OCR Level 3 Advanced GCE in Computer Science (H446)

“At its heart lies the notion of computational thinking: a mode of thought that goes well beyond software and hardware, and that provides a framework within which to reason about systems and problem.” (Computer Science a Curriculum for Schools)

Computer Science is a practical subject where learners can apply the academic principles learned in the classroom to real world systems.

It is an intensely creative subject that combines invention and excitement, and can look at the natural world through a digital prism. OCR A Level in Computer Science will value computational thinking, helping learners to develop the skills to solve problems, design systems and understand the power and limits of human and machine intelligence. Learners will develop an ability to analyse, critically evaluate and make decisions. The project approach is a vital component of ‘post-school’ life and is of particular relevance to Further Education, Higher Education and the workplace.

The aims of this qualification are to enable learners to develop:

- the capacity for thinking creatively, innovatively, analytically, logically and critically
- the capacity to see relationships between different aspects of computer science
- mathematical skills
- the ability to articulate the individual (moral), social (ethical), legal and cultural opportunities and risks of digital technology.
- an understanding of and ability to apply the fundamental principles and concepts of computer science including; abstraction, decomposition, logic, algorithms and data representation
- the ability to analyse problems in computational terms through practical experience of solving such problems including writing programs to do so

Computer systems -Two and a half hour written exam in Y13

- The characteristics of contemporary processors, input, output and storage devices
- Software and software development Exchanging data
- Data types, data structures and algorithms
- Legal, moral, cultural and ethical issues

Algorithms and programming -Two and a half hour written exam in Y13

- Elements of computational thinking
- Problem solving and programming
- Algorithms

Programming project -Programming languages internally marked in Y13

Python (with a suitable graphical interface), C family of languages (for example C# C+ etc.) Java, Visual Basic, PHP, Delphi

- Analysis of the problem
- Design of the solution
- Developing the solution
- Evaluation

Content of Computer systems (Component 01)

1.1 The characteristics of contemporary processors, input, output and storage devices
Components of a computer and their uses

1.1.1 Structure and function of the processor	a) The Arithmetic and Logic Unit; ALU, Control Unit and Registers (Program Counter; PC, Accumulator; ACC, Memory Address Register; MAR, Memory Data Register; MDR, Current Instruction Register; CIR). Buses: data, address and control: How this relates to assembly language programs. b) The Fetch-Decode-Execute Cycle, including its effect on registers. c) The factors affecting the performance of the CPU: clock speed, number of cores, cache. d) The use of pipelining in a processor to improve efficiency. e) Von Neumann, Harvard and contemporary processor architecture.
1.1.2 Types of processor	a) The differences between, and uses of, CISC and RISC processors. b) GPUs and their uses (including those not related to graphics). c) Multicore and Parallel systems.
1.1.3 Input, output and storage	a) How different input output and storage devices can be applied as a solution of different problems. b) The uses of magnetic, flash and optical storage devices. c) RAM and ROM. d) Virtual storage
1.2.1 Systems Software	a) The need for, function and purpose of operating systems. b) Memory management (paging, segmentation and virtual memory). c) Interrupts, the role of interrupts and Interrupt Service Routines (ISR), role within the fetch decode execute cycle. d) Scheduling: round robin, first come first served, multi-level feedback queues, shortest job first and shortest remaining time. e) Distributed, embedded, multi-tasking, multi-user and real time operating systems. f) BIOS. g) Device drivers. h) Virtual machines, any instance where software is used to take on the function of a machine, including executing intermediate code or running an operating system within another.
1.2.2 Applications Generation	a) The nature of applications, justifying suitable applications for a specific purpose. b) Utilities. c) Open source vs closed source. d) Translators: interpreters, compilers and assemblers. e) Stages of compilation (Lexical analysis, Syntax analysis, Code generation and Optimisation). f) Linkers and loaders and use of libraries.
1.2.3 Software Development	a) Understand the waterfall lifecycle, agile methodologies, extreme programming, the spiral model and rapid application development. b) The relative merits and drawbacks of different methodologies and when they might be used. c) Writing and following algorithms.

1.2.4 Types of Programming Language	a) Need for and characteristics of a variety of programming paradigms. b) Procedural languages. c) Assembly language (including following and writing simple programs with the Little Man Computer instruction set). See appendix 5e. d) Modes of addressing memory (immediate, direct, indirect and indexed). e) Object-oriented languages (See appendix 5e for pseudo-code style) with an understanding of classes, objects, methods, attributes, inheritance, encapsulation and polymorphism.

1.3 Exchanging data -
How data is exchanged between different systems

1.3.1 Compression, Encryption and Hashing	a) Lossy vs Lossless compression. b) Run length encoding and dictionary coding for lossless compression. c) Symmetric and asymmetric encryption. d) Different uses of hashing.
1.3.2 Databases	a) Relational database, flat file, primary key, foreign key, secondary key, entity relationship modeling, normalisation and indexing. See appendix 5g. b) Methods of capturing, selecting, managing and exchanging data. c) Normalisation to 3NF. d) SQL - Interpret and modify. See appendix 5e. e) Referential Integrity. f) Transaction processing, ACID (Atomicity, Consistency, Isolation, Durability), record locking and redundancy.
1.3.3 Networks	a) Characteristics of networks and the importance of protocols and standards. b) The internet structure: The TCP/IP Stack. DNS Protocol layering. LANs and WANs. Packet and circuit switching. c) Network security and threats, use of firewalls, proxies and encryption. d) Network hardware. e) Client-server and peer to peer.
1.3.4 Web Technologies	a) HTML, CSS and JavaScript. See appendix 5e. b) Search engine indexing. c) PageRank algorithm. d) Server and client side processing.

1.4 Data types, data structures and algorithms -
How data is represented and stored within different structures. Different algorithms that can be applied to these structures

1.4.1 Data Types	a) Primitive data types, integer, real/floating point, character, string and Boolean. b) Represent positive integers in binary. c) Use of sign and magnitude and two's complement to represent negative numbers in binary. d) Addition and subtraction of binary integers. e) Represent positive integers in hexadecimal. f) Convert positive integers between Binary Hexadecimal and denary. g) Representation and normalisation of floating point numbers in binary. h) Floating point arithmetic, positive and negative numbers, addition and subtraction. i) Bitwise manipulation and masks: shifts, combining with AND, OR, and XOR. j) How character sets (ASCII and UNICODE) are used to represent text.
1.4.2 Data Structures	a) Arrays (of up to 3 dimensions), records, lists, tuples. b) The following structures to store data: linked-list, graph (directed and undirected), stack, queue, tree, binary search tree, hash table. c) How to create, traverse, add data to and remove data from the data structures mentioned above. *(This can be* ***either*** *using arrays and procedural programming* ***or*** *an object-oriented approach).*
1.4.3 Boolean Algebra	a) Define problems using Boolean logic. See appendix 5e. b) Manipulate Boolean expressions, including the use of Karnaugh maps to simplify Boolean expressions. c) Use the following rules to derive or simplify statements in Boolean algebra: De Morgan's Laws, distribution, association, commutation, double negation. d) Using logic gate diagrams and truth tables. See appendix 5e. e) The logic associated with D type flip flops, half and full adders.

1.5 Legal, moral, cultural and ethical issues
The individual (moral), social (ethical) and cultural opportunities and risks of digital technology. Legislation surrounding the use of computers and ethical issues that can or may in the future arise from the use of computers

1.5.1 Computing related legislation	a) The Data Protection Act 1998. b) The Computer Misuse Act 1990. c) The Copyright Design and Patents Act 1988. d) The Regulation of Investigatory Powers Act 2000.
1.5.2 Ethical, moral and cultural issues	a) The individual (moral), social (ethical) and cultural opportunities and risks of digital technology: b) Computers in the workforce c) Automated decision making d) Artificial intelligence e) Environmental effects f) Censorship and the Internet g) Monitor behavior h) Analyse personal information i) Piracy and offensive communications j) Layout, colour paradigms and character sets.

2c. Content of Algorithms and programming (Component 02)

Elements of computational thinking -
Understand what is meant by computational thinking

2.1.1 Thinking abstractly	a) The nature of abstraction.
	b) The need for abstraction.
	c) The differences between an abstraction and reality.
	d) Devise an abstract model for a variety of situations.
2.1.2 Thinking ahead	a) Identify the inputs and outputs for a given situation.
	b) Determine the preconditions for devising a solution to a problem.
	c) The nature, benefits and drawbacks of caching.
	d) The need for reusable program components.
2.1.3 Thinking procedurally	a) Identify the components of a problem.
	b) Identify the components of a solution to a problem.
	c) Determine the order of the steps needed to solve a problem.
	d) Identify sub-procedures necessary to solve a problem.
2.1.4 Thinking logically	a) Identify the points in a solution where a decision has to be taken.
	b) Determine the logical conditions that affect the outcome of a decision.
	c) Determine how decisions affect flow through a program.
2.1.5 Thinking concurrently	a) Determine the parts of a problem that can be tackled at the same time.
	b) Outline the benefits and trade-offs that might result from concurrent processing in a particular situation.

2.2 Problem solving and programming
How computers can be used to solve problems and programs can be written to solve them

2.2.1 Programming techniques	a) Programming constructs: sequence, iteration, branching.
	b) Recursion, how it can be used and compares to an iterative approach.
	c) Global and local variables.
	d) Modularity, functions and procedures, parameter passing by value and by reference.
	e) Use of an IDE to develop/debug a program.
	f) Use of object oriented techniques
2.2.2 Computational methods	a) Features that make a problem solvable by computational methods.
	b) Problem Recognition.
	c) Problem Decomposition.
	d) Use of divide and conquer.
	e) Use of abstraction.
	f) Learners should apply their knowledge of backtracking, data mining, heuristics, performance modelling, pipelining, and visualisation to solve problems.

Programming project

- Described and justified the features that make the problem solvable by computational methods, explaining why it is amenable to a computational approach.
- Identified suitable stakeholders for the project and described them explaining how they will make use of the proposed solution and why it is appropriate to their needs.
- Researched the problem in depth looking at existing solutions to similar problems, identifying and justifying suitable approaches based on this research.
- Identified the essential features of the proposed computational solution explaining these choices.
- Identified and explained with justification any limitations of the proposed solution.
- Specified and justified the requirements for the solution including (as appropriate) any hardware and software requirements.
- Identified and justified measurable success criteria for assessment

- Provided evidence of each stage of the iterative development process for a coded solution relating this to the breakdown of the problem from the analysis stage and explaining what they did and justifying why.
- Provided evidence of prototype versions of their solution for each stage of the process.
- The solution will be well structured and modular in nature
- Code will be annotated to aid future maintenance of the system
- All variables and structures will be appropriately named.
- There will be evidence of validation for all key elements of the solution
- The development will show review at all key stages in the process

- Provided evidence of testing at each stage of the iterative development process.
- Provided evidence of any failed tests and the remedial actions taken with full justification for any actions taken.

- Provided annotated evidence of post development testing for function and robustness.
- Provided annotated evidence for usability testing.

- Used the test evidence to cross reference with the success criteria to evaluate the solution explain how the evidence shows that the criteria has been fully, partially or not met in each case.
- Provided comments on how any partially or unmet criteria could be addressed in further development.
- Provided evidence of the usability features justifying their success, partial success or failure as effective usability features.
- Provided comments on how any issues with partially or unmet usability features could be addressed in further development.
- Considered maintenance issues and limitations of the solution.
- Described how the program could be developed to deal with limitations of potential improvements / changes.
- There is a well-developed line of reasoning which is clear and logically structured. The information presented is relevant and substantiated.