|  | **COMPUTER SCIENCE** | **102 & 103** | **103 & 104** | **Year 10** | **Year 11** |
| --- | --- | --- | --- | --- | --- |
| **1** | 3.1.1 Representing algorithms | Understand and explain the term algorithm. An algorithm is a sequence of steps that can be followed to complete a task. Be aware that a computer program is an implementation of an algorithm and that an algorithm is not a computer program.  Understand and explain the term  decomposition. Understand and explain the term decomposition. Decomposition means breaking a problem into a number of sub-problems, so that each subproblem  accomplishes an identifiable task,  which might itself be further subdivided.  Understand and explain the term abstraction. Abstraction is the process of removing unnecessary detail from a problem.  Explain simple algorithms in terms of their inputs, processing and outputs. Students must be able to identify where inputs, processing and outputs are taking place within an algorithm. | Understand that more than one algorithm can  be used to solve the same problem. Compare the efficiency of algorithms explaining how some algorithms are more efficient than others in solving the same problem.  Determine the purpose of simple algorithms Students should be able to use trace tables and visual inspection to determine how simple algorithms work and what their purpose is. | **102 & 103** | **103 & 104** |
| **2** | 3.1.3 Searching algorithms | Understand and explain how the linear search algorithm works. Students should know the mechanics of the algorithm.  Understand and explain how the binary search algorithm works. Students should know the mechanics of the algorithm.  Compare and contrast linear and binary search algorithms. Students should know the advantages and disadvantages of both algorithms. | Understand and explain how the merge sort algorithm works. Students should know the mechanics of the algorithm.  Understand and explain how the bubble sort algorithm works. Students should know the mechanics of the  algorithm.  Compare and contrast merge sort and bubble sort algorithms. Students should know the advantages and  disadvantages of both algorithms. | **102 & 103** | **103 & 104** |
| **3** | 3.2.1 Data types | Students need a theoretical understanding of all the topics in this section for the exams even if the programming language(s) they have been taught do not support all of the topics.  Understand the concept of a data type.  Understand and use the following appropriately:  • integer  • real  • Boolean  • character  • string.  Use, understand and know how the following statement types can be combined in programs:  • variable declaration  • constant declaration  • assignment  • iteration  • selection  • subroutine (procedure/function). The three combining principles (sequence,  iteration/repetition and selection/choice) are basic to all imperative programming languages.  Students should be able to write programs  using these statement types. They should be able to interpret algorithms that include these statement types.  Students should know why named constants and variables are used. | Use nested selection and nested iteration  structures.  Use meaningful identifier names and know why it is important to use them. Identifier names include names for variables, constants and subroutine names.  Use definite and indefinite iteration, including indefinite iteration with the constants at the  start or the end of the iterative structure. A theoretical understanding of constant(s) at  either end of an iterative structure is required,  regardless of whether they are supported by the language(s) being used. | **103 & 104** | **102 & 103** |
| **4** | 3.2.3 Arithmetic operations in a programming language | Be familiar with and be able to use:  • addition  • subtraction  • multiplication  • real division  • integer division, including remainders  Integer division, including remainders is usually a two stage process and uses modular arithmetic:  eg the calculation 11/2 would generate the  following values:  Integer division: the integer quotient of 11 divided by 2 (11 DIV 2) = 5  Remainder: the remainder when 11 is divided by 2 (11 MOD 2) = 1 |  | **103 & 104** | **102 & 103** |
| **5** | 3.2.4 Relational operations in a programming language | Be familiar with and be able to use:  • equal to  • not equal to  • less than  • greater than  • less than or equal to  • greater than or equal to.  Students should be able to use these operators within their own programs and be able to interpret them when used within algorithms. Note that different languages may use different symbols to represent these operators. In assessment material we will use the following  symbols: =, ≠, <, >, ≤, ≥ |  | **103 & 104** | **102 & 103** |
| **6** | 3.2.5 Boolean operations in a programming language | Be familiar with and be able to use:  • NOT  • AND  • OR.  Students should be able to use these operators, and combinations of these operators, within for iterative and selection structures |  |  | **102 & 103** |
| **7** | 3.2.6 Data structures | Understand the concept of data structures. | Use arrays (or equivalent) in the design of  solutions to simple problems. Use records (or equivalent) in the design of solutions to simple problems |  | **103 & 104** |
| **8** | 3.2.7 Input/output and file handling | Be able to obtain user input from the keyboard Be able to output data and information from a program to the computer display. Be able to read/write from/to a text file. |  |  | **103 & 104** |
|  | 3.2.8 String handling operations in a programming language | Understand and be able to use:  • length  • position  • substring  • concatenation  • convert character to character code  • convert character code to character  • string conversion operations. | Expected string conversion operations:  • string to integer  • string to real  • integer to string  • real to string. |  | **103 & 104** |
|  | 3.2.9 Random number generation in a programming language | Be able to use random number generation. Students will be expected to use random  number generation within their computer  programs. An understanding of how pseudorandom numbers are generated is not required. |  |  | **102 & 103** |
|  | 3.2.10 Subroutines (procedures and functions) | Understand the concept of subroutines.  Explain the advantages of using subroutines in programs.  Describe the use of parameters to pass data within programs. Students should be able to use subroutines that require more than one parameter. Students should be able to describe how data is passed to a subroutine using parameters. Use subroutines that return values to the calling routine. Students should be able to describe how data is passed out of a subroutine using return values  Know that subroutines may declare their own variables, called local variables, and that local variables usually:  • only exist while the subroutine is  executing  • are only accessible within the subroutine | Know that a subroutine is a named ‘out of line’  block of code that may be executed (called) by  simply writing its name in a program statement  Use local variables and explain why it is good  practice to do so. |  | **103 & 104** |
|  | 3.2.11 Structured programming | Describe the structured approach to  programming. Students should be able to describe the structured approach including modularised programming, clear, well documented interfaces (local variables, parameters) and return values.  Teachers should be aware that the terms  'arguments' and 'parameters' are sometimes used but in examinable material we will use the term 'parameter' to refer to both of these. | Explain the advantages of the structured  approach. |  | **102 & 103** |
|  | 3.2.12 Robust and secure programming | Be able to write simple data validation routines. Students should be able to use data validation techniques to write simple routines that check the validity of data being entered by a user.  The following validation checks are examples of simple data validation routines:  • checking if an entered string has a  minimum length  • checking if a string is empty  • checking if data entered lies within a  given range (eg between 1 and 10).  Be able to write simple authentication routines.  Students should be able to write a simple  authentication routine that uses a username and password. Students will only be required to use plain text usernames and passwords (ie students will not need to encrypt the passwords). | Be able to select suitable test data that covers  normal (typical), boundary (extreme) and  erroneous data. Be able to justify the choice of test data. |  | **103 & 104** |
|  | 3.2.13 Classification of programming languages | Know that there are different levels of  programming language:  • low-level language  • high-level language.  Explain the main differences between low-level and high-level languages. Students should understand that most computer programs are written in high-level  languages and be able to explain why this is the case.  Understand that ultimately all programming code written in high-level or assembly languages must be translated into machine code.  Understand that machine code is expressed in binary and is specific to a processor or family of processors.  Understand that there are three common types  of program translator:  • interpreter  • compiler  • assembler.  Explain the main differences between these three types of translator.  Understand when it would be appropriate to use each type of translator. | Know that machine code and assembly language are considered to be low-level languages and explain the differences between them.  Understand that processors execute machine code and that each type of processor has its own specific machine code instruction set.  Understand that assembly language is often used to develop software for embedded systems and for controlling specific hardware components. Understand that assembly language has a 1:1 correspondence with machine code.  Understand the advantages and disadvantages of low-level language programming compared with high-level language programming. |  | **102 & 103** |
|  | 3.3 Fundamentals of data representation | Understand the following number bases:  • decimal (base 10)  • binary (base 2)  • hexadecimal (base 16).  Understand that computers use binary to  represent all data and instructions. | Students should be familiar with the idea that a  bit pattern could represent different types of  data including text, image, sound and integer  Explain why hexadecimal is often used in  computer science. |  | **102 & 103** |
|  | 3.3.2 Converting between number bases | Understand how binary can be used to  represent whole numbers.  Understand how hexadecimal can be used to  represent whole numbers. | Be able to convert in both directions between:  • binary and decimal  • binary and hexadecimal  • decimal and hexadecimal.  The following equivalent maximum values will  be used:  • decimal: 255  • binary: 1111 1111  • hexadecimal: FF |  | **103 & 104** |
|  | 3.3.3 Units of information | Know that:  • a bit is the fundamental unit of information  • a byte is a group of 8 bits. | Students might benefit from knowing that  historically the terms kilobyte, megabyte, etc  have often been used to represent powers of 2.  The SI units of kilo, mega and so forth refer to  values based on powers of 10. When referring  to powers of 2 the terms kibi, mebi and so forth  would normally be used but students do not  need to know these. |  | **103 & 104** |
|  |  | Know that quantities of bytes can be described using prefixes. Know the names, symbols and corresponding  values for the decimal prefixes:  • kilo, 1 kB is 1,000 bytes  • mega, 1 MB is 1,000 kilobytes  • giga, 1 GB is 1,000 Megabytes  • tera, 1 TB is 1,000 Gigabytes. |  |  | **102 & 103** |
|  | 3.3.4 Binary arithmetic | Be able to add together up to three binary numbers. Students will need to be able to add together up to three binary numbers using a maximum of 8 bits per number.  Students will only be expected to add together a maximum of three 1s in a single column. Answers will be a maximum of 8 bits in length and will not involve carrying beyond the eight bits. | Be able to apply a binary shift to a binary number. Students will be expected to use a maximum of  8 bits.  Students will be expected to understand and  use only a logical binary shift.  Students will not need to understand or use  fractional representations. |  | **102 & 103** |
|  |  | Binary shifts can be used to perform simple multiplication/division by powers of 2. | Describe situations where binary shifts can be  used. |  | **103 & 104** |
|  | 3.3.5 Character encoding | Understand what a character set is and be able to describe the following character encoding methods:  • 7-bit ASCII  • Unicode. | Students should be able to use a given  character encoding table to:  • convert characters to character codes  • convert character codes to characters. |  | **102 & 103** |
|  |  | Understand that character codes are commonly grouped and run in sequence within encoding tables. | Students should know that character codes are  grouped and that they run in sequence. For  example in ASCII ‘A’ is coded as 65, ‘B’ as 66,  and so on, meaning that the codes for the other  capital letters can be calculated once the code  for ‘A’ is known. This pattern also applies to  other groupings such as lower case letters and  digits. |  | **103 & 104** |
|  |  | Describe the purpose of Unicode and the  advantages of Unicode over ASCII.  Know that Unicode uses the same codes as  ASCII up to 127. | Students should be able to explain the need for  data representation of different alphabets and  of special symbols allowing a far greater range  of characters.  It is not necessary to be familiar with UTF-8,  UTF-16 or other different versions of Unicode. |  | **103 & 104** |
|  | 3.3.6 Representing images | Understand what a pixel is and be able to  describe how pixels relate to an image and the way images are displayed. Students should know that the term pixel is short for Picture Element. A pixel is a single point in a graphical image.  VDUs display pictures by dividing the display screen into thousands (or millions) of pixels, arranged into rows and columns. | Calculate bitmap image file sizes based on the  number of pixels and colour depth. Convert binary data into a black and white image. Convert a black and white image into binary data. |  | **102 & 103** |
|  |  | Describe the following for bitmaps:  • size in pixels  • colour depth.  Know that the size of a bitmap image in pixels (width x height) is known as the image resolution. |  |  | **102 & 103** |
|  |  | Describe how a bitmap represents an image  using pixels and colour depth. Describe using examples how the number of pixels and colour depth can affect the file size  of a bitmap image. |  |  | **103 & 104** |
|  | 3.3.7 Representing sound | Understand that sound is analogue and that it must be converted to a digital form for storage and processing in a computer. | Understand that sound waves are sampled to create the digital version of sound. |  | **102 & 103** |
|  |  | Describe the digital representation of sound in  terms of:  • sampling rate  • sample resolution. | Describe the digital representation of sound in  terms of:  • sampling rate  • sample resolution. |  | **103 & 104** |
|  | 3.3.8 Data compression | Be able to calculate the number of bits required to store a piece of data compressed using Huffman coding.  Be able to calculate the number of bits required to store a piece of uncompressed data in ASCII. | Explain what data compression is. Understand why data may be compressed and that there are different ways to compress data. |  | **102 & 103** |
|  |  | Represent data in RLE frequency/data pairs. Students could be given a bitmap  representation and they would be expected to show the frequency and value pairs for each row,  eg 0000011100000011  would become 5 0 3 1 6 0 2 1. | Explain how data can be compressed using Huffman coding.  Be able to interpret/create Huffman trees. Explain how data can be compressed using run length encoding (RLE). |  | **103 & 104** |
|  | 3.4.1 Hardware and software | Define the terms hardware and software and understand the relationship between them. |  |  | **102 & 103** |
|  | 3.4.2 Boolean logic | Construct truth tables for the following logic  gates:  • NOT  • AND  • OR. | Construct truth tables for simple logic circuits.  Interpret the results of simple truth tables. Create, modify and interpret simple logic circuit diagrams. |  | **102 & 103** |
|  | 3.4.3 Software classification | Explain what is meant by:  • system software  • application software.  Give examples of both types of software |  |  | **102 & 103** |
|  |  | Understand the need for, and functions of, operating systems (OS) and utility programs. | Understand that the OS handles management  of the:  • processor(s)  • memory  • I/O devices  • applications  • security. |  | **102 & 103** |
|  | 3.4.4 Systems architecture | Explain the Von Neumann architecture. | Explain the role and operation of main memory  and the following major components of a central  processing unit (CPU):  • arithmetic logic unit  • control unit  • clock  • bus. |  | **103 & 104** |
|  |  | Understand and explain the Fetch-Execute  cycle. The CPU continuously reads instructions stored in main memory and executes them as required:  • fetch: the next instruction is fetched to the CPU from main memory  • decode: the instruction is decoded to work out what it is  • execute: the instruction is executed  (carried out). This may include reading/  writing from/to main memory. | Explain the effect of the following on the  performance of the CPU:  • clock speed  • number of processor cores  • cache size  • cache type. |  | **103 & 104** |
|  |  | Understand the differences between main  memory and secondary storage.  Understand the differences between RAM and ROM. | Understand why secondary storage is required. Be aware of different types of secondary storage (solid state, optical and magnetic). Explain the operation of solid state, optical and magnetic storage. Discuss the advantages and disadvantages of solid state, optical and magnetic storage. |  | **103 & 104** |
|  |  | Explain the term 'cloud storage'. Explain the advantages and disadvantages of  cloud storage when compared to local storage. | Understand the term 'embedded system' and explain how an embedded system differs from a non-embedded system. |  | **103 & 104** |
|  | 3.5 Fundamentals of computer networks | Define what a computer network is.  Discuss the benefits and risks of computer networks. | Explain the following common network  topologies:  • star  • bus. |  | **102 & 103** |
|  |  | Describe the main types of computer network including:  • Personal Area Network (PAN)  • Local Area Network (LAN)  • Wide Area Network (WAN). | Define the term‘network protocol’. |  | **102 & 103** |
|  |  | Understand that networks can be wired or  wireless.  Discuss the benefits and risks of wireless networks as opposed to wired networks. | Explain the purpose and use of common  network protocols including:  • Ethernet  • Wi-Fi  • TCP (Transmission Control Protocol)  • UDP (User Datagram Protocol)  • IP (Internet Protocol)  • HTTP (Hypertext Transfer Protocol)  • HTTPS (Hypertext Transfer Protocol  Secure)  • FTP (File Transfer Protocol)  • email protocols:  • SMTP (Simple Mail Transfer Protocol)  • IMAP (Internet Message Access  Protocol). |  | **102 & 103**  **103 & 104** |
|  |  | Understand the need for, and importance of, network security. | Explain the following methods of network  security:  • authentication  • encryption  • firewall  • MAC address filtering. |  | **102 & 103** |
|  |  | Describe the 4 layer TCP/IP model:  • application layer  • transport layer  • internet layer  • link layer. | Understand that the HTTP, HTTPS, SMTP, IMAP and FTP protocols operate at the application layer. Understand that the TCP and UDP protocols operate at the transport layer. Understand that the IP protocol operates at the internet layer. |  | **103 & 104** |
|  | 3.6 Fundamentals of cyber security | Be able to define the term cyber security and be able to describe the main purposes of cyber security. | Understand and be able to explain the following  cyber security threats:  • social engineering techniques  • malicious code  • weak and default passwords  • misconfigured access rights  • removable media  • unpatched and/or outdated software. |  | **102 & 103** |
|  |  | Explain what penetration testing is and what it is used for. |  |  | **102 & 103** |
|  | 3.6.1.1 Social engineering | Define the term social engineering.  Describe what social engineering is and how it can be protected against.  Explain the following forms of social  engineering:  • blagging (pretexting)  • phishing  • pharming  • shouldering (or shoulder surfing). |  |  | **102 & 103** |
|  | 3.6.1.2 Malicious code | Define the term 'malware'. Describe what malware is and how it can be protected against.  Describe the following forms of malware:  • computer virus  • trojan  • spyware  • adware. |  |  | **102 & 103** |
|  | 3.6.2 Methods to detect and prevent cyber security threats | Understand and be able to explain the following security measures:  • biometric measures (particularly for  mobile devices)  • password systems  • CAPTCHA (or similar)  • using email confirmations to confirm a  user’s identity  • automatic software updates. |  |  | **103 & 104** |
|  | 3.7 Ethical, legal and environmental impacts of digital  technology on wider society, including issues of privacy | Explain the current ethical, legal and  environmental impacts and risks of digital technology on society. Where data privacy issues arise these should be considered. |  |  | **103 & 104** |

|  |  |  |
| --- | --- | --- |
|  |  |  |