3. Write down the exponent to say how far the binary point needs to be moved to undo the action (for example if you move the binary point to the right 3 places, to undo this you would need to move the binary point 3 places to the left, so you would write the binary equivalent to -3 in two's complement).

Example 1 - Positive Number (6.75) $6.75_{10} = 0110.11_2$
Normalise so $0110.11 \implies 0.11011$
The binary point was moved 2 places to the left so the exponent is $2_{10} = 010_2$ Now, we are going to use an 8 bit mantissa and 4 bit exponent, so we are going to have to extend the numbers
For the mantissa, we can append zeros to the end and have no change in the value so $0.11011 \implies 0.110110$
For the exponent, we can repeatedly append the MSB (in this case a 0) to the beginning of the number so $010 \implies 0010$
Putting the two parts together and removing the binary point, we get the final answer 01101100010
Example 2 - Negative Number (-0.09375) $-0.09375_{10} = 1.11101_2$
Normalise so $1.11101 \implies 1111.01 = 1.01$
The binary point was moved 3 places to the right so the exponent is $-3_{10} = 101_2$ Now, we are going to use an 8 bit mantissa and 4 bit exponent, so we are going to have to extend the numbers
For the mantissa, we can append zeros to the end and have no change in the value so $1.01 \implies 1.0100000$
For the exponent, we can repeatedly append the MSB (in this case a 1) to the beginning of the number so $101 \implies 1101$
Putting the two parts together and removing the binary point, we get the final answer 101000001101

4.5.4.5 Rounding Errors occur when you can't represent a number exactly due to the number of bits that we use, and example is decimal is trying to write out $\frac{1}{3}$, if we only were allowed to use 4 decimal places, we would write $\frac{1}{3}$ as 0.3333, or if we had 10 decimal places, we would write it as 0.3333333333. In both of the above, there is some rounding of the results as $\frac{1}{3}$ can only represented in decimal using infinite decimal places, so if we use any finite number of decimal places, we will always have some small error due to this rounding. The same thing can happen in binary, for example if you try and write out 0.1_{10} in binary, then you'll find that it has an infinite binary expansion, and cutting it off at any point will cause some sort of rounding error. Rounding errors often occur in floating point number when you try and add two numbers of vastly different magnitudes, for example, $1000000_{10} + 0.000001_{10} = 1000000_{10}$ may occur if you're only allowed 7 significant figures maximum.

4.5.4.6 Absolute and Relative Errors are ways to measure the difference between the number to be represented and the approximation of that number in binary. The easier of the two to explain is the absolute error, this is simply the difference between the two, and can be memorised as $|\text{actual} - \text{expected}|$ where $||$ means the sign doesn't matter. This is often a useful measure for accuracy for numbers around 1, but not so much as for numbers of very large or small magnitude, as the absolute error will be larger for large numbers and smaller for small numbers, but this doesn't mean that they are necessarily inaccurate. Thus we talk about the relative error, which is a way to calculate the error of a number relative to its size meaning that, for example that a large number with a absolute error of 0.5 would have a smaller relative error than a number with an absolute error of 0.5, the formula is $|\frac{\text{actual}-\text{expected}}{\text{expected}}| = |\frac{\text{absolute error}}{\text{expected}}|$

4.5.4.7 Fixed point and floating point numbers can have their own relative advantages, for floating point numbers this is:

- A much wider range of numbers can be represented compared to a fixed point number with the same number of bits
- Useful when a wide range of numbers need to represented

Fixed point numbers have the following advantages

- No manipulation of binary point allows for quicker calculations
- The absolute error is always the same, whereas with floating point numbers, the absolute value will vary with the magnitude of the number
- Useful for applications which require speed over precision (gaming or signal processing)
- Useful when an absolute level of precision is needed, e.g. currency where you only need to go down to pennies.

4.5.4.8 The point of normalising a floating point number is to ensure that there is only one way to represent a number, and that the way in which it is represented is as efficient as possible, you can tell if a number is normalised by seeing if the first two bits are the same or not, if they are the same, then the number is not normalised. There is a quick way to normalise non-normalised floating point numbers: while the first two bits are the same, strip the first bit and subtract one from the exponent.

4.5.4.9 Underflow occurs when a number is too small to represented in a given the number of bits allocated, for example, the smallest positive value for an 8 bit fixed point number (with binary point directly in the middle) is 0.0625, so if you wanted to represent the number 0.05, it would be too small to be represented, thus causing an underflow error. Overflow occurs when a number is too large to be represented in a given number of bit allocated, for example, the largest positive value for an 8 bit two's complement number is 127, so if you wanted to represent the number 128, the number represented within the computer would be -128, and thus an overflow error would occur.

5.5 Information Coding Systems

4.5.5.1 When using character codes (which are binary codes that represent different characters) We need to realise that the binary representation of 9_{10} does not equal 1001_2 but has a completely different value, due to the fact it is representing the character "9", not the value of 9.

4.5.5.2 Before standardised coding systems, different programs would use different character codes for each character, which made it harder to understand different programs as they may use completely different character codes to represent the same character.
For this reason, ASCII (American Standard Code for Information Interchange) was invented to create a standard for all the keys on a keyboard. ASCII uses 7 bits meaning it could show 128 different characters, extended ASCII uses 8 bits meaning it could show 256 different characters. ASCII has many limitation:

- It only allows a maximum of 256 characters, which isn't sufficient to store all possible and necessary characters, numbers and symbols.
- It was initially developed in English, so it doesn't represent other languages and scripts of the world.
- Worldwide use of the internet made a universal international coding system more important.
- The range of platforms and programs has drastically increased, meaning developers from around the world are using a wider range of characters.

Because of this, a new standard was created called Unicode. Unicode is a superset of ASCII (or you could say ASCII was subsumed by Unicode) meaning for the first 256 characters (8 bits), the two are the same. For example in ASCII the character code for "A" is 65 so in Unicode the character code for "A" is also 65. Unicode comes in two different forms UTF-8 which uses 8 bits, and therefore resembles extended ASCII, and UTF-16 which uses 16 bits, meaning it can store 65536 different characters, making it very useful for storing all characters worldwide.

4.5.5.3 There are several ways of checking whether a sequence of bits has an error. Three ways of doing this are:

- Parity Bit

  This is a method of checking binary codes by counting the number of 0s and 1s in the code, parity always depends on the number of 1s in a number. There are 2 different types of parities

  - Even Parity

    This process makes sure there are an even number of 1s in the number. This means that if there are an odd number of 1s within the number, the parity bit will be 1, but if there are an even number of 1s, then the parity bit will be 0.
  - Odd Parity

    This process makes sure there are an odd number of 1s in the number. This means that if there are an odd number of 1s within the number, the parity bit will be 0, but if there are an even number of 1s, then the parity bit will be 1.

- Majority Voting

  This is a method of checking binary codes via producinng the same data serveral (3) times and taking the majority. So for example if you wanted to send the data 1001, we would send the data 111000000111, however due to possible interferences the actual data sent may be 101001000111. We can split the data into blocks of three and we get

  - $101 \to 1$
  - $001 \to 0$
  - $000 \to 0$
  - $111 \to 1$

  meaning we get the final datum of 1001.

- Checksums

  This process involves calculating some sort of sum from the data to be sent, and sending this data as a type of validation, to be checked by the receiving party. This is used within packet switching to check that all of the data within the packets have been transmitted quickly.

- Check Digits

  This has many uses outside of binary, the central idea is that a digit is added to the end of the number, which is the result of some sort of simple arithmetic calculation, this calculation can then be done again at the other end to check whether the number transmitted is valid. A system using weights is often used as it usually means that both changing values and translation (the position of numbers changing)

5.6 Representing Images, Sound and Other Data

4.5.6.1 Different patterns of bits can be used to represent different forms of data for example, in an image each pixel may be given 3 bytes (1 byte for red, 1 byte for green, 1 byte for blue), whereas a sound may use 1 byte to represent an amplitude every set time period (e.g. every 0.01 seconds).

4.5.6.2 Analogue data are data that are infinitely variable and are often represented in the form of a wave, for example a sound wave, whereas digital data are often represented by discrete values (often 1 and 0). Analogue signals are signals that can vary in both frequency and amplitude whereas digital signals is described as using binary (0s and 1s), and therefore, cannot take on any fractional values.

4.5.6.3 There are two main conversion to do with audio:

- Analogue to Digital Converter

  An ADC works by taking a series of readings are taken from the wave at fixed intervals in order to create discrete data values, which can be easily converted to binary. These readings can then be stored as binary codes. This is called sampling because you don't take the reading every time the amplitude changes, instead you use set points.

- Digital to Analogue Converter

  A DAC is usually embedded in a device that plays the audio data and signal is passed in analogue form to the loudspeaker or headphones, which then play the audio out loud.

4.5.6.4 Bit-mapped graphics are images made up of individual pixels, with each pixel being represented by a set number of bits. A pixel is an individual picture element. Resolution is width (of image in pixels) $\times$ height (of image in pixels). Colour depth is the number of bits allocated to represent the colour of a pixel in a bit-mapped graphic. We can also calculate the number of pixels per inch (PPI) by dividing the number of pixels in a given direction, by the length of the screen in that direction. For example a monitor that is 12×9 image with resolution 1366×768. The PPI in the horizontal axis would be $\frac{1366}{12} \approxeq 114$ PPI and the PPI in the vertical axis would be $\frac{768}{9} \approxeq 85$ PPI.
We can calculate the amount of storage a picture needs by first finding out the number of pixels a picture is made up of (the resolution) and then multiplying this by the number of bits used to represent each pixel (colour depth), so we end up with the formulas: Storage = resolution $\times$ Colour Depth
We also need to note that the picture may also contain meta data, meaning that the bit-map stores information about itself, such as width, height, colour depth, file type, and this will add to the size of the file. The meta data will normally be found at the beginning of a file within the header.

4.5.6.5 Vector Graphics are used to represent an image so that it can easily be rescaled via mathematical operations. Vector graphics are represented using a list of geometrical objects/ shapes such as lines, arcs, points in space, closed path shapes, and shape fill colour. For example, a square can be represented using 4 co-ordinates, and a line between each of the edges. Common properties are width, height, radius, co-ordinates, fill colour, stroke weight. A primitive data type is a data type provided by a programming language as a basic building block, within vector graphics, there are many primitives within vector graphics, these include:

- Rectangle
- Line
- Circle
- Text

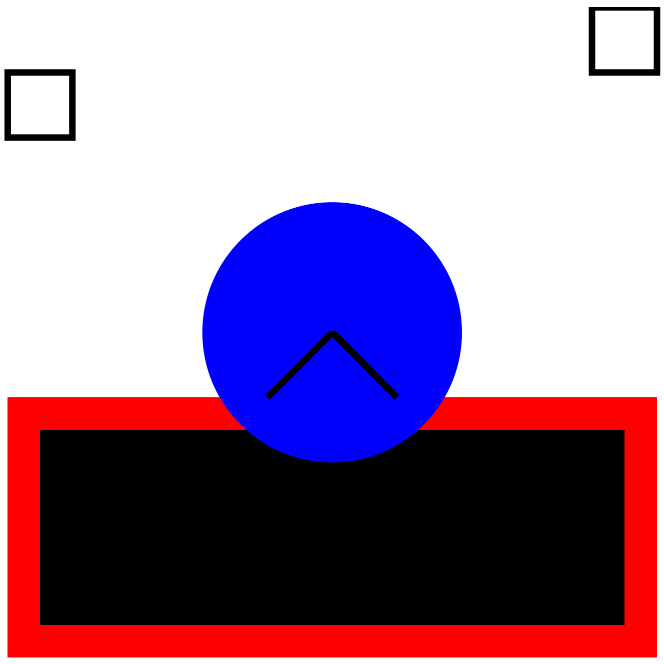
For the exam you should know how to use each of these primitives to draw a shape. To define objects, a layout similar to that of an xml/ css, where you have tags, and then define a bunch of properties, the properties that can be defined for each are as follows:

- Rectangle
  - width
  - height
  - x
  - y
  - fill
  - stroke (colour)
  - stroke-width
- Line
  - x1
  - y1
  - x2
  - y2
  - stroke-width
  - stroke (colour)
- Circles
  - cx
  - cy
  - r
  - fill
  - stroke (colour)
  - stroke-width
- Text
  - x
  - y

Note that not all of the properties need to be defined in order to define a shape, below I will show a simple example of how a vector graphic can be made and saved, using the .svg (scalable vector graphic) format:

```
<svg width="14" height="14">
<rect x="11" y="0" width="1" height="1"
stroke="black" stroke-width="0.1"
fill="white"/>
<rect x="2" y="1" width="1" height="1"
stroke="black" stroke-width="0.1"
```

```
fill="white"/>
<rect x="2" y="6" width="10" height="4"
fill="red"/>
<rect x="2.5" y="6.5" width="9" height="3"
fill="black"/>
<circle cx="7" cy="5" r="2"
fill="blue"/>
<line x1="7" y1="5" x2="6" y2="6" stroke="black" stroke-width="0.1"/>
<line x1="7" y1="5" x2="8" y2="6" stroke="black" stroke-width="0.1"/>
</svg>
```



4.5.6.6 Advantages of vector Graphics compared with bitmap graphics:

- Vector images are scalable without the loss of image quality

  This is due to the fact that within vector graphics, an image can be rescaled via their properties and then re-rendered to the screen, whereas for a bitmap image, you have to somehow guess what the details between pixels are when the resolution is increased.

- Vector images are in general smaller than bitmap images

  If you change the dimension of a vector graphic, you don't change the content of the Vector Graphic so its size doesn't change. Within a bitmap graphic changing the dimension of the file, may increase or decrease the amount of pictures within the file, and thus change the file size of the bitmap graphic

- Vector images are easy create, read and edit

  Vector graphics are often stored in a text format so they can easily be edited using a text editor, or using special software such as inkscape. Whereas bitmap graphics are often saved in binary files, and thus would be almost impossible to read for a human, and are thus edited using special software.

- Vector images has no resolution

  Vector images can be scaled to fit any size without a loss in quality, meaning they can be shown on screens of many different resolutions. Whereas with bitmap graphics, they have a set resolution, meaning that if they are shown on any screen except from their original resolution, either the image won't fit on the screen, or the image is stretched, losing quality

- Vector objects are reusable

  easy to change and modify a vector object from one image, and insert it into another vector image.

Disadvantages of vector graphics compared with bitmap graphics:

- There are many images that are impossible to represent using vector graphics.

  This includes images that include complex textures such as skin, present within digital photos. As well as images where the texture of the surface being drawn on is important, such as a painting, which can't be emulated by colour gradients within vectors, but can be emulated within bitmap graphics.

4.5.6.7 The sample rate of a sound represents the number of samples that will be taken per second, the higher the sample rate, the more accurate the sound will be compared to the original analogue sound. The sampling resolution is the number of bits allocated to representing the (amplitude of the) sound. When deciding the optimum sampling rate, many programmers refer to Nyquist's Theorem, which states that to faithfully recreate the analogue signal, you should sample at least twice the highest frequency for example, the human ear can hear a maximum frequency of 20,000Hz, so a sampling rate of at least 40,000Hz should be used. This is because using double the frequency allows the sample to cover the complete range of peaks and troughs. To calculate the space a sound sample will take up, we do the following calculation: Sample rate (Hz) $\times$ length of recording (seconds) $\times$ sampling resolution (bits)

4.5.6.8 Musical Instrument Digital Interface (MIDI) is a way to convert Analogue signals from instruments to digital signals. MIDI uses event messages to control various properties of the sound. These messages provide communication between MIDI devices or between a MIDI device and a processor. For Example, an event message may contain data on:

- When to play a note
- When to stop playing a note
- Timing a note to play with other notes or sounds
- Timing a note to play with other MIDI-enabled devices
- What pitch a note is
- How loud to play it
- What effect to use when playing it

The advantage of using midi files over other digital audio formats are:

- MIDI files tend to be smaller
- MIDI files are completely editable as individual elements and sounds can be selected and edited
- MIDI supports a wide range of instruments, providing more choices for music production
- MIDI files can produce very high quality authentic reproduction of the instrument

4.5.6.9 Often we find that files can be extremely large, taking up megabytes or gigabytes of space. Compression is the process of reducing the number of bits required to represent data, this is useful for making files smaller, which in turn makes the files easier to transmit.
Lossless compression is compression where all the data is kept (no data is lost), the compressed is as accurate as the uncompressed file. Lossy compression is compression that causes some degradation in the quality of the file, for example, a grainier photo.
Two of the methods used for lossless compression are as follows:

- Run Length Encoding (RLE)

  This method compresses data by eliminating repeated data, for example, NNNNNN $\rightarrow$ 6N.

- Dictionary-Based Methods

  This method is used to compress data files by using a dictionary to associate certain groupings of characters with a certain token, as shown below, so to create the string "creation" we would simply use 41.

Data	Token
tion	1
sta	2
na	3
crea	4

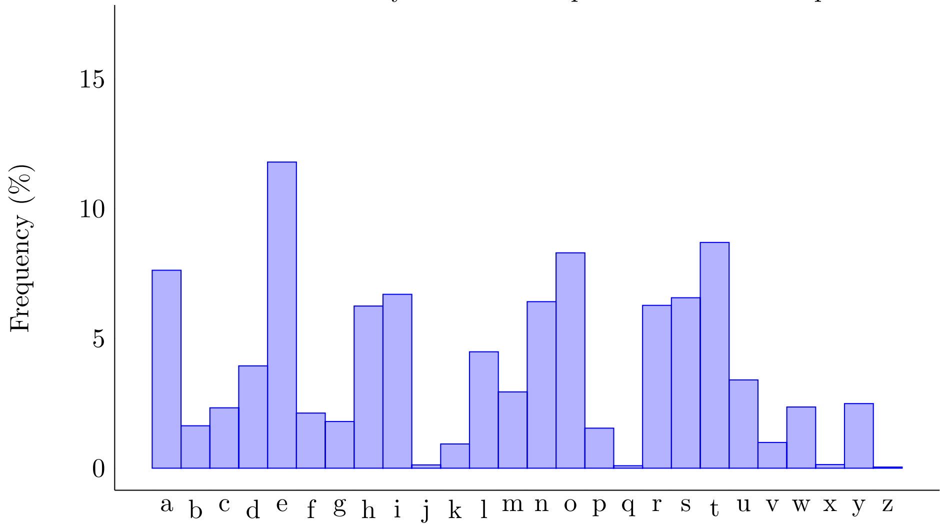
4.5.6.10 Encryption is the process of turning plaintext (A normal piece of text) into scrambled ciphertext, which can only be understood if decrypted. A key in encryption is a piece of data used that defines the way in which plaintext is converted into ciphertext and therefore implies how to convert ciphertext to plaintext. There are many ciphers that can be used when to encrypt data, Two examples are:

- Caesar Cipher

  The caesar cipher is a substitution cipher where one character of plaintext is substituted for another, which becomes the cipher text. This works by shifting all of the letters in the cipher text backwards or forwards a certain amount along the Alphabet, the amount shifted is dictated by the key. For example, if you were to use a key of 2 A→C, B→D, C→E, ...etc.

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

  The caesar cipher is one of the easiest due to the fact that you can do simple frequency analysis of the cipher text as all you would see is a shift in the frequency of certain letters, and can therefore see what the key was for the cipher and undo the cipher.



- Vernam Cipher/ One-time Pad

  The Vernham cipher is a method of encryption that uses a one-time pad (key) to create ciphertext. the one time pad is a randomly generated string with the same length as the cipher text. To encrypt a message, each character in the plaintext is combined with the character at the corresponding position in the key by converting them into a binary code and performing a bitwise xor on the binary representations to obtain new binary codes, which can in turn be mapped to new characters. So For example

Plaintext Message	H	E	L	L	O
Key	I	S	W	I	B
Ciphertext	L		E	H	E

The Vernham is said to be mathematically impossible to crack, this is due to the fact that a random key is used, meaning that all characters have an equal chance of appearing, so no form of analysis can be done to figure out what the key is, and thus is impossible to get the original message. This is different to other cyphers which may be computationally secure, meaning that they can theoretically be solved with enough time and cipher text, but not in any time that would be considered useful using current technology.

6 Fundamentals of Computer Systems

6.1 Hardware and Software

3.6.1.1 Hardware is a generic term for the physical parts of the computer, both internal and external. Software is a generic term for any program that can be run on a computer. Thus we get the relationship that software is run on hardware.

3.6.1.2 There are several different types of software:

- Application Software

  This is software that perform specific tasks for the user. Some examples are word processors, spreadsheets, and web browsers.

3.6.1.3
- System Software

  This is software that is needed for the computer to run, they support the application software. There are several forms of application software:

  - Utility Programs

    This is software that is written to carry out certain housekeeping tasks on hardware, they usually come with the OS and help to enhance your computer, however your computer can work without them, examples are compression software, anti-virus software, back-up software, registry cleaners.
  - Library Programs

    this is a large amount of different pieces of software that are in general important for applications to work, as they usually call data from within this software. An example is Windows Dynamically linked libraries, which are called upon whenever windows needs it, they contain code, data, and resources.
  - Translators (Compilers, Assemblers, and Interpreters)

    A translator is software that converts high level programming languages into machine code (1s and 0s) so that it can carried out by the machine.
  - Operating Systems (OS)

    This is a suite of programs designed to control the operations of the computer, it acts as the interface between the user and computer.

3.6.1.4 The OS links together the hardware, applications and the user, but hides the true complexity of the computer from the user and other software by using a virtual machine. Some examples of the tasks that the OS performs are as follows:

- Configure the start configuration of your computer
- recognising inputs from I/O devices and deciding what to do
- Attempts to cope with errors as they occur
- controls the shutting down of the computer
- controls print queues
- manages users on a network
- Resource Management

  This is the way in which an OS manages hardware and software to optimise the performance of the computer. If a computer features one processor, then only one process (and thus one application) can actually be alive at any time, so to simulate having multiple applications being run at the same time the computer has to schedule how different programs are able to use the processor. A common type of scheduling is Round-Robin scheduling, where each task has an equal amount of time using the processor. This can be inefficient as some processes may need little or none of the time allocated to them, so a more advanced scheduling system would allocate this time to another task before the time slice ran out.

- Managing I/O devices

  The operating controls the way in which I/O devices are allocated, controlled and used by programs that are using them. Due to the fact that I/O devices work relatively slow when compared to the processor, a queue of commands for a certain device, which the device can carry out in order when it is ready. Every I/O device has a driver which allows the OS to interact with that device.
- Memory Management

  This deals with how RAM is allocated to different files and applications. The OS stores data of all unallocated memory addresses in a section of memory called the heap, when a file is opened, it is assigned memory from the heap, and when it is closed, the memory locations are put back into the heap. It is te task of the memory management routines to manage the assignment of memory and to load the necessary files to their respective memory address. The OS uses a memory so that it knows what places in the memory are taken up so the OS can control more than one task in the ram at any one time. The amount of RAM a program depends it's size and complexity.
- Virtual Memory and paging

  When a file or application is too big to fit in RAM, part of the secondary memory may be allocated to store memory that is usually held in RAM and treats this part of the secondary memory as if it were RAM, this is called virtual memory. An alternative method is to store the central block of the code in the RAM and then call upon other sections of code (called pages) from secondary memory as they are needed, this allows large applications to run in a small part of the RAM.
- File Management

  This concerns how the OS stores and retrieves files from secondary memory. The OS uses a hierarchical structure, which allows you to have files of the same name as long as they are in different folders. Larger hard disks have lended themselves to the idea of partitions, which means that one hard drive can act as two logical drives so, to your computer, it would seem like you have two hard drives.

6.2 Classification of Programming Languages

3.6.2.1 There are many different programming languages, some closely reflect the architecture of the computer (low-level languages) while others more reflect the way we naturally speak (high-level languages).
There are two main low level languages:

- Machine Code

  This is the lowest level of code which is made up of 0s and 1s, this is the only format that the processor can process. Due to the fact that it is simply a sequence of 1s and 0s, it can make it very unwieldy to read and write, and also means there's a large chance of mistakes. One way to make it easier is to Hexadecimal or decimal numbers to show a sequence of bits. If the code does have an error, it can be very hard track down, also due to the fact that you are writing directly to one specific processor, it is unlikely to be portable, however this also means that it is likely to run quickly on that specific processor and do exactly what you said.

- Assembly Language

  Assembly language is more programmer as it allows programs to use mnemonics which translate to one command in machine code, there is a one-to-one relationship. The number of mnemonics used in an assembly language is usually quite small. Assembly language can be quite difficult to read but has many benefits: programs are executed quickly due to the fact that a compiler cannot optimise code as well as a programmer, program code is relatively compact for the same reason, direct manipulation of registers leads to a high level of control. For theses reasons it is used for projects where quick responses are needed or low level interactions, such as embedded systems, real time apps, and device drivers.

A High level programming language is a programming language that allows programs to be written using English keywords and that is platform independent. These are problem-oriented, meaning that they are created to overcome a certain problem, not made for a specific computer architecture, so they are portable. Imperative languages (Also known as procedural languages) are languages that work by typing in lists of instructions (subroutines/procedures) that the computer has to follow. Every time the program is run, the same set of instructions are run. High level languages are related to low level languages because high level languages must be translated to low level languages (specifically machine code) in order to be understood by the computer. Some of the main features are:

- Easy to identify what a town does
- They need to be translated
- One high level command translates to many machine code commands, there is a one-to-many relationship
- They are portable
- They are easy to maintain via use of a wide range of program structures.

6.3 Types of Program Translator

3.6.3.1 There are three types of translators:

- Assembler

  An assembler is a program that translates a program written in assembly language into machine code.

- Compiler

  A compiler is a program that translates high level language into machine code by translating all of the code.

- Interpreter

  An interpreter is a program that translates high level language by reading each statement in the source code and immediately performing the action.

Interpreter		Compiler	
Advantage	Disadvantage	Advantage	Disadvantage
You don't need to compile the whole program in order to run a section of the code	Sections of the code that are revisited in a program will need translating each time, meaning longer execution time	Once compiled, source code no longer needed	Whole program has to be translated when slight alterations are made, meaning debugging takes a long time
Program code can be run on processors with different instruction sets.	Source Code can only be translated thus executed on a computer with the same interpreter.	Difficult to reverse engineer source code from object code.	The object code will only run on a computer that has the same platform.
Mostly like to be used during development.	The source code must be distributed.		

Some programming languages use bytecode, which is an instruction set that can be executed using a virtual machine. This virtual machine can emulate the instruction set of the architecture of the computer, meaning that the source code written using bytecode can be executed on any platform.

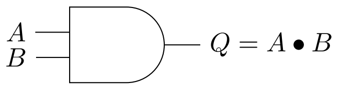
6.4 Logic Gates

3.6.4.1 These are the logic gates that you need to know:

Logic Gate Name | **Truth Table** | **Logic Gate Symbol**

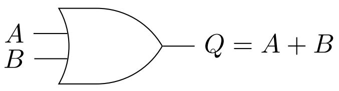
AND

A	B	$A \bullet B$
0	0	0
0	1	0
1	0	0
1	1	1



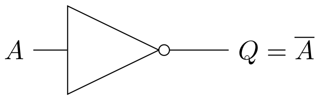
OR

A	B	$A + B$
0	0	0
0	1	1
1	0	1
1	1	1



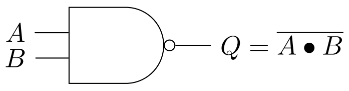
NOT

A	$\overline{A}$
0	1
1	0



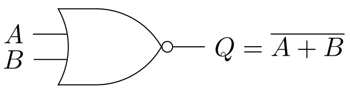
NAND

A	B	$\overline{A \bullet B}$
0	0	1
0	1	1
1	0	1
1	1	0



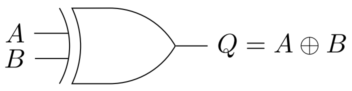
NOR

A	B	$\overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

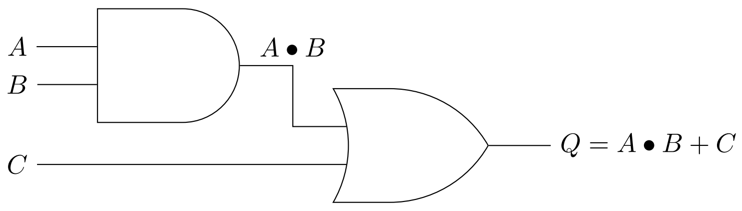


XOR

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0



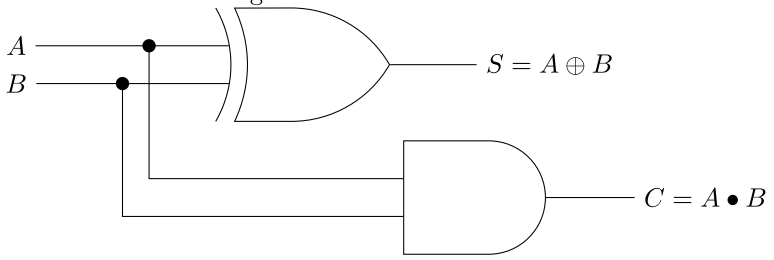
Logic Gates can be combined in order to create more complex boolean expressions, for example $Q = A \bullet B + C$ can be expressed using the following logic gate:



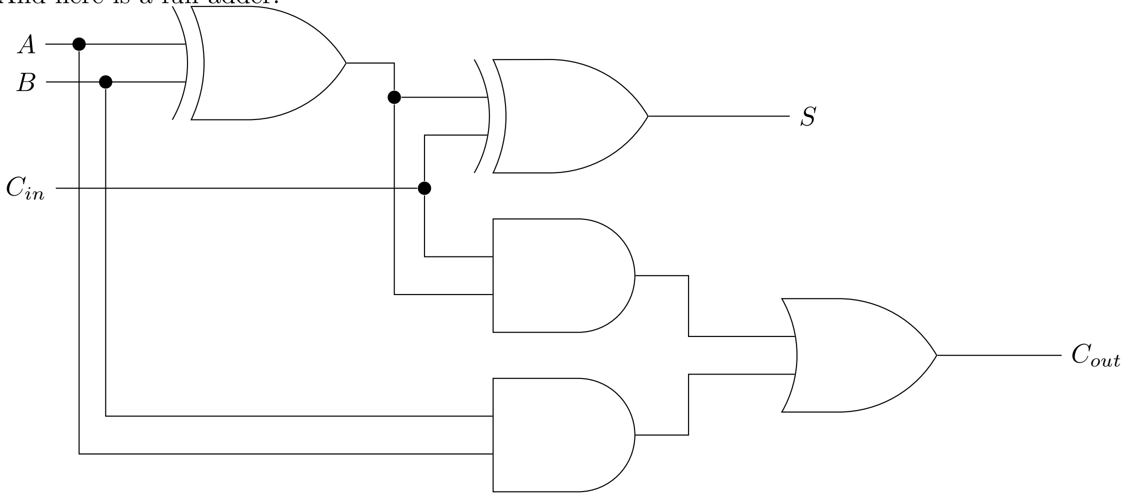
We can also construct logic tables from logic diagrams, we can show how this may be done using the logic diagram above as an example:

A	B	C	$A \bullet B$	$A \bullet B + C$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

Here is the circuit diagram for a half adder:



And here is a full adder:



Where $S = (A \oplus B) \oplus C$ and $C_{out} = (A \bullet B) + ((A \oplus B) \bullet C)$. The purpose of the logic diagrams shown above are to add to binary numbers together, where a half adder can return the result of adding two bits together, and a full adder can add sequences of bits together, by feeding the carry bit C_{out} back into itself as C_{in} and then having the relevant bits inputted to A and B, their respective logic gates look like as follows:
Half Adder

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

Full Adder

A	B	C_{in}	C_{out}	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

An edge triggered D-type flip-flop is a small memory unit that allows one bit of data to be stored, and is often used within the processor to store the results of previous calculations so that they can be used later. Due to the way in which they are designed, the state of the output can only be updated whenever clock signal changes (from low to high or high to low, depending on how it is designed). An input signal is needed to change the output of an edge-triggered d-type flip-flop, as well as the change in clock state.

6.5 Boolean Algebra

4.6.5.1 There are several boolean identities that can be used to simplify a boolean expression, these are as follows:

Identity Name	**AND Form**	**OR Form**
Identity	$A \bullet 1 = A$	$A + 0 = A$
Null (or Dominance) Law	$A \bullet 0 = 0$	$A + 1 = 1$
Idempotence Law	$A \bullet A = A$	$A + A = A$
Inverse Law	$A \bullet \overline{A} = 0$	$A + \overline{A} = 1$
Commutative Law	$A \bullet B = B \bullet A$	$A + B = B + A$
Associative Law	$(A \bullet B) \bullet C = A \bullet (B \bullet C)$	$(A + B) + C = A + (B + C)$
Distributive Law	$A + B \bullet C = (A + B) \bullet (A + C)$	$A \bullet (B + C) = A \bullet B \ + A \bullet C$
Absorption Law	$A \bullet (A + B) = A$	$A + A \bullet B = A$
De Morgan's Law	$\overline{A \bullet B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \bullet \overline{B}$
Double Complement Law	$\overline{\overline{A}} = A$	

Here's an example of how you could simplify a boolean expression:

$$
\begin{aligned}
(A \oplus B) \oplus B &= (A \bullet \overline{B} + \overline{A} \bullet B) \oplus B \\
&= (A \bullet \overline{B} + \overline{A} \bullet B) \bullet \overline{B} + \overline{(A \bullet \overline{B} + \overline{A} \bullet B)} \bullet B \\
&= A \bullet \overline{B} \bullet \overline{B} + \overline{A} \bullet B \bullet \overline{B} + \overline{(A \bullet \overline{B} + \overline{A} \bullet B)} \bullet B \\
&= A \bullet \overline{B} + \overline{(A \bullet \overline{B} + \overline{A} \bullet B)} \bullet B \\
&= A \bullet \overline{B} + \overline{A \bullet \overline{B}} \bullet \overline{\overline{A} \bullet B} \bullet B \\
&= A \bullet \overline{B} + (\overline{A} + B) \bullet (A + \overline{B}) \bullet B \\
&= A \bullet \overline{B} + (\overline{A} \bullet A + \overline{A} \bullet \overline{B} + A \bullet B + B \bullet \overline{B}) \bullet B \\
&= A \bullet \overline{B} + (\overline{A} \bullet \overline{B} + A \bullet B + B) \bullet B \\
&= A \bullet \overline{B} + \overline{A} \bullet \overline{B} \bullet B + A \bullet B \bullet B \\
&= A \bullet \overline{B} + A \bullet B \\
&= A \bullet (\overline{B} + B) \\
\therefore (A \oplus B) \oplus B &= A
\end{aligned}
$$

7 Fundamentals of Computer Organisation and Architecture

7.1 Internal Hardware Components of a computer

4.7.1.1 There are several component that lie within a computer, the main internal components are:

- Processor

  A device that carries out computation on data by following instructions, in order to produce an output.

- Main Memory

  This stores data and instructions that will be used by processor.

- Address Bus

  A mono-directional (from processor to memory) bus that is used to specify a physical address in memory so that the data bus can access it.

- Data Bus

  This is a bi-directional bus that transfers data between the processor and memory.
- Control Bus

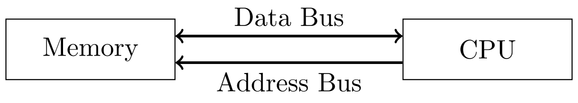
  This is a bi-directional bus that sends control signals to the registers, the data, and the address buses.
- I/O Controllers

  These control the flow of information between the processor and the input and output devices.

There are two main architectures you need to know about:

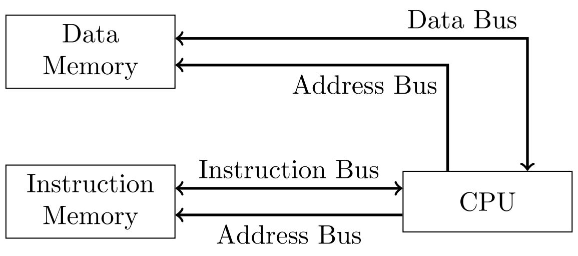
- Von Neumann Architecture

  A technique for building a processor where data and instructions are stored in the same memory and accessed via buses This is used for common computing devices, e.g. a computer.



- Harvard Architecture

  A technique for building a processor that uses separate buses and memory for data and instructions. These are mainly used in embedded computer systems (e.g. mobile phones, burglar alarms, etc.) as data and instructions don't have to run down the same bus, meaning faster executions of programs.



Addressable memory is the concept that memory is made up of millions addressable cells with each cell being given a uniquely identifiable address, allowing programs and files to be stored across a number of these cells. The systematic way in which memory is organised allows different programs to be stored in different parts of the memory. A memory map could be made to show what type of programs are stored in what part of the memory.

7.2 The Stored Program Concept

4.7.2.1 This is the concept that instructions and data are stored together in memory, and the instructions are fetched and executed serially by a processor that performs arithmetic and logical operations.

7.3 Structure and Role of the Processor and its Components

4.7.3.1 There are many parts within the processor, the major components are:

- Arithmetic Logic Unit (ALU)

  This is the part of the processor that processes and manipulates data. It can do two different types of operations, arithmetic and logical.
- Control Unit

  This is the part of the processor that manages the execution of instructions in the fetch-execute cycle.
- Clock

  This is a device that generated a signal used to synchronise the components of a computer. clock speed is measured in megahertz (MHz) or gigahertz(GHz) and shows how many pulses are sent per second.

- General-Purpose registers

  A small section of temporary storage that is part of the processor. Stores data or control instructions during fetch-decode-execute cycle.

- Dedicated Registers, including:
  - Program Counter (PC)

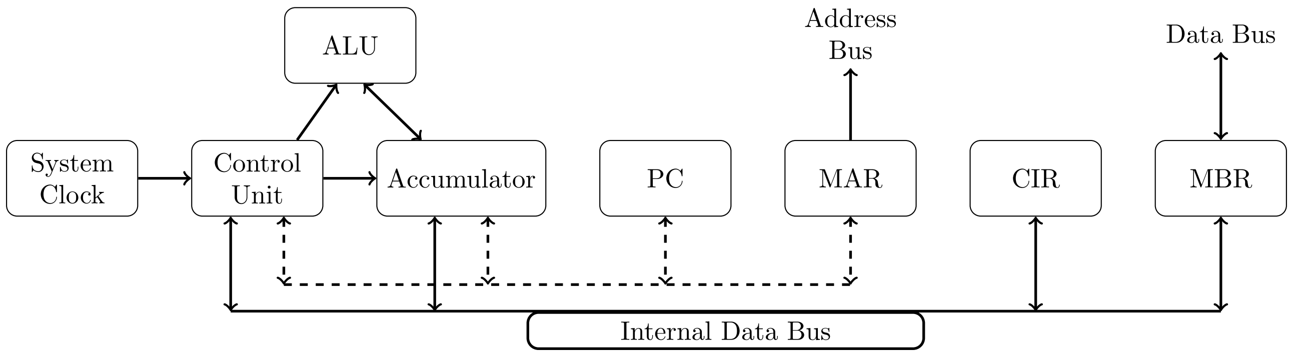
    This is a register that stores the address of the next instruction to be taken from main memory into the processor.
  - Current Instruction Register (CIR)

    This is a register that stores the instructions that the CPU is currently decoding/ executing.
  - Memory Address Register (MAR)

    This register stores the location of the address that data is either written to or copied from by the processor.
  - Memory Buffer Register (MBR)

    This register holds data that is either written to or copied from the CPU.
  - Status Register

    This keeps track of the various functions of the computer such as if the result of the last calculation was positive or negative.



4.7.3.2 There are three steps for the Fetch-execute cycle

1. Fetch

   The program counter holds the address for the next instruction. The proceeor sends this address along the address bus to the main memory. The contents of the memory location at that address are sent via the data bus to the current instruction register and the program counter is incremented. The details of the addresses are initially loaded into the memory address register and the data initially goes to the memory buffer register. Some instructions need to load a number of bytes or words, so they may need to be fetched as successive parts of a single instruction.

2. Decode

   The processor then takes the instruction from the CIR and decides what to do with it. It does this by referring to the instruction set. These instruction sets are either classed as an RISC (Reduced Instruction Set) or CISC (Complex Instruction Set). An instruction set is a library of all the things the processor can be asked to do. Each instruction in the instruction set is accompanied by details of what the processor should do when it receives that particular instruction. This might be to send the contents of the memory buffer register to the arithmetic logic unit.

3. Execute

   Once the instruction that has just been taken from the memory has been decoded, the processor now carries out the instruction. It then goes back to the top of the cycle and fetches the next instruction. A simple instruction will require only a single clock cycle, whereas a complex instruction may need three or four. The results of any calculations are written either to a register or a memory location.

4.7.3.3 An instruction set is a set of instructions that a processor can perform written in machine code, each processor has its own unique instruction set. A simple instruction can be split into three different parts:

- Opcode

  an operation code or instructions used in assembly language.

- Operand

  a value or memory address that forms part of an assembly language instructions. The number of operands a needed depends on the opcode being used.

- Addressing Mode

  The way in which the operand is interpreted (a memory address or an actual value)

Here's an example of a possible instruction using a 32 bit scheme (the precise format is up to the designer of the processor, the following uses 16 bits for the op code, 4 bits for the addressing code, and 12 bits for the operand): $\underbrace{1101010011101000}_{\text{Opcode}} \quad \underbrace{0001}_{\text{Adressing Mode}} \quad \underbrace{010011110011}_{\text{Operand}}$

4.7.3.4 There are two main types of addressing modes:

- Direct addressing

  This means that the operand refers to the address of the datum, meaning that the data is held in RAM or in a register, and the operand says where to find the data.

- Immediate addressing

  This means the operand is the datum, so for example #10 would mean immediate address and that the value to use is the number 10.

4.7.3.5 There are a large array of opcodes, here are the ones you will need to know:

- Load

  This loads the value stored in the memory location specified by `<memory ref>` into register `d`: `LDR Rd, <memory ref>`

- Add

  Add the value specified in `<operand2>` to the value in register `n` and store the result in register `d`: `ADD Rd, Rn, <Operand2>`

- Subtract

  Subtract the value specified in `<operand2>` to the value in register `n` and store the result in register `d`: `SUB Rd, Rn, <Operand2>`

- Store

  Store the value that is in register `d` into the memory location specified by `<memory ref>`: `STR Rd, <memory ref>`

- Branching

  - Conditional

    Conditionally branch to the instruction at position `<label>` in the program if the last comparison met the criteria specified by the `<condition>`. Possible values for `<condition>` and their meanings are: `EQ`-Equal to, `NE`-Not equal to, `GT`-Greater than, `LT`-Later than: `B<condition> <label>`

  - Unconditional

    Always branch to the instruction at position `<label>` in the program: `B <label>`

- Compare

  Compare the value stored in register `n` with the value specified by `<operand2>`: `CMP Rn, <operand2>`

- Logical Bitwise Operators

  - AND

    Perform a bitwise logical AND operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`: `AND Rd, Rn, <operand2>`

  - OR

    Perform a bitwise logical OR operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`: `ORR Rd, Rn, <operand2>`
  - NOT

    Perform a bitwise logical NOT on the value specified by `<operand2>` and store the result in register `d`: `MVN Rd, <operand2>`
  - XOR

    Perform a bitwise logical XOR operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`: `EOR Rd, Rn, <operand2>`

- Logical
  - Shift Left

    Logically shift left the value stored in register `n` by the number of bits specified by `<operand2>` and store the result in register `d`: `LSL Rd, Rn, <operand2>`
  - Shift Right

    Logically shift right the value stored in register `n` by the number of bits specified by `<operand2>` and store the result in register `d`: `LSR Rd, Rn, <operand2>`

- Halt

  Stops the execution of the program.

4.7.3.6 Interrupts are signals sent by an external source or program to the processor, usually to request the processor to start doing something. Some examples are:

- Printer requesting more data be sent
- Key press
- Run time error
- hardware fault
- requesting termination of a program
- power down
- switch user

To incorporate interrupts into the fetch execute cycle, we just add checking for interrupts on the end of the cycle:

1. Fetch
2. Decode
3. Execute
4. Check for Interrupts

If an interrupt is found, the interrupt service routine (ISR) calls the routine required to service the interrupt. As most interrupts are temporary, the processor needs to be able to store the data from the current task somewhere while it fulfils the interrupts. It does this by storing the PC and CIR in the system stack, then recovering them when the interrupt has been serviced, restoring them to their appropriate registers.
To deal with the fact that an interrupt may interrupt an interrupt, all interrupts are given a priority, meaning the most important interrupts are serviced first, then less important interrupts are serviced, the order of priority from most to least important are as follows

- Hardware Failure

  E.g. Power failure

- Reset interrupt

  E.g. Resetting a computer to factory settings

- Program Error

  E.g. Arithmetic overflow
- Timer

  Real time clock, allows actions to be performed at regular intervals
- Input/ Output

  E.g. Keyboard key pressed

4.7.3.7 There are several factors that can impact the performance of a processor:

- Multiple Cores

  Multiple cores means that multiple instructions can be carried out at once, so more processsors can lead to an increase in processor performance.
- Cache Memory

  This involves a technique where instructions and data that are frequently used are stored in cache memory (a high-speed temporary area of memory), so programs run faster.
- Clock Speed

  This indicates how many pulses the clock sends out per second and thus indicates the maximum number of instructions that can be performed per second, so increasing clock speed may increase processor performance.
- Word Length

  This indicates the number of bits the can be addressed, transferred or manipulated as one unit. Increasing the word length means that more data can be processed at each pulse of the unit clock.
- Address Bus Width

  This indicates the number of bits that can be sent down the address bus in one go (during one clock pulse), increasing the address bus width means more memory can be addressed, so more memory can be installed on the computer.
- Data Bus Width

  This indicates the number of bits that can be sent down the data bus in one go (during one clock pulse), increasing the data bus width means more data can be sent down the bus during each clock pulse, so more data can be processed within a given time.

7.4 External Hardware Devices

4.7.4.1 We need to know the purpose and principles of operation for 4 I/O devices:

- Barcode Reader

  Barcodes are used primarily for inputting product details at the point of sale. This can be done by passing the object over a scanner or by using a handheld scanner to scan the barcode. They work in the following way:

  1. A light (an LED or laser) is passed over an image
  2. Some form of light sensor is used to measure the intensity of light that is reflected which is converted into a current, effectively generating a waveform by using a photodiode or charge coupled device (CCD).
  3. white areas reflect the most light whereas black reflects the least, so the waveform can be used to identify the pattern of black and white bars.
  4. The analogue waveform needs to be converted into digital form using and ADC, encoding the white and black into binary (for example, black=0, white=1).
  5. The signal is decoded into a form that can be interpreted by software.
- Digital Camera

  Digital Camera's are useful for creating digital images of photographs, which can then be printed ir transferred onto a computer to be manipulated and stored. The quality of a camera is dictated by how many pixels it uses to record the image, e.g. a 12 megapixel camera stores data on 12,000,000 pixels. They work in the following way:

1. When a photograph is taken the shutter opens and lets light in through the lens.
2. The light is focused onto a sensor (charge coupled device (CCD) or a complementary metal oxide semiconductor (CMOS)).
3. The sensors are made up of millions of transistors, each of which stores the data for one or more pixels.
4. As the light hits the sensors, it is converted into electrons and the amount of charge is recorded for each pixel in digital form.
5. With light, all colours can be created from red, green and blue (RGB). Therefore to record colour, the camera will either have three different sensors, or use three different sensors, one for each colour.
6. The data are typically stored on removable storage devices, usually referred to as flash memory, which uses programmable ROM and are usually stored in compressed files, e.g. TIFF, JPEG or PNG. (Raw files can also be generated, which are uncompressed and therefore contain all of the data from the original paragraph)
7. This digital data can now be decoded and manipulated using specialised software.

- Laser Printer

  This is a device that uses lasers and toner to create mono and colour prints, its major advantage over standard home printers is that it is much faster, and is built to print many different documents. it works in the following way:

  1. A rotating drum inside the printer is coated in a chemical which holds an electrical charge.
  2. The laser beam is reflected by a mirror onto the drum and where the light hits the drum the charge is discharged, effectively creating the image on the drum.
  3. As the drum rotates it picks up toner which is attracted to the charged part of the drum.

     If printing in colour, then four colours will need to be used: cyan, yellow, magenta, and key (black) (CMYK), so either the paper has to go around the drum 4 times, or a transfer belt may be used to hold all of the colours.
  4. Paper is passed over the drum and by charging the paper with the opposite charge to the toner, the toner is attracted to the paper and away from the drum. The paper is heat treated to fuse the toner onto the paper.

- Radio Frequency Identification (RFID)

  This is a microscopic device that stores data and transmits it using radio waves - usually used in tags to track items. It works in the following way:

  – The tag contains a chip, which contains data about the item and a modem to modulate and demodulate the radio signals, it also contains an antennae to send and receive signals.
  – Tags can be either active, meaning they have their own power supply or passive, meaning that they will pick up electromagnetic power when they are in range of a RFID reader.
  – Signals, and therefore data, can be transmitted in both directions using radio frequencies. This may be over a short or long distance depending on how the tag is built. The typical range is between 1 and 100m.
  – Tags may be used to simply track the physical location of an item or the data may transmit data back.

  RFID is a relatively new tech and it is being put to various uses, some of which are controversial. Applications include:

  – Tracking individuals, particularly vulnerable adults such as Alzheimer's patients.
  – Use in electronic passports to keep track of where people travel.
  – Tags have been added to credit and debit cards allow users to make contactless payment via RFID in a shop.
  – Transport and distribution companies can use RFID to track shipments and deliveries

  - Tags have been added to high value items, for example artworks in museums or equipment in hospitals.

4.7.4.2 Secondary storage devices are used within computers to allow data to be permanently stored within it. There are three main secondary storage devices:

- Hard Disk (HDD)

  Hard disks are constructed of hard metallic material and are hermetically sealed (airtight). This is to protect them from being corrupted by dust or other debris. Most hard disks are in fact made up of a number of disks arranged in a stack. The disks are coated with a thin film of magnetic material. Changes in direction of magnetism represents zeros and ones.

  - hard disks spin at speeds between 3600 and 12500 RPM, as a series of heads read from and write to the disks.
  - The heads do not actually touch the surface of the disk but float slightly above it by virtue of the speed at which the disk spins.
  - There is an actuator arm which moves the head across the surface of the disk as it spins.
  - The combination of the rotating of the disks with lateral movement of the arm means that the head can access every part of the disk surface.
  - The surface of the disk is organised into concentric tracks and each track is split into sectors each of which can be addressed by the OS.
  - Because all of the actuator arms move at the same time due to the assembly head, the cylinder is often referenced, which simply just says what track the data is on for all of the disks.
  - Each sector has the same capacity and a large file will be stored over a number of sectors.
  - The OS groups sectors together into clusters to make storage easier to manage, there will be many occasions when a whole cluster is not needed, leading to redundant space.

    For example, a file may need four sectors and part of the fifth, so it will be allocated all 5 sectors, meaning there's empty (redundant) space that can't be used.

- Optical Disk

  Optical disk is a generic term for all variations of CD, DVD and Blu-Ray that use laser technology to read and write data. An optical disk is made up of one single spiral track that starts in the middle and work it's way to the edge of the CD. The laser will read data that are contained within this track by reading the pits (lower parts) and lands (higher parts) in combination with a sensor that measures how much light is reflected.

  - For read-only optical disks, when data is written it is encoded as a series of pits and lands within the track of the disk.
  - A protective layer is then put over the surface to prevent any data corruption, the pattern of pits and lands are used to represent data.
  - When the CD is read, the pits and lands are read by the laser which then interprets each as different electrical signals, then the electrical signals can be converted into binary codes.

  - For writeable optical disks, rather than using pits and lands, the disk is coated photosensitive dye, which is translucent, when writing to the disk, the laser will alter the state of a dye spot that is coated onto the surface making it opaque.
  - The dye reflects a certain amount of light.
  - A write laser alters the density of the dye and a read laser interprets the different densities to create binary patterns which in turn can represent data, write lasers are higher powered than read lasers.

- Solid-State Disk (SSD)

  Solid-state disks use programmable ROM chips, similar to memory cards, hence they are more often referred to as Solid-state drive instead of solid-state disk.

  - These are stored inside a unit that looks like a hard disk and commonly uses a type of memory called NAND memory.
  - This organises data into blocks in a similar way to traditional hard disk as described earlier, with a controller being used to manage the blocks of data.
  - Blocks of a set physical size will be made up of binary data.
  - When reading and writing, data can only be accessed in blocks.
  - With SSD, blocks will be allocated to particular semiconductors, meaning data can be added and deleted to different blocks to different areas of the drive, sso that only small parts of the drive have to be erased and written to, enabling very fast access times.
  - It uses a floating gate transistor to be able to trap and store charge, it contains two gates: floating gate and control gate, a thin layer of oxide is placed between the two gates, trapping the charge inside the floating gate even when power is turned off.

Here is a table comparing the different devices:

	Hard Disk (HDD)	**Solid State Disk (SDD)**	**Optical Disk (CD/DVD)**	**Optical Disk (Blu-ray)**
Typical Capacity	High (1 TB)	Medium (500 GB)	Low (900 MB to 1.7 GB)	Low to Medium (25-50 GB)
Relative Cost	Medium	High	Low	Low
Easily portable	External disks are available	External disks are available	Yes	Yes
Relative Power Consumption	High	Low	High	High
Relative Speed of access	Medium	High	Low	Low
Latency	High	Low	Very High	High
Fragmentation		None		
Reliability	Good	Very Good	Fair	Fair
Relative Physical size	Large	Small	Large	Large

8 Consequences of Uses of Computing

8.1 Individual (Moral), Social (Ethical), Legal and Cultural Issues and Opportunities

3.8.1 Here, we define a moral issue as an issue that concerns our personal concepts about what is right and what is wrong, whereas ethical issues are about what society finds right and wrong. Development in technology has led to many people worrying about use and misuse of data:

- Personal privacy
- Data security
- Misuse of Data
- Governmental Monitoring
- Online Profile
- Profiling

There are also many other issues associated with computer science:

- Unauthorised access
- Unauthorised use of software

- Inappropriate behaviour
- Inappropriate Content
- Freedom of Speech
- Unemployment
- Access to the internet

When using a computer as part of an institution, you will normally have to agree to a code of conduct. The British Computer Society (BSC) has produced a code of conduct that guide organisations on the ethical use of computer systems in general. Any breaches in the code can't be punished legally but could lead to dismissal of an employee or asking a student to leave their school. The general principle are:

- Always operate in the interest of the public
- Have a duty to the institution you represent
- Have a duty to the profession
- Maintain professional competence and integrity

For this course you may be ask questions on how laws manage the use of computers, the specification is super vague so they can basically ask you about anything, so here's a bunch of laws you may need to know:

- Data Protection Act

  This law states that (with a few exceptions) any person or organisation storing personal data must register with the Information Comissioner, and that data subjects have the right to know what data are stored about them by any particular individual or organisation.

- Freedom of Information Act

  This gives general rights of access to information held by public authorities such as hospitals, doctors, dentists, the police, schools and colleges.

- Computer Misuse Act

  This was introduced to prevent hacking (data misuse) and deals with three specific offences: unauthorised access to computer programs or data, unauthorised access with further criminal intent, unauthorised modification of computer material.

- Regulation of Investigatory Powers (RIP) Act

  This talks about the powers the government have when investigating and have two parts that relate to computing (out of 5 parts), part one which relates to interception of communication, and part three which relates to investigation of encrypted data.

- Copyright, Designs and Patents Act

  This gives certain rights to the creators of certain kinds of material allowing them control over the way in which the material is used, covering the copying, adapting and renting of materials.

  Copyright can be dealt with in a few ways, two techniques are: Digital Rights Management (DRM) which uses access control software to limit what the user can do, and Licensing where the user has to use some sort of paper or digital proof to be able to use a piece of software.

- Official Secrets Act

  Prevents the disclosure of government data relating to national security

- Defamation Act

  Prevents people from making untrue statements about others which will lead to their reputation being damaged

- Obscene Publications Act and the Protection of Children Act

  Prevent people from disseminating pornographic or violent images

- Health and Safety Regulations

  Provides regulations on the correct use of screens and on how to keep employees safe in general

- Equality Act

  Illegal to discriminate against anyone because of their age, sex, sexual orientation, ethnicity, religion, disability, or age.

Here are some cultural issues that could possibly be talked about:

- Over-use of data: Too dependant on data
- Invasive technologies: Data collected without consent
- Over-reliance on computers: Computer systems failing may lead to loss of lives
- Over-reliance on technology companies: Some companies have large internet presence
- Governmental monitoring: Beliefs that the government may track everything we do
- Globalisation: We can be influenced by other countries and cultures

9 Fundamentals of Communication and Networking

9.1 Communication

4.9.1.1 There are several forms of transmission:

- Serial Transmission

  Data is transmitted one bit at a time down a single wire.
- Parallel Transmission

  Data is transmitted several bits at a time using multiple wires.
- Synchronous Transmission

  Data is transmitted where the pulse of the clock of the sending and receiving device are in time with each other. The devices may share a common clock.
- Asynchronous Transmission

  Data is transmitted between two devices that do not share a common signal.

Serial transmission has 3 main advantages over parallel transmission:

- It requires less wires so it is cheaper
- It degrades less over distance compared to parallel transmission (due to the fact that the multiple wires cause interference between them)
- Serial transmission doesn't need to be synchronised whereas parallel transmission does.

4.9.1.2 Here are some keywords to do with communication:

- Bit Rate

  The rate at which data is transmitted across a digital network in bits per second
- Baud Rate

  The number of electrical state (symbol) changes per second, The baud rate can be different than the bit rate if more than one bit is encoded into each symbol change.
- Bandwidth

  Bandwidth is the difference between the upper and lower frequency of a range of frequencies, it is typically measured in hertz (Hz). The bit rate is directly proportional to the bandwidth of the network.
- Latency

  The time delay that occurs when transmitting data between devices.
- Protocol

  Rules and conventions for communication between network devices.

9.2 Networking

4.9.2.1 A physical star topology is a when a network of devices is connected in such a way that each workstation has a dedicated cable to a central computer or switch.

Advantages of Star topology	Disadvantages of star topology
Fast connection speed as each client has a dedicated cable	Expensive to set up due to increasing cabling costs
Will not slow down as much as other network topologies when many users are online.	If the cable fails then that client may not be able to receive data
Fault- finding is simper as individual faults are easier to trace	Difficult to install as multiple cables are needed. The problem is exaggerated where the LAN is spread over multiple buildings
Relatively secure as the connection from client to server is unique	the server can get congested as all communications must pass through it
New clients can be added without affecting the other clients	
If one cable or client fails, then only that client is affected	

A logical bus network topology is the concept of a network layout that uses one main data cable as a backbone to transmit data.

Advantages of Bus topology	Disadvantages of Bus topology
Cheaper to install than a star topology as only one main cable is required	Less secure than a star topology as all data are transmitted down one main cable
Easier to install than a star topology	Transmission times get slower when more users are on the network
Easy to add new clients by branching them off the main cable	If main cable fails, then all clients are affected
	Less reliable than a star network due to reliance on the main cable
	More difficult to find faults

the difference between a physical and logical topology is that a physical topology is how a network is laid out in the real world, and the logical topology is the conceptual way in which data is transmitted around a network. It is possible for a certain physical topology to act as a different logical topology, for example a physical switch topology that uses a switch could be made to emulate a logical bus topology by switching the switch for a hub and making sure all the workstations can follow bus network protocols.

4.9.2.2 Peer-to-peer networks are networks where all the devices in the network share resources rather than having a dedicated server (each workstation can act as a client or a server), it is mainly used in homes to allow all computers access to the printer and the internet. Client-server networks are networks where one computer has the main processing power and storage and the other computers act as clients requesting services from the server, such as access to files, the internet, printer, emails, and applications, it is mainly used in LANs with a large number of users.

4.9.2.3 WIFI enables us to create a wireless local area network that is based on international standards (IEEE 802.11), and is used to enable devices to connect to a network wirelessly. To be able to connect to a wireless network, you will need two components, a wireless network adapter and a wireless access point. There are many ways to secure wireless networks:

- Strong encryption of transmitted data

  This can be done using WPA (WiFi Protected Access)/ WPA2

- SSID broadcast disabled

  The SSID (Service Set Identifier) is the way in which a machine identifies a certain network, so by disabling broadcast, only computers that know the SSID can access the network.

- MAC address white list

The MAC (Media Access Control) address is unique for each device that is connected to a network and is used set when the manufacturer makes the NIC (network interface card). Due to the fact that each MAC address is unique, you can use this to allow only certain people with a specific MAC address to use the network.

There is one protocol which you need to know about, which is the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). The basic premise of the protocol is that a device checks with the transmission medium to see whether if it is idle or if it is busy, if it's busy it will wait a random amount of time before checking again (as this reduces the chances of two devices asking at the same time). If it's idle, then the computer will try to send data along the transmission medium and wait for a response to say that all the data got there, if it doesn't get this response, it waits a random amount of time before trying again. An extension to this protocol is RTS/CTS (Request to Send/ Clear to Send) in which the device first sends out a RTS to the receiving device/ node and if a CTS message is received from the receiving node then the data is sent to the receiving node, which in turn sends back acknowledgement of receiving the data.

9.3 The Internet

4.9.3.1 The internet is a global wide area network of networks. Packet switching is a method for transmitting packets of data via the quickest route on a network. Pieces of data are broken down into smaller packets, within these packages, as well as the original data, there is also information on the destination address. Using this information, the packet can be sent from the sender to the receiver. The router is a device that receives packets or from one host or router and uses the destination IP address that they contain to pass them correctly formatted, to another host or router. The router is responsible for carrying out the packet switching.
The main contents of a package are as follows:

- Header
  - Source Address
  - Destination Address
- Body
  - Data
- Footer
  - Checksum

A router ia a device that receives packets or from one host (computer) or router and uses the destination IP address that they contain to pass them correctly formatted, to another host (computer) or router. A gateway is A device used to connect networks using different protocols so that information can be successfully passed from one system to another. Thus routers can be used within a network that uses the same protocols within, and a gateway is used to connect networks that don't use the same protocol.
Routing is the process of directing packets of data between or within a network. This is achieved via packet switching, The process of packet switching is done via routers and gateways.
A Uniform Resource Locator (URL) is a method for identifying the location of resources on the website. The domain name is the part of a network address which identifies it as belonging to a particular domain. The IP address is a unique string of numbers separated by full stops that identifies each computer using the Internet Protocol to communicate over a network.
4.9.3.2 A firewall is a piece of hardware or software for protecting against unauthorised access to a network. It does three different ways:

- Packet Filtering

  This examines the contents of packets on a network and rejecting them if they do not conform to certain rules (it checks the header of the packet to see if it is from a recognised source). This is done after the packet has been routed through the internet, but before the packet is routed through a LAN connected to a machine

- Proxy Server

  In this case, all requests to the internet are done through a separate proxy server, which is itself connected to the internet, so the original device itself is not directly connected to the internet. This also means that all of the packets can be processed by the proxy server, and it can return only legitimate packages to the original machine

- Stateful Inspection

  This is where the contents of a package are checked, and the package as a whole is rejected if it is either from an unauthorised source or it may be rejected if it is not part of a recognised on-going communication.

There are two main types of encryption techniques that are used over a network:

- Symmetric Encryption
  1. Alice encrypts data use a key
  2. Alice sends the encrypted data to Bob
  3. Alice sends the key to Bob
  4. Bob decrypts the data using the Key
- Asymmetric Encryption
  1. Alice and Bob both have a public key
  2. Alice has a private key A, and Bob has a private key B
  3. The public key, as well as Alice's and Bob's private key are combined to create a new key, which they both know (note, this new key is never sent across the server, neither are the private keys, if you want to understand how this works, look up Diffie-Hellman Key excange), this new key is called their common secret
  4. Alice encrypts the data using the common secret
  5. Alice sends the encrypted data to Bob
  6. Bob decrypts the data using the common secret

The power in asymmetric encryption is that because the private keys of the two people communicating is never explicitly sent along the network, the message is nigh on impossible to decrypt. I'm going to quickly go into the Diffie-Hellman key exchange, you don't need to understand the process, you just need to know that because of the way the keys are exchanged, it is nigh on impossible to find the private keys, and thus to find the common secret:

1. Bob and Alice have a public key p (keep in mind all of the key are integers)
2. Bob and Alice agree before hand to use base g (where g is a primitive root modulo p)
3. Alice has a private key a
4. Bob has a private key b
5. Alice calculates $B = g^a \mod p$
6. Bob calculates $A = g^b \mod p$
7. Alice sends Bob B
8. Bob sends Alice A
9. Alice calculate $C = B^a \mod p$
10. Bob calculate $C = A^b \mod p$
11. The number Bob and Alice calculate at the end is the same, thus they have calculated the primary key

You can now see that the private key is never sent, and both private keys are needed to calculate the common secret, so even if the transfer was intercepted, the user would need to have knowledge of the private keys, as well as g in order to decrypt any messages (p can be gotten as it is the public key)
A Digital certificate is a way of ensuring that an encrypted message is from a trusted source, they are also known as Secure Socket Layer (SSL) certificates. They are obtained from a certification authority, who is a trusted organisation which is responsible for issuing digital certificates, as well as digital signatures. A digital signature is a method of ensuring that an encrypted message is from a trusted source, this is done as follows:

- Alice has a message M, and also has her own private key, and a public key
- The message goes through a public hashing algorithm, we will call this message H(M)
- H(M) is encrypted using Alice's private key (This is the digital signature)
- The message M and digital signature are both sent to Bob
- M goes through the hashing algorithm, and digital signature is decrypted. If these two values are the same, then the message was not altered, and is thus safe.

There are 3 types of malware you need to be aware of:

- Worms

  These are a type of virus that are able to replicate themselves and are able to propagate around a computer network. It doesn't need to be attached to a file to infect a machine. It's a small program that exploits a network security weakness (security hole) to replicate itself through computer networks.
- Trojans

  This is a program that hides in or masquerades as desirable software, such as utility or a game, but attacks computers it infects.
- Viruses

  this is a small program attached to another program or data file. It replicates itself by attaching itself to other programs.

There are a few things that we can do to improve our security:

- Improve the code quality of code, so it is harder, preferably impossible, to find exploits in code, so it is harder to infect a machine. An example of an exploit that may be used is buffer overflow.
- Monitoring of incoming and outgoing data, to make sure that it is all legitimate.
- Protection of the machine via a fire-wall and anti-virus software which is able to scan files on a machine whenever they are accessed to check if they are valid.

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

4.9.4.1 TCP/IP is made up of 4 layers

- Application Layer

  This layer is responsible for saying which protocols are being used within the current application (e.g. FTP, HTTP, HTTPS, POP3, SMTP, SSH, etc.)
- Transport Layer

  This layer is responsible for setting up a two-way connection between the two hosts and implements the Transmission Control Protocol (TCP)
- Network Layer

  This layer is responsible for defining the IP address of the devices involved in the communication, and handles creation and routing of packets.

- Link Layer

  This layer is responsible for actually sending the data across the physical network.

A socket is an endpoint of a communication flow across a computer network (it is software). The role of the socket within the TCP/IP stack is to say where all of the data should go once it has reached the machine, it tells the machine which application should deal with this data, this is done via the use of relevant ports. The Media Access Control (MAC) address is the unique address for all network adapter (it comprises of 6 numbers, 2 hex digits long each). It is used to identify which machine a packet should go to within a network. If the MAC address of a machine matches the MAC address of the destination of a packet, then the packet is sent to the machine's link layer.
Here are a list of Common ports:

Service Name	Port Number	Description
FTP-data	20	File Transfer Protocol (data)
FTP	21	File Transfer Protocol (control)
SSH	22	Secure Shell Protocol
SMTP	25	Simple Mail Transfer Protocol
HTTP	80	Hypertext Transfer Protocol
POP3	110	Post Office Protocol 3
HTTPS	443	Hypertext Transfer Protocol Secure

4.9.4.2 There are several standard application layer protocols you need to know:

- FTP - File Transfer Protocol

  This is a protocol for the transfer of a file across a network. The file transfer can be anonymous or non-anonymous (protected)

- HTTP - Hypertext Transfer Protocol

  This is a protocol for the transmission and display of web pages.

- HTTPS - Hypertext Transfer Protocol Secure

  This is a protocol for the transmission and display of web pages, using an encrypted form of transmission.

- POP3 - Post Office Protocol 3

  This is a protocol for receiving mail. The webserver keeps your emails on an email server until you are ready to download all of the mail.

- SMTP - Simple Mail Transfer Protocol

  This is a protocol for sending mail. This sends the mail to an online email server

- SSH - Secure Shell

  This is a protocol for remote access to machines. An SSH client is used to make a TCP connection to a remote port for the purpose of sending commands (e.g. GET for HTTP, commands sending mail for SMTP, commands receiving mail for POP3), It can be used to access a computer remotely and execute commands.

4.9.4.3 An IP address is made up of 2 parts, a network identifier (to say what network a machine is on) and a host identifier (to say what machine in the network specifically requested the data).

4.9.4.4 Subnet masking is a technique used to divide a network into smaller networks. To apply a subnet mask, we write it out in binary (using the bits required by the IP version you are using), then perform a bitwise AND with to IP address and subnet mask. To get the network identifier for an IPv4 address, we use the subnet mask 255.255.255.0, and can see that in IPv4, the network identifier is the first 3 numbers.

4.9.4.5 There are 2 IP standards, IPv4 (which is 32 bits long, made up of 4 8 bit numbers) and IPv6 (which is 128 bits long, made up if 8 16 bit numbers). We introduced IPv6 to allow for more IP addresses as we are quickly running out of IPv4 addresses as more devices become network enabled.

4.9.4.6 Non-routable IP addresses (which could be considered private) are IP addresses that cannot be routed to directly from the internet (these IP addresses are used in private Networks, such as if multiple devices are connected to the same router, this forms a private network). Routable IP addresses can be routed to directly to from the internet (e.g. a router).

4.9.3.1 Dynamic Host Configuration Protocol (DHCP) is a set of rules for allocating locally unique IP addresses to devices as they connect to a network.

4.9.3.1 Network Address Translation (NAT) is used to match private IP addresses to a public one. This is useful as it means not every device on the network has to have a unique IP address, only the router they are attached to does. It also means that only the IP address of only the router has to be registered in the DNS.

4.9.3.1 Port Forwarding is a method for devices with non-routable IP addresses to route data that is received on specific port on a router.

4.9.3.1 Client Server Model can be described as follows. Client sends a request to a server, server responds to a request by replying with a response message to client. Websocket protocol is a set of rules that creates a persistent connection between two computers on a network to enable real-time communication. To set it up, client sends server a handshake request to open up a socket, server sets up socket to allow for bidirectional communication between server and client, server sends back acknowledgement of request, client and server can now communicate with each other in real time.
Now to deal with web applications that use databases. These are known as CRUD applications, because they allow to perform four main operations on data: create, retrieve, update, and delete. In order for the CRUD operations to be used, a REST (REpresentational State Transition) API must be run on the server hosting the database, it is used to map HTTP request methods to SQL commands. JavaScript allows the browser to call the REST API, and thus get data from the database through HTTP requests. Here is a table mapping equivalence of different operations.

CRUD Operations	HTTP Request methods	SQL Methods
Create	POST	INSERT
Retrieve	GET	SELECT
Update	PUT	UPDATE
Delete	DELETE	DELETE

The data sent across the connection between the server and the client application may be transmitted as JavaScript Object Notation (JSON) or as eXtensible Markup Language (XML).
JSON is in general more useful than XML as JSON is more easily readable, more compact, easier to create, and are faster to Parse than XML. However, XML allows complete freedom in the data types that can be used, whereas JSON has limited range of data types, thus XML is useful when dealing with data types not native to JSON.

4.9.3.1 Thin-client computing is where all of the main resources, processing power, and storage capacity needed by the client machines are stored within one central machine in a network, and all of the client machines get the resources they need from this central machine. The client machines are called terminals in this case as they have little to no internal processing or local hard drive space.

- Advantages
  - Easy and Cheap to set up new clients, as few resources needed
  - New software and hardware only has to be installed on the server
  - Greater security as clients unable to install unauthorised software
  - Thin clients don't use a large amount of power
- disadvantages
  - Clients dependant on server, if server goes down, clients are impacted
  - Applications requiring a large amount of resources may not work well
  - High specification servers are expensive
  - A high bandwidth may be required to cope with client requests

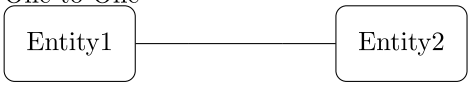
Thick-client computing is where all of the client machines contain the resources, processing power, and storage capacity needed to run their computers. Resources can be shared within the network.

- Advantages
  - Clients not completely dependant on server
  - Are able to run programs that may require more processing without affecting others
  - Server doesn't need to be high specification
  - A lower bandwidth can be used for connection to the server
- disadvantages
  - Can be expensive to set up new clients
  - New software must be installed on all of the new clients
  - Less security as clients are able to install unauthorised software
  - Thick clients require more power than thin clients

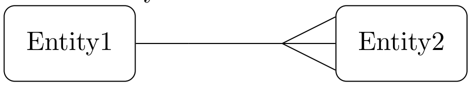
10 Fundamentals of Databases

10.1 Conceptual data models and entity relationship modelling

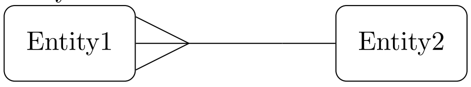
During the analysis stage of development, the developer will decide the way in which databases will be designed in order to hold the needed data. To do this they will create a data model of the database. One way to represent this data model is to create entity descriptions which are often formatted as follows:
Entity1 (Attribute1, Attribute2, Attribute3, ...)
Where the entity is the object data is being stored about, and the attribute is a piece of information to be stored about an entity, underline may be used to show entity identifier. An example of a data model for a Library may be:
Book (BookID, Title, Author, Publisher, PublishingDate)
Member (MemberID, Forename, Surname, Age)
Rental (LoanID, MemberID, BookID, DateofRelease, DateofReturn)
Another way to represent a data model is to use an entity relationship diagram, there are several relations that can be shown using this diagram:
One to One



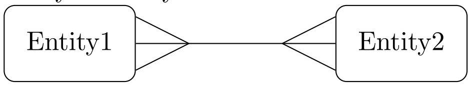
One to Many



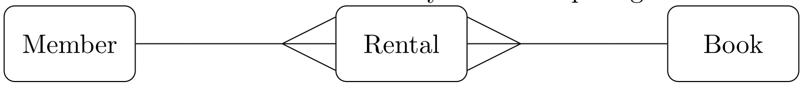
Many to One



Many to Many



If we were to represent the top example using an entity relationship diagram, we would first need to decide how each of the entities are related to one another. We can see that there is no direct relationship between a member and a person, they are only related to each other via rentals. We can see that each member can rent multiple books, so each member can have multiple rentals, but the reverse isn't true, each rental cannot have multiple members, each rental is specific to one of the library members. This means there's a one to many relationship between the members and the rentals. We can also see that each book may be part of multiple rentals, but the reverse isn't true (a rental cannot contain multiple books), so we can say there is a many to one relationship between rentals and books. An entity relationship diagram of this would be as follows:



10.2 Relational databases

A relational database is a collection of tables which are linked together by means of primary and foreign keys, these are what form the relationship between tables. There are 4 key terms which we need to define:

- Attribute

  A property or characteristics of an entity

- Primary Key

  An attribute which uniquely identifies each record in the table, this is linked to the concept of an Entity identifier, which is an attribute that uniquely identifies each instance of an entity.

- Composite Primary Key

  A set of attributes which uniquely identify each record in a table

- Foreign Key

  An attribute in a table that is a primary key in another table and is used to link tables together.

10.3 Database design and normalisation techniques

Normalisation is a technique used to produce a normalised set of entities in a database. It is done to make sure that the data within a database is stored efficiently. This involves removing redundant data, and making sure each attribute is atomic (meaning that the attribute can't be broken down further. An example of atomisation is breaking up the attribute `Name` into `forename` and `surnname`). You need to know the third normalised form (3NF), to do this we also need to define first normal form (1NF) and second normal form (2NF).
For our description of normalisation, we will start from the following table:

StudentID	Name	Teacher	Class	Subject
1	Nathan Douglas	Ms. Maharaj, Mrs. Ozgu, Mr. Hamshar	R1,R2,R3	English, Chemistry, Maths
2	Helen Douglas	Mrs. Maharaj, Mr. Johnston, Mrs. Read	R1,R4,R5	English, Biology, History
3	Ryan Potter	Mr. Hamshar, Mr. Hillary, Mr. Johnston	R3, R6, R4	Maths, Physics, Biology

First Normal Form (1NF) - This involves making sure each attribute is atomic (can no longer be broken down further) and only a single value is entered in each attribute (all data in the table is atomic). So to convert our table to first normal form, we would get:

StudentID	Forename	Surname	Teacher	Class	Subject
1	Nathan	Douglas	Ms. Maharaj	R1	English
1	Nathan	Douglas	Mrs. Ozgu	R2	Chemistry
1	Nathan	Douglas	Mr. Hamshar	R3	Maths
2	Helen	Douglas	Ms. Maharaj	R1	English
2	Helen	Douglas	Mr. Johnston	R4	Biology
2	Helen	Douglas	Mrs. Read	R5	History
3	Ryan	Potter	Mr. Hamshar	R3	Maths
3	Ryan	Potter	Mr. Hillary	R6	Physics
3	Ryan	Potter	Mr. Johnston	R4	Biology

So we have broken down the name attribute into forename and surname, we have also made sure that all of the data is atomic, creating new records to make sure this is the case.

Second Normal Form (2NF) - This is achieved by first making sure the table is in first normal form, and then removing attributes that only partially depend on the key by creating new tables. So for example, the process of converting from 1NF to 2NF may be as follows

StudentID	Forename	Surname	TeacherID
1	Nathan	Douglas	1
1	Nathan	Douglas	2
1	Nathan	Douglas	3
2	Helen	Douglas	1
2	Helen	Douglas	4
2	Helen	Douglas	5
3	Ryan	Potter	3
3	Ryan	Potter	6
3	Ryan	Potter	4

TeacherID	Teacher	Class	Subject
1	Ms. Maharaj	R1	English
2	Mrs. Ozgu	R2	Chemistry
3	Mr. Hamshar	R3	Maths
4	Mr. Johnston	R4	Biology
5	Mrs. Read	R5	History
6	Mr. Hillary	R6	Physics

This first table is still not in 2NF as a composite key of StudentID and TeacherID would be needed, however then Forename and Surname only depend on StudentID, thus it needs to be broken down further, thus we will need to create a new table to split this up again, obtaining:

StudentID	Forename	Surname
1	Nathan	Douglas
2	Helen	Douglas
3	Ryan	Potter

StudentID	TeacherID
1	1
1	2
1	3
2	1
2	4
2	5
3	3
3	6
3	4

TeacherID	Teacher	Class	Subject
1	Ms. Maharaj	R1	English
2	Mrs. Ozgu	R2	Chemistry
3	Mr. Hamshar	R3	Maths
4	Mr. Johnston	R4	Biology
5	Mrs. Read	R5	History
6	Mr. Hillary	R6	Physics

So we have now broken the first table down into 2 tables where all of the data is entirely dependant on the primary key, and now the whole database is in 2NF.
Third Normal Form (3NF) - The database must be in second normal form, and then all non-key attributes that are dependant on other non-key attributes are removed via creation of a new table. In the above example, this means we will get rid of the subject attribute as it depends on class, and then create a new table containing class and subject:

Student

StudentID	Forename	Surname
1	Nathan	Douglas
2	Helen	Douglas
3	Ryan	Potter

StudentTeacher

StudentID	TeacherID
1	1
1	2
1	3
2	1
2	4
2	5
3	3
3	6
3	4

Teacher

TeacherID	Teacher	Class
1	Ms. Maharaj	R1
2	Mrs. Ozgu	R2
3	Mr. Hamshar	R3
4	Mr. Johnston	R4
5	Mrs. Read	R5
6	Mr. Hillary	R6

Subject

Class	Subject
R1	English
R2	Chemistry
R3	Maths
R4	Biology
R5	History
R6	Physics

10.4 Structured Query Language (SQL)

There are 5 things you need to be able to do in SQL
Retrieve Data
This is used to query a database, and easily get data from inside a database, the query is often of the current form:

```
SELECT FieldName1, FieldName2, FieldName3, ...
FROM Table1, Table2, Table3, ...
WHERE Condition,
ORDER BY FieldName DESC/ASC;
```

You can easily view a whole table by using the following form:

```
SELECT * FROM Table;
```

So for example, if we wanted to find all the Subjects that Nathan Douglas had, we could use

```
SELECT Student.Forename, Student.Surname, Subject.Subject
FROM Student, StudentTeacher, Teacher, Subject
WHERE Student.Forename = "Nathan" AND Student.Surname = "Douglas" AND Student.StudentID = Student
ORDER BY Subject.Subject ASC;
```

and it would return:

Forename	Surname	Subject
Nathan	Douglas	Chemistry
Nathan	Douglas	English
Nathan	Douglas	Maths

Here are all of the boolean comparators you should know:

Operator	Description	Example
=	Equal	`Name = ‘Bob’`
¡¿	Not Equal	`Income <> 0`
¿	Greater Than	`Height > 1.4`
¡	Less Than	`Weight < 100`
¿=	Greater Than or Equal To	`Age >= 18`
¡=	Less Than or Equal To	`Temperature <= 25`
BETWEEN	Between an Inclusive range	`Cousins BETWEEN 1 and 7`
LIKE	Search for a Pattern (% equivalent to .* regex, _ equivalent to . regex)	`Name LIKE ‘%oh_’`
IN	Specify multiple possible values of a row	`Siblings IN (1,2,3)`

(If you want to be snazzy, you could teach yourself about inner joins, which can make queries above shorter and easier to read, but they aren't necessary so...)
Update Data
This is used to change data that is already present in a table, the general format for this sort of SQL is

```
UPDATE Table
SET FieldName1 = value1, FieldName2 = value2, ...
WHERE Condition;
```

for example, if we wanted to change Nathan Douglas's Name to Wally West, I would write the following SQL statement:

```
UPDATE Student
SET Forename = ’Wally’, Surname = ’West’
WHERE Forename = ’Nathan’;
```

Insert Data
This is used to insert data into an existing table, if you want to insert data into specific columns (leaving the others null, or letting them be auto-completed via the set-up of the table), you would use the following

```
INSERT INTO Table (FieldName1, FieldName2, FieldName3, ...)
VALUES (value1, value2, value3, ...);
```

If you want to insert into all of the columns in a table, you can use the above, or the following

```
INSERT INTO Table
VALUES (value1, value2, value3, ...);
```

If I wanted to add a new student into the database with StudentID of 4 called Anik Roy, I would write the following statement:

```
INSERT INTO Student
VALUES (1, Anik, Roy);
```

Delete Data
This is used to delete records within a table, it has general form:

```
DELETE FROM Table
WHERE Condition;
```

To delete all the records in a table, we can use:

```
DELETE FROM Table
```

So if we wanted to delete Helen Douglas from the Student table, we would write the following:

```
DELETE FROM Student
WHERE Forename = 'Helen' AND Surname = 'Douglas';
```

Create Table
To add a table to a data base we can use the following form:

```
CREATE TABLE Table (
FieldName1 datatype,
FieldName2 datatype,
FieldName3 datatype,
...
);
```

A list of common data types are:

character(n)	A string of fixed length n
varchar(n)	A string of variable length less than or equal to n
boolean	True or False
int	Any integer (positive or negative whole number)
decimal(p,s)	A decimal number of length p , and s digits after the decimal point
real	Any real number
date	The date (format `DATE'YYYY-MM-DD'`)
time	The time (format `TIME'HH-MM-SS'`)

To create the Student table, I would write the following statement:

```
CREATE TABLE Student (
StudentID int AUTO_INCREMENT,
Forename varchar(255),
Surname varchar(255),
PRIMARY KEY (StudentID)
);
```

The `AUTO_INCREMENT` means that the StudentID does not need to be inputted every time a new set of data is added, and the `PRIMARY KEY (StudentID)` sets StudentID as the primary key.

10.5 Client server databases

Within some projects involving more than one person, multiple people may need to be able to access and edit the database. A way to implement this is to use a client-server database, where the database is hosted on a server that different users can access, all processing is then done on this main server.
When working with this sort of database, we need to take into account what may occur if two people were to access and update the same piece of data at the same time (concurrently). If the situation isn't dealt with properly, this could lead to loss of data, as the data from one user could be overwritten by another user, but the first users data may be more accurate. There are 4 main ways to deal with concurrent access:

- Record Lock

  For this, you just only allow one user at a time to edit a record, and lock the record for other users. This means that the first user is able to read and write data, whereas anyone else can only read data, until the first user is done.

- Serialisation

  For this, we allow multiple users to edit the database at the same time, however if a user updates the database, all of the other users are forced to restart, and are thus presented with an up to date database. This means that only one action occurs at a time.

- Timestamp ordering

  This is a way to order the commands so that they can be serialised. Whenever a user accesses a database they are given a timestamp (which can just be represented by an incremented integer). To be able to write to the database, you must have timestamp later than the latest read or write transaction. To be able to read from the database you must have timestamp later than the latest write transaction.

- Commitment ordering

  In this, the transactions are created on the device, and then sent to the server to be scheduled, depending on the commands within the transaction and the time which the transaction occurred.

11 Big Data

11.1 Big Data

Big data is a general name for extremely large quantities of data which are difficult to store and analyse. There are three ways in which big data can be described:

- volume

  To large to fit onto one server

- velocity

  Large amounts of incoming data

- variety

  Data in many different forms (structured, unstructured text, multimedia, etc.)

The main problem within big data is the lack of structure to the data that is received, being unable to structure the data, means that it becomes very difficult to try and spot trends within the data. This also means that relational databases are not very useful, as they require fixed rows and columns and (as stated above) this data is quite varied and would thus not fit into a relational database. To analyse this sort of data, machine learning is needed in order to spot trends and pick out interesting pieces of data. Another fact to consider is that even if we were able to model it into a relational database, the data is often larger than one server, so we have to consider scalability, and relational databases don't scale well across multiple machines.
Example of sources of Big Data are:

- Networked sensors
- Smartphones
- video surveillance
- mouse clicks
- real-time application
- the internet

These are all sources of big data, which continuously stream data.
To highlight the main issues that arise within big science:

- Datasets are extremely large, leading to difficulties in storage and analysis of the data
- Unstructured data can be very hard to analyse
- Machine learning needed to require any useful information from the data set
- Massive storage and processing power needed
- Data is constantly being updated and added
- Correlation does not equal causation

- Concurrent access

Whenever the data that needs to be stored is too large to fit on a single server, a distributed system must be used in order to process the data. To perform this type of processing, functional programs are often used as they allow for easy to write code that is correct, and can also work efficiently on a distributed system. Functional programming languages has 3 main properties that make them useful for this:

- Immutable data structures

  This means that once a variable is instantiated, it's value cannot be changed, for example, if you said `x=5`, for the rest of the programs lifetime, `x=5`

  This makes it easier to write correct code as a function can't accidentally change the value of a variable

- higher order functions

  This means that one function takes another function as input, for example if you defined a function `applyTwice` to apply a given function twice on a given parameter, you may use the following (written using haskell):

  ```
  applyTwice :: (a -> a) -> a -> a
  applyTwice f x = f (f x)
  ```

  This makes it easier to write correct code as you don't have to re-define certain functions to be used within another. This also allows it to bee run on multiple devices, as independent functions can be run on different devices, for example if you wanted to calculate x^6 by using the fact it equals $x^3 \times x^2 \times x$, you can have the three terms be calculated on different devices, then return the result to the original device to finally calculate x^6

- statelessness

  This means that within a functional program there is no idea of having a state. Within an imperative or an object oriented program, during the programs lifetime, the value of global variables will change due to work done by a function, thus the program changes state, the change caused to the variables due to the function, is called the side effect of the function. Whereas in pure functional programs, the variables cannot change value, therefore there is no change in state, the functions have no side effect and are thus said to be pure functions. Thus pure functional programming languages are stateless.

  This is useful for running on a distributed system as it means that each of the separate machines don't need to worry about the state of the whole program, as the program has no inherent state, thus allowing different parts of the code to be easily run in parallel.

In order to represent a big data dataset, a fact based model may be used, within this conceptual model, the dataset is represented as a large amount of atomic facts (facts that cannot be broken down further). Within this model, facts are immutable, meaning they cannot be changed (only deleted if there was an error when the data was inputted), they can not be directly updated, to do this a new fact must be added which would be more up to date.
To represent the relationship between different facts, a graph schema may be used, where the nodes represent entities such as student and teachers, and edges represent the relationship between two entities. Properties are also shown, which is simply information about an entity, e.g. age, height, etc.
Fact-based model represents the data, graph schema represents the relationship between pieces of data.

12 Fundamentals of Functional Programming

12.1 Functional programming paradigm

Before we get into this, the easiest way to understand most of this is to start learning a functional programming language, I suggest learning Haskell and using *http://learnyouahaskell.com/chapters*
4.12.1.1 within functional programming, each function which you define has a type. A functions type dictates what inputs it can take, and what it will output. For example, the function `f :: a -> b`

defines a function called `f`, which takes an input of type `a`, and outputs a value of type `b`. in the above example, `a` is the domain and `b` is the co-domain. The following is somewhat implicit, but the domain is of type `a` and the co-domain is of type `b`.

4.12.1.2 A first-class object fulfils the following criteria, it can:

- Appear in an expression
- Be assigned to a variable
- be assigned as an argument
- be returned in function calls

In functional programming languages, and object oriented programs which support a function object, functions are first-class objects, this means they can be used as arguments of functions, as well as output of functions.

4.12.1.3 Function application is the act of applying a function to its arguments, for example if we have the function (where A x A is the cartesian crossproduct of the set of real numbers with itself):

```
multFour :: (Num A) => A x A x A x A -> A
multFour w x y z = w * x * y * z
```

Then `multFour 2 3 4 5` would represent function application of the function add on the arguments 2, 3, 4, and 5 (which would produce 120)

4.12.1.4 Partial function application is when you provide a function less arguments than it can take, meaning that a function is returned with the number of arguments that you left out. For example, if we were to define a function as follows

```
multFour :: (Num a) => a -> a -> a -> a -> a
multFour w x y z = w * x * y * z
```

Now if we were to do `multFour 3 4`, this is a partially applied function, as we have only supplied 2 of the 4 arguments needed, so two fully apply the function, two more arguments would needt to provided to fully apply the function. We can somewhat see this if we rewrite the type of the function in its full form. Its type would be:

```
multFour :: (Num a) => a -> (a -> (a -> (a -> a)))
```

And thus we see that this function is in fact a series of partial function applications, that when done 4 times, results in a number. When we normally write this out, we leave out brackets as they are unnecessary, and make the type look more confusing than is needed.

4.12.1.5 Composition of functions, is the process of taking a function of a function, producing a new function, for example if we have $f(x) = x^5$ and $g(x) = 2x{+}3$, then $f(g(x)) = f(2x{+}3) = (2x{+}3)^5$. Another way to represent $f(g(x))$ is $f \circ g$. To define composition in general using haskell:

```
composite :: (B -> C) -> (A -> B) -> A -> C
composite f g x = f (g x)
```

12.2 Writing functional programs

4.12.2.1 As I said at the beginning you should learn how to at least do some basic programs in a functional programming language, not only to help you understand the concepts in this chapter, but also because it is a part of the specification.
Here is all you need to know about higher order functions. It is a function that takes a function as an argument or returns a function as a result, or does both.
There are three functions that are core to most functional programs that involve a list:

- Map

  This is a higher order function which applies a provided function into a provided list of values, return a list of the results, for example, if you wanted to add 2 to [1,2,3,4,5], you could use: `map (+2) [1,2..5]`, which would return `[3,4,5,6,7]`

- Filter

  This is another higher order function which is able to filter a list of values when provided a filtering function (it must return true of false and take only one argument which is the same type as the list provided), an example for a list of even numbers from 1 to 10:

```
even n = if n 'mod' 2 == 0 then True else False
filter even [1,2..10]
```

- Reduce/ Fold

  This higher order function reduces a list of values to a single value by repeatedly applying a combining function to the list of values, In haskell, there is `foldl` and `foldr`. They both work slightly differently, but in most cases, you will probably need `foldl` (if you are seriously interested you can google the difference, it's to do with the order in which the function is applied). So for example, if we wanted to calculate 10!, we could use `foldl * 1 [1,2..10]`. The random 1 in the middle simply says what the starting value should be.

12.3 Lists in functional programming

4.12.3.1 Time for random things functional programs do with lists. Whenever you write a list you will most likely write it like [1,2,3,4,5]. In functional programming languages, a list can be written as a concatenation of its head (its first element) and its tail (the rest of the list), so for example, in haskell [1,2,3,4,5] = 1:[2,3,4,5] = 1:2:[3,4,5] = 1:2:3:[4,5] = 1:2:3:4:[5] = 1:2:3:4:5:[]. [1,2,3,4,5] is syntatic sugar for 1:2:3:4:5:[] (it represents the same thing, and the first is easier to read and write). As I said above, the head of the list is the first element of the list, and the tail is the rest of the list. A list can be empty and is represented by [].
There are several operations you need to know:

- return the head of a list

  `head [1,2,3,4,5]` $\Longrightarrow$ `1`

- return the tail of a list

  `tail [1,2,3,4,5]` $\Longrightarrow$ `[2,3,4,5]`

- test for empty list

  `null [1,2,3,4,5]` $\Longrightarrow$ `False`, `null []` $\Longrightarrow$ `True`

- return length of a list

  `length [1,2,3,4,5]` $\Longrightarrow$ `5`

- construct an empty list

  `Let list = []`

- prepend an item to a list

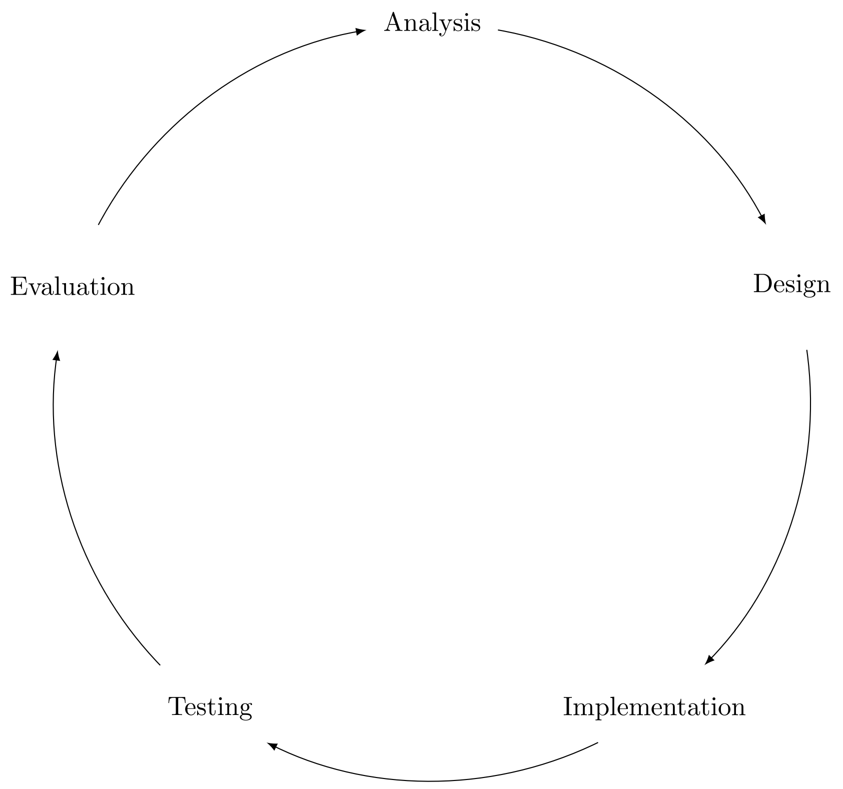
  `[0] ++ [1,2,3,4,5]` $\Longrightarrow$ `[0,1,2,3,4,5]`

- append an item to a list

  `[1,2,3,4,5] ++ [6]` $\Longrightarrow$ `[1,2,3,4,5,6]`

13 Systematic Approach to Problem Solving

13.1 Aspects of Software Development



3.3.1.1 Analysis is first stage of system development cycle where the problem is initially identified (through understanding what has promted the need for a new computer system) and researched(through gathering information and feasibility study).
During this stage the problem you will need to try and define the actual problems that will need to be solved, and try to be realistic with the resources available, a list of constraining factors may be identified, to obtain a feasible solution. A specification and discussion of the scope of the work should be done if you are working for someone else.
There are several reasons why a company may want a new computer system, some of the reasons are:

- The existing system is not capable of handling the increasing volume of data being required.
- New technology means the old system is obsolete.
- The current system may be inflexible or inefficient.
- New technology has created new opportunities for expansion/ creation of systems.
- Commercial Reasons, Improved to generate increased demand of the system.
- Develop systems for a new platform or Operating System.
- To take advantage of increased processing and network power

If you are analysing an existing system, then you should aim to gather information about the current system, this can be done in many ways such as:

- Interview people who are involved with the current system.

  This can be quite time consuming but allows you to easily obtain in-depth information about the system, although this information will be somewhat subjective.
- Asking people currently involved with the system to answer a questionnaire or survey.

  Questionnaires allow for quick data gathering, however they often only allow limited information to be gathered as if you want easily analysable data, you will have to use closed questions.

- Observation of current practices when interacting with the system.

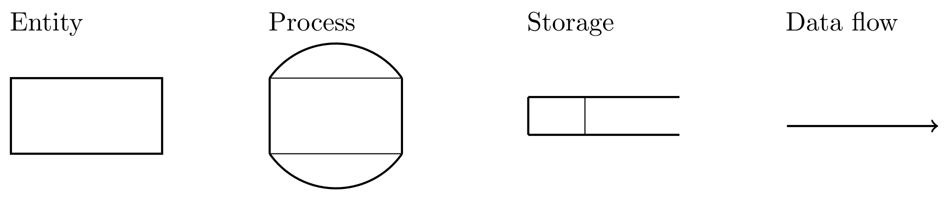
  Although this can be very time consuming, it is more objective rather than subjective and can lead to spotting something others did not.

- Examination of the current system files, paperwork and processes.

  This will help to inform you of the data requirements of the new system, how to develop the human-controller interface (HCI), and the overall scope of the problem.

A feasibility study is an analysis of whether it is possible or desirable to create a system. The report will indicate how practical a solution may be in terms of many factors such as time, availability of software and hardware, and UI.

3.3.1.2 Design is the second stage of the system development cycle and is where the algorithms, data and interface are designed.
On big projects it is often difficult to consider the problem as a one problem, so the problem is broken down into smaller units, which can then themselves be broken down further, this is known as top-down design. This is similar to modular design where the main problem is broken down into several self-contained modules. Both approaches allow multiple programmers to work on the solution, and both require a defined feel so that the different parts of the program flow together. The main risk with Top-down design is that a person may get too invested in a specific detail without having completed the overall section fo the program.
Data Flow Diagrams are a visual method of showing how data passes around a system, it only requires 4 symbols:



During the design phase, you may want to identify and describe the algorithms to be used within the system. It may be appropriate to write the algorithms in pseudo-code that reflects the programming paradigm being used.
A data dictionary is a list of all the data being used in the system including its name, length, name, data type and validation. This should be used only when relevant (mainly in databases).
A variable table is a list of all the variables that a program will use, including names, scope and data types. this is useful as it allows the programmer to have tighter control of the program being made. At this stage, names of subroutines can also be identified and the type of data to be returned.
When building a system, the volumetrics of the system should be considered. The volumetrics of a system is the amount of data the system has to deal with in a given time span. Larger volumetrics implies a different way of handling data, leaning towards mass optimisation of a system. Volumetrics are very important when dealing with databases and the realisation needs to be made that the databases we have seen (e.g School Registers) are microscopic when compared to databases with over 20 million members and 100s of fields (e.g DVLA database).
The human-computer interface is the term given to any form of interaction between a computer and its user. Factors to consider are:

- Ease of Use: The user should be able to use the system reasonably easily.
- Target Audience: The age and personality of the user will directly impact how they will use the system.
- Technology: The technology the user uses will directly impact the features and limits of the system being developed.
- Ergonomics: The comfortability of the system should be taken into high regard.

3.3.1.3 Implementation is the third stage of the system development cycle and is where the actual code and data structures are created. Just to note, not all of the program needs to be written at once for example, one module may be being implemented while others are being designed.
A prototype is a stripped down version of a whole system built at the design stage to test whether the concept works. At this stage the end user is asked to comment on the product so far, and they will check to make sure all the major functions work as expected.

3.3.1.4 Testing is the fourth stage of the system development cycle and is where a range of tests are performed on the program using a variety of data. There are three different types of data that is used once each module has been completed. These are:

- Normal

  This is data that the system is expected handle without issue.
- Boundary

  This is data that is on the extremes of the acceptable values.
- Erroneous.

  This is data that is clearly incorrect and expect the program to catch the error.

During Development several different tests are done on the modules, these test are:

- Black Box Testing

  This test checks whether a specific input causes the expected output, without seeing how it is actually done.
- White Box Testing

  This test checks all pathways through the code, looking inside it and potentially adding extra commands to check what is happening.
- Unit Testing

  This checks that each unit/ module/component is fully functional by itself. It uses both white and black testing.

Once all the units have been tested and are fully functional, the units are first put together to create large modules and undergo integration testing, and checks whether the different modules made from individual units, will work together. Once integration testing is complete, the program then undergoes system testing as one large single unit. System testing involves a range of tests to be carried out on the system to check whether the system satisfies the specification set out in the analysis stage. The main parts of system testing are:

- Alpha Testing

  this is done in-house once the system is complete and tries to simulate possible eventualities of the system under normal conditions. The benefits are it allows the program to be corrected before the full system is released, and the original set of data is known and can thus be more easily analysed.
- Beta Testing

  This occurs after alpha and is when the system is given to users, who are expected to give feedback about bugs, so that the programmers can resolve them. The benefits of this is that due to the fact it has been given to people who have never used the system, they may use it in different ways and therefore highlight problems they may not have been seen in alpha testing.
- Acceptance Testing

  This is where the intended user test the system by entering their own data and making sure the system matches the specification that was agreed with the program writers.

3.3.1.5 Evaluation is the final stage of system development where the system is judged according to certain criteria (i.e. The specification set in the Analysis stage) and improvements are suggested. There may be several criteria used to evaluate a system, e.g:

- Functionality
- Ease of Use
- Ease of Implementation (How easy is it to go from old system to new system)
- Reliability
- Performance
- Cost Effectiveness
- Ease of Maintenance and Adaptability
- Longevity