OCR A Level Computer Science – Unit 1 – Glossary of terms

Strand	Sub Strand	Term	Meaning
1.1 The characteristics of contemporary processors, input, output and storage devices - Components of a computer and their uses	1.1.1 Structure and function of the processor	Register	A small, fixed width piece of memory inside the CPU. Usually between 16-128 bits wide. Registers are used to hold data or instructions as they are being worked on. Registers can also be used to represent the status of various CPU components.
		Program Counter (PC)	A register used to hold the address of the next instruction to be executed. It is possible to switch tasks that the CPU is carrying out by changing the address held in the PC.
		Accumulator (ACC)	A general purpose register, usually used to hold the result of the last calculation or instruction carried out
		Memory Address Register (MAR)	Used to hold the address in memory which will be accessed next.
		Memory Data Register (MDR)	Used to hold the data which ahs just been fetched from RAM or is about to be written to RAM
		Current Instruction Register (CIR)	Where the instruction about to be run is decoded – held in the form Operator, Operand
		Data Bus	The connection between the CPU and RAM – literally a collection of wires. Used to send data between CPU and RAM
		Control Bus	Used to connect the various components in the CPU and also to RAM. Used to send control signals to co-ordinate the timing/workings of the CPU and also to tell RAM whether a read/write operation is needed.
		Address Bus	Used to connect the CPU and RAM so that addresses which need to be read from or written to can be sent to the memory controller.
		Control Unit	Used to coordinate the actions of the CPU during the FDE cycle. Sends signals down the control bus to components.
		FDE Cycle	The purpose of the CPU – the Fetch, Decode, Execute cycle. Happens continuously whilst power is on. The number of FDE cycles per second is called the “clock speed” and is measured in Ghz (Gigahertz)

		Performance - Speed	The number of FDE cycles a CPU can carry out per second. Has a direct impact on the speed at which program instructions can be executed. Measured in Ghz – billions of cycles per second.
		Performance - Cores	A core is a discrete processing unit inside a CPU – The more cores a CPU has, the more instructions can be executed simultaneously. Has a direct impact on the multitasking ability of a CPU
		Performance - Cache	Cache is a small amount of memory inside a CPU. Usually comes in different "levels" which differ in speed and size. L1 cache is the fastest and smallest, L3 is the slowest and largest. All are much, much faster than accessing RAM and therefore it is used to hold frequently accessed instructions to speed up program execution. Uses predictive algorithms to pre-fetch what the CPU thinks it will need to execute next.
		(d) The use of pipelining in a processor to improve efficiency.	Pipelining is a method of processing instructions more efficiently. Instructions are ordered by the CPU in such a way as to avoid idle cycles where nothing is being executed. In simple terms, the CPU executes one instruction, whilst decoding the second, whilst fetching a third.
		(e) Von Neumann and Harvard Architecture	The Von Neumann architecture is the basic concept on which all modern computers are based. It loosely consists of input, output, a memory store to hold programs and a CPU which has an ALU, Control Unit and Registers. One common criticism of it is the fact that instructions and data are stored in the same memory space. The Harvard architecture seeks to remove that bottleneck by storing instructions separately to data.
	1.1.2 Types of processor	(a) The differences between and uses of CISC and RISC processors.	CISC – Complex Instruction Set, RISC – Reduced Instruction Set. CISC processors are those made by AMD and Intel, they have large instruction sets designed to make them easier to program for. However, there are downsides. Complex instructions take more cycles to execute and the CPU's are usually more power hungry and produce more heat RISC CPU's are those found usually in phones such as those made by Apple, ARM, Snapdragon etc. They have the advantage that each instruction usually takes one cycle to execute, they consume low amounts of power and produce less heat. However, they are have the disadvantage of fewer instructions, meaning more may be needed to carry out similar tasks than on CISC processors
		(b) GPUs and their uses (including those not related to graphics).	Graphics Processors are highly optimised to perform calculations in parallel on large amounts of data. Other than making them ideal for generating graphics, this also makes them suitable for scientific research, medical research, cryptography and cryptocurrency applications.
		(c) Multicore and Parallel systems.	See cores above. Parallel systems are those that are optimised for running multiple tasks simultaneously.
	1.1.3 Input, output and storage	(a) How different input, output and storage devices can be applied to the solution of	Requires you to have a knowledge of common input and output devices – largely common sense.

		different problems.	
		(b) The uses of magnetic, flash and optical storage devices.	Magnetic storage includes hard drives and backup tapes. Hard drives are largely being phased out in favour of solid state drives, even in server based systems as longevity and capacity increases as costs fall. Flash storage is any non-volatile storage which uses flash memory chips to store data. Has huge advantages over other methods of storage including physical size (tiny), density of storage, low power requirements and robustness. Optical storage is largely relegated to games consoles and tv connected media players. A range of discs which are read via a laser can be used to store data. Examples are DVD, CD and Blu-Ray. Blu-Ray is the most common standard for data storage due to its capacity of between 25-50gb
		(c) RAM and ROM.	RAM = Random Access Memory. Used to hold open, running programs and data. Has to be managed by the operating system to avoid programs over writing each other or, worse, accessing the data for another program. ROM = Read Only Memory. Used to hold program code in a non-volatile flash memory chip. Usually refers to the BIOS/UEFI which is the code used to boot up a system at power on.
		(d) Virtual storage.	The concept of expanding the amount of RAM available for open, running programs. The operating system uses a section of secondary storage and makes it “appear” as extra memory. The OS manages the movement of pages of data in and out of physical memory as and when necessary

1.2 Software and software development - Types of software and the different methodologies used to develop software	1.2.1 Systems Software	(a) The need for, function and purpose of operating systems	The operating system is software which controls all hardware and software in the system. Consists of a user interface, kernel and drivers. Operating systems provide a platform for all software to run on – what this basically means is it provides a core set of services so you don't have to write code in every single program for simple things like controlling mouse movement, printing, saving and so forth. The OS will also control the resources available in a "fair" way to prevent programs monopolising the CPU, RAM or Storage
		(b) Memory Management (paging, segmentation and virtual memory).	Memory management is a key role of the OS. RAM is split into "pages" and these are allocated to programs. They provide flexibility and efficiency because they can be moved around an re-organised as necessary. The OS allocates RAM to programs, ensures programs cannot overwrite each other or access memory that does not belong to them. It also includes the handling of virtual memory where necessary. Without memory management, an OS could not offer multitasking or multiprogramming.
		(c) Interrupts, the role of interrupts and Interrupt Service Routines (ISR), role within the Fetch-Decode-Execute Cycle.	Interrupts are a key hardware feature of all CPU's. an interrupt is a request for CPU time – in other words, something has happened and it requires CPU execution time to process. Interrupts can be "requested" by hardware devices or software. The CPU recognises an interrupt has occurred but it is **not** the job of the CPU to decide when an interrupt should be serviced – this is controlled by an algorithm in the OS.
		(d) Scheduling: round robin, first come first served, multi-level feedback queues, shortest job first and shortest remaining time.	Scheduling is the algorithm used by the OS to decide when a program should be executed by the CPU. The scheduling algorithm must perform a careful balancing act to ensure all processes receive attention from the CPU, that programs receive adequate execution time so that they appear to run smoothly and finally, must not allow a process to prevent others from executing.
		(e) Distributed, embedded, multi-tasking, multi-user and Real Time operating systems.	Operating systems are usually optimised/specialised to handle a certain type of environment. Most devices we use are single user operating systems designed to give full access to all system resources to one user and their requirements. However, in a networked environment or distributed, the OS needs to handle the resources of multiple machines and users to ensure that performance is maintained for all users.
		(f) BIOS.	Basic Input Output System. More modern systems make use of UEFI – Unified, Extensible Firmware Interface. Both exist to boot a PC, the code is stored in ROM on the motherboard and called by the CPU at power on. The BIOS is designed to detect hardware, perform basic self tests and then find a device to boot from such as SSD/HDD
		(g) Device drivers.	A device driver is a piece of software which allows hardware to communicate with the operating system. An OS provides "generic" services and hardware calls, it is up to the manufacturers of specific devices to write the code which will allow their hardware to interact with an operating system, taking care of all the specifics of how it works. This is essential as it simply isn't possible for OS developers to write code for every possible device that may be connected to a system, it also enables future devices to work with operating systems without the need for them to be updated

		(h) Virtual machines, any instance where software is used to take on the function of a machine, including executing intermediate code or running an operating system within another.	Virtual machines are software which emulates (pretends to be) all of the hardware in a PC system. They are extremely useful in server environments as they allow "machines" to be more easily managed. For example, one physical server may run many virtual machines – a much more efficient use of hardware and ensures that resource use is maximised. VM's can also be backed up, moved, restored as necessary as well as have their states saved – making it possible to recover to an earlier known good point in the case of failure.
	1.2.2 Applications Generation	(a) The nature of applications, justifying suitable applications for a specific purpose.	This is common sense – you should be able to suggest an appropriate application for a given task. For example "suggest an application for the gathering and storage of large volumes of customer information" you would suggest… a database!
		(b) Utilities	Small programs which usually run in the background. They normally have one specific purpose and are often used for the maintenance of systems. For example, a backup utility would run regularly and save changes to the backup device specified. Other examples include anti-virus, compression, encryption and file management
		(c) Open source vs closed source	Open source is where the code for a program is made freely available. Usually it can be used, re-used, changed and re-released as necessary. It can be excellent for the rapid development of new systems and can reduce workload because parts of the code will be pre-written. The only downside is there is no guarantee as to the quality of the code, how robust it will be and whether it will be supported in future. Closed source or "proprietary" code is usually the property of a company or individual that has chosen not to share the code with anyone outside their organisation. This is usually done to protect the intellectual property of the developer and enable them to make money from their work/creations.
		(d) Translators/Interpreters, compilers and assemblers.	Tools used to turn high level code into machine code. Remember that the CPU cannot understand anything other than binary strings which relate to an instruction in the instruction set. Assemblers are specific to assembly language and are one of the first types of translators to be developed. They allow the quick conversion of assembly in to machine code, usually producing an executable. An assembler will take care of some memory management, the automatic calculation of addresses and simple variable control, but they offer very little support to developers beyond that. Compilers take high level code and convert to an executable in one go. They link/load other libraries as necessary and build this code into the executable automatically. Compiled programs usually run more quickly than interpreted and can be easily distributed between similar systems Interpreters are usually used to provide cross platform compatibility for code – think java script on the WWW. An interpreter is installed on the local machine which takes the code, converts it as it is executed,

			producing machine specific code as it does so. This has the huge advantage of “write once, run anywhere” but has the obvious overhead of several different interpreters needing to be created.
		(e) Stages of compilation (lexical analysis, syntax analysis, code generation and optimisation).	Compilation is not a simple operation of “code in, binary out.” Lexical analysis removes unnecessary data and information such as comments and spaces, it outputs a token stream which represents the code written. Syntax analysis takes the token stream and checks it is valid and outputs an abstract syntax tree Code generation is the complex part which parses the syntax tree and then generates appropriate machine code for each part, links in external libraries and produces the executable. Optimisation is carried out by the compiler to increase execution speed and reduce executable size. For example, it may re-order the instructions into a sequence it thinks is more likely to happen.
		(f) Linkers and loaders and use of libraries.	External code libraries allow developers to use standard code, which is fully tested and robust. Operating systems are full of libraires, and developers are free to make their own. A 3d game engine would be an example of this. Libraries contain code which is likely to be used over and over again in programs, so instead of re-inventing the wheel, we put it in a library and simply import it into our code and reference it. The problem with this is that executables can rapidly grow in size and so we mostly use dynamic linking of libraries (DLL’s in Windows) which means that the code can reference a library, but then it is only loaded into memory when it is required, saving space. Static libraries, such as those which may be created by a developer themselves, are included in the executable. This has one advantage – it is guaranteed that the library is available. Dynamic libraries may not be present for some reason or may have been updated and now be incompatible.
	1.2.3 Software Development	(a) Understand the waterfall lifecycle, agile methodologies, extreme programming, the spiral model and rapid application development.	Software development projects should follow a lifecycle. The purpose of a life cycle is to manage the stages of development to ensure that the intended outcomes actually happen, deadlines are met and budgets are not exceeded. Each of the specified methods have their own specific goal, drawbacks and advantages.
		(b) The relative merits and drawbacks of different methodologies and when they might be used.	Briefly: Waterfall: the most traditional model, followed from analysis to review with each step having to be fully completed before the next can begin. Rigid in structure and inflexible if requirements change, however, there is always documentation and accountability throughout the project Spiral: An iterative model which recognises the need for regular review and development. Each iteration sees the phases grow longer, more expensive and hopefully more productive. It is said to be “risk aware” in that the regular reviews allow developers and managers to spot and fix issues as they happen.

			Agile: Extreme programming and RAD are actually methods of implementing an agile lifecycle. As the name suggests, the agile methodology is designed to be fully flexible in order to meet the demands of customers. Agile lifecycles are determined mainly by developers who decide how long each prototyping/iteration cycle will take and what they think can be produced in that time scale. Overall the focus is on the rapid production of software with the flexibility to change/add in requirements as necessary.
		(c) Writing and following algorithms.	This objective is down to you – this is the part of the exam spec where it says you should be able to program….!
	1.2.4 Types of Programming Language	(a) Need for and characteristics of a variety of programming paradigms.	Paradigms are the concepts on which a language are built, but simply just boils down to "different ways of doing the same thing." In terms of programming it is the study of/acknowledgment that different languages may achieve the same things in different ways and there are subtle advantages/disadvantages to this. Each language has its own specific goal – Some are aimed at logic and mathematical modelling (R for example), some are designed to be as safe as possible for mission critical systems, some focus on AI such as LISP and so forth. Each tailors its syntax, key words and structure to best suit the intended application. Bottom line – recognise that whilst you *can* do something in a language, doesn't always mean you *should* or that it's the "correct" language for the job.
		(b) Procedural languages.	Procedural languages are those which are designed to create programs which are structured in small "blocks" of code called procedures or functions. A Procedure is simply a block of code, with a name, which can be "called" from anywhere within the code. They have the obvious advantage of simplicity of design and the fact that these code blocks can be reused. Procedural languages are intrinsically linked to the concept of Stacks – each time a procedure is called, the current state of the program must be pushed to the stack. There are hardware and software implications of this.
		(c) Assembly language (including following and writing simple programs with the Little Man Computer instruction set). See appendix 5e.	Assembly language consists of short three or four letter mnemonics which directly represent a machine code CPU instruction. It exists purely to make machine code programming human accessible. Instead of writing programs in pure binary, we can write in assembly language and then use an assembler to convert the code back into binary. One of the earliest/first types of programming language. LMC is a "learning language" designed to introduce you to assembly through 10 simple commands and is essential that you practise for your exam.
		(d) Modes of addressing memory (immediate, direct, indirect and indexed).	When issuing an assembly language instruction, it usually comes in the form Operator – Operand, or more simply Instruction, Data. The data in an instruction can either be: A number An address

			An offset The type of data is determined by its “addressing mode” which is represented by a symbol. Immediate: interpret as a number Direct: interpret as an address Indirect: an address which, when visited in memory, holds the address we actually want Indexed: a base address which we then add an offset to, in order to find the address we want.
		(e) Object-oriented languages (see appendix 5e for pseudocode style) with an understanding of classes, objects, methods, attributes, inheritance, encapsulation and polymorphism.	Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data and code: data in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods). There are *countless* object oriented languages, most commonly Java, C++, C# and Python

1.3 Exchanging data - How data is exchanged between different systems	1.3.1 Compression, Encryption and Hashing	(a) Lossy vs Lossless compression.	Lossy compression reduces the size of files by removing unnecessary information, or by reducing quality in a way which is either “acceptable” or likely to not be noticed by the user. In music, for example, MP3 removes frequencies that the human ear cannot resolve. This has the obvious advantage of large reductions in file size, but the disadvantage that we can never restore the original quality. Lossless compression reduces the file size without losing any quality or the ability to reproduce the original file. Common techniques include dictionary and run length encoding which reduce data by removing the need to represent repeated data explicitly.
		(b) Run length encoding and dictionary coding for lossless compression.	Run length encoding – anywhere in a file where there is repeated data (colour in an image, words or letters in a text file), store only the colour and number of pixels, or store only the letter and the number of repetitions. This reduces the amount of data stored. Dictionary encoding – commonly used to compress text. Represents each new word as a “token” in a lookup table. The document is then reduced to a collection of numbers. This reduces overall file size due to only storing each word once.
		(c) Symmetric and asymmetric encryption.	Symmetric encryption is any form of encoding in which the process to encrypt is the simply reversed in order to decrypt. This is obviously very insecure. Asymmetric encryption uses the concept of public and private keys in order to encode and decode data. It relies on an incredibly complex mathematical discovery that it is possible to create “one way” algorithms where the method to encode cannot be simply reversed in order to return to the original data. This is further

			helped by the fact that "key pairs" mean one key can be published and used to encrypt and then a separate, secret, key can be used to decrypt. This is essential for secure transmission of data.
		(d) Different uses of hashing.	Hashing uses the concept of an algorithm which, for any given input, will always produce the same output. It also has the useful property that a small change in the input will result in a large, obvious change in the output. This has several applications including: File hashing – a quick and convenient way of ensuring two copies of a file are indeed identical and have not been corrupted or tampered with. File systems – a quick method of locating a files by generating a hash which points to its location.
	1.3.2 Databases	(a) Relational database, flat file, primary key, foreign key, secondary key, entity relationship modelling, normalisation and indexing. See appendix 5g.	Databases are "an organised collection of information" which are usually manipulated using SQL. Flat file: The simplest type of database – all data is kept in one table. Has significant disadvantages such as repeated data (redundancy) and difficulty keeping data accurate/up to date Relational: A database where each entity has its own table (e.g. customer, appointment etc). Each table is linked by primary/foreign keys which helps to reduce redundancy. Primary key: A unique identifier for a record in a database table Foreign key: A primary key from another table acting as a link Normalisation: The process of taking database design and turning it into a relational design. Normalisation removes repetition, separates data into tables and ensures that there are no many to many relationships. There are 3 normal forms that we care about in the A Level Entity: An object in a database, e.g. "Vehicle", "Customer." Always singular. Entity relationship modelling: A method of displaying the relationships between tables in a database. Relationships can be 1:1, 1:Many, Many:Many. A many to many relationship breaks normalisation rules and indicates a link table must be created.
		(b) Methods of capturing, selecting, managing and exchanging data	This is down to you – same as with input and output devices, you should be able to suggest sensible methods of collecting data in a database.
		(c) Normalisation to 3NF.	1NF: A table is in First Normal Form (1NF) if there are no repeating groups. In other words, each column must contain only a single value and each row must have an item in every column. This can usually be done by putting the data into two tables ... separating the repeated data into a separate group. 2NF: To move to 2NF, any partial dependencies must be removed. This basically means each record should not have a composite primary key (the primary key should be unique in its own right). This removes many to many relationships and repeated data 3NF: 3rd Normal Form removes something called "Transitive Dependency." In plain English this means that the data in each table should only be related to the primary key. If it isn't, then it needs to be in another table.

		(d) SQL – Interpret and modify. See appendix 5e	SELECT (fields) FROM (table name) WHERE (criteria) INSERT INTO (table) VALUES (comma separated list of values) UPDATE (table) SET (field=criteria) WHERE (criteria) DELETE FROM (table) WHERE (criteria) DROP (table)
		(e) Referential integrity	Referential integrity refers to the accuracy and consistency of data within a relationship. In relationships, data is linked between two or more tables. This is achieved by having the foreign key (in the associated table) reference a primary key value (in the primary – or parent – table). Because of this, we need to ensure that data on both sides of the relationship remain intact. So, referential integrity requires that, whenever a foreign key value is used it must reference a valid, existing primary key in the parent table.
		(f) Transaction processing, ACID (Atomicity, Consistency, Isolation, Durability), record locking and redundancy.	**Atomicity:** All changes to data are performed as if they are a single operation. That is, all the changes are performed, or none of them are. For example, in an application that transfers funds from one account to another, the atomicity property ensures that, if a debit is made successfully from one account, the corresponding credit is made to the other account. **Consistency:** Data is in a consistent state when a transaction starts and when it ends. For example, in an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction. **Isolation:** The intermediate state of a transaction is invisible to other transactions. As a result, transactions that run concurrently appear to be serialized (happen one at a time in sequence). For example, in an application that transfers funds from one account to another, the isolation property ensures that another transaction sees the transferred funds in one account or the other, but not in both, nor in neither. **Durability:** After a transaction successfully completes, changes to data persist and are not undone, even in the event of a system failure. For example, in an application that transfers funds from one account to another, the durability property ensures that the changes made to each account will not be reversed.
	1.3.3 Networks	(a) Characteristics of networks and the importance of protocols and standards.	Network protocols are sets of established rules that dictate how to format, transmit and receive data so computer network devices -- from servers and routers to endpoints -- can communicate regardless of the differences in their underlying infrastructures, designs or standards. To successfully send and receive information, devices on both sides of a communication exchange must accept and follow protocol conventions. Support for network protocols can be built into software, hardware or both.

		(b) The internet structure: • The TCP/IP Stack. • DNS • Protocol layering. • LANs and WANs. • Packet and circuit switching	LAN: A Local area network, usually confined to one small geographic location (e.g. a house or office). A LAN is an internal network, all devices are usually connected using switches and use dedicated equipment, meaning bandwidth is not shared with any external device or organisation. WAN: A Wide Area Network, covers a large geographic area and usually makes use of shared telecommunications equipment meaning bandwidth is shared between multiple users. DNS: Domain Name Server/Services, a hierarchy of public servers which are responsible for converting URL's into IP addresses. Protocol Layers: Protocols are complicated and therefore are usually split into different layers. Each layer will have distinct functionality and will be self contained. A Layer expects certain specific inputs and will provide specific outputs to the next layer. The advantage of layers is that they can be implemented in hardware, software or split between both. This can have significant cost savings when developing a system, or could provide significant performance gains if implemented purely in hardware. A further advantage is that each layer can be implemented, updated or developed separately without affecting the others. Packets and Packet Switching: All data sent across a network is split into parts called packets. Packets are small, fixed size pieces of data designed to improve the reliability of data transmission. If a packet is lost on a network, the time taken to re-send it is far less significant than if an entire piece of data needs to be sent again. Furthermore, packets can be sent through different paths on a mesh network like the internet meaning they can take the most efficient/fastest path to their destination.
		(c) Network security and threats, use of firewalls, proxies and encryption. (d) Network hardware. (e) Client-server and peer to peer.	Firewall: Proxy: Encryption: Networking Hardware: Client Server Peer to peer
	1.3.4 Web Technologies	(a) HTML, CSS and JavaScript. See appendix 5e.	See appendix 5 for details of the level of HTML, CSS and Javascript you need to know. I recommend the W3C Schools tutorials for your revision here.
		(b) Search engine indexing.	In order to allow you to search the WWW, a search engine must first know about the web pages that exist, what they are about and have some way of categorising those pages. The process of finding, exploring and storing data regarding web pages is called "indexing." To index pages, a search engine will use a program called a "crawler." A crawler is simply a program which visits websites, finds all the hyperlinks on that page and then visits those. It is obvious that this process rapidly grows in complexity and time requirements. As it visits each page it will store information it finds (meta data) in the index.

		(c) PageRank algorithm	Page rank is specific to Google. The initial idea came from a simple principle – The more times a website is linked to, the more important it must be. Strangely, this concept had not been explored before Larry Page and Sergey Brin attempted it, and it obviously turned out quite well. The principle was expanded to include adding weighting to the quality of website that was linking to a site and other factors that could affect its "reputation." All of these factors are boiled down to return an indication of where each page should rank when searched for.
		(d) Server and client side processing	Nearly all modern web pages and applications rely on some form of processing to generate dynamic or responsive content. There are two choices for this processing – let it take place on the client machine or do everything on the server. Server side processing has the advantage that it is more secure and less prone to tampering. However, there is the obvious hardware demands/requirements that come with this. The more clients there are, the bigger the demand on the server will be. Server side processing is essential for websites that include things like shopping carts, payments, profiles and so forth. Client side processing is suitable for things like graphical effects, automatic updating of content and interactive features that don't rely on the retrieval of data (simple games for example). Client side processing has the advantage of removing the demand on the web server, however, because there is no guarantee of the type of device a client is running, then the code may run slowly, give variable performance or not even run at all due to incompatibilities (such as wrong browser or script blocking)

1.4 Data types, data structures and algorithms - How data is represented and stored within different structures. Different algorithms that can be applied to these structures	1.4.1 Data Types	(a) Primitive data types, integer, real/floating point, character, string and Boolean.	Data types have a significant effect on the compilation of a program. Different data types require different amounts of memory, methods of processing and so forth. Incorrect data type, e.g. integer when you need real, can cause significant logic errors in a program. Integer = a whole number Real = a number with fractional parts Character = a single letter, number or symbol String = zero or more characters Boolean = true or false
		(b) Represent positive integers in binary.	See the notes hand out in the lesson folder for methods
		(c) Use of sign and magnitude and two's complement to represent negative numbers in binary.	See the notes hand out in the lesson folder for methods
		(d) Addition and subtraction of binary integers.	See the notes hand out in the lesson folder for methods
		(e) Represent positive integers in hexadecimal.	See the notes hand out in the lesson folder for methods

		(f) Convert positive integers between binary hexadecimal and denary.	See the notes hand out in the lesson folder for methods
		(g) Representation and normalisation of floating point numbers in binary.	See the notes hand out in the lesson folder for methods
		(h) Floating point arithmetic, positive and negative numbers, addition and subtraction.	See the notes hand out in the lesson folder for methods
		(i) Bitwise manipulation and masks: shifts, combining with AND, OR, and XOR.	Shifting allows us to quickly perform integer division and multiplication, logical shifts ignore all sign bits and simply insert zeroes as necessary, arithmetic shifts preserve the sign bit. Masking is simply a method of applying a Boolean expression, usually to a register, in order to explore the state of individual bits, rather than changing the entire value stored. AND masks are used to check whether a bit is on or off, OR is used to set a bit and XOR masks are often used as a form of encoding/encryption.
		(j) How character sets (ASCII and UNICODE) are used to represent text.	Text in a computer is represented as numerical data. There are two standards – ASCII and UNICODE, and both do the same thing, which is to assign a unique numeric value to each character that the computer is to represent or store. ASCII is actually now part of the UNICODE standard. ASCII is traditionally a 7 bit standard, which can also be represented in 8 bits per character. Obviously, the more bits per character a standards has, the more symbols it can represent, however there is the storage implication meaning it will take up more space to store each character. ASCII is limited to 127 characters, whereas UNICODE (depending on which version is used) is a 16 bit standard and can store every printable character in the known world. Even languages that are “extinct” are available in Unicode.

			These standards are essential, as without them, computers and devices from different manufacturers could not easily communicate.
	1.4.2 Data Structures	(a) Arrays (of up to 3 dimensions), records, lists, tuples.	Arrays: A data structure which acts like a table. Each "row" is accessed via an indexing system, usually zero based. Arrays are **fixed** size and hold multiple values of the same data type and cannot hold values of differing types. Record: A custom data structure which holds values of differing data types. For example a record "person" may contain fields such as "name", "age", "height" and so forth. Variables can then be declared as type "person" and the fields accessed through the object.property method Lists: A dynamic data structure which can grow and shrink with the requirements of the program. Each node in a list holds a piece of data and a pointer to the next piece of data in the list. Tuples: A strange data type which holds a collection of data, usually ordered and it cannot be changed. Accessed in much the same way as an array.
		(b) The following structures to store data: linked-list, graph (directed and undirected), stack, queue, tree, binary search tree, hash table.	Linked list: a data structure consisting of nodes. Each node is a pair of "data" and "link" entities. The link points to the next node in the list. By manipulating the links we can grow, shrink, re-arrange and otherwise manipulate the data stored within. Dynamic. Graph: A data structure similar in nature to a linked list in that it consists of connected nodes. However, in a graph each node can be connected to multiple other nodes. Stack: A FILO (First in, last out) data structure which is used in countless computing applications. Data is either "pushed" on to the stack or "popped" off it. Elements cannot be accessed at random and must be stored and retrieved one at a time. Often used in programming (call stacks), operating systems, interrupts and so forth. Queue: A First in, First Out (FIFO) data structure. Elements in the queue are removed in the order they were added. Tree: A data structure consisting of nodes. Each node contains data and a number of pointers. The tree is organised in a "root" and "leaf" structure. There is one root node at the top of the tree, this may have zero or more "leaf" nodes coming off it. In turn, those leaf nodes may be connected to layers below. The only difference between a tree and binary tree is that a binary tree node may only have a maximum of two child nodes. Hash Table: Basically a table of pointers to data. Each row of the table is accessed by hashing the input to discover the row to be accessed.

		(c) How to create, traverse, add data to and remove data from the data structures mentioned above. (NB this can be either using arrays and procedural programming or an object-oriented approach).	See lesson.
	1.4.3 Boolean Algebra	(a) Define problems using Boolean logic. See appendix 5e.	See lesson notes
		(b) Manipulate Boolean expressions, including the use of Karnaugh maps to simplify Boolean expressions.	See lesson notes
		(c) Use the following rules to derive or simplify statements in Boolean algebra: De Morgan's Laws, distribution, association, commutation, double negation.	See lesson notes

		(d) Using logic gate diagrams and truth tables. See appendix 5e. (e) The logic associated with D type flip flops, half and full adders.	See lesson notes.

1.5 Legal, moral, cultural and ethical issues - The individual moral, social, ethical and cultural opportunities and risks of digital technology. Legislation surrounding the use of computers and ethical issues that can or may in the future arise from the use of computers	1.5.1 Computing related legislation	(a) The Data Protection Act 1998.	See https://www.gov.uk/data-protection
		(b) The Computer Misuse Act 1990	See https://www.bbc.co.uk/bitesize/guides/z8m36yc/revision/5
		(c) The Copyright Design and Patents Act 1988.	See https://www.york.ac.uk/records-management/copyright/law/
		(d) The Regulation of Investigatory Powers Act 2000.	See https://www.bbc.co.uk/bitesize/guides/zpjm6sg/revision/1
	1.5.2 Moral and ethical Issues	The individual moral, social, ethical and cultural opportunities and risks of digital technology: • Computers in the workforce. • Automated decision making. • Artificial intelligence. • Environmental effects. • Censorship and the	This section will almost exclusively lead to long answer questions in the exam. To be able to answer any of these questions you need an understanding of what is going on in the wider world of technology. This involves… reading! There's no two ways around it. There are issues we will discuss in class, but there is no substitute for you actively engaging in this aspect of your learning. Read through our school twitter feed for a number of links to get you going. In summary, you need to be aware of how technology affects the wider world than merely consumers and the companies that provide them. The list of topics in the specification give you a clear picture of the things you need to pay particular attention to and each one has clear, real world examples.

		Internet. • Monitor behaviour. • Analyse personal information. • Piracy and offensive communications. • Layout, colour paradigms and character sets.	

OCR A Level Computer Science – Glossary of Terms – Unit 2

Strand	Sub Strand	Criteria	Meaning
2.1 Elements of computational thinking - Understand what is meant by computational thinking	2.1.1 Thinking abstractly	(a) The nature of abstraction.	Abstraction is the removal of unnecessary detail from a problem. Abstraction can also be the hiding of complexity from a user in order to make a system easier to use, for example, a GUI is an example of abstraction because it is intuitive and easy to use, yet hides the underlying complexity of what is really happening when a user performs an action
		(b) The need for abstraction.	Abstraction is necessary in algorithm design as it helps us to focus solely on the problem, it prevents the objectives becoming too broad. In HCI, abstraction helps us to ensure the system is as simple and easy to use as possible for the end user.
		(c) The differences between an abstraction and reality.	Abstract concepts may not fully represent their real world counterparts. Attempting to mimic real world objects is called "skeuomorphism." for example a program in which we take notes may look like actual lined notepaper. An abstracted version of the same program would remove this detail as it is unnecessary to solve the problem/perform the task it is designed to do.
	2.1.2 Thinking ahead	(a) Input and output	Input = data going IN to a computer system or program Output = data coming OUT of a computer system of program
		(b) Pre-Conditions	Criteria which must be met before something can happen. In terms of thinking ahead, pre-conditions are the things you must consider before designing a program or block of code – what inputs do you need, what data do you need from external sources, what actions must/will the user perform etc.
		(c) Caching	Caching (not in the CPU sense of the word Cache) is when small amounts of data are kept in memory or in a buffer so that they are available to a program without delay. Caching data not only speeds up program execution but can also be used to smooth out variability in things like network connections (caching a video before playing it, for example, would hide any dips in network performance as there would be enough data cached to continue playback without interruption whilst the network performance increases again)
		(d) Reusable program components.	These are functions, procedures, classes, libraries or DLL's. These are blocks of pre-written, pre-tested code which we can use in our programs to reduce the time taken to develop software. It has the distinct advantage that the code should be robust and reliable.
		(d) sub-procedures/functions	Sub procedures and functions are used to break up complex problems into smaller component parts. Each function or procedure will perform a specific task and have a set of inputs and expected outputs. By using these components it makes programs easier to design, reduces repetition, increases the reliability and robustness of code and enables us to delegate parts of programs to different developers who can work on their own blocks of code independently.
	2.1.5 Thinking concurrently	(a) Concurrent Processing	Concurrent processing usually makes use of "threads" which is where specific tasks may be carried out in parallel to other tasks within a program. This usually speeds up execution as a programmer can design code which splits complex tasks into threads that perform tasks simultaneously. Concurrent processing should also mean that a program rarely enters a wait state where it is doing nothing, waiting for an event to occur such as user input.

2.2 Problem solving and programming How computers can be used to solve problems and programs can be written to solve them (Learners will benefit from being able to program in a procedural/imperative language and object oriented language.)	2.2.1 Programming techniques	(a) Programming constructs: sequence, iteration, branching	Sequence – a set of instructions executed in order Iteration – looping or repeated execution of the same code Branching – use of decisions or IF statements to change the flow of execution depending on the evaluation of a rule.
		(b) Recursion.	The act of a procedure or function calling itself. A seemingly complex concept whereby a problem can be defined in terms of itself. Classic examples include factorial and Fibonacci sequences. Recursive code must have a “base case” which, when hit, will result in the “call stack” unwinding itself, resolving all the previous function calls to produce a final answer.
		(c) Global and local variables	Global variables are those which are available to any procedure or function, anywhere in code. Generally considered a bad thing as if they are changed unexpectedly somewhere in the code it may not be obvious that this has happened, or where, making debugging and code maintenance difficult. Local variables only have “scope” in the procedure or function in which they are declared. This means once a procedure or function terminates, these variables no longer exist. They are the preferred method of using variables as it is easy to keep track of where a variable exists/belongs and how it is being accessed.
		(d) parameter passing by value and by reference.	Functions and procedures rely on data being sent to them as inputs. These inputs are called parameters and take the form of variables that are sent to them when called. Parameters may be passed by value – a copy of the variable is sent to the function and discarded after use, or by reference – the variable is sent as a pointer to the data and any changes are persistent.
		(e) Use of an IDE to develop/debug a program.	IDE’s are a set of tools which allow programmers to edit, create, debug and publish program code. Features usually include a code editor, syntax highlighting, code completion, break points, variable watch, error diagnostics, a translator/interpreter and a compiler.
		(f) object oriented techniques.	OO is a technique used to write programs which attempt to mimic real life “objects.” Each object is a “class” which is used as a template. When an object is “instantiated” then a copy of the class is created which can be accessed (roughly) like an “intelligent” variable.

			A class will specify several methods (things it can do) which are used to perform actions or manipulate the contents of variables. A class will also posses “properties” which are data and variables which describe the object. OO has the advantage that classes should be completely reusable across different projects and they should be “safe” to use because of “encapsulation” which means variables and data are “private” to a class and can only be accessed or manipulated by calling class methods.
	2.2.2 Computational methods	(a) Features that make a problem solvable by computational methods.	Computational methods = maths. Features that make a problem solvable by computational methods = can you turn your problem into a mathematical form? In other words, there are certain problems which are well suited to computing – such as those which involve massive data sets, calculations, analysis of data, repetitive tasks, complex maths/physics simulations. There are others which are less trivial/suitable in nature and you should be able to recognise these (image recognition for example)
		(b) Problem recognition.	Being able to extract the parts of a problem which need to be solved or could be solved with code.
		(c) Problem decomposition.	Being able to break a problem down into a set of sub-tasks or problems which themselves could be easily turned into code (in other words they are obviously functions, procedures or classes)
		(d) divide and conquer.	A method of problem solving by splitting the problem into ever smaller tasks until they become trivially easy to solve.
		(e) abstraction	As before.
		backtracking	A technique used to “retrace the steps” an algorithm has taken. Backtracking has many applications, usually in well known algorithms like Dijkstra’s algorithm and other path finding solutions. Backtracking means a path has been followed, evaluated and found to be sub-optimal, the algorithm then goes back to a previously known good point and continues from there in an attempt to evaluate a better path.
		data mining	The process of taking “big data” (extremely large data sets) and creating queries which elicit useful data/conclusions. Data mining recognises that a data set may be more useful than its initial intended purpose. Techniques may include uses of AI, Neural Networks or other machine learning techniques.

		heuristics	The use of a “best fit” approach to the solution of a problem, or the use of “best guess.” Heuristics are used to solve problems which are NP problems (in other words we cannot prove them to be solvable and therefore there exists no known “complete” solution). Heuristics may be used in other applications such as anti virus software to monitor “virus like” behaviour and best guess that a program may or may not be malicious.
		visualisation to solve problems	As it says on the tin – the use of diagrams to simplify the finding of solutions to problems.
		pipelining	The ordering of instructions into the most efficient form so that program execution speed is maximised. Compilers and CPU’s will both have pipelining optimisations which enable them to re-order instructions into a more efficient or “likely” form. The processor can then maximise its execution speed by simultaneously executing one instruction, whilst decoding the next, whilst fetching the next + 1 instruction.
		performance modelling.	The use of big O notation to investigate/represent the best and worst case time and space complexity of problems.

2.3 Algorithms - The use of algorithms to describe problems and standard algorithms	2.3.1 Algorithms	Big O notation	A mathematical representation of the best or worst case time and space complexity of a problem. Translated – how long will this take to execute and how much memory will it require?
		Constant	The time taken will be the same regardless of the size of the problem/data set.
		linear	The time/space requirements will grow directly in proportion with the size of the data set.
		polynomial	The time/space requirements will grow as some form of function of n (the size of the data set)
		Exponential	The time/space requirements will grow at an exponential rate as the data set grows (as the data set increases, time/space doubles for each increase of n)
		logarithmic	Usually a best case scenario, the time/space requirements will grow rapidly at first as data set size increased, however this will then “level out” and increasing the size of a data set will have less and less of an effect.
		Stacks – algorithm	You need to be able to recognise/create the code which will implement a stack. Stacks are LIFO data structures which can only be accessed from the top by “Popping” data off the top of the stack.
		Queues - algorithm	You need to be able to recognise/create the code which will implement a queue using either an array or a list. A queue is a FIFO data structure which can only be accessed at the “head” and “tail.”
		Tree	See Unit 1 notes.
		Linked list	See Unit 1 notes.
		Depth first traversal	A method of searching a tree data structure by following the left most nodes until the bottom of the tree is reached and then backtracking up to each level and traversing the right hand nodes.
		Breadth first traversal	A method of searching a tree by systematically accessing all nodes on the same level before then visiting nodes of a lower level.
		bubble sort	A method of sorting data by comparing pairs of numbers in a list and swapping when one is bigger than the other. Continues until no swaps are made. Computationally very expensive due to the number of iterations and comparisons made, grows exponentially in complexity/time requirements
		Insertion sort	A method of sorting data by creating two lists – sorted and unsorted. Each iteration sees one value taken from the unsorted list and then “inserted” in the correct place using one of the following rules: If the element is smaller than the first element of the sorted list – place at the start

			If the element is bigger, then compare to the next element in the list. If bigger, move to the next element, if smaller, place in the list before this element. If the end of the list is reached, place at the end.
		Merge sort	Splits numbers to be sorted into lists and then merges pairs of lists: Take the first number from each list. Compare. Place the smallest in the new list. Repeat until both lists are empty.
		Quick sort	A “divide and conquer” algorithm which splits a data set up into two arrays around a “pivot” in the data set. All numbers bigger than the pivot are stored in one array, all smaller ones in the other array. This method is applied recursively, meaning it splits the data until each array holds only one element and then winds back up the call stack, placing elements in the correct positions.
		Dijkstra’s algorithm	A path finding algorithm using a graph structure which evaluates all possible paths in order to find the shortest route to a destination. Best understood with a demonstration…
		A* path finding	A method of path finding often used in gaming and can be seen as an extension to dijkstra’s algorithm. Used to find the shortest sensible path to a destination. Uses a form of “best first” search to try and find the “least expensive” route to a destination. Again, best understood visually/using an example.
		Binary search	A method of searching a data set, which **must** be sorted first. Finds the median value and compares to the target. If they are equal then the value is found. If not, the median is compared to the target again. If the target is larger than the median, then all smaller numbers are discarded. If the target is smaller than the median then all larger numbers are discarded. The algorithm then repeats until a match is found. Can be implemented recursively. Has logarithmic complexity, meaning it is extremely good for large data sets.
		Linear search	The most trivial of searching algorithms, each element in a list is systematically compared to a target until it is found or the list exhausted. Has the advantage that the list does not have to be sorted, has the obvious disadvantage that it generally slows down as the size of the data set increases.