Content

Topic 1: Problem solving

Students are expected to develop a set of computational thinking skills that enable them to understand how computer systems work, and design, implement and analyse algorithms for solving problems.

Students should be given repeated opportunities to tackle computational problems of various sorts, including some substantial problem-solving tasks.

Students will be expected to use the pseudo-code listed in *Appendix 1* and the flowchart symbols shown in *Appendix 2*.

Subject content	Students should:	
1.1 Algorithms	1.1.1	understand what an algorithm is, what algorithms are used for and be able to interpret algorithms (flowcharts, pseudo-code, written descriptions, program code)
	1.1.2	understand how to create an algorithm to solve a particular problem, making use of programming constructs (sequence, selection, iteration) and using appropriate conventions (flowchart, pseudo-code, written description, draft program code)
	1.1.3	understand the purpose of a given algorithm and how an algorithm works
	1.1.4	understand how to determine the correct output of an algorithm for a given set of data
	1.1.5	understand how to identify and correct errors in algorithms
	1.1.6	understand how to code an algorithm in a high-level language
	1.1.7	understand how the choice of algorithm is influenced by the data structures and data values that need to be manipulated
	1.1.8	understand how standard algorithms (bubble sort, merge sort, linear search, binary search) work
	1.1.9	be able to evaluate the fitness for purpose of algorithms in meeting specified requirements efficiently using logical reasoning and test data
1.2 Decomposition and abstraction	1.2.1	be able to analyse a problem, investigate requirements (inputs, outputs, processing, initialisation) and design solutions
	1.2.2	be able to decompose a problem into smaller sub-problems
	1.2.3	understand how abstraction can be used effectively to model aspects of the real world
	1.2.4	be able to program abstractions of real-world examples

Topic 2: Programming

Learning to program is a core component of a computer science course. Students should be competent at designing, reading, writing and debugging programs. They must be able to apply their skills to solve real problems and produce robust programs.

Students should be given repeated opportunities to develop and practise their programming skills.

Students will be expected to use the pseudo-code listed in *Appendix 1* and the flowchart symbols shown in *Appendix 2* and at least one of the programming languages shown in *Appendix 4*.

Subject content	Students should:	
2.1 Develop code	2.1.1	be able to write programs in a high-level programming language
	2.1.2	understand the benefit of producing programs that are easy to read and be able to use techniques (comments, descriptive names (variables, constants, subprograms), indentation) to improve readability and to explain how the code works
	2.1.3	be able to differentiate between types of error in programs (logic, syntax, runtime)
	2.1.4	be able to design and use test plans and test data (normal, boundary, erroneous)
	2.1.5	be able to interpret error messages and identify, locate and fix errors in a program
	2.1.6	be able to determine what value a variable will hold at a given point in a program (trace table)
	2.1.7	be able to determine the strengths and weaknesses of a program and suggest improvements
2.2 Constructs	2.2.1	understand the structural components of a program (variable and type declarations, command sequences, selection, iteration, data structures, subprograms)
	2.2.2	be able to use sequencing, selection and iteration constructs in their programs
2.3 Data types and structures	2.3.1	understand the need for, and understand how to use, data types (integer, real, Boolean, char)
	2.3.2	understand the need for, and understand how to use, data structures (records, one-dimensional arrays, two-dimensional arrays)
	2.3.3	understand the need for, and how to manipulate, strings
	2.3.4	understand the need for, and how to use, variables and constants
	2.3.5	understand the need for, and how to use, global and local variables when implementing subprograms

Subject content	Students should:
2.4 Input/output	2.4.1 understand how to write code that accepts and responds appropriately to user input
	2.4.2 understand the need for, and how to implement, validation
	2.4.3 be able to write code that reads/writes from/to a text file
2.5 Operators	2.5.1 understand the purpose of, and how to use, arithmetic operators (add, subtract, divide, multiply, modulus, integer division)
	2.5.2 understand the purpose of, and how to use, relational operators (equal to, less than, greater than, not equal to, less than or equal to, greater than or equal to)
	2.5.3 understand the purpose of, and how to use, logic operators (AND, OR, NOT)
2.6 Subprograms	2.6.1 understand the benefits of using subprograms and be able to write code that uses user-written and pre-existing (built-in, library) subprograms
	2.6.2 understand the concept of passing data into and out of subprograms (procedures, functions)
	2.6.3 be able to create subprograms that use parameters

Topic 3: Data

Computers are able to store and manipulate large quantities of data. They use binary to represent different types of data.

Students are expected to learn how different types of data are represented in a computer.

Subject content	Students should:
3.1 Binary	3.1.1 understand that computers use binary to represent data (numbers, text, sound, graphics) and program instructions
	3.1.2 understand how computers represent and manipulate numbers (unsigned integers, signed integers (sign and magnitude, two's complement))
	3.1.3 be able to convert between binary and denary whole numbers (0–255)
	3.1.4 understand how to perform binary arithmetic (add, shifts (logical and arithmetic)) and understand the concept of overflow
	3.1.5 understand why hexadecimal notation is used and be able to convert between hexadecimal and binary
3.2 Data representation	3.2.1 understand how computers encode characters using ASCII
	3.2.2 understand how bitmap images are represented in binary (pixels, resolution, colour depth)
	3.2.3 understand how sound, an analogue signal, is represented in binary
	3.2.4 understand the limitations of binary representation of data (sampling frequency, resolution) when constrained by the number of available bits
3.3 Data storage and compression	3.3.1 understand how to convert between the terms 'bit, nibble, byte, kilobyte (KB), megabyte (MB), gigabyte (GB), terabyte (TB)'
	3.3.2 understand the need for data compression and methods of compressing data (lossless, lossy) and that JPEG and MP3 are examples of lossy algorithms
	3.3.3 understand how a lossless, run-length encoding (RLE) algorithm works
	3.3.4 understand that file storage is measured in bytes and be able to calculate file sizes
3.4 Encryption	3.4.1 understand the need for data encryption
	3.4.2 understand how a Caesar cipher algorithm works
3.5 Databases	3.5.1 understand the characteristics of structured and unstructured data
	3.5.2 understand that data can be decomposed, organised and managed in a structured database (tables, records, fields, relationships, keys)

Topic 4: Computers

Students must be familiar with the hardware and software components that make up a computer system and recognise that computers take many forms from embedded microprocessors to distributed clouds.

Subject content	Students should:	
4.1 Machines and computational modelling	4.1.1	understand the input-process-output model
4.2 Hardware	4.2.1	understand the function of the hardware components of a computer system (CPU, main memory, secondary storage, input and output devices) and how they work together
	4.2.2	understand the function of different types of main memory (RAM, ROM, cache)
	4.2.3	understand the concept of a stored program and the role of components of the CPU (control unit (CU), arithmetic/logic unit (ALU), registers, clock, address bus, data bus, control bus) in the fetch-decode-execute cycle (the Von Neumann model)
	4.2.4	understand how data is stored on physical devices (magnetic, optical, solid state)
	4.2.5	understand the concept of storing data in the 'cloud' and other contemporary secondary storage
	4.2.6	understand the need for embedded systems and their functions
4.3 Logic	4.3.1	be able to construct truth tables for a given logic statement (AND, OR, NOT)
	4.3.2	be able to produce logic statements for a given problem
4.4 Software	4.4.1	know what an operating system is and how it manages files, processes, hardware and the user interface
	4.4.2	understand the purpose and functions of utility software (managing, repairing and converting files; compression; defragmentation; backing up; anti-virus, anti-spyware)
	4.4.3	understand how software can be used to simulate and model aspects of the real world
4.5 Programming languages	4.5.1	understand what is meant by high-level and low-level programming languages and understand their suitability for a particular task
	4.5.2	understand what is meant by an assembler, a compiler and an interpreter when translating programming languages and know the advantages and disadvantages of each.

Topic 5: Communication and the internet

Computer networks and the internet are now ubiquitous. Many computer applications in use today would not be possible without networks. Students should understand the key principles behind the organisation and of computer networks. Ideally, they should be able to experiment by setting up a simple network.

Subject content	Students should:	
5.1 Networks	5.1.1	understand why computers are connected in a network
	5.1.2	understand the different types of networks (LAN, WAN) and usage models (client-server, peer-to-peer)
	5.1.3	understand wired and wireless connectivity
	5.1.4	understand that network data speeds are measured in bits per second (Mbps, Gbps)
	5.1.5	understand the role of and need for network protocols (Ethernet, Wi-Fi, TCP/IP, HTTP. HTTPS, FTP, email (POP3, SMTP, IMAP))
	5.1.6	understand that data can be transmitted in packets using layered protocol stacks (TCP/IP)
	5.1.7	understand characteristics of network topologies (bus, ring, star, mesh)
5.2 Network security	5.2.1	understand the importance of network security and be able to use appropriate validation and authentication techniques (access control, physical security and firewalls)
	5.2.2	understand security issues associated with the 'cloud' and other contemporary storage
	5.2.3	understand different forms of cyberattack (based on technical weaknesses and behaviour) including social engineering (phishing, shoulder surfing), unpatched software, USB devices, digital devices and eavesdropping
	5.2.4	understand methods of identifying vulnerabilities including penetration testing, ethical hacking, commercial analysis tools and review of network and user policies
	5.2.5	understand how to protect software systems from cyber attacks, including considerations at the design stage, audit trails, securing operating systems, code reviews to remove code vulnerabilities in programming languages and bad programming practices, modular testing and effective network security provision
5.3 The internet and the world wide web	5.3.1	understand what is meant by the internet and how the internet is structured (IP addressing, routers)
	5.3.2	understand what is meant by the world wide web (WWW) and components of the WWW (web server URLs, ISP, HTTP, HTTPS, HTML)

Topic 6: The bigger picture

Students should be aware of the influence of computing technology and recognise that computing has an impact on nearly every aspect of the world in which they live.

Subject content	Students should:
6.1 Emerging trends, issues and impact	6.1.1 understand the environmental impact of technology (health, energy use, resources) on society
	6.1.2 understand the ethical impact of using technology (privacy, inclusion, professionalism) on society
	6.1.3 understand the legal impact of using technology (intellectual property, patents, licensing, open source and proprietary software, cyber-security) on society