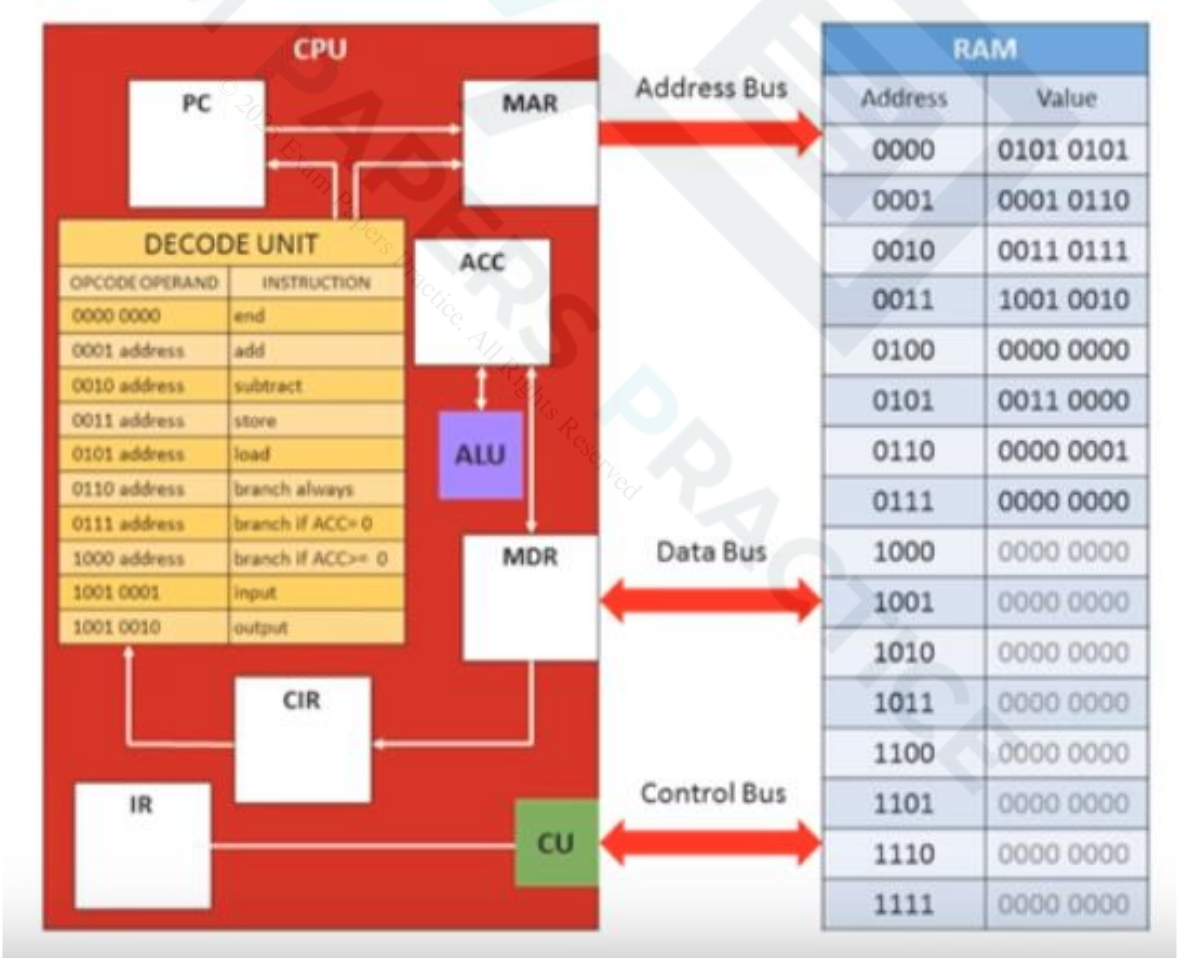
OCR A level Computer Science notes

The characteristics of contemporary processors, input, output and storage devices

Spec reference:

(a) The Arithmetic and Logic Unit; ALU, Control Unit and Registers (Program Counter; PC, Accumulator; ACC, Memory Address Register; MAR, Memory Data Register; MDR, Current Instruction Register; CIR). Buses: data, address and control: how this relates to assembly language programs.



DECODE UNIT

OPCODE OPERAND	INSTRUCTION
0000 0000	end
0001 address	add
0010 address	subtract
0011 address	store
0101 address	load
0110 address	branch always
0111 address	branch if ACC= 0
1000 address	branch if ACC>= 0
1001 0001	input
1001 0010	output

Address	Value
0000	0101 0101
0001	0001 0110
0010	0011 0111
0011	1001 0010
0100	0000 0000
0101	0011 0000
0110	0000 0001
0111	0000 0000
1000	0000 0000
1001	0000 0000
1010	0000 0000
1011	0000 0000
1100	0000 0000
1101	0000 0000
1110	0000 0000
1111	0000 0000

What does the ALU do? It's in the name, it does arithmetic and logical operations

What does the CU do? **It sends control signals to coordinate the movement of data around the processor**

What does the program counter do? **It stores the address of the next instruction to be processed**

What does the accumulator or ACC do? **It stores the results of the processing**

What does the MAR do? **It stores the address of the data or instruction to be fetched from memory**

What does the MDR do? **It stores data from memory and acts as a buffer**

What does the CIR do? **Stores the instruction currently being executed sent by the MAR**

What does the address bus do? **Sends the address of the data or instruction from the MAR in the processor to memory**

What does the control bus do? **Sends control signals from the CU to the components of the computer**

What does the data bus do? **Sends data between the processor and the memory**

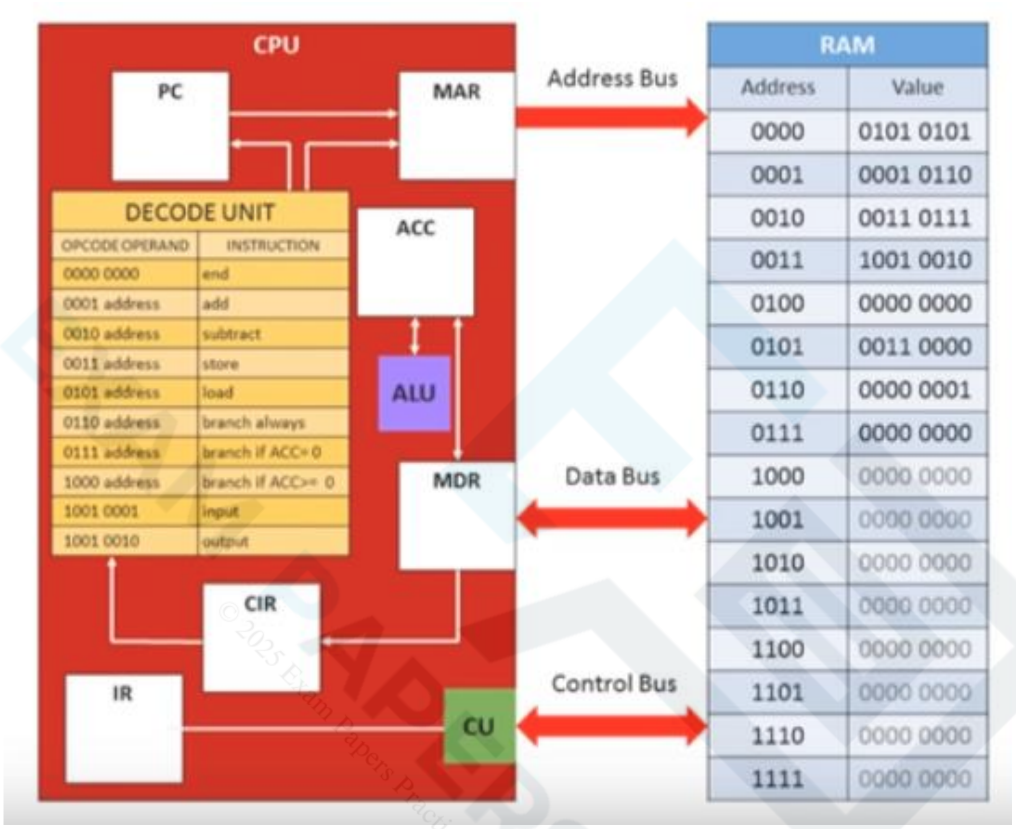
Right you looked at this, now you need to learn about something called program branch.

Program branch is when there is a branch instruction in the CIR so the contents of the PC are changed to the contents of the operand in the CIR (The CIR is split into the operand and opcode and you will learn it in the next page)

Spec reference:

(b) The Fetch-Decode-Execute Cycle; including its effects on registers.

Again let's go back to our good old diagram



DECODE UNIT

OPCODE OPERAND	INSTRUCTION
0000 0000	end
0001 address	add
0010 address	subtract
0011 address	store
0101 address	load
0110 address	branch always
0111 address	branch if ACC= 0
1000 address	branch if ACC>= 0
1001 0001	input
1001 0010	output

RAM

Address	Value
0000	0101 0101
0001	0001 0110
0010	0011 0111
0011	1001 0010
0100	0000 0000
0101	0011 0000
0110	0000 0001
0111	0000 0000
1000	0000 0000
1001	0000 0000
1010	0000 0000
1011	0000 0000
1100	0000 0000
1101	0000 0000
1110	0000 0000
1111	0000 0000

What happens in the Fetch Execute Cycle and its effects on registers will be broken down into steps, read these carefully and look at the diagram to make sure you know what is being said

1. The PC contains the address of the next instruction to be processed, the contents of the PC(Which is basically the address of the next instruction) are copied to the MAR.
2. The MAR sends the address of the instruction to be fetched from memory via the data bus to the memory where it waits for a control signal from the CU

3. The CU sends a control signal through the control bus into the memory to do a memory read in order to fetch the instruction from the address given by the MAR
4. *You got it so far? If you are lost read the first three steps again as the next ones WILL confuse you if you don't understand the first three*
5. The instruction is fetched and the data bus carries it to the CIR, not the MDR as it's not data.
6. The instruction in the CIR is split into two parts, opcode which are the commands and the operand which is the actual data or address to be used
7. The CIR sends the address to the MAR which sends the address of the **data**(The reason why it's not an instruction anymore is because the CIR now needs the data as it already has the instruction) to be fetched from the memory
8. At this point the PC increments
9. Step three is repeated except it's not instruction, it's data now
10. *Still alive? If not read through the previous points again and do not read them in a rush, take your time*
11. The data is fetched from memory and sent to the MDR (Since it's data this time not an instruction therefore it goes to the MDR not the CIR) via the data bus
12. The contents of the MDR are then copied into the ACC

Well moving on we will be looking at some factors affecting CPU performance. Yay.....or not as you are going back to GCSE

Spec reference:

(c) The factors affecting the performance of the CPU: clock speed, number of cores, cache.

Factors affecting CPU performance:

Clock speed: The higher the clock speed, the more cycles can be done per second. However, this can cause more heat to be produced from the computer therefore a cooling fan needs to be purchased and this costs money

Cache: Memory located closer to the CPU than RAM, it holds instructions and data so that the CPU does not have to do a memory re-read to re-fetch instructions or data

Number of cores: The higher the number of cores, the more instructions can be processed at the same time per second

Spec reference:

(d) The use of pipelining in a processor to improve efficiency.

No we are not talking about fixing pipes

What does pipelining mean? Is when the instruction is fetched whilst the previous one is being decoded and the one before that is being executed

What are some of the problems with pipelining?

- Only works if the subsequent instructions can be predicted
- The pipe may flush, meaning if the wrong instruction is fetched it will have to be thrown away and the correct one has to be fetched whilst the previous instruction is being decoded

Spec reference:

(e) Von Neumann, Harvard and contemporary processor architecture.

In this section we will be talking about what a Hungarian dude did and what a British or American I dunno what the other guy's nationality is.

What are the features of Von Neumann?

- Instructions and data are stored together in the same memory
- Instructions and data are stored in the same format
- Instructions are executed sequentially

What are the features of Harvard?

- Instructions and data are stored in separate memories
- The instructions and data each have their own bus
- Used by RISC processors

Now on to some contemporary architectures.

SIMD (Single Instruction Multiple Data) - As the name suggests this is when a processor processes the same instruction on multiple data sites.

Example of where this might be used? Loading computer graphics on a video game

MIMD (Multiple Instruction Multiple Data) - Again just look at the name, this is when a processor processes multiple instructions on multiple data sites

Distributed computing/system - This is when multiple computers work together to process a single instruction

Spec reference:

(a) The differences between and uses of CISC and RISC processors

What are some of the uses of CISC processor?

Well to put it simply, they are used in laptops and desktop computers

What are some of the uses of the RISC processor?

Again to put it simply, they are used in smartphones and tablets

CISC vs RISC

CISC	RISC
Has more complex hardware	Has simpler hardware
It's more expensive	It's more cheaper
Takes multiple machine cycles to complete a single instruction	Takes just one machine cycle to complete one instruction
Has greater energy consumption	Has lower energy consumption
Processes more complex instructions (Hence the name)	Processes more simpler instructions
Can't support pipelining	Can support pipelining

Put it like this yeah, just think of the CISC as the bad processor and the RISC as the good one, why? Well if you look at the table RISC is better in every aspect compared to the CISC EXCEPT the part of where the CISC can process complex instructions and RISC can't

Spec reference:

(b) GPUs and their uses (including those not related to graphics).

How do GPUs compare to CPUs? They run slower and have fewer features than CPUs

What do GPUs do? They excel at doing simple operations on large sets of data. They are also very efficient (Posh way of saying good) at manipulating computer graphics and image processing

Where are they used? They are used in computers, smartphones and game consoles

Spec reference:

(c) Multicore and Parallel systems.

Multicore system - These are processors with two or more cores, each core is capable of processing one instruction and they all process instructions at the same time. Keep in mind this DOES NOT mean faster speed of processing

Parallel system – This is a processor that splits the job of processing an instruction between multiple cores, this results in faster processing speed

Spec reference:

(a) How different input, output and storage devices can be applied to the solution of different problems.

In the exam you will be given a scenario and will be expected to use your brain in deciding which is the best input/output/storage device for that situation often justifying your reasoning.

What is an input device? It is a device which allows you to input data into the system

What is an output device? It is a device which displays the results of the processing to the user

What is a storage device? A peripheral device which stores data for long term and short-term storage

<u>Input devices:</u>

- Keyboard:

  Nice and easy, allows you to input data into the computer.

- Mouse:

  Again nice and easy one, allows you to input data into the computer.

- Microphone:

  Allow you to input your voice into the computer.

- Sensors:

These are devices which detect changes in the environment like changes in temperature, pressure, position, acceleration etc... and send it to a computer.

Examples of sensor devices would be a thermometer and a measuring balance

- Keypad:

Allow you to type in data into a security door, ATM etc...

- Specialist keyboard:

Like the name suggests, it is a keyboard specialised to a person's needs. An example would be a braille keyboard which allows a person to feel the characters they are typing in

A really weird keyboard indeed, wonder who uses that....

- Touchpads:

Again same as keyboard except it allow you to see the results of the processing as well so it can be used as a output device too

- Remote controls:

Same as keyboard but is wireless so it can be used to input data from a distance without the need to connect anything

- Scanner:

Scans images, printed texts, handwriting or objects and converts them into a digital image

This is a scanner

- Camera/webcams:

Used to input moving motions into a computer.

- Touch screens:

Do I even need to say?

- Barcode reader:

Like the name suggests, they are used to read barcodes

A barcode scanner in a supermarket

- Magnetic Stripe Reader:

A device which reads information enclosed in magnetic strips found at the back of special cards such as bank cards

magnetic stripe
CreditCard
3456 789
YOUR BANK
back side

Output devices:

- Monitor:

Displays the results of the processing to the user infront of their eyes (Well you don't actually need to say this but you get what I mean)

- Printer:

Outputs physical copies of digital documents and texts

- Speakers:

Outputs sound

- Headphones:

Mini speakers that you can put in your ears basically

- Digital projector:

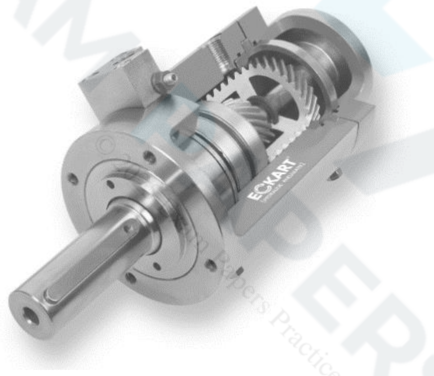
Displays an enlarged image on a screen, they are usually found in 21st century schools and theatres

- Plotter:

A printer used to print out graphics

- Actuators:

A device which moves a system or mechanism



Spec reference:

(b) The uses of magnetic, flash and optical storage devices

Magnetic storage

Advantages of magnetic storage:

- It’s cheap
- Has a large storage space

Disadvantages of magnetic storage:

- Has slow access times
- It’s fragile

Examples of magnetic storage:

- Hard drives

<u>Optical storage</u>

Advantages of optical storage:

- Cheap
- Light in weight
- Very portable

Disadvantages of optical storage:

- Has slow access times
- Can easily be damaged by scratch

Examples of optical storage devices:

- CD-R

- DVD-R (Same as CD but can hold more data and is an excellent choice to store videos, movies and similar)
- Blue-ray (Same as DVD but can hold more data)

Solid state storage

Advantages of solid state storage:

- Very resilient
- Has fast access times

Disadvantages of solid state storage:

- Expensive
- Has a limited lifespan

Examples of solid state storage:

- USB stick
- SD cards

Spec reference:

(c) RAM and ROM.

RAM is a type of memory that is volatile. It stores the operating system, programs and data that are all currently being used. It allows the program to read and write to its contents.

ROM however is non-volatile. It only stores the program which is needed to boot up the computer when it starts. It allows the computer to read into its contents only

Spec reference:

(d) Virtual storage.

Virtual storage is also known as cloud storage.

Advantages:

- Can be accessed from anywhere there is a internet connection
- The amount of space that can be added to it it's unlimited
- It can be easily shared

Disadvantages:

- It's expensive if more memory space is needed
- Cannot access it from a place where there is no internet connection

Typical example of virtual storage would be One Drive

And that's it! Well done you have come to the end of the first section of A level Computing which is the characteristics of contemporary processors, input, output and storage devices. Now take a break *before you begin to start learning the next section as it will be WAY longer than what you have just learnt.*

Software and Development

Spec reference:

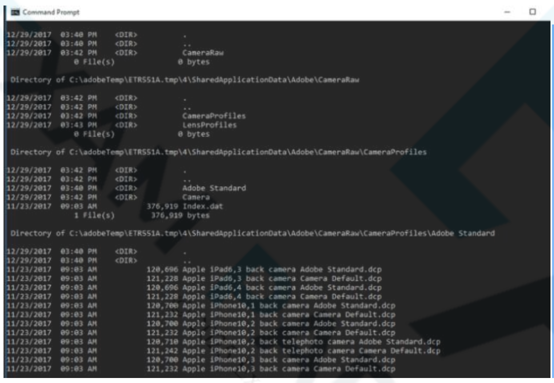
(a) The need for, function and purpose of operating systems.

There are many reasons why operating systems are needed, here are a few examples of why:

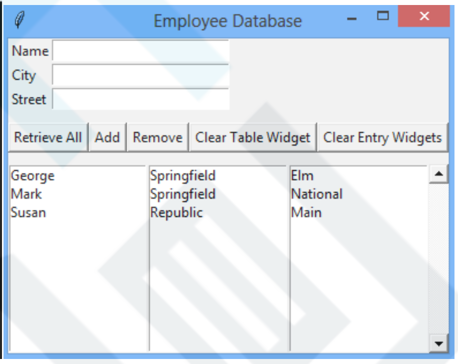
- To manage the memory from processes such as segmentation and pagination
- To provide security in order to restrict unauthorised access to user files and computer resources

- Provides networking through protocols between different computers
- Provides a platform for software and utility programs to run (By utility programs I mean programs which allow you to do useful things like word etc..)

- Provides a user interface to allow the user to communicate with applications, through menu bars etc…for example



```
Command Prompt

12/29/2017  03:40 PM    <DIR>          .
12/29/2017  03:40 PM    <DIR>          ..
12/29/2017  03:42 PM    <DIR>          CameraRaw
               0 File(s)              0 bytes

 Directory of C:\adobeTemp\ETR551A.tmp\4\SharedApplicationData\Adobe\CameraRaw

12/29/2017  03:42 PM    <DIR>          .
12/29/2017  03:42 PM    <DIR>          ..
12/29/2017  03:42 PM    <DIR>          CameraProfiles
12/29/2017  03:43 PM    <DIR>          LensProfiles
               0 File(s)              0 bytes

 Directory of C:\adobeTemp\ETR551A.tmp\4\SharedApplicationData\Adobe\CameraRaw\CameraProfiles

12/29/2017  03:42 PM    <DIR>          .
12/29/2017  03:42 PM    <DIR>          ..
12/29/2017  03:40 PM    <DIR>          Adobe Standard
12/29/2017  03:42 PM    <DIR>          Camera
11/23/2017  09:03 AM           376,919 Index.dat
               1 File(s)        376,919 bytes

 Directory of C:\adobeTemp\ETR551A.tmp\4\SharedApplicationData\Adobe\CameraRaw\CameraProfiles\Adobe Standard

12/29/2017  03:40 PM    <DIR>          .
12/29/2017  03:40 PM    <DIR>          ..
11/23/2017  09:03 AM           120,696 Apple iPad6,3 back camera Adobe Standard.dcp
11/23/2017  09:03 AM           121,228 Apple iPad6,3 back camera Camera Default.dcp
11/23/2017  09:03 AM           120,696 Apple iPad6,4 back camera Adobe Standard.dcp
11/23/2017  09:03 AM           121,228 Apple iPad6,4 back camera Camera Default.dcp
11/23/2017  09:03 AM           120,700 Apple iPhone10,1 back camera Adobe Standard.dcp
11/23/2017  09:03 AM           121,232 Apple iPhone10,1 back camera Camera Default.dcp
11/23/2017  09:03 AM           120,700 Apple iPhone10,2 back camera Adobe Standard.dcp
11/23/2017  09:03 AM           121,232 Apple iPhone10,2 back camera Camera Default.dcp
11/23/2017  09:03 AM           120,710 Apple iPhone10,2 back telephoto camera Adobe Standard.dcp
11/23/2017  09:03 AM           121,242 Apple iPhone10,2 back telephoto camera Camera Default.dcp
11/23/2017  09:03 AM           120,700 Apple iPhone10,3 back camera Adobe Standard.dcp
11/23/2017  09:03 AM           121,232 Apple iPhone10,3 back camera Camera Default.dcp
```



Employee Database

Name

City

Street

George	Springfield	Elm
Mark	Springfield	National
Susan	Republic	Main

Command line and GUI are some of the most typical examples for user interfaces

- Managing the CPU so it knows what instructions to execute next and handling interrupts

- Manages peripherals, this is done by device drivers translating instructions from the OS to what is understood by specific peripherals

Spec reference:

(b) Memory Management (paging, segmentation and virtual memory).

Paging – This is when memory is split into equal sizes and contains different sections from all over the memory (Basically different contents from the memory, no matter where they came from, where they belong etc..). It stores them as physical divisions

Segmentation – This is when memory is split into unequal sizes and contains whole sections of the program. It stores them as logical divisions

Virtual memory – This is when unused data in <u>the forms of pages or segments are moved from memory to the virtual memory in the hard disk which is part of the secondary storage</u> for long term retention. (Meaning it will be taken back from the hard disk to be used again in memory) in order to free up space in the memory for other programs to use.

What problems may there be with virtual memory? Too much time may be spent moving the pages/segments between the hard disk and memory than actually executing them.

Is there a name for this problem? You bet! It's called disk thrashing

Spec reference:

(c) Interrupts, the role of interrupts and Interrupt Service Routines (ISR), role within the Fetch-Decode-Execute Cycle.

What are interrupts and what they do? They are signals which force the CPU to stop executing the current instruction and process them instead if they have a higher priority

How are interrupts generated? They are generated from a faulty hardware or software and are sent to the Interrupt Service Routine (Which produces the interrupt)

How do interrupts work? Well if the interrupts have a higher priority than what is currently being executed, the address of the next instruction which is currently being executed is replaced in the PC (If you are confused with this part go back to the very first spec reference and learn about how the contents of the CPU change in that cycle, then it will make much more sense) with the address of the instruction stored in the interrupt. So the address of the previous instruction which was being processed is put into a stack and the interrupt is processed. Once it has finished processing it, the CPU goes back to the stack to finish processing the instruction before it was interrupted

There are different types of interrupts, you need to know three of them:

- Hardware interrupts – There are the interrupts like a power button being pressed or the memory being corrupted
- Software interrupts – These are the interrupts like illegal instructions being encountered (Trojan horse attack and all that), arithmetic overflow and new log-on request
- Input/output interrupts – The buffer being nearly empty or the completion of data transfer to a peripheral (Printer etc..)

Spec reference:

(d) Scheduling: round robin, first come first served, multi-level feedback queues, shortest job first and shortest remaining time.

The purpose of scheduling is as follows:

- To give fair processing times to all jobs
- To make the most efficient use of the CPU
- To process as many jobs as possible in the least amount of time possible
- To handle data input with a guaranteed time so applications don't fail to run

There are different scheduling methods, you only need to know the ones below:

Round Robin – This is where all jobs are given a set of CPU time to be processed, if the job is not processed by then it will be put at the back of the queue

First come First serve – It's in the name, jobs are processed in the order in which they are received

Multi-level feedback queues – This is where jobs are split into different queues with each queue having a higher priority than others, the CPU may move jobs between different queues if their priorities change

Shortest job first – The job that takes the least amount of time to be processed is processed first

Shortest remaining time – Same as shortest job first except in the exam you will have to word it a bit differently, this is when the job that has the shortest amount of processing time remaining is processed first

Spec reference:

(e) Distributed, embedded, multi-tasking, multi-user and Real Time operating systems.

There are many types of operating systems, here are the ones you will need to know:

- Distributed – This is when data is shared between different computers in order to complete a single task. The operating system coordinates and controls all the computers involved to make it look like as if they were a single system
- Embedded – This is when an operating system is embedded into a hardware like a TV or calculator. They have CPU's which consume less power and they run with more efficiency. However, they have less memory
- Mutli-user – Like the name suggests, this is a operating system which allows more than one user to use the computer at the same time
- Multi-tasking – Again like the name suggests, this is a operating system which processes multiple tasks at **simultaneously**
- Real-time – This is a operating system which handles data input with a guaranteed time and takes action immediately. They are used around critical conditions like airplanes and hospitals

Spec reference:

(f) BIOS.

No no, this has nothing to do with Biology

So BIOS is a system which checks that peripherals are connected to the computer and working. This step is also known as POST (No nothing to do with Postman Pat)

So the BIOS does that before handing full control of the peripherals to the device drivers so the operating system can communicate with them, and this brings on to the next spec reference....

Spec reference:

(g) Device drivers.

Device drivers are basically devices which allow the peripherals to communicate with the operating system. Different peripherals have different device drivers

Spec reference:

(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 machine is software which **runs** on a hardware and allows programs to be emulated in them

Advantages of virtual machines:

- Allows multiple operating systems to run on the same computer
- Allows older applications not compatible with newer OS to be run on the same computer

- Allows shielding from malware since the malware will only affect the virtual machine and not the actual computer

Disadvantages of virtual machines:

- Programs will not run as smoothly in a virtual machine as they would on the computer
- You still need a license to operate the applications and the operating system in the virtual machine

Spec reference:

(a) The nature of applications, justifying suitable applications for a specific purpose.

There are many applications out there which allow people to do lots of useful shit, you need to know about these ones specifically and be able to apply them in certain scenarios:

- Word processors – Allows the making and printing of documents
- DTP – Allows the making of promotional materials (Leaflets, business cards etc…)
- Database managers – Helps to manage and store large amounts of related data
- Spreadsheets – These help to do calculations and make financial modellings (Storing bank account details, how much they paid etc…)

- Presentation and slide software – These display slides and the next slide to be shown can be determined by previous user input
- Animation – Helps to make moving images, used for a wide range of stuff like showing how weather patterns move in a Geography class
- Social networking – Helps people to communicate and share ideas with others much more easily
- Image and graphics manipulation – Improves the quality of images or graphical material
- Media and video editing – Helps to edit or create new videos like movies, ads etc...
- Gaming – Allows people to play games for leisure purposes (Which some of you have been doing for too long hence you are failing A levels)
- Email clients – Allows you to send emails
- Web browser – Allows you to access the World Wide Web

Spec reference:

(b) Utilities.

Utilities are software which help to maintain the computer and keep it running.

Here are some examples you need to know:

Anti-virus – This removes malicious programs from the computer

Disk fragmentation - (Before I dive in let me say what disk fragmentation is, basically is files scattered all over the memory) This allows all files to be compiled together so more space is available for more files to be stored

Compression – This reduces the size of files so they take up less space when they are stored

Firewalls – This protects computers or a network of computers from unauthorised access. It is used to protect against hackers but to also stop employees visiting inappropriate websites like PornHub during work hours

File managers – They allow the editing, deletion or creation of files. So they basically allow the management of files hence the name

Backing up – Automatically makes copies of files so there is a copy to return to in case the original file is lost or corrupted

Spec reference:

(c) Open source vs closed source.

Open source software is software which is available to the public while closed source is software.....come on still not getting it? Ah! It is software not available to the public. Closed source software is also known as proprietary software

Here are the differences between them:

Open source	Closed source
It is supplied with the source code	Has a highly polished user interface
It's free	It is copyrighted
Encourages mass user development	Has high development costs
Promotes creativity and sharing	Much more safer from being hacked into (Since hackers can't tamper with the code and find out its weaknesses)

Spec reference:

(d) Translators: Interpreters, compilers and assemblers

A translator is a software which translates source code into object code (Code which is understood by the computer)

Here are the different types of translators you need to know about:

Interpreters: These translate high-level source code into machine code. They translate one line of the source code at a time into multiple lines of machine code. If there is an error in the code, they stop and point the position of the error.

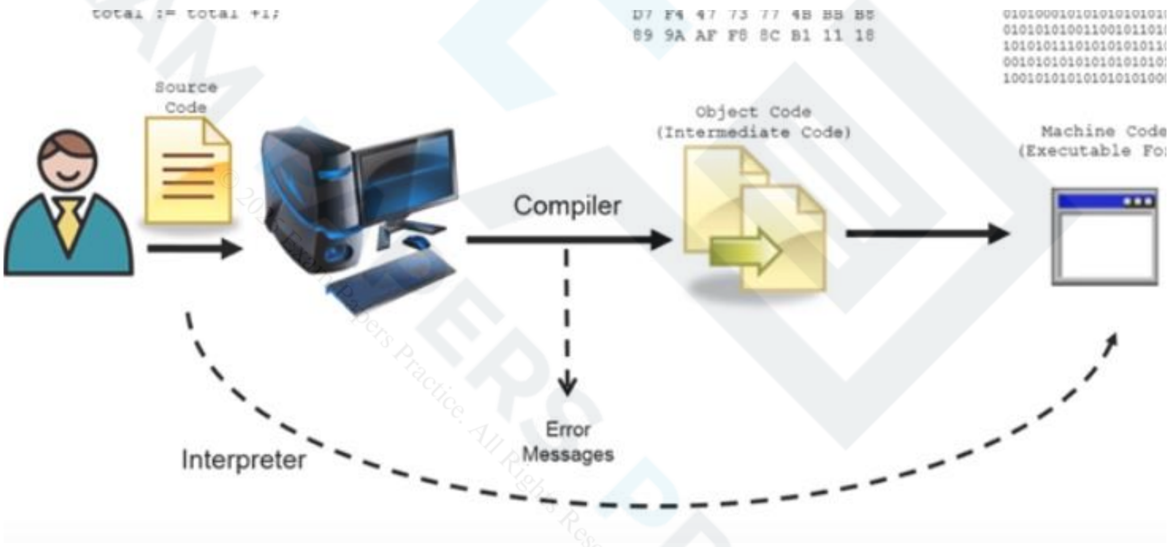
```
If (num1 >= 12) Then
total := total +1;
```

```
1010100010010010101110100100101111110010
0100101010100010110110101010101101011101
0101010100101011101001001011111100100100
1010101000101101101010101010110101101010 1
0101001010111010010010111111001001001010
1010001011011010101010101101011010101010 1
0010101110100100101111110010010010101010
0010110110101010101101011010101010101001 0
1010100010101010110101001010010010101010111
```

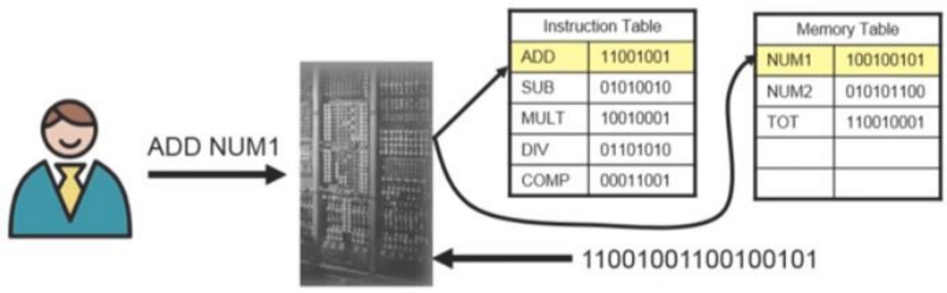
Few lines of high-level source code being translated to multiple lines of machine code

Is there a problem with interpreters? Well yes, each time the lines of high-level source code are translated, the interpreter has to look at the WHOLE source code which makes the process really slow

Compilers: These translate high-level source code into machine code. However, unlike the interpreters, they translate the whole source code into object code (This is different from machine code) from start to finish. The object code can then later be run as machine code



Assemblers: These translate low-level language into machine code. It translates one line of the source code at a time into a single line of machine code. However, assembly language is processor specific so it's not portable between different computers



Spec reference:

(e) Stages of compilation (lexical analysis, syntax analysis, code generation and optimisation).

The code has to be compiled in order to run as fast and as smoothly as possible in the computer, there are 4 stages all of them in order. Learn them

Lexical analysis – This is when all the white space is removed and the remaining code is turned into tokens (Basically components like plus signs, strings, letters etc..). A symbol tree is then created

Syntax analysis – The syntax of the code is checked against the rules of the language and if any of them breaks the rules, an error is generated. An abstract syntax tree is also created in this stage

Code generation – The abstract syntax tree created before is turned into object code

Code optimisation - (Just to let you know, optimisation means making the most efficient use of a resource) The code is optimised to run as fast as possible and any redundant code is removed

Spec reference:

(f) Linkers and loaders and use of libraries.

By libraries what we mean is a library of code, still stuck? Well what I am trying to say is…ehm…how do I explain this? Basically, they are bits of codes from different sources, for example remember how in your GCSE Computing or A level Computing courseworks you need to copy bits of codes from different sites in order to make a working solution? Yeah that is basically a library of code, all those bits of code stored together make a library of code

What do linkers do? They include links to any library code that is needed and put all of them into a single executable file.

Is there a problem with this? Yes, it makes the final code that is being compiled really large in size.

Is there a solution for this? Of course, there is! Use something called dynamic linking, this is where all the library code is stored on the computer and the operating system links them directly to the final compiled code when it is being run. This reduces the size of the final compiled code

What does the loader do? It's in the name, the loader loads the compiled code in memory ready for use

Spec reference:

(a) Understand the waterfall lifecycle, agile methodologies, extreme programming, the spiral model and rapid application development.

In this section you will learn about the different ways programmers develop programs for their clients. Here are some of the ways:

- Waterfall lifecycle – The program is developed into stages which are put in order, you might have to go back to the previous stages in order get new requirements and improvements
- Agile methodologies - Requirements of the program will increase as it's being developed. Iterations of the program are made with each iteration having increased requirements
- Extreme programming – This is when a user is involved in the development of the code. Iterations of the programs are made and the user can give instant feedback so the program is built into successive iterations with new requirements
- Spiral model – The requirements of the program are set, then the risks are identified and solved. This keeps on going round and round the cycle until the program is accepted
- RAD – This is when a program is developed with a reduced functionality, it is then tested by the user who gives feedback. The feedback is used to improve the code and

this keeps on going round and round the cycle until it is accepted

Spec reference:

(b) The relative merits and drawbacks of different methodologies and when they might be used.

Advantages of the waterfall cycle:

- An easy model to manage
- Has clear deliverables

Disadvantages of waterfall cycle:

- Lack flexibility
- Has a high risk factor
- There is no collaboration with the user
- Not suitable for programs with unsure requirements

Advantages of RAD:

- User is involved in the making of the program
- The final version of the program is more likely to fit the user's needs

Disadvantages of RAD:

- Heavy collaboration with the user is required

- Unsuitable for large projects
- Not suitable if efficiency of code is a priority

Advantages of the spiral model:

- Risks are identified and solved
- Works well with large projects

Disadvantages of the spiral model:

- Risk management skills are costly

Advantages of extreme and agile programming:

- There will be fewer bugs in the code
- The quality of the final program will be higher

Disadvantages of extreme and agile programming:

- Heavy collaboration with the user is required
- It's expensive

Spec reference:

(c) Writing and following algorithms.

This is basically writing and following flowcharts and pseudocode, later in the sections probably when you learn about all the concepts of paper 2 you will learn more about pseduocode and flowcharts and how to follow them. You should already have an understanding of flowcharts and how to follow them from your studies during key stage 3 and 4 of Computer Science

Spec reference:

(a) Need for and characteristics of a variety of programming paradigms

Let's make one thing clear about what do we mean by programming paradigms, programming paradigms is, in simple words different **kinds** of programming languages.

There are many different types of programming languages, three of them are:

- Procedural
- Object Orientated
- Assembly/low level language

Examples of Procedural will be Pascal, Python or C++

Examples of Object Orientated will be Java or Processing

Examples of assembly/low level language (Well there is only one example of assembly language lol) is LMC or Little Man Computer as some may call it

Why do we need different programming paradigms? Well simply because some problems are much better suited at being solved in a different programming paradigm than the others

Spec reference:

(b) Procedural languages.

This section will be focusing on a little description of procedural languages. So here it is -

Procedural languages are a series of instructions which tell how things need to be done step by step in a sequence. It has a selection to make choices and iteration about how many times certain instructions need to be done. The instructions are coded in functions or procedures

Procedural languages are also known as imperative languages because they give you orders and how they should be done.

Spec reference:

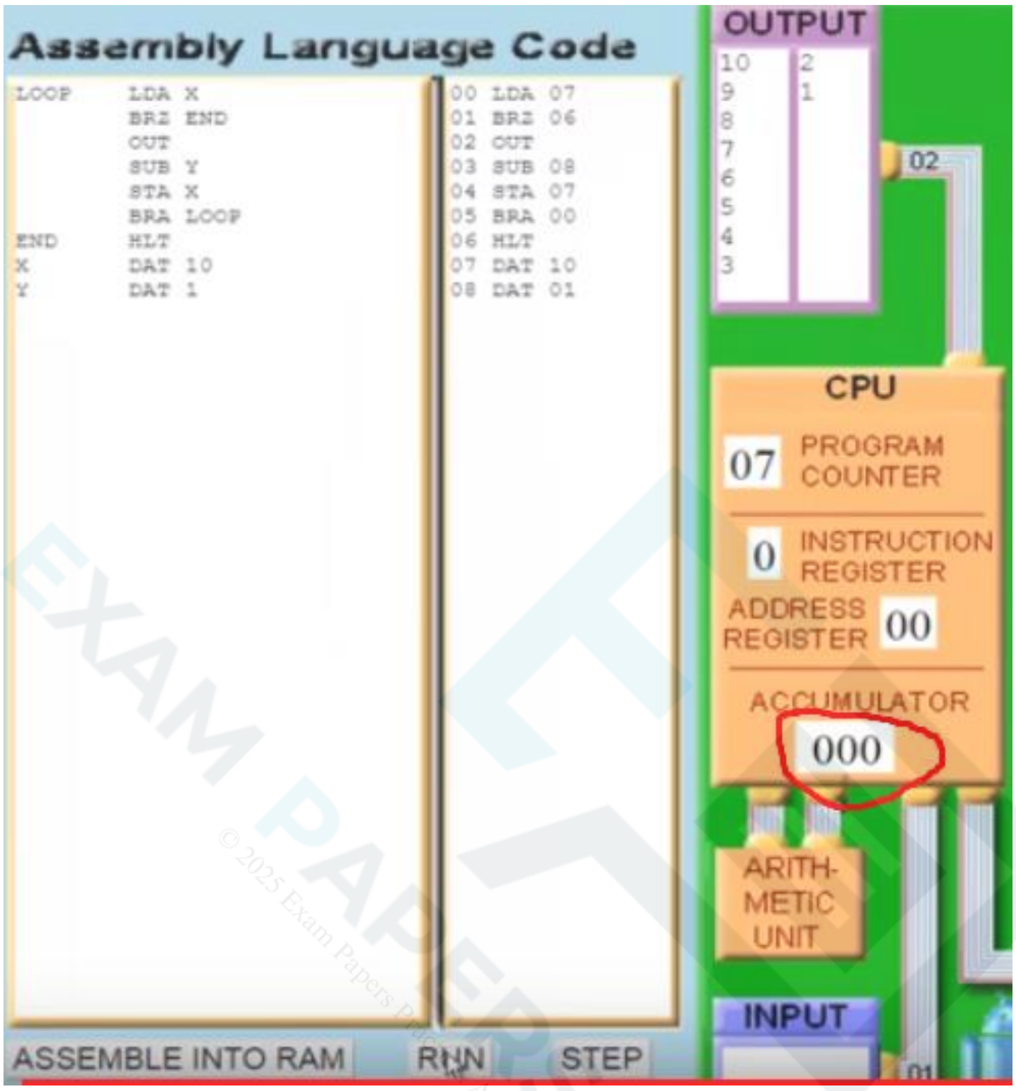
(c) Assembly language (including following and writing simple programs with the Little Man Computer instruction set). See appendix 5d

Assembly language is low level language and is processor specific meaning it will not **run** on all computers

There are some commands you need to learn, here are the ones you will need for the exams:

- ADD – This add data
- SUB – This subtracts data
- LDA – This loads data
- HLT – This ends the program
- OUT – This outputs data or the result
- DAT – This declares like a variable where input data which brings us to the next command
- INP – Used to take in data
- BRA – This is branch always, basically what it means is that branch a specific part of the code no matter what the result is
- BRP – Branch if positive, what this means is that if the result is a positive integer branch a specific part of the code
- BRZ – This is branch if zero, what this means is that if the result is 0 then branch a specific part of the code

I realize that a lot of students will find the BRA, BRP and BRZ commands difficult to understand so let me explain how they work:

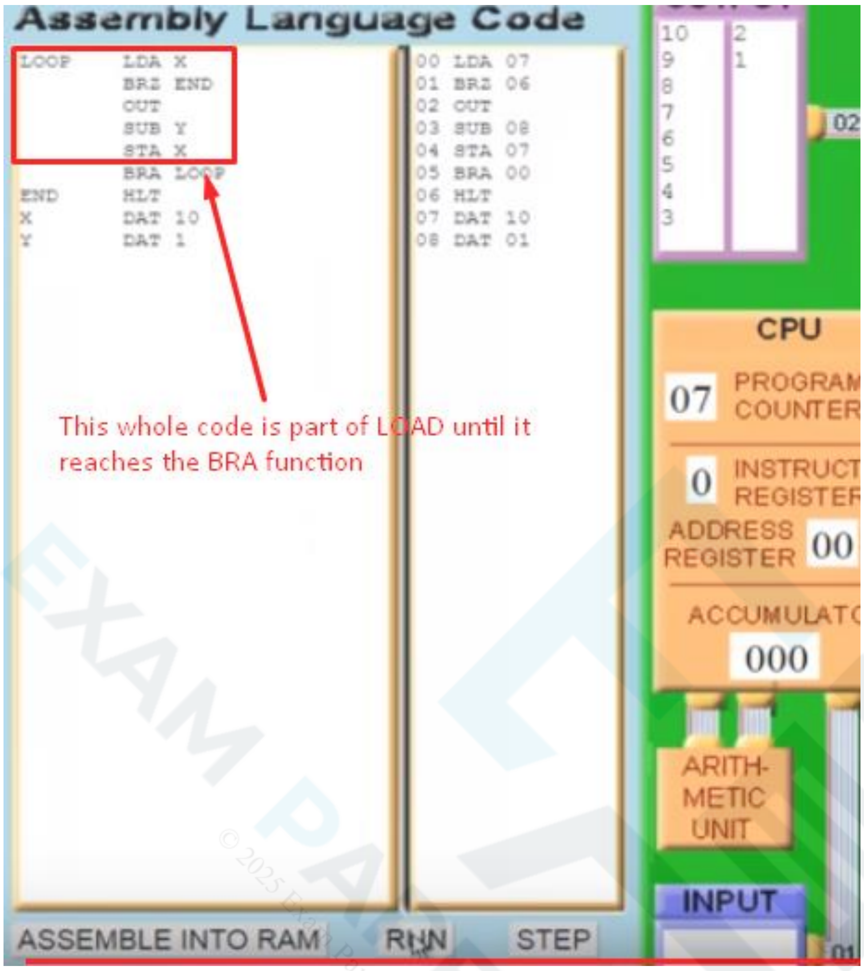


```
LOOP    LDA X
        BRZ END
        OUT
        SUB Y
        STA X
        BRA LOOP
END     HLT
X       DAT 10
Y       DAT 1
```

```
00 LDA 07
01 BRZ 06
02 OUT
03 SUB 08
04 STA 07
05 BRA 00
06 HLT
07 DAT 10
08 DAT 01
```

So here is our LMC interface, as you can see the BRZ and BRP commands have words written next to them, these words indicate which part of the code to branch again.

So for example BRA has the word LOOP next to it, this means that no matter what the result of the code so far is, which is displayed in the accumulator, the part of the code which has the word LOOP written next to it will be branched (executed) again and again until it reaches a 0 where the BRZ function will end the code:

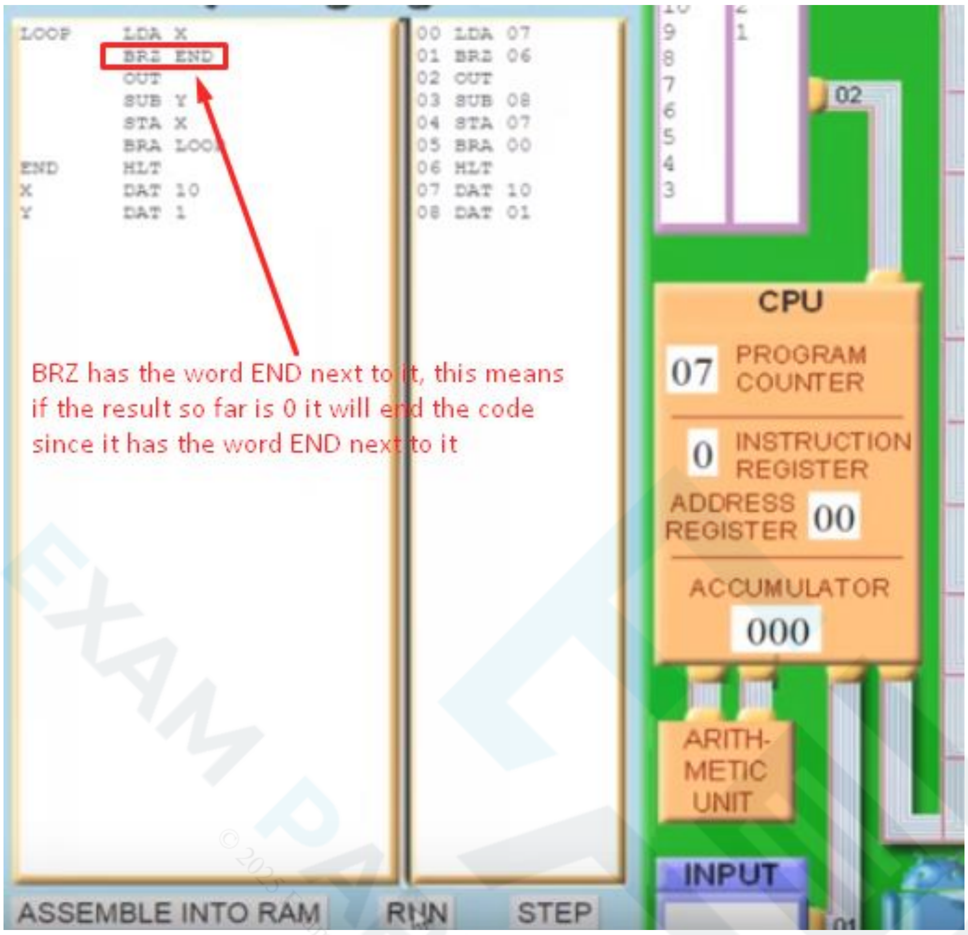


Assembly Language Code

```
LOOP      LDA X
          BRZ END
          OUT
          SUB Y
          STA X
          BRA LOOP
END       HLT
X         DAT 10
Y         DAT 1
```

```
00 LDA 07
01 BRZ 06
02 OUT
03 SUB 08
04 STA 07
05 BRA 00
06 HLT
07 DAT 10
08 DAT 01
```

This whole code is part of LOAD until it reaches the BRA function



```
LOOP    LDA X
        BRZ END
        OUT
        SUB Y
        STA X
        BRA LOO
END     HLT
X       DAT 10
Y       DAT 1
```

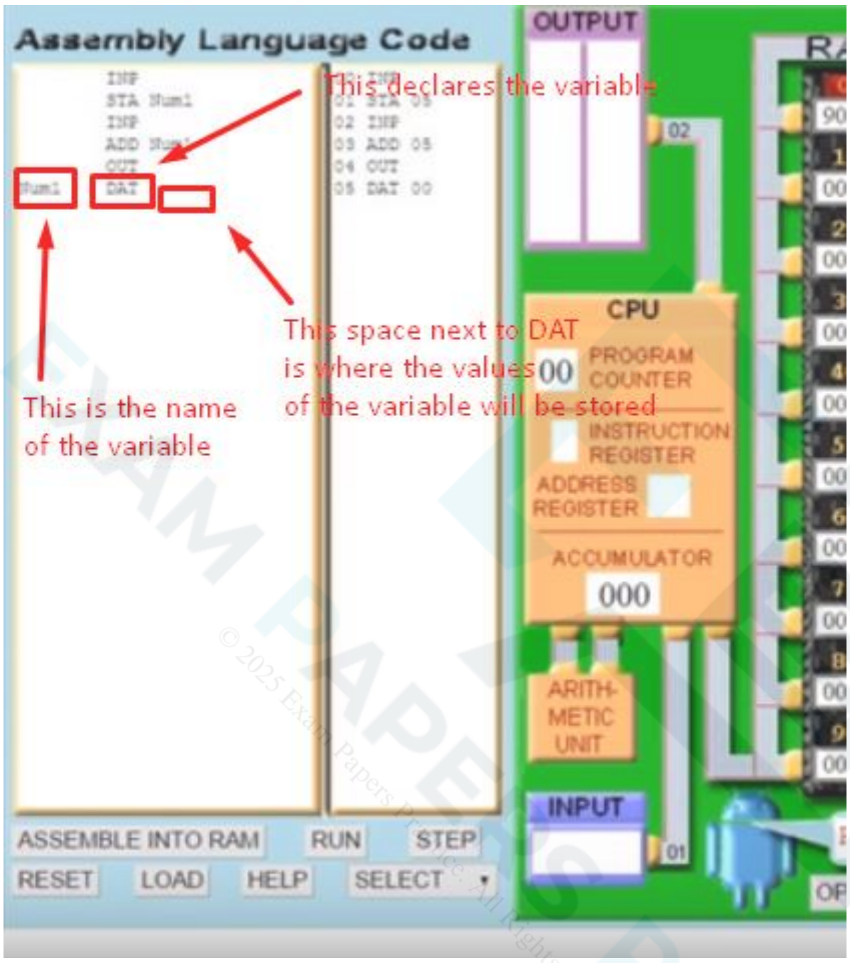
```
00 LDA 07
01 BRZ 06
02 OUT
03 SUB 08
04 STA 07
05 BRA 00
06 HLT
07 DAT 10
08 DAT 01
```

BRZ has the word END next to it, this means if the result so far is 0 it will end the code since it has the word END next to it

The same logic applies to BRP except if the result of the code so far is a positive number (Which always will be probably) then you branch the code from a certain point indicated by the word next to it.

So you could argue that the BRP command is the same as the BRA command but then again it's A level Computer Science where some things are just ?@&%ed up, and therefore you have to keep this difference in mind. They might for example ask you to state the difference between them in the exam so it's good to know the differences between them, even tho they appear to have the same function.

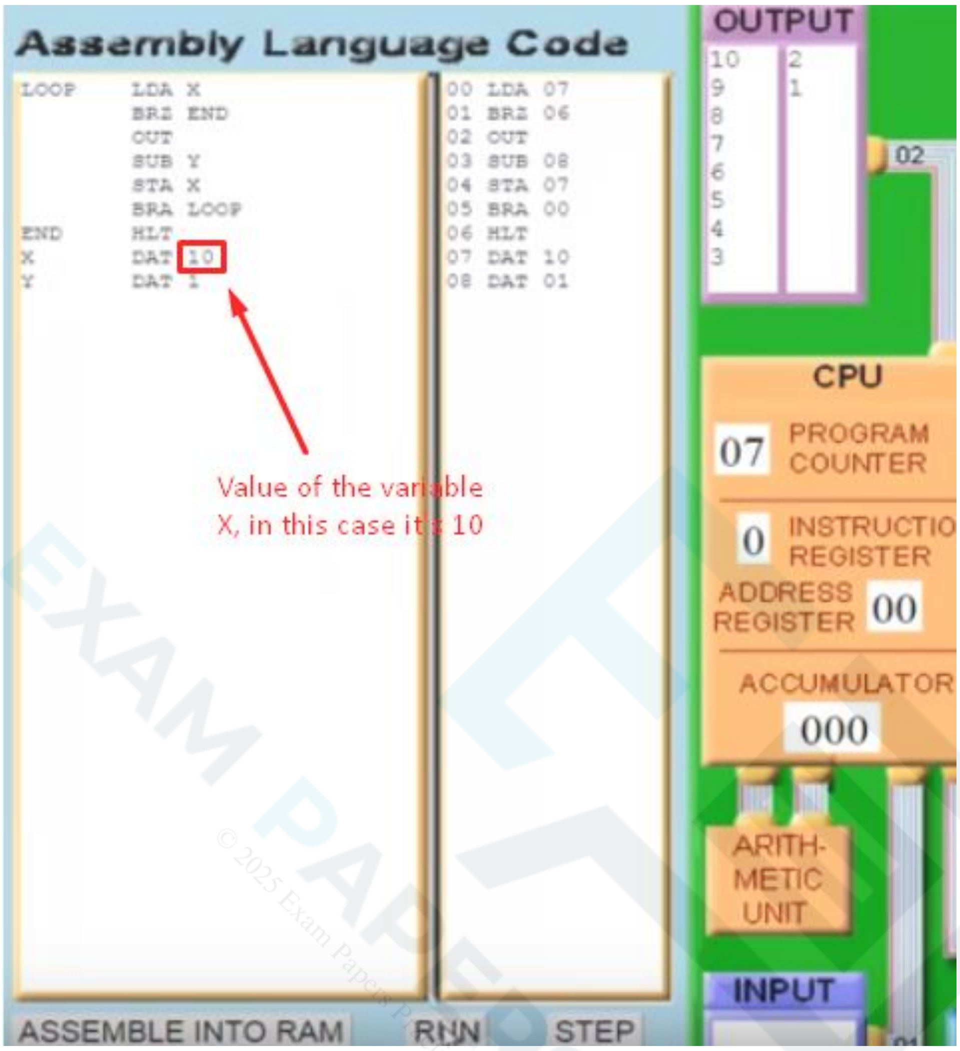
Some of you may be confused with the DAT command, like it has been said before, it is used to declare a variable. The picture below will show you in more detail:



```
     INP
     STA Num1
     INP
     ADD Num1
     OUT
Num1 DAT
```

```
00 INP
01 STA 05
02 INP
03 ADD 05
04 OUT
05 DAT 00
```

So in this picture, the STA command is saying “Store the user input in the variable Num1” and it does just that, the value of the user input is now stored in the variable Num1.



```
LOOP    LDA X
        BRZ END
        OUT
        SUB Y
        STA X
        BRA LOOP
END     HLT
X       DAT 10
Y       DAT 1
```

```
00 LDA 07
01 BRZ 06
02 OUT
03 SUB 08
04 STA 07
05 BRA 00
06 HLT
07 DAT 10
08 DAT 01
```

Another picture just to understand the DAT command better

Spec reference:

(d) Modes of addressing memory (immediate, direct, indirect and indexed).

Now we are going to learn about addressing modes, basically the whole purpose of this is to fetch the right data for execution.

There are ways of addressing the memory (Which means providing addresses of the memory location of where the data is) in order to do this. You need to learn about the following ways:

Immediate addressing – This is when the address part of the instruction is the actual data to be used.

Example:

Instruction:

ADD 10

The address part of the instruction is 10 but actually 10 is not a memory location but the actual data to be used.

Direct addressing – This is where the address part of the instruction **contains** the memory location of where the data should be fetched from

Example:

Instruction:

ADD 7

The address part of the instruction is 7, this means our data is located at memory location 7. GET THERE NOW

Indirect addressing – This is where the address part of the instruction **contains** the memory location which will contain the real memory location of where the data should be fetched from

Example:

Instruction:

ADD 7

The address part of the instruction is 7, this means location 7 will contain another memory location, say for example 30. So our data really is located at memory location 30

Indexed addressing – This is where the address part of the instruction is added to a value in the index register (Which is a constant value) to obtain the memory location of where the data should be fetched from. It is used in order to carry out the same command a number of times

Example:

Instruction:

ADD 7 ADD 5

Index Register

7

The address part of the **first** instruction is 7 so 7 is added to index register value 7. This gives a number of 14 so the data needed is found in memory location 14. Then the address part of the **second** instruction, 5 is added to 7, so them added together gives a value of 12. Therefore our second data to be used is found in memory location 12 and so on

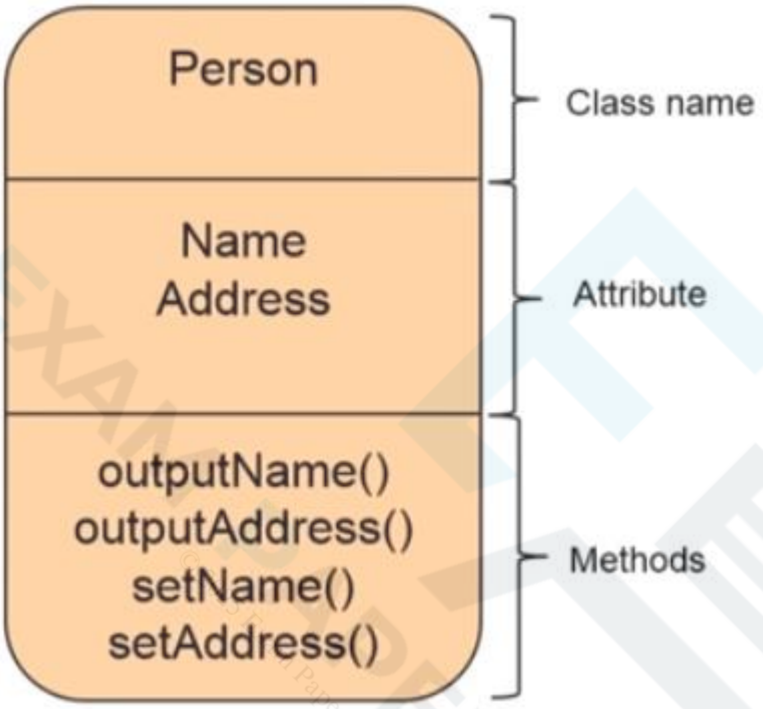
Spec reference:

(e) Object-oriented languages (see appendix 5d for pseudocode style) with an understanding of classes, objects, methods, attributes, inheritance, encapsulation and polymorphism.

Objects, classes, methods, attributes and inheritance

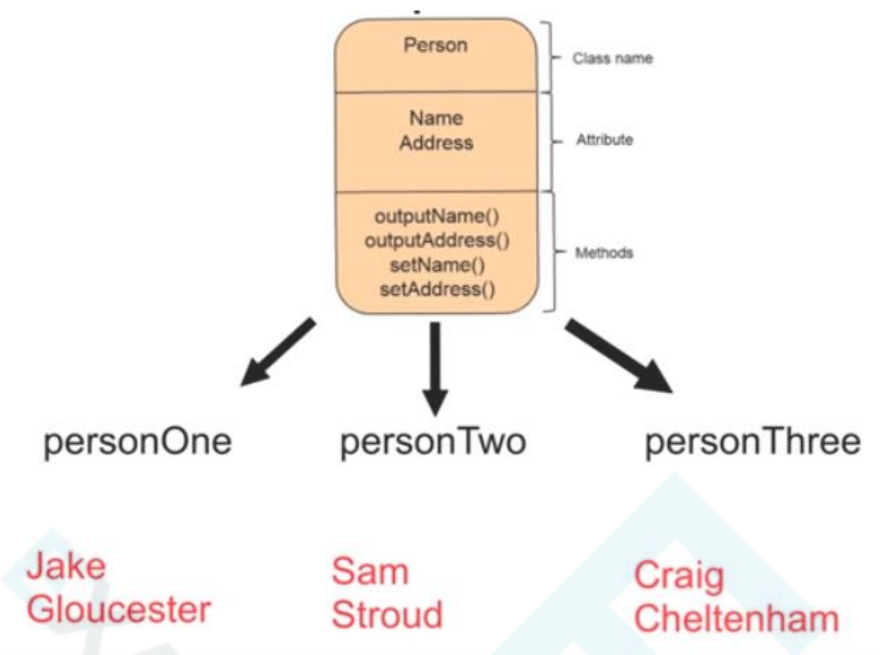
An object is..well a thing like a tree etc..but you don't need to know anything about its definition. Just know what it is so you know the next things I am going to talk about.

Here is your object:



Here is the object, the class is what it defines it, the attributes are the features of the object and the methods are the methods of accessing the attributes in order to change the attributes in this case you can change things like name, address etc...

There is a definition you need to know (At least in my opinion), it's called **instantiation**. Instantiation is the process of creating an object from a class template, like this:



So here you can create new objects like the person's friends. They will inherit all of the original class's (Which is called the super class) attributes and methods of accessing the attributes. The only things these new objects (which are referred to as sub-classes) won't inherit from the original class is the class's name, obviously.

Now if you wish to change the names of one of your friends in the sub class and a method for doing that is already available in the super class (In this case it is, setName()) then you just follow this format:

*Subclass name.*super.*method of accessing the attribute*

So in this example, if I want to change the name of personOne, I do the following:

PersonOne.super.**setName(And here you set whatever name you like)**

PersonOne is the name of the sub class, super is the keyword that needs to be used in order to access the method and **setName()** is the appropriate method of changing the name. The dots are obviously needed to separate out the words.

Still confused? Read over the whole concept again, trust me you will get it, is not hard to understand.

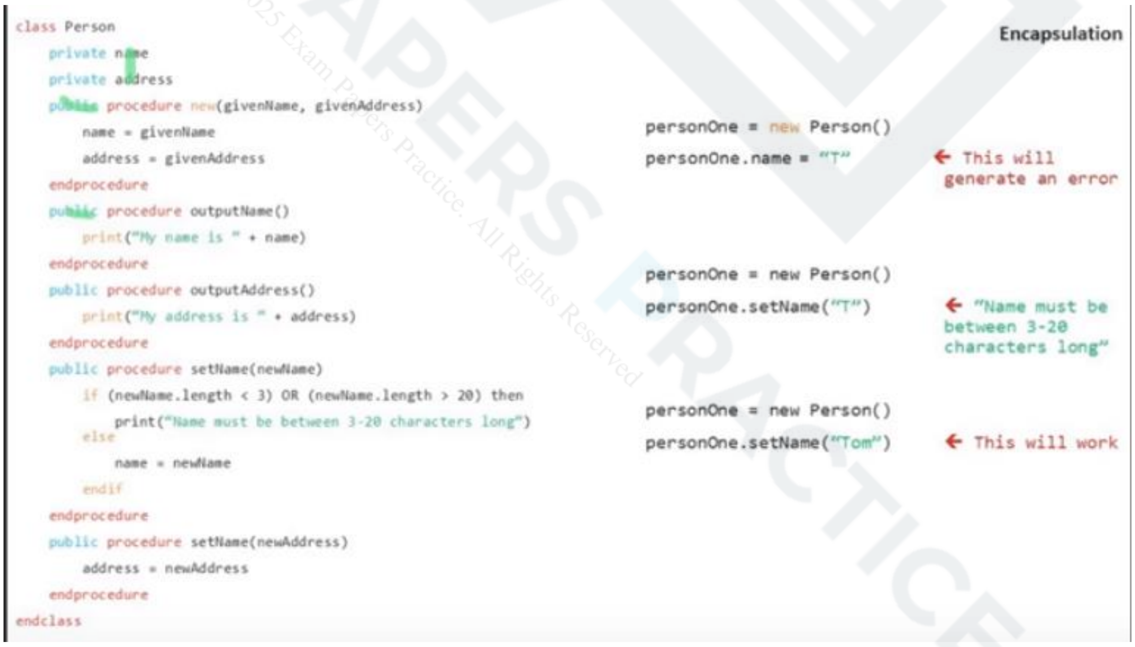
Encapsulation

Now let's learn about something called encapsulation. What is encapsulation?

Encapsulation is a way of protecting the attributes and the methods of accessing attributes of an object so that they can't be altered by other objects. These things can only be accessed through certain methods.

Why is this used? So that the programmer is in control of their code and the code can only be used in the way they want it to be used

Here is encapsulation in action:



Encapsulation

```
class Person
    private name
    private address
    public procedure new(givenName, givenAddress)
        name = givenName
        address = givenAddress
    endprocedure
    public procedure outputName()
        print("My name is " + name)
    endprocedure
    public procedure outputAddress()
        print("My address is " + address)
    endprocedure
    public procedure setName(newName)
        if (newName.length < 3) OR (newName.length > 20) then
            print("Name must be between 3-20 characters long")
        else
            name = newName
        endif
    endprocedure
    public procedure setName(newAddress)
        address = newAddress
    endprocedure
endclass
```

```
personOne = new Person()
personOne.name = "T"              ← This will
                                    generate an error

personOne = new Person()
personOne.setName("T")            ← "Name must be
                                    between 3-20
                                    characters long"

personOne = new Person()
personOne.setName("Tom")          ← This will work
```

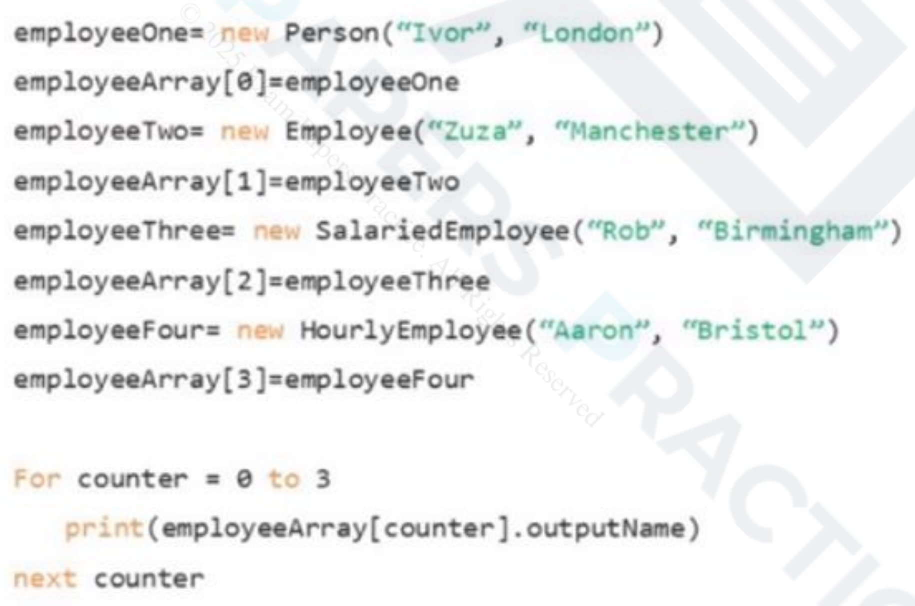
Here you have to use the method .setName() in order to declare a name within a new object and the name has to be at least 3 letters and maximum of 20 letters

Polymorphism

And now let’s move on to something called polymorphism. Polymorphism is when a token of a code has more than one function. For example, a + sign may add numbers or join two words together to make a phrase. So, the + sing acts both as a arithmetic operator and a **concatenation** operator. Or the = sign, it can either be used to assign a value to a variable, change the value of a variable or compare values. So, it acts as an initiating operator, an assignment operator and a comparison operator.

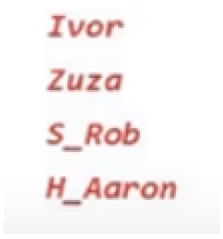
If you are confused remember, we are not talking about Python or any high-level language here. Just pseudocode

Let’s look at a complex polymorphism example:



```
employeeOne= new Person(“Ivor”, “London”)
employeeArray[0]=employeeOne
employeeTwo= new Employee(“Zuza”, “Manchester”)
employeeArray[1]=employeeTwo
employeeThree= new SalariedEmployee(“Rob”, “Birmingham”)
employeeArray[2]=employeeThree
employeeFour= new HourlyEmployee(“Aaron”, “Bristol”)
employeeArray[3]=employeeFour

For counter = 0 to 3
   print(employeeArray[counter].outputName)
next counter
```

So here is some code, it is showing people, employees and who is a salaried or hourly one. The method .outputName should only output the names of these people when the code is run, right? Well think again, here are the results when this code is run:



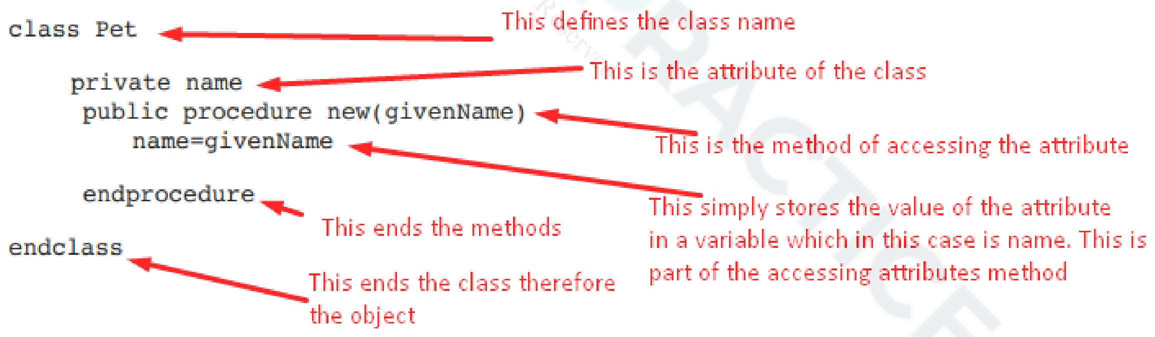
```
Ivor
Zuza
S_Rob
H_Aaron
```

Sure the names have been outputted, but it has also outputted who is a hourly employee and who is a salary one. So what the method .outputName here did was output all the names as well as outputting who is a salaried employee and who is a hourly one but these are not names obviously. This is polymorphism in action

Pseudocode for object-oriented languages

You will also need to learn some pseudocode for object-oriented languages so you can familiarise yourself with them in the exam

Methods and attributes pseudocode



```
class Pet

     private name
      public procedure new(givenName)
          name=givenName

      endprocedure

endclass
```

This defines the class name

This is the attribute of the class

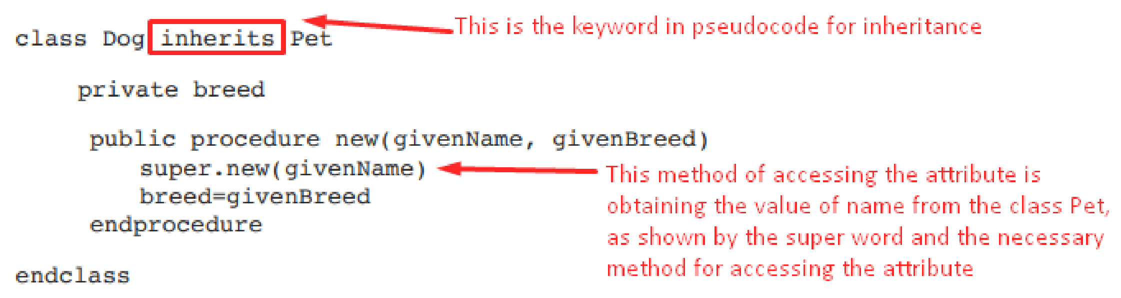
This is the method of accessing the attribute

This simply stores the value of the attribute in a variable which in this case is name. This is part of the accessing attributes method

This ends the methods

This ends the class therefore the object

Inheritance and constructors pseudocode



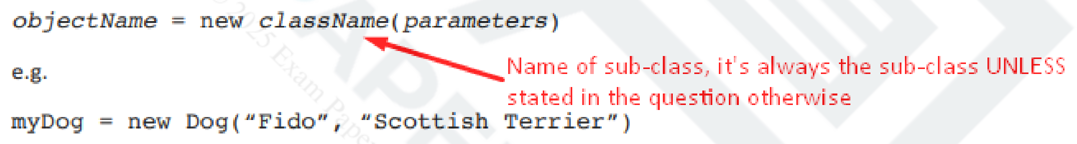
```
class Dog inherits Pet

    private breed

     public procedure new(givenName, givenBreed)
         super.new(givenName)
         breed=givenBreed
     endprocedure

endclass
```

This is the keyword in pseudocode for inheritance

This method of accessing the attribute is obtaining the value of name from the class Pet, as shown by the super word and the necessary method for accessing the attribute

The constructors pseudocode is show by the word new, this is just showing that a new method of accessing the **new attributes** are being created. In this case there is an additional attribute: breed, so there is a need for a new method of accessing the attribute breed.

And finally the pseudocode or format to create a new object from instantiation is the below:



```
objectName = new className(parameters)

e.g.

myDog = new Dog("Fido", "Scottish Terrier")
```

Name of sub-class, it's always the sub-class UNLESS stated in the question otherwise

And by the way, one final thing: the attributes and methods of accessing the attributes can always be assumed to be public unless stated otherwise.

Now you may be thinking, what the %@&/ is private and public? It's simple, anything public can be accessed by other objects while anything private cannot and can only be accessed within that object/class. So, in these above examples, it makes sense that the attributes of objects (The names of the dog and their breed) are kept private so that they can't be altered by other objects but the methods of accessing them should be public so that other objects can obtain their values.

Yay! Well done! You have come to the end of the second section of OCR A level Computer Science!

Exchanging Data

Spec reference:

(a) Lossy vs Lossless compression.

There are two ways to compress a file, in other words there are two ways to reduce the size of a file:

Lossy compression: This is where actual data is removed from the file, an algorithm is used to remove the least important data from the file so it has its size reduced. An original file compressed in this way **cannot** be recreated

Lossless compression: Actual data is removed from the file but the removed data is encoded in a way so that the file can be recreated. An original file compressed in this way **can** be recreated

Spec reference:

(b) Run length encoding and dictionary coding for lossless compression.

Remember how we said that in lossless compression removed data is encoded in a way so that the original fine can be recreated? Well here you are going to learn exactly what those ways are and there are probably many ways out there but since the exam board has some.......mercy on you they decided to make life a little bit less hard for you, oui? (Dunno why I said yes in French but whatever)

Run length coding: This is where redundant data is stored once along with an index of how many times the redundant data has been repeated

Dictionary coding? This is where the data items in the file are stored with indexes and a sequence of occurrences so the original file can be recreated

Spec reference:

(c) Symmetric and asymmetric encryption.

This section focuses on how to send messages safely through the internet so that you can send pics to your partner through Grindr in a way that none else can see your pics.

Symmetric encryption – This is where a message is encrypted and decrypted using the same key.

These are the problems with this method:

- Production of the key can be copied in order to obtain a copy of the key so the message can be intercepted and read by hackers

But one benefit of this encryption method is:

- The key can be used as many times as you like

Asymmetric encryption – Different keys are used to encrypt and decrypt the message; the sender shares its public key with the recipient and then the sender encrypts the message with their public **and private** key, and the recipient decrypts the message using the sender's public key and their own private key.

Benefits:

- Unauthorised people can't read the message
- Message will be authentic
- Unauthorised people can't modify the message

Spec reference:

(d) Different uses of hashing.

Hashing is when data is turned into a numerical value (Like passwords being turned into dots)

Here are two uses of hashing:

- Generating disk addresses to store data on a random access device
- Checking and storing passwords

Spec reference:

(a) Relational database, flat file, primary key, foreign key, secondary key, entity relationship modelling, normalisation and indexing. See appendix 5f.

Ah databases, you probably come across them and for A levels you don't need to know the exact definition of a database unlike GCSE, as long as you just know what a database is you should be fine with this section

A relational database contains multiple tables which are all linked together.

Some benefits/features of relational databases:

- Improves data integrity
- Has reduced data redundancy
- Adds user security
- Querying (Like modifying) is easier in a relational database

In comparison, a flat file database is a single table which contains all records and fields.

Some features(Drawbacks in my opinion) of a flat file database:

- Has data redundancy
- Simple to use
- Querying is harder

<u>Primary and foreign keys</u>

You should know what a primary key is from GCSEs, if you don't then behold the definition of a primary key: a unique identifier!

A foreign key is a primary key which acts as an attribute in another table, creating a relationship between two or more tables.

<u>Secondary key and indexing</u>

A secondary key is an index which can be used to search a database quicker by humans.

So for example if a table has the records for a photoshoot and the records are Name of photographer, Place, Date, Customer name what will be the appropriate secondary key to search for your photo?

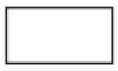
Customer name of course, because 1. It's easier for the customers themselves and 2. It's unlikely two customers will have the same name.

And indexing is well as already been said, the searching of a database.

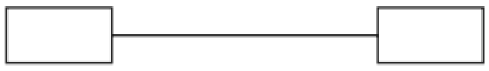
<u>Entity diagrams</u>

Now here are some entity relationship diagrams, learn them:

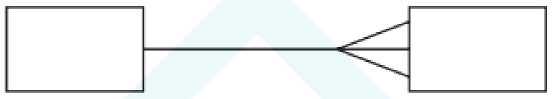
Entity



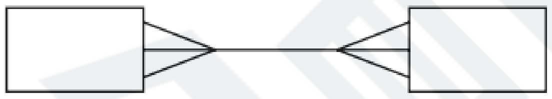
One-To-One relationship



One-To-Many relationship



Many-To-Many relationship



As for normalisation, it will be covered in the spec reference: Normalisation to 3NF

Spec reference:

(b) Methods of capturing, selecting, managing and exchanging data

There are three ways of capturing data, these are:

- Text boxes, these can tell the user what data input is needed and forces the user to give a data input
- OCR (Not the exam board obviously), this scans the characters in a document and compares them to a library of character images to find the best match. An example of

where OCR might be used is reading documents to blind people or track number plates on cars

- OMR, this scans the position of a mark in a document. It uses this as the data input. An example of where this might be used is marking multi-choice exam questions

Selecting data

There is not much to say for this, there are two ways of selecting data:

- SQL (It’s a language)
- Drop down boxes

Managing data

This is done by the database management (DBMS), you need to know how the database management manages the data:

- Provides a manipulation language to access and change the data
- Provides an interface for other programs to access and use the data
- Provides additional security
- Enforces validation rules
- Maintains integrity to make sure the efficiency and structure of the database is not disrupted

Exchanging data

Data can be exchanged in many forms like XML, JASON or CSV

A feature called EDI can exchange data between databases.

How does EDI work? It connects two interfaces of databases together so they can read and write to each other's tables, this can be done by importing or exporting routines or by live connection

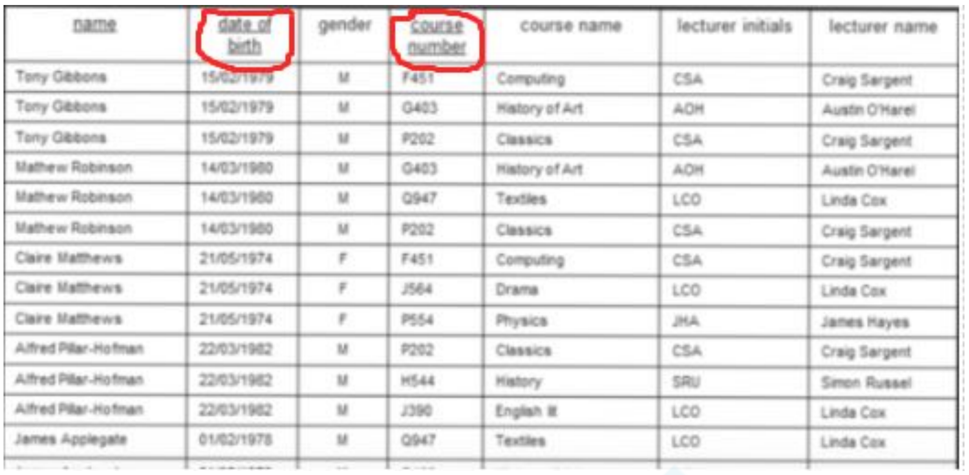
Spec reference:

(c) Normalisation to 3NF.

This will be a pretty challenging topic to explain, the concepts themselves are simple to understand but teaching it the first time round is difficult. Just hang on....

Normalisation to 3NF basically means turning a flat file database into a relational database. In this section you will learn how that is done

Before diving in you need to understand what a composite key is, a composite key is when more than one attribute of a database is used as a primary key so the picture below will make it a lot more clearer:



name	date of birth	gender	course number	course name	lecturer initials	lecturer name
Tony Gibbons	15/02/1979	M	F451	Computing	CSA	Craig Sargent
Tony Gibbons	15/02/1979	M	G403	History of Art	AOH	Austin O'Harel
Tony Gibbons	15/02/1979	M	P202	Classics	CSA	Craig Sargent
Mathew Robinson	14/03/1980	M	G403	History of Art	AOH	Austin O'Harel
Mathew Robinson	14/03/1980	M	Q947	Textiles	LCO	Linda Cox
Mathew Robinson	14/03/1980	M	P202	Classics	CSA	Craig Sargent
Claire Matthews	21/05/1974	F	F451	Computing	CSA	Craig Sargent
Claire Matthews	21/05/1974	F	J564	Drama	LCO	Linda Cox
Claire Matthews	21/05/1974	F	P554	Physics	JHA	James Hayes
Alfred Pillar-Hofman	22/03/1982	M	P202	Classics	CSA	Craig Sargent
Alfred Pillar-Hofman	22/03/1982	M	H544	History	SRU	Simon Russel
Alfred Pillar-Hofman	22/03/1982	M	J390	English II	LCO	Linda Cox
James Applegate	01/02/1978	M	Q947	Textiles	LCO	Linda Cox

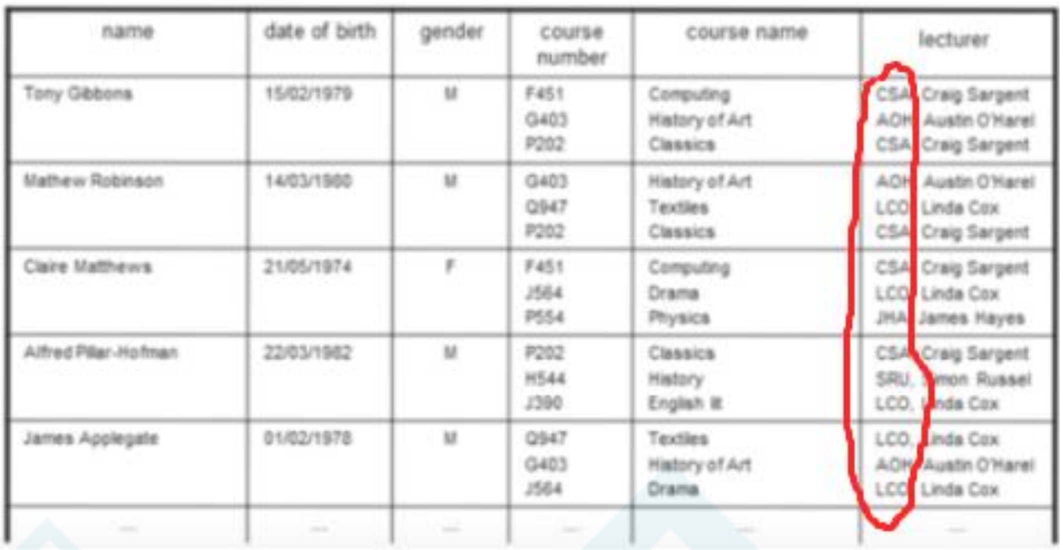
(In case you are confused with what an attribute is, it's basically the heading of a column) So in this table there is no attribute I can use as my primary key as none of them have unique values, therefore I will use BOTH course number and date of birth as my primary key and this is called a composite key, a primary key made out of more than one attribute as there can be a limitless number of attributes in a primary key. (A composite key may well contain 100 attributes but then again, what kind of database is that?)

So here are the steps for normalisation to 3NF:

First you need to turn your database into 1NF format, to do that you:

- Remove any duplicate attributes
- Remove any groups of repeating data
- Identify the primary key
- Separate out any attributes which are not atomic into separate attributes

You may be confused with the last point, allow me to explain:



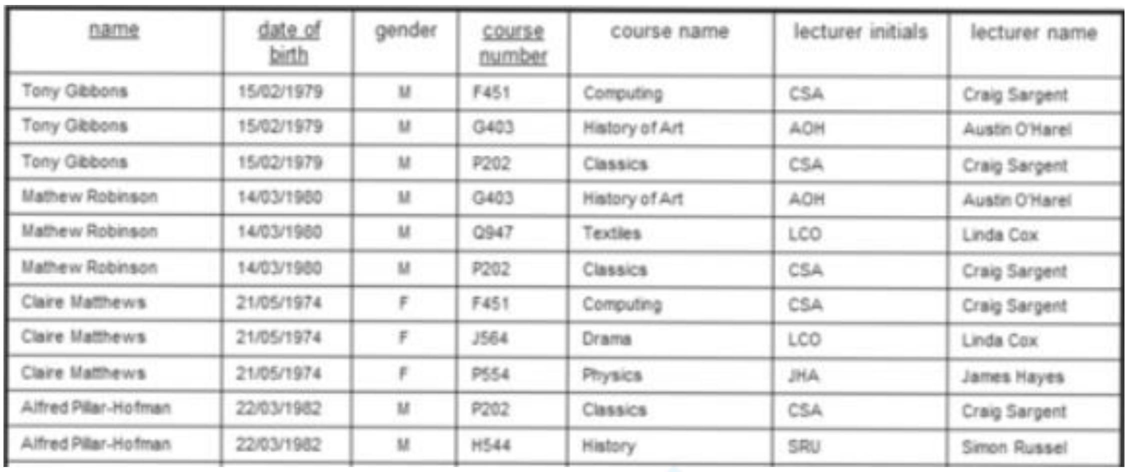
name	date of birth	gender	course number	course name	lecturer
Tony Gibbons	15/02/1979	M	F451 G403 P202	Computing History of Art Classics	CSA Craig Sargent AOH Austin O'Harel CSA Craig Sargent
Mathew Robinson	14/03/1980	M	G403 Q947 P202	History of Art Textiles Classics	AOH Austin O'Harel LCO Linda Cox CSA Craig Sargent
Claire Matthews	21/05/1974	F	F451 J564 P554	Computing Drama Physics	CSA Craig Sargent LCO Linda Cox JHA James Hayes
Alfred Pillar-Hofman	22/03/1982	M	P202 H544 J390	Classics History English lit	CSA Craig Sargent SRU, Simon Russel LCO, Linda Cox
James Applegate	01/02/1978	M	Q947 G403 J564	Textiles History of Art Drama	LCO, Linda Cox AOH Austin O'Harel LCO Linda Cox
—	—	—	—	—	—

Here is your flat file database, in the attribute 'lecturer' you will notice that not only does it have the names of the lecturers but also their codes like CSA, AOH etc...these need to be separated into separate attributes as they are separate things

Second you need to convert your database into 2NF format, to do that you:

- Check that the database is already in 1NF format
- Remove any partial dependencies
- Fix any Many to Many relationships

A partial dependency is when an attribute depends on just some parts of a composite key as their primary key, it does not use all parts of the composite key as their primary key, here is an example which will explain it better:

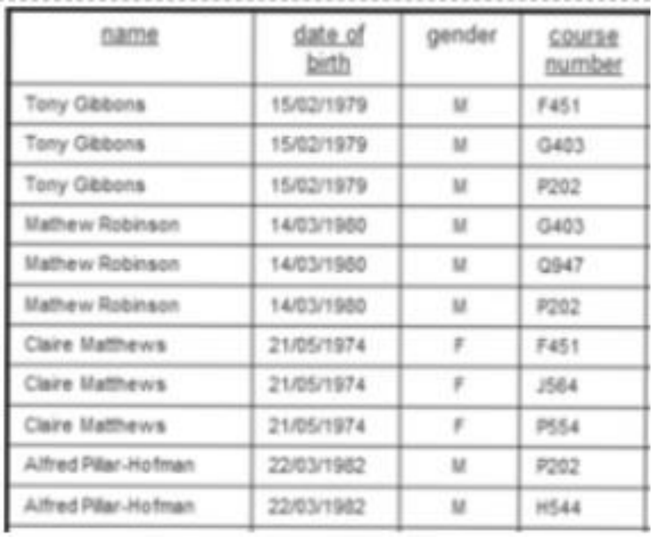


name	date of birth	gender	course number	course name	lecturer initials	lecturer name
Tony Gibbons	15/02/1979	M	F451	Computing	CSA	Craig Sargent
Tony Gibbons	15/02/1979	M	G403	History of Art	AOH	Austin O'Harel
Tony Gibbons	15/02/1979	M	P202	Classics	CSA	Craig Sargent
Mathew Robinson	14/03/1980	M	G403	History of Art	AOH	Austin O'Harel
Mathew Robinson	14/03/1980	M	Q947	Textiles	LCO	Linda Cox
Mathew Robinson	14/03/1980	M	P202	Classics	CSA	Craig Sargent
Claire Matthews	21/05/1974	F	F451	Computing	CSA	Craig Sargent
Claire Matthews	21/05/1974	F	J564	Drama	LCO	Linda Cox
Claire Matthews	21/05/1974	F	P554	Physics	JHA	James Hayes
Alfred Pillar-Hofman	22/03/1982	M	P202	Classics	CSA	Craig Sargent
Alfred Pillar-Hofman	22/03/1982	M	H544	History	SRU	Simon Russel

Let's go back to this database, so my composite key is date of birth and course number however three attributes which are course number, lecturer initials and lecturer name are partial dependencies, how do I know this?

Well you have to use logic and common sense here, I am not going to know anything with just date of birth but with course number I know the course name, the lecturers and their initials by just using the course number. So, in order to know these three things there is no need to use date of birth as well therefore these attributes are partial dependencies.

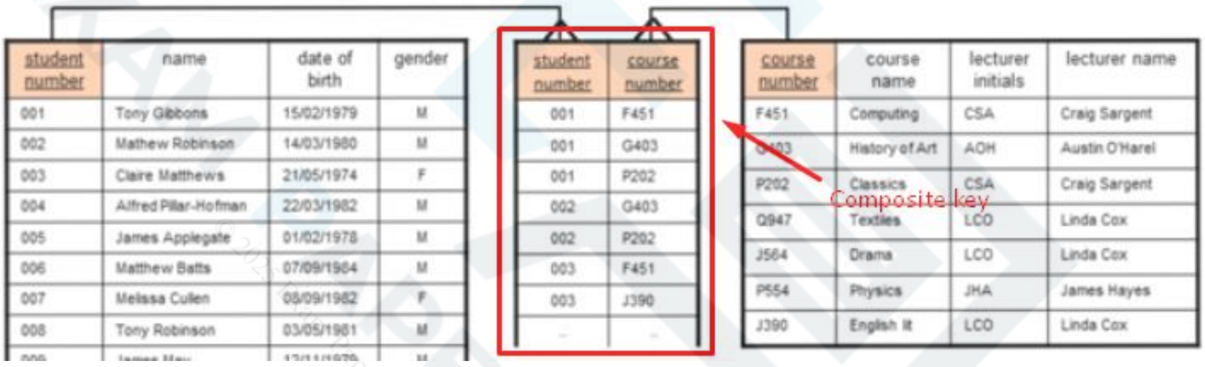
So that means you remove those three attributes, yup you are removing almost half of the table



name	date of birth	gender	course number
Tony Gibbons	15/02/1979	M	F451
Tony Gibbons	15/02/1979	M	G403
Tony Gibbons	15/02/1979	M	P202
Mathew Robinson	14/03/1980	M	G403
Mathew Robinson	14/03/1980	M	Q947
Mathew Robinson	14/03/1980	M	P202
Claire Matthews	21/05/1974	F	F451
Claire Matthews	21/05/1974	F	J564
Claire Matthews	21/05/1974	F	P554
Alfred Pillar-Hofman	22/03/1982	M	P202
Alfred Pillar-Hofman	22/03/1982	M	H544

And the final thing you need to do is convert the database into 3NF format, to do that you:

- Check to see if the database is already in 2NF form
- Remove any non-key dependencies (This means remove any attributes which are determined by an attribute not part of a composite key


Composite key

student number	name	date of birth	gender
001	Tony Gibbons	15/02/1979	M
002	Mathew Robinson	14/03/1980	M
003	Claire Matthews	21/05/1974	F
004	Alfred Pillar-Hofman	22/03/1982	M
005	James Applegate	01/02/1978	M
006	Matthew Batts	07/09/1984	M
007	Melissa Cullen	08/09/1982	F
008	Tony Robinson	03/05/1981	M
009	James May	12/11/1979	M

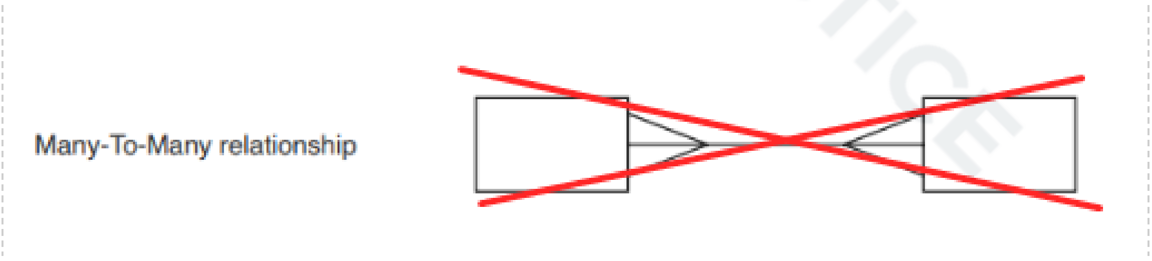
student number	course number
001	F451
001	G403
001	P202
002	G403
002	P202
003	F451
003	J390
...	...

course number	course name	lecturer initials	lecturer name
F451	Computing	CSA	Craig Sargent
G403	History of Art	AOH	Austin O'Harel
P202	Classics	CSA	Craig Sargent
Q947	Textiles	LCO	Linda Cox
J564	Drama	LCO	Linda Cox
P554	Physics	JHA	James Hayes
J390	English lit	LCO	Linda Cox

Final 3NF version of the database

Issues with many to many relationships

You cannot have many to many relationships because they are not normalised. They cause data redundancy



Yes, there is a way to do many to many relationships but still, you should not use them OR approve them

Spec reference:

(d) SQL – Interpret and modify. See appendix 5d.

You need to know the following commands of SQL for your exams:

`SELECT` (including nested SELECTs)

`FROM`

`WHERE`

`LIKE`

`AND`

`OR`

`DELETE`

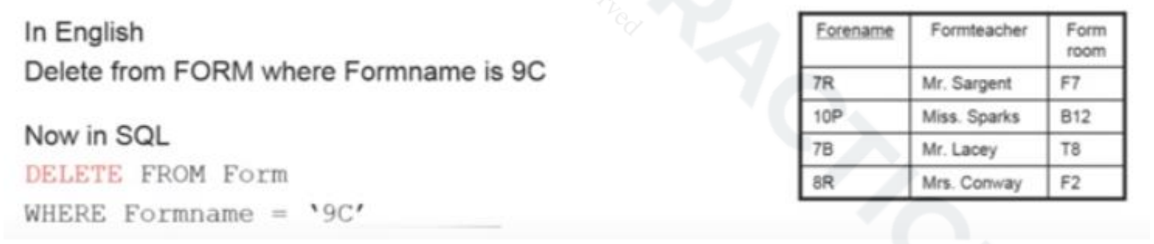
`INSERT`

`DROP`

`JOIN` (Which is equivalent to INNER JOIN, there is no expectation to know about outer, left and right joins)

`WILDCARDS` (Learners should be familiar in the use of '*' and '%' as a wildcard to facilitate searching and matching where appropriate)

Here is an example of how to delete data from forms using SQL:



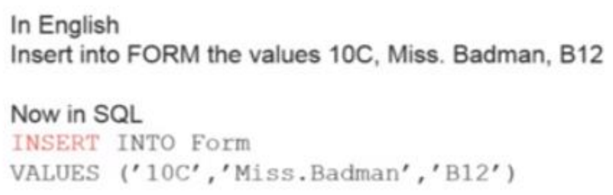
In English
Delete from FORM where Formname is 9C

Now in SQL

```
DELETE FROM Form
WHERE Formname = '9C'
```

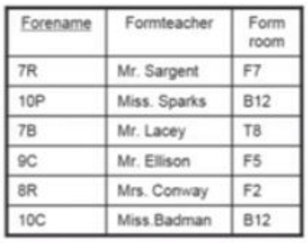
Forename	Formteacher	Form room
7R	Mr. Sargent	F7
10P	Miss. Sparks	B12
7B	Mr. Lacey	T8
8R	Mrs. Conway	F2

And how to add values into a form:



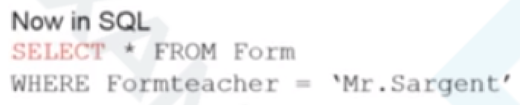
In English
Insert into FORM the values 10C, Miss. Badman, B12

Now in SQL
```
INSERT INTO Form
VALUES ('10C','Miss.Badman','B12')
```

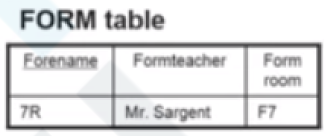


Forename	Formteacher	Form room
7R	Mr. Sargent	F7
10P	Miss. Sparks	B12
7B	Mr. Lacey	T8
9C	Mr. Ellison	F5
8R	Mrs. Conway	F2
10C	Miss Badman	B12

And search values:



Now in SQL
```
SELECT * FROM Form
WHERE Formteacher = 'Mr.Sargent'
```



FORM table

Forename	Formteacher	Form room
7R	Mr. Sargent	F7

The wildcard * means "all fields" and HAS to be used when searching or matching **all fields** of a particular record unless otherwise

And by the way you do know each time in these examples they say INSERT, DELETE or whatever into Form the table in SQL is called a Form? It is not referring to the form as in the form classes, oh no no no....

Spec reference:

(e) Referential integrity.

Referential integrity means deleting or modifying records in multiple tables that have the same primary key, if for example you delete one record, all records from all tables with the same primary key should also be deleted.

A cascade delete or modifying restrain is applied to a primary key of a record so that all records from all tables with the same primary key can be deleted or modified.

Why is it used? It's simply much quicker to alter or delete multiple records in this way

Where it might be used? It can be used to delete the records of a student who left the school, the records of that student will be stored in multiple tables in the school's database so they all have to be deleted and doing this one by one will take so, SO long

Spec reference:

(f) Transaction processing, ACID (Atomicity, Consistency, Isolation, Durability), record locking and redundancy.

No we are not talking about hydrochloric acid, just stay out of London

Transaction processing is when the processing of an instruction is split into individual operations. Each operation must succeed or fail as a whole unit

Transaction processing must follow the rules of CRUD

Create

Read

Update

Delete

In order to endure data integrity in a database, whenever a change in a database is made they have to follow the ACID rule which stand for Atomicity, Consistency, Isolation and Durability. Let's run through each of them:

Atomicity – A change in the database must be processed completely or not at all. A half-change must not be saved back into the database

Consistency – A change in the database must retain the overall state of the database

Isolation – A change in the database must be processed in isolation so that other processes or users do not have access to the concerned data used in the process

Durability – A change in the database must not be lost due to a **system** failure

Well not that big of a system failure, and what is this guy doing with a...stick in the middle of a huge blazing fire?

Spec reference:

(a) Characteristics of networks and the importance of protocols and standards.

Networks are computers connected together

Let's have a look at some of the benefits and drawbacks of networks

Benefits:

- Allows sharing of resources like printers
- Allows the sharing of data stored on hard disks
- It's much easier to back up data from a central server than on individual machines
- Allows different computers to be connected together

Drawbacks:

- It’s hard to make a network safe from outside threats
- The user may become too dependent on the network
- If there is a lot of data traffic in the network, the performance can degrade
- If the network goes down, various resources cannot be accessed

Protocols are a set of rules governing communication in a network. An example of some of the rules might be that the devices involved in the communication must all have the same bit rate, use the same transport medium, use the same character set or make sure their computers are compatible with different software.

These are the different protocols:

- TCP (Transmission control protocol), it is used with the IP and makes sure that packet transmission is error free
- IP (Internet Protocol), this is involved when data packets are sent across the internet between different routers (Routers are devices which receive data, you will learn more about it later on)
- FTP (File transfer protocol), used by servers to download files from the internet
- HTTP (Hypertext Transfer Protocol), used by web servers to transfer webpages

- SMTP (Simple Mail Transfer Protocol), used by servers when transferring emails across the internet

Spec reference:

(b) The internet structure:

• The TCP/IP Stack.

• DNS

• Protocol layering.

• LANs and WANs.

• Packet and circuit switching.

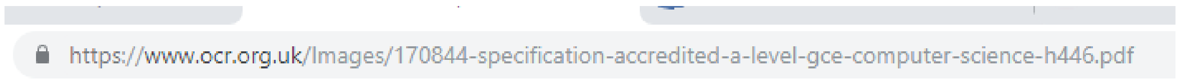
The TCP/IP stack

There are four stacks you need to know about:

- **Application**, this is where data is shown to the user on an application (Like Fortnite)
- **Transport**, keeps track of segments within a network, checking if packet transmission and switching was successful
- **Internet**, carries the data packets
- **Host to Host**, this just checks on network devices and transmission medium

DNS

DNS are names given on resources of a network, most notably in the internet:


https://www.ocr.org.uk/Images/170844-specification-accredited-a-level-gce-computer-science-h446.pdf

It is structured into a hierarchy, with the top **domain** on the top (obviously) followed by the sub-domains and the name of the resource

Why is this used? To replace the IP address of the device with a more, user friendly name.

<u>Protocol layering</u>

This is pretty similar to the TCP/IP stack address: the layering of the protocol are: Physical, Data Link, Network, Transport, Session, Presentation and Application with Physical at the bottom.

Physical – This is just network devices and transmission medium

Data link – This takes care of control access, error checking and correction

Network – This transmits the packets

Transport – This checks on the segments of the network, ensuring that packet transmission has been successful

Session – This is where a session is created, managed and ended. It also decides on the type of transmission like half-duplex, full-duplex or simplex

Presentation - This controls the interaction between the data sent and the need of data from the operating system, so packets are decrypted or encrypted

Application – This is where data is moved from the presentation layer to the application where it is shown to the user

LANs and WANs

Local area network – This is where computers are all connected together in a small geographical area, like a building site or an office

Wide area network – This is where computers are all connected together in a wider geographical area, like the world or between different counties

Local area network is much safer than wide area network

Packet and circuit switching

There are two ways of sending packets: packet switching or circuit switching:

In circuit switching there is only one route and all packets are carried through this same route so the route is already established.

In packet switching however the packets take different routes, each packet has a header of information, the address of the node (The computer who is going to receive the packets) is read out and the best route for each packet is found so all the packets travel through different routes. This means some packets will take longer to arrive at their destination than others

Packet switching is more efficient and if any packets are missing, the **receiving device** can simply request the missing packets from the **sending device**

Spec reference:

(c) Network security and threats, use of firewalls, proxies and encryption.

There are three ways to protect your network from threats as already shown from the spec point reference above:

Firewalls – These protect the network from unauthorised access and check files for viruses

Proxies – They act as a hub where they process user requests, this avoids direct contact with the user's computer and the internet

Encryption – This encrypts messages so unauthorised people can't read them

Spec reference:

(d) Network hardware.

Network Interface Card – This sends electronic signals across the network

Router – This transmits data between different networks

Switches – They connect devices together on a network

Hubs – Same as switches (No no like literally, I am not joking)

Wireless Access Point – Connects a network to a WIFI network

Spec reference:

(e) Client-server and peer to peer.

This section focuses on how networks may be structured, if you did Computing GCSE then you may be familiar with some but this is just a little bit more detailed, read on...

In client-server all the computers are connected together through a single server. The benefit is that if one computer goes down then the network will be unaffected. However, a drawback will be that if the server goes down, then the whole network goes down.

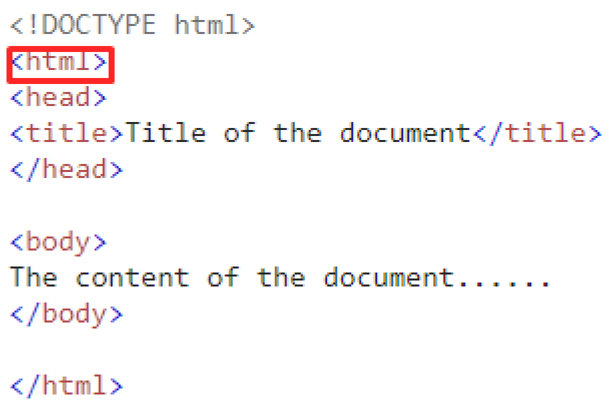
In peer to peer all the computers act both as the client and as the server to each other. The benefits are that it's cheap and the files can be easily shared. However, this network allows for mass piracy of content

Spec reference:

(a) HTML, CSS and JavaScript. See appendix 5d.

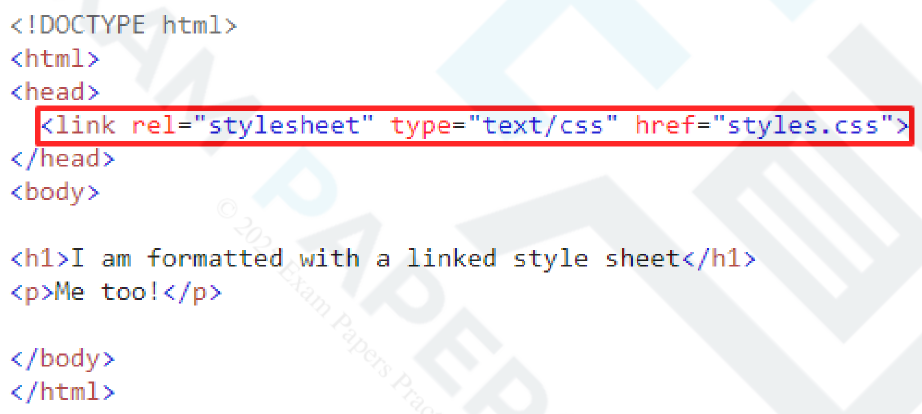
Here you need to learn tags about HTML, CSS and JavaScript, let's start with HTML:

<html> This defines that the page will be written in HTML

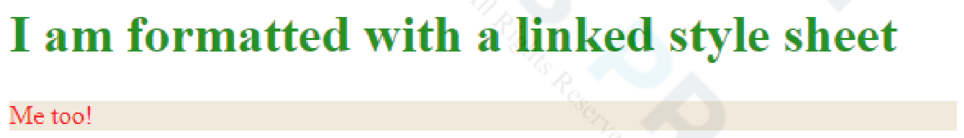


```
<!DOCTYPE html>
<html>
<head>
<title>Title of the document</title>
</head>

<body>
The content of the document......
</body>

</html>
```

<link> This is used to link a CSS file, like this:



```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>

<h1>I am formatted with a linked style sheet</h1>
<p>Me too!</p>

</body>
</html>
```

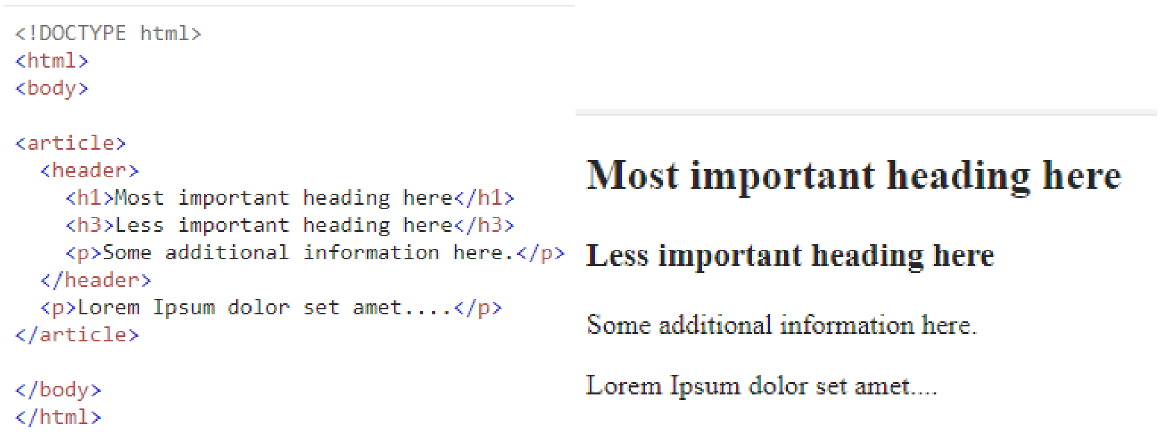


I am formatted with a linked style sheet

Me too!

The CSS file linked in here affects both the header (Show between the <h1></h1> tags) and the paragraph (Shown by the <p></p> tags) but you don't need to worry too much about it

<head> This just defines a header



```
<!DOCTYPE html>
<html>
<body>

<article>
  <header>
    <h1>Most important heading here</h1>
    <h3>Less important heading here</h3>
    <p>Some additional information here.</p>
  </header>
  <p>Lorem Ipsum dolor set amet....</p>
</article>

</body>
</html>
```

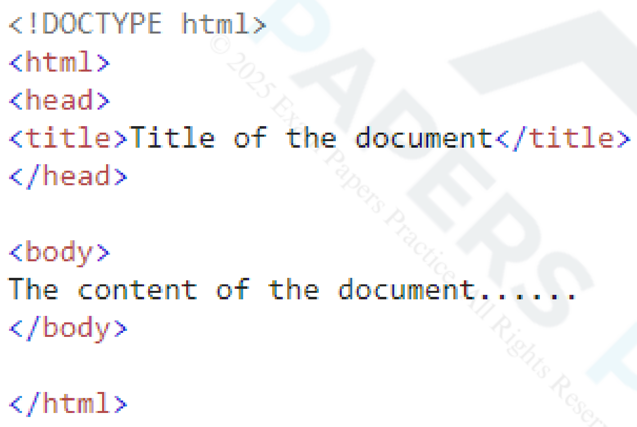
Most important heading here

Less important heading here

Some additional information here.

Lorem Ipsum dolor set amet....

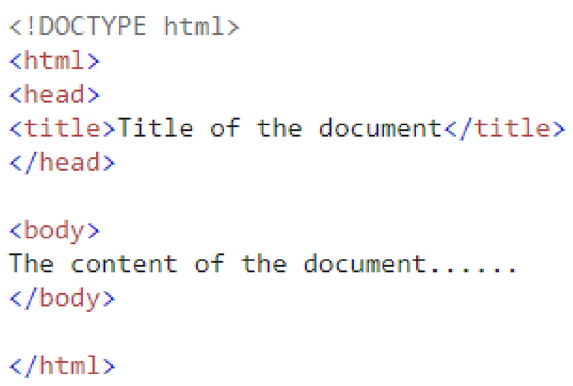
<title> This defines a title



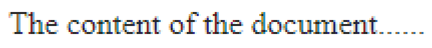
```
<!DOCTYPE html>
<html>
<head>
<title>Title of the document</title>
</head>

<body>
The content of the document......
</body>

</html>
```

The title will only appear on the tab the HTML code creates and not in the actual webpage, keep that in kind

<body> This defines the body of a HTML page, where the main content (Paragraphs, heading, lists etc..) will be put

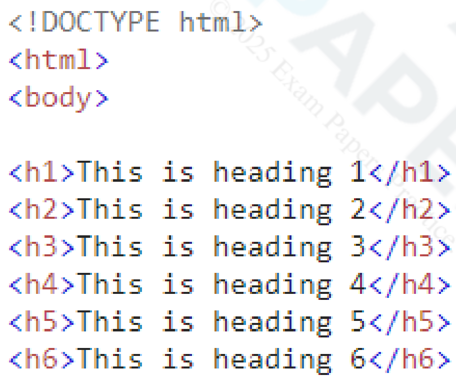


```
<!DOCTYPE html>
<html>
<head>
<title>Title of the document</title>
</head>

<body>
The content of the document......
</body>

</html>
```

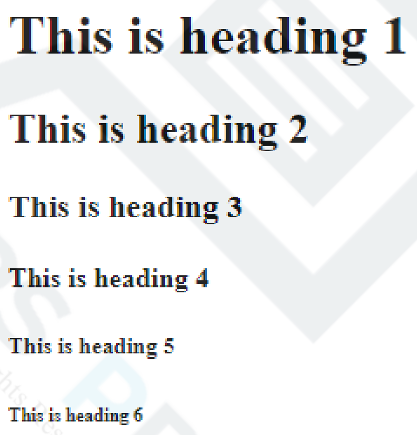


The content of the document......

<h1> <h2> <h3> These have exactly the same functions as the <header> tags except they just decide in what order should the headers be displayed and how big should they be, with <h1> displaying a header first and the largest header:

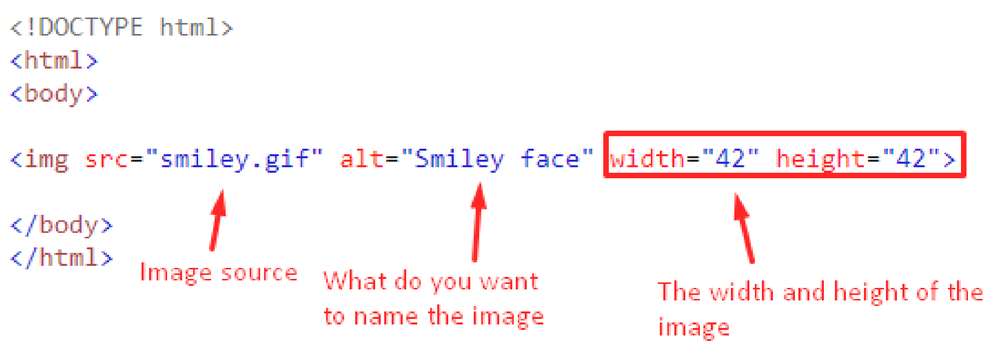


```
<!DOCTYPE html>
<html>
<body>

<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
<h4>This is heading 4</h4>
<h5>This is heading 5</h5>
<h6>This is heading 6</h6>
```



This is heading 1
This is heading 2
This is heading 3
This is heading 4
This is heading 5
This is heading 6

<img> These insert an image

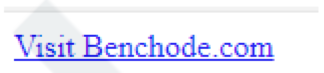


```
<!DOCTYPE html>
<html>
<body>

<img src="smiley.gif" alt="Smiley face" width="42" height="42">

</body>
</html>
```

Image source
What do you want to name the image
The width and height of the image

<a> Defines a hyperlink

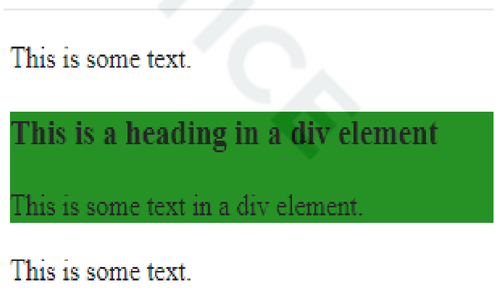
```
<!DOCTYPE html>
<html>
<body>

<a href="https://www.OCR.com">Visit Benchode.com</a>

</body>
</html>
```



Visit Benchode.com

<div> Defines a section of the HTML page, often used with a style attribute:

```
<!DOCTYPE html>
<html>
<body>

<p>This is some text.</p>

<div style="background-color:green">
  <h3>This is a heading in a div element</h3>
  <p>This is some text in a div element.</p>
</div>

<p>This is some text.</p>

</body>
</html>
```



This is some text.

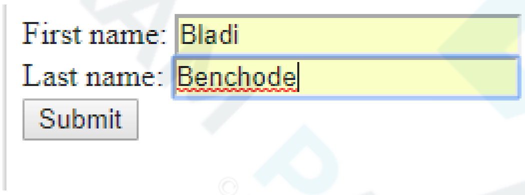
This is a heading in a div element

This is some text in a div element.

This is some text.

<form> Defines a section of the HTML of where user can input data

```
<!DOCTYPE html>
<html>
<body>

<form action="/action_page.php">
First name: <input type="text" name="FirstName"><br>
Last name: <input type="text" name="LastName"><br>
<input type="submit" value="Submit">
</form>


</body>
</html>
```



First name: Bladi
Last name: Benchode

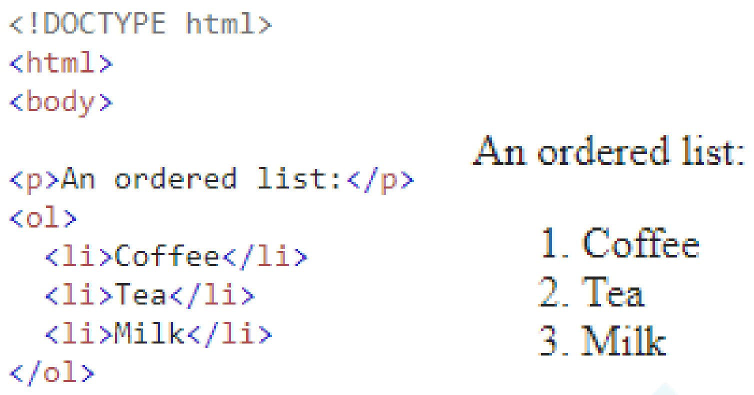
<input> Used with the <form> function to create a button or textbox, along with their names; see picture above

<p> Defines a paragraph

```
<!DOCTYPE html>
<html>
<body>


<p>Computing is boring, especially if it's done with OCR the devil</p>

</body>
</html>
```

```
Computing is boring, especially if it's done with OCR the devil
```

<li> Defines item in a list

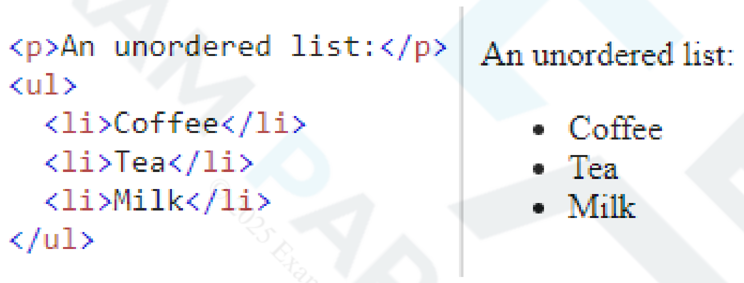


```
<!DOCTYPE html>
<html>
<body>

<p>An ordered list:</p>
<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

An ordered list:

1. Coffee
2. Tea
3. Milk

<ul> Defines an <u>unordered</u> list

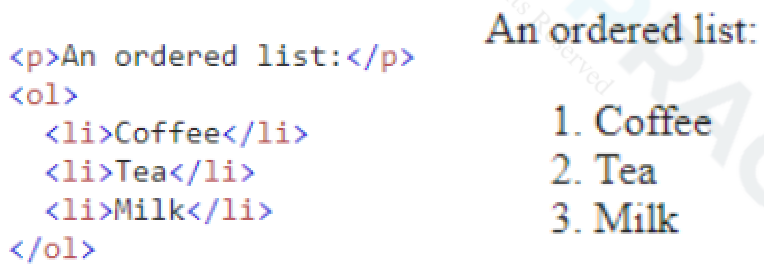


```
<p>An unordered list:</p>
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

An unordered list:

- Coffee
- Tea
- Milk

<ol> Defines an <u>ordered</u> list

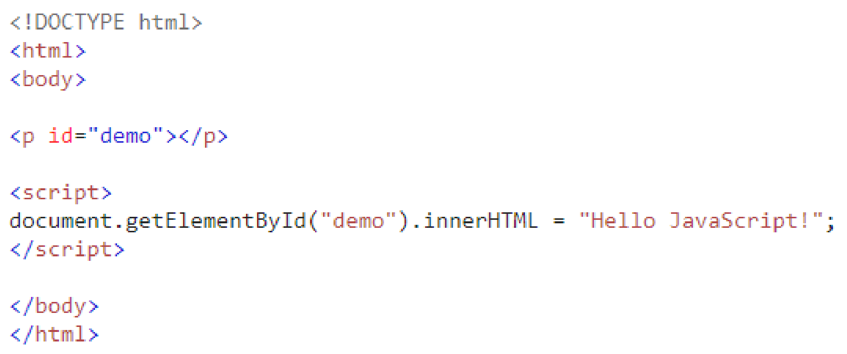


```
<p>An ordered list:</p>
<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

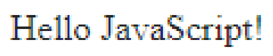
An ordered list:

1. Coffee
2. Tea
3. Milk

<script> Defines a section of JavaScript



```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "Hello JavaScript!";
</script>

</body>
</html>
```



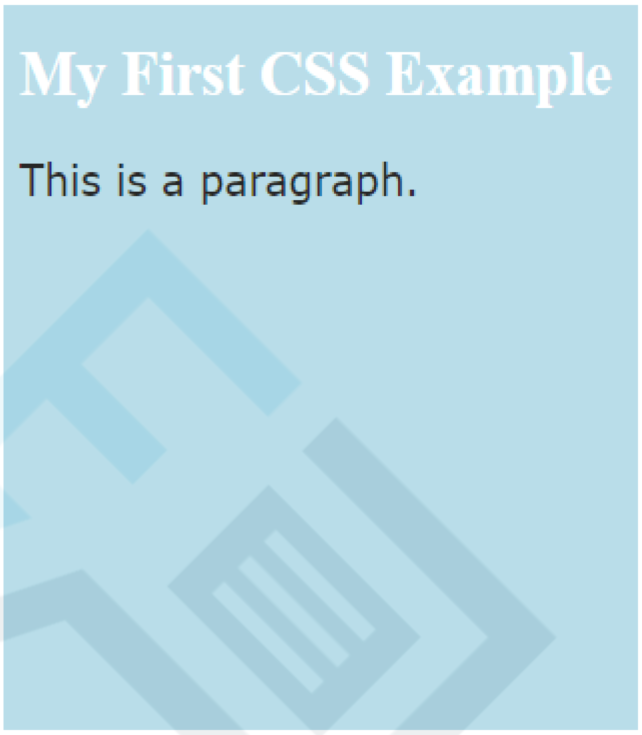
```
Hello JavaScript!
```

<u>CSS</u>

CSS is slightly easier to understand and there is not much to understand, CSS is used to style webpages, HTML is used to create them but now with CSS we are styling them.

I will just paste a picture here and hopefully you should understand what is going on and how to use CSS in JavaScript:

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
  background-color: lightblue;
}

h1 {
  color: white;
  text-align: center;
}

p {
  font-family: verdana;
  font-size: 20px;
}
</style>
</head>
<body>

<h1>My First CSS Example</h1>
<p>This is a paragraph.</p>

</body>
</html>
```



My First CSS Example

This is a paragraph.

When you are adding CSS always make sure you add them between the tags <style></style> as these are used to define CSS

Also you need to be familiar with these commands:

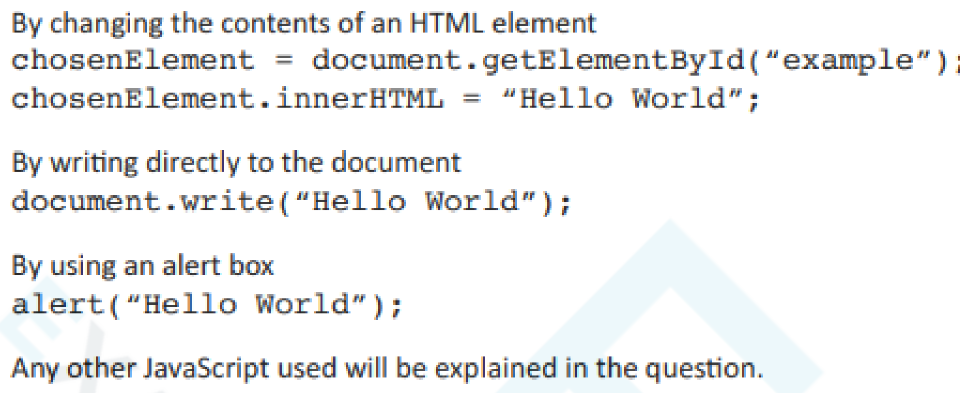
```
background-color
border-color
border-style
border-width
color with named and hex colours
font-family
font-size
height
width
```

Any other properties used will be explained in the question.

The commands are exactly what they say they are

Javascript

All you need to know for Javascript are these commands:



By changing the contents of an HTML element

```
chosenElement = document.getElementById("example");
chosenElement.innerHTML = "Hello World";
```

By writing directly to the document

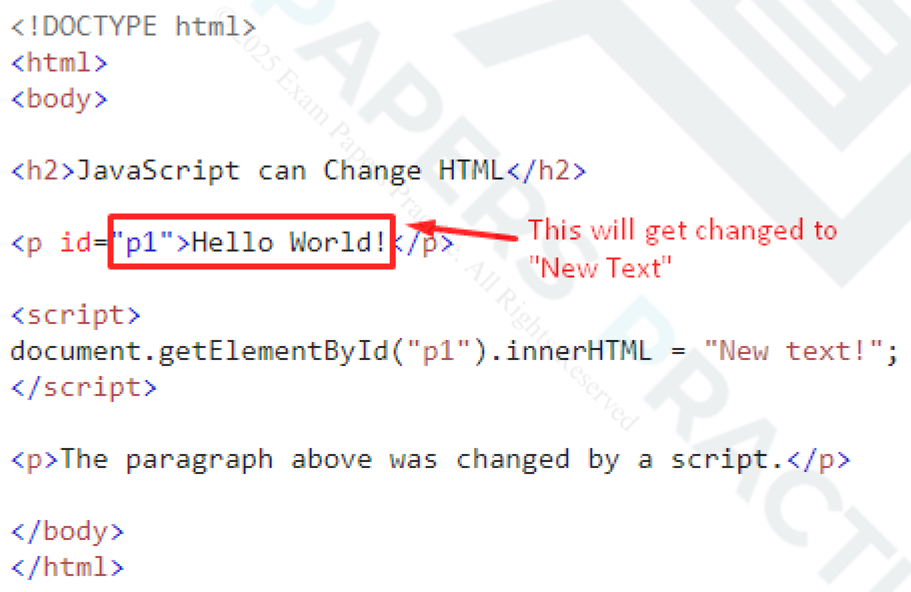
```
document.write("Hello World");
```

By using an alert box

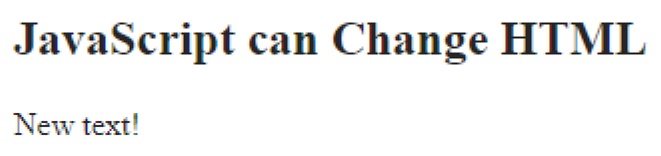
```
alert("Hello World");
```

Any other JavaScript used will be explained in the question.

Let's go through each of them:

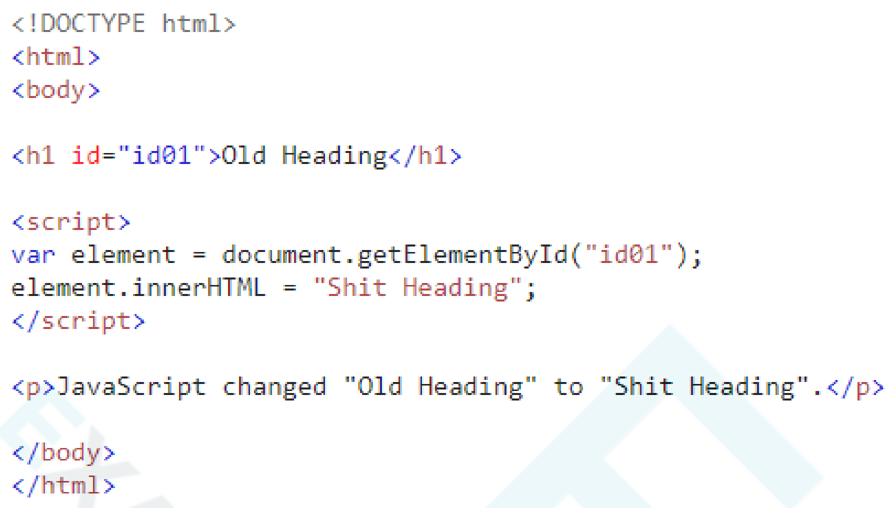


```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript can Change HTML</h2>

<p id="p1">Hello World!</p>

<script>
document.getElementById("p1").innerHTML = "New text!";
</script>

<p>The paragraph above was changed by a script.</p>

</body>
</html>
```

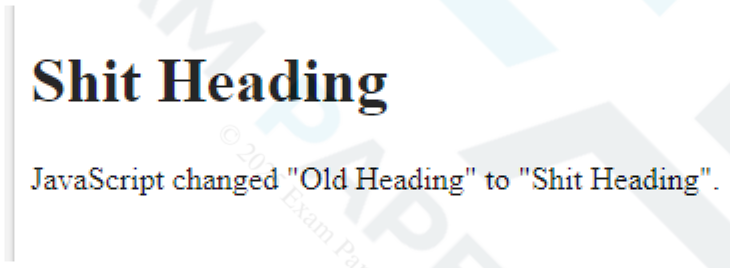


```
JavaScript can Change HTML

New text!
```

And the other way of changing HTML content with Javascript:

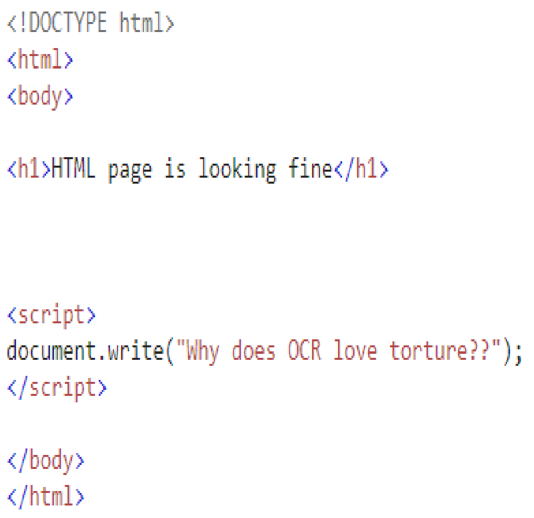


```
<!DOCTYPE html>
<html>
<body>

<h1 id="id01">Old Heading</h1>

<script>
var element = document.getElementById("id01");
element.innerHTML = "Shit Heading";
</script>

<p>JavaScript changed "Old Heading" to "Shit Heading".</p>

</body>
</html>
```

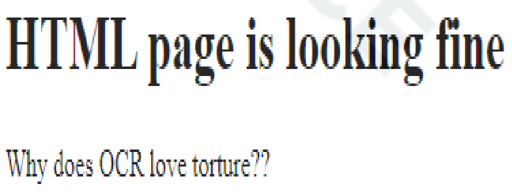


```
Shit Heading

JavaScript changed "Old Heading" to "Shit Heading".
```

Writing into a HTML document using Javascript:



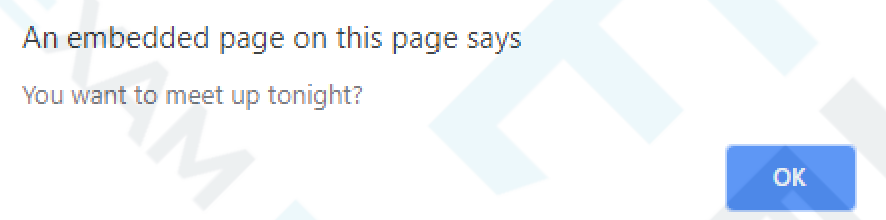
```
<!DOCTYPE html>
<html>
<body>

<h1>HTML page is looking fine</h1>



<script>
document.write("Why does OCR love torture??");
</script>

</body>
</html>
```



```
HTML page is looking fine

Why does OCR love torture??
```

And finally creating a HTML alter box using Javascript:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Alert</h2>

<script>
alert("You want to meet up tonight?");
</script>

</body>
</html>
```


An embedded page on this page says

You want to meet up tonight?

Spec reference:

(b) Search engine indexing

Search engine indexing is when a search engine like Google runs a user's request (To look up porn or something) against its own index of webpages. It's faster to run the user's request in this way but the index has to be constantly updated.

How is the index created? Programs called "crawlers" **travel** around the World Wide Web, indexing letters they find and mapping out (planning the details of something) links between webpages from internal and external links.

Spec reference:

(c) PageRank algorithm

This section will focus on how exactly search engines like Google display your search results like your favourite types of porn on your screen in like a few seconds. This is the job of the PageRank algorithm

So how does the PageRank algorithm work? Here are the steps:

1. The algorithm assigns a rank to a page based on the user's search terms
2. The algorithm ignores the spelling mistakes
3. The algorithm attempts to deal with millions of results usefully (Like it has to sort out all the million and millions of different results that come up after a search)
4. The algorithm receives inward links and makes use of them
5. The algorithm assigns each inward link a rank
6. The algorithm regularly recalculates the ranks of the pages

Spec reference:

(d) Server and client side processing.

Here we will focus on how your search results are processed, there are two ways to do this as mentioned in the spec reference:

Client-side processing:

This is done by the Javascript. The server has a reduced web traffic flow so it has less processing to do which means there are fewer delays in response time from the server. The data can then be validated by the client (you)

Server- side processing:

Sensitive data will never be processed on the client side so the querying of databases in done on the server. The databases are then stored on the server. The server may slow down if it's carrying out lots of processing

Well done once again! You have come to the end of the third section of A level Computer Science, only two more to go and you will be finally free! Oh and after that you have to learn stuff for your second paper……

Data types, structures and algorithms

Spec reference:

(a) Primitive data types, integer, real/floating point, character, string and Boolean.

This section is piss easy, just read on….

Integer – This is just a number like 7

Real/floating point – This is a decimal number like 8.7

Character – This is a single…character lol like 5 or $ or L or even "   "(Space basically)

String – This is a word made out of characters like 6T74£$ or just words like HELLO

Boolean – This is just TRUE or FALSE

Spec reference:

(b) Represent positive integers in binary

This section is just about presenting positive numbers like 97 into binary form, if you did in GCSE Computing you should know it EASILY, and if you didn't….then well just research about it.

Go to Google and type in "Converting denary numbers into binary form"

Spec reference:

(c) Use of sign and magnitude and two's complement to represent negative numbers in binary.

Note: In order to understand this section you MUST first understand the previous spec point.

Here you just have to know that representing negative numbers in binary, the binary format ALWAYS starts with a 1 whilst with a positive number the binary format start with a 0.

If you want to convert say –93 into binary form you MUST place a 1 in your binary table and just remove 128 and place a 1 underneath its former place, the rest can be positive so here is a picture below:

Your normal binary table:

128	64	32	16	8	4	2	1

Now looks like this since we are trying to convert a negative number into binary:

+/-	64	32	16	8	4	2	1

Simply place a 1 on the start of the binary table:

+/-	64	32	16	8	4	2	1
1							

And start at 64, remember your number is negative so you have to take away numbers using the numbers in this binary table from the number you are trying to convert into binary:

+/-	64	32	16	8	4	2	1
1	1	0	1	1	1	0	1

And here is your answer: 11011101

As for positive numbers you do it the normal way (Just like in GCSE)

However what you did right here is created a binary number via sign and magnitude form, with two complement's form it is a little different.

You have a different table in two complement's form which is this one:

-128	64	32	16	8	4	2	1

As you can see, not much has changed, the only thing which has changed is that 128 ONLY is now negative. But the same rule of; if the number is positive, the binary form starts with a 0 and if the number is negative, the binary form starts with a 1 still applies

So, if you want to convent –93 into binary form using this method, you do it the same way as the sign and magnitude form except you include the –128 in your calculations obviously, here is a picture below with the result:

-128	64	32	16	8	4	2	1
1	0	1	0	0	0	1	1

Do not be confused with this mindset of "Do I add or subtract when it's a – and -?" You just subtract and whatever result you get in this case 35 (Just ignore what sign it should be) carry on as you would in sign and magnitude form

Spec reference:

(d) Addition and subtraction of binary integers

Again, a lot of you should be familiar of how to add binary numbers from GCSEs, if you forgot here is a little run down:

Rules for adding:

1 + 1 = 0 with a 1 carried

0 + 1 = 1

0 + 0 = 0

1 + 0 = 1

1 + 1 + 1 = 1 with a 1 carried

Now for subtracting here are the rules:

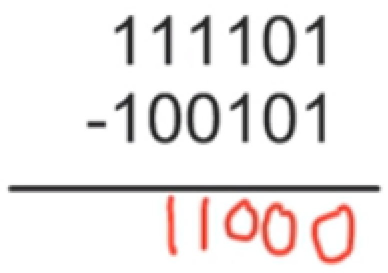
1 – 1 = Cancellation, this does not give a value and whatever value is left behind (1 or a 0) that is the result

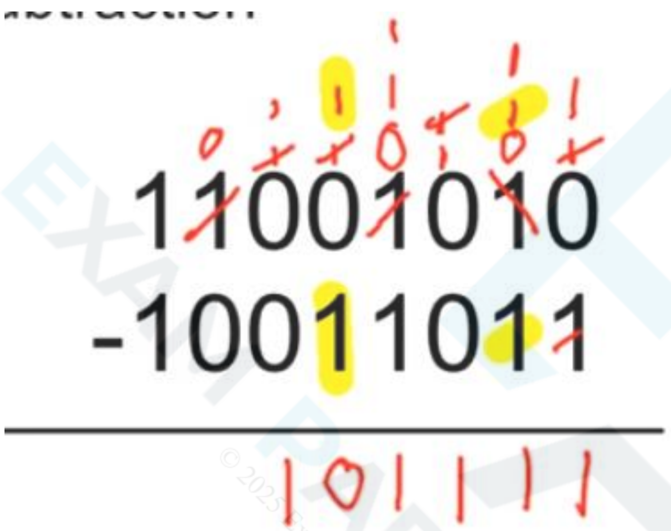
1 – 0 = 1

0 – 1 = Borrowing, this mean you have to borrow 2 1s from the 1 in the nearest column, you just have to borrow from ONE 1 to get two ones.

0 – 0 = 0

Here is an example to help you understand the rules better:





Keep in mind when borrowing from the nearest 1 that 1 becomes a 0 and the next column (backwards) receives the two ones

Spec reference:

(e) Represent positive integers in hexadecimal.

(f) Convert positive integers between binary hexadecimal and denary

Again, you should know this from GCSE, but just in case you forgot here is a rundown of the normal numbers to hex index:

Hex value	Normal number (Denary)
1	1
2	2
3	3
4	4

5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15

To convert a binary number into hexadecimal, you first split an 8 bit binary number into 2 and convert each of the now 4 bit binary numbers into hex value, keep in mind the binary table now used for each of the two binary values goes up to only 8.

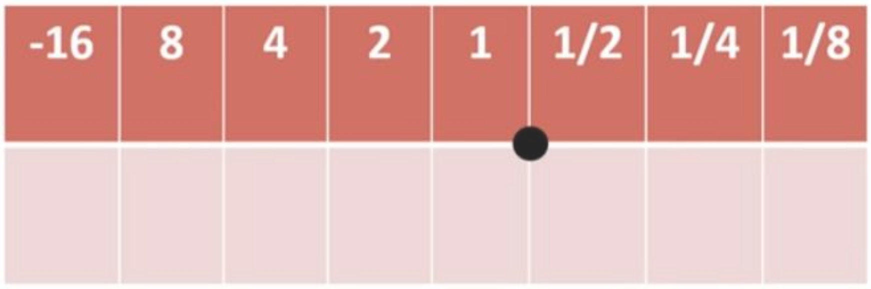
To do hex decimal to binary you do this process vice-versa, if you are still confused go research on this topic.

Spec reference:

(g) Representation and normalisation of floating point numbers in binary.

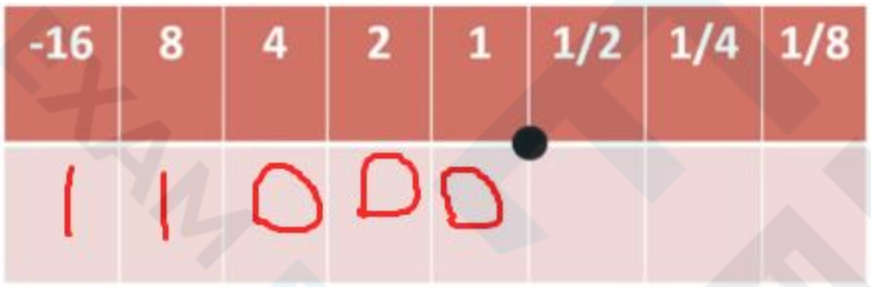
So far you have looked how to convert positive, whole numbers (referred as denary) into binary. In this section, you will look at how DECIMAL numbers, both positive and negative are represented in binary.

To convert decimal numbers into binary you use this binary table:



-16	8	4	2	1	1/2	1/4	1/8

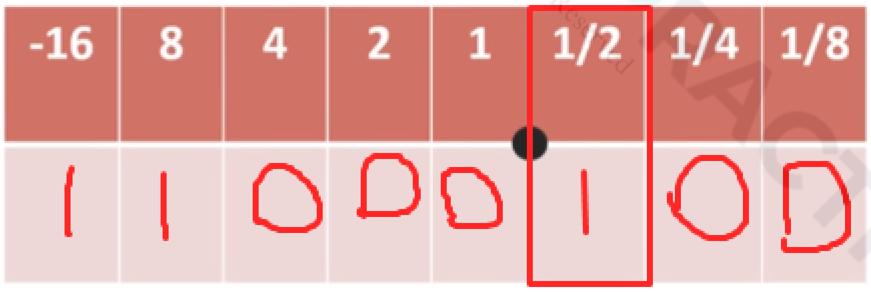
So, if I want to convert a NEGATIVE denary like –7.5 into binary what I do first is work out the nearest whole number and that is –8. So I have to get –8 in the number table:



-16	8	4	2	1	1/2	1/4	1/8
1	1	0	0	0			

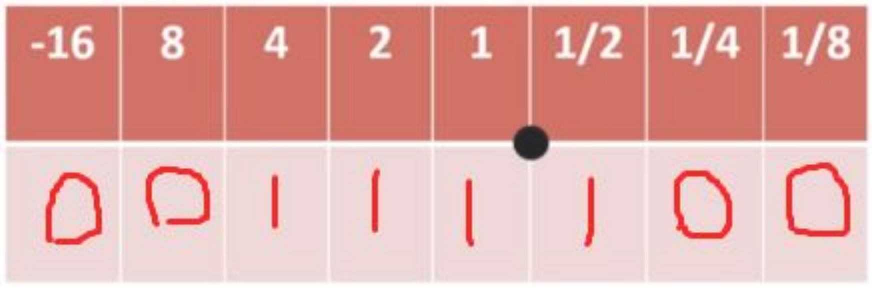
REMEMBER: With a negative number, its binary form ALWAYS starts with a 1.

So, I have my number –8, now what you need to do is subtract 8 from 7.5 and that gives you 0.5. So, looking at the decimal part of the binary table, you have to get 0.5 and that is easy:



-16	8	4	2	1	1/2	1/4	1/8
1	1	0	0	0	1	0	0

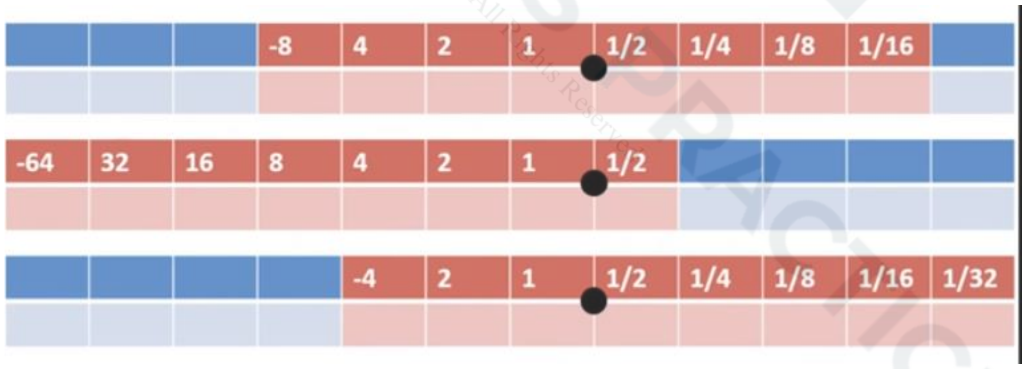
With a positive number, it's much easier, remember a positive number's binary form starts with 0 so if I want to convert positive 7.5 into binary, I simply try to get the number 7 in the number part of the binary table, and 0.5 in the decimal part of the binary table:



-16	8	4	2	1	1/2	1/4	1/8
0	0	1	1	1	1	0	0

You think this is your final answer? Oh no I am sorry to say! In the question you will often be asked to “normalise” your answer in the table in the form of the mantissa and exponent and you will see how to do that very soon. The reason why they ask you to normalise is because you can’t really tell if the number is decimal or not just by giving its binary value as it is in the table.

However, in the question they might give you a number to convert which just won’t fit with this table, so you need to increase the size of your number table, both with the section of whole numbers and the decimal (fraction) ones like this:



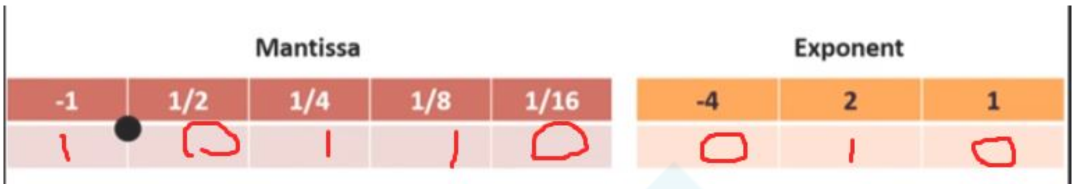
			-8	4	2	1	1/2	1/4	1/8	1/16	

-64	32	16	8	4	2	1	1/2				

				-4	2	1	1/2	1/4	1/8	1/16	1/32

You can kind of see where floating point gets its name from, the number line always floats (Like changes its size) depending on the number to convert into binary. To extend the part which contains the fractions, you always double the denominator as seen here with 1/2, 1/4, 1/8, 1/16, and so on. Also, in the whole number side, the biggest number in that side becomes a negative as seen with –8, -64 and –4. The reason being so that you can convert both positive and negative numbers into binary.

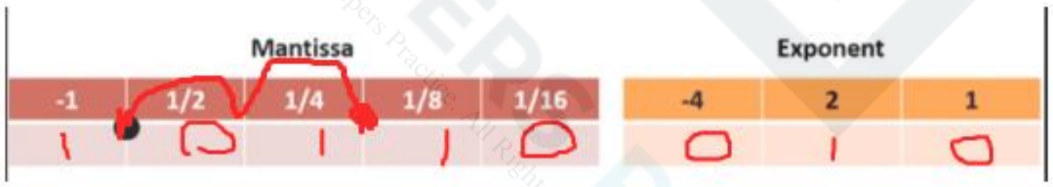
You also need to learn what the exponent and the mantissa:



Mantissa					Exponent		
-1	1/2	1/4	1/8	1/16	-4	2	1
1 ●	0	1	1	0	0	1	0

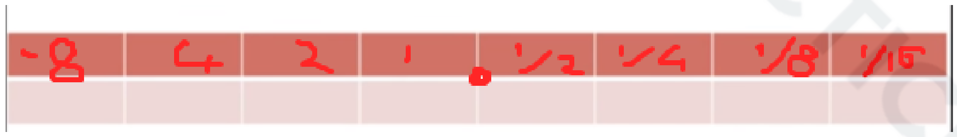
The exponent is always worked out using the orange table in the picture, the exponent tells how far the point in the mantissa table should move. The mantissa table stores the actual binary number. If the exponent is negative, the point moves towards the left, if it's positive it moves towards the right.

So an exponent of 2 means the point will move two places to the right:



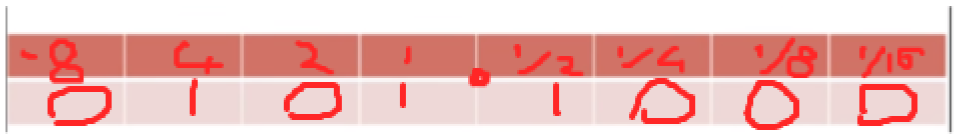
Mantissa					Exponent		
-1	1/2	1/4	1/8	1/16	-4	2	1
1	0	1	1 ●	0	0	1	0

So now we get a new binary table:



-8	4	2	1	●	1/2	1/4	1/8	1/16

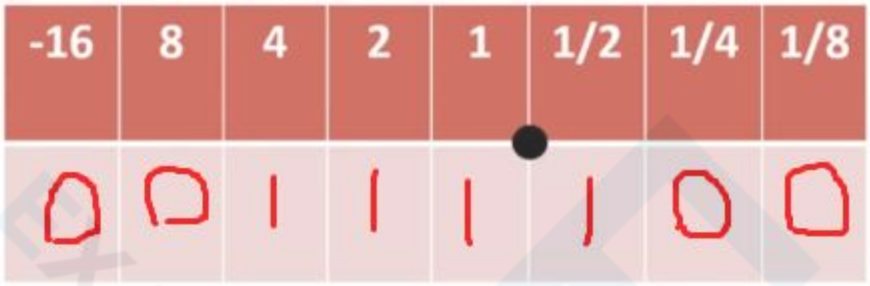
We then put our mantissa value into this number table according to the new position of the floating point:



-8	4	2	1	●	1/2	1/4	1/8	1/16
0	1	0	1		1	0	0	0

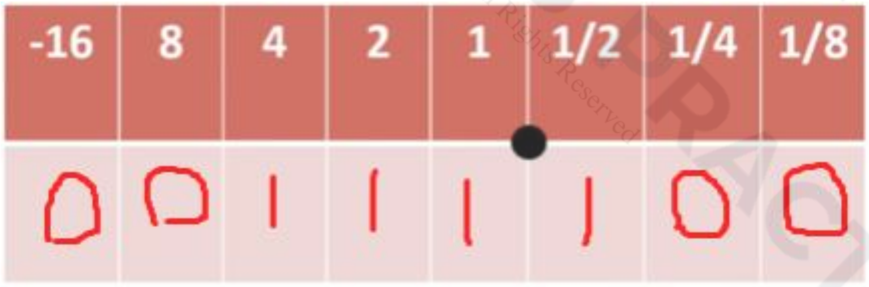
Now we just add all of these numbers together to get a floating point number, in this case 5.5 (As ½ = 0.5)

Now that you understood this, let's go back at normalisation mentioned earlier. So how do we normalise this binary number:



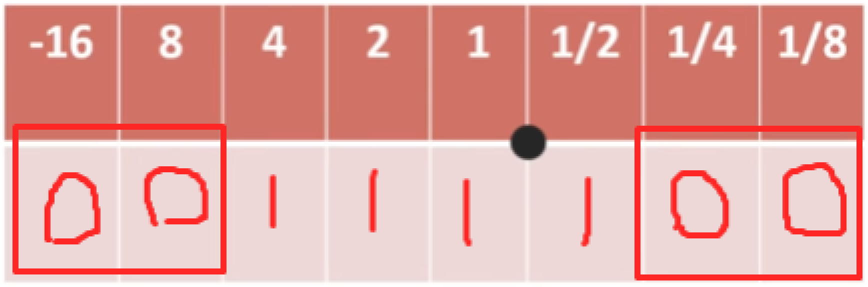
-16	8	4	2	1	1/2	1/4	1/8
0	0	1	1	1	1	0	0

First you need to know what decimal number is it for, does this binary number represent a positive or negative decimal number? This one presents a positive.
The reason for this check-up is because if the decimal number was negative you need to look for a non-stop flow of 1s in your table whereas if it was a positive decimal number, you need to look for a non-stop flow of 0s. Let me explain:



-16	8	4	2	1	1/2	1/4	1/8
0	0	1	1	1	1	0	0

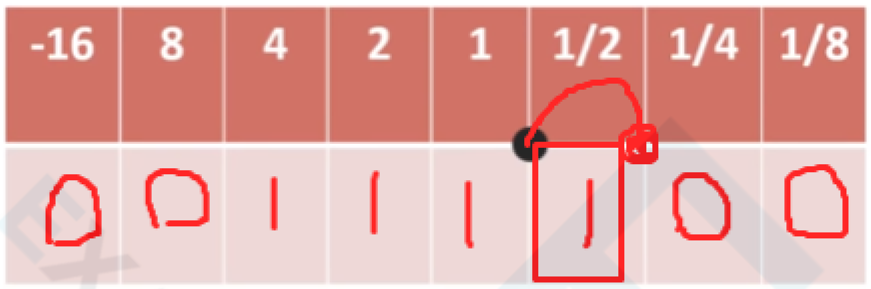
So there are continuous flows of 0s in both sides:



-16	8	4	2	1	1/2	1/4	1/8
0	0	1	1	1	1	0	0

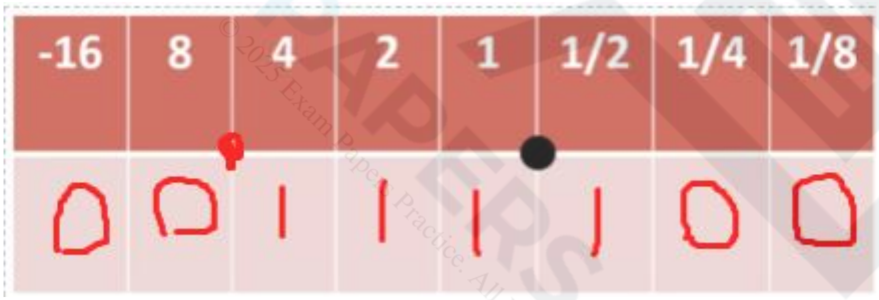
But you have to remember the new position of the floating point has to have the number 0 before it (With a negative decimal number, the new position of the floating point has to have the number 1 before it)

So if we move out floating point to the right:



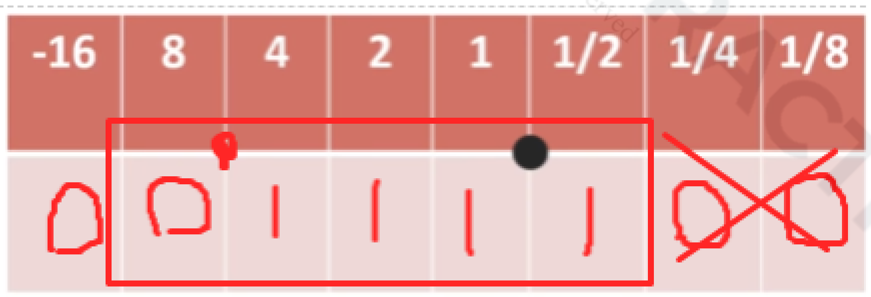
-16	8	4	2	1	1/2	1/4	1/8
0	0	1	1	1	1	0	0

This won't work as there is a 1 before it so we move our floating point to the left instead:



-16	8	4	2	1	1/2	1/4	1/8
0	0	1	1	1	1	0	0

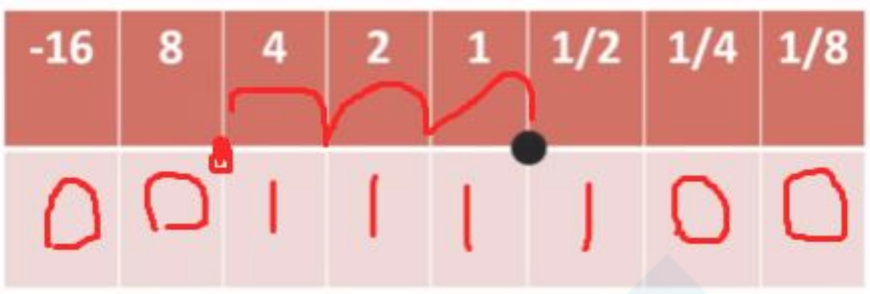
Now we find out our mantissa:



-16	8	4	2	1	1/2	1/4	1/8
0	0	1	1	1	1	0	0

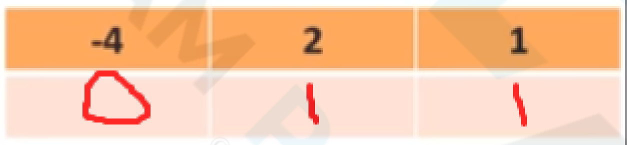
The reason why we ignore the **continuous** flow of 0s after the position of the new floating point is because they are not significant.

You have just worked out the mantissa which is as shown here 01111 but what of the exponent? Well that's easy to work out too. So, from your new position of the floating point, do you have to go right or left and how many times before reaching the old position of the floating point:



-16	8	4	2	1	1/2	1/4	1/8
0	0	1	1	1	1	0	0

Three places to the right so it is going to be exponent 3 and you need to write it in binary form, use the exponent table:

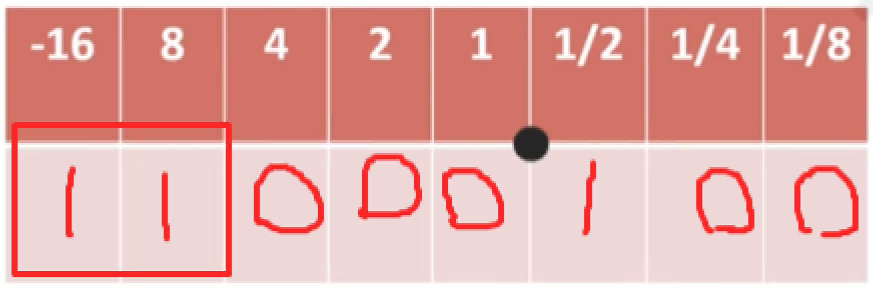


-4	2	1
0	1	1

And well done! You have your answer! So, the normalised form for 7.5 is 01111 011 with 01111 being the mantissa and 011 being the exponent.

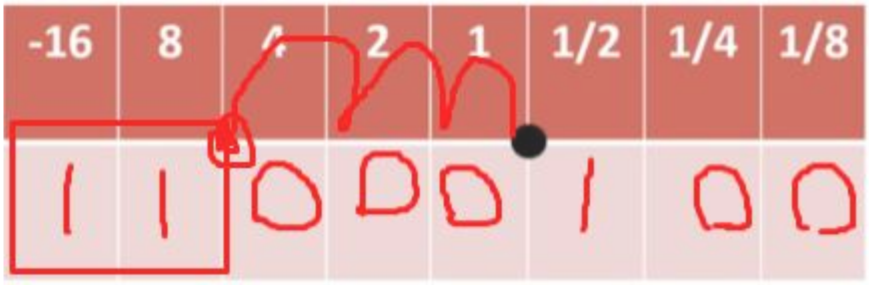
If we want to convert –7.5 into normalised form this is how it's done:

I look for a continuous flow of 1s:



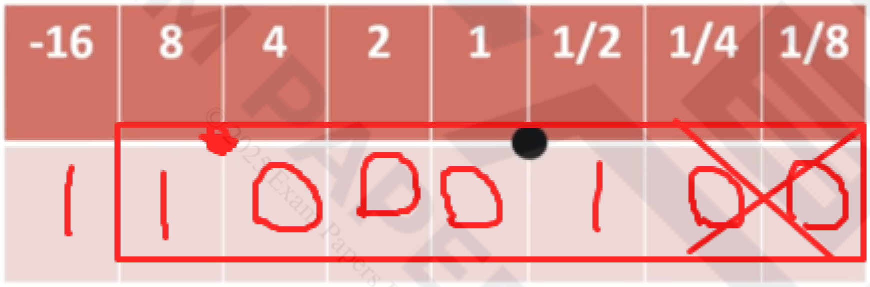
-16	8	4	2	1	1/2	1/4	1/8
1	1	0	0	0	1	0	0

So I move the floating point to the left up until the continuous flow of 1s:



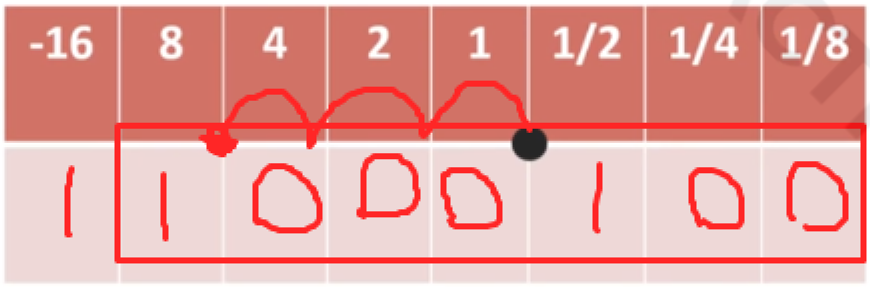
-16	8	4	2	1	1/2	1/4	1/8
1	1	0	0	0	1	0	0

So my mantissa is 10001, remember you ignore the **continuous** flow of 0s after the new position of the floating point because they are not significant



-16	8	4	2	1	1/2	1/4	1/8
1	1	0	0	0	1	~~0~~	~~0~~

And now to work out the exponent, how many times does the floating point have to move to reach its old position from its new position, and does it have to travel left or right?



-16	8	4	2	1	1/2	1/4	1/8
1	1	0	0	0	1	0	0

I would have to travel 3 places to the right so the exponent is 3, time to represent this in binary form using the exponent table:

-4	2	1
0	1	1

So the normalised version of floating point number –7.5 is 10001 011 with 10001 as the mantissa and 011 as the exponent

Note: When deciding how long your mantissa should be, unless it strictly says in the question your mantissa has to be this x number of bits long, calculate the mantissa as shown above

Spec reference:

(h) Floating point arithmetic, positive and negative numbers, addition and subtraction.

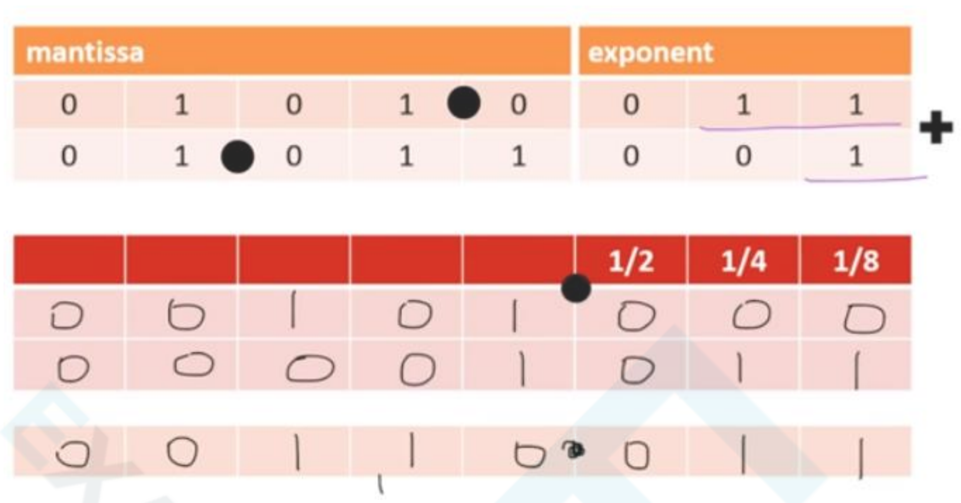
Once you master the previous spec point learning, learning this part will be pretty easy

To add floating point numbers in normalised form, your first of all get some floating point binary tables and work out their binary forms like this:

mantissa					exponent		
0	1	0	1 ●	0	0	1	1
0	1 ●	0	1	1	0	0	1

					1/2	1/4	1/8
0	0	1	0	1 ●	0	0	0
0	0	0	0	1	0	1	1

Now you just apply the rules of adding binary numbers to get your answer:

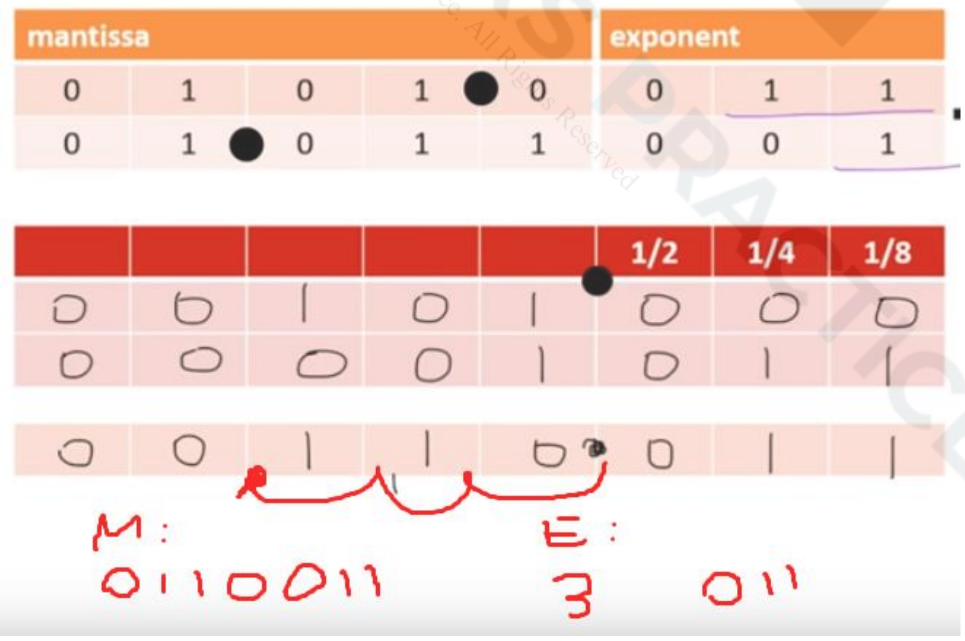


mantissa					exponent		
0	1	0	1 ●	0	0	1	1
0	1 ●	0	1	1	0	0	1

+

					1/2	1/4	1/8
0	0	1	0	1 ●	0	0	0
0	0	0	0	1	0	1	1
0	0	1	1	0 ●	0	1	1

Now you just have to normalise your answer, since we are adding numbers we will have to move the floating point to a point of continuous flow of 0s where the number before the new position of the floating point has to be 0 (Basically you apply the rules of normalisation for binary numbers of positive decimals)



mantissa					exponent		
0	1	0	1 ●	0	0	1	1
0	1 ●	0	1	1	0	0	1

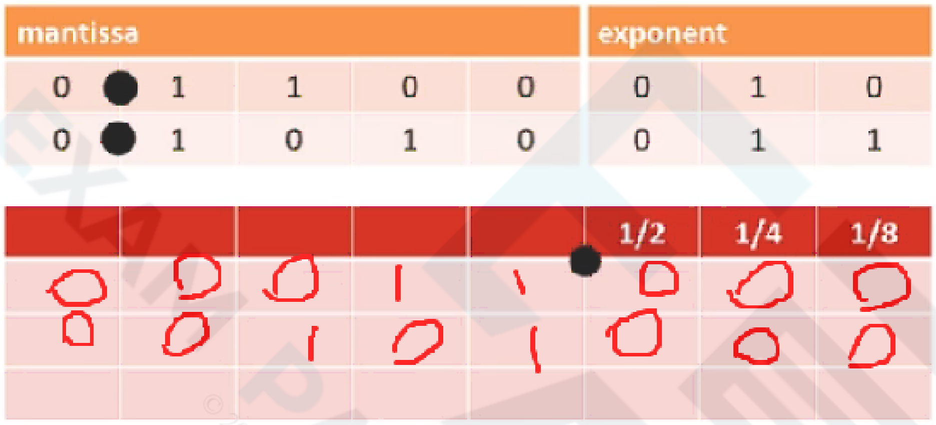
					1/2	1/4	1/8
0	0	1	0	1 ●	0	0	0
0	0	0	0	1	0	1	1
0	0	1	1	0 ●	0	1	1

M: 0110011

E: 3 011

So, the answer is 011011 011 with the mantissa being 0110011 and exponent being 011

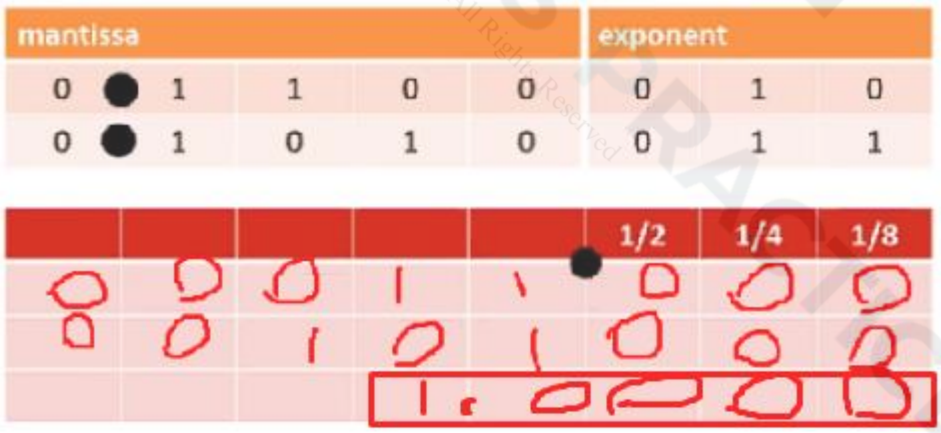
To subtract it's not very simple, you will have to convert both normalised forms into their decimal denary forms, work out the answer and turn that answer into normalised form.



mantissa						exponent		
0	●	1	1	0	0	0	1	0
0	●	1	0	1	0	0	1	1

					1/2	1/4	1/8
0	0	0	1	1 ●	0	0	0
0	0	1	0	1	0	0	0

This is basically 3 – 5 which is –2, the binary table you use should only be as large as the number of bits in the mantissa they give you. So convert –2 into normalised form:



mantissa						exponent		
0	●	1	1	0	0	0	1	0
0	●	1	0	1	0	0	1	1

					1/2	1/4	1/8
0	0	0	1	1 ●	0	0	0
0	0	1	0	1	0	0	0
			1.	0	0	0	0

As you can hopefully see, we used a smaller binary table to convert –2 into normalised form because there is no need for a larger binary table, so a binary table with –2 at the highest is enough. And now how many points to move, left or right from the new floating point position before returning to the original floating point position?

					1/2	1/4	1/8
0	0	0	1	1	0	0	0
0	0	1	0	1	0	0	0
			1.	0	0	0	0

We moved one place to the right so the exponent is 1.

Therefore the answer is 10000 001 with 10000 as the mantissa and 001 as the exponent

Spec reference:

(i) Bitwise manipulation and masks: shifts, combining with AND, OR, and XOR.

<u>Shifts</u>

A left shift is when you move a binary number on a binary table to the left:

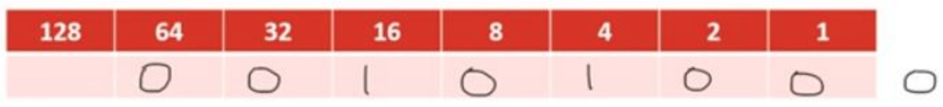
128	64	32	16	8	4	2	1
0	0	1	0	1	1	0	1

	128	64	32	16	8	4	2	1
0	0	1	0	1	1	0	1	

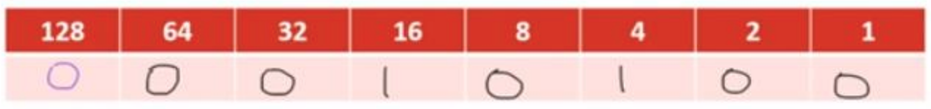
128	64	32	16	8	4	2	1
0	1	0	1	1	0	1	0

And the 0 simply gets lost, the empty space it left behind is replaced with a new 0

A right shift is when the binary number on a binary table shifts to the right:



128	64	32	16	8	4	2	1	
	0	0	1	0	1	0	0	0



128	64	32	16	8	4	2	1
0	0	0	1	0	1	0	0

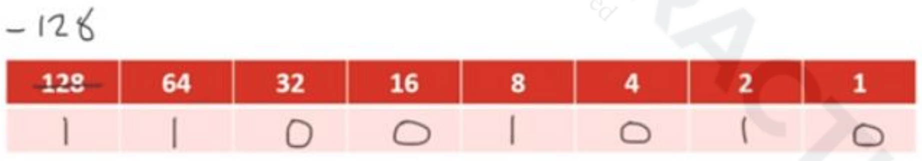
The 1 or 0 that are moved out are simply lost and their place is replaced with a 0. BUT this is a right **logical or unsigned** shift and its symbol is this:

Right shift (>>>)

With a right **arithmetic or signed shift** represented like this:

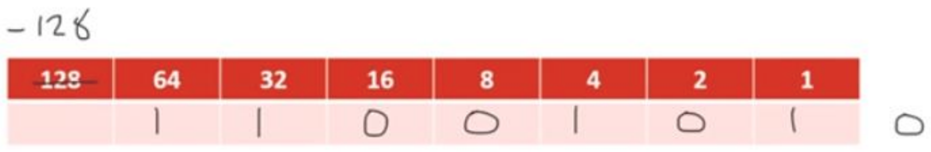
Right shift (>>)

The binary table has –128 or the largest number in the binary table is a negative:

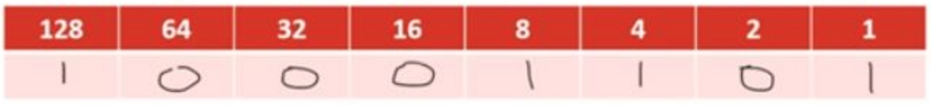


~~128~~ -128	64	32	16	8	4	2	1
1	1	0	0	1	0	1	0

So, when the binary number is shifted to the right, the 0 or 1 is lost but their place is replaced with a 1:

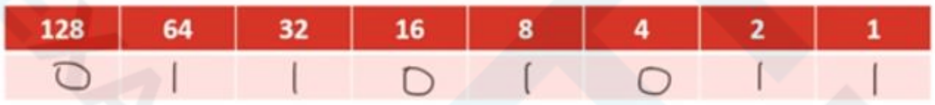


~~128~~ -128	64	32	16	8	4	2	1	
	1	1	0	0	1	0	1	0

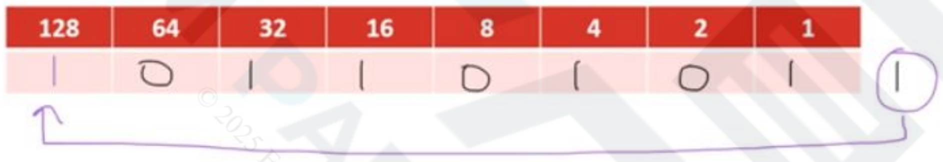


128	64	32	16	8	4	2	1
1	0	0	0	1	1	0	1

Circular shifting is as its name suggests, when the binary number in a binary table is moved to the right or left but the 1 or 0 that is squeezed out returns to the other side of the binary table:



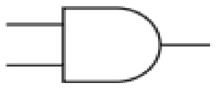
128	64	32	16	8	4	2	1
0	1	1	0	1	0	1	1



128	64	32	16	8	4	2	1	
1	0	1	1	0	1	0	1	1

Masks

A run-down of the logic gates if you forgot them, as well some new logic gates (There is only one of them lol):

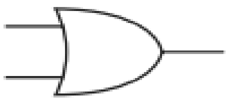


Notation used:
∧ e.g. A ∧ B

A	B	A∧B
T	T	T
T	F	F
F	T	F
F	F	F

Alternatives accepted:

AND e.g. A AND B
e.g. A.B

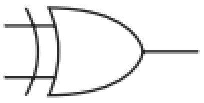


Notation used:
∨ e.g. A ∨ B

A	B	A∨B
T	T	T
T	F	T
F	T	T
F	F	F

Alternatives accepted:

OR	e.g. A OR B
+	e.g. A+B

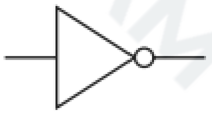


Notation used:
⊻ e.g. A⊻B

A	B	A⊻B
T	T	F
T	F	T
F	T	T
F	F	F

Alternatives accepted:

XOR	e.g. A XOR B
⊕	e.g. A ⊕B



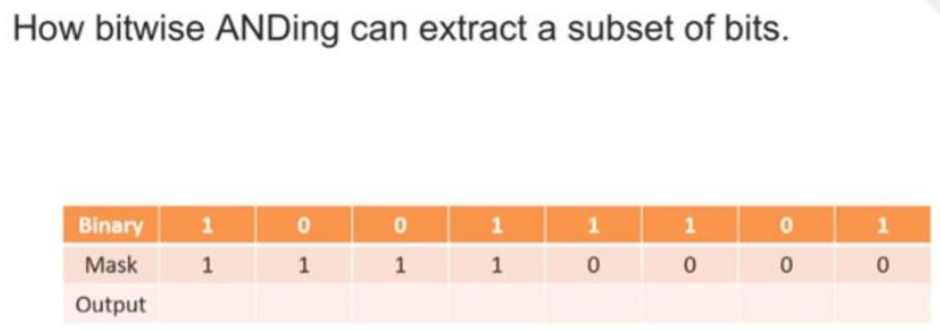
Notation used:
¬ e.g. ¬A

A	¬A
T	F
F	T

Alternatives Accepted:

bar	e.g. $\overline{A}$
~	e.g. ~A
NOT	e.g. NOT A

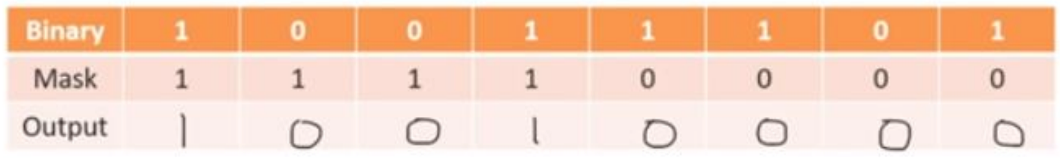
Masks mean when they give you these logic gates operations in a table like this:



How bitwise ANDing can extract a subset of bits.

Binary	1	0	0	1	1	1	0	1
Mask	1	1	1	1	0	0	0	0
Output								

In the question, they will tell you which gate this table is representing, in this case it’s an AND gate (ANDing) so you simply apply the logic of AND gates to get your answers:



Binary	1	0	0	1	1	1	0	1
Mask	1	1	1	1	0	0	0	0
Output	1	0	0	1	0	0	0	0

And the same applies to XORring (XOR) and ORing (OR) gates

Spec reference:

(j) How character sets (ASCII and UNICODE) are used to represent text.

ASCII is used to represent English characters only and uses 8 bits to represent each character, UNICODE represents characters from all over the world and use 16 bits to represent each character. A downside of UNICODE is that it requires twice as much space to represent each character than ASCII

Each character (Both in ASCII and UNICODE) is represented by a unique binary number.

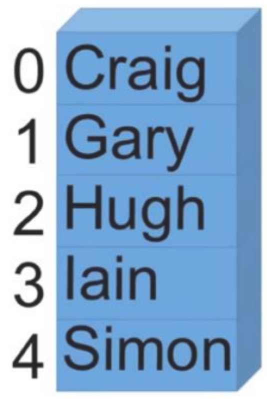
Spec reference:

(a) Arrays (of up to 3 dimensions), records, lists, tuples

Arrays come in three forms: records, lists and tuples.

In records, arrays can be accessed with just one index

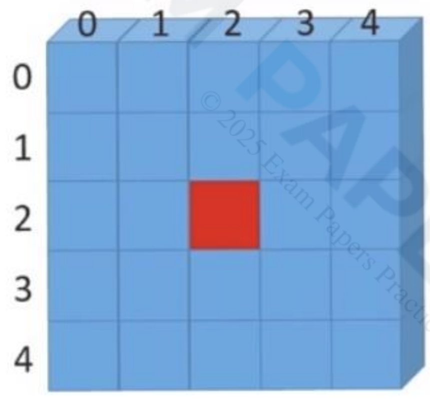
```
array students[5]
```



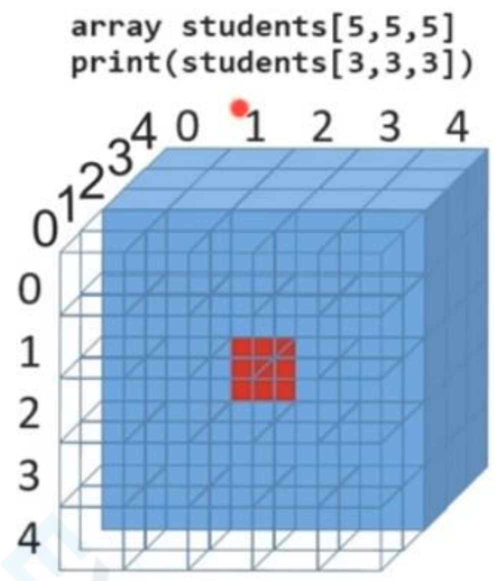
```
students [0]="Craig"
students [1]="Gary"
students [2]="Hugh"
students [3]="Iain"
students [4]="Simon"
```

In lists, arrays are 2D and can be accessed by two indexes:

```
array students[5,5]
print(students[3,3])
```



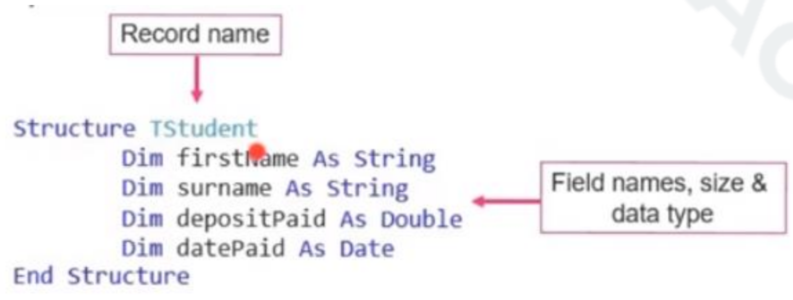
In tuples, arrays are 3D and can be accessed by three indexes:



Lists are dynamic data structures, meaning they can change size during execution. Tuples on the other hand are static, meaning they cannot change size during execution.

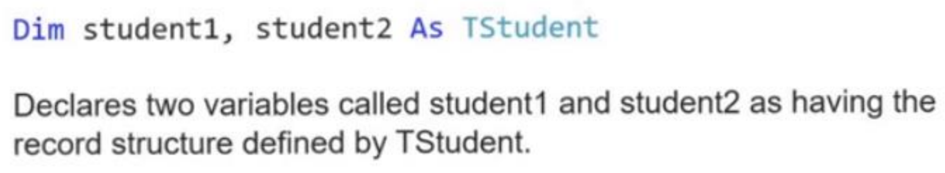
Arrays in general also store ONE type of data only, they can contain either integers or strings but not a mixture of them.

You also need to know how to use records in programming. First you define a record structure (Like a record template):



```
Structure TStudent
        Dim firstName As String
        Dim surname As String
        Dim depositPaid As Double
        Dim datePaid As Date
End Structure
```

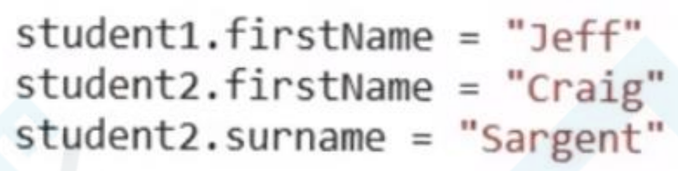
Then you declare variables (creating new records basically) for the record structure, so here you are creating two records for two students:



```
Dim student1, student2 As TStudent
```

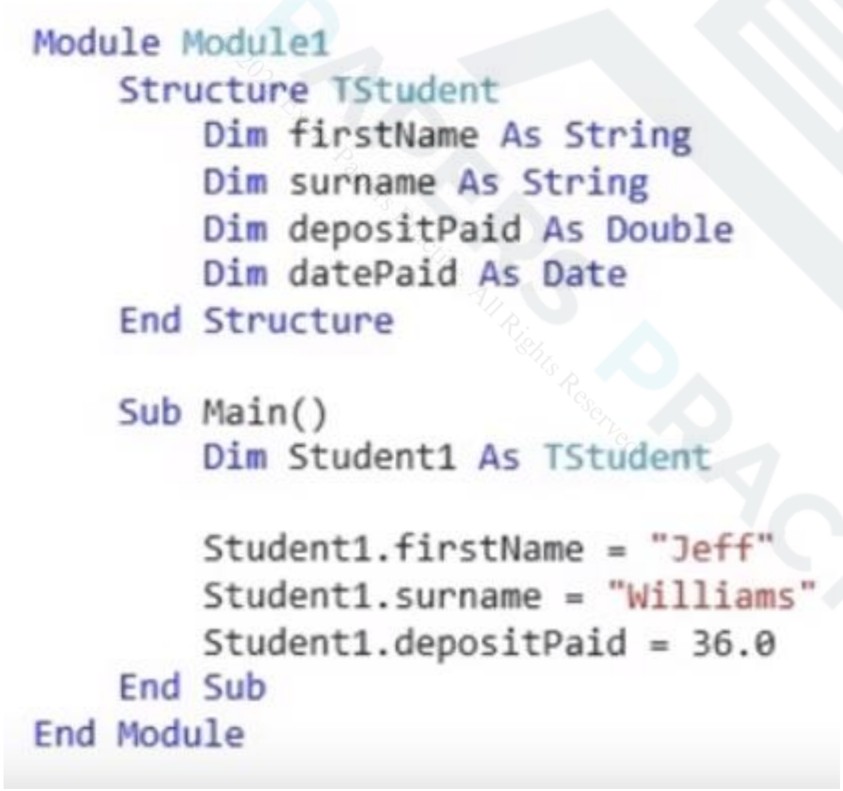
Declares two variables called student1 and student2 as having the record structure defined by TStudent.

Then you assign data to the fields of the variables (records) you have just created:



```
student1.firstName = "Jeff"
student2.firstName = "Craig"
student2.surname = "Sargent"
```

Here is the overall code from the three parts of code combined:



```
Module Module1
    Structure TStudent
        Dim firstName As String
        Dim surname As String
        Dim depositPaid As Double
        Dim datePaid As Date
    End Structure

    Sub Main()
        Dim Student1 As TStudent

        Student1.firstName = "Jeff"
        Student1.surname = "Williams"
        Student1.depositPaid = 36.0
    End Sub
End Module
```

Spec reference:

(b) The following structures to store data: linked-list, graph (directed and undirected), stack, queue, tree, binary search tree, hash table.

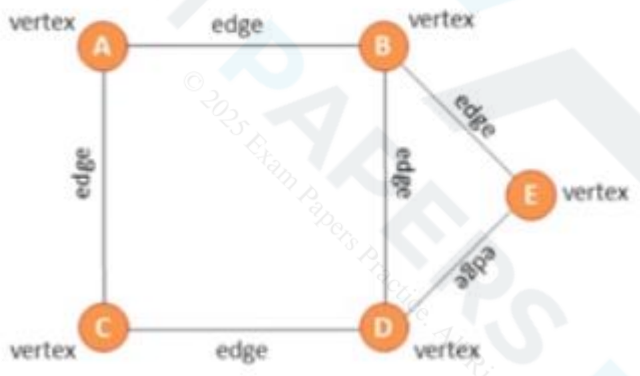
There is not much to know for this apart from actually looking at the data structures and maybe knowing some of their functions.

Graphs

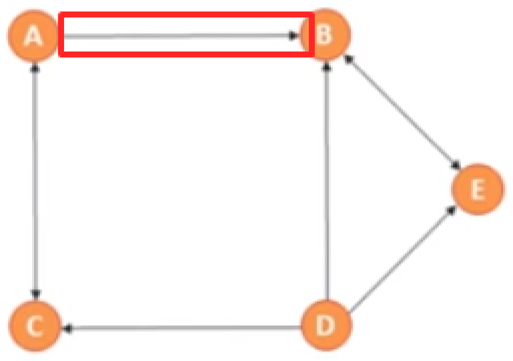
Graphs are dynamic data structures.

What are they used for? They are used to model things like: Navigation systems, data transmissions, web page links and social media trends.

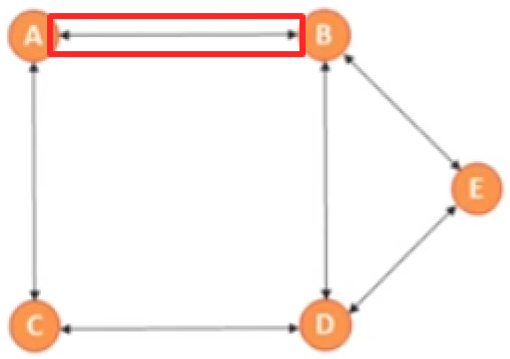
This is a graph:



A directed graph is a graph that has edges pointing to one direction only:

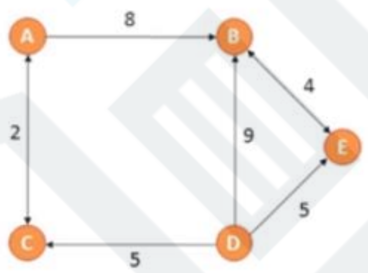


A undirected graph is a graph that has edges pointing to both directions:

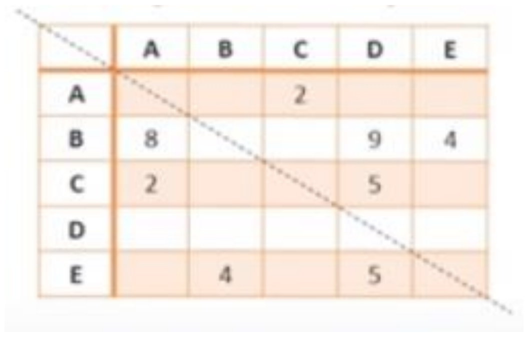


You will also need to be familiar with something like this:

```
Vertices {A,B,C,D,E}

Edges { (A,B), (A,C), (B,E), (C,A),
        (D,B), (D,C), (D,E), (E,B) }

Adding in costs gives us:

Edges { (A,B,8), (A,C,2), (B,E,4),
        (C,A,2), (D,B,9), (D,C,5),
        (D,E,5), (E,B,4) }
```



And you will be excepted to draw a graph, either a directional or unidirectional by looking at the indexes on the left. They can ask you to draw from a graph as well, like this:



	A	B	C	D	E
A			2		
B	8			9	4
C	2			5	
D					
E		4		5	

Which literally means the same thing as this, the only difference is the way they presented it:

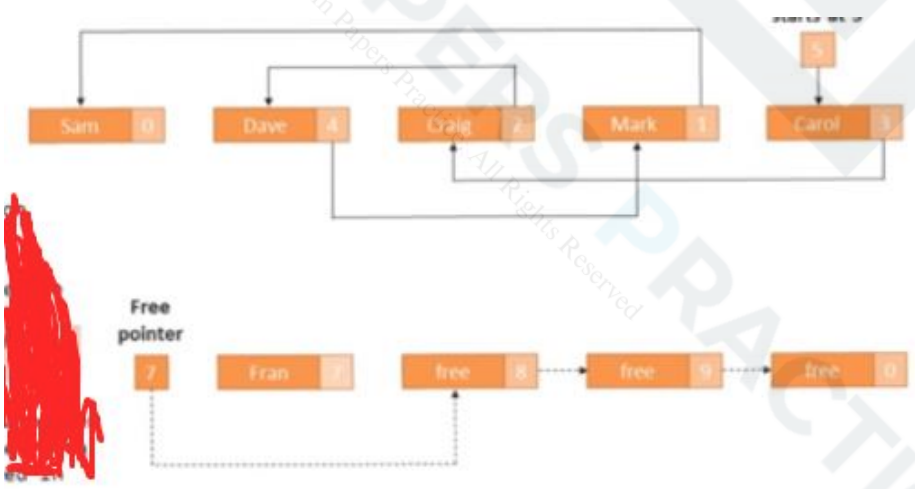
```
Edges { (A,B,8), (A,C,2), (B,E,4),
        (C,A,2), (D,B,9), (D,C,5),
        (D,E,5), (E,B,4) }
```

So don't let that fool you and when drawing graphs in the exams, USE A PENCIL so that if you make a mistake you can rub it off and correct it.

Spec reference:

(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).

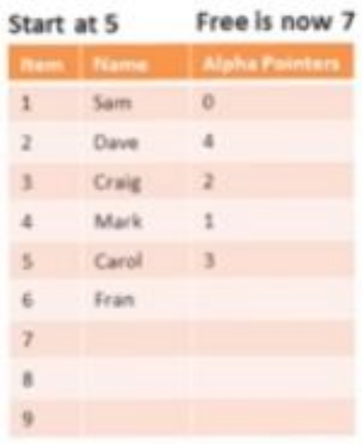
Here is a linked list:



[redacted]

Just ignore the red colour, I had to hide something, no not porn...

This is the table that goes with it:



Start at 5 Free is now 7

Item	Name	Alpha Pointers
1	Sam	0
2	Dave	4
3	Craig	2
4	Mark	1
5	Carol	3
6	Fran	
7		
8		
9		

Now we are going to add new data into this linked list, this is how it's done:

1. Insert the new data item into the location pointed by the free storage pointer (Which is in item location 7)
2. Update the free storage pointer
3. Work out where in the list the new item should be added
4. Once that has been decided on, update the **alpha** pointer value of the item that will precede the new data item to the item value of the new data item
5. And update the alpha pointer value of the new data item to the item value of the data item that will succeed it

In order to remove data…

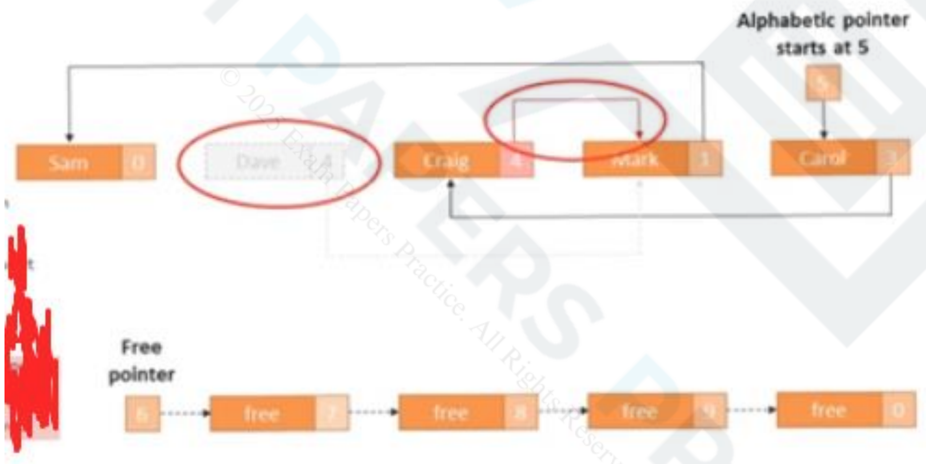
If the data item to be removed is in the first position:

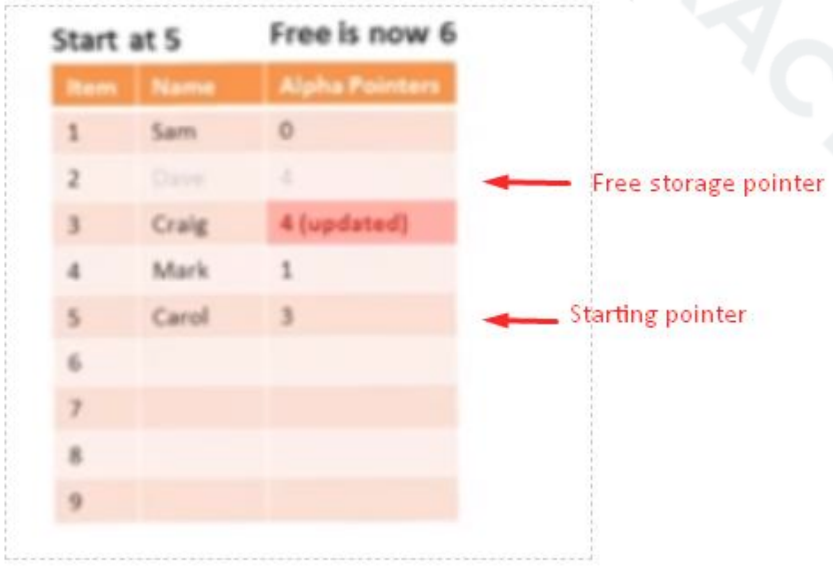
1. Update the **starting** pointer value to the item value of the data item you want to delete
2. Update the free storage pointer

If the case is otherwise:

1. Update the alpha pointer value of the data item located before the data item you want to delete, to the item value of the data item you want to delete
2. Update the free storage pointer
3. Starting pointer does not go anywhere, it stays where it is

Note that free storage pointers point to the location that is free of data and starting pointers point to the newest data item in the linked list:





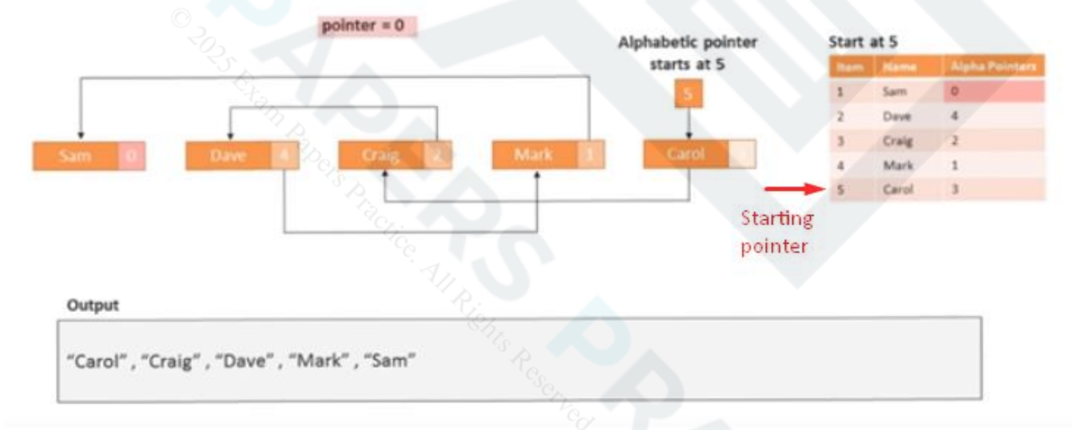
Start at 5 Free is now 6

Item	Name	Alpha Pointers
1	Sam	0
2	Dave	4
3	Craig	4 (updated)
4	Mark	1
5	Carol	3
6		
7		
8		
9		

Free storage pointer
Starting pointer

In order to traverse (scour) a linked list:

1. Start at the data item pointed by the starting pointer value
2. Go to the data item pointed by the starting pointer value
3. Output the data item
4. Set the pointer value to the item value of the next data item according to the alpha pointer value of the current data item
5. You do this until the pointer value reaches 0



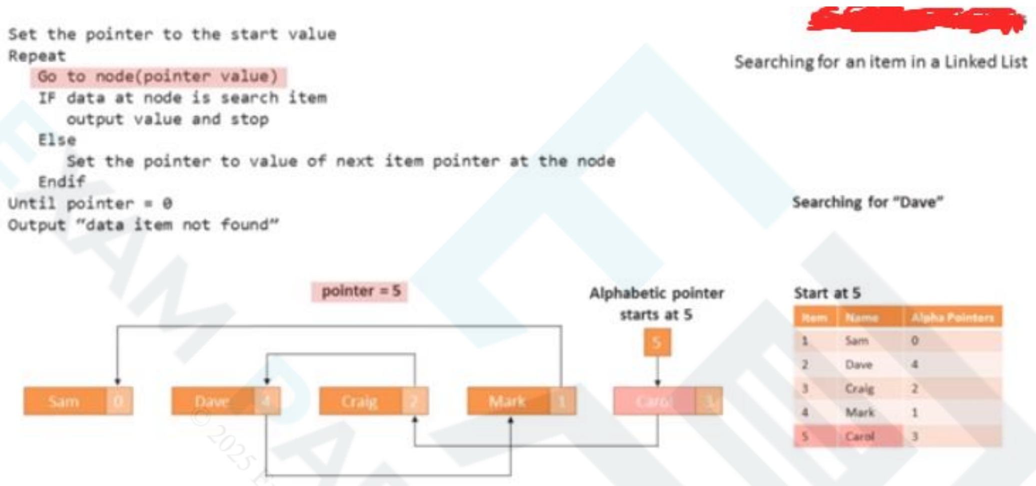
Item	Name	Alpha Pointers
1	Sam	0
2	Dave	4
3	Craig	2
4	Mark	1
5	Carol	3

"Carol", "Craig", "Dave", "Mark", "Sam"

In order to search a linked list for a data item:

1. Start at the data item pointed by the starting pointer value
2. If the data item at the start is the data item wanted, then output that data item

3. Else, set the pointer value to the item value of the next data item according to the alpha pointer of the current data
4. Repeat this until the data item is found or the pointer value is 0 (Which means the data has not been found)
5. If it's not found, just output "Data item not found"



[redacted]

Searching for an item in a Linked List

```
Set the pointer to the start value
Repeat
  Go to node(pointer value)
  IF data at node is search item
    output value and stop
  Else
    Set the pointer to value of next item pointer at the node
  Endif
Until pointer = 0
Output "data item not found"
```

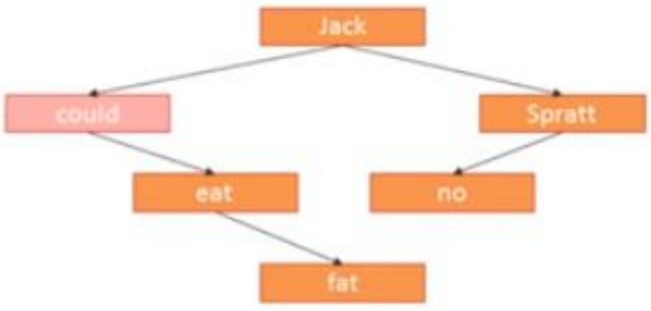
Searching for "Dave"

Start at 5

Item	Name	Alpha Pointers
1	Sam	0
2	Dave	4
3	Craig	2
4	Mark	1
5	Carol	3

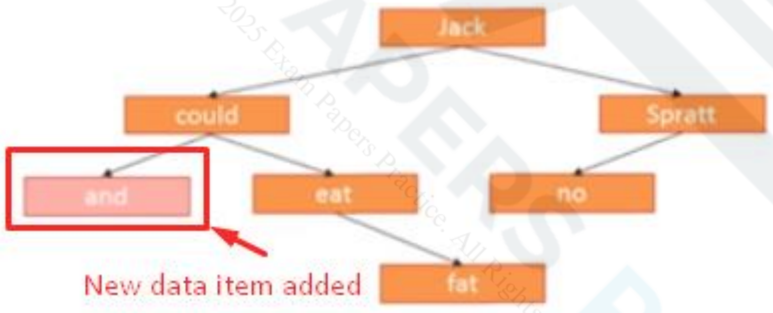
Trees and binary search trees

This is a tree:



Funny example I know, but we are trying to add the word and, in this tree, so in order to insert data into this tree, this is the algorithm you have to follow:

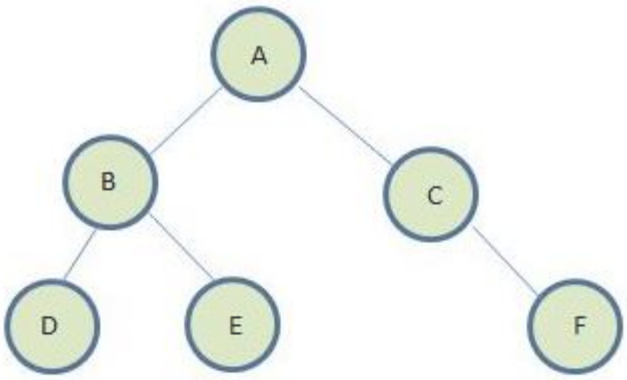
1. If tree is empty (As in like this whole tree is empty), enter data item at the root (the first node) and stop
2. The root is the first node
3. Repeat steps 4 and 5 until a node which is null is reached
4. If new data item to be inserted is less than the value in the current node, go left, else if the new data item to be inserted is larger than the value in the current node, go right
5. The current node is the node reached, the node is null if there is no node
6. If the node is null, create new node and add new data item into it



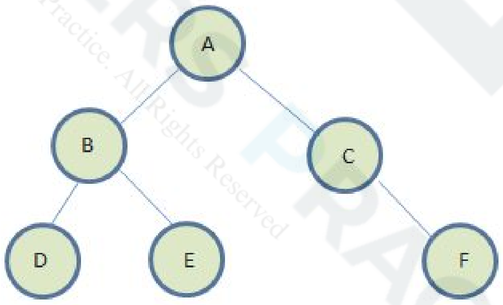
We are performing this search alphabetically just in case you are confused.

There are three ways to traverse (scour) a tree: pre-order, inorder and post-order.

Pre-order:



In order to traverse this tree pre-order way you start from the root and go left, THEN right. You always take the left side of the tree and once you reach the end of the left side you go back up one node at a time and back down to get at the right nodes. You will then end up at the root (the first node) again and will have to traverse the right side of the tree BUT still you have to do the left branches of the right side first. If this is all confusing you then don't worry, this example should make quick work of your understanding:



In the above tree, a preorder traversal would begin at the root, **A**.

It takes the left branch, visiting **B**.

It takes the left branch, visiting **D** and finding the end of the branch.

It returns up one level, and takes the right branch, visiting **E**.

It finds the end of the branch, returns up one level, finds no more branches and goes up again. It then takes the right branch, visiting **C**.

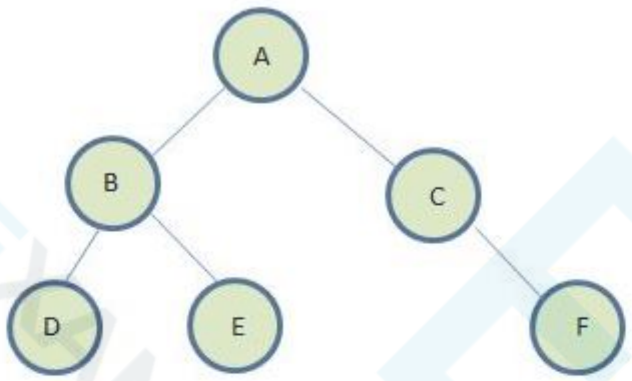
It takes the only branch available, and visits **F**.

So for the above tree, a preorder traversal visit order would be **A, B, D, E, C, F.**

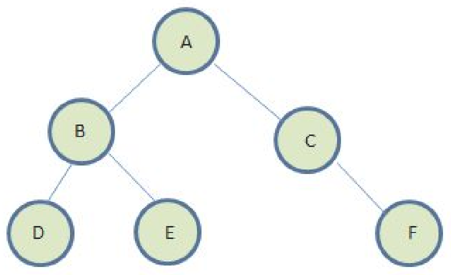
Key rule to remember: Parent node, left THEN right

Inorder:

Here is the same tree:



The rule is left, parent root and right. What this means is that you start from the left side of the tree with the most left side node which in this case it's D. It then goes back up one node to visit the parent node (In this case B) and goes back down to visit the right nodes. Goes back up again to find any other nodes connected and repeats this process. It eventually ends up in the root and begins to scour the right tree but always scours the left side of the right tree first, **ignoring any parent nodes**, once it reaches the left most side of the right tree that is where the **scouring actually begins** and you repeat that process mentioned earlier of left, parent node and right. If there is no left side on the right side of the tree then you just scour the nodes as you move down like shown here:



It begins at the bottom-most, left-most node **D**.

It visits the parent node **B**

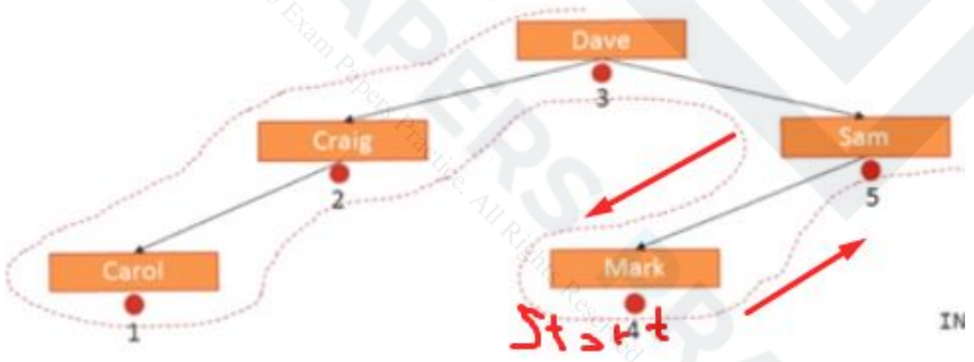
It goes back down to visit the remaining child node **E**

It returns back up, and finding no more child nodes goes one level further to **A**.

Now it is at the root, it travels down the right-hand side of the tree, visiting **C** then **F**.

So the visit order for this tree would be **D, B, E, A, C, F**

Alternatively, if there is a left side in the right side of the tree this is what happens:



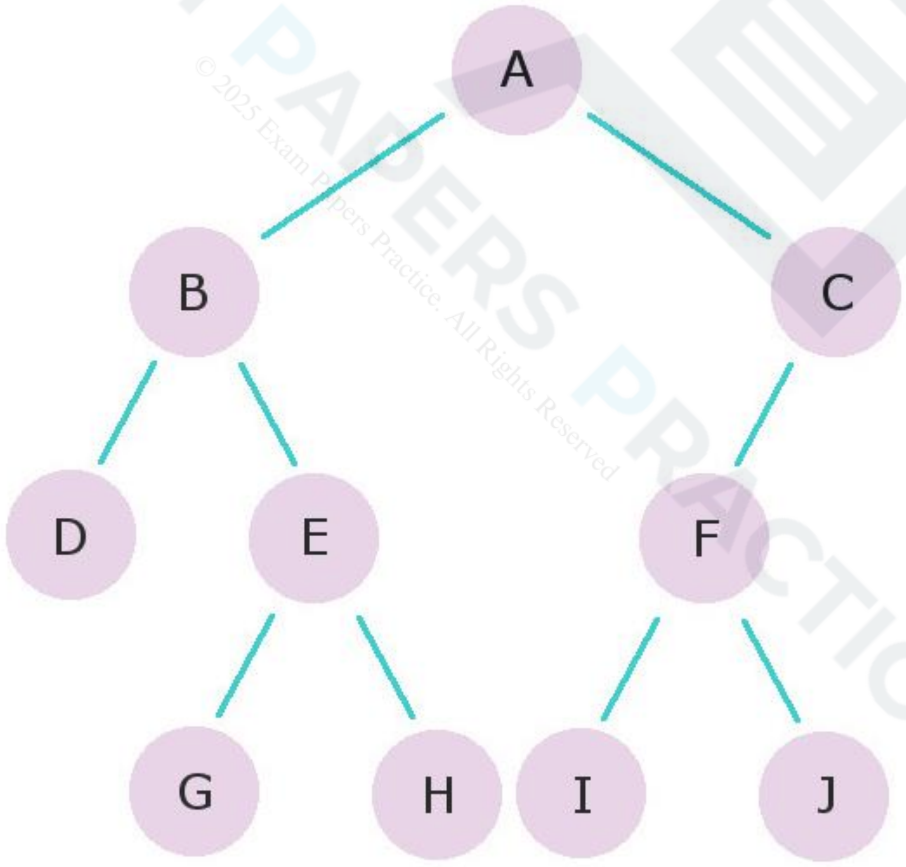
So, the inorder traversal of this tree is: Carol, Craig, Dave, Mark and Sam

Key rule to remember: Left, parent node THEN right

Post-order:

This is pretty much like inorder traversal except you don't scour the parent nodes, just the nodes connected to the parent nodes (excluding the ones who are parent nodes themselves like E in the picture below) and yes you always start from the left side of the tree.

Once all the nodes have been visited, you can visit the parent nodes by traversing up and up but once you reach the root, it's always the LAST one to be visited. Again, like in inorder traversal, scouring begins in the left most side of the right side of the branch, repeating the left, parent node and right processes except you leave out the parent nodes. And then just like mentioned earlier, you can visit the parent nodes by working your way up them. Here is an example to simply this BS in your head:



For the above tree, it begins at **D**, the left-most, bottom-most node. It goes up and back down to visit **G**, then **H**. With no more nodes at the bottom-most level, it visits parent nodes **E,** then **B**.

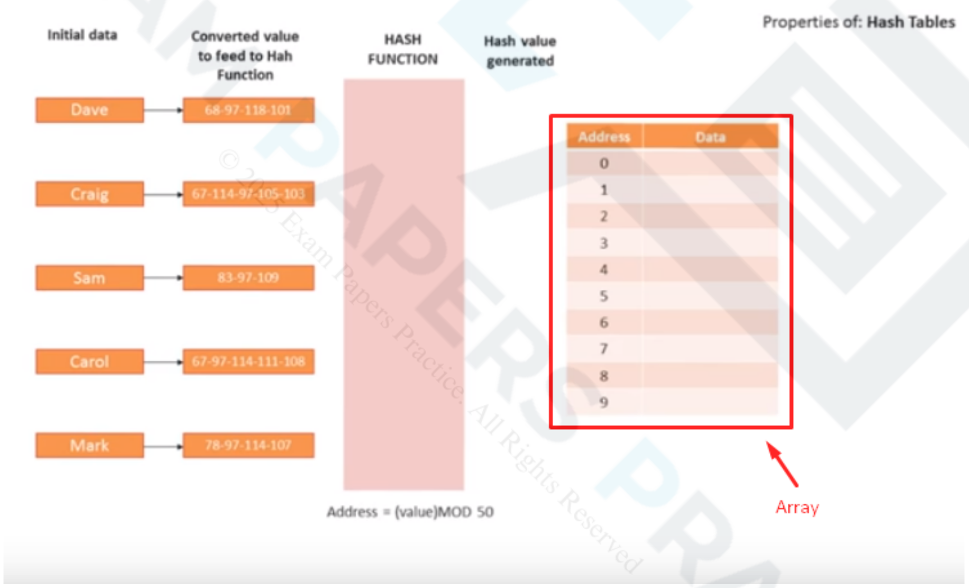
The left side of the tree is done, so it begins again with the right side, visiting **I**, then **J**, then the parent nodes **F** and **C**, before finishing at the root **A**.

So the visitation order is **D, G, H, E, B, I, J, F, C, A**

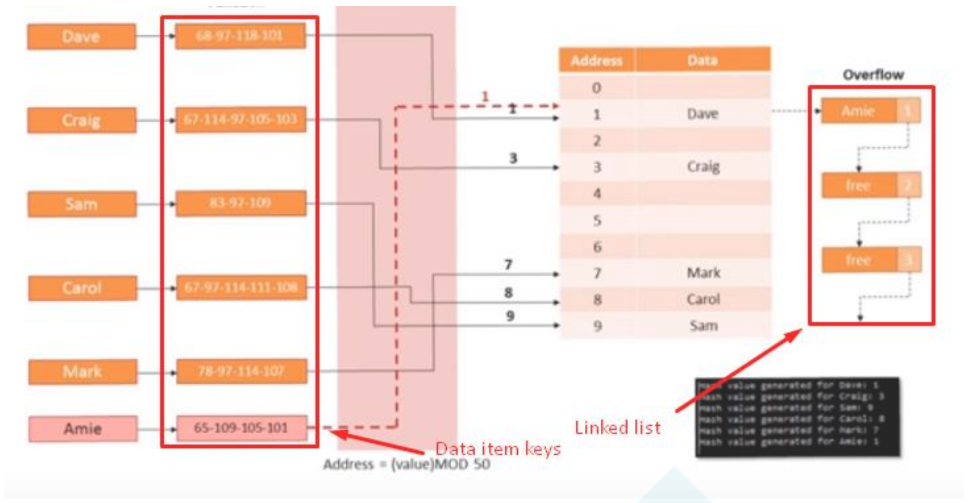
Key rule to remember: Left, right THEN parent node

<u>Hash tables:</u>

This is a hash table, this whole thing is a hash table.



So, what is happening here is the keys of the data items are used within the hash function to produce a hash value. That hash value is then used to point to a location within the array, if the location is free the data item is stored there, if not it looks through the locations in a linked list which is part of the overflow until a free storage location is found.

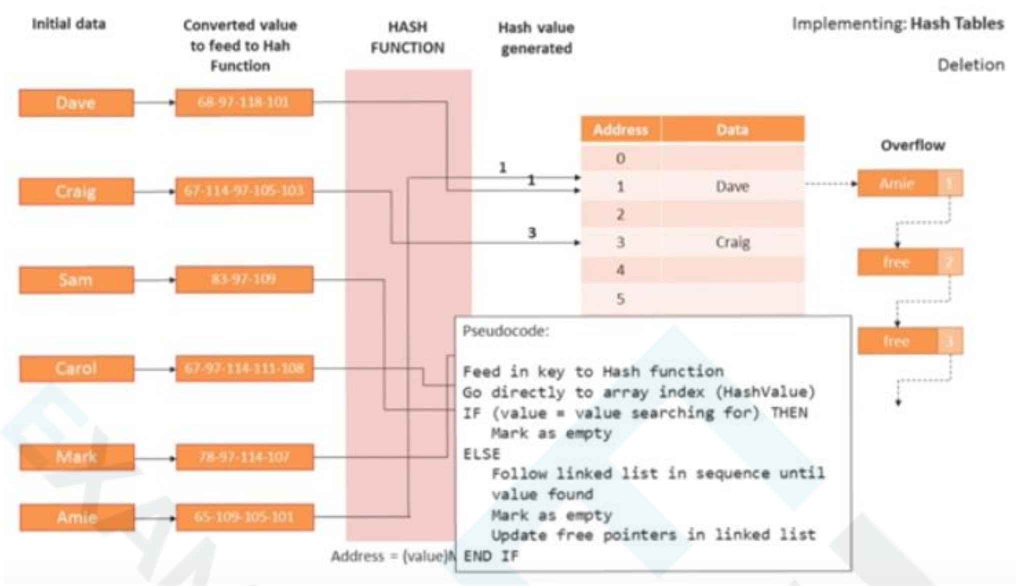


```
Hash value generated for Dave: 1
Hash value generated for Craig: 3
Hash value generated for Sam: 9
Hash value generated for Carol: 8
Hash value generated for Mark: 7
Hash value generated for Amie: 1
```

So, here we are trying to add Amie, the Hash value generated was 1 so we go to location 1 only to find out it has been taken over by Dave, so we look through the linked list part of the overflow and find a free storage location so store Amie there.

In order to search an item, we firstly use the keys of the data items into the hash function to produce a hash value. The hash value will point to a location of the array so we go to that location, if the data item wanted is found there, we output the data item. Else we look through the linked list in the overflow until the wanted data item is found.

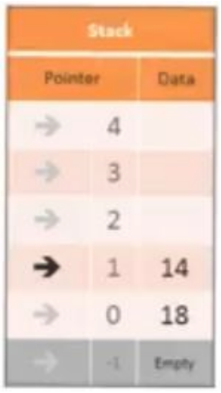
To delete a data item from the hash function is very similar, we use the keys of the data items in the hash function to produce a hash value, the hash value will point to a location in the array. If the data to be deleted is found there, mark that location as free storage space, if not look through the linked list in the overflow until the data item to be removed is found there and mark that location as free storage space. You then update the free storage pointers in the linked list



```
Feed in key to Hash function
Go directly to array index (HashValue)
IF (value = value searching for) THEN
   Mark as empty
ELSE
   Follow linked list in sequence until
   value found
   Mark as empty
   Update free pointers in linked list
END IF
```

<u>Stacks</u>

Stacks are a LIFO (Last In First Out) data structure – This means that the last data item inserted in a stack will always be the first one to come out



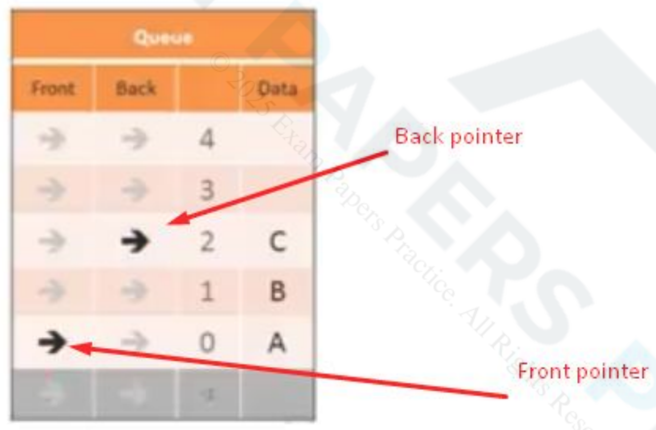
Stack	
Pointer	Data
→ 4	
→ 3	
→ 2	
➔ 1	14
→ 0	18
→ -1	Empty

In order to insert a new data item into the stack, you first check to see if the stack is full, if it is you report an error and stop (Because since it’s full you can’t add anything). Otherwise you increment the stack pointer and insert the new data item into the location pointed by the stack pointer

In order to read/delete a data item from the stack you first check to see if the stack is empty, if it is you report an error and stop (Because what is there to delete or read if there is nothing in the stack?) Otherwise you copy the data item from the location pointed by the stack pointer and decrement the stack pointer.

<u>Queues</u>

This is a queue:



To insert a new data item into a queue, you first check to see if the queue is full, if it is you report an error and stop. Otherwise increment the **back** pointer and insert the new data item into the location pointed by the back pointer

To delete/read a data item from the queue, you first check to see if the queue is empty, if it is, report an error and stop.

Otherwise you copy the data item from the location pointed by the **front** pointer and increment the front pointer

Spec reference:

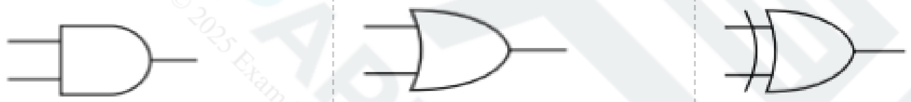
(a) Define problems using Boolean logic. See appendix 5d.

Pretty much the same as:

Spec reference:

(i) Bitwise manipulation and masks: shifts, combining with AND, OR, and XOR.

Except the notations outlined by the red boxes are all you need to know for the gates (But you will obviously have to know how to draw the gates themselves as well):



Notation used:
$\wedge$ e.g. $A \wedge B$

A	B	$A \wedge B$
T	T	T
T	F	F
F	T	F
F	F	F

Alternatives accepted:

AND e.g. A AND B
e.g. A.B

Notation used:
$\vee$ e.g. $A \vee B$

A	B	$A \vee B$
T	T	T
T	F	T
F	T	T
F	F	F

Alternatives accepted:

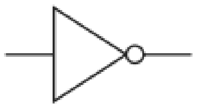
OR e.g. A OR B
+ e.g. A+B

Notation used:
$\veebar$ e.g. $A \veebar B$

A	B	$A \veebar B$
T	T	F
T	F	T
F	T	T
F	F	F

Alternatives accepted:

XOR e.g. A XOR B
$\oplus$ e.g. $A \oplus B$



Notation used:
¬ e.g. ¬A

A	¬A
T	F
F	T

Alternatives Accepted:

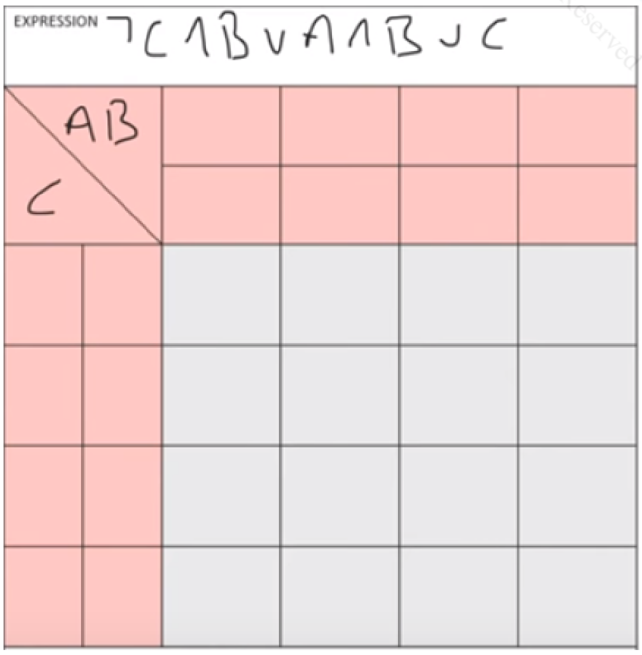
bar	e.g. $\bar{A}$
~	e.g. ~A
NOT	e.g. NOT A

Spec reference:

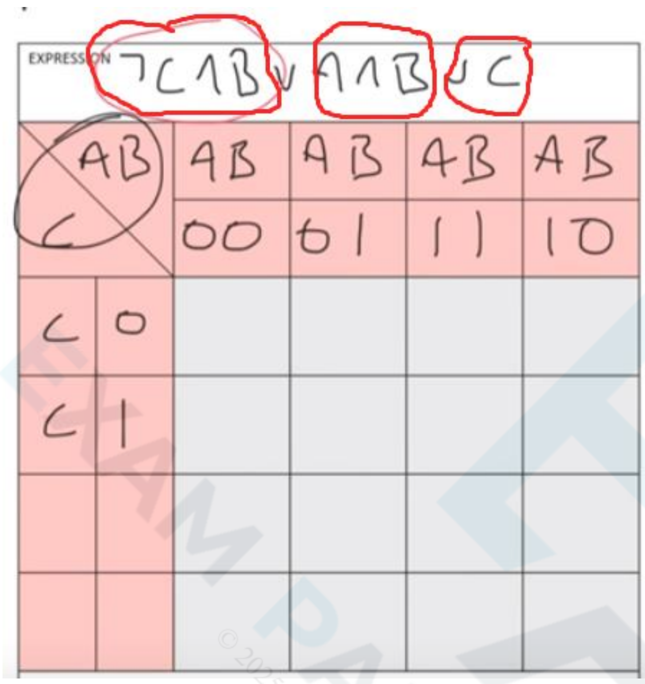
(b) Manipulate Boolean expressions, including the use of Karnaugh maps to simplify Boolean expressions.

Again, this will be pretty challenging to explain, but again the concept is easy to understand

So here is your Karnaugh map:



So, you have your expression here, you place the values of the expression in the left side of the table corner. You then split your expression into small chunks like this:

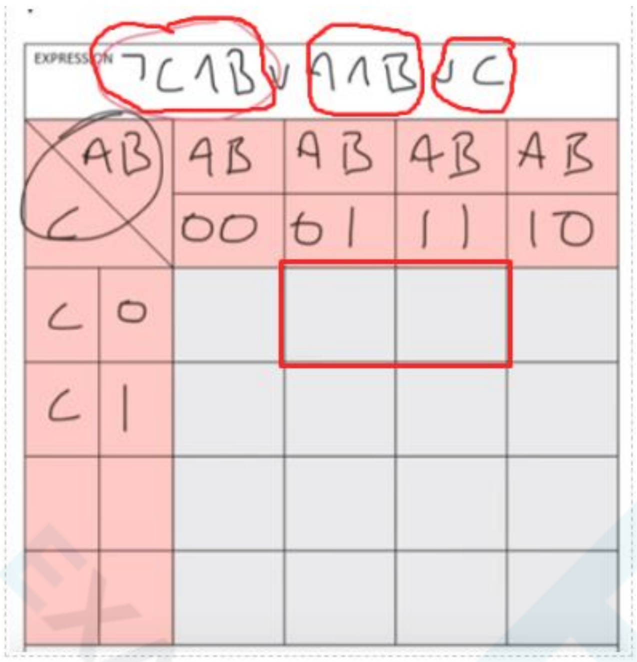


EXPRESSION $\neg C \wedge B \vee A \wedge B \vee C$					
AB / C		AB	AB	AB	AB
		00	01	11	10
C	0				
C	1				

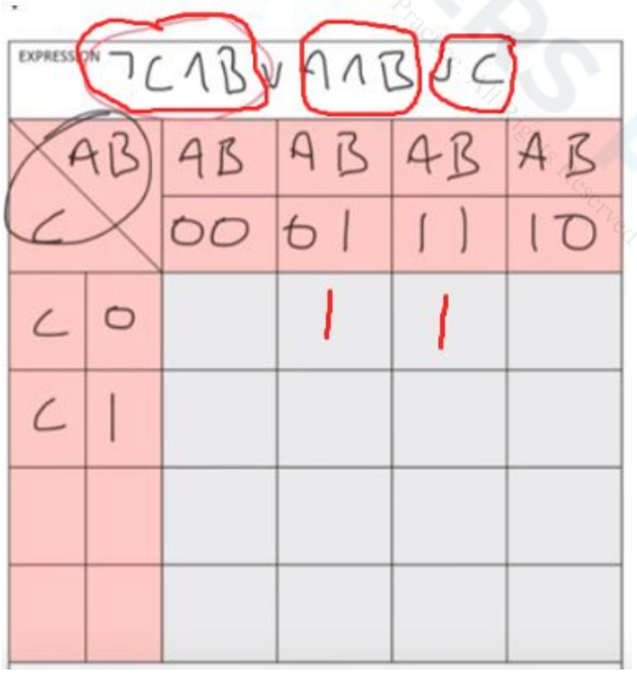
You have to split up your equation into small chunks, otherwise it WILL be impossible to do this map. Then with AB you place in different and possible input combinations as you would in a truth table, same for NOT C with a 1 and 0.

Now what you do is work out each chunk of the equation in your head, before doing that you need to look at the values the equation contains and choose the grids that have those values with 1 with it. So, for the first equation we will choose grids AB with 01 and 11 as the Bs in both of these have a 1 with them. As for NOT C we will choose the C with the 0 as NOT 0 equals 1.

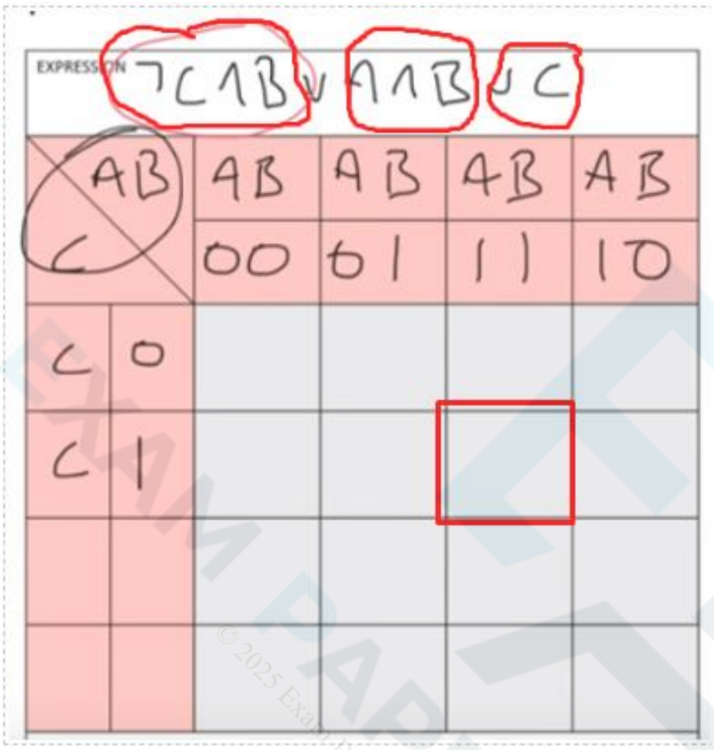
So, we are only going to use these girds:



And now we do the equation in our head, ignore the As, just ignore them. You work with the values in the chunk of the equation only and ignore the rest. If the result is 1 we place it in the grids, otherwise we place a 0:

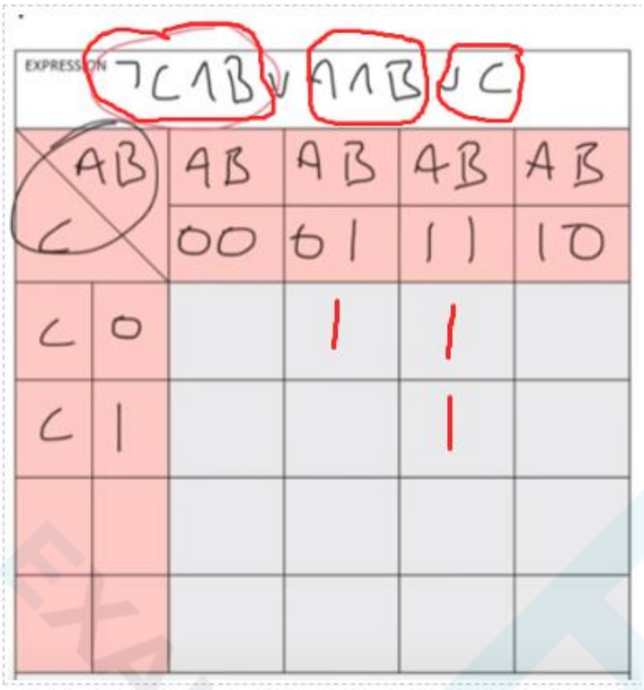


The results were 1s so we put them into the grid. Now for the second equation it says A AND B so we need to find a free grid with both AB having 1s and there is one:



You might be wondering "Wait, why don't we use the gird with AB 11 and C 0 too?" Well that grid has been taken (Can be with 1 or 0) so you just ignore it, any grids which have been taken you ignore them.

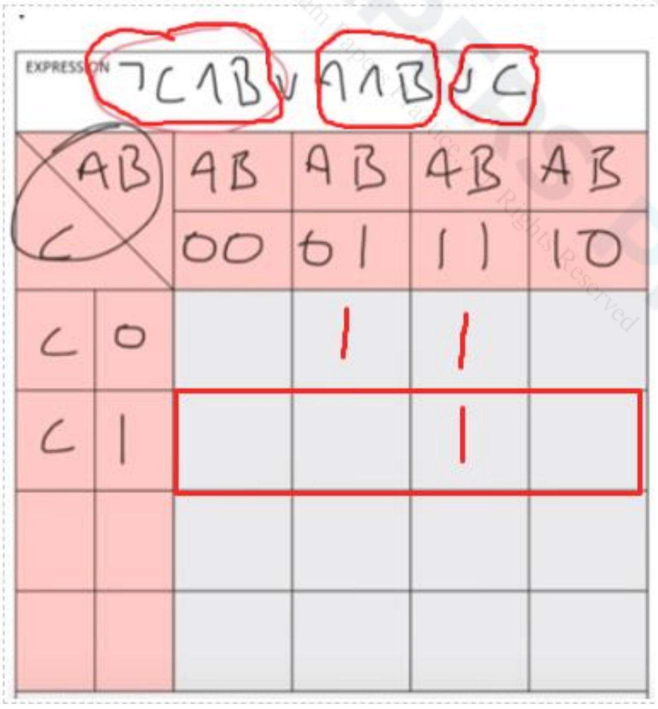
So once again we do the equation in our head, just ignore the C, remember, always work with the values in the chunk of the equation and NOTHING else outside that chunk:



EXPRESSION ¬C∧B ∨ A∧B ∨ C

AB / C		AB 00	AB 01	AB 11	AB 10
C	0		1	1	
C	1			1	

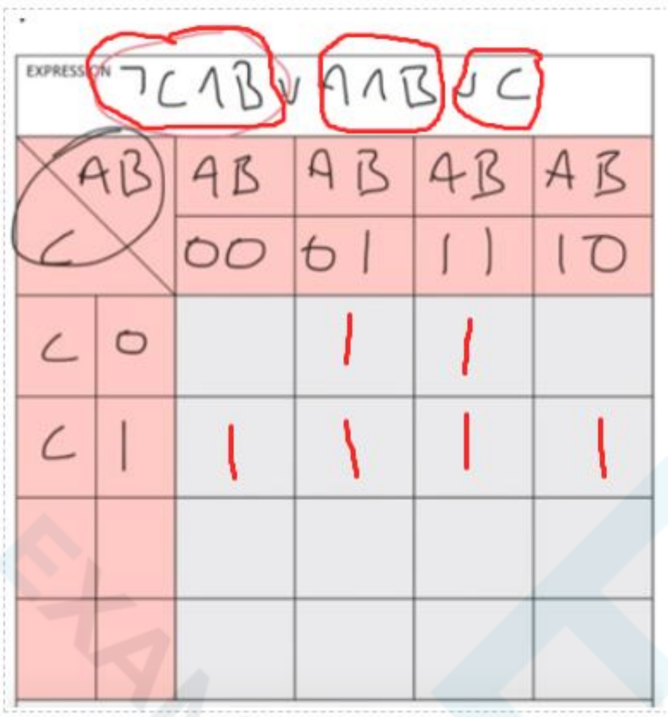
And finally. it's OR C, so we need to find grids with Cs that have 1s and there happens to be 4 of them in the bottom:



EXPRESSION ¬C∧B ∨ A∧B ∨ C

AB / C		AB 00	AB 01	AB 11	AB 10
C	0		1	1	
C	1			1	

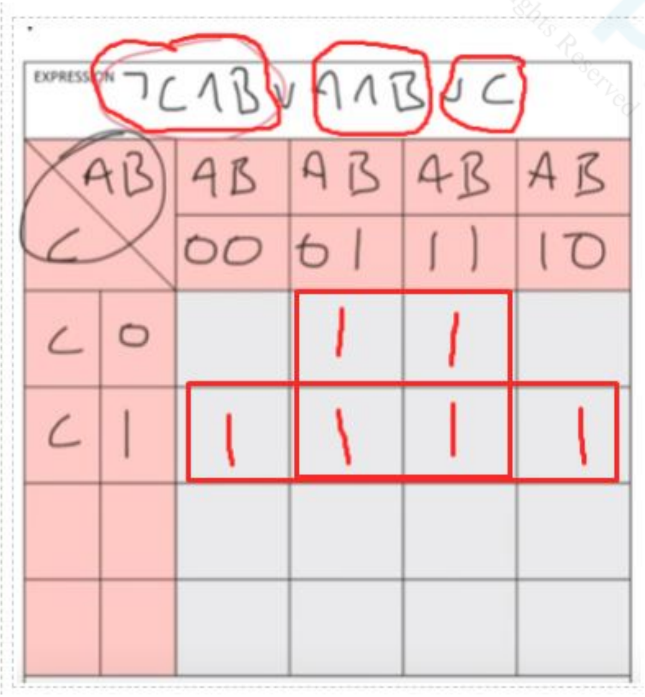
Since one of the girds has been taken, you just ignore that grid and work with the rest:



EXPRESSION $\neg C \wedge B \vee A \wedge B \vee C$

AB / C		A B	A B	A B	A B
		00	01	11	10
C	0		1	1	
C	1	1	1	1	1

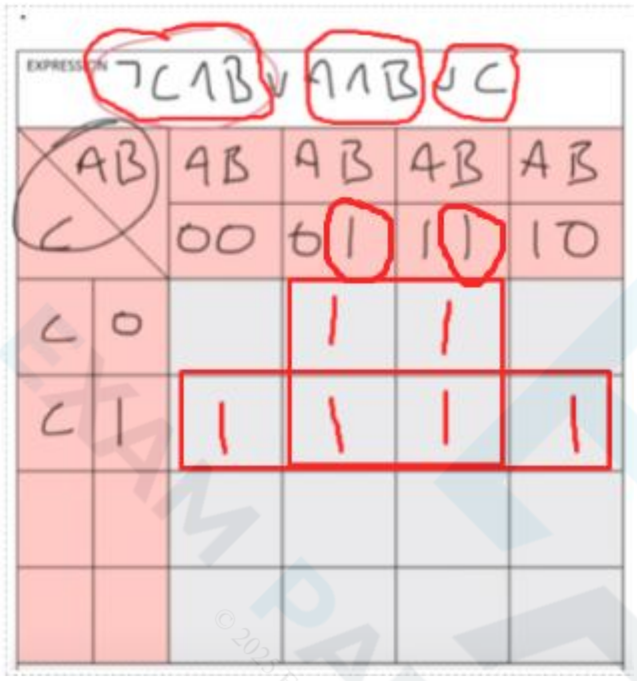
You completed the gird, but you are not finished yet, you need to simplify your equation. How do we do that? Well first group all 1s in 4s or 2s in any way possible BUT they must not contain a 0. Whatever equation they give you, you always group the 1s in 4s or 2s, any other way is wrong:



EXPRESSION $\neg C \wedge B \vee A \wedge B \vee C$

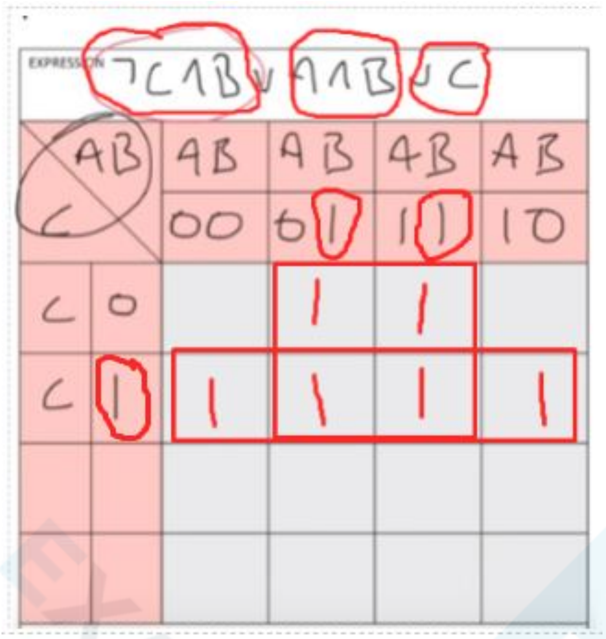
AB / C		A B	A B	A B	A B
		00	01	11	10
C	0		1	1	
C	1	1	1	1	1

And now we ask ourselves, which values (A, B, C or NOT C) are **constantly** significant with 1s in these boxes? Well for the box that has 1s in a box shape, B is the one with 1s:

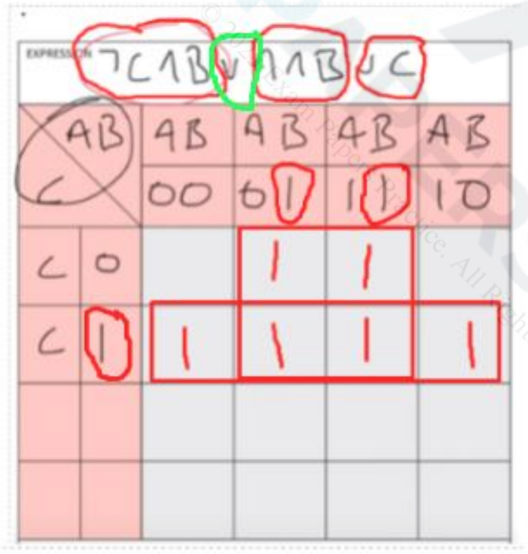


It cannot be C because it’s not constantly significant with 1s. (As the C contains both 0 and 1)

So that is the first box done, what about the second box with 1s stored in a linear way? Well the only value constantly significant with 1s in that box is C:



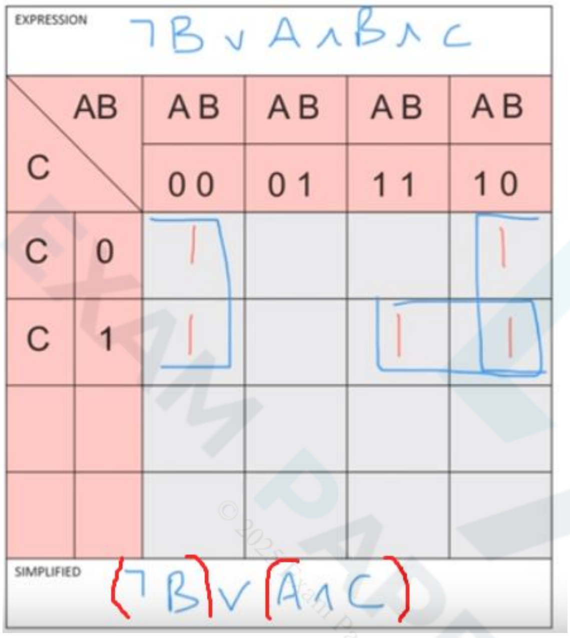
And now all we do is write B with whatever gate was left behind when we were splitting our equation and write C



$B \vee C$

Or vice-versa; C v B

If more than one gate value was left behind whilst splitting up the equation in chunks, you will need to use brackets where appropriate. If more than one gate value was left behind, it's very likely your simplified equation will contain at least three letters



EXPRESSION $\neg B \vee A \wedge B \wedge C$

AB / C		AB	AB	AB	AB
		00	01	11	10
C	0	1			1
C	1	1		1	1

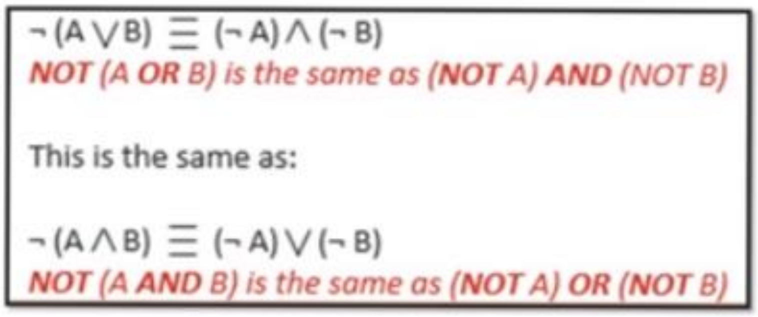
SIMPLIFIED $(\neg B) \vee (A \wedge C)$

Spec reference:

(c) Use the following rules to derive or simplify statements in Boolean algebra: De Morgan's Laws, distribution, association, commutation, double negation.

This section is all about simplifying Boolean equations, there are some laws or rules you need to remember, here they are:

Rule 1: De Morgan's Laws:



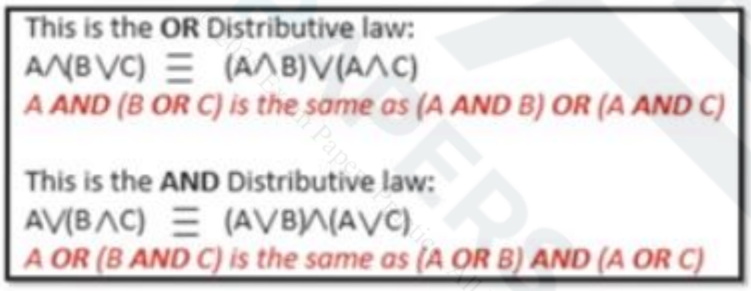
¬(A ∨ B) ≡ (¬A) ∧ (¬B)
NOT (A OR B) is the same as (NOT A) AND (NOT B)

This is the same as:

¬(A ∧ B) ≡ (¬A) ∨ (¬B)
NOT (A AND B) is the same as (NOT A) OR (NOT B)

This says that an OR can be changed to an AND or vice versa, however when this change is done, both of the terms have to be NOTted (Basically you have to put NOT near both of the terms)

Rule 2: Distribution

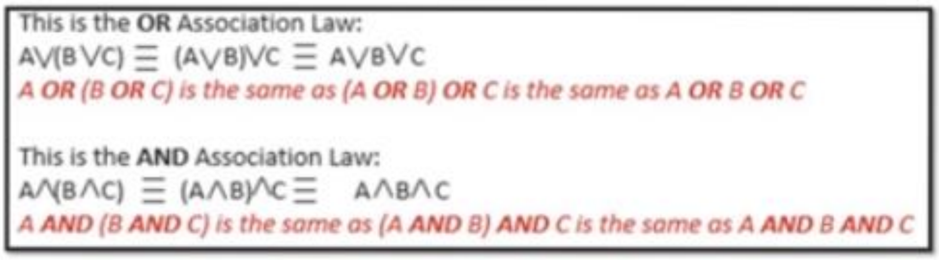


This is the **OR** Distributive law:
A∧(B∨C) ≡ (A∧B)∨(A∧C)
A AND (B OR C) is the same as (A AND B) OR (A AND C)

This is the **AND** Distributive law:
A∨(B∧C) ≡ (A∨B)∧(A∨C)
A OR (B AND C) is the same as (A OR B) AND (A OR C)

This law says that you can multiply the first term twice and put each of them alongside the other terms in the equation using the operator near the term that was multiplied before it was multiplied (If you get what I mean) and put it in between. The two term brackets are then held together by the other operator

Rule 3: Association



This is the **OR** Association Law:

$A\vee(B\vee C) \equiv (A\vee B)\vee C \equiv A\vee B\vee C$

*A **OR** (B **OR** C) is the same as (A **OR** B) **OR** C is the same as A **OR** B **OR** C*

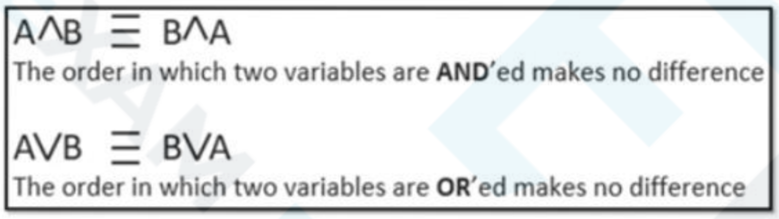
This is the **AND** Association Law:

$A\wedge(B\wedge C) \equiv (A\wedge B)\wedge C \equiv A\wedge B\wedge C$

*A **AND** (B **AND** C) is the same as (A **AND** B) **AND** C is the same as A **AND** B **AND** C*

This law says you can regroup, add or remove brackets

Rule 4: Commutation



$A\wedge B \equiv B\wedge A$

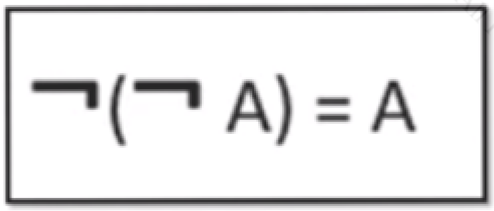
The order in which two variables are **AND'**ed makes no difference

$A\vee B \equiv B\vee A$

The order in which two variables are **OR'**ed makes no difference

The order of the terms is not important

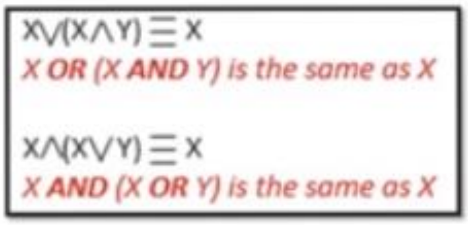
Rule 5: Double negation



$\neg(\neg A) = A$

You can cancel out double NOTs

Rule 6: Absorption



$X\vee(X\wedge Y)\equiv X$

*X **OR** (X **AND** Y) is the same as X*

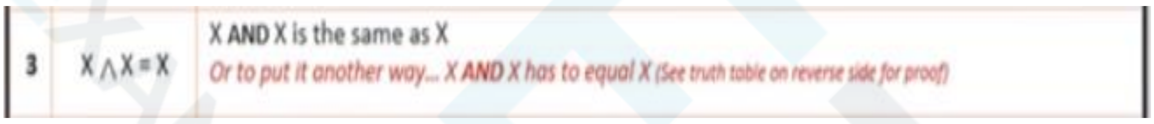
$X\wedge(X\vee Y)\equiv X$

*X **AND** (X **OR** Y) is the same as X*

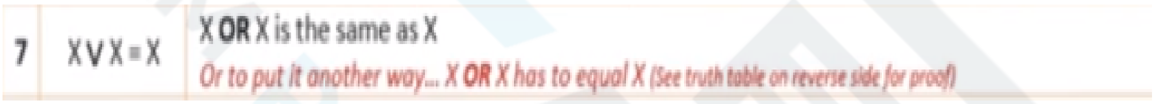
The outside term can always “absorb” all the terms and operators associated with it, but it can only happen if these two rules are met:

1. The operators (the gates) must be different from eachother inside and outside the brackets
2. The outside term must be inside the brackets as well

And here are some additional rules that needs to be learnt:

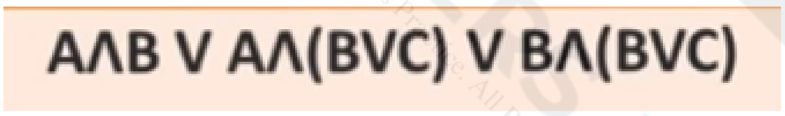


3	$X \wedge X = X$	X AND X is the same as X *Or to put it another way... X **AND** X has to equal X (See truth table on reverse side for proof)*



7	$X \vee X = X$	X **OR** X is the same as X *Or to put it another way... X **OR** X has to equal X (See truth table on reverse side for proof)*

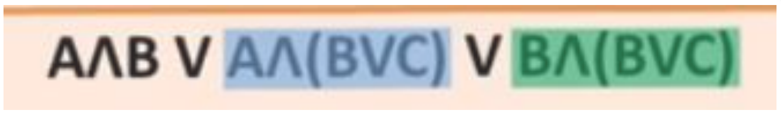
Here is a worked example:



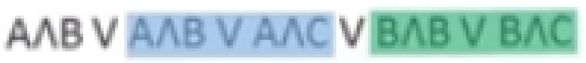
A∧B V A∧(BVC) V B∧(BVC)

This is the equation we are trying to simplify and here are the steps:

Step 1: Remove the brackets -

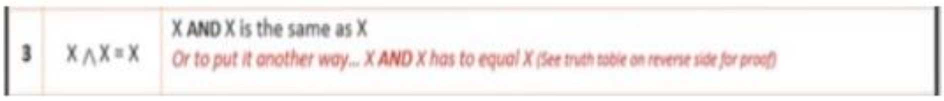


A∧B V A∧(BVC) V B∧(BVC)



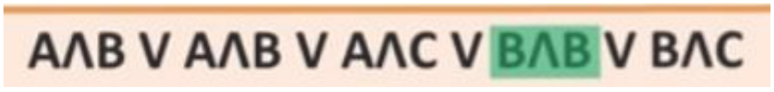
A∧B V A∧B V A∧C V B∧B V B∧C

Step 2: We can apply one of the general rules

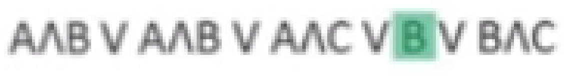


3	X∧X=X	X AND X is the same as X *Or to put it another way... X AND X has to equal X (See truth table on reverse side for proof)*

On the 4th term:

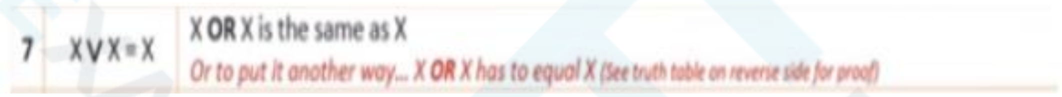


A∧B V A∧B V A∧C V B∧B V B∧C



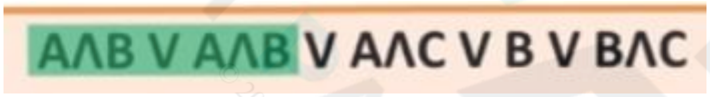
A∧B V A∧B V A∧C V B V B∧C

Step 3: We can apply one of the general rules

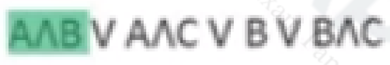


7	XVX=X	X OR X is the same as X *Or to put it another way... X OR X has to equal X (See truth table on reverse side for proof)*

On the 1st and 2nd terms:

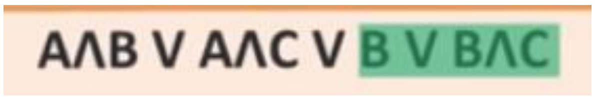


A∧B V A∧B V A∧C V B V B∧C

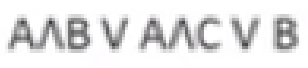


A∧B V A∧C V B V B∧C

Step 4: We can apply the rule of absorption on the 3rd and last terms:

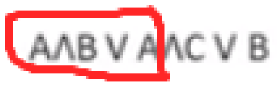


A∧B V A∧C V B V B∧C



A∧B V A∧C V B

Step 5: We can once again apply the rule of absorption on the first and second terms:



A∧B V A∧C V B

$A \wedge C \vee B$

And this is as simple as the equation can get

Spec reference:

(d) Using logic gate diagrams and truth tables. See appendix 5d.

Again should have done this at GCSE and is already covered by the spec point -

Spec reference:

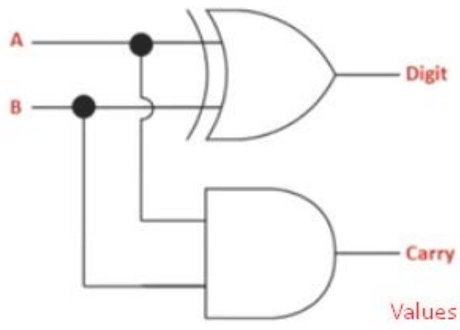
(i) Bitwise manipulation and masks: shifts, combining with AND, OR, and XOR.

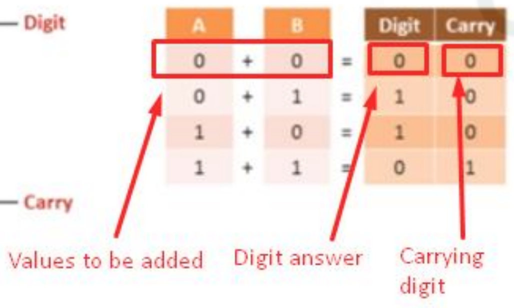
Spec reference:

(e) The logic associated with D type flip flops, half and full adders

<u>Half adders and full adders</u>

This is just binary addition done in a table and represented by logic gates, literally....



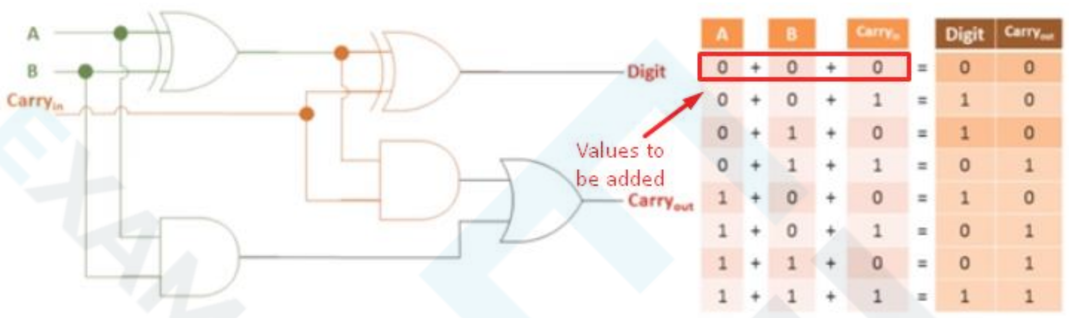


A		B		Digit	Carry
0	+	0	=	0	0
0	+	1	=	1	0
1	+	0	=	1	0
1	+	1	=	0	1

Values to be added
Digit answer
Carrying digit

This is a half adder, so here 0 + 0 in binary is 0, so the digit is 0 and the carrying digit is 0. 1 +1 in binary is 0 with 1 carry so digit is 0 and the carry is 1. Easy

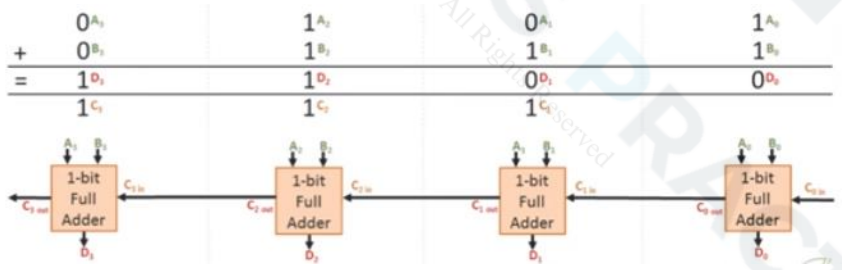
A full adder is not much different from a half adder, it just has three values to be added:



A		B		Carry$_{in}$		Digit	Carry$_{out}$
0	+	0	+	0	=	0	0
0	+	0	+	1	=	1	0
0	+	1	+	0	=	1	0
0	+	1	+	1	=	0	1
1	+	0	+	0	=	1	0
1	+	0	+	1	=	0	1
1	+	1	+	0	=	0	1
1	+	1	+	1	=	1	1

So same old rule applies here, 1 +1 +1 in binary is 1 with 1 to carry, therefore the digit is 1 with a 1 to carry and so on....

They might give you something like this in the exam:

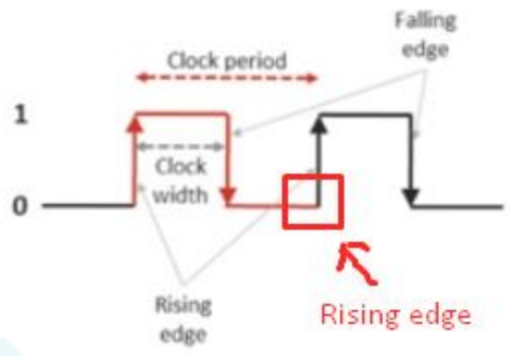


Don't be fooled by the new features, this is just binary addition. You do 1 + 1 = 0 with 1 to carry so you put the 1 in the next column as a carry and use that 1 with your next calculation and so on.....

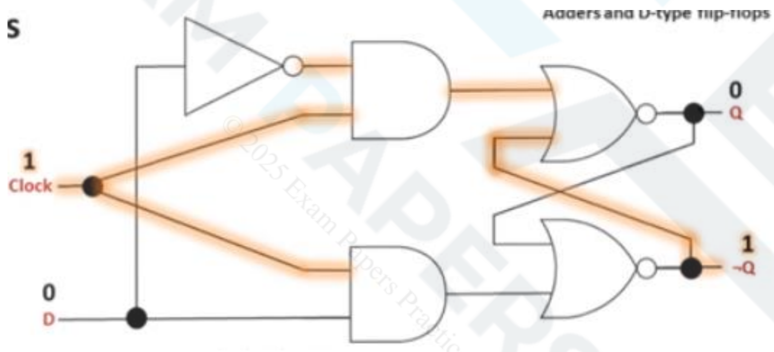
D type flip flops

How did your exam go? I flopped, I flopped and I flopped....

So here all you need to understand is something called a clock:

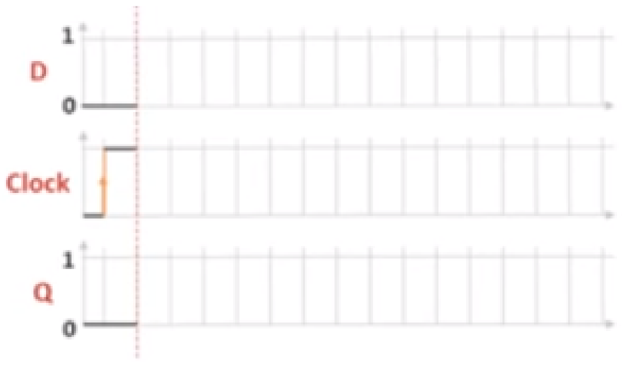


You will be given something like this in your exam:

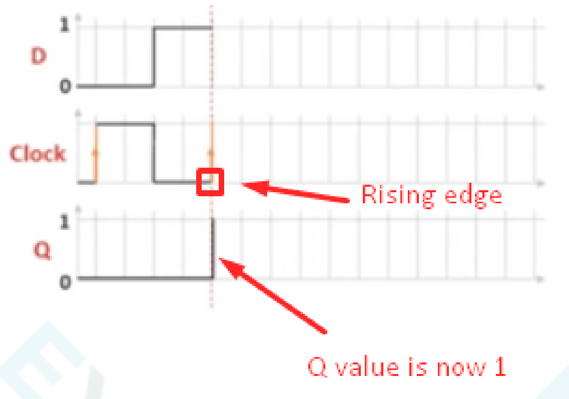


Now you simply apply your logic gates rules here, the only catch is the values of Q and D, how do you get them?

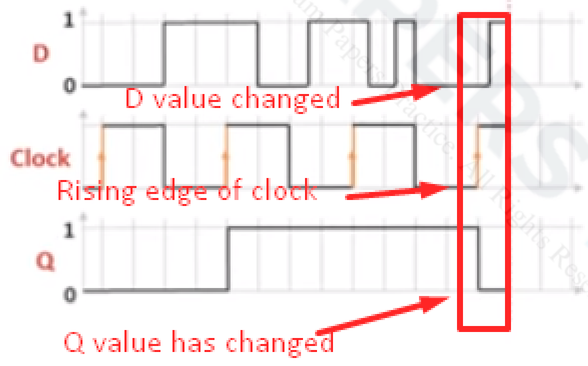
You use this graph they will give you:



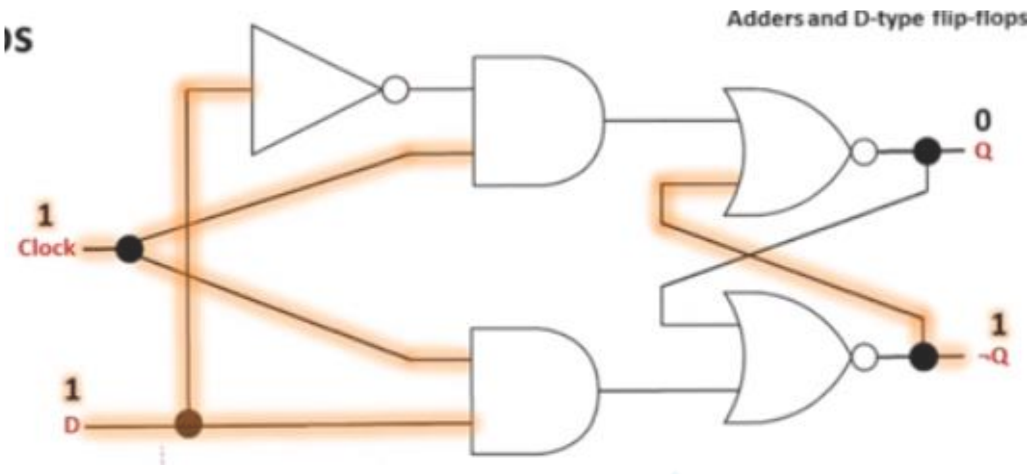
And it will already be completed so don't worry about drawing the lines, all you need to know is that the **Q will reach one once the clock is at a rising edge**. Just ignore the D



So now the value of Q is 1, the only way the value of Q will change again is that if the value of D changes at the **same time during the rising edge of a clock**:



As you can see, the value of D changed just as there was a rising edge from the clock, therefore the value of Q has now changed. So here are our updated values on the table -



YES! FUCK YEAHHHHHH! Finally, you finished learning about all of this graph, mathsy kind of shite. Just one more section to go before you finish everything needed for paper one....on and forward!

Legal, moral, ethical and cultural issues

Ah finally we get to leave all that sciency, mathsy kind of stuff behind and move on to some humanity, no we are not talking about saving humans. You will be learning about laws and ethical issues involved with using computers, if you are thinking of studying law at university then you can start off by learning these few laws to.....you know give you a headstart 😉

Spec reference:

(a) The Data Protection Act 1998.

This law sets out the requirements for storing and using personal data of individuals.

This is what the law involves:

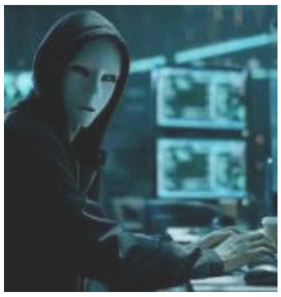
- Data must be collected lawfully, so customer rights are not breached
- Data collection should not be excessive in order to NOT encourage spam emails
- Data should not be kept for any longer than needed
- The owner of the data has the right to demand the data to be changed
- Data should be accurate and kept up to date
- The organisation has to set in place good security measures to protect the data from unauthorised access
- Data should be used for the specified purposes only

Spec reference:

(b) The Computer Misuse Act 1990

This is a law which makes unauthorised access to a computer illegal. It makes the following things illegal:

- Accessing a computer with the intent to commit or facilitate further crimes
- Accessing a computer with the intent to sabotage its operating



He is my guy!

Spec reference:

(c) The Copyright Design and Patents Act 1988.

This law protects the property rights belonging to an individual or organisation.

It makes it illegal to:

- Copy work
- Modify work
- Distribute work

Of other people without their permission or a license

Spec reference:

(d) The Regulation of Investigatory Powers Act 2000.

This law gives the right to certain organisations (Like the police, MI6 etc..) to monitor internet behaviour and communications

The law gives these organisations the right to:

- Demand customer surveillance from internet service providers
- Carry out mass surveillance
- Monitor an individual's internet activity
- Demand that internet service providers fit in equipment to facilitate surveillance
- Demand confidential information from internet service providers

Spec reference:

The individual moral, social, ethical and cultural opportunities and risks of digital technology:

• Computers in the workforce.

Computers can work for longer, without breaks, they can work faster and are more accurate in their work. Many computers have replaced low level jobs that humans used to do, an example is scanning in the checkout at a supermarket.

Automated decision making

Computers can also make decisions for humans in everyday situations based on a set of rules. For example, should a door fitted with a computer mechanism call the police when someone other than the owner enters the house? Or a fridge which can decide which foods to throw away based on their expiration date, or a kettle deciding how hot or cold does it need to boil certain drinks etc....the list is endless

These algorithms should only be SUPPORTING the decisions of humans and not replacing their judgment.

Artificial intelligence

The artificial intelligence or AI for short is an algorithm which works in order to appear intelligent. An example would be a knowledge-based system.

A knowledge-based system attempts to replicate the performance of an expert human for a specific task

A knowledge-based system is made out of three parts:

- The knowledge/rule base: This contains all the knowledge a human expert would have
- Inference engine: This searches the knowledge/rule base for possible answers to queries

- User interface: This allows the user to input queries so they can be processed by the knowledge/rule base and display the results back to the user

Environmental effects

Computer parts have toxic chemicals and metals in them which are very harmful, however they can be recycled.

Computers can help with environmental effects, for example:

- Computers can control car engines, making sure they use as less fuel as possible and creating as less pollution as possible
- Computers mean people can work from home than drive to work
- Working from home means less electricity is used to power up big offices

Censorship and the internet

Censorship is the blocking of resources from being published, distributed or viewed. People or organisations may censor certain things on the internet for ethical, religious, safety or political reasons.

Perfect example is North Korea where the dear Leader Kim Jong Un censorships anything on the internet which goes against the regime’s values. This results in a huge amount of ignorance within the population

!

Monitor behaviour

There are three types of behaviour monitoring:

- There is one where the user wants to be monitored by using sport watches etc..to help improve their lifestyle
- One called passive monitoring like CCTV cameras where it can help reduce crimes, but some argue this is invasion of privacy
- Last one is forced tracking, where people like criminals are forced to wear tracking devices so authorities can track their movements

Also, employers monitor their employee's behaviour by keeping a record of the websites they visited during work hours. Some argue this is necessary so employees do their jobs, others argue this is invasion of privacy.

Analyse personal information

Data mining is when large amounts of data are analysed to find out patterns in order to predict the next actions. This is used when recommending products to customers after they have brought something online like in Amazon. It can be used by many businesses to know what their customers want to buy so they can save on a lot of money by not spending on buying unnecessary stocks, and spend more on buying things which their customers want. This can help to increase the profits of many businesses.

Data collected on individuals is not just about shopping, it can also be about:

- Gaming patterns
- Social media interactions
- Search histories
- Travel plans
- Online chats

The dates and times of when you were doing these things can be logged.

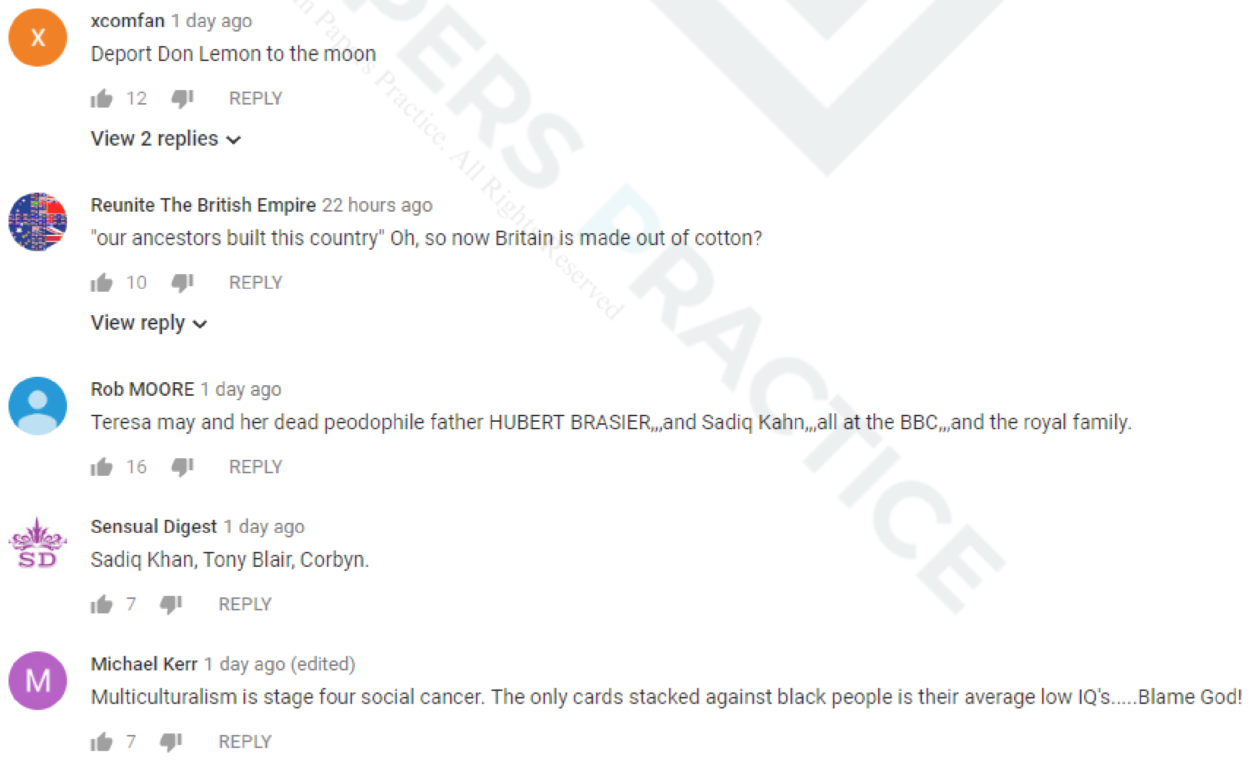
Piracy and offensive communications

Piracy is when copyrighted material is distributed on a large scale, materials can be accessed for free or for a reduced price when others other than the owners distribute them. The good side is that people obviously enjoy watching and using these resources but should the owners of these materials really suffer major financial loses because of this? Some of them have jobs based on this, so they can they can go bankrupt if piracy continues.

Ah offensive communications! A lot of people, including YOU refer to this as trolling! And you know what trolling is right? I bet at some point on the internet, you were a troll yourself.

You should well know what a troll is, but for those who have been living under a rock, a troll is someone who posts offensive messages online in order to cause arguments.

The YouTube comments section is a great example of this, especially on far-right political channels. Here is one comment section from one of these channels:



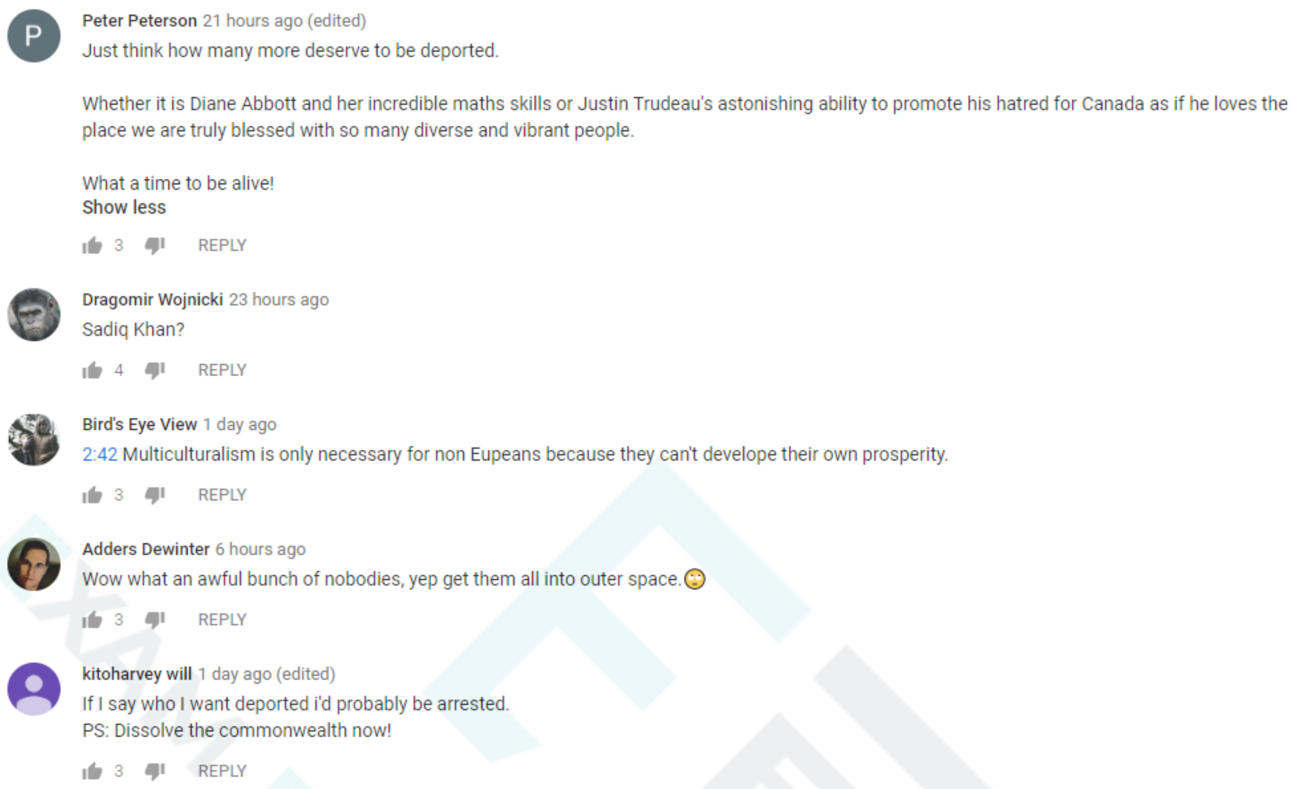
xcomfan 1 day ago
Deport Don Lemon to the moon
12 REPLY
View 2 replies

Reunite The British Empire 22 hours ago
"our ancestors built this country" Oh, so now Britain is made out of cotton?
10 REPLY
View reply

Rob MOORE 1 day ago
Teresa may and her dead peodophile father HUBERT BRASIER,,,and Sadiq Kahn,,,all at the BBC,,,and the royal family.
16 REPLY

Sensual Digest 1 day ago
Sadiq Khan, Tony Blair, Corbyn.
7 REPLY

Michael Kerr 1 day ago (edited)
Multiculturalism is stage four social cancer. The only cards stacked against black people is their average low IQ's.....Blame God!
7 REPLY


Peter Peterson 21 hours ago (edited)
Just think how many more deserve to be deported.

Whether it is Diane Abbott and her incredible maths skills or Justin Trudeau's astonishing ability to promote his hatred for Canada as if he loves the place we are truly blessed with so many diverse and vibrant people.

What a time to be alive!
Show less
3 REPLY

Dragomir Wojnicki 23 hours ago
Sadiq Khan?
4 REPLY

Bird's Eye View 1 day ago
2:42 Multiculturalism is only necessary for non Eupeans because they can't develope their own prosperity.
3 REPLY

Adders Dewinter 6 hours ago
Wow what an awful bunch of nobodies, yep get them all into outer space.🙄
3 REPLY

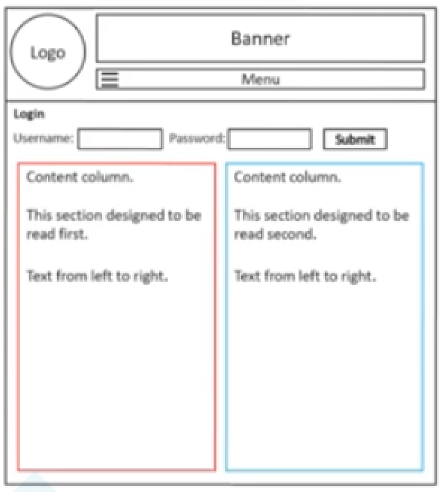
kitoharvey will 1 day ago (edited)
If I say who I want deported i'd probably be arrested.
PS: Dissolve the commonwealth now!
3 REPLY

Now I am no political expert, but any sane person reading comments like this would always ask themselves "What are these people on about? Can they even think?" Because nothing which trolls like these say makes sense, not to mention all of these people WANT to start arguments or some kind of aggressive behaviour online. Typical troll behaviour

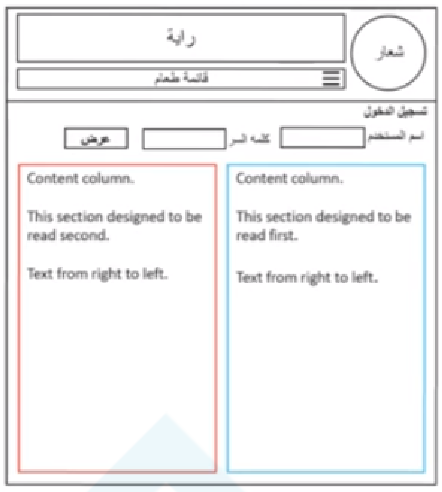
Layout, colour paradigm and character sets

In different countries, text on webpages are layered differently. In the west, writing on a webpage will start from the left whilst in Arabic writing will start from the right:

Typical Western Layout to a web page



Typical Layout for a webpage using Arabic text



So, it will be more appropriate to put some of the user interface features on the left when creating a website, to make it more suitable for everyone in your audience to use it.

Also, on the topic of colours, colours mean different things to different people. To some people, the colour red may mean love and caring, but to someone else it can mean danger, evil etc...

Colours are also seen different by many cultures, in the west green means nature and luck, but in China it means infidelity (unfaithful). So, colours need to be chosen carefully when developing a website for international use

And finally, for character sets, if your website is for an English audience then just use ASCII, otherwise use UNICODE.

OCR A level Computer Science, Algorithms and Programming

This section will be focusing on the second paper, the second paper there is a lot of programming and logic apply questions so there is not much to learn. However, you will have to use your brain in order to get the top marks in this paper.

Elements of computational thinking

Spec reference:

(a) The nature of abstraction.

Abstraction is removing unnecessary detail, and only keeping or adding the relevant details useful to the user. An example would be the London Underground map, that map is easy to understand right?

Spec reference:

(b) The need for abstraction

We need abstraction to make it easier for users to understand certain models. For example, a tourist would only want a city map, a satellite map of the city is still a map of the city but the tourist will find it very hard to use it for their needs. Therefore, they are better off using a tourist map, and that is an abstraction of the satellite map:

Satellite map of London

Tourist map of London

Spec reference:

(c) The differences between an abstraction and reality.

Well...isn't it obvious? Reality is reality and abstraction is...not the reality lol. Reality is where no detail is removed or added to a model, it's kept exactly as it is. In an abstract model, certain things are removed, added or highlighted. So let's go back to our satellite and tourist map. The tourist map has all the physical features like grass etc..removed. It has many famous tourist attractions highlighted and roads are given label as well as highlighted. All the small urban areas have been removed and only the big ones are highlighted.

Spec reference:

(d) Devise an abstract model for a variety of situations.

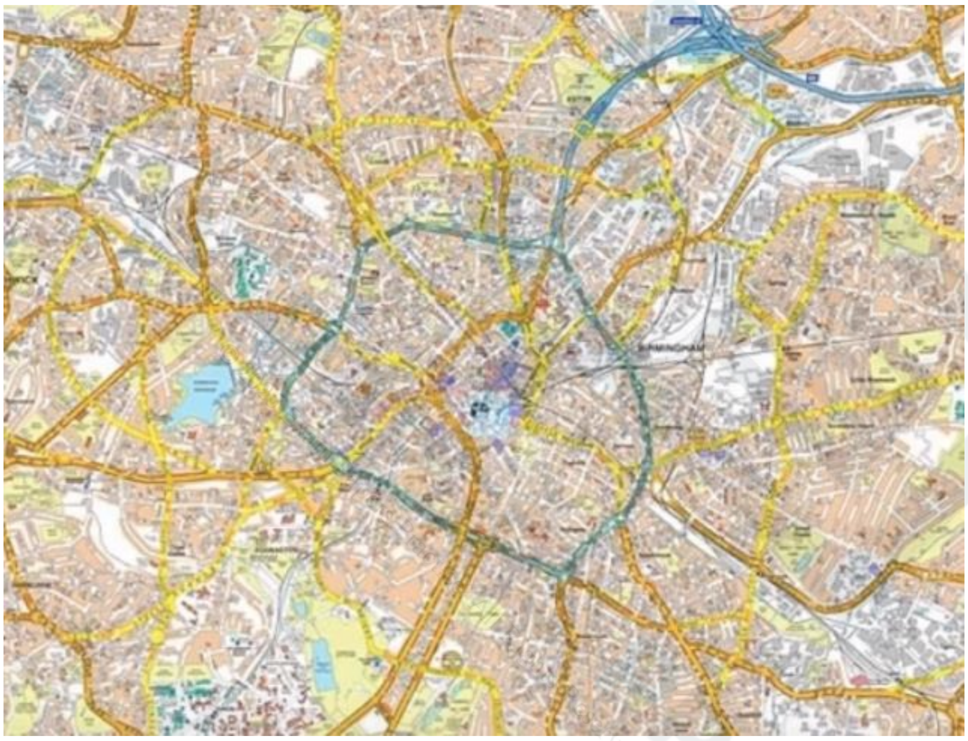
This is where you will be given certain scenarios in the exam and be asked to propose an abstract model of something. These are the tips you can follow:

1. What does the user want?
2. Who is the user?
3. What is the scenario?

4. What are the unnecessary details to be removed?

When answering questions on this, just name the features of the abstract model, don't say what would you remove AND add instead as it will take too long. Just say what would you add, highlight or remove in separate sentences.

So, for example, propose an abstract model for this map for a tourist:



1. Highlight important roads
2. Give road labels
3. Highlight important tourist attractions
4. Remove urban areas which are not significant
5. Highlight public services
6. Add a key

You don't say: "Remove the useless roads and highlight important roads" all in one sentence as:

1. They kinda mean the same thing and

2. It's just too long and you won't be given enough space to write it in the exam

Spec reference:

(a) Identify the inputs and outputs for a given situation

This is where you will be given a scenario to develop a program or something like that, and you will have to identify the inputs and outputs of the program or algorithm that you are going to create. So, for example to create a program which runs ATMs, what would the outputs and input required for the programs?

ALWAYS start with the outputs as this is generally easy, a program cannot be a program if it does not give any outputs. So, the outputs (showing the results of the processing TO THE USER) of the ATM program would be:

1. Screen display of options and balance
2. Printing a receipt
3. Moving the ATM lid to dispense cash

Now that you have your outputs, it's easy to identify the inputs, what inputs would be needed in order to generate these outputs: (remember an input is anything which the user inserts into the program)

1. PIN number
2. Card read
3. Selecting the amount of cash to receive
4. Selecting the option of wanting the receipt

Of course, always think of the obvious inputs before moving to the more detailed ones likes 3 and 4.

Spec reference:

(b) Determine the preconditions for devising a solution to a problem

This is where you will have to determine existing conditions which can affect the development of your solution, here are the questions you can ask yourself before developing a program?

1. Is the solution technically feasible?
2. Is the solution economically feasible?
3. What is the budget available?
4. How much time is there to develop the solution?
5. Is the solution compatible with different operating systems?
6. Does the solution have to cater to disabled people as well?

Spec reference:

(c) The nature, benefits and drawbacks of caching.

Caching is when instructions or data are moved from the hard disk on secondary storage to the main memory just in case they are needed again by the program using them until it ends. A more advanced version is prefetching, where instructions and data are fetched from the RAM to the registers or cache within the CPU before the CPU actually requires them.

(This is how caching works which you looked at in the second spec point way back, but you don't really need to know this)

Benefits of caching:

- Results in faster processing speed

- Results in more efficient processing

Drawbacks of caching:

- It is difficult to program it
- The wrong instruction or data may be fetched, so it has to be thrown away to fetch the right one. This can disrupt the sequence of the data or instructions which will make it hard to maintain their sequence

Spec reference:

(d) The need for reusable program components.

The reason why we need to reuse program components is because it saves time and money to develop a code. Here are the ways in which you can reuse components of a program:

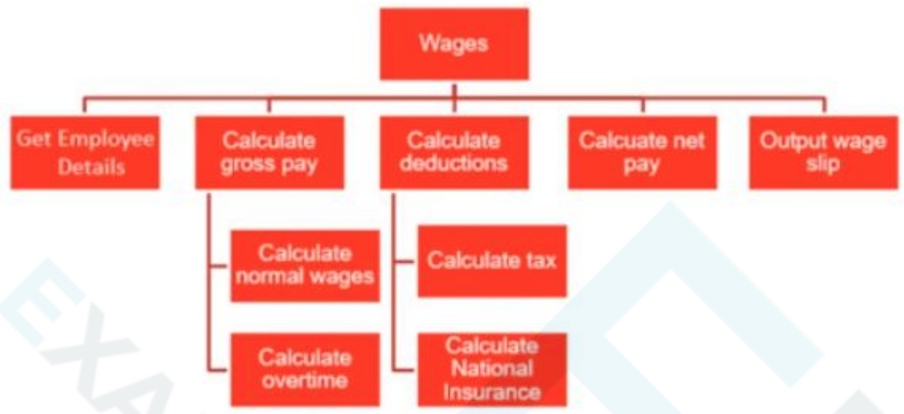
- Code the components as functions or procedures
- Use a library of code
- Use a component of one program across different programs (Basically use a feature of a program like a button from one program into other programs like Word etc..)
- Sell a component of a program to a third party (Like Facebook, this is why when you want to invite friends into a game, it links you directly to a screen with Facebook style)

Spec reference:

(a) Identify the components of a problem.

This section is about stepwise refinement.

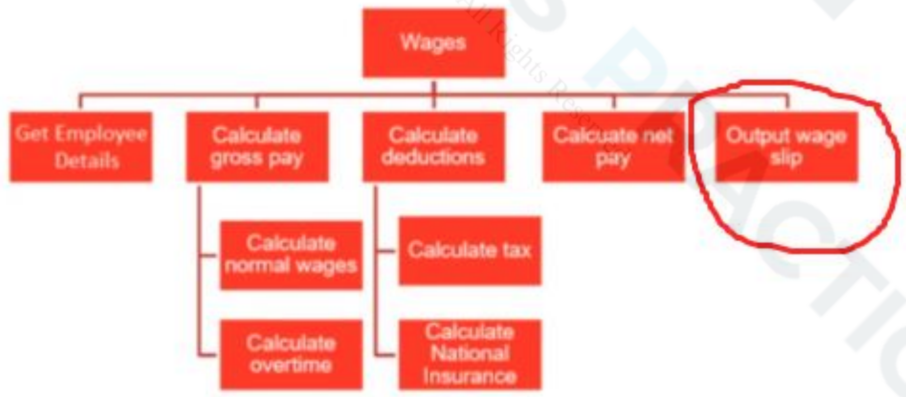
Stepwise refinement is when a problem is spilt into smaller tasks, and those smaller tasks are further split into smaller sub-tasks until a programmable solution for each of them can be found.



Spec reference:

(b) Identify the components of a solution to a problem.

This is when after you have done stepwise refinement, you need to identify the components which will allow you to solve the sub-tasks. So, let's go back to our wages diagram:



In order to solve this task, we need to identify the components (things we need) which will help us solve this task. So, to output a wage slip you need:

- The employee's home address

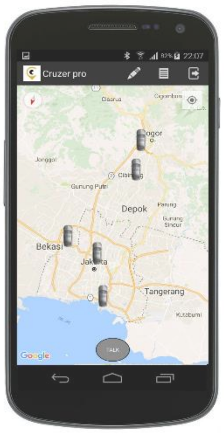
- Employee's name
- Number of hours worked
- Pay per hour
- Employer's name
- National Insurance Number

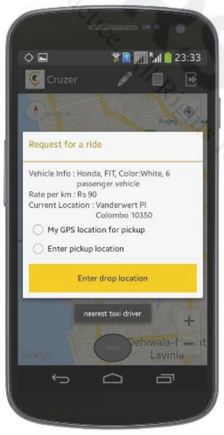
And many other things in order to output a wage slip

Spec reference:

(c) Determine the order of the steps needed to solve a problem.

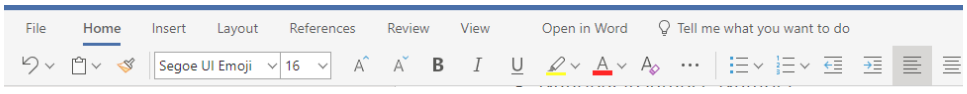
When writing a program, you need to think of the order of steps that a user will have to take in for the program to work. For example, when creating a taxi booking app, the user will first have to put in their starting place and their destination. THEN they can select which taxi car they may want, and then they can choose their prices and so on.





Vehicle Info : Honda, FIT, Color:White, 6 passenger vehicle
Rate per km : Rs 90
Current Location : Vanderwert Pl Colombo 10350
◯ My GPS location for pickup
◯ Enter pickup location

However, in a program like Word, there is no way to predict the user's order of steps. You can't really use Word in a particular way to make it work, so you will have to program your solution in a way which lets the user access the various features of it in any order



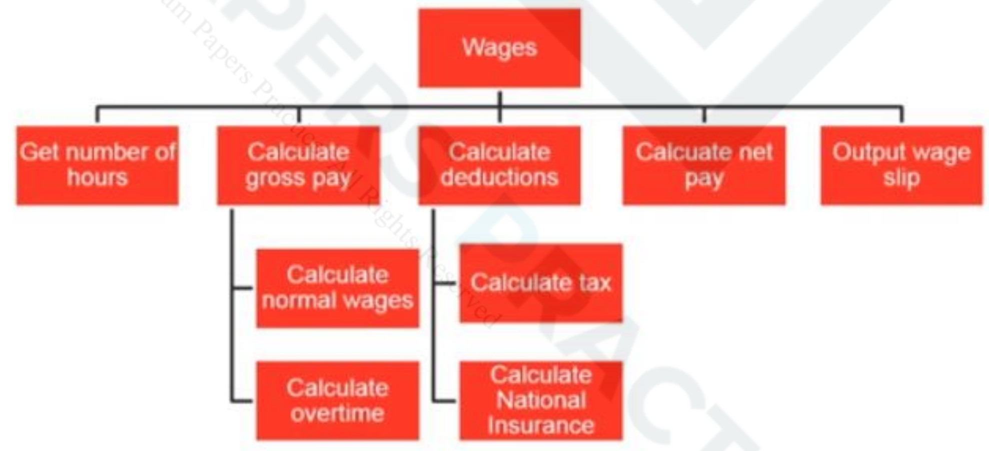
Spec reference:

(d) Identify sub-procedures necessary to solve a problem.

Literally the same as the following spec point:

(a) Identify the components of a problem.

Except with procedures instead, so if you are going to create a procedure which will calculate wages, you need to create sub-procedures of it until you can find a programmable solution to each of them. So, in order to create a procedure which will calculate wages, I need a procedure which calculates the daily pay, a procedure which deductions, a procedure which calculates the number of hours worked etc....basically the sub-procedures are everything in your stepwise refinement diagram:



Wages	
GetHours()	function which returns an integer in range 0 to 60
CalculateWages(Hours)	function which returns gross wage
CalculateNormalWages(Hours)	function which returns wage for up to 40 hours
CalculateOvertime(Hours)	function which returns pay for any hours over 40
CalculateDeductions(GrossWage)	function which returns total decutions
CalculateTax(GrossWage)	function which returns tax due
CalculateNI(GrossWage)	function which returns NI due
CalculateNetWage(GrossWage, Deductions)	function which returns net wage after deductions
OutputResults(Hours, GrossWage, Tax,NI, Deductions, NetWage)	Procedure to print the wage slip

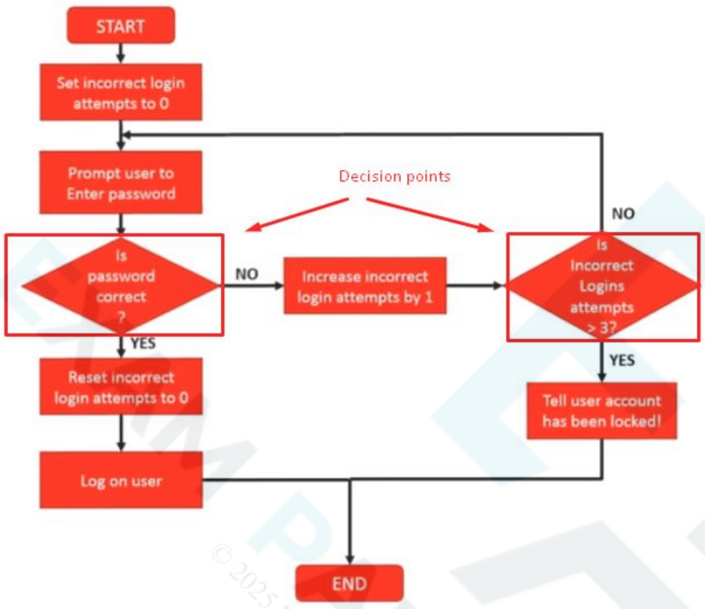
Spec reference:

(a) Identify the points in a solution where a decision has to be taken.

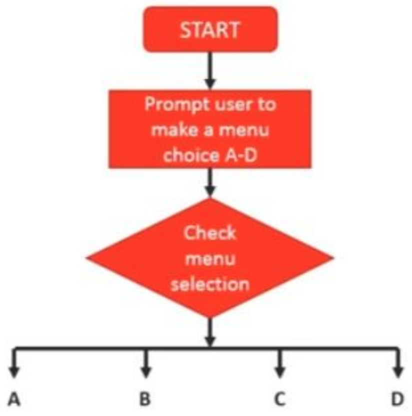
(b) Determine the logical conditions that affect the outcome of a decision.

(c) Determine how decisions affect flow through a program.

This is basically flowcharts, no like literally I am being serious.


START
Set incorrect login attempts to 0
Prompt user to Enter password
Is password correct ?
NO
YES
Increase incorrect login attempts by 1
Decision points
Is Incorrect Logins attempts > 3?
NO
YES
Reset incorrect login attempts to 0
Tell user account has been locked!
Log on user
END

With decision points of a flowchart, it is possible to have many decisions called cases like this:



```
SELECT CASE <Test expression>
   CASE <Expression list>
      ...
   CASE Else
      ...
END SELECT
```

Spec reference:

(a) Determine the parts of a problem that can be tackled at the same time.

All I can say here is an example: Imagine we are building a house, whilst one guy fits the roofs, another one can keep on making the walls, and another one can make the floors, and another one can install the pipes and electricity etc..all at the same time.

The same can be done with programs, where different codes can each run on more than IDE (Like Python GUI) at the same time.

Spec reference:

(b) Outline the benefits and trade-offs that might result from concurrent processing in a particular situation.

Concurrent processing: This is when more than one process from the same program is being executed, with each process being given a slice of the processing time.

Parallel processing: This is literally the same as Multicore processing, when more than one core is each executing instructions the same time

Benefits of concurrent processing:

- More instructions are executed per second
- Processor time is not wasted

But a drawback would be:

- Processing speed can slow down if more than one user is trying to perform the same action

Benefits of parallel processing:

- More instructions are being executed at the same time

- Improves the performance of graphics processing

But a drawback would be:

- It’s pretty expensive as more cores have to be brought

Problem solving and programming

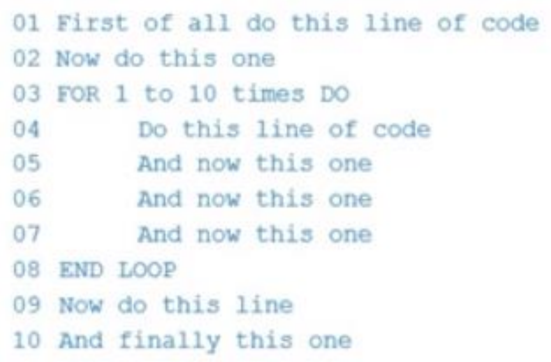
Spec reference:

(a) Programming constructs: sequence, iteration, branching.

Three programming constructs are: sequence, iteration and branching.

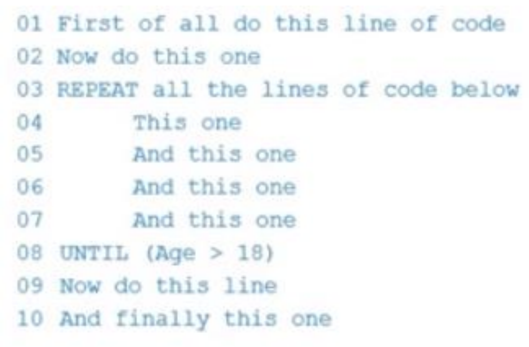
Branching/Iteration: This is when a set statements are executed again for a certain number of times. There are two types of branching: count controlled iterations and condition controlled iterations.

Count controlled iteration is when a set of statements are executed again for a set number of times:



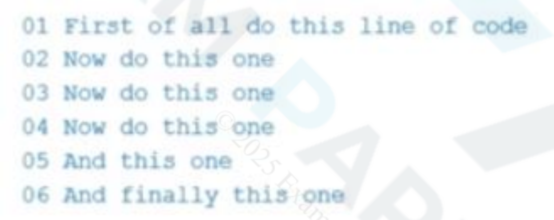
```
First of all do this line of code
Now do this one
FOR 1 to 10 times DO
      Do this line of code
      And now this one
      And now this one
      And now this one
END LOOP
Now do this line
And finally this one
```

Condition controlled iteration is when a set of statements are executed again and again until a condition is met:



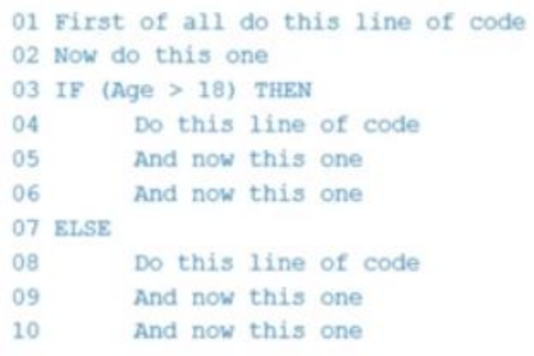
```
First of all do this line of code
Now do this one
REPEAT all the lines of code below
      This one
      And this one
      And this one
      And this one
UNTIL (Age > 18)
Now do this line
And finally this one
```

Sequence is when statements are executed in order, one after another:



```
First of all do this line of code
Now do this one
Now do this one
Now do this one
And this one
And finally this one
```

Selection is when a set of statements are executed based on a condition:



```
First of all do this line of code
Now do this one
IF (Age > 18) THEN
      Do this line of code
      And now this one
      And now this one
ELSE
      Do this line of code
      And now this one
      And now this one
```

Spec reference:

(b) Recursion, how it can be used and compares to an iterative approach.

Recursion is when a function calls itself, without user input.

Before we dive into some example, you need to know what a factorial number is.

A factorial number is the product of all the numbers multiplied together, which are all less than the original number (So like 5x4x3x2x1 = 120 or vice-versa 1x2x3x4x5 = 120 so 120 is the factorial number of 5, or 3x2x1 = 6 so 6 is the factorial number of 3 and so on. Remember, you don’t need to know the actual definition, just know what it is)

A pseudocode for a factorial number generator in iteration mode will be:

```
Function factor(n):
Answer = 1
Counter = 1
For counter = 1 to n:
    Answer = Answer x Counter
Next
Factor = Answer
End function
```

A pseudocode for a factorial number generator in recursive would be:

```
Function factor(n):
If n = 0 THEN
    Factor = END
Else THEN
```

```
        Factor = n x factor(n-1)
    End if
    End function
```

Recursion is not very efficient for memory, this is because each time a recursive function call itself, the memory has to remember where it is before executing the new call so it knows where to **return** afterwards. It also stores all the values produced by the variables because they are all local to the function

This is done by using stacks.

Spec reference:

(c) Global and local variables

Local variables are defined within a **module** and are only accessible and usable within that module. They cease to exist once the module in which they are in has finished executing. They can overwrite the values of global variables with the same names.

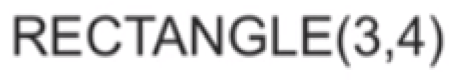
Global variables are defined at the start of the program and are accessible and usable throughout the whole program. They cease to exist when the program stops executing.

Spec reference:

(d) Modularity, functions and procedures, parameter passing by value and by reference.

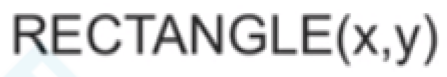
The difference between a function and a procedure is that a function takes in parameters, carries out a specific task and **returns a value** whilst a procedure takes in parameters and carries out a specific task but does not return anything.

Parameters can be passed by their value or reference. By value is when the actual data to be used are passed on:



```
RECTANGLE(3,4)
```

By reference, is when the locations containing the data to be used, are passed on



```
RECTANGLE(x,y)
```

Spec reference:

(e) Use of an IDE to develop/debug a program.

An IDE is a user interface which provides a set of tools and resources to help a programmer code. The most common example you know very well about, is Python GUI.

An IDE has the following tools which help the programmer to code:

- Run-time environment: Allows the programmer to run the code during development time to check for logical errors
- Code editor: Comes with features like automatic indentation and syntax completion to help speed up programming
- Error diagnosis: Finds errors, points to their location and suggests solutions to them
- Translators: These compile high level or low level language code into executable machine code

- Auto-documentation: This tracks all the variables, modules and comments made by the programmer to help with program maintenance

Spec reference:

(f) Use of object-oriented techniques.

Basically, you only use object-oriented techniques when you want to create objects like buttons, icons or real life objects into a program like trees and monsters in a game etc...

Spec reference:

(a) Features that make a problem solvable by computational methods.

Although a problem can be technically solvable, it does not mean it's practically solvable. So, what do I mean by this?

For example, in order to crack the password of a bank account, you can do that using brute force, however it will take more than 1 million years to actually do that! And even then, the cracking will be like 2/3 complete so cracking the password of a bank account will be solvable with a computer but not in a practical sense as...come on we will all be dead including the universe after 1 million years!

Features which make a problem solvable by computation thinking:

- Enumeration, can the problem be solved by an algorithm which performs an extensive search?
- Simulation, can the problem be modelled in a computer?
- Theoretical approach, can the problem be broken down into pure theory? So, it can be solved by mathematics?

Spec reference:

(b) Problem recognition.

Of course, in A level Computer Science you will have to know the more advanced definition of problem recognition, this is the ability to know there is an issue or some parts of an existing process need attention.

When you need to recognise a problem in the exam, mainly with codes, ask yourself these questions?

1. Is there a problem?
2. If so, what is it?
3. What data do I need to fully understand this problem?
4. What variables are there which can change the current state of the problem?
5. What processes do I need to consider when trying to solve the problem?

And finally:

To what extent is this problem solvable?

Spec reference:

(c) Problem decomposition.

Same as spec point:

Spec reference:

(b) Identify the components of a solution to a problem.

Spec reference:

(d) Use of divide and conquer.

Divide and conquer is reducing the size of a problem with successive iterations.

So, for example when the police are trying to shortlist suspects, they use a set of criteria which build up in successive iterations to try and find their man.

So, a man has committed a bank robbery:

The police have their suspects and this is how it goes:

1. First remove all suspect above the age of 40
2. Then remove all suspects below the age of 20
3. Remove all female suspects
4. Remove all suspect who don't have facial hair
5. Remove all suspects who are below 6 foot tall
6. Remove all suspects who are above 6 foot tall
7. Remove all suspects who are not wearing a hoodie
8. Remove all suspects who are not wearing a hat
9. Remove all suspects without glasses

10. And finally remove all non-black suspects (Not being racist!)

And there you have your man.

Similar thing works with algorithms, perfect example is the binary search.

So here is a list of numbers:

8 5 9 3 7 5 0 9

Wanted number: 3

In binary search, first you order the list and then find the median of the list (the middle data item):

0 3 5 5 7 8 9 9

(5 + 7 which are the medians is 12, divide that in half is 6 so 6 is the median, even tho is not on the list, it's still a median and we use 6 as our median)

If the middle data item is less the wanted data item, the list containing all the data items less than the middle data item, including the middle data item itself is removed, otherwise, all data items above the middle data item including the middle data item itself is removed, a new middle point within the new list is worked out and the process is repeated again and again until the wanted data item it's found. You will learn about this in more detail later on.

Spec reference:

(e) Use of abstraction

Already covered by spec points:

Spec reference:

(a) The nature of abstraction.

(b) The need for abstraction

Spec reference:

(f) Learners should apply their knowledge of:

• backtracking

• data mining

• heuristics

• performance modelling

• pipelining

• visualisation to solve problems.

<u>Backtracking</u>

This is the process of building up towards a solution. Abandoning partial success when the solution can't be completed and taking an alternative path to complete the solution.

So, for example, you know when you are playing a game of chess, you have that "wonderful" strategy to beat your opponent, and late in the game you are like "Oh, you know what, this strategy sucks" because you can't beat your opponent and therefore you start to look for an alternative strategy. This is backtracking

<u>Data mining</u>

This is when large amounts of data are analysed and patterns are found, they are used to predict the next actions of a user.

It is used for recommending similar product to customers when they buy something online.

Big data is a word which means collecting large amounts of data on a particular topic like food, travel times, gaming etc..

Heuristics

This simply using experience to try and find a solution, really simple.

Heuristics is used because it may cost too much time and money to find an exact solution. So, a solution produced by heuristics is good enough but it won't be the best solution.

Heuristics is also known as "The rule of thumb" methods

Performance modelling

This is the process of using mathematical approximations to test **models**.
It may be used because actually testing the **models** may be:

- Unpractical (If a game needs to be tested with more than 1 million players to see how it handles that many requests, it would be very hard to do this, so mathematical approximations are used)
- Very costly

Pipelining

Same as spec point:

(d) The use of pipelining in a processor to improve efficiency.

Pipelining obviously results in faster processing speed

Visualisation

This is the ability to picture a problem and a solution (Obviously, something ALL human beings use)

This is mainly about using diagrams and flowcharts, visual ways of representing solutions.

Algorithms

Spec reference:

(a) Analysis and design of algorithms for a given situation.

Again, this is just flow chart and pseudocode. You need to know how to follow algorithms, read these steps in order to help you:

1. What does the algorithm actually do? What is its purpose?
2. Work through each line of the algorithm step by step
3. Apply a different set of data to the algorithm and work through it again

Spec reference:

(a) Analysis and design of algorithms for a given situation.

(b) The suitability of different algorithms for a given task and data set, in terms of execution time and space.

There are two algorithms for searching:

1. Linear search

2. Binary search

This is Binary search:

So here is a list of numbers:

8 5 9 3 7 5 0 9

Wanted number: 3

In binary search, first you order the list and then find the median of the list (the middle data item):

0 3 5 5 7 8 9 9

(5 + 7 which are the medians is 12, divide that in half is 6 so 6 is the median, even tho is not on the list, it's still a median and we use 6 as our median)

If the middle data item is less the wanted data item, the list containing all the data items less than the middle data item, including the middle data item itself is removed, otherwise, all data items above the middle data item including the middle data item itself is removed, a new middle point within the new list is worked out and the process is repeated again and again until the wanted data item it's found. You will learn about this in more detail later on.

Linear search is when the wanted data item is compared with every single data item in the list from the start until it's found. It's simple and the data items do not need to be ordered, however it's pretty time consuming.

And then you have two algorithms for sorting:

1. Bubble sort
2. Insertion sort

With a bubble sort, two data items near each other in a list swap their places if they are in the wrong order, then again two data items near each other (including the one which was just swapped) swap their positions if they are in the wrong order and so on:

Example:

5 6 7 32 5 3

So 5 and 6 are fine as they are in order, 6 and 7 are fine as they are in order, 7 and 32 are fine as they are in order, however 32 and 5 are not fine since they are not in order, so we swap the places of 32 and 5:

5 6 7 5 32 3

Again, 32 and 3 are not fine since they are not in order, so they need to have their places swapped:

5 6 7 5 3 32

And we simply work through this list again, using the same process until the list is in order

With an insertion sort method, it's pretty simple, basically take one item from the list, one at a time and place it directly into the correct position:

7 9 5 8 3 2 6

So, 7 just stays there, 9 also stays there, but 5 needs to be put before 7:

5 7 9 8 3 2 6

Now we work through the list again, from the start; so 5 is fine, 7 is fine, 9 is fine but 8 is not and needs to be put before 9:

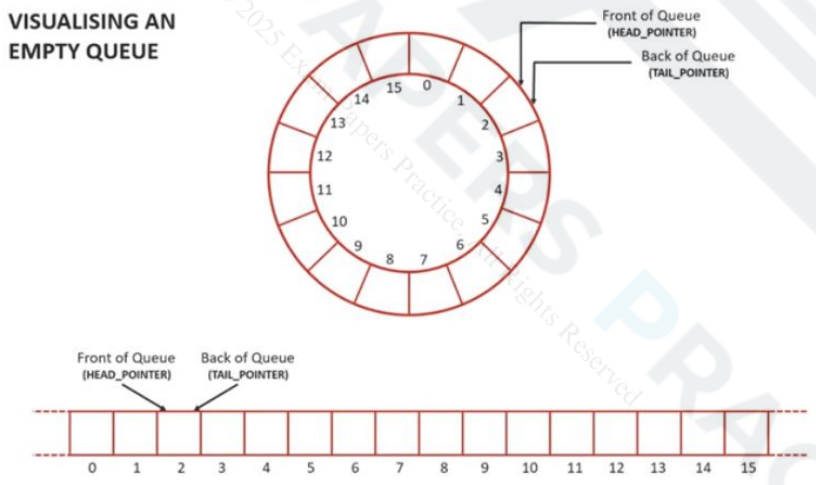
5 7 8 9 3 2 6

And we work through the list again, from the start, 3 is not fine so we put it into the correct order:

3 5 7 8 9 2 6

And repeat the same process again, until we have an ordered list.

This is a queue by the way:

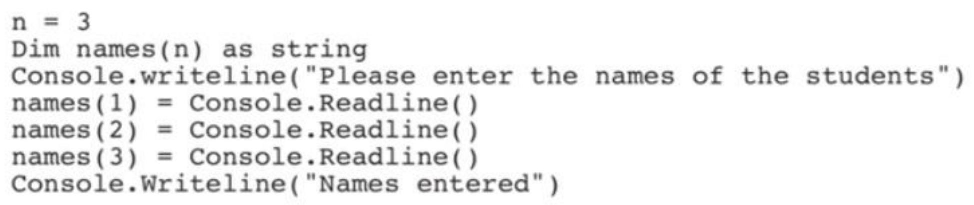

VISUALISING AN EMPTY QUEUE
Front of Queue (HEAD_POINTER)
Back of Queue (TAIL_POINTER)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Front of Queue (HEAD_POINTER)
Back of Queue (TAIL_POINTER)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

And it works exactly in the same way as a normal queue, in terms of adding, removing and reading data. The only difference is its visualisation

Spec reference:

(c) Measures and methods to determine the efficiency of different algorithms, Big O notation (constant, linear, polynomial, exponential and logarithmic complexity).

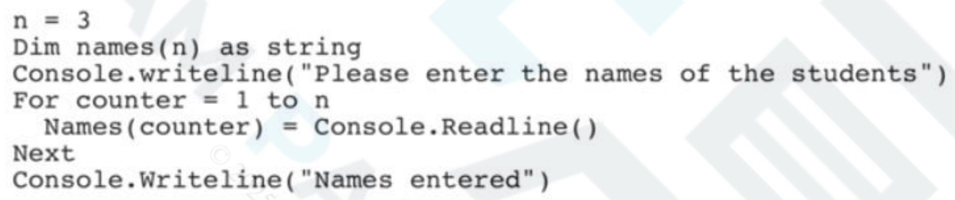
You can calculate the complexity of the code in two ways. One is where you count the lines of the code:



```
n = 3
Dim names(n) as string
Console.writeline("Please enter the names of the students")
names(1) = Console.Readline()
names(2) = Console.Readline()
names(3) = Console.Readline()
Console.Writeline("Names entered")
```

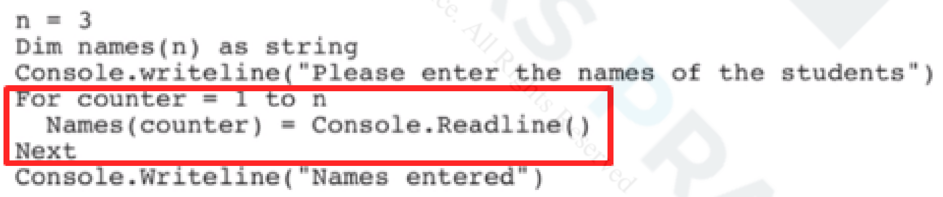
This code has 7 lines therefore it has a complexity of 7

However, with a code that has loops, you will have to find out its complexity the other way:



```
n = 3
Dim names(n) as string
Console.writeline("Please enter the names of the students")
For counter = 1 to n
  Names(counter) = Console.Readline()
Next
Console.Writeline("Names entered")
```

So it has 7 lines of code BUT it has a FOR loop and three of those lines:

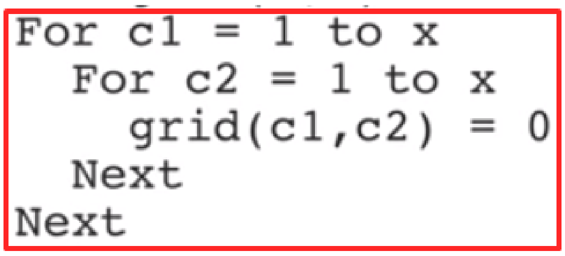


```
n = 3
Dim names(n) as string
Console.writeline("Please enter the names of the students")
For counter = 1 to n
  Names(counter) = Console.Readline()
Next
Console.Writeline("Names entered")
```

Are part of the FOR loop, so the complexity of the code will be 3 +3n +1

The 3 is the first 3 lines of code, 3n being the FOR loop and 1 is the last line of code.

What about for a code which contains nested FOR loops like this:



```
For c1 = 1 to x
   For c2 = 1 to x
      grid(c1,c2) = 0
   Next
Next
```

Part of the nested FOR loops

In that case, the complexity of the code will be $5n^2$

In order to turn the complexity of a code into big O notation you firstly start off with the complexity of a code:

3 + 3n + 1

And follow these two rules:

1. Remove all values except for the ones with the biggest factor or exponent (squaring)
2. Remove any constants (the numbers)

So we remove 3 and 1 (As 3n is the only one with an exponent, 3n:

$\cancel{3} + 3n + \cancel{1}$

And we are left with 3n, now we follow the second rule which is remove any constants, so we remove the 3 near the n:

$\cancel{3}n$

And are left with just n, put a big O and some brackets and you got yourself O(n) which is the big O notation.

There are many different big O notations, you need to learn them all:

O(1) Constant complexity – This is where the algorithm always executes in the same time, no matter what the size of the data set is. This is the best algorithm

O(log n) Logarithmic complexity – This is where the algorithm halves the data set with each execution. It's very efficient with large data sets. It's a good algorithm

O(n) Linear complexity - This is where the performance of the algorithm will be **proportional** to the size of the data set. So, if the data set shrinks, the performance will also shrink. It's a fair algorithm

O(n^2) Polynomial complexity - This is where the performance of the algorithm is proportional to square size of the data set. It will massively have its efficiency reduced with large data sets. It's a poor algorithm

O(2^n) Exponential complexity – This is where the algorithm will double its data set with each execution. It's a very inefficient algorithm and should be avoided AT ALL COSTS

Spec reference:

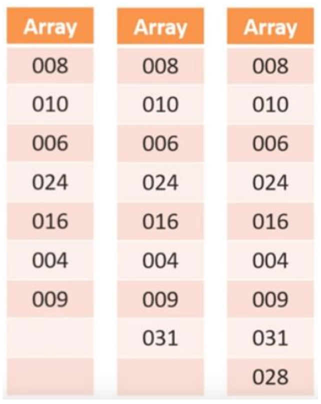
(d) Comparison of the complexity of algorithms

This section will be focusing on applying the different big O notations to different situations:

Here are the examples we will work through to help you understand better:

Example 1 -

Adding items to a list, just adding them, what Big O notation/complexity would it be?

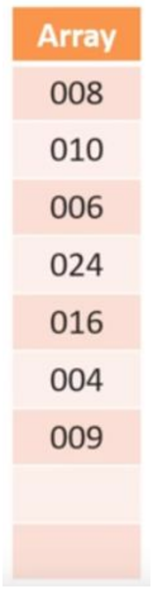


Array	Array	Array
008	008	008
010	010	010
006	006	006
024	024	024
016	016	016
004	004	004
009	009	009
	031	031
		028

Answer: **O(1) Constant complexity**, because the size of the data set does not matter as we are just adding data items into the array.

Example 2 -

Searching data items in the array, what Big O notation/complexity would it be?

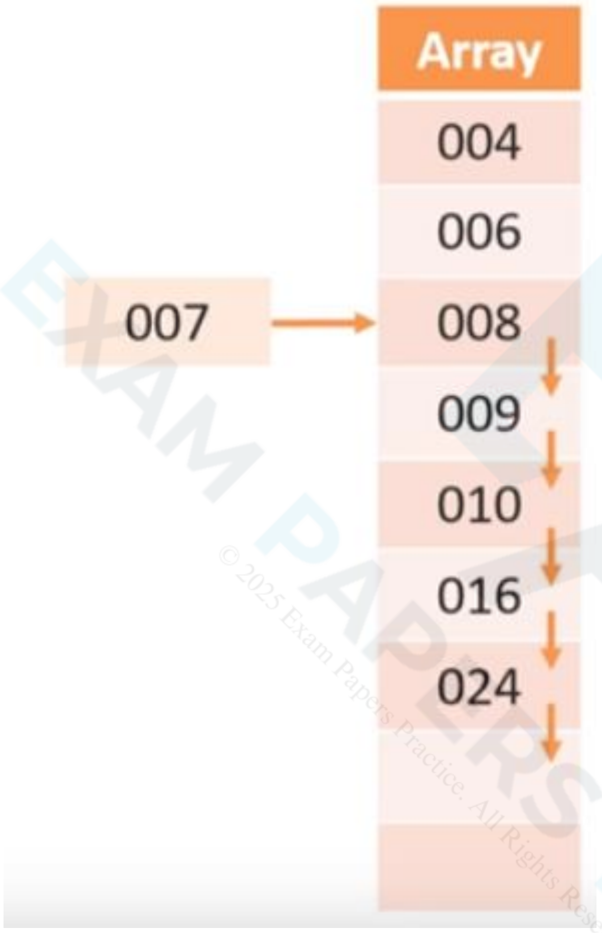


Array
008
010
006
024
016
004
009

Answer: **O(n) Linear complexity**, because the longer the data set, the more items you have to search through

Example 3 -

In order to add new items into an array, within the sequence of items in an array, the items will have to shuffle downwards, one at a time to make space for the new data item to be inserted. What Big O notation/complexity would it be?



Answer: **O(n) Linear complexity**, because the larger the data set means the algorithm will have to shuffle more items

Example 4 -

What about performing a binary search in an array? What Big O notation/complexity would it be?

Answer: Of course, **O(log n) Logarithmic complexity**, because in binary search you always have to halve your list in order to complete your search. So, the algorithm will have to halve its data set with each execution, to find the wanted data item

Example 5-

Pushing and popping data items from stacks and queues? What Big O notation/complexity would it be?

Answer: **O(1) Constant complexity**, because it does not matter how large the data set is, we are simply pushing data items into stack and queues, and popping them out

Example 6-

Searching through stacks and queues? What Big O notation/complexity would it be?

Answer: **O(n) Linear complexity**, because the larger the data set means the more data items, we will have to search through

Just remember that the Big O notation/complexity for a nested loop is obviously **O(n^2) Polynomial complexity** and for a recursive function it is **O(2^n) Exponential complexity** (Because you insert a data item into a recursive function, but since the function always calls itself, it technically doubles the data set with each execution. Or at least doubles the data set on the first execution)

Spec reference:

(e) Algorithms for the main data structures, (stacks, queues, trees, linked lists, depth-first (post-order) and breadth-first traversal of trees).

Well the algorithms for stacks, queues, trees and linked lists have already been covered. All you have to make sure is that you write the algorithms for adding, removing or searching data for the structures named above in pseudocode.

So, for example, for searching through a linked list, the pseudocode will be:

```
Start at data item pointed by the pointer value
REPEAT
    IF DataItem != WantedData THEN
        Update pointer value to the item value of the next
    data Item, according to the alpha pointer of the current
    data item
    UNTIL DataItem == WantedData
    ELSE THEN
        OUTPUT DataItem
    END IF
    END
UNTIL pointer value = 0
OUTPUT DataItem not found
```

Or a pseudocode for searching through a binary tree:

```
Start at root
IF WantedItem == Value in root THEN
    OUTPUT WantedItem
    END
ELSE:
    IF WantedItem > Value in root THEN
        Go right
    ELSE THEN
        Go left
    END IF
END IF
Node = Current Node
REPEAT
    IF WantedItem >Value in Current Node THEN
        Go right
    ELSE THEN
        Go left
    END IF
UNTIL WantedItem == Value in Current Node
ELSE THEN
OUPUT "Item not found"
```

<u>Traversing graphs</u>

There are two ways to traverse graphs:

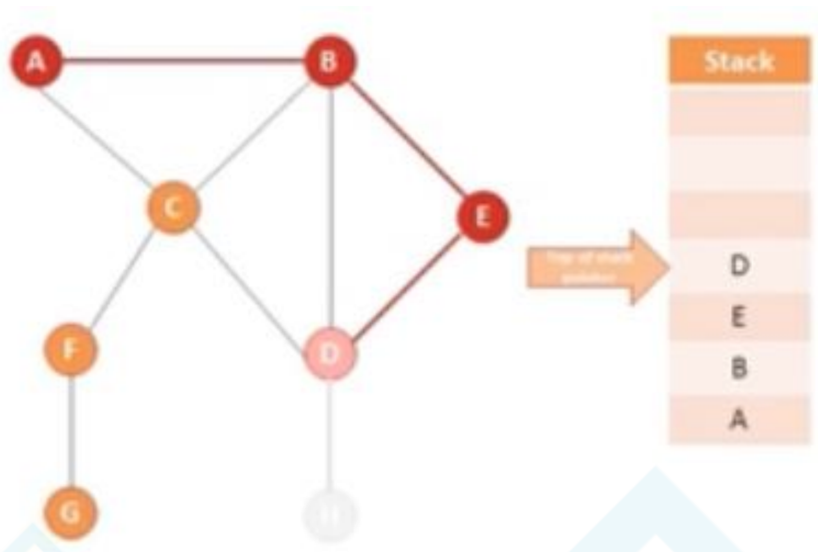
- Depth-first, this is where you visit the first vertex, and follow its branch as far as it will go and backtrack

- Breadth-first, visit the first vertex, then visit every vertex connected to it and continue visiting connected vertexes from left to right

A depth-first is done using a stack whilst a breadth-first is done using a queue.

This is the algorithm for a depth-first:

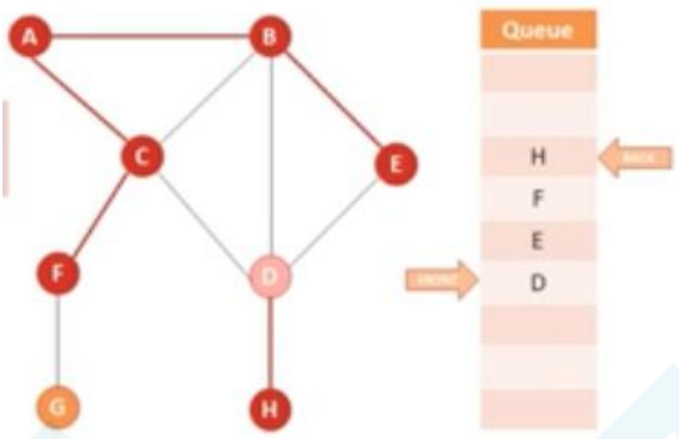
```
Visit the first vertex
Mark it as visited
Push vertex into the stack
REPEAT
    Visit the next unvisited vertex according to the one on top of the stack
    Mark it as visited
    Push vertex into the stack
    IF no more vertex to visit THEN
        Pop vertex off the stack
    END IF
UNTIL stack is empty
```



This is the algorithm for a breadth-first:

```
Visit first vertex
Mark it as visited
Push vertex into the queue
REPEAT
    Visit all vertexes connected to the first vertex
    Push the vertexes into the queue
UNTIL all vertexes have been visited
REPEAT
    Pop vertex pointed by the front pointer off the queue
    REPEAT
        Visit all vertexes connected to the current vertex
        Mark the vertexes as visited
        Push the vertexes into the queue
    UNTIL all vertexes have been visited
```

UNTIL queue is empty



Popping vertexes means removing them from the stacks or queues, the vertexes to be popped off are shown by a stack pointer for the stack and front pointer for the queue

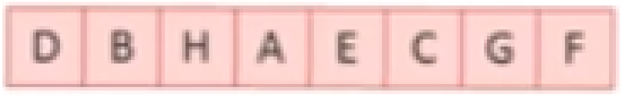
Spec reference:

(f) Standard algorithms (bubble sort, insertion sort, merge sort, quick sort, Dijkstra's shortest path algorithm, A* algorithm, binary search and linear search).

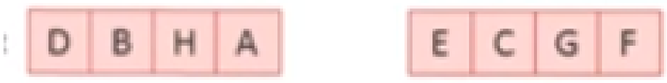
<u>Merge Sort</u>

It's kind of like binary search except it has its differences. Let's work through an example:

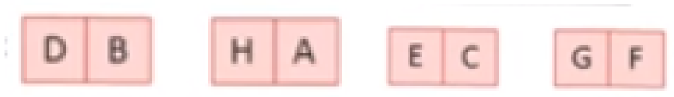
So here is the list of letters which we need to sort on alphabetical order:



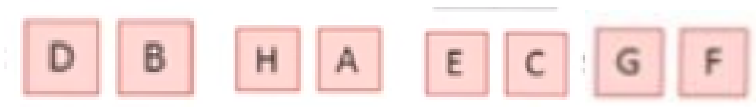
First, split the list in two:



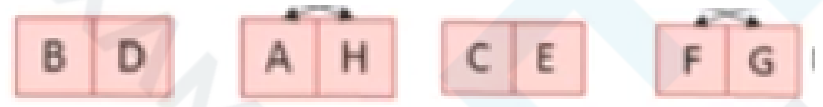
Then split the list further:



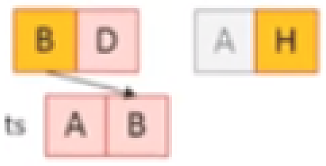
And split them even further! Lol:



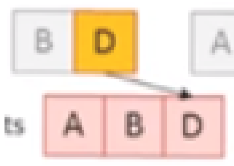
And now for each list (Two letters next to each other) you have to swap the position of the letters with each other if they are in the wrong order:

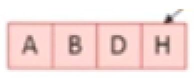


And now you have to work through these lists (2 groups of letters together in 4s) separately. Compare the position of the first letter in your first group (Which is B) with all the letters in the other group of your list (In this case is A and H) and put them in order, however if a letter comes after the position of the first letter (H comes after B) then don't put it down (So you don't put H down).

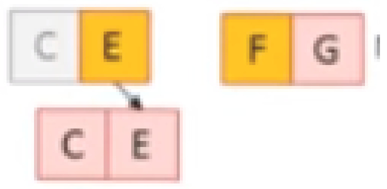


And do the same for the second letter (Which is D), except this time you put down all the letters:

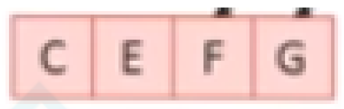




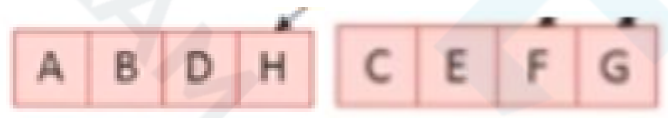
And follow the same process for the other list:



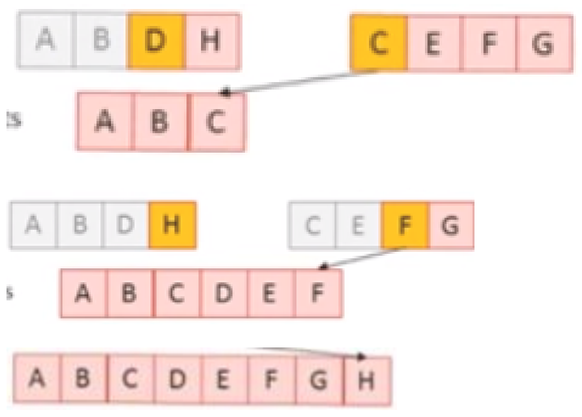
(Since bot F ang G come after C, you can’t put them down, only C)



So now you have two lists, but you are not really done yet:



All you have to do now, is compare the first letter in the first list with all the letters in the second list, and put all the letters that come before down, before putting that letter you use to compare with all the letters down. You then do the same with all the other letters in the first list, comparing them with all the letters in the second list and putting all the letters that come before them before putting those letters themselves down:



And now for some pseudocode of this lovely algorithm:

Some function definition to help you understand this pseudocode better:

MergeSort = Actually sorts out the values

SplitList = Splits the list

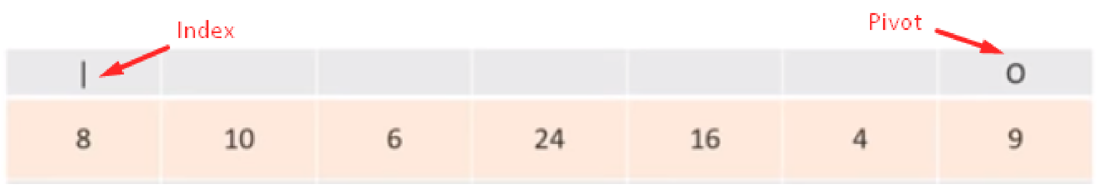
MergeTwoList = Merges the lists

```
Function MergeSort(Array)
        IF UpperBound(Array) == 0 THEN
        RETURN Array
        END IF
        END
Call SplitList(Array, ArrayRightHalf, ArrayLeftHalf)
ArrayRightHalf = MergeSort(ArrayRightHalf)
ArrayLeftHalf = MergeSort(ArrayLeftHalf)
RETURN MergeTwoList( ArrayRightHalf, ArrayLeftHalf)
End function
```

Quick Sort

Another way to sort out numbers, let's work through another example to see how quick sort is done.

So we want to order our numbers from lowest to the highest:



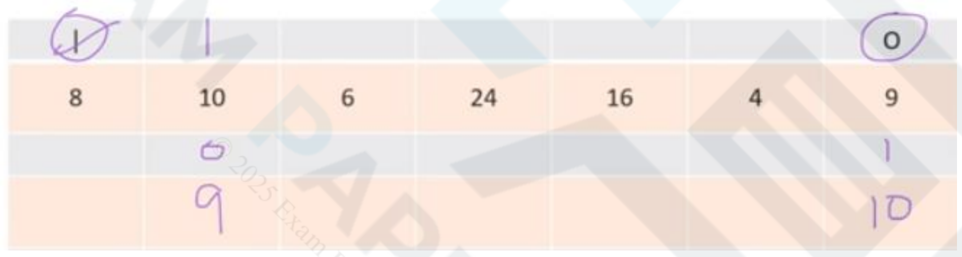
I						O
8	10	6	24	16	4	9

First we need to set some things, an index and a pivot and they HAVE to be in the opposite ends. Technically you can choose any pivot you like but it will just make life harder, here we are trying to make your life easier 😉

But still, the index HAS to be at the starting number. No exceptions

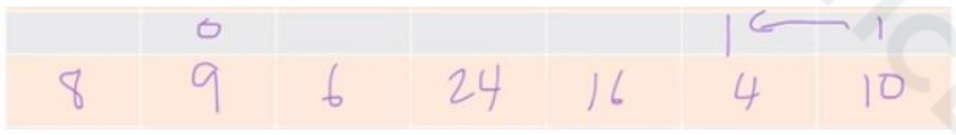
Now what you do basically is, since we are sorting our numbers from lowest to highest, we want to get all of our high numbers in the right side, and all of the lower ones in the left side.

The left side is before the pivot, the right side is after the pivot



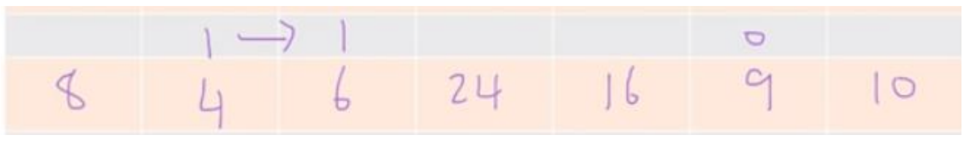
I	I					o
8	10	6	24	16	4	9
	o					I
	9					10

8 is not bigger than 9 so it stayed where it was and we moved the index, but once the index reached 10, 10 is bigger than 9 so they swap places. Notice the pivot and the index move with their numbers when they swap places, when they do the index ALWAYS moves towards the pivot. (Think of it as the pivot being the one always getting pushed around, as it does not change values)



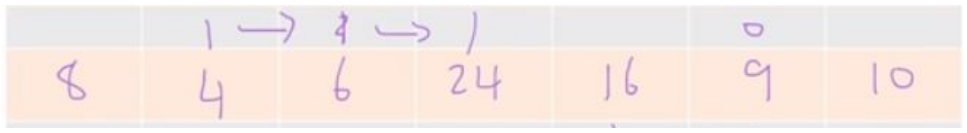
	o				I ←	I
8	9	6	24	16	4	10

4 is lower than the pivot 9, plus 4 is on the right side where all the big numbers should be, so 4 is swapped with 9:



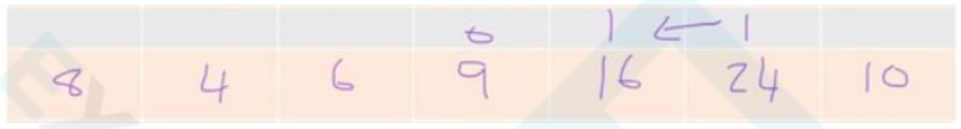
	I →	I			o	
8	4	6	24	16	9	10

Index keeps on moving towards the pivot, 6 is lower than 9 BUT 6 is on the left side, where all the small numbers should be so it stays there



	1 →	1 →	1		0	
8	4	6	24	16	9	10

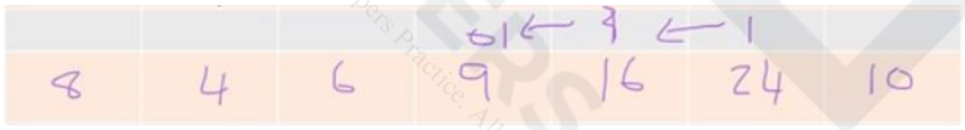
24 is bigger than 9 and is on the left side where all the small numbers should be, so it swaps:



			0	1 ←	1	
8	4	6	9	16	24	10

16 is bigger than 9 but does not get swapped because is not the right side, where all the big numbers should be so it stays there.

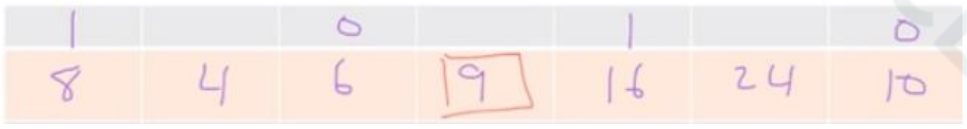
Now a point reaches where the pivot and the index meet each other:



			01 ←	1 ←	1	
8	4	6	9	16	24	10

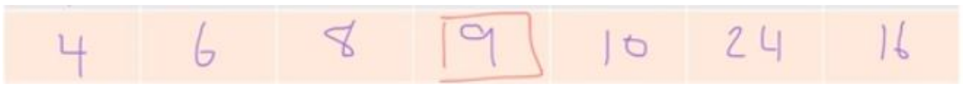
So 9 stays where it is and two separate new lists form, one containing all the low numbers (8, 4 and 6) and the others one containing all the high numbers (16, 24 and 10).

So two new lists are formed meaning two new indexes and pivots are made:



1		0		1		0
8	4	6	$\boxed{9}$	16	24	10

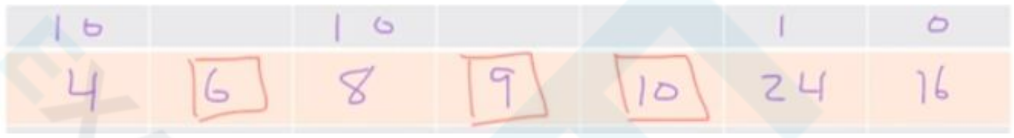
And same process and logic applies, all high numbers on the right, all the low ones to the left. So 8 is bigger than 6 so it swaps, then 4 is lower than 6 but it's on the left where the small numbers should be so it stays where it is:



4	6	8	$\boxed{9}$	10	24	16

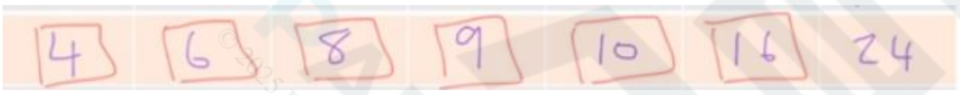
16 was bigger than 10 and it's on the left so it swaps. Then 24 is bigger than 10 but 24 is on the right side where all the big numbers should be, so it stays there. But the list is still not in order.

So once again there is a new list created, therefore there are new indexes and pivot:



1 6		1 6			1	0
4	$\boxed{6}$	8	$\boxed{9}$	$\boxed{10}$	24	16

24 is larger than 16 but is on the left side, where all the small numbers should be, so it swaps with 16:



$\boxed{4}$	$\boxed{6}$	$\boxed{8}$	$\boxed{9}$	$\boxed{10}$	$\boxed{16}$	24

FINALLY YOU HAVE YOUR ORDERED LIST

In all seriousness I think this is a very shitty way to sort out numbers, but hey that's how computers think apparently….

<u>Pseudocode for Binary Search</u>

You need to know the following pseudocode for Binary Search:

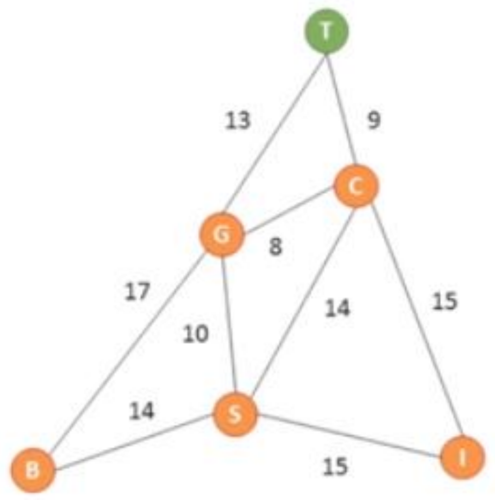
```
LowerBound = 0 (The lowest value of the list)
UpperBound = LengthOfList – 1 (The highest value in the list)
Found = FALSE
WHILE Found == FALSE AND LowerBound<=UpperBound THEN
MidPoint = ((LowerBound + UpperBound)/2)
IF MidPoint < WantedItem THEN
```

```
        LowerBound = MidPoint + 1 (Value after MidPoint
becomes the lowest value)
ELSE THEN
        UpperBound = MidPoint – 1 (Value before MidPoint
becomes the highest value)
END IF
END WHILE
IF MidPoint == WantedItem THEN
        OUTPUT "Wanted item is at", + MidPoint
ELSE THEN
        OUTPUT "Wanted item, not in list"
END IF
```

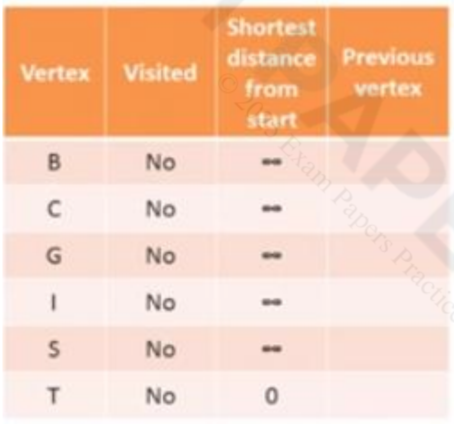
<u>Dijkstra's shortest path</u>

Shit name, shit concept. Although is not hard, still a confusing concept to get your head around. Come on, you are almost done, just this topic and one more...

So Dijkstra's shortest path is as it suggests in the name, it's about finding the shortest path within a graph. So let's work through an example as with these kind of things, the best way to understand is through examples:



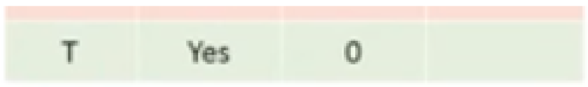
So, here is our graph. We are trying to find the shortest path between T and S. You also need this table to keep track of your paths:



Vertex	Visited	Shortest distance from start	Previous vertex
B	No	∞	
C	No	∞	
G	No	∞	
I	No	∞	
S	No	∞	
T	No	0	

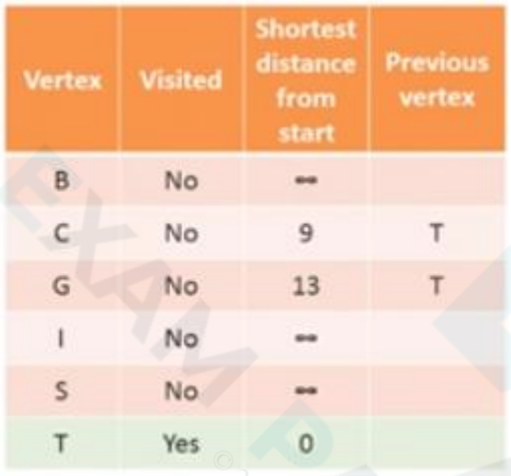
The values in the shortest distances from the start, are all currently infinitive (Like forever values of millions).

So first of all, we visit the first node and we insert the previous node in the previous vertex column. In this case, T has a value of 0 in the shortest distance column because T is our starting point, so there is no distance. And there are no previous vertexes either as T is our starting point. So we put nothing down in the previous vertex column for T, and we mark it as visited.



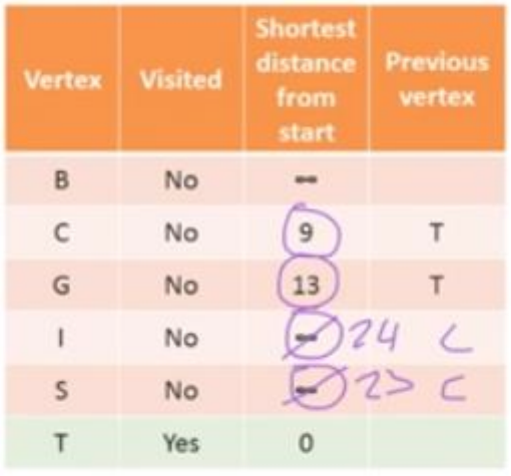
T	Yes	0	

Now we work out the distances between T and all the vertexes connected to T and those are G and C. So in the columns for C and G we are going to put their distances from the start (T) and their previous vertexes:



Vertex	Visited	Shortest distance from start	Previous vertex
B	No	--	
C	No	9	T
G	No	13	T
I	No	--	
S	No	--	
T	Yes	0	

Now what you do is visit the vertex with the lowest distance from the start, and it has to be a vertex that has not been visited. Vertex C fits that description so we will visit vertex C and mark it as visited:

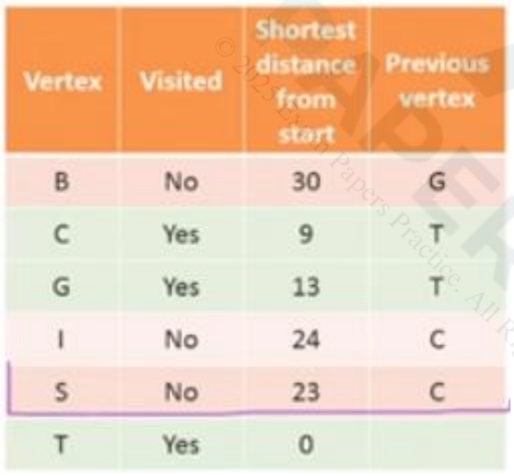


Vertex	Visited	Shortest distance from start	Previous vertex
B	No	--	
C	No	9	T
G	No	13	T
I	No	-- 24	C
S	No	-- 25	C
T	Yes	0	

So, what happened was that we worked out the distances between all the vertexes connected to C which are I, S and G HOWEVER, since the column it says shortest distance from the

START, we add up the distance values between C and I or S or G and add it with the distance value between C and T (The starting point) which gives the values 24 for I and 23 for S. Since they are lower than the infinitive values, we put them down, but for G when we add 9 and 8 together it gives us a distance of 17 from the starting point, we already have a value for vertex G which is 13 for the **shortest distance from the starting point** and 17 is higher than 13 so we don't put it down.

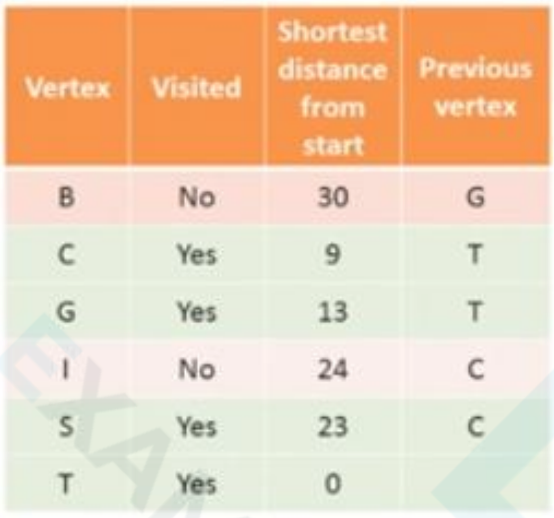
Now we visit the next unvisited vertex with the lowest distance from the start and that is G, same process as described in the paragraph before applies:



Vertex	Visited	Shortest distance from start	Previous vertex
B	No	30	G
C	Yes	9	T
G	Yes	13	T
I	No	24	C
S	No	23	C
T	Yes	0	

The distance between B and the starting point is an infinitive value, and 30 is lower than the infinitive value therefore we put it down, as well as our previous vertex which is G, as that column was empty before. With S, when we add the distance between S and G and G and T (Starting point) we get a value of 23 but there is already a 23 in the in the shortest distance from the start and they are the same, so no need to change the value. Mark G as visited.

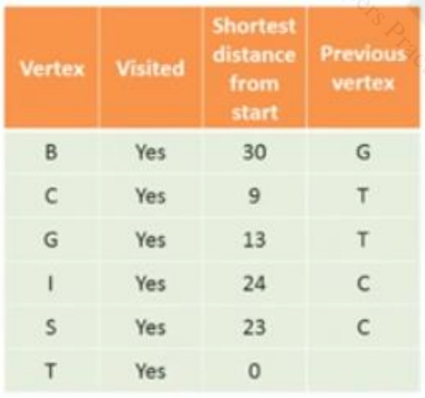
Now again, you visit the next unvisited vertex with the shortest distance from the starting point and that is S, same processes apply:



Vertex	Visited	Shortest distance from start	Previous vertex
B	No	30	G
C	Yes	9	T
G	Yes	13	T
I	No	24	C
S	Yes	23	C
T	Yes	0	

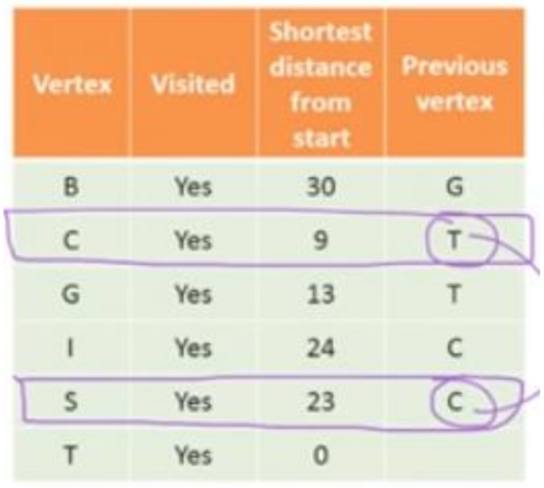
When we get to I, all the vertexes connected to it have already been visited so we just mark I as visited.

So here is your completed table:



Vertex	Visited	Shortest distance from start	Previous vertex
B	Yes	30	G
C	Yes	9	T
G	Yes	13	T
I	Yes	24	C
S	Yes	23	C
T	Yes	0	

And the shortest distance from T to S should be obvious, it goes from S to C and then to T:



Vertex	Visited	Shortest distance from start	Previous vertex
B	Yes	30	G
C	Yes	9	T
G	Yes	13	T
I	Yes	24	C
S	Yes	23	C
T	Yes	0	

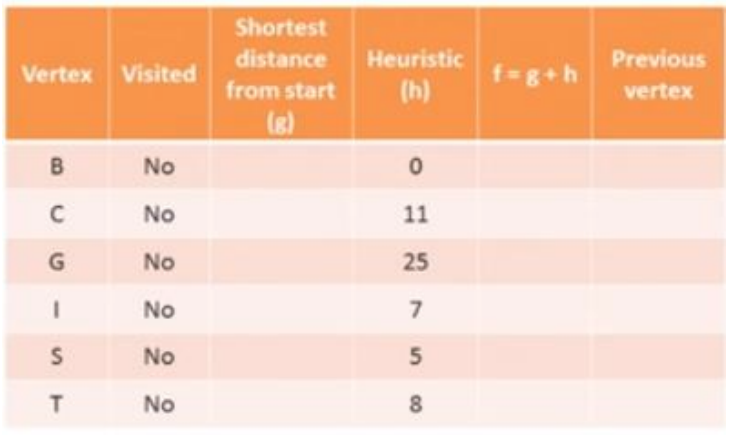
Remember, you only fill in the previous vertex column if it's empty, if it's already full then don't change anything. The only column whose values you should change is the visited column and the shortest distance from the start column.

A key thing to remember is that, when working out the distance between a **visiting vertex and the vertexes connected to them, if any of the connected vertexes have already been visited, then you just ignore them**.

A* Algorithm

The only thing I can think of when reading the title of this concept is me getting that A in the exams, WE WILL HAVE THAT A* BITCHES...*

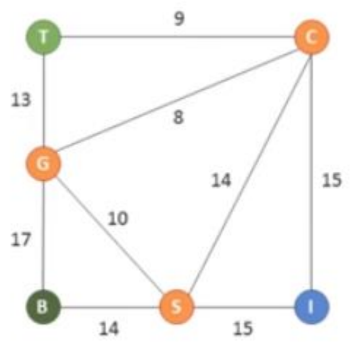
This is simply an extension of Dijkstra's algorithm, it involves this table:

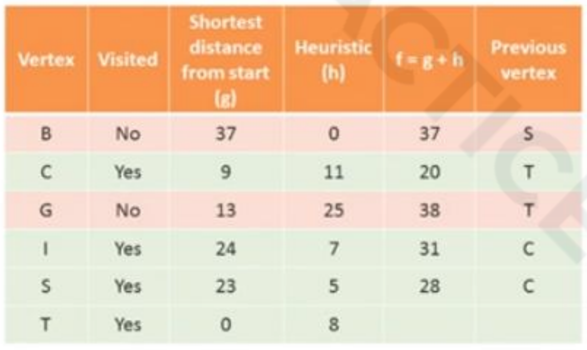


Vertex	Visited	Shortest distance from start (g)	Heuristic (h)	f = g + h	Previous vertex
B	No		0		
C	No		11		
G	No		25		
I	No		7		
S	No		5		
T	No		8		

As you can see, the only difference is that there are two extra columns, h and f = g +h. In a motorway, you will have traffic jams and all that, so sometimes that needs to be taken into account, and the h values represent just that, traffic jams or any other factor which can you know..slow down our short path and instead feels like our short path is just endless.

How it works is pretty simple, like with Dijkstra's shortest path, you work out the distance between the vertexes and the starting point, if they are lower put them down, if not don't put them down. The just add that value with H and that is your answer.





Vertex	Visited	Shortest distance from start (g)	Heuristic (h)	f = g + h	Previous vertex
B	No	37	0	37	S
C	Yes	9	11	20	T
G	No	13	25	38	T
I	Yes	24	7	31	C
S	Yes	23	5	28	C
T	Yes	0	8		

We are trying to get from T to B, so T is our starting point. And from this graph is pretty clear which one is our shortest path, B to S to C and to T

Congratulations, you finished everything in A level Computer Science, if you have not done the coursework you better get started with it NOW as trust me it WILL be a pain in the ass.