Paper 1: Principles of Computer Science

Overview

In order to become proficient computer scientists, it is essential that students have knowledge and understanding of the field's fundamental principles and concepts.

Content

Topic 1: Computational thinking

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

Students are expected to be familiar with and use the Programming Language Subset (PLS) document provided on the GCSE in Computer Science section of our website.

The flowchart symbols students are expected to be familiar with and use are shown in *Appendix 2.*

Subject content	Students should:	
1.1 Decomposition and abstraction	1.1.1	understand the benefit of using decomposition and abstraction to model aspects of the real world and analyse, understand and solve problems
	1.1.2	understand the benefits of using subprograms
1.2 Algorithms	1.2.1	be able to follow and write algorithms (flowcharts, pseudocode*, program code) that use sequence, selection, repetition (count-controlled, condition-controlled) and iteration (over every item in a data structure), and input, processing and output to solve problems
	1.2.2	understand the need for and be able to follow and write algorithms that use variables and constants and one- and two-dimensional data structures (strings, records, arrays)
	1.2.3	understand the need for and be able to follow and write algorithms that use arithmetic operators (addition, subtraction, division, multiplication, modulus, integer division, exponentiation), relational operators (equal to, less than, greater than, not equal to, less than or equal to, greater than or equal to) and logical operators (AND, OR, NOT)
	1.2.4	be able to determine the correct output of an algorithm for a given set of data and use a trace table to determine what value a variable will hold at a given point in an algorithm
	1.2.5	understand types of errors that can occur in programs (syntax, logic, runtime) and be able to identify and correct logic errors in algorithms
	1.2.6	understand how standard algorithms (bubble sort, merge sort, linear search, binary search) work

Subject content	Students should:
	1.2.7 be able to use logical reasoning and test data to evaluate an algorithm’s fitness for purpose and efficiency (number of compares, number of passes through a loop, use of memory)
1.3 Truth tables	1.3.1 be able to apply logical operators (AND, OR, NOT) in truth tables with up to three inputs to solve problems

*In this specification, the term ‘pseudocode’ is used to denote an informal written description of a program. Pseudocode uses imprecise English language statements and does not require any strict programming syntax.

Topic 2: Data

Computers 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:	
2.1 Binary	2.1.1	understand that computers use binary to represent data (numbers, text, sound, graphics) and program instructions and be able to determine the maximum number of states that can be represented by a binary pattern of a given length
	2.1.2	understand how computers represent and manipulate unsigned integers and two's complement signed integers
	2.1.3	be able to convert between denary and 8-bit binary numbers (0 – 255, -127 – 128)
	2.1.4	be able to add together two positive binary patterns and apply logical and arithmetic binary shifts
	2.1.5	understand the concept of overflow in relation to the number of bits available to store a value
	2.1.6	understand why hexadecimal notation is used and be able to convert between hexadecimal and binary
2.2 Data representation	2.2.1	understand how computers encode characters using 7-bit ASCII
	2.2.2	understand how bitmap images are represented in binary (pixels, resolution, colour depth)
	2.2.3	understand how analogue sound is represented in binary (amplitude, sample rate, bit depth, sample interval)
	2.2.4	understand the limitations of binary representation of data when constrained by the number of available bits
2.3 Data storage and compression	2.3.1	understand that data storage is measured in binary multiples (bit, nibble, byte, kibibyte, mebibyte, gibibyte, tebibyte) and be able to construct expressions to calculate file sizes and data capacity requirements
	2.3.2	understand the need for data compression and methods of compressing data (lossless, lossy)

Topic 3: Computers

Students must be familiar with the hardware and software components that make up a computer system.

Subject content	Students should:	
3.1 Hardware	3.1.1	understand the von Neumann stored program concept and the role of main memory (RAM), CPU (control unit, arithmetic logic unit, registers), clock, address bus, data bus, control bus in the fetch-decode-execute cycle
	3.1.2	understand the role of secondary storage and the ways in which data is stored on devices (magnetic, optical, solid state)

Topic 4: Networks

Most computer applications in use today would not be possible without networks. Students should understand the key principles behind the organisation of computer networks.

Subject content	Students should:
4.1 Networks	4.1.8 understand characteristics of network topologies (bus, star, mesh)

Topic 5: Issues and impact

Students should be aware of the influence of digital technology and recognise some of the issues and the impact on wider society associated with its use.

Subject content	Students should:	
5.2 Ethical and legal	5.2.1	understand ethical and legal issues associated with the collection and use of personal data (privacy, ownership, consent, misuse, data protection)
	5.2.2	understand ethical and legal issues associated with the use of artificial intelligence, machine learning and robotics (accountability, safety, algorithmic bias, legal liability)
	5.2.3	understand methods of intellectual property protection for computer systems and software (copyright, patents, trademarks, licencing)
5.3 Cybersecurity	5.3.1	understand the threat to digital systems posed by malware (viruses, worms, Trojans, ransomware, key loggers) and how hackers exploit technical vulnerabilities (unpatched software, out-of-date anti-malware) and use social engineering to carry out cyberattacks
	5.3.2	understand methods of protecting digital systems and data (anti-malware, encryption, acceptable use policies, backup and recovery procedures)

Paper 2: Application of Computational Thinking

Overview

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 readable, robust programs.

Content

Topic 6: Problem solving with programming

All problems set in the practical programming tasks in the examination can be solved with the functionalities presented in the Programming Language Subset (PLS) document provided on the GCSE in Computer Science section of our website.

Subject content	Students should:	
6.1 Develop code	6.1.1	be able to use decomposition and abstraction to analyse, understand and solve problems
	6.1.2	be able to read, write, analyse and refine programs written in a high-level programming language
	6.1.3	be able to convert algorithms (flowcharts, pseudocode*) into programs
	6.1.4	be able to use techniques (layout, indentation, comments, meaningful identifiers, white space) to make programs easier to read, understand and maintain
	6.1.5	be able to identify, locate and correct program errors (logic, syntax, runtime)
	6.1.6	be able to use logical reasoning and test data to evaluate a program's fitness for purpose and efficiency (number of compares, number of passes through a loop, use of memory)

*In this specification, the term 'pseudocode' is used to denote an informal written description of a program. Pseudocode uses imprecise English language statements and does not require any strict programming syntax.

Subject content	Students should:	
6.2 Constructs	6.2.1	understand the function of and be able to identify the structural components of programs (constants, variables, initialisation and assignment statements, command sequences, selection, repetition, iteration, data structures, subprograms, parameters, input/output)
	6.2.2	be able to write programs that make appropriate use of sequencing, selection, repetition (count-controlled, condition-controlled), iteration (over every item in a data structure) and single entry/exit points from code blocks and subprograms
6.3 Data types and structures	6.3.1	be able to write programs that make appropriate use of primitive data types (integer, real, Boolean, char) and one- and two-dimensional structured data types (string, array, record)
	6.3.2	be able to write programs that make appropriate use of variables and constants
	6.3.3	be able to write programs that manipulate strings (length, position, substrings, case conversion)
6.4 Input/output	6.4.1	be able to write programs that accept and respond appropriately to user input
	6.4.2	be able to write programs that read from and write to comma separated value text files
	6.4.3	understand the need for and be able to write programs that implement validation (length check, presence check, range check, pattern check)
	6.4.4	understand the need for and be able to write programs that implement authentication (ID and password, lookup)
6.5 Operators	6.5.1	be able to write programs that use arithmetic operators (addition, subtraction, division, multiplication, modulus, integer division, exponentiation)
	6.5.2	be able to write programs that use relational operators (equal to, less than, greater than, not equal to, less than or equal to, greater than or equal to)
	6.5.3	be able to write programs that use logical operators (AND, OR, NOT)
6.6 Subprograms	6.6.1	be able to write programs that use pre-existing (built-in, library) and user-devised subprograms (procedures, functions)
	6.6.2	be able to write functions that may or may not take parameters but must return values, and procedures that may or may not take parameters but do not return values
	6.6.3	understand the difference between and be able to write programs that make appropriate use of global and local variables