L2 Computing - Unit 1 - Computer Science

Relevant LINKS

BACK TO COMPUTING UNITS **[1]**

Handbook home page **[2]**

Overview

Computer Science at Gold Level requires the candidate to understand and describe the basic building blocks of computing such as the use of abstractions to formulate real world models; algorithms for planning program structures; programming languages, both textual and visual; and binary and boolean patterns and statements. As a result of reviewing their work, they will be able to identify and use automated methods or alternative ways of working to improve programming and using computers. Unfamiliar aspects will require support and advice from other people.

A work activity will typically be 'straightforward or routine' because:

The task or context will be familiar and involve few variable aspects. The techniques used will be familiar or commonly undertaken.

Example of context – designing, planing, implementing and testing a basic program for controlling a physical system.

Support for the assessment of this award

Example of typical Computing work at this level (Coming Soon)

Assessor's guide to interpreting the criteria

General Information

QCF general description for Level 2 qualifications

- Achievement at QCF level 2 (EQF Level 3) reflects the ability to select and use relevant knowledge, ideas, skills and procedures to complete well defined tasks and address straightforward problems. It includes taking responsibility for completing tasks and procedures and exercising autonomy and judgement subject to overall direction or guidance.
- Use understanding of facts, procedures and ideas to complete well defined tasks and address straightforward problems. Interpret relevant information and ideas. Be aware of the types of information that are relevant to the area of study or work.
- Standards must be confirmed by a trained Level 2 Assessor or higher
- Assessors must, at a minimum, record assessment judgements as entries in the online mark book on the INGOTs.org certification site.
- Routine evidence of work used for judging assessment outcomes in the candidates' records of their day to day work will be available from their ePortfolios and online work. Assessors should ensure that relevant web pages are available to their Account Manager on request by supply of the URL.
- When the candidate provides evidence of matching all the criteria to the specification subject to the guidance below, the assessor can request the award using the link on the certification site. The Account Manager will request a random sample of evidence from candidates' work that verifies the assessor's judgement.

- When the Account Manager is satisfied that the evidence is sufficient to safely make an award, the candidate's success will be confirmed and the unit certificate will be printable from the web site.
- This unit should take an average level 2 learner 40 guided hours of work to complete.

Assessment Method
Assessors can score each of the criteria N, L, S or H. N indicates no evidence and it is the default setting. L indicates some capability but some help still required to meet the standard. S indicates that the candidate can match the criterion to its required specification in keeping with the overall level descriptor. H indicates performance that goes beyond the expected in at least some aspects. Candidates are required to achieve at least S on all the criteria to achieve the full unit award.

Once the candidate has satisfied all the criteria by demonstrating practical competence in realistic contexts they achieve the unit certificate. Candidates that meet the requirements for all units achieve 30 marks and are then eligible to take the grading examination. Grades from the exam are A*, A, B, C. No grade can be awarded at level 2 without taking the examination with at least 40% of the marks coming from the examination.

Expansion of the assessment criteria

1. The candidate will design, use and evaluate computational abstractions

1.1 I can develop abstractions to make efficient code

Candidates should be able to develop simple abstractions that can make code efficient.

Evidence from assessor observations, content of learner portfolios.

Additional information and guidance

The candidate should be able to produce code that represents a simplified version of a physical object or system. This can be part of a larger program or it can be a smaller pieces of standalone code. The aim should be to produce something that is non-trivial i.e. something beyond drawing a simple shape that is produced once and has no particular purpose. An example would be to produce the code to represent something that is used repeatedly in a program in different circumstances. Key words in programming languages such as PRINT are ready made abstractions because the command PRINT invokes some code that causes a range of different possible printing effects. e.g. PRINTTAB(5) would use the PRINT abstraction together with the tabulation abstraction to start printing in a particular place. Note the word PRINT is causing the activation of a lot of hidden and complex code. Its just used so frequently it is an obvious part of the language to provide “ready made”. In a procedural language, the student abstraction could be a Procedure that is used to do different things if different parameters are included. e.g. ProcedureDrawBox(5) might draw a box with side 5 units and ProcedureDrawBox(10) might draw a box with side 10 units. It is then possible to invoke ProcedureDrawBox whenever we need a box with a different side. We could get the effect of an expanding box by repeating the drawing with gradually increasing length parameter. For this criterion the aim is that the learner appreciates that if a piece of code can be given a name so that it can be used used many times just by calling that name, it is much more efficient than having to keep repeating the code in slightly different forms each time it is needed.

1.2 I can use computational techniques to show how to store patterns more efficiently

Candidates should be able to come up with methods for reducing the amount of data needed to store sequences of data where there is a pattern.

Evidence: from assessor observations, documentation in portfolios.

Additional information and guidance

Candidates should build on work in level 1 to consider how information is stored and how it can be compressed. If patterns repeat they can be compressed. Take the simple case of 5 red pixels in an image. This can be represented as RRRRR or it could be 5R. RRRRR takes 5 characters and 5R takes 2. If we think in terms of bytes, a byte of data can be any number from 0 to 255 so in 2 bytes we could store 1R, 2R..up to 255R or indeed 1B, 2B …..up to 255B (for blue) This is because the first character is just a number so it can be represented by a number and the second, a colour can also be represented by a number. The computer would have to know that the first byte was a number, the second byte was a colour and which colour in order to get the hardware to put that colour on the screen. This ordering and data representation is the file format. Try opening a small .jpg or .png file in a text editor. You will see a lot of characters some slightly weird because there are about half of the characters in the 255 available that are not normally used for text printing.

What would the limits be? First of all 0,1 and 2 colours don't give us any gain. 0R for no red would be a disaster because you would have 2 bytes for every pixel that wasn't a red one! That would make the file much bigger. 1R is worse than just R and 2R is no better than RR. The first byte is limited to 255 because that is the biggest number we can have in one byte. What if there are 256 Rs? We would have to start again with a new counter for each set of 255 Rs. Of course this is still a good saving because for every 255 Rs we use only 2 bytes. So a file that was all Rs could at best be 255/2 times smaller. A 127k file would then be just 1k. Or a 127MB file just 1 MB. For big files transferred over the internet this makes a big difference.

If a file has a lot of colour variation in it, the compression will work a lot less well. If every byte is different from the next one our system will not work well. To test this use a program like Inkscape to draw a plain coloured circle and export the image to a .png file. .png files compress files in a similar but more complicated way to what we have been considering. Now apply a colour gradient to the image to make the colour tone change across the image. In this case there are many more different colours. Now export the .png using a new name but everything else identical (the size and image resolution etc) to make it a fair comparison. Now go to the file and check the size of each image. The one with the gradient will be significantly bigger. .png files still contain all the information to reconstruct the original file. .jpg get even greater compression by throwing away some of the information if it is not going to make the picture look a lot better. This is more complicated but the quality of the image is traded off against the size of the file. Once you jpg a file, you can't get back to the original data. Best always to store an original in e.g. .png if it is likely to be important and take .jpg copies from it. For line diagrams, charts, cartoon style clip art etc, it is best to design using a vector format like .svg and produce .png and .jpgs from it as required.

1.3 I can modify a software abstraction to serve a new purpose

Candidates should appreciate the power of an abstraction in terms of the way relatively small modifications can give it a whole new set of uses. They should have opportunities to demonstrate this in software.

Evidence from assessor observations, content of learner portfolios.

Additional information and guidance

The candidate can take an existing piece of open source code and make modifications to parts of it to make it do something different. This could be making a procedure more general by adding a variable parameter. It could be adding to a structure to extend its purpose or it could be simplifying a structure to make it more general.

1.4 I can describe software abstractions that model real world systems

Candidates should be able to describe some typical software abstractions and say how they relate to real world systems.

Evidence: from assessor observations, content of learner portfolios.

Additional information and guidance

In the case of e.g. Numpty Physics [3]candidates should be able to say that the objects they draw on the screen are abstractions of real objects because they are limited to 2 dimensions whereas real objects are 3 dimensions. In the case of a violent computer game, the abstraction is not real because no real people get killed.

1.5 I can describe strengths and weaknesses in computer models

Candidates should be able to describe the strengths and weaknesses in models that they use.

Evidence: From content of learner portfolios.

Additional information and guidance

This differs from level 1 in that the candidate should be able to describe strengths and weaknesses from a generic starting point and provide some detail beyond identification. e.g. In Numpty Physics, a strength is that you can experiment with how different objects behave in a gravity field in different environments. This can help understand how real objects behave even though the objects are not fully identical to real objects. A weakness is that you can't make it more realistic by providing wind or different materials. Plastic, wood and metal might make some of the situations different.

2. The candidate will understand algorithms

2.1 I can write complex algorithms and conditional loops

Candidates should be able originate algorithms that have some complexity

Evidence: Assessor observations, local testing, portfolios.

Additional information and guidance

Algorithms are step-by-step problem-solving procedures. By complex we mean multiple components and some variation in the structure. This could be as an example, from a nested loop. e.g. generating a set of times tables with something like:

```
FOR mtable = 1 TO 10
 FOR number = 1 TO 10
    IF NOT(mtable = 5) THEN
        PRINT mtable “x” number “=” mtable*number
  ENDIF
 NEXT number
NEXT mtable
```

This will then print the multiplication tables minus the 5 times table.

Candidates should be encouraged not to make things more complex than necessary. It is best to break a program down into many small easy to understand parts and link these together. To further this process using a combination of their own and other people's code is perfectly legitimate and to be encouraged. Just make sure there is sufficient evidence of the candidate being able to meet the criterion with their own work.

2.2 I can describe different algorithms that target the same task

Candidates should be able to describe at least two different methods of approach to a problem that lends itself to an algorithm

Evidence: Assessor observations, portfolios.

Additional information and guidance

A useful resource to underpin this learning outcome is at CS Uplugged [4].

A bubble sort and a merge sort are both sort algorithms but they can result in different efficiencies in achieving the outcome. A linear search and a binary search are again different methods targeted on the same problem. At level 2 candidates would be able to outline some fundamental reasons why although two methods are achieving the same result one might be a better choice.

2.3 I can compare algorithms on the basis of efficiency

Candidates should be able to compare algorithms based on how efficient they are in solving a particular problem.

Evidence: From local testing and portfolios

Additional information and guidance

Candidates should be able to see that some algorithms can achieve a task more quickly than others or with less lines of code. They should always be looking to reduce the amount of code they need to achieve a task. (Code as opposed to documentation. Note that it is a common mistake to assume documentation and comments slow programs down. There will be no discernible difference) They should appreciate that in some cases execution of a program with more code could be faster than one with less. By level 2 candidates should be thinking about how many iterations of an algorithm are needed and how altering an algorithm could make it operate more efficiently [5].

2.4 I can explain the relationship between instructions and data in an algorithm

Candidates should be able to recognise the inputs and outputs of algorithms as data and the instructions as operators on the data.

Evidence: From assessor observations and portfolios

Additional information and guidance

Generally algorithms take input data and do some operations on it and then provide some output data. The input data might have to be qualified first through a “precondition”. Without qualified data the algorithm could fail to give a valid result so data is often validated before it can be input into the algorithm. For example, an algorithm to find square roots of numbers might check to see if all the input numbers are positive since finding the square root of a negative number is not possible (at least not with real numbers). Instructions need data on which to
operate. An instruction such as ADD is not much use without knowing what it is to ADD and where it must put the result. ADD N1,N2,R is more useful if it adds two numbers in N1 and N2 and puts the result in R. Algorithms of just about any complexity can be built up from simple instructions and data like this. Indeed this is how modern digital computers work. They do billions of simple operations like this in a second and a billion simple operations can be combined and configured into very complex algorithms. Fortunately, most of the common simple instructions are defined for us and even much more commonly used complex algorithms are provided by key words in programming languages. The HTML tag is effectively an instruction to make all the text after it appear in a bold type face. That will be achieved by different methods in different browsers on different computers with many steps between the tag and what actually appears on the screen. We don’t have to code all these steps each time just putting in the tag is enough.

2.5 I can explain the words iteration and recursion

Candidates should be able to explain the words and recognise examples of them in algorithms

Evidence: Assessor observations and portfolios.

Additional information and guidance

Iteration means the act of repeating a process usually to converge to a result. Recursion is a process in which a function calls itself as a subroutine. Recursion is really a special case of iteration. At this level it is sufficient to be able to explain iteration in terms of loops.

Iterations happen with loops. Each time a loop is executed an iteration takes place. A program can specify a fixed number of iterations e.g. FOR iteration=1 TO Page 97100:PRINT iteration:NEXTiteration will execute 100 iterations. Alternatively a condition might be set to end the iteration. Recursion allows a function or procedure to be repeated several times, since it calls itself during its execution. Recursion is often seen as an efficient method of programming since it requires the least amount of code to perform the necessary functions. However, recursion must be incorporated carefully, since it can lead to an infinite loop if no condition is met that will terminate the function.

3. The candidate will use programming languages

3.1 I can modify an existing program to extend the scope of its use

The candidate should be able to modify an existing useful program to extend its scope of use.

Evidence: Portfolios

Additional information and guidance

Candidates should take an existing application that is open source and modify it to produce something different. This could be an extension of the program, an improvement or a change in its function. As an example there are Javascript games here [6]. Take the game Pairs 1 and increase the number of pairs to match and make the subject of matching different. This is just an example. There is flexibility to to find and change anything that the candidate has as an interest.

3.2 I can distinguish between a mark-up language and a programming language

The candidate should able to use an example of each to illustrate the difference

Evidence: From Portfolios.

Additional information and guidance

Candidates should be able to give examples such as HTML as a mark-up language and Javascript as a programming language. HTML uses tags to display information in different ways. It does not have loops, structures and logical constructs that a programming language has. Mark-up languages can be used for file formats e.g. XML as the basis of OpenDocument and docX. On the Web, Javascript is used on the client side when programming is needed to make web pages interactive. Server side programming languages such as PHP can provide database processing that can then be displayed in the web page using HTML tags.

3.3 I can originate code to solve a problem

The candidate should be able to solve a non-trivial problem using their own code.

Evidence: From portfolios

Additional information and guidance

Candidates should be given opportunities to solve simple problems in the subjects of the curriculum at a level they have mastered using code. They do not have to originate all the code themselves but at least 50 lines of the code should be their own. Examples might be to program a tune to play

within an application, to program a control system to simulate automated machinery, a program in science to simulate simple circuits, a program in geography to test knowledge of sustainable development.

3.4 I can test code using systematic methods

The candidate should be able to use systematic methods including control of variables and breaking the code into smaller sections.

Evidence: From assessor observations and documentation in portfolios

Additional information and guidance

Showing practical competence in systematic debugging in keeping with the overall level 2 descriptor is the aim. Candidates at level 2 will need some pointers but should be showing consistent techniques to fault finding with a degree of self sufficiency.

3.5 I can explain the difference between source code and executable code

The candidate should be able to recognise source code and executable code from the contents of files and explain the difference.

Evidence: From portfolios

Additional information and guidance

Candidates should know that source code is designed to be easily understandable by humans and executable code is understandable by machines. This can be related back to abstraction. A hardware abstraction layer is software that provides a means for applications to access hardware resources. The source code of an application is an abstraction further from the hardware to make things more humanly friendly. Source code is what humans produce in programming languages.

Languages like Python are designed to make them more like the language humans are used to. At the other end of the scale Assembly languages are much closer to what the machine understands. To get from source code to executable code requires translation. This can be done through an interpreter which is a program that converts the source code to executable code as the program is run. The other more common method with large modern applications is to use a compiler that converts all the source code to executable code so that it is ready to run. The disadvantage of compiling is that it takes time to compile large amounts of source code e.g. Apache OpenOffice takes several hours to build. If you need to compile the code you can’t easily just change a few lines of code and immediately see the effect. The advantage is that the executable code will be a lot more compact and most people don’t need the source code so it makes distribution easier and for owners of the source code prevents other people from seeing how the program works. Open Source software makes the source code freely available so that other people can see the source code and help improve it.

4. The candidate will understand Boolean Logic, binary and hexadecimal numbers

4.1 I can show how NOT AND and OR gates can be made from NAND gates only

The candidate should be able to show how combinations of NAND Gates can produce the other three fundamental gate types.

Evidence: From internal testing and portfolios

Additional information and guidance

Diagrams and explanations are freely available from here [7]and here [8].

Some practical electronics projects from here [9].

Candidates should be familiar with simple truth tables and the symbols for NOT, AND, OR and NAND gates. Also know the name Inverter for a NOT gate. The importance of NAND gates is that they can be used to make any other gate so in practice only one type of logic gate is needed in electronic engineering.

4.2 I can add and subtract binary numbers

Candidates should be able to perform simple arithmetic operations on binary numbers

Evidence: From internal testing and portfolios.

Additional information and guidance

Binary [10].

The above link shows how to do this and provides exercises. Note that at a fundamental level in a computer all the operations are with binary numbers, these are just abstracted to higher levels so that humans can directly recognise them. e.g. the binary number 1000001 = 65 in our more familiar decimal numbers. The ASCII code 65 is the letter A in the text you see on the screen, so somewhere in your computer there is the pattern 1000001 in any computer displaying the letter A. B is 66 so 1000010 Imagine trying to read this page if all the codes were converted to binary. This is why we translate from binary codes to decimal numbers and characters. The computer does all its calculations in binary, specialised chips and software convert from binary to our more familiar number systems and mathematics based on this so we don’t need to learn to think in binary just to understand the principles.

4.3 I can relate 4 bit binary to hexadecimal numbers

The candidate should be able to recognise the relationship between 4 bit numbers and hexadecimal.

Evidence: Internal testing, portfolios.

Additional information and guidance

In some ways it is unfortunate that we have 10 fingers because that is why we count in lots of ten. It is so ingrained into us from being little children anything else seems alien. Computers lend themselves much better to counting in 2s, 4s, 8s. 16s etc because these relate better to binary numbers. A lot of computer numbers are based on 4 and 8 bits. (A bit is a binary digit) in binary 0=0 10 = 2 100 =4 1000=8 and 10000=16. So 15=1111. its all based on the number of possible unique patterns possible from the number of bits. Using 4 bits we can have all the numbers from 0 to 15 - 16 patterns including 0000 through to 1111.. To store these in a single place value they are 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. The letters are used because there is no single character to represent 10, 11, 12, 13, 14 and 15 because single place value in decimal numbers only requires 0-9. This is really just the same as if we had 16 fingers, we would count them as individual units up to 15 and for 16 we would say 10 but mean sixteen not ten. In computers you will see numbers like &FF, or &10 or #FF or %FF. The &,# and % all mean it is a hexadecimal number following, not a decimal number. If there are letters in it it is easy but of course some hex numbers look like decimal numbers. This is actually true of binary too. 10 in binary is two not ten. Why bother with Hex numbers? It simply makes it more compact to store numbers and relates better to the number systems best suited to machines. &FF is the same as decimal 255. &FFFF is 65535 or 64K. Notice that if we have a file size of 64K it is actually a bit bigger than 64 thousand bytes in decimal because of the different number systems.

&F is 1111 ie the maximum 4 bit number
&FF is 11111111 ie the maximum 8 bit number

&FFFF is 1111111111111111 - 16 bits
&FFFFFF is 111111111111111111111111 - 24 bits as in colour 8 bits for each of red, green, blue (In HTML you will see
&FFFFFFFF is 11111111111111111111111111111111 - 32 bits

If you think of those 1s as wires, how many possible locations could be connected to 8 wires - answer &FF in hexadecimal or 255 in decimal. What about 32? &FFFFFFFF or 4,294,967,295 so about 4.3 billion. (There is a converter here [11])

This is why a 32 bit computer is much more than just 4 times the power of an 8 bit computer. Now we have 64 bit computers. The maximum decimal number with 64 bits is 9,223,372,036,854,775,807. Very big!
1111111111111111111111111111111111111111111111111111111111111111 If we can make instructions in a computer with 8 bits we can process several at once in 64 bits and directly address many more memory locations.

The main aim here is to give an insight of why number systems are important and how they might relate to the physical electronics in a computer. We are not expecting average Level 2 candidates to fully grasp everything or remember all the details. If some can so much the better. They should recognise hex numbers by the &,#,% or Hex. know that 1111 in binary is 15 and F in hexadecimal, that 10 could represent many different numbers in different number systems.

4.4 I can relate binary numbers to the voltage state of a connector

Candidates should be able to relate states of wires, LEDs etc to bit patterns.

Evidence: From assessor observation and portfolios

Additional information and guidance

Digital electronics works by settings that are high or low, on or off. High or on represents a 1 and low or off a zero. Back to abstraction again. 1 is an abstraction of a wire raised to a potential of 5 volts. 0 is an abstraction of a wire at 0 volts. In fact as long as there is a measurable difference between two states they can represent 1 and 0. The smaller we can make the physical representation of 1 and 0 the more data we can pack into a given space. We could make a computer with light bulbs as memory locations but it would be massive and consume a lot of power. On the other hand humans have to be able to communicate with computers so we have to be able to connect big things like keyboards to very little things like memory locations. If I push the key with a letter A on it, it will connect to a physical pattern 1000001 etched microscopically in silicon via a series of steps and that pattern will also be sent back through a series of steps to the screen. It happens so quickly it seems instantaneous.

Practical applications can be explores e.g. using the RaspberryPI [12]

There are many simple applications of bit patterns being used to control things. Take a 7 segment display. It is based on representing numbers and letters by 7 lines. If all lines are present we get a number 8

How can you take away 2 lines to make a 3?

How many bits does it take to control the display? Answer 7 because there are 7 lines that can be switched on or off. What number corresponds to 1111111 ? Answer 127 or &F7. It depends on which bit controls which line as to what specific bit pattern displays which number or letter. All we know for

certain is 1111111 which include all of them so sending the number 127 to the display as a binary pattern produces what looks like a number 8 on the 7 segment display.

4.5 I can explain analogue to digital conversion

Candidates should be able to explain the basic principles of sampling analogue signals to produce digital data and reconstructing an analogue signal from digital data.

Evidence: From portfolios

Additional information and guidance

A good concrete example is in music. Sound we hear is analogue. It is a continuous series of pressure changes in the air caused by a mechanical oscillation. Vocal chords, guitar strings or vibrating reeds all cause continuous sound waves. when the sound wave wobbles tiny bones in our ears they send the sound signal to our brains and we hear things. A microphone picks up sound and converts it to an electrical signal. So how can we record this signal and store the recording in a computer? Computers can only store 1s and 0s so we need a way of converting the electrical signal produced by the sound wave into patterns of 1s and 0s. This is where an analogue to digital converter is used. This is a chip that can sample a varying voltage very rapidly. More details here [13].

The key thing with audio is that the samples have to represent the level of the voltage (amplitude) - effectively the loudness of the sound - and the frequency that the voltage changes to get the pitch of the note or frequency. A simpler example is to make a digital thermometer. If you have a device where it's electrical properties change with temperature we can use it to measure temperature. As the temperature rises and falls the voltage across the device will change. This is again a continuous analogue signal. Sampling the voltage with an analogue to digital converter (ADC) will provide numbers corresponding to the voltage and hence the temperature. Let's say we put the device in pure melting ice at normal pressure. The number generated from the ADC is then 00C put it in steam at normal pressure and that number is 1000C. Now divide into 100 parts and we have a digital thermometer. If the ADC uses 8 bits what is the smallest temperature the thermometer can measure. If 00C happens to coincide with the lowest possible sample for the ADC and 1000C the highest, there are 255 possible readings in 8 bits so 2.55 for each degree. It is very unlikely that zero and a hundred will be at the two extreme measurements but certainly an 8 bit ADC should be able to be made to 1 degree of precision. A 16 bit ADC would divide the scale into 64,000 parts and so it would probably be far more precise than needed. With audio a 16 bit ADC will sample sound so well that there will be no real discernible gain by going to higher values. Sample frequency (how fast the sound wave is sampled) is also important though. With a thermometer, if the reading is up-dated every second it is probably good enough, with a sound wave the oscillations causing the sound are at tens of thousands per second. If the sample rate is not fast enough a lot of information will simply be missed. In general the sample rate should be at least double the highest frequency. For standard CDs this was set at 44KHz because humans don't hear much above 20KHz. On modern electronics this is easy to achieve so it is possible to go to much better rates. Whether or not the ear and brain can tell the difference is moot. Once a sound is converted to numbers is is easy to do mathematical operations on the numbers and produce a lot of effects. Echo, noise removal, pitch shifting can all be done using software. Its so fast we can even deaden sound by sampling an incoming sound wave working out its exact opposite wave form and then playing it back before the original sound wave (vibrating 10s of thousands of times a second and moving at the speed of sound) has have time to change its form or position appreciably. This is the principle of expensive headphones that remove all external noise.

Digital to analogue conversion is the reverse process, creating a sound wave in a speaker from a set of numbers stored in the computer. Students should be familiar with the basic concept of ADC/DAC in terms of sampling analogue signals to produce sets of numbers. Getting hands on experience e.g. via RaspberryPI or similar is desirable.

Moderation/verification

The assessor should keep a record of assessment judgements made for each candidate and make

notes of any significant issues for any candidate. They must be prepared to enter into dialogue with their Account Manager and provide their assessment records to the Account Manager through the online mark book. They should be prepared to provide evidence as a basis for their judgements through reference to candidate e-portfolios and through signed witness statements associated with the criteria matching marks in the on-line mark book. Before authorizing certification, the Account Manager must be satisfied that the assessors judgements are sound.

Links
[1] http://theingots.org/community/Computing_qualification_info_units
[2] https://theingots.org/community/sites/default/files/uploads/site_icons/handbook-computing-L1-L2.jpg
[3] http://numptyphysics.garage.maemo.org/
[4] http://csunplugged.org/searching-algorithms/
[5] https://fiftyexamples.readthedocs.org/en/latest/algorithms.html
[6] http://ingotgames.org/
[7] http://www.instructables.com/id/Use-NAND-gate-to-make-NOT-gate/
[8] http://www.luishernandezengineeringportfoli.weebly.com/nandnor-gates.html
[9] http://www.dummies.com/how-to/content/electronics-projects-how-to-use-nand- gates-to-crea.html
[10] http://www.cimt.plymouth.ac.uk/projects/mepres/book9/bk9i1/bk9_1i2.html
[11] http://myhandbook.info/table_hex.html?original=1&finaltext=FFFF%0D%0A
[12] http://http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/robot/morse_code/
[13] http://en.wikipedia.org/wiki/Analog-to-digital _converter