AQA Computer Science AS Level

3.4.1 Abstraction and automation

Intermediate Notes

Specification:

3.4.1.1 Problem-solving:

Be able to develop solutions to simple logic problems.

Be able to check solutions to simple logic problems

3.4.1.2 Following and writing algorithms:

Understand the term algorithm.

Be able to express the solution to a simple problem as an algorithm using pseudocode, with the standard constructs:

- sequence
- assignment
- selection
- iteration

Be able to hand-trace algorithms.

Be able to convert an algorithm from pseudocode into high level language program code.

Be able to articulate how a program works, arguing for its correctness and its efficiency using logical reasoning, test data and user feedback.

3.4.1.3 Abstraction:

Be familiar with the concept of abstraction as used in computations and know that:

- representational abstraction is a representation arrived at by removing unnecessary details
- abstraction by generalisation or categorisation is a grouping by common characteristics to arrive at a hierarchical relationship of the 'is a kind of' type

3.4.1.4 Information hiding:

Be familiar with the process of hiding all details of an object that do not contribute to its essential characteristics.

3.4.1.5 Procedural abstraction:

Know that procedural abstraction represents a computational method.

3.4.1.6 Functional abstraction :

Know that for functional abstraction the particular computation method is hidden.

3.4.1.7 Data abstraction:

Know that details of how data are actually represented are hidden, allowing new kinds of data objects to be constructed from previously defined types of data objects.

3.4.1.8 Problem abstraction/reduction:

Know that details are removed until the problem is represented in a way that is possible to solve, because the problem reduces to one that has already been solved.

3.4.1.9 Decomposition:

Know that procedural decomposition means breaking a problem into a number of sub-problems, so that each sub-problem accomplishes an identifiable task, which might itself be further subdivided.

3.4.1.10 Composition:

Know how to build a composition abstraction by combining procedures to form compound procedures.

Know how to build data abstractions by combining data objects to form compound data, for example tree data structure.

3.4.1.11 Automation:

Understand that automation requires putting models (abstraction of real world objects/ phenomena) into action to solve problems. This is achieved by:

- creating algorithms
- implementing the algorithms in program code (instructions)
- implementing the models in data structures
- executing the code

Problem Solving

Problem solving is the process of finding a solution to a difficult or complex issue.

In an exam, you might be given a series of statements from which you have to find the answer to a question.

Example: Given the two statements

George is a student

and

All students like chocolate

which of the following conclusions could be drawn?

George lives in Finland	✘	We can't tell anything about where George lives from the statements, so this conclusion can't be made. This doesn't mean that George doesn't live in Finland, we just don't know for sure.
All chocolate is eaten by students	✘	This could be true, because we're not told that anyone other than students eat chocolate, but we can't say for sure.
George likes chocolate	✔	This must be true. We're told that George is a student and that all students (including George) like chocolate.

Exam questions often contain more than two statements, but the process of forming a reasonable conclusion is the same.

Algorithms

An algorithm is a sequence of steps that can be followed to complete a task. Algorithms always terminate rather than going on forever in a loop.

Algorithms can be written in pseudocode: a way of describing instructions that is independent of any particular programming language.

Assignment in pseudocode

Assignment is the process of giving a value to a variable or constant. In pseudocode, assignment is represented using an arrow pointing towards the variable or constant that is being given a value.

```
counter ← 27
name ← "Sarah"
```

The pseudocode above assigns the value 27 to the variable `counter` and the value Sarah to the variable `name`.

Sequence in pseudocode

Sequence is the name given to instructions that follow on from one another.

```
counter ← 18
counter ← counter + 1
remainingIterations ← 20 - counter
```

In the pseudocode above, the variable `counter` is set to 18 and then incremented by one. Following that, the variable `remainingIterations` is set to twenty minus the value of `counter`. The operations will be executed in the order that they appear.

Selection in pseudocode

Selection is the process of choosing an action to take based on the result of a comparison of values.

```
IF name = "Brian" THEN
    OUTPUT "Hello Brian"
END IF
```

The pseudocode above compares the value of the variable `name` to the value "Brian" and outputs "Hello Brian" depending on the result of the comparison.

In pseudocode, the statements `IF`, `ELSE IF`, `ELSE` and `END IF` can all be used.

Iteration in pseudocode

Iteration is the process of repeating an operation. Iteration structures include FOR and WHILE loops.

```
FOR number ← 6 to 12
     OUTPUT number / 2
END FOR

WHILE number < 18
     Number ← number + (number / 4)
END WHILE
```

The code within an iteration structure is indented, allowing for easy identification of different loops.

Synoptic Link

Indentation is used in programming languages to identify different structures.

Indentation is covered in the notes for **programming** under **fundamentals of programming**.

Abstraction

Abstraction is the name given to the process of omitting unnecessary details from a problem.

When solving a problem, abstraction can be used to simplify the problem which can in turn make finding a solution easier.

There are two distinct forms of abstraction: representational abstraction and abstraction by generalisation / categorisation.

Representational abstraction	Abstraction by generalisation / categorisation
A representation of a problem arrived at by removing unnecessary details from the problem.	A grouping by common characteristics to arrive at a hierarchical relationship of the "is a kind of" type.

The definitions of these two forms of abstraction are often asked for in exams, so it's worth learning them.

Information hiding

Information hiding is defined as the process of hiding all details of an object that do not contribute to its essential characteristics. For example, if you're designing a program that works out how many cars can fit onto a ferry, information about the manufacturer or the colour of a car can be disregarded and just information about the size and weight of cars retained.

Procedural abstraction

Procedural abstraction involves breaking down a complex model into a series of reusable procedures. The actual values used in a computation are abstracted away and a computational method is achieved.

For example: To calculate the area of a rectangle, this procedure could be used:

```
CalcualteArea = width * height
```

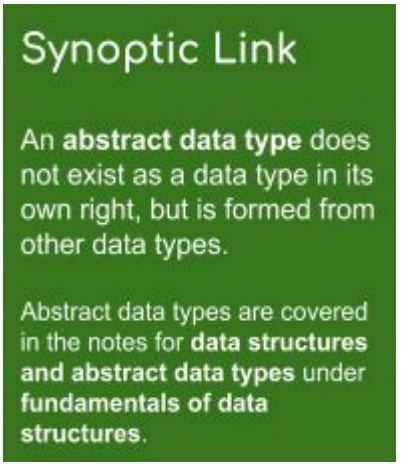
Functional abstraction

Procedural abstraction results in a procedure. Abstracting further disregards the particular method of a procedure and results in just a function.

For example: Abstracting the procedure from the previous example leaves us with a function: `RectangleArea = CalculateArea()`

Data abstraction

In data abstraction, specific details of how data is actually represented are abstracted away, allowing new kinds of data structures to be created from previously defined data structures. Data abstraction forms the basis of abstract data types.



Problem abstraction / reduction

In problem abstraction (which is sometimes called reduction), details are removed from a problem until it is represented in a way that is solvable. This works because a simplified problem is often similar to a problem that has already been solved, meaning that a solution for the problem can be found.

Decomposition

When using decomposition, a problem is divided into a series of smaller problems. These smaller problems can be solved individually until all parts of the original problem have been solved.

Composition

When dealing with a complex problem, composition can be used to combine procedures to form a larger system. Composition is used in abstract data types, where a complex abstract data type is formed for smaller and simpler data types.

Automation

Automation is defined as the process of putting abstractions of real world phenomena into action to solve problems.