

A Level Computer Science OCR

Content Overview	Assessment Overview	
<ul style="list-style-type: none"> • 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 • Elements of computational thinking • Problem solving and programming • Algorithms to solve problems and standard algorithms <i>The learner will choose a computing problem to work through according to the guidance in the specification.</i> • Analysis of the problem • Design of the solution • Developing the solution • Evaluation 	<p style="text-align: center;">Computer systems (01)</p> <p style="text-align: center;">140 marks</p> <p style="text-align: center;">2 hours and 30 minutes</p> <p style="text-align: center;">written paper</p> <p style="text-align: center;">(no calculators allowed)</p>	<p>40%</p> <p>of total</p> <p>A level</p>
	<p style="text-align: center;">Algorithms and programming (02*)</p> <p style="text-align: center;">140 marks</p> <p style="text-align: center;">2 hours and 30 minutes</p> <p style="text-align: center;">written paper</p> <p style="text-align: center;">(no calculators allowed)</p>	<p>40%</p> <p>of total</p> <p>A level</p>
	<p style="text-align: center;">Programming project 03* – Repository or 04* – Postal or 80 – Carry forward (2018 onwards)*</p> <p style="text-align: center;">70 marks</p> <p style="text-align: center;">Non-exam assessment</p>	<p>20%</p> <p>of total</p> <p>A level</p>

* Indicates synoptic assessment

This component will introduce learners to the internal workings of the Central Processing Unit (CPU), the exchange of data and will also look at software development, data types and legal and ethical issues. It is expected that learners will draw on this underpinning content when studying computational thinking, developing programming techniques and

devising their own programming approach in the Programming project component (03 or 04).

Learners will be expected to apply the criteria below in different contexts including current and future uses of the technologies.

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	<ul style="list-style-type: none"> (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 effects 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	<ul style="list-style-type: none"> (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	<ul style="list-style-type: none"> (a) How different input, output and storage devices can be applied to the solution of different problems. (b) The uses of magnetic, flash and optical storage devices. (c) RAM and ROM. (d) Virtual storage.

1.2 Software and software development

Types of software and the different methodologies used to develop software

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 5d.
- (d) Modes of addressing memory (immediate, direct, indirect and indexed).
- (e) Object-oriented languages (see appendix 5d for pseudocode 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 modelling, normalisation and indexing. See appendix 5f.
- (b) Methods of capturing, selecting, managing and exchanging data.
- (c) Normalisation to 3NF.
- (d) SQL – Interpret and modify. See appendix 5d.
- (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 5d.
- (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.
*(NB 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 5d.
- (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 5d.
- (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 Moral and ethical Issues

The individual moral, social, ethical and cultural opportunities and risks of digital technology:

- Computers in the workforce.
- Automated decision making.
- Artificial intelligence.
- Environmental effects.
- Censorship and the Internet.
- Monitor behaviour.
- Analyse personal information.
- Piracy and offensive communications.
- Layout, colour paradigms and character sets.

Content of Algorithms and programming (Component 02)

This component will incorporate and build on the knowledge and understanding gained in the Computer systems component (01).

In addition, learners should:

- understand what is meant by computational thinking
- understand the benefits of applying computational thinking to solving a wide variety of problems
- understand the principles of solving problems by computational methods
- be able to use algorithms to describe problems
- be able to analyse a problem by identifying its component parts.

2.1 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
(Learners will benefit from being able to program in a procedure/imperative language and object oriented language.)**

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
 - visualisation to solve problems.

2.3 Algorithms

The use of algorithms to describe problems and standard algorithms

2.3.1 Algorithms

- (a) Analysis and design of algorithms for a given situation.
- (b) The suitability of different algorithms for a given task and data set, in terms of execution time and space.
- (c) Measures and methods to determine the efficiency of different algorithms, Big O notation (constant, linear, polynomial, exponential and logarithmic complexity).
- (d) Comparison of the complexity of algorithms.
- (e) Algorithms for the main data structures, (stacks, queues, trees, linked lists, depth-first (post-order) and breadth-first traversal of trees).
- (f) Standard algorithms (bubble sort, insertion sort, merge sort, quick sort, Dijkstra's shortest path algorithm, A* algorithm, binary search and linear search).

String Handling

To get the length of a string:

```
stringname.length
```

To get a substring:

```
stringname.substring(startingPosition, numberOfCharacters)
```

NB The string will start with the 0th character.

Example

```
someText="Computer Science"
```

```
print(someText.length)
```

```
print(someText.substring(3,3))
```

Will display

```
16
```

```
put
```

Subroutines

```
function triple(number)
```

```
    return number*3
```

```
endfunction
```

Called from main program

```
y=triple(7)
```

```
procedure greeting(name)
```

```
    print("hello"+name)
```

```
endprocedure
```

Called from main program

```
greeting("Hamish")
```

Unless stated values passed to subroutines can be assumed to be passed by value.

If this is relevant to the question byVal and byRef will be used. In the case below x is passed by value and y is passed by reference.

```
procedure foobar(x:byVal, y:byRef)
```

```
    ...
```

```
    ...
```

```
endprocedure
```

Arrays

Arrays will be 0 based and declared with the keyword *array*.

```
array names[5]
names[0]="Ahmad"
names[1]="Ben"
names[2]="Catherine"
names[3]="Dana"
names[4]="Elijah"

print(names[3])
```

Example of 2D array:

```
Array board[8,8]
board[0,0]="rook"
```

Reading to and Writing from Files

To open a file to read from `openRead` is used and `readLine` to return a line of text from the file.

The following program makes `x` the first line of `sample.txt`

```
myFile = openRead("sample.txt")
x = myFile.readLine()
myFile.close()
```

`endOfFile()` is used to determine the end of the file. The following program will print out the contents of `sample.txt`

```
myFile = openRead("sample.txt")
while NOT myFile.endOfFile()
    print(myFile.readLine())
endwhile
myFile.close()
```

To open a file to write to `openWrite` is used and `writeLine` to add a line of text to the file. In the program below `hello world` is made the contents of `sample.txt` (any previous contents are overwritten).

```
myFile = openWrite("sample.txt")
myFile.writeLine("Hello World")
myFile.close()
```

Comments

Comments are denoted by `//`

```
print("Hello World") //This is a comment
```

Object-Oriented

Object oriented code will match the pseudocode listed above with the following extensions:

Methods and Attributes:

Methods and attributes can be assumed to be public unless otherwise stated. Where the access level is relevant to the question it will always be explicit in the code denoted by the keywords.

```
public and private.
```

```
private attempts = 3
```

```
public procedure setAttempts(number)
    attempts=number
endprocedure
```

```
public function getAttempts()
    return attempts
endfunction
```

Methods will always be instance methods, learners aren't expected to be aware of static methods. They will be called using object.method so

```
player.setAttempts(5)
```

```
print(player.getAttempts())
```

Constructors and Inheritance

Inheritance is denoted by the `inherits` keyword, superclass methods will be called with the keyword `super`. i.e. `super.methodName(parameters)` in the case of the constructor this would be `super.new()` Constructors will be procedures with the name `new`.

```
class Pet
    private name
    public procedure new(givenName)
        name=givenName
    endprocedure
endclass
```

```
class Dog inherits Pet
    private breed
    public procedure new(givenName, givenBreed)
        super.new(givenName)
        breed=givenBreed
    endprocedure
endclass
```

Constructors and Inheritance

Constructors will be procedures with the name new.

```
class Pet
  private name
  public procedure new(givenName)
    name=givenName
  endprocedure
endclass
```

Inheritance is denoted by the `inherits` keyword, superclass methods will be called with the keyword `super`. i.e. `super.methodName(parameters)` in the case of the constructor this would be `super.new()`

```
class Dog inherits Pet
  private breed
  public procedure new(givenName, givenBreed)
    super.new(givenName)
    breed=givenBreed
  endprocedure
endclass
```

To create an instance of an object the following format is used

```
objectName = new className(parameters)
```

e.g.

```
myDog = new Dog("Fido", "Scottish Terrier")
```

HTML

Learners are expected to have an awareness of the following tags. Any other tags used will be introduced in the question.

`<html>`

`<link>` to link to a CSS file

`<head>`

`<title>`

`<body>`

`<h1>` `<h2>` `<h3>`

`` including the `src`, `alt`, `height` and `width` attributes.

`<a>` including the `href` attribute.

`<div>`

<form>

<input> where the input is a textbox (i.e. has the attribute type="text" and another attribute name to identify it) or a submit button (i.e. has the attribute type="submit")

<p>

<script>

Any other elements used will be explained in the question.

CSS

Learners are expected to be able to use CSS directly inside elements using the style attribute

```
<h1 style="color:blue;">
```

and external style sheets. In the style sheets they should be able to use CSS to define the styling of elements:

```
h1{  
    color:blue;  
}
```

classes

```
.infoBox{  
    background-color: green;  
}
```

and Identifiers

```
#menu{  
    background-color: #A2441B;  
}
```

They are expected to be familiar with the following properties.

```
background-color  
border-color  
border-style  
border-width  
color with named and hex colours  
font-family  
font-size  
height  
width
```

Any other properties used will be explained in the question.

JavaScript

Learners are expected to be able to follow and write basic JavaScript code. It is hoped they will get practical experience of JavaScript in their study of the course. They will not be expected to commit exact details of syntax to memory. Questions in the exam will not penalise learners for minor inaccuracies in syntax. Learners *will* be expected to be familiar with the JavaScript equivalents of the structures listed in the pseudocode section (with the exception of input and output (see below)). They will not be expected to use JavaScript for Object Oriented programming or file handling. Questions will not be asked in JavaScript where something is passed to a subroutine by value or reference is relevant.

Input

Input will be taken in by reading values from a form. *NB learners will not be expected to memorise the method for doing this as focus will be on what they do with that input once it is received.*

Output

By changing the contents of an HTML element

```
chosenElement = document.getElementById("example");  
chosenElement.innerHTML = "Hello World";
```

By writing directly to the document

```
document.write("Hello World");
```

By using an alert box

```
alert("Hello World");
```

Any other JavaScript used will be explained in the question.

Little Man Computer Instruction Set

In questions mnemonics will always be given according to the left hand column below. Different implementations of LMC have slight variations in mnemonics used and to take this into account the alternative mnemonics in the right hand column will be accepted in learners' answers.

| Mnemonic | Instruction | Alternative mnemonics accepted |
|----------|--------------------|--------------------------------|
| ADD | Add | |
| SUB | Subtract | |
| STA | Store | STO |
| LDA | Load | LOAD |
| BRA | Branch always | BR |
| BRZ | Branch if zero | BZ |
| BRP | Branch if positive | BP |
| INP | Input | IN, INPUT |
| OUT | Output | |
| HLT | End program | COB, END |
| DAT | Data location | |

Structured Query Language (SQL)

Learners will be expected to be familiar with the structures below. Should any other aspects of SQL be used they will be introduced and explained in the question.

SELECT (including nested SELECTs)

FROM

WHERE

LIKE

AND

OR

DELETE

INSERT

DROP

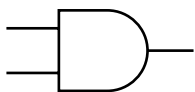
JOIN (Which is equivalent to INNER JOIN, there is no expectation to know about outer, left and right joins)

WILDCARDS (Learners should be familiar in the use of '*' and '%' as a wildcard to facilitate searching and matching where appropriate)

Boolean Algebra

When Boolean algebra is used in questions the notation described below will be used. Other forms of notation exist and below are also a list of accepted notation we will accept from learners.

Conjunction



Notation used:

\wedge e.g. $A \wedge B$

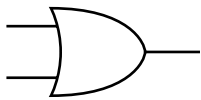
| A | B | $A \wedge B$ |
|---|---|--------------|
| T | T | T |
| T | F | F |
| F | T | F |
| F | F | F |

Alternatives accepted:

AND e.g. A AND B

e.g. A.B

Disjunction



Notation used:

\vee e.g. $A \vee B$

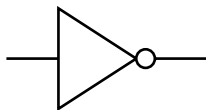
| A | B | $A \vee B$ |
|---|---|------------|
| T | T | T |
| T | F | T |
| F | T | T |
| F | F | F |

Alternatives accepted:

OR e.g. A OR B

+ e.g. A+B

Negation



Notation used:

\neg e.g. $\neg A$

| A | $\neg A$ |
|---|----------|
| T | F |
| F | T |

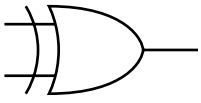
Alternatives Accepted:

bar e.g. \bar{A}

\sim e.g. $\sim A$

NOT e.g. NOT A

Exclusive Disjunction



Notation used:

$\underline{\vee}$ e.g. $A \underline{\vee} B$

| A | B | $A \underline{\vee} B$ |
|---|---|------------------------|
| T | T | F |
| T | F | T |
| F | T | T |
| F | F | F |

Alternatives accepted:

XOR e.g. $A \text{ XOR } B$

\oplus e.g. $A \oplus B$

Equivalence / Iff

Notation used:

\equiv e.g. $(A \wedge B) \equiv \neg(\neg A \vee \neg B)$

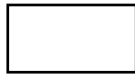
Alternatives accepted:

\leftrightarrow

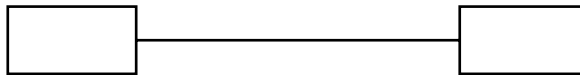
5f. Entity relationship diagrams

The following symbols are used for entities and their relationships.

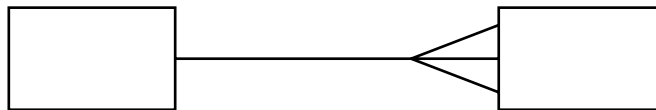
Entity



One-To-One relationship



One-To-Many relationship



Many-To-Many relationship

