2 The specification overview

2a. OCR's GCSE (9–1) in Computer Science (J276)

Learners take Components: 01, 02, and 03; or 01, 02 and 04 to be awarded the OCR GCSE (9–1) in Computer Science.

Content Overview	Assessment Overview	
Computer systems • Systems Architecture • Memory • Storage • Wired and wireless networks • Network topologies, protocols and layers • System security • System software • Ethical, legal, cultural and environmental concerns	Computer systems (01) 80 marks 1 hour and 30 minutes Written paper (no calculators allowed)	**40%** of total GCSE
Computational thinking, algorithms and programming • Algorithms * • Programming techniques • Producing robust programs • Computational logic • Translators and facilities of languages • Data representation	Computational thinking, algorithms and programming (02) 80 marks 1 hour and 30 minutes Written paper (no calculators allowed)	**40%** of total GCSE
Programming project ** • Programming techniques • Analysis • Design • Development • Testing and evaluation and conclusions	Programming project (03/04) 40 marks Totalling 20 hours Non-Exam Assessment (NEA)	**20%** of total GCSE

* Algorithm questions are not exclusive to Component 02 and can be assessed in all components.

**Indicates inclusion of synoptic assessment.

Learners who are retaking the qualification may carry forward their result for the non-examined assessment component.

2b. Content of Computer systems (J276/01)

This component will introduce learners to the Central Processing Unit (CPU), computer memory and storage, wired and wireless networks, network topologies, system security and system software. It is expected that learners will become familiar with the impact of Computer Science in a global context through the study of the ethical, legal, cultural and environmental concerns associated with Computer Science. It is expected that learners will draw on this underpinning content when completing the Programming Project component (03 or 04).

1.1 Systems architecture

Learners should have studied the following:

- the purpose of the CPU
- Von Neumann architecture:
  - MAR (Memory Address Register)
  - MDR (Memory Data Register)
  - Program Counter
  - Accumulator
- common CPU components and their function:
  - ALU (Arithmetic Logic Unit)
  - CU (Control Unit)
  - Cache
- the function of the CPU as fetch and execute instructions stored in memory
- how common characteristics of CPUs affect their performance:
  - clock speed
  - cache size
  - number of cores
- embedded systems:
  - purpose of embedded systems
  - examples of embedded sytems.

1.2 Memory

Learners should have studied the following:

- the difference between RAM and ROM
- the purpose of ROM in a computer system
- the purpose of RAM in a computer system
- the need for virtual memory
- flash memory.

1.3 Storage

Learners should have studied the following:

- the need for secondary storage
- data capacity and calculation of data capacity requirements
- common types of storage:
  - optical
  - magnetic
  - solid state
- suitable storage devices and storage media for a given application, and the advantages and disadvantages of these, using characteristics:
  - capacity
  - speed
  - portability
  - durability
  - reliability
  - cost.

1.4 Wired and wireless networks

Learners should have studied the following:

- types of networks:
  - LAN (Local Area Network)
  - WAN (Wide Area Network)
- factors that affect the performance of networks
- the different roles of computers in a client-server and a peer-to-peer network
- the hardware needed to connect stand-alone computers into a Local Area Network:
  - wireless access points
  - routers/switches
  - NIC (Network Interface Controller/Card)
  - transmission media
- the internet as a worldwide collection of computer networks:
  - DNS (Domain Name Server)
  - hosting
  - the cloud
- the concept of virtual networks.

1.5 Network topologies, protocols and layers

Learners should have studied the following:

- star and mesh network topologies
- Wifi:
  - frequency and channels
  - encryption
- ethernet
- the uses of IP addressing, MAC addressing, and protocols including:
  - TCP/IP (Transmission Control Protocol/Internet Protocol)
  - HTTP (Hyper Text Transfer Protocol)
  - HTTPS (Hyper Text Transfer Protocol Secure)
  - FTP (File Transfer Protocol)
  - POP (Post Office Protocol)
  - IMAP (Internet Message Access Protocol)
  - SMTP (Simple Mail Transfer Protocol)
- the concept of layers
- packet switching.

1.6 System security

Learners should have studied the following:

- forms of attack
- threats posed to networks:
  - malware
  - phishing
  - people as the 'weak point' in secure systems (social engineering)
  - brute force attacks
  - denial of service attacks
  - data interception and theft
  - the concept of SQL injection
  - poor network policy
- identifying and preventing vulnerabilities:
  - penetration testing
  - network forensics
  - network policies
  - anti-malware software
  - firewalls
  - user access levels
  - passwords
  - encryption.

1.7 Systems software

Learners should have studied the following:

- the purpose and functionality of systems software
- operating systems:
  - user interface
  - memory management/multitasking
  - peripheral management and drivers
  - user management
  - file management
- utility system software:
  - encryption software
  - defragmentation
  - data compression
  - the role and methods of backup:
    - full
    - incremental.

1.8 Ethical, legal, cultural and environmental concerns

Learners should have studied the following:

- how to investigate and discuss Computer Science technologies while considering:
  - ethical issues
  - legal issues
  - cultural issues
  - environmental issues.
  - privacy issues.
- how key stakeholders are affected by technologies
- environmental impact of Computer Science
- cultural implications of Computer Science
- open source vs proprietary software
- legislation relevant to Computer Science:
  - The Data Protection Act 1998
  - Computer Misuse Act 1990
  - Copyright Designs and Patents Act 1988
  - Creative Commons Licensing
  - Freedom of Information Act 2000.

2c. Content of Computational thinking, algorithms and programming (J276/02)

This component incorporates and builds on the knowledge and understanding gained in Component 01, encouraging learners to apply this knowledge and understanding using computational thinking. Learners will be introduced to algorithms and programming, learning about programming techniques, how to produce robust programs, computational logic, translators and facilities of computing languages and data representation. Learners will become familiar with computing related mathematics.

It is expected that learners will draw on this underpinning content when completing the Programming Project component (03 or 04).

2.1 Algorithms

Learners should have studied the following:

- computational thinking:
  - abstraction
  - decomposition
  - algorithmic thinking
- standard searching algorithms:
  - binary search
  - linear search
- standard sorting algorithms:
  - bubble sort
  - merge sort
  - insertion sort
- how to produce algorithms using:
  - pseudocode
  - using flow diagrams
- interpret, correct or complete algorithms.

2.2 Programming techniques

Learners should have studied the following:

- the use of variables, constants, operators, inputs, outputs and assignments
- the use of the three basic programming constructs used to control the flow of a program:
  - sequence
  - selection
  - iteration (count and condition controlled loops)
- the use of basic string manipulation
- the use of basic file handling operations:
  - open
  - read
  - write
  - close
- the use of records to store data
- the use of SQL to search for data
- the use of arrays (or equivalent) when solving problems, including both one and two dimensional arrays
- how to use sub programs (functions and procedures) to produce structured code
- the use of data types:
  - integer
  - real
  - Boolean
  - character and string
  - casting
- the common arithmetic operators
- the common Boolean operators.

2.3 Producing robust programs

Learners should have studied the following:

- defensive design considerations:
  - input sanitisation/validation
  - planning for contingencies
  - anticipating misuse
  - authentication
- maintainability:
  - comments
  - indentation
- the purpose of testing
- types of testing:
  - iterative
  - final/terminal
- how to identify syntax and logic errors
- selecting and using suitable test data.

2.4 Computational logic

Learners should have studied the following:

- why data is represented in computer systems in binary form
- simple logic diagrams using the operations AND, OR and NOT
- truth tables
- combining Boolean operators using AND, OR and NOT to two levels
- applying logical operators in appropriate truth tables to solve problems
- applying computing-related mathematics:
  - +
  - –
  - /
  - *
  - Exponentiation (^)
  - MOD
  - DIV

2.5 Translators and facilities of languages

Learners should have studied the following:

- characteristics and purpose of different levels of programming language, including low level languages
- the purpose of translators
- the characteristics of an assembler, a compiler and an interpreter
- common tools and facilities available in an integrated development environment (IDE):
  - editors
  - error diagnostics
  - run-time environment
  - translators.

2.6 Data representation

Learners should have studied the following:

Units

- bit, nibble, byte, kilobyte, megabyte, gigabyte, terabyte, petabyte
- how data needs to be converted into a binary format to be processed by a computer.

Numbers

- how to convert positive denary whole numbers (0–255) into 8 bit binary numbers and vice versa
- how to add two 8 bit binary integers and explain overflow errors which may occur
- binary shifts
- how to convert positive denary whole numbers (0–255) into 2 digit hexadecimal numbers and vice versa
- how to convert from binary to hexadecimal equivalents and vice versa
- check digits.

Characters

- the use of binary codes to represent characters
- the term 'character-set'
- the relationship between the number of bits per character in a character set and the number of characters which can be represented (for example ASCII, extended ASCII and Unicode).

Images

- how an image is represented as a series of pixels represented in binary
- metadata included in the file
- the effect of colour depth and resolution on the size of an image file.

Sound

- how sound can be sampled and stored in digital form
- how sampling intervals and other factors affect the size of a sound file and the quality of its playback:
  - sample size
  - bit rate
  - sampling frequency.

Compression

- need for compression
- types of compression:
  - lossy
  - lossless.

2d. Content for the non-exam assessment (NEA) Programming Project (J276/03/04)

OCR will issue three assessment tasks at the start of the terminal academic year of assessment. Only tasks designated for that examination series can be submitted unless carrying forward marks from a previous year. The tasks will provide opportunities for the learners to demonstrate their practical ability in the skills outlined in the specification.

Learners will need to create suitable algorithms which will provide a solution to the problems identified in the task. They will then code their solutions in a suitable programming language. The solutions must be tested at each stage to ensure they solve the stated problem and learners must use a suitable test plan with appropriate test data.

The code must be suitably annotated to describe the process. Test results should be annotated to show how these relate to the code, the test plan and the original problem.

Learners will need to provide an evaluation of their solution based on the test evidence.

Learners should be encouraged to be innovative and creative in how they approach solving the tasks.

Learners are not allowed access to the internet within the non-exam assessment controlled environment, unless the centre is using an online IDE (Integrated Development Environment). In which case, only access to the IDE website is allowed.

All work submitted by a learner must have been done under observation by their teacher and the final report must be only their own work. External sources can be used but must be referenced and no marks can be awarded for materials submitted which are not the learner's own. Common coded solutions identified as being used by learners will not be given credit during moderation.

Group work can be used to deliver the content and skills but any work submitted as non-exam assessment must be the learner's own.

A form (to be confirmed) will be available at www.ocr.org.uk and will be required upon submission to confirm the validity of the learner's work by the learner, the teacher, and a member of the senior leadership team at the centre.

The non-exam assessment should take a total of 20 hours to complete unless there are specific access requirements that should be considered.

The non-exam assessment should be done using a suitable high level language such as:

- Python
- C family of languages (for example C# C++ etc.)
- Java
- JavaScript
- Visual Basic/.Net
- PHP
- Delphi
- SQL
- BASH

Computational thinking is in essence the ability to model problems in a manner that makes them amenable to computational solutions; it is not simply instructions and actions. Computational thinkers are able to see algorithms, processes and data and know how to then implement them in their chosen language.

In Component 03/04 learners must think computationally to solve a task and while doing so create a report detailing the creation of their solution, explaining what they did and why they did it.

The project can be carried out in many ways but is best approached using an iterative process for developing solutions to the task such as below:

- **Success criteria** (what will a successful solution be)
- **Planning and design** (flow charts and pseudocode)
- **Development** (narrative of the process with explanations of code)
- **Testing and remedial actions** (with narrative of changes made)
- **Evaluation** (clearly linked to success criteria).

This process will allow learners to demonstrate the key elements of computational thinking:

- **Thinking abstractly** – removing unnecessary detail
- **Thinking ahead** – identifying preconditions and inputs and outputs
- **Thinking procedurally** – identifying components of problems and solutions
- **Thinking logically** – predicting and analysing problems
- **Thinking concurrently** – spotting and using similarities.

3.1 Programming techniques

Learners should have studied the following:

- how to identify and use variables, operators, inputs, outputs and assignments
- how to understand and use the three basic programming constructs used to control the flow of a program: Sequence; Selection; Iteration
- how to understand and use suitable loops including count and condition controlled loops
- how to use different types of data, including Boolean, string, integer and real, appropriately in solutions to problems
- how to understand and use basic string manipulation
- how to understand and use basic file handling operations:
  - open
  - read
  - write
  - close
- how to define and use arrays (or equivalent) as appropriate when solving problems
- how to understand and use functions/sub programs to create structured code.

3.2 Analysis

Learners should have studied the following:

- how to analyse and identify the requirements for a solution to the problem
- how to set clear objectives that show an awareness of the need for real world utility
- how to use abstraction and decomposition to design the solution to a problem
- how to identify the data requirements for their system
- how to identify test procedures to be used during and after development to check their system against the success criteria
- how to use validation to ensure a robust solution to a problem.

3.3 Design

Learners should have studied the following:

- how to design suitable algorithms to represent the solution to a problem
- how to design suitable input and output formats and navigation methods for their system
- how to identify suitable variables and structures with appropriate validation for their system
- how to use appropriate data types in their system
- how to use functions/sub programmes to produce structured reusable code
- how to select suitable techniques for the development of the solution.

3.4 Development

Learners should have studied the following:

- how to develop a solution to the identified problem using a suitable programming language(s)
- how to demonstrate testing and refinement of the code during development
- how to explain the solution using suitable annotation and evidence of development
- how to use suitable techniques to solve all aspects of the problem
- how to take a systematic approach to problem solving
- how to deploy practical techniques in an efficient and logical manner
- how to show an understanding of the relevant information by presenting evidence of the development of their solutions
- how to show an understanding of the technical terminology/concepts that arise from their investigation through analysis of the data collected
- how to use the terminology/concepts surrounding their topic and contained in the information collected correctly when it comes to producing analysis in the supporting script.

3.5 Testing and evaluation and conclusions

Learners should have studied the following:

- how to produce a full report covering all aspects of the investigation
- how to present the information in a clear form which is understandable by a third party and which is easily navigatable
- how to critically appraise the evidence that they have presented
- how to test their own solution
- how to present their evaluation in a relevant, clear, organised, structured and coherent format
- how to use specialist terms correctly and appropriately
- how to present a conclusion to the report
- how to justify their conclusions based on the evidence provided.